Preface

This study was completed as a master thesis at the institute of Informatics and Mathematical Modeling (IMM) - Technical University of Denmark (DTU). The project has been completed in the period between September 1st 2003 and Marts 1st 2004 (and is rated as 35 points). Associate Professor Paul Fischer also IMM, DTU has been supervisor for the project.

The report deals with a general classification problem in the area of bioinformatics, namely secondary structure prediction. However since we do not have a biological background, the problem is treated as a mathematical classification problem.

The report presents the results of the conducted analyses, conclusions and implementations. The report describes the analyses, methods and ideas to a level of details, making it possible to reconstruct the tests and the results presented in this text.

The structure of the report is chronological and may be read from one end to the other. However some of the more theoretical sections may be closely connected to the corresponding result sections.

Several different notations are used in the report. Literature references are given as a number in index parentheses [] in a smaller font-size, like [1]. Footnotes appear in superscript and usually in the end of a sentence, like this[1]. Equations, formulas and other expressions are numbered using a notation like (X.Y which is the Yth expression in section X.

IMM, DTU, Marts the 1st 2004.

Simon Larsen, s973583 *Claus Thomsen, s973607*

_____________________ _____________________

Abstract

This project deals with a specific classification problem in the area of bioinformatics and biology. The problem, typically referred to as secondary structure prediction deals with how the structure of protein sequences may be classified using a number of predefined structure classes.

This project analyses the possible use of Markov models for this classification problem. Markov models are statistical models which may be used to infer the different structure classes for protein sequences based on some training data.

The performance of the developed models are compared to other known models in the area, specifically the GOR models, which are similar to Markov models since they are both statistical models.

The obtained results show that Markov models may be used for secondary structure prediction achieving better performances than just guessing at the most frequent structure class. Starting out with a simple Markov model able to predict around 51% of the structures correctly, the model has been extended and combined with other methods resulting in a prediction accuracy of 57.2% (an increase of around 6%). This resulting model may be characterized as a first generation secondary structure predictor.

Given the time needed several of the weaknesses found in the Markov models may be removed or at least minimized possibly resulting in better performances. The models proposed in this project are not directly usable compared with some of the best predictors current available (having prediction accuracies of around 80%). However there may be room for further development incorporating biological background knowledge into the proposed Markov models.

Resume

Dette projekt omhandler et specifikt klassifikationsproblem indenfor bioinformatik og biologi. Problemet bliver typisk betegnet som forudsigelse af sekundær protein struktur og omhandler hvordan protein sekvenser kan klassificeres ved brug af et bestemt antal struktur klasser.

Projektet analyserer brugen af Markov modeller til at løse dette klassificeringsproblem. Markov modeller er statistiske modeller, som (givet en mængde træningsdata) kan bruges til at udlede de forskellige struktur klasser for proteinsekvenser.

Præcisionen af forudsigelserne for de udviklede modeller er sammenlignet med andre kendte modeller indenfor området. Specifikt benyttes GOR modellerne, som minder noget om Markov modellerne, idet begge er statistiske modeller.

Resultater viser at Markov modeller kan bruges til forudsigelse af sekundær struktur og opnå en højere klassifikationspræcision end ved blot at gætte på den mest hyppige klassifikationstype. Den simple Markov model er grundstenen i projektet og kommer op på en præcision på 51%. Udvides og kombineres denne model med andre metoder opnås en præcision på 57.2% (en stigning på 6% point). Denne resulterende model kan karakteriseres som en første generations model til forudsigelse af sekundær struktur.

Med tiden er det muligt at nogle af de svagheder, der er fundet i Markov modellerne kan fjernes eller minimeres således at præcisionen kan blive endnu højere. Modellerne, der præsenteres i denne rapport kan ikke direkte bruges, sammenlignet med de bedste nuværende modeller på markedet (som har en klassifikationspræcision på omkring 80%). Alligevel er der gode muligheder for yderligere udvikling af modellerne, f.eks. ved at indføre brugen af biologisk baggrundsviden og biologiske metoder i de foreslåede Markov modeller.

Contents

1. Introduction

Classification problems are common in many scientific areas and in many cases these problems are not trivial to solve. In the area of bioinformatics and biology secondary structure prediction is characterized as such a classification problem.

One problem may be characterized as a classification problem, if the solution to the problem involves assigning different classes to the source context. This is the case with secondary structure prediction. Several structure classes are defined which must then be assigned to the (protein) sequence yielding a classified sequence (the procedure involving the assignment of the structure classes are typically referred to as predicting, hence the name).

Secondary structure prediction has been (and still is) an area of great interest and this is due to the fact that the problem has not been completely solved yet. A relatively large number of methods have been developed for the single purpose of determining the secondary structure (typically) of protein sequences.

The information in knowing the three-dimensional structure of proteins is of great value in determining the functions of unknown proteins. However, knowing the secondary structure of a given protein does not automatically supply the three-dimensional structure of the protein, since secondary structure is only two-dimensional (this is addressed in more details in the following section) [16].

Therefore, in the long run, knowledge of the secondary structure is not sufficient to tell how a particular protein works and interacts with other biological entities nor is it the final goal in secondary structure prediction [15]. Typically secondary structure prediction is used as a mean of predicting tertiary structure. The tertiary structure deals with another (higher) level of information, a more detailed level, where folds and bonds define the level of details. This form of structure gives even more information about the functions of proteins and it is therefore of great value for scientists in the area.

Several methods and technologies have been applied in the field of secondary structure prediction. These methods are usually used for classifying a particular residue (one letter in the protein sequence) to a particular structure class or group (referred to as classification type or group). Neural networks, information theory, Bayesian statistics and (hidden) Markov models are some of the technologies which have been used in an effort to create an accurate secondary structure classifier. In other words: a predictor able to predict the secondary structure with such precision, that it is in fact usable.

In this particular project Markov models are used as a technology to implement a classifier able to predict secondary structure of proteins. The Markov models used are mostly based on simple statistics. We start out with a very simple Markov model which is the basis for all further extensions. The analysis and development of the models are conducted as an iterative process, where one scheme is analyzed, tested, discussed and improved if possible. One model may then be the basis for another type of model and so on.

As a method of comparing our implemented classifiers with other known predictors, we analyze and implement the GOR model in different versions [1, 2, 3]. The GOR models are similar to the Markov models used in that both models are based (almost) only on relatively simple statistics.

1.1. Purpose of the Project

The purpose of the project is to analyze and implement some of the various GOR models and using these models for comparison with our own Markov models developed. The Markov models will be based on a simple model (that will be described in details in a later section) and from this model all extensions will be developed. It is interesting to see how the Markov models will match the prediction accuracies of the GOR model and of the other methods available. The Markov models will be analyzed, implemented and tested, and based on these tests we may be able to conclude whether or not it is a good idea to use Markov models for this kind of classification problem.

The best predictors currently available have a prediction accuracy of around 75-80%. It is not our intention to break this limit, but merely to analyze the usability and possibility of using Markov models for secondary structure prediction.

Typically the current predictors having high accuracies have been developed over several years (possibly several decades) and have been under constant improvement during that period. This is the case for the GOR model and also for some of the neural network models [2, 3, 16]. Many of the known predictors are based on a statistical foundation and on top of that biological knowledge may be incorporated in several steps [3, 8]. The Markov models proposed in this text are basically used and interpreted as a mathematical (statistical) model. No background knowledge has been applied and the improvements of the models are based on observations and experience from testing our earlier models.

Since we do not have a biological background, we have treated the problem as an ordinary classification problem using Markov models as a mathematical and statistical technique. For the same reason, the models proposed in this text may therefore be subject to further investigation and possibly improvement (for instance by incorporating biological background knowledge).

2. Secondary Structure Prediction

2.1. Introduction

The subject of predicting the physical structure of proteins has been researched for more than three decades [16]. The reason that so much effort has been made to create a good classifier able to predict the structure with high accuracy, is that the knowledge gained is highly valuable.

Proteins are the fundamental molecules of all organisms and the three-dimensional structure provides great functional information of the proteins. To be able to determine the structure having only the sequence of amino acids (which define the protein) would supply great information on the relationship between the structure and the functions of proteins.

Today the protein structure may be determined 100% accurately using X-ray crystallographic or NMR (Nuclear Magnetic Resonance) techniques, but this is both very time consuming and expensive [16]. At the same time more proteins are determined (or extracted) in large projects like the Humane Genome Project [17] and the need for a good structure predictor is growing even larger.

2.2. Protein Structures

There are several different levels of protein structure, which are referred to as primary, secondary, tertiary and quaternary structure. The different levels of structure will be described here without going into the very details of the structures.

The primary structure of a protein is the sequence of amino acids in that particular order. Amino acids (also referred to as residues) are the building blocks of proteins. There are 20 different amino acids. The amino acids are again defined by the nucleotide sequence (DNA) [16, 18].

The secondary structure of a protein is defined by classes of repeated patterns. The folding of the protein backbone determines the current class. The two most common patterns are the alpha helix and the beta sheet. The alpha helix coils around the imaginary helix axis (clockwise) [18]. This is shown in Figure 1.

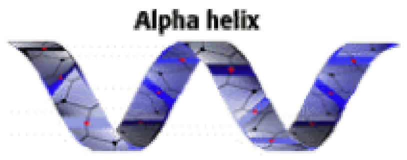


Figure 1: A simplified alpha helix.

Other kinds of helical structures are possible (π-helix and 3-helix) [7], although the alpha helix is the most common helical structure.

The beta sheet is the other most common secondary structure and is known for the backbone to be almost fully extended. A simplified beta sheet is depicted in Figure 2.

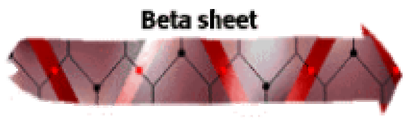


Figure 2: A simplified beta sheet.

Combining several beta sheets is typically referred to as a beta ladder or beta strand.

Usually in secondary structure prediction [2, 3, 8, 11] the possible structures are grouped into 3 classification groups (structure classes), thereby simplifying the problem, which may be referred to as **h**elices (H), (beta) sh**e**ets (E) and **c**oils (C), where the coil group is all the structures not contained in the first two groups. In some studies the beta sheet group may be referred to as (beta) strand [3] and the coil group may be referred to as loop [16].

Figure 3 depicts the secondary (and tertiary) structure of a protein, showing the three different structure classes as described above.

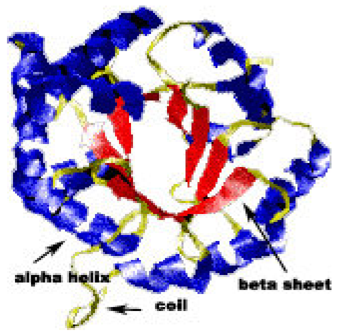


Figure 3: Secondary structure elements shown in a protein.

Typically proteins consists of about 32% alpha helices, 21% beta sheets and 47% coils (or non-regular structure) [16]. Proteins having similar primary structure (similar amino acids sequences) or similar secondary structure are called homologous proteins and are not suited as a basis for secondary structure prediction, since the resulting predictor may be biased towards the proteins. Homologous proteins are often evolved from a common ancestor. Non-homologous sequences (proteins), on the other hand, are useful for secondary structure prediction, since they (ideally) simulate any existing protein (given that the amount of data, i.e. the number of protein sequences is large enough).

The tertiary structure is defined as the overall folding of the protein in three dimensions, showing how the secondary structure parts is connected and how the protein in general is shaped [18].

The quaternary structure describes the overall form of the subunits in the protein (many proteins are formed from more than just one sequence) [18].

This project focuses on the secondary structure prediction as a classification problem and the other structure levels will not be addressed any further.

2.3. Programs and Methods for Secondary Structure Prediction

Many different classifiers (also referred to as predictors) have been developed over several decades [16]. Some of these predictors have been improved and extended over the years while others are build on more resent discoveries in the field. This section will briefly mention some of the known predictors and how they perform[1].

Many of the models developed have been verified by The Protein Structure Prediction Center at Lawrence Livermore National Laboratory, California, USA. This institute has from time to time organized experiments on the Critical Assessment of Techniques for Protein Structure Prediction (CASP) [19]. Many authors like to use CASP as a way to verify their predictor and the claimed prediction accuracy.

[1] The performance of the predictors is usually given as a prediction accuracy which will be defined more precisely in a later section. For now it suffices to say that the prediction accuracy is the main measure for prediction schemes and the value is ranging in the interval from 0 to 100%. It is usually defined as the percentage of correctly predicted residues.

Over the years of trying to improve methods for secondary structure prediction, the data availability has grown larger. More and more proteins are classified with secondary structure classes, providing more training data for the prediction methods. In some cases the predictors obtained higher prediction accuracies when using larger training databases [2, 3].

It is common to see people that have been working in the field for several years, publish assembled protein databases containing a large number of protein sequences and their corresponding structure classes (the correct classifications) [12]. This makes it much easier for new studies to compare their results obtained directly with other methods using the same database.

2.3.1. The GOR Model

This model was first developed in the 60's and has been more or less under constant development since then. The first four iterations of the GOR model are based only on statistics. GOR I used single residue statistics while the later models incorporate pair statistics (GOR III and GOR IV) [2]. However in GOR IV the training data were inspected and corrected for faults. Also some post-processing of the sequences was added to the basic statistical method. In the newest version of the GOR model (GOR V) triplet statistics have been incorporated. Also multiple sequence alignments are utilized in GOR V [3].

The first GOR model (GOR I) claimed to predict around 55% of the residues correctly (predicting one structure class for each residue in the sequence), while GOR V claims to have a prediction accuracy of around 75% [2, 3, 16].

The GOR models will be presented and analyzed in more details in the section 4.The GOR Classifier. Also several of the known GOR models have been implemented, tested and compared with the Markov models developed (and implemented) in this project.

2.3.2. The Chou-Fasman Algorithm

This algorithm was also one of the earlier attempts to implement a good secondary structure classifier for proteins. The method is based on statistics of the classification groups to be predicted (helices, sheets and coils) and were predicting around 55-60% [16].

2.3.3. Neural Networks Using Backpropagation

Qian and Sejnowski have used a classic neural network solution using the backpropagation algorithm to create a classifier having a prediction accuracy of 64.3%. The construction was based on a combination of two neural networks having three layers [16].

2.3.4. The DSC Algorithm

This algorithm uses linear statistics identifying several key elements of the sequences, such as sequence edge effects, residue ratios, secondary structure feedback effects and so on. The algorithm predicted around 70% of the residues correctly [16].

2.3.5. The PREDATOR Algorithm

This algorithm is based on local pair-wise alignment implemented on the sequence to be predicted. It is claimed to predict at 75% [16].

2.3.6. The PHD Program

This method incorporates several cascading neural networks. Each network has a specific function, the first is to predict the secondary structure and the next networks are used to post-process the predictions, making corrections and so on.
The claimed prediction accuracy is 72.2% although CASP2 [16, 19] gets 71.6%.

2.3.7. Neural Networks and Multiple Alignments

Another neural network model has been proposed by Petersen et al. [8]. The model is again based on several combined networks and using information from homologous sequences: multiple alignments. The method claims to have a prediction accuracy of around 75-80% but evaluation by CASP4 is awaited.

2.3.8. Current State of the Art Predictors

In general many of the newest methods are based on statistics or neural networks and multiple sequence alignments. It seems that the performance of many of the previously developed predictors is increased by incorporating some scheme for handling multiple sequence alignments. Multiple sequence alignments are homologous sequences generated from the sequence to be predicted using existing programs. Several programs are developed for this which may be used directly [12, 20]. This makes it a lot easier to incorporate multiple sequence alignments in an existing prediction scheme.

In general the prediction accuracies reported in different papers vary somewhat, making it difficult to get a clear picture of the best current predictors. However it is clear that the performance almost reaches 80%, but is documented around 75%. Prediction accuracies at this level is (amongst others) reported by the latest GOR model, GOR V [3] and the neural network method proposed by Petersen et. al. [8].

Some of the earlier predictors presented (i.e. GOR I) predicted four different structure classes [1, 2] but for many years it have been very common to predict only three different structure classes, namely the three classes mentioned in the previous section (H, E and C) [2, 3, 15, 16]. This fact may have some influence on the performances reported for some of the older methods, since four different structure types are in general more difficult to predict than three.

3. Using Markov Models for Classification

3.1. Introduction

This section and the following subsections introduce the Markov model as a relatively simple statistical model, which may be used for classification problems such as secondary structure prediction.

The section will first present several standard terms which will be used throughout the remainder of the report. Then Markov models in general and how these models may be used for this classification problem will be presented.

Later the basic Markov model used in the project will be presented and all developed extensions to (or combinations of) this model will be discussed. The extended Markov models are all derived from the simple initial model (referred to as the basic Markov model).

The result section later in the report (see the section 7.Tests and Results) addresses the performance of the different models presented.

3.2. General Terms

The following section presents several general terms and expressions. The terms will be used throughout the rest of the report.

3.2.1. Residue

A residue is the one character representation of an amino acid. Amino acids are the building blocks of proteins. There are 20 different amino acids. Several residues (amino acids) in a particular order define a (protein) sequence.

3.2.2. Alphabet

The Markov models are defined over an alphabet of all possible residues. The alphabet is represented using the symbol Σ. In this case there are 20 different residues (each corresponding to one amino acid):

$$\Sigma = \{A, C, D, E, F, G, H, I, K, L, M, N, P, Q, R, S, T, V, W, Y\} \quad \textbf{(3.1}$$

However in the protein databases available, the alphabet may be extended with a few more residues, which typically are used as wildcards. Wildcards are used in places where several of the 20 residues above may occur (see the section 6.Test and Training Data for more information on this).

The length of the alphabet is referred to as $|\Sigma|$.

3.2.3. Sequence

A sequence, *S*, is an ordered set of residues in the alphabet:

$$S = \{\bar{x} \mid \bar{x} \in \Sigma^*\} \quad \textbf{(3.2}$$

If a sequence, *S,* contains *n* residues it is said to be of length *n* and is written:

$$S = x_1 x_2 x_3 ... x_n, \qquad \text{where } x_1,..,x_n \in \Sigma$$ (3.3

3.2.4. Classification Alphabet and Group

A new alphabet is introduced. This alphabet is called the classification alphabet. In this case it contains the one character representations for each of the structure classes we want to predict:

$$\Sigma_{classification} = \{H, E, C\}$$ (3.4

The elements in the classification alphabet are referred to as classification groups or just groups. The group notation is used since each structure class (H for helix, E for beta sheet/strand and C for coil) actually consists of several possible structures.[2]

3.2.5. Classification Sequence

Every sequence of residues, *S*, of length *n* has an associated classification sequence, $S_{classification}$ of length *n*:

$$S_{classification} = \{\bar{y} \mid \bar{y} \in \Sigma^*_{classification}\}$$ (3.5

If a classification sequence, $S_{classification}$, contains *n* letters from the classification alphabet it is said to be of length *n* and is written:

$$S_{classification} = y_1 y_2 y_3 ... y_n, \qquad \text{where } y_1,..,y_n \in \Sigma_{classification}$$ (3.6

The term classification sequence may be referred to as just the classification for some sequence *S*.

3.2.6. Prediction

Given a sequence *S* of residues of length *n*:

$$S = x_1 x_2 x_3 ... x_n, \qquad \text{where } x_1,..,x_n \in \Sigma$$ (3.7

and its associated classification sequence, $S_{classification}$:

$$S_{classification} = y_1 y_2 y_3 ... y_n, \qquad \text{where } y_1,..,y_n \in \Sigma_{classification}$$ (3.8

If the sequence *S* is taken from one of the databases of pre-classified proteins together with the associated classification sequence, $S_{classification}$, then the classification sequence is said to be the *correct classification sequence* for *S* (assuming that the classifications given in the published database does not contain any errors). The residue x_i is said to be correctly classified as group y_i, where $i \in \{1,2,3,...,n\}$. The goal is to be able to predict the correct classification sequence using Markov models.

[2] As mentioned before there are several different helix structures which are grouped in the helix classification group denoted H in the classification alphabet. The same applies for the other groups where the coil group, C, contains all the non-regular structures not in the H or E group.

Given that the Markov models have generated another classification sequence *S_{prediction}*:

$$S_{prediction} = z_1 z_2 z_3 ... z_n, \qquad \text{where } z_1,.., z_n \in \Sigma_{classification} \tag{3.9}$$

then the classification sequence $S_{prediction}$ represents the result obtained using the Markov models to predict the groups for the residues in *S*. The residue x_i is said to be predicted as z_i by the Markov models where $i \in \{1,2,3,..., n\}$.

It is important to separate the two different types of classification sequences. There is a *correct classification* sequence given by the database of pre-classified proteins, and there is a *predicted classification* sequence generated by the Markov models (this will be addressed further later on).

3.2.7. Subsequence

The term subsequence is (in this context) a short description for a maximum uniform subsequence, given by a part of a sequence where all residues are correctly classified as the same group. Given a sequence *S* of length *n*:

$$S = x_1 x_2 x_3 ... x_n, \qquad \text{where } x_1,.., x_n \in \Sigma \tag{3.10}$$

and its associated correct classification sequence, $S_{classification}$, (also of length *n*):

$$S_{classification} = y_1 y_2 y_3 ... y_n, \qquad \text{where } y_1,.., y_n \in \Sigma_{classification} \tag{3.11}$$

A subsequence $S_{subsequence}$ is now defined as:

$$\begin{aligned} & S_{subsequence} = x_i ... x_j, \qquad \text{where } x_i,..., x_j \in \Sigma \\ & \text{and} \\ & \qquad i \geq 1 \\ & \qquad j \geq i \\ & \qquad j \leq n \\ & \qquad \text{if } i > 1 \text{ then } y_{i-1} \neq y_i \\ & \qquad \text{if } j < n \text{ then } y_{j+1} \neq y_j \\ & \qquad y_i = y_{i+1} = ... = y_j \end{aligned} \tag{3.12}$$

In this way a sequence of residues can be divided into several subsequences. Each subsequence also has an associated classification subsequence. The classification subsequence is defined from the correct classification sequence using indices *i* and *j* (both y_i and y_j are included in the classification subsequence).

The definition of the subsequence above forces the classification subsequence to contain only one type of groups. This is the reason that the subsequence of residues is said to belong to a group: all residues in the subsequence are correctly classified as the same group in the classification alphabet (H, E or C).

3.3. Markov Models for Secondary Structure Prediction

A Markov model is a simple statistical model which may be used for classification problems like secondary structure prediction.

The goal in secondary structure prediction can be described as:

- Having a non-classified protein sequence, *S*, we want to assign a group for each residue in *S*, resulting in a predicted classification sequence, $S_{classification}$, associated to *S*.

Furthermore it is wanted that the predicted classification sequence obtained, $S_{classification}$, should be as identical as possible to the correct classification sequence.

Figure 4 shows the first 60 residues of the protein 1add-1-AS.all from the DSSP database [12] and the associated correct classification sequence (for more information on the DSSP database see the section 6.Test and Training Data).

Sequence	`TPAFNKPKVELHVHLDGAIKPETILYFGKKRGIALPADTVEELRNIIGMDKPLSLPGFLA`
Classification	`CCCCCCCEEEEEEEHHHCCCHHHHHHHHHHHCCCCCCCCHHHHHHHHCCCCCCCHHHHCC`

Figure 4: An example of a sequence of residues and the corresponding correct classification sequence.

The goal is to be able to predict the correct classification sequence shown in color codes in Figure 4. The classification sequence consists only of the three types of groups to be predicted (H, E and C) and the figure shows how these groups are associated to each residue.

To be able to predict the classifications above, three Markov models may be used. Each model represents one group, i.e. one model for the helical structures (denoted H), one for the beta sheets (denoted E) and one for the non-regular structures, coils (denoted C).

Each model uses a part of the protein sequence (also referred to as the window residues) as input and produces one output (for each residue), which is the probability that the residue in question may be classified as the group represented by the current model. The result using the three models is three outputs (one for each model) which may be compared. The model having the largest value is the winner and the group can be predicted (the group that the winning model represents is assigned as the classification for the current residue). This procedure is then repeated for every residue in the sequence until every residue has a group assigned.

The three Markov models used for this prediction are trained prior the prediction process using a large database of correctly classified proteins (protein sequences having their associated correct classification sequences given).

The above outline of the prediction process using the Markov models will be described more formally in the following subsections.

3.3.1. Definition of the Markov Models

The Markov models used in this project are defined by a number of parameters, which are estimated based on the training database of classified protein sequences. The models calculate an output using a part of the input sequence (the window of residues) and the estimated parameters.

It is the idea that Markov models may be able to infer the two-dimensional secondary structure, using only one-dimensional sequences as input.

A Markov model, M_X, representing the group X (X $\in \Sigma_{classification}$), is described by $s^2 + s$ parameters: $p(a)$ and $p(a|b)$, where $a, b \in \Sigma$, $s = |\Sigma|$. [3]

Using a number of training sequences and their associated correct classification sequences it is possible to estimate these parameters. This is done in the following way:

1. A large database of sequences and their correct classification sequences is loaded into a suitable data structure.
2. All subsequences are extracted from every loaded protein sequence and stored in three collections, one for each group (H, E and C). Every sequence in the training database is processed once. The result of the procedure is three collections of subsequences, one collection for each classification group.
3. One Markov model is trained for each collection of subsequences. This means that in total three Markov models are obtained, M_H, M_E, M_C. The following method is used to estimate $p(a)$ and $p(a|b)$ in a single Markov model using one of the collections of subsequences:
    a. Loop through all subsequences for the current collection.
    b. Estimate $p(a)$ by counting how often a subsequence starts with an *a*, and dividing this number with the total number of subsequences in the collection. The result is $p(a)$.
    c. Estimate $p(a|b)$ by counting how often an *a* is found immediately after a *b* in all subsequences in the collection and dividing this number with the number of times *a* occur after the first position in all subsequences in the collection. The result is $p(a|b)$.

The formulas for estimating $p(a)$ and $p(a|b)$ may be written more formally as in the following.

$$\begin{array}{ll} \text{Given that:} & \\ a,b & \text{are residues in the alphabet}, \Sigma \\ y_1^j & \text{is the first residue in subsequence } j \\ k & \text{is the total number of subsequences in the collection} \\ i & \text{is the position in the current subsequence}, i > 0 \end{array}$$

(3.13

$$p(a) := \frac{\left|\{j \mid y_1^j = a\}\right|}{k}$$

$$p(a \mid b) := \frac{\left|\{(i,j) \mid i > 1 \wedge y_i^j = a \wedge y_{i-1}^j = b\}\right|}{\left|\{(i,j) \mid i > 1 \wedge y_i^j = a\}\right|}$$

As a consequence of the above the following is always true for $p(a)$ and $p(a|b)$:

$$\forall a \in \Sigma : p(a) \in [0;1] \wedge \sum_{a \in \Sigma} p(a) = 1$$

(3.14

$$\forall (a,b) \in \Sigma^2 : p(a \mid b) \in [0;1] \wedge \forall a \in \Sigma : \sum_{b \in \Sigma} p(a \mid b) = 1$$

[3] The parameters of the Markov model are a consequence of the Markov assumption, which is explained in more details in Appendix A – The Markov Assumption.

All modifications of $p(a)$ and $p(a|b)$ should respect these equations. If a certain residue does not exist in the training data the alphabet is reduced (leaving that residue out) in order to respect the above formulas.

The above procedure may be illustrated with a set of examples. Assume that our entire training database consists of only one sequence consisting of the 60 residues depicted in Figure 4. Then the three collections of subsequences may be extracted as shown in Figure 5.

Sequence	TPAFNKPKVELHVHLDGAIKPETILYFGKKRGIALPADTVEELRNIIGMDKPLSLPGFLA
Classification	CCCCCCCEEEEEEEHHHCCCHHHHHHHHHHHCCCCCCCCHHHHHHHHCCCCCCCHHHHCC

Groups	Helices, H	Beta Sheets, E	Coils, C
Subsequences	LDG PETILYFGKKR VEELRNII LPGF	KVELHVH	TPAFNKP AIK GIALPADT GMDKPLS LA
Total number of subsequences in each group	4	1	5

Figure 5: Extraction of subsequences for the 60 residue sequence shown with its associated correct classification sequence.

Now the parameters $p(a)$ and $p(a|b)$ may be estimated for each of the three Markov models representing one group each. This is done following the above instructions. Starting with the Markov model, M_H, the $p(a)$ parameter is estimated by counting how many times each residue in the alphabet $\sum$ begins the subsequence (and dividing this number with the total number subsequences for group H):

$$
\begin{aligned}
& p(A) := 0 \\
& p(B) := 0 \\
& . \\
& p(L) := 2/4 \\
& . \\
& p(P) := 1/4 \\
& . \\
& p(V) := 1/4 \\
& p(W) := 0 \\
& p(Y) := 0
\end{aligned}
\qquad (3.15
$$

$$\sum_{a \in \Sigma} p(a) = 2/4 + 1/4 + 1/4 = 1$$

As for the $p(a|b)$ parameters the number of occurrences of b following a are counted and divided by the number of times a occurs after the position in the current subsequence.

$$
\begin{aligned}
&p(A \mid A) := 0\\
&p(A \mid B) := 0\\
&p(A \mid C) := 0\\
&.\\
&p(D \mid L) := 1/1
\end{aligned}
\qquad (3.16
$$

This continues until all parameters all estimated.

The same procedure is repeated for the Markov models M_E and M_C using the collection of subsequences associated with the current group. This procedure is referred to as training the Markov models, since the parameters are estimated based on the training data (which is a sample for simulating all possible protein sequences).

The estimated parameters just described are used to calculate the output of the models from some input. This output and input for the Markov models will be described in the following sections.

3.3.2. Output of the Markov model – the Score Function

The goal of the Markov models is to predict a classification sequence that is as identical as possible to the correct classification sequence. Predicting the classification group for each residue in the sequence is possible after completing the training of the three models.

The training data contains both the protein sequences and the correct classifications which makes it possible to check how well the models classify a sequence by comparing the resulting classification sequence with the correct one. This will be addressed further in the following sections.

As mentioned earlier three Markov models are trained (by estimating their parameters), one model for each group in the classification alphabet (M_H, M_E, M_C). Each of these models will be used when a residue is classified (the classification group is predicted).

The output of the three different Markov models determines which group is the most likely classification for that residue. The output of the models may be referred to as the score function or just the score value.

Given the (unclassified) sequence $S = x_1x_2x_3...x_n$, where $x_1,...,x_n \in \Sigma$ and the index i of the residue we want to classify in S, we define the window size as the number of residues which is taken into account, when trying to find the classification for the residue x_i. Having a window size of l, the residues x_i to x_{i+l-1} are used as input for the Markov models trying to classify the residue x_i.

Now given the window size l and the parameters defining the model $p(a)$ and $p(a|b)$ the score function, sc_X for the Markov model predicting group X (where $X \in \Sigma_{classification}$) is then defined as:

$$
\begin{aligned}
l = 1: &\quad sc_X(S,i,l) = p(x_i)\\
l > 1: &\quad sc_X(S,i,l) = p(x_i)p(x_{i+1} \mid x_i)p(x_{i+2} \mid x_{i+1})...p(x_{i+l-1} \mid x_{i+l-2})
\end{aligned}
\qquad (3.17
$$

The window size is actually the number of terms in the score function. This also means that l is the number of residues that affects the final probability and therefore also affects the final predicted classification for the residue x_i. The residues that are used in the score function represent a window in the sequence (having length l) starting at position i and ending at position $i+l-1$.

All individual terms in the score function have a value between 0.0 and 1.0. This means that the window size affects the size of the score value returned by the score function. A larger window size yields a smaller score value. Comparing score values based on different window sizes is therefore unfair (normalization methods compensating for this are presented later).

It is important to notice that to predict the classification, z_i, for the residue x_i, the residues x_i to x_{i+l-1} (all residues in the window) are used to calculate one score value (for each Markov model, M_H, M_E and M_C). The way to decide which group is assigned as the classification group for the residue x_i is described in the next subsection.

3.3.3. Using the Markov Models as a Classifier

The general Markov model has been defined by the estimated parameters and the model may be evaluated given a part of a sequence as input. For utilizing the models for this type of classification (determining a group for each residue in the test sequence) we start out with a simple procedure.

For predicting the classification for the residue at position i in the sequence S, each of the three Markov models, M_H, M_E and M_C are used. The three score functions represent the three Markov models. This also means that the three score functions represent one group each in the classification alphabet. The score functions sc_H, sc_E and sc_C use the parameters from the corresponding Markov models M_H, M_E and M_C (every model uses the same window size l). The three score values are calculated and compared for each position i = 1, 2, ..., $n-l+1$ in the sequence S (where n is the length of S). At every position in the sequence the Markov model with the highest score value is chosen as the most probable classification group for the current residue.

After the residue has been classified the window is shifted, so that $i := i+1$. The window size l and the sequence S is the same. The classification procedure is repeated until the end of the window has reached the end of the sequence. At this point the Markov models have classified the sequence for the positions 1 to $n-l+1$.

However there are some problems with this method. The last $l-1$ residues can not be classified since the window would exceed the end of the sequence, S. Those residues are ignored and not taken into account when the prediction accuracy is calculated (see the section 5.3.Prediction Accuracy).

The above definition of the Markov models and the classification procedure is the basis for our study. All the Markov models and/or extensions to these are derived from this model.

The above classification procedure using the defined Markov models will now be illustrated using an example. Assume that we have an alphabet Σ_{ex} which only consist of 4 different residues:

$$\Sigma_{ex} = \{A, B, C, D\}$$

A single Markov model over the alphabet Σ_{ex} may be depicted as in Figure 6. In the figure the estimated parameters $p(a)$ and $p(a|b)$ are shown.

The drawing is one visual interpretation of the Markov model in question and is very similar to a directed graph.

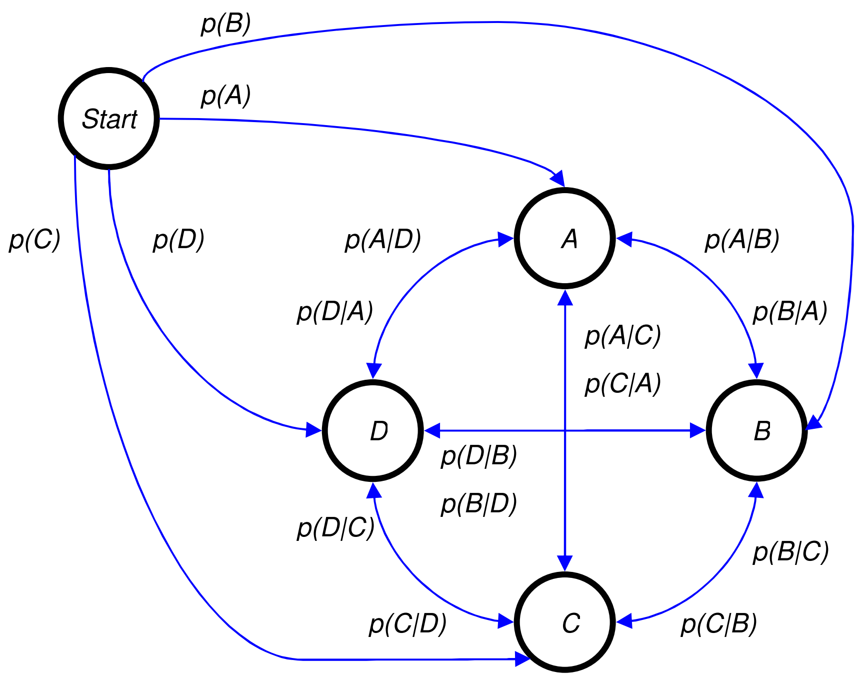


Figure 6: A visual interpretation of a simple Markov model.

The figure is somewhat simplified in that every edge in the graph (except the ones going from the *Start* state) actually represents two edges going in each direction. The parameter *p*(*a*) is the probability of starting in state *a*, i.e. the probability of observing a subsequence in the training database starting with the residue *a*. The *p*(*a*|*b*) parameter is then the probability of going from one state *b* to another *a*.

Beginning in the *Start* state and taking *q* steps would generate a sequence of *q* residues.

Assume that three Markov models of the above kind are trained (that is, the parameters are estimated based on some training data). Each model would represent a classification group (in this case it is the groups H, E and C) and each model could be evaluated (by calculating the score values for each possible window) on some sequence, $S = x_1x_2...x_n$, where $x_i \in \sum_{ex}$ and $i \in [1;n]$.

Given the parameters *p*(*a*) and *p*(*a*|*b*) for each of the three models and assuming that the window size is 3, that is we are using three residues to predict the classification for the first residue in the window, then we may find the predicted classification sequence for the sequence, S_{ex}:

$$\begin{aligned} S_{ex} &= \bar{x}_{ex} = ACCDBAA \\ x_1 &= A \\ x_2 &= C \\ . \\ x_7 &= A \end{aligned} \qquad (3.18$$

Having the three Markov models, one for each classification type, M_H, M_E and M_C, we calculate the score functions for each model and compare them. Input to the Markov models is the first three residues (*ACC*) in the sequence S_{ex} since the window size is 3. The result z_1 is the classification group for the residue at position 1:

$$\begin{aligned} sc_H &= p_H(A)p_H(C \mid A)p_H(C \mid C) \\ sc_E &= p_E(A)p_E(C \mid A)p_E(C \mid C) \\ sc_C &= p_C(A)p_C(C \mid A)p_C(C \mid C) \end{aligned} \tag{3.19}$$

$$z_1 := \arg\max(sc_H, sc_E, sc_C)$$

The model with the highest score value is the most probable classification for the first residue *A*. The process is continued until all possible residues have been classified. In the end a classification sequence is obtained for the sequence S_{ex}:

$$\begin{aligned} S_{ex} &= \bar{x}_{ex} = x_1 x_2 \ldots x_7 \\ S_{classification} &= \bar{z}_{ex} = z_1 z_2 \ldots z_5 \end{aligned} \tag{3.20}$$

The classification sequence will only be of length 5 since the last two residues can not be classified due to the window size of 3 (as mentioned earlier these residues are ignored and not taken into account when the accuracy of the predicted classification is calculated). Using the above procedure it is possible to classify any protein sequence.

In the next subsections a few more terms regarding the Markov models will be explained. These terms may be used throughout the report.

3.3.4. Trivial Classification

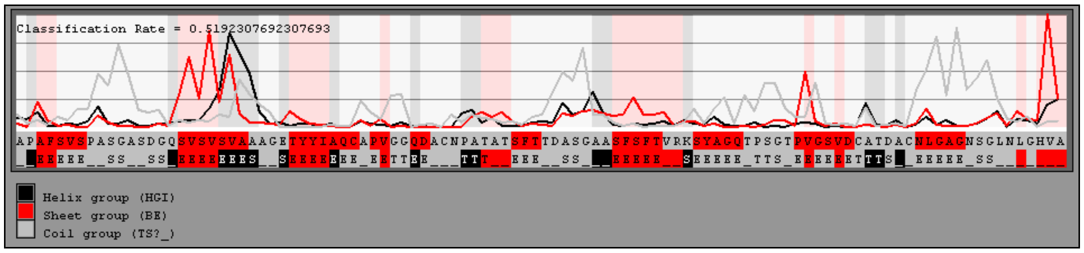
To be able to have a benchmark value when testing the Markov models, a trivial classification may be used. The trivial classification classifies all residues as the most frequent classification group. The most frequent group is determined by investigating the distribution of the different classification groups in the training database used. These analyses and results about the database can be found in the section 6.Test and Training Data.

If the prediction accuracy of the tested Markov models is better than the trivial classification it makes sense to use the Markov models in general.

3.3.5. Output Graph

The implemented program outputs a graph displaying the individual outputs of the Markov models for the particular test sequence. (See the section 8.Implementation for further details regarding the actual program implementation).

Figure 7 is an example of such a graph.



```
Classification Rate = 0.5192307692307693
APAFSVSPASGASDGQSVSVSVAAAGETYYIAQCAPVGGQDACNPATATSFTTDASGAASFSFTVRKSYAGQTPSGTPVGSVDCATDACNLGAGNSGLNLGHVA
__EEEEE__SS__SS_EEEEEEES__SEEEEEEE_EETTEE___TTT__EEE__SS___EEEEE__SEEEEE_TTS_EEEEEEETTTS__EEEEE_SS_______
```

■ Helix group (HGI)
🟥 Sheet group (BE)
□ Coil group (TS?_)

Figure 7: An example of the output displayed when classifying a single sequence.

In the upper left corner the prediction accuracy (denoted as the classification rate) is shown, this fraction shows how well the classification is compared to the correct classification (the value 1.0 is the same as having a 100% correctly classified sequence).

The graph itself is actually three different graphs. There is one graph for each output of the Markov models, M_H, M_E and M_C. For every classified residue the score value for each Markov model is shown in the graph. The score values are connected with lines (to show the tendency of the values) and are colored to show which model they represent.

Under the three graphs there are two lines of letters. The first line is the protein sequence where each letter in the line represents a residue. In the next line the correct classification sequence is presented. The correct classification sequence is given by the protein database. The correct classification sequence consists of several structure types: {*'H','G','I','B','E','S','T','?','_'*}. Our Markov models only predict three different groups, so there is a mapping from the correct classifications used in the database to three classification groups that are used in this context (the H, E and C groups).

The mapping is shown in Table 1.

Classifications in DB	Mapped to group	Name and color code
HGI	H	Helices
BE	E	Beta Sheets
ST?_	C	Coils

Table 1: Color codes and mappings of the classifications in the database to the groups in $\Sigma_{classification}$

When this mapping is applied we derive a new correct classification sequence. This sequence contains only three types of groups (as previously mentioned). These groups are presented by three colors. The mapping showed in Table 1 between the colors and the groups (H, E and C) is used to color the two sequences presented under the graph.

In the first line the correct classification sequence is showed with colors (after it is mapped into three groups). In the next line the Markov model prediction is shown using the same approach. Looking at the first residue in the sequence (*A*) we see that H and E outputs (the score values) from the Markov models are rather small at around 12.5% of the maximum value (each vertical line in the graph represents 25% of the maximum value). The score value of the coil model is about 25%. This means that the coil group is chosen as the (predicted) classification for this residue.

Looking at the first line under the graph we see that the correct classification of the residue is a coil (gray). In the line below we see that the Markov models also have predicted the residue as a coil (gray). This is the way to read the test graphs like the one above.

The graphs are used in later sections to present the ideas and observations, which have led to one or more extensions of the basic Markov model (the initial model which have been presented in the previous subsections).

The following sections will describe the models that we have implemented for prediction of secondary structure in this project. The models are developed based on the initial Markov model defined in the previous sections and also on ideas and observations made during tests of the basic model. These ideas may have led to new extended models which may or may not be better than the original models. In any case the process has been iterative, trying out new ideas and improving those or discarding them.

3.4. The Basic Markov Model

The basic Markov model just presented has two interesting parameters to explore. These parameters are the *window size* and the *pseudo count*. This section will contain an explanation of these parameters and an outline of the test procedure needed to explore the parameters. The section 7.Tests and Results will present the results of these tests.

The first parameter the *window size*, *l*, denotes as previously mentioned the number of terms in the score function. It represents the window of residues $x_i x_{i+1} \ldots x_{i+l-1}$ which is used by each Markov model to determine the classification z_i for the residue x_i. We expect the performance to increase when the window size is increased from one to two. Two residues contain more information than just one.

Having a window size of more than one means that the score function consists of one or more $p(a|b)$ terms. Having a window size of one means that the score function consists of one term only, the $p(a)$ parameter.

A more detailed analysis of the individual terms in the score function $p(a)$ and $p(a|b)$ will follow in the next subsection. At this point, it is uncertain if the model is better or worse when the window size is increased (having a window size, $l > 1$).

Completing the following tests will provide evidence of how the Markov models perform using different window sizes (referenced as Test Case 1).

Test Case 1	Method	Possible Conclusion(s)
Explore the *window size* parameter.	Vary the *window size* and measure the overall performance of the model.	How the *window size* affects the overall performance of the basic Markov model.

Test Case 1: Exploring the window size parameter in the basic Markov model.

The basic Markov model described in this section is almost working as the simple Markov model described in the previous sections. There is one difference however. The basic Markov model incorporates the option of using *pseudo counts*. The following part explains what pseudo counts are and why it may be a good idea to use them.

Having some (limited) training data, estimating the parameters $p(a)$ and $p(a|b)$ in the Markov models may result in some unwanted features. These unwanted features will now be outlined.

Recall the method of estimating the parameters of a Markov model from the last section. In order to estimate the $p(a)$ parameter we need to count how often a given residue will occur (this is done for every residue in the alphabet). Having a limited set of training data could cause some of these counts to be zero. This happens when the residue never appears in the subsequences of the group represented by the Markov model. If the count of residue *a* is zero then $p(a)$ is zero. If the score function uses $p(a)$ then the score function will return zero (because all terms in the score function are multiplied). Similar argumentation can be used to show that the same problem exists for the parameter $p(a|b)$.

The main problem is that a single zero-probability parameter will cause the score function to return zero regardless of all other terms in the function. To avoid this problem a constant (the *pseudo count* constant) is added to the total number of occurrences when counting a certain residue (or pair of residues). The value of the pseudo count constant prevents the probability of any of the parameters $p(a)$ or $\mathrm{p}(a|b)$ from being zero. All terms in the score function will now affect the final score value.

The new pseudo count constant affects the formulas used for estimating the parameters *p*(*a*) and *p*(*a*|*b*) for each Markov model. The parameters are now estimated using the following expressions:

Given that:

a, b	are residues in the alphabet, Σ
y_1^j	is the first residue in subsequence j
k	is the total number of subsequences in the collection
i	is the position in the current subsequence, $i > 0$
c	is the pseudo count constant

$$p(a) := \frac{c + \left|\{j \mid y_1^j = a\}\right|}{|\Sigma| c + k}$$

$$p(a \mid b) := \frac{c + \left|\{(i, j) \mid i > 1 \wedge y_i^j = a \wedge y_{i-1}^j = b\}\right|}{\left|\{(i, j) \mid i > 1 \wedge y_i^j = a\}\right| + |\Sigma| c}$$

(3.21

A proof that these new definitions of the parameters *p*(*a*) and *p*(*a*|*b*) still respect the earlier conditions that each parameter sums to 1 varying *a* over the alphabet, Σ, may be found in Appendix B – Using Pseudo Counts.

Setting the pseudo count constant *c* = 0, reduces the above definitions of the parameters to the original ones in the simple model.

The two new definitions of the parameter estimations introduce the new variable, the pseudo count constant, *c*. A number of tests will investigate the effect (if any) on the overall performance of the basic Markov model. Test Case 2 is aimed at this.

Test Case 2	Method	Possible Conclusion(s)
Explore the *pseudo constant, c.*	Vary *c* and measure the overall performance of the model.	How *c* affects the overall performance of the basic Markov model.

Test Case 2: Investigating the effect of the overall performance of the models when introducing the pseudo count constant.

It is anticipated that the tests using the models that incorporate the pseudo count constant have a better performance than the models not using pseudo counts (*c* = 0), since the individual terms in the score functions will never evaluate to 0 and will therefore not force the score value to 0.

The results of the tests indicate that the window size is the most important parameter. The pseudo count constant appears to be of less importance which means that the training data is large enough to contain most of all possible pairs.

Again the actual tests and evaluation of these tests will be described in the section 7.Tests and Results.

3.5. Extensions to the Basic Markov Model

This section contains a more thorough analysis of the Markov models and presents new extensions or enhancements of the model. This section treats every enhancement and extension separately and comments on the results (specific test results and evaluations are again reserved for the section 7.Tests and Results).

3.5.1. Extensions Based On the Individual Terms in the Score Function

Previously the window size and the pseudo count constant have been investigated. This section analyses the score function (e.g. the individual terms) in more detail. Given a Markov model $M = M_X$ for some group X $\in \sum_{classification}$. Let us recall the score function for the model M in (3.22.

$$sc(\overline{x}) = p(x_i)p(x_{i+1} \mid x_i)p(x_{i+2} \mid x_{i+1})...p(x_{i+l-1} \mid x_{i+l-2}) \qquad \text{(3.22}$$

where the notation from the previous sections is used.

3.5.1.1. Analysis of the Term *p*(*a*)

The first term $p(a)$ in the score function is, as previously described, the probability of the residue a being the first residue in a subsequence. The score function will supply us with a probability of the fact that the residues in the current window represents the first part of a subsequence with the first residue being a.

In the following we will look at a sequence S,

$$S = \overline{x} = x_1 x_2 ... x_n \text{, where } x_1,..., x_n \in \sum$$

and the score functions sc_H, sc_E and sc_C for the three Markov models (M_H, M_E and M_C). Each score function (model) represents one classification group. The sequence S and the correct classification sequence may be given as:

Sequence, *S*	PETILYFGKKRGIALPADTVEELRNIIGMDKPLSLPGFLA
Classification	HHHHHHHHHHHCCCCCCCCHHHHHHHHCCCCCCCHHHHCC

For simplicity we have that the first m residues in S form a subsequence (all the residues are classified as the helix group) and that the window size is much smaller than the size of the subsequence (a window size of 4 will be used in the following). In this case m is 11.

If we assume that the Markov model simulates the real world extremely well then the score function, sc_H, will return a large value whenever the current window is the first part of the subsequence. The other two score functions (for the Markov models M_E and M_C) might return lower values (assuming that there is a connection between the order of certain residues and the groups they are correctly classified as). This means that the value returned by the function sc_H is greater than the values returned by sc_E and sc_C, and the first residue is then classified as a helix, H. This behavior is as expected. The problem appears when the window is shifted by one, and the next residue should be predicted.

This may be illustrated using the above sequence S. Having a window size of 4 the first residue in the sequence S is classified using the score functions (for M_H, M_E and M_C) for the residues in the window (shown in blue):

Sequence, *S*	$\boxed{\text{PETI}}$LYFGKKRGIALPADTVEELRNIIGMDKPLSLPGFLA
Classification	HHHHHHHHHHHCCCCCCCCHHHHHHHHCCCCCCCHHHHCC
Prediction	?

Now the residue at position 1 (P) has been classified as H and the window is shifted one step as in:

```
Sequence, S      PETILYFGKKRGIALPADTVEELRNIIGMDKPLSLPGFLA
Classification   HHHHHHHHHHHCCCCCCCCHHHHHHHHCCCCCCCHHHHCC
Prediction       H?
```

After the window has been shifted by one the window will no longer contain the first residue in the subsequence. The window now consists of the residues at position 2-5.

This means that the term *p*(*a*) will cause the score function sc_H to return a lower value, again we assume that there actually is a connection between the order of certain residues and the groups they are correctly classified as. Now the score function, sc_H may not return a higher value than the score functions sc_E and sc_C. This may result in the second residue being classified as something else than the helix group.

The definition of the *p*(*a*) parameter ensures (to some extent) that the Markov models in many cases are able to predict the first residue of a subsequence correctly, but when predicting the next residues the score values may be much lower which may cause the model to assign wrong classifications.

This phenomenon where the returned score values peak (having large values) at the beginning of a new subsequence is partly illustrated in Figure 8. The figure shows that in several cases there is a peak whenever a new subsequence starts (that is when the sequence of residues changes classification).

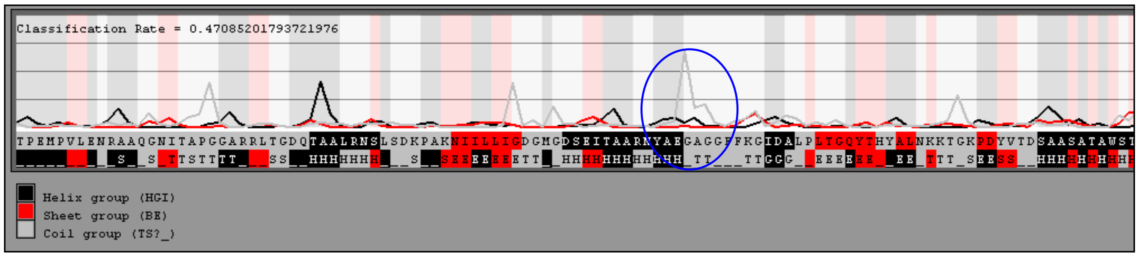


Figure 8: An illustration of a classification for some sequence. The peaks are in several cases positioned in the beginning of a new subsequence. The blue circle shows a gray peak. The peak is positioned over the first residue in the subsequence classified as coils (the C group).

The point of the above reasoning is that the *p*(*a*) parameter could have a large effect on the values returned by the score functions for each Markov model (the effect is larger for small window sizes than for large window sizes). However it may be that the parameter focuses too much on the first residue in a subsequence and may be irrelevant when classifying residues that are not the first in a subsequence.

To test the arguments and statements above we present a new definition of the parameter *p*(*a*) called *p*'(*a*):

- The parameter *p*'(*a*) is the probability that the residue *a* exists (classified as the current group).

Estimating the new parameter denoted *p*'(*a*) may be expressed formally as in the following:

Given that :

$$\begin{array}{ll} a & \text{is a residue in the alphabet, } \Sigma \\ y_i^j & \text{is the residue at position } i \text{ in the subsequence } j, i > 0 \\ n & \text{is the total number residues in all subsequences in the current collection} \\ c & \text{is the pseudo count constant} \end{array} \quad (3.23$$

$$p'(a) \quad := \frac{c + \left|\left\{(i,j) \mid y_i^j = a\right\}\right|}{|\Sigma| c + n}$$

The parameter *p*(*a*|*b*) is estimated as usual. For each Markov model representing a classification group (H, E or C) the parameters *p*'(*a*) and *p*(*a*|*b*) are estimated and the models may be applied as usual for the classification process.

This new definition of one of the parameters defining the Markov models must be tested to see if there is a difference between the original models using the *p*(*a*) parameter and the new models using the *p*'(*a*) parameter. Test Case 3 describes how this will be determined.

Test Case 3	**Method**	**Possible Conclusion(s)**
Is *p*'(*a*) better than *p*(*a*) ?	Set the window size and pseudo count variables as fixed values. Produce two tests: 1) Use the term *p*(*a*) and find the performance of the models. 2) Use the term *p*'(*a*) and find the performance of the models.	*p*'(*a*) is better or worse than *p*(*a*). The test may be inconclusive showing no apparent difference between the two definitions.

Test Case 3: Asserting whether *p*'(*a*) is better or worse than *p*(*a*).

Performing the above test case makes it possible to measure the (possible) difference between the two types of models. Presenting the training data to both models and calculating the performance will reveal which model is most accurate (see the section 7.Tests and Results for specific results). The results show that the new definition of the *p*(*a*) term denoted *p*'(*a*) appears to be better than the original definition.

While analyzing the impact of the *p*(*a*) terms on the resulting score function another test may be interesting. The test will reveal whether the term, *p*(*a*) is necessary at all. If the term *p*(*a*) is left out of the score function, the function is reduced to the following (using the same notation as earlier):

$$\begin{array}{ll} l = 1: & sc_X(S,i,l) = 1 \\ l > 1: & sc_X(S,i,l) = p(x_{i+1} \mid x_i)p(x_{i+2} \mid x_{i+1})...p(x_{i+l-1} \mid x_{i+l-2}) \end{array} \quad (3.24$$

This new definition of the score function is investigated in Test Case 4.

Test Case 4	**Method**	**Possible Conclusion(s)**
Is the terms *p*(*a*) or *p*'(*a*) necessary at all?	Modify the score functions for the individual Markov models (M_H, M_E and M_C) leaving out the *p*(*a*) / *p*'(*a*) term and measure the performance. The performance is compared to the results of the previous tests.	The terms *p*(*a*) or *p*'(*a*) is necessary in order to achieve good performances.

Test Case 4: Testing whether the *p*(*a*) or *p*'(*a*) term is necessary in the individual score functions for the three Markov models, M_H, M_E and M_C.

The tests (see the section 7.Tests and Results) show that the performance of the models are slightly worse indicating that the *p*(*a*) or *p*'(*a*) term is indeed necessary.

3.5.1.2. Analysis of the Term *p*(*a*|*b*)

This section deals with the *p*(*a*|*b*) term to investigate the effect of this term in the score function associated with a given Markov model. The analysis in this section uses the original definition of *p*(*a*) unless otherwise mentioned.

To test the importance of the *p*(*a*|*b*) terms a simple test leaving out all the *p*(*a*|*b*) terms in the score functions of the Markov models may be conducted. The score function is then reduced to one term only, the *p*(*a*) term (this effect may be accomplished setting the window size to 1):

$$sc_X(S,i) = p(x_i) \qquad \textbf{(3.25}$$

Test Case 5 describes the possible tests that may be conducted using this new definition of the score function.

Test Case 5	Method	Possible Conclusion(s)
Does it make sense to include the *p*(*a*|*b*) terms in the score functions for each Markov model?	Set window size = 1and use *p*(*a*) as the only parameter defining the Markov models. The same test may be conducted using the *p*'(*a*) definition of the parameter.	Is it necessary to include the *p*(*a*|*b*) terms to get a reasonable classification?

Test Case 5: Investigating the performance of Markov models based on score functions where the *p*(*a*|*b*) terms are left out.

This test reveals that the *p*(*a*|*b*) terms are indeed necessary in order to get a reasonable result (see the section 7.Tests and Results). This result is anticipated since the Markov models do not contain much information when using only the *p*(*a*) parameter.

3.5.1.3. The Reversed Pair Method

Knowing that the *p*(*a*|*b*) parameter is crucial for the Markov model, we investigate possible changes for this parameter. With the Markov assumption in mind the above tests show that the order of the residues in the sequence to be predicted appears to be important for the model.

In some of our earlier tests we applied a trick, which may seem random at first glance. The trick was used when estimating the parameters *p*(*a*) and *p*(*a*|*b*). The idea was that during the process of extracting all the subsequences in the training data for each classification, whenever the subsequence $S_{subsequence} = x_1x_2...x_n$ was observed the reversed subsequence, $S_{reversed} = x_nx_{n\text{-}1}...x_1$ was also added to the collection of subsequences for that particular classification group (normally only the non-reversed subsequence would be added).

The reason for this was to somehow supply the Markov models with more information on the subsequences extracted, in this case information on the reversed subsequences.

Looking more closely at how this trick affects the individual parameters estimated in the model (especially the *p*(*a*|*b*) parameter) reveals some interesting features. This is illustrated with a series of examples.

Having the subsequence S_x = *ABBAAAAB* and the reversed subsequence $S_{reversed}$ = *BAAAABBA*. Now assume that we want to estimate the parameter *p*(*A*|*B*) as usual on the subsequence S_x as would normally be done. This may be illustrated in the following:

Sequence, *S*	ABBAAAAB	
Pairs (*BA*)	BA	1
A after the first position.	AAAA	4

This results in the parameter *p*(*A*|*B*) being estimated as 1/4 (assuming that this subsequence is the only one in the collection for the current group).

Now trying to estimate the parameter *p*(*A*|*B*) on both the subsequence S_x and the reversed subsequence $S_{reversed}$ yields another value for the *p*(*A*|*B*) parameter.

Sequence, *S*	ABBAAAAB, BAAAABBA	
Pairs (*BA*)	BA BA BA	3
A after the first position.	AAAA AAAA A	9

The resulting value for the *p*(*A*|*B*) parameter is 3/9. This is (as expected) different from the *p*(*A*|*B*) parameter estimated using only the subsequence S_x. The trick just described is therefore a new definition of the parameter *p*(*A*|*B*) which will be denoted *p'*(*A*|*B*) in the following.

Looking more closely at the way the *p'*(*A*|*B*) parameter is determined reveals that the *p'*(*A*|*B*) parameter may be estimated using the procedure for estimating *p*(*A*|*B*) as usual and in addition extending this procedure a bit. This will be explained now.

When estimating the parameter *p*(*a*|*b*) every subsequence for the current group is investigated. Normally this would include estimating the probability of finding the pair *ba* which is done by counting the number of times the *ba* pair occurs in total and dividing that by the number of times *a* occurs after the first position in the subsequence.

Now we extend this procedure. In addition to the above, we check for the reversed pair *ab* in the following manner. The number of times the pair *ab* occurs in the subsequence is counted and the number of times the residue *a* occurs in any position except the last is counted. This corresponds to the definition of the *p'*(*a*|*b*) parameter above.

The above is illustrated with an example.

Having a single subsequence, S_x = *ABBAAAAB*, over the alphabet Σ_x = {*A*,*B*} the parameter *p'*(*A*|*B*) is estimated in the following way.

First the normal procedure is applied. The number of times the pair *BA* occurs is summed up and the number of times *A* occurs after the first position is summed up:

Sequence, *S*	ABBAAAAB	
Pairs (*BA*)	BA	1
A after the first position.	AAAA	4

This returns 1 and 4 respectively. Now the new trick is applied for the same subsequence, counting the number of times that the pair *AB* occurs and the number of times *A* occurs except in the last position of the subsequence.

Sequence, *S*	ABBAAAAB	
Pairs (*AB*)	AB AB	2
A before last position.	A AAAA	5

This returns 2 and 5 respectively. These results are added to the result obtained in the first investigation, resulting in 3 (1 + 2) and 9 (4 + 5) which is 3/9. This is the same as when reversing the subsequences and using the original method for estimating *p*(*a*|*b*). The method will be referred to as the *reversed pair* method, since the pair *ba* is reversed to *ab* in the above procedure.

Introducing the parameter *p*'(*a*|*b*) in the above way ensures that the conditions for the parameters defining the Markov models are respected (see (3.14.)

The new definition of the parameter *p*'(*a*|*b*) described informally above may be given more formally as in the following.

$$\begin{array}{ll}\text{Given that :} & \\ a,b & \text{are residues in the alphabet, } \Sigma \\ y_i^j & \text{is the residue at position i in the subsequence } j, i > 0 \\ z_i^{\;m} & \text{is the residue at position i in the reversed subsequence } m, \text{i} > 0 \\ c & \text{is the pseudo count constant}\end{array} \qquad \textbf{(3.26}$$

$$p'(a \mid b) := \frac{c + \left|\{(i,j) \mid i > 1 \wedge y_i^j = a \wedge y_{i-1}^j = b\}\right| + \left|\{(i,j) \mid i > 1 \wedge z_i^{\;m} = a \wedge z_{i-1}^m = b\}\right|}{\left|\{(i,j) \mid i > 1 \wedge y_i^j = a\}\right| + \left|\{(i,j) \mid i > 1 \wedge z_i^{\;m} = a\}\right| + |\Sigma| c}$$

The description given above where the subsequences are reversed does in fact have an impact on the *p*(*a*) parameter as well. However defining the parameter *p'*(*a*|*b*) separately makes it possible to keep the original *p*(*a*) definition.

The reason for substituting the *p*(*a*|*b*) parameter with this new definition *p'*(*a*|*b*) is that this model may prove to be better than the original model. This assumption is based on observations using this new definition *p'*(*a*|*b*) instead of *p*(*a*|*b*).

Test Case 6 explores the introduction of *p*'(*a*|*b*).

Test Case 6	**Method**	**Possible Conclusion(s)**
Does the *reversed pair* method increase the performance (compared to the original model) ?	Substitute the *p*(*a*|*b*) parameter with the new definition *p*'(*a*|*b*) and conduct a test session as usual. The same test may be conducted using *p*'(*a*) instead of *p*(*a*).	The *reversed pair* method proves to increases or decreases the performance compared to the original model.

Test Case 6: Exploring the new definition of the *p*(*a*|*b*) parameter denoted *p*'(*a*|*b*).

The test results indicate that the performance actually do increase when this new definition *p'*(*a*|*b*) is introduced (see 7.2.2.1.3.The Reversed Pair Method).

To find out how this extension affects the model we look at the *p*(*a*|*b*) and *p'(a|b)* parameters and see how they relate. Recall that the parameter *p'(a|b)* is estimated as the original *p*(*a*|*b*) parameter including the reversed subsequences. Let us recall the example presented earlier where we estimated the parameter *p'(a|b)*:

Sequence, *S*	ABBAAAAB	
Pairs (*BA*)	BA	1
A after the first position.	AAAA	4

In the first step the values (counting the pair *BA* and number of *A*'s) are found as usual. Now in the second step for estimating *p'(a|b)* the number of times the reversed pair occurs is counted and divided with the number of times *A* occurs before the last position.

Sequence, *S*	`ABBAAAAB`	
Pairs (*AB*)	`AB    AB`	2
A before last position	`A  AAAA`	5

We can now see that the number of times *A* occurs is not the same in step one and step two. This is because there is an *A* in the first position of the sequence which means that the first step only finds 4 occurrences of *A*. If the first and the last residue in the subsequence is not an *A*, the same number of *A*'s will be found in step one and step two. In other words the sum of the values found for *A* in step one and step two will be close to 2*n* where *n* is the total number of *A*'s in the subsequences.

To find the final result of *p'(a|b)* we add the number of normal and reversed pairs: 2+1 = 3 and this number is divided with the total number of times *A* occurs (using the rules of skipping the first and last position already described) resulting in a total count of 9 in the example.

The above description of *p'(a|b)* is transferred into a more general description using that the number of times *a* occur will be close to 2*n*. An approximated formula for estimating *p'(a|b)* may now be given:

$$p'(a \mid b) \cong \frac{pairs_{rev} + pairs_{normal}}{2n} = \frac{\left(\dfrac{pairs_{rev} + pairs_{normal}}{2}\right)}{n} = \frac{pairs_{avr}}{n} \tag{3.27}$$

where:

pairs$_{rev}$ is the number of reversed pairs
pairs$_{normal}$ is the number of normal pairs
pairs$_{avr}$ is the average of the sum of reversed and normal pairs.
n is the number of times *a* occurs using the usual rules.

If *pairs*$_{avr}$ is replaced with *pairs*$_{normal}$ the definition of *p'(a|b)* looks like the original definition of *p*(*a*|*b*). The formula may be used to see that the new definition of *p'(a|b)* means that the value now also depends on the reversed pair *ab* in the subsequence instead of *only* the original pair *ba* in the subsequence.

If the pair *ba* is more frequent than the pair *ab* then the actual value of *p'(a|b)* is reduced compared to the value of *p*(*a*|*b*). This will also mean that the value of *p'*(*b*|*a*) is increased compared to the value of *p*(*b*|*a*).

3.5.1.4. Using Only *p'*(*a*) Terms in the Score Function

The Markov models take the order of the residues into account by using the term *p*(*a*|*b*) as mentioned several times in the previous sections. Tests have already been conducted leaving out the *p*(*a*|*b*) term and resulting in a very simple model (which corresponds to using window size 1). This model was too simple.

However if we discard the information about the order of the residues it is possible to come up with the following score function (given the usual conditions):

$$sc_X(S,i,l) = p'(x_i)p'(x_{i+1})\dots p'(x_{i+l-1}) \tag{3.28}$$

This will give us the probability of the residues occurring in the current window (starting from index *i* and ending with index *i*+*l*-1). Test Case 7 describes a test session monitoring the performance of the models using the above score function.

Test Case 7	Method	Possible Conclusion(s)
Does the new score function (3.28 affect the performance of the models?	Complete a training session using the new score function for each Markov model, M_H, M_E and M_C.	Is this method better or worse than the original basic model?

Test Case 7: Testing the models without the order constraints between the residues.

This new definition of the score function appears to be slightly worse than the model using p'(a) and p(a|b) – using window size 3 it is actually better though. The interesting part is that the new score function presented above, does not contain any dependencies between the residues (as it did before using the term *p*(*a*|*b*)). In other words, there is no information about the order of the residues in the window.

Distracting from the fact that the score function supplies a probability and seeing it as a simple score function just producing a number, we could try altering the score function to the following:

$$sc_X(S,i,l) = p'(x_i)^l \, p'(x_{i+1})^{l-1} \ldots p'(x_{i+l-1})^1 \qquad \text{(3.29}$$

Looking a bit closer at this expression we see that the first term is to the power of *l* (the window size). We already know that all terms are 1.0 or less which means that the exponent acts like a discount factor. If the first term is near 1.0 then it means that the residue represented by this term is very likely to be a part of the group that the score function (Markov model) represents. The fact that the first term is to the power of *l* means that if this term is near 0.0 the resulting value of the score function will drop heavily. This effect decreases as the power of the terms decreases. The last term is to the power of 1 and is therefore not weighted in any way.

In other words this new approach demands that the first residues in the window have to be observed frequently to avoid the score value to drop. If the value of the score function drops too much it will lose when it is compared to the other two score functions (representing the two other groups).

Test Case 8 investigates the performance of the models using this new definition of the score function.

Test Case 8	Method	Possible Conclusion(s)
Does the new score function (3.29 affect the performance of the models?	Complete a training session using the new score function for each Markov model, M_H, M_E and M_C.	Is this method better or worse than the original basic model?

Test Case 8: Test the altered score functions impact on the performance of the models.

The test is compared to the basic model where *p*(*a*) is replaced with *p*'(*a*) and where the *p*(*a*|*b*) terms are used as usual. The results show that the method using the discount factor yields better results than both the basic model and the basic model with *p*(*a*) replaced by *p'*(*a*). A possible reason why the performance of the models using the discount function are increased compared to the simple order less score function (not using the discount factor) is that the discount introduces a sort of order again. The discount factor causes the score function to change value if the position of the residues in the window are changed (this was not a fact before the discount factor was added). The section 7.Tests and Results shows the results of the tests.

3.5.2. Extensions Based On the Overall Score Function

This section describes various extensions that are primarily based on changing the overall score function of the Markov models. It is actually possible to increase the performance of the models by using these extensions (of course some are better than others). Every modification and alteration is based on the original basic Markov model.

3.5.2.1. Adding Noise

It is not uncommon that algorithms are using random terms when making decisions. Based on this fact the Markov models are altered to incorporate random noise.

Using the Markov models for classifying protein sequences involve calculating three score values (one for each classification group). Suppose random noise is added to the results returned by the three different score functions, before the score values are compared.

Having a function *rand()* which returns a real random number in the interval [0;1[and a scale factor *s* (determining how much noise is added) the new overall score function, sc'_X for the Markov model M_X may be given as:

$$sc'_X = sc_X + s\left(rand() - \frac{1}{2}\right) \qquad \text{(3.30}$$

where sc_X is the original score function defined earlier.

Using the scale factor *s* it is possible to add different amounts of noise to the score value (*s* is the same for each Markov model M_H, M_E and M_C). The noise added is in the interval [-0.5s;+0.5s[.

Test Case 9 explores adding noise to the models.

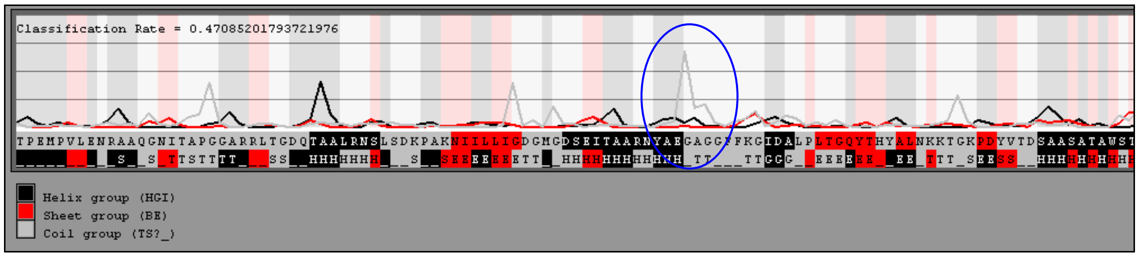
Test Case 9	Method	Possible Conclusion(s)
How are the models affected when noise is added?	Test the models in iterations using different amounts of noise from s=0 to s=$1.0\cdot10^{-4}$.	This method is worse or better than the basic Markov model.

Test Case 9: Exploring the noise amount added to the models.

The tests show that the models perform worse when adding noise. This subject is not investigated any further.

3.5.2.2. Using Decision Constants

The following extensions are mainly based upon observations from a typical output of a classification graph (for a single test sequence). Figure 9 show such a classification graph.



```
Classification Rate = 0.47085201793721976
TPEMPVLENRAAQGNITAPGGARRLTGDQTAALRNSLSDKPAKNIILLIGDGMGDSEITAARNYAEGAGGFFKGIDALPLTGQYTHYALNKKTGKPDYVTDSAASATAWST
________S__S_TTSTTTTT___SS__HHHHHHH___S__SEEEEEEETT__HHHHHHHHHHHHH_TT___TTGGG__EEEEEEE__EE_TTT_SEESS__HHHHHHHHHH
```

■ Helix group (HGI)
🟥 Sheet group (BE)
□ Coil group (TS?_)

Figure 9: An example of a classification graph showing the score value for each classification group and both the correct and the predicted classification sequence. One peak is pointed out, which will be investigated further in the following.

The classification graph shows a number of peaks. In the section 3.5.1.1.Analysis of the Term *p*(*a*) an explanation is given why such peaks may occur at the start of a new subsequence. These peaks often seem to appear whenever the Markov model finds a new subsequence. The peaks usually exist for one residue only.

If a window is placed at position *i* in a given sequence, the peaks typically start at the previous position in the sequence *i*-1 with a low performance (possible incorrect classification). After that the performance increases drastic at position *i* (which is then correctly classified) and then decreases again at position *i*+1.

Figure 9 shows such a peak (marked with a blue circle). The peak starts at a subsequence and ensures that the first residue in the subsequence is predicted correctly as a coil. In this case the following residues are predicted correct but problems could occur after the first residue. The value of the score function representing the coil group (and the peak) drops after position *i* in the sequence and it is possible for the other score functions (for the H and E group) to win the comparison and cause the prediction to be incorrect.

Introducing a new rule may avoid this problem in some cases. The following states the rule:

- A new winning group (represented by a Markov model) is only accepted whenever the score function representing the group returns a larger value than the second greatest score function (representing some other group) by a certain amount.

This new rule will be explained in details in the following.

Assume that we have already classified the residue at position *i*-1 as a coil and we are ready to classify the next residue at position *i*. Now this is normally accomplished by calculating the score values using the three score functions for every Markov model M_H, M_E and M_C and assigning the classification group associated with the highest score value.

Now this rule is changed a bit. Since the last residue was classified as a coil, the next residue may be one as well. Now for some of the other groups (H and E) to be assigned as the classification for that residue, the highest score value returned must be higher than the next highest score value by some amount. If not the classification for the residue in position *i*-1 (coil in this case) is assigned as the classification for the current residue at position *i*.

Given that the residue at position *i*-1 is classified as a coil, and the residue at position *i* must be classified now, the classification is found calculating the three score values for each group:

$$\begin{aligned} sc_H &= x \\ sc_E &= y \\ sc_C &= z \\ sc_{winner} &= \max(x, y, z) \end{aligned} \tag{3.31}$$

Assume that $sc_{winner} = x$. Now the helix group is assigned as the new classification if and only if the following is true:

$$(sc_{winner} - y) \geq r_1 \wedge (sc_{winner} - z) \geq r_1 \tag{3.32}$$

where r_1 is the decision constant (the difference value), which may be chosen experimentally.

If the above expression is not true, then the group classified for the residue at position *i*-1 is assigned as the classification for the residue at position *i*. This is called the *difference method*, since the difference between the highest scoring model and the other two are compared with the decision constant r_1.

Another method to decide which classification should be assigned for the residue at position *i* is called the *ratio method*. It is similar to the difference method, but now the ratio between the highest score value and the other two are compared with a decision constant, r_2 and may be written as (following the example used for the difference method):

$$\frac{sc_{winner}}{y} \geq r_2 \wedge \frac{sc_{winner}}{z} \geq r_2 \tag{3.33}$$

where r_2 is the decision constant. If this expression is true then the helix group is assigned as the classification for the residue at position *i*.

The two decision constants used in the above definitions may be determined experimentally performing some tests for several distinct values of the constants. One way to extend this method is to have several decision constants, one for each classification group. This has not been tested though.

Test Case 10 explores using decision constants.

Test Case 10	**Method**	**Possible Conclusion(s)**
Conduct a test using the difference method.	Complete several test sessions varying the decision constant r_1 and observe the results.	The difference method increases or decreases performance.
Conduct a test using the ratio method.	Complete several test sessions varying the decision constant r_2 and observe the results.	The ratio method increases or decreases performance.

Test Case 10: Using decision constants.

Tests have confirmed that both of these methods lead to an increase in performance (compared to the basic Markov model).

Another method has been tried out. Instead of assigning the previous classification for the current residue, we assign the most frequent classification. This method decreases the performance of the models.

Again the specific tests and results are presented and commented in the section 7.Tests and Results.

3.5.2.2.1. The Ratio-II Method

This subsection presents yet another method (denoted the *ratio-II method*) which affects the way the classifications are chosen for each residue in the sequence. Suppose we want to ensure that a drastic change in the highest score value occurs from position *i*-1 to position *i*, before a new group is assigned as the classification for the residue at position *i*.

This may be accomplished calculating the ratio between the highest score value in position *i* and in position *i-1*. If this ratio exceeds some constant, r_3, then the group having the highest score value is chosen as the classification group for the residue at position *i*.

This ratio is calculated as:

$$ratio = \frac{\max\left(sc_H(S,i,l), sc_E(S,i,l), sc_C(S,i,l)\right)}{\max\left(sc_H(S,i-1,l), sc_E(S,i-1,l), sc_C(S,i-1,l)\right)} \tag{3.34}$$

To assign the group associated with the highest score value for the residue at position *i*, it must be true that *ratio* ≥ r_3, where r_3 is the decision constant. If this is not the case the classification for the previous residue is assigned for the current residue.

The decision constant may be determined running several tests and varying the size. This is investigated in Test Case 11.

Test Case 11	**Method**	**Possible conclusion(s)**
Apply the *ratio-II method*	Vary the constant r_3 and observed the performance of the models.	The method increases or decreases the performance.

Test Case 11: Testing the *ratio-II method*.

Tests show that performance of the models is increased using this method compared to the performance of the basic Markov model (see the section 7.Tests and Results).

3.5.2.3. The Momentum Method

This subsection introduces the *momentum method*, which is a way to let the score function for step *i* be dependent on the score function for one or more steps before *i*. In other words the score value found for the residue at position *i* is dependent on the score values found for one or more of the previous residues.

The peaks observed in the classification graph (which have been discussed earlier) are the main reason behind using the momentum methods. A peak means that the value of one of the score functions is high. It also means that the probability of classifying the residue as the group the score function represents is high.

One of the problems addressed earlier is that the high value of the score function only exists for the current residue and not for the next. In order to let the score function remembers that it just recognized a group with high probability we introduce the momentum. The momentum will act like a sort of memory meaning that after a few residues the memory of the peak still remains. After a while though, the peak value observed for some previous residue should be of less importance.

The behavior described above can be implemented by a momentum. A new temporary function $temp_X$ is introduced (where $X \in \Sigma_{classification}$). This function depends on the original score function but is used in the comparison when the winning group is found instead of the score function.

Given a sequence *S*, and *i* as the current position in the sequence and the window size *l*, the temporary function may be defined using a weight *w* and the already defined score function, *sc*:

$$temp_X(S,i,l) = sc_X(S,i,l) + w \cdot sc(S,i-1,l) \qquad \textbf{(3.35}$$

The function introduces a new variable *w* which determines how much the previous score value is weighted. This means that the new temporary function is based on the previous value of the score function (weighted using constant *w*) and the value of the score function calculated for the current position in the sequence. The method is investigated in Test Case 12.

Test Case 12	**Method**	**Possible Conclusion(s)**
Testing the primitive momentum method.	Run several test sessions varying the weight parameter and measure the performance of the models.	The primitive momentum method increases or decreases performance.

Test Case 12: Exploring the primitive momentum method by varying the weight, *w*.

The results using the momentum method defined above is as good (in fact slightly better) as most of the previous extensions. Investigating the formula a bit further reveals that the *memory* only goes one step back (that is the temporary function is only affected by the last value of the score function). This version of the momentum method is also called *primitive momentum*.

Updating the current value of the score function will cause the current score value to have memory of all previous values of the score functions. The method is referred to as *momentum* or *real momentum*. The following formula shows the effect:

$$sc(S,i,l) := sc(S,i,l) + w \cdot sc(S,i-1,l) \qquad \text{(3.36}$$

where the same parameters as before are used.

To show that the function incorporates memory it is possible to resolve the previous values of the score function:

$$\begin{aligned} sc(S,i,l) &:= sc(S,i,l) + w \cdot sc(S,i-1,l) \\ sc(S,i,l) &:= sc(S,i,l) + w \cdot \left[sc(S,i-1,l) + w \cdot sc(S,i-2,l) \right] \\ sc(S,i,l) &:= sc(S,i,l) + w \cdot sc(S,i-1,l) + w^2 \cdot sc(S,i-2,l) \end{aligned} \qquad \text{(3.37}$$

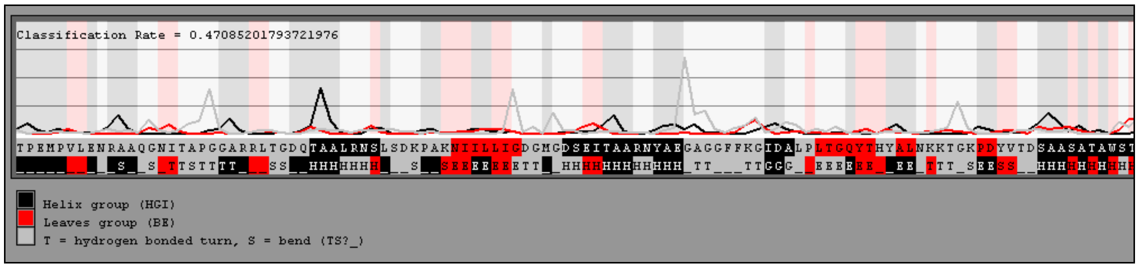
Continuing this way it is possible to expand the formula until i-n = 1 where n is the number of possible steps one may go back in the sequence. This method is contained in Test Case 13.

Test Case 13	**Method**	**Possible Conclusion(s)**
Testing the real momentum method.	Run several test session varying the weight parameter and measure the performance of the models.	The real momentum method increases or decreases performance.

Test Case 13: Exploring the real momentum method by varying the weight, *w* and incorporating more memory.

This method yields better results than the first momentum method (see the section 7.Tests and Results).

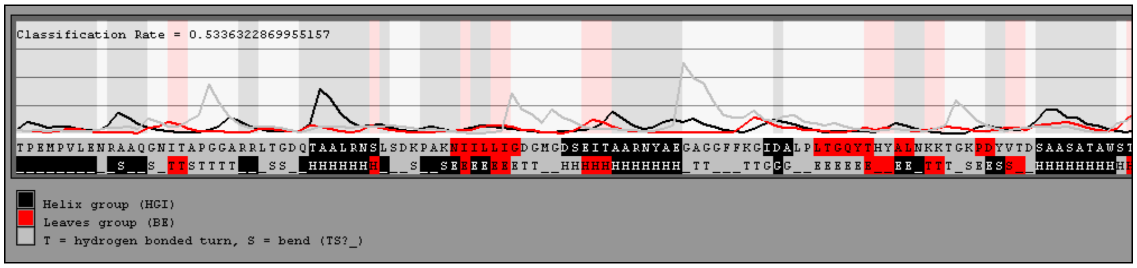
It is interesting to see the outputted classification graphs when using the basic model and the real momentum method. The peaks have clearly been smoothed out a bit. The first figure shows the output graph for the first part of a sequence using the basic Markov model without momentum (see Figure 10). The second figure shows the output graph when the real momentum method is used (see Figure 11).



```
Classification Rate = 0.47085201793721976
TPEMPVLENRAAQGNITAPGGARRLTGDQTAALRNSLSDKPAKNIILLIGDGMGDSEITAARNYAEGAGGFFKGIDALPLTGQYTHYALNKKTGKPDYVTDSAASATAWST
________S__S_TTSTTTT____SS__HHHHHHHH___S__SEEEEEEETT__HHHHHHHHHHHHH_TT___TTGGG__EEEEEE__EE_TTT_SEESS__HHHHHHHHH
```

■	Helix group (HGI)
🟥	Leaves group (BE)
□	T = hydrogen bonded turn, S = bend (TS?_)

Figure 10: An example of an output graph showing the output of the score functions (using the basic Markov model).



```
Classification Rate = 0.5336322869955157

TPEMPVLENRAAQGNITAPGGARRLTGDQTAALRNSLSDKPAKNIILLIGDGMGDSEITAARNYAEGAGGFFKGIDALPLTGQYTHYALNKKTGKPDYVTDSAASATAWST
__________S___S_TTSTTTTT___SS___HHHHHHH___S___SEEEEEEETT__HHHHHHHHHHH_TT___TTGGG__EEEEEE__EE_TTT_SEESS__HHHHHHHHHH
```

■ Helix group (HGI)
🟥 Leaves group (BE)
□ T = hydrogen bonded turn, S = bend (TS?_)

Figure 11: An output graph of the same sequence used in Figure 10, except now the real momentum method is applied. The memory effect is definitely visible.

3.5.3. Extensions Based On Alternating the Training Procedure

The training of the Markov model is the same thing as estimating the parameters *p*(*a*) and *p*(*a*|*b*) for that model. When the parameters are estimated all subsequences for each classification group are extracted from all sequences in the training data.

A new way of extracting the subsequences is now introduced. The reason for introducing this new method will become apparent later in this section. The following describes the new method for extracting subsequences.

The window size used is given by *l*. For every extracted subsequence the next *l*-1 residues in the sequence are appended to the subsequence. Given a sequence *S* of length *n* and a subsequence S_{sub} of length (*j*-*i*+1) in *S*:

$$
\begin{aligned}
&S = x_1 x_2 \ldots x_n \\
&S_{sub} = x_i x_{i+1} \ldots x_j , \qquad 1 \le i \le j \le n
\end{aligned}
\tag{3.38}
$$

Given the window size *l*, the new subsequence may be generated by appending the next *l*-1 residues from *S*:

$$
S_{newsub} = x_i x_{i+1} \ldots x_j x_{j+1} \ldots x_{j+l-1} \tag{3.39}
$$

It is important to note that the number of appended residues is the same as the window size minus 1.

This is illustrated with an example. Suppose we have the sequence, S_x:

Sequence, S_x	TPAFNKPKVELHVHLDGAIKPETILYFGKKRGIALPADTVEELRNIIGMDKPLSLPGFLA
Classification	CCCCCCCEEEEEEEHHHCCCHHHHHHHHHHHCCCCCCCCHHHHHHHHCCCCCCCHHHHCC

Now assume that we want to extract the subsequences for the coil group and assume that the window size is 3. First the subsequences are extracted as usual. After that the next 2 residues (the window size minus 1) are appended to the subsequences:

```
Sequence, Sx        TPAFNKPKVELHVHLDGAIKPETILYFGKKRGIALPADTVEELRNIIGMDKPLSLPGFLA
Classification      CCCCCCCEEEEEEEHHHCCCHHHHHHHHHHHCCCCCCCCHHHHHHHHCCCCCCCHHHHCC

1. subsequence      TPAFNKPKV
2. subsequence                       AIKPE
3. subsequence                                      GIALPADTVE
4. subsequence                                                     GMDKPLSLP
5. subsequence                                                              LA
```

The procedure yields the subsequences above. Note that the last subsequence is extracted as usual, since there are no residues to append. The Markov models will now be trained using the new subsequences.

The idea behind using this method will now be explained. Assume that all subsequences are extracted in the original way (described in the introduction). The parameters for the Markov models are estimated as usual and some sequence S is to be classified.

Now at some point when classifying the sequence S, the window will be placed in such a way that it overlaps (at least) two different subsequences (for different groups). Now since each Markov model, M_H, M_E and M_C are only trained on the subsequences for the H, E and C group respectively, one may assume that the models have problems classifying whenever the window overlaps two or more groups. Assuming that the order of the residues in the sequence is connected to the classifications, then the score functions will return small probabilities for these situations where the windows overlaps different groups, since those situations are not in the training subsequences.

Including the next l-1 residues in each subsequence extracted means that the models are trained on these overlaps as well. The assumption is that the models may be better to classify the residues around the overlapping sections in the sequences.

This method is dealt with in Test Case 14.

Test Case 14	Method	Possible Conclusion(s)
Test the new method of generating the subsequences for training of the models.	Conduct a test session using this new method.	The models may perform better, incorporating knowledge about the overlapping subsequences.

Test Case 14: Testing the new method of extracting the subsequences.

Implementing the new training method does not yield better results. The issue could be that a new problem is introduced.

Assume that the window is placed at the first residue of a subsequence starting at index i in the sequence S. The first l-1 residues (beginning at position i) have been seen by the Markov model that represents the correct classification group for the residue at position i-1, but these residues has also been seen by the Markov model representing the correct classification of the current residue (at position i).

The method introduced above means that the Markov models representing different groups are in fact trained on parts of the same data. This introduces new problems when classifying the residues, for which more than one model are trained on. The method might have that effect and will therefore cause the classification for these residues to be unreliable.

Using the original training method (only using the exact subsequences for the particular group) the Markov models had problems classifying the ends of the subsequences. Now the problem has been moved to the beginning of the subsequences.

The next subsection analyses this potential problem in more details.

3.5.3.1. Classifying the Ends of Subsequences

In order to clarify whether the Markov models have problems classifying the ends of the subsequences a new test is constructed. The idea is that the residues, which are believed to be problematic for the models to classify, are skipped. If the test shows that the performance of the model is increased by skipping the problematic residues, we may conclude that the model has problems classifying residues whenever the window overlaps two or more subsequences.

The test does not result in another possible classifier, since it directly uses information from the correct classification sequences associated with each sequence in the training data.

The correct classifications associated for each sequence are used to detect whenever a new subsequence starts and this knowledge is used in the following:

Assuming that the window size is *l*:

- Whenever the window contains residues of two or more different subsequences the window is shifted to the beginning of the next subsequence (i.e. skipping the residues in between).
- Skipped residues are not classified. The skipped residues are not taken into account when calculating the accuracy of the models.

In this way the ends of the subsequences are not classified. Given a subsequences of length n and window size l where $n \geq l$. The first n-l+1 residues will be classified using the standard method and the rest of the residues will be skipped. If $n < l$ then all residues in the subsequence are skipped (that is, not classified). Test Case 15 is used to see if the Markov model really has problems classifying the ends of the subsequences.

Test Case 15	**Method**	**Possible Conclusion(s)**
Test whether or not the Markov model fails to predict overlapping subsequences in window.	Whenever the window contains residues of two or more different subsequences the window is shifted to the beginning of the next subsequence. All skipped residues are not classified. The window size is varied.	If performance increases the Markov models may have problems classifying small subsequences or ends of the subsequences.

Test Case 15: Skipping classification of ends of the subsequences.

Test results show that the performance of the model increases when the window size is increased. Further analysis of the data shows surprising results. It seems that it is not possible to conclude whether or not the Markov models have problems classifying the ends of the subsequences using the above test. The reason is that by skipping the residues (and thereby the subsequences of smaller lengths than the window size) the distribution of the classification groups (H, E and C) in the actual training data is changed.

Using the calculated accuracy matrices it is possible to see the exact distribution of the three classification groups. This distribution reveals that the helix group gets more frequent when the window size is increased. This also means that the prediction accuracy of the trivial classifier (now always predicting the helix group) is increased. The performance measured in the test is compared to the trivial classifier for every window size. This shows that the actual performance has not increased. For more details regarding the tests and results see section 7.Tests and Results.

3.5.3.2. The Swap Residue Method

A new method called *swap method* is introduced. The idea is to investigate whether it is possible that two residues right next to each other may switch place in the sequence of residues without affecting the correct classification sequence for the entire sequence.

Again the estimated parameters $p(a)$ and $p(a|b)$ defining the Markov models are affected by this trick. Suppose a subsequence $S_x = x_1x_2...x_n$ is given, then for each possible pair in the sequence, the residues in the pair are switched (one pair at a time) yielding n-1 new subsequences. These subsequences (plus the original one) are all added to the collection of subsequences for that particular group during the extraction of subsequences (prior parameter estimation).

Then the parameters $p(a)$ and $p(a|b)$ defining the Markov model, M_X may be estimated as usual, now using more subsequences (and therefore more information). The method will be illustrated with an example.
Given the subsequence S_x = *AGGTANSCL* the following subsequences are generated:

Original subsequence, S_x	AGGTANSCL
1.	GAGTANSCL
2.	AGGTANSCL
3.	AGTGANSCL
4.	AGGATNSCL
5.	AGGTNASCL
6.	AGGTASNCL
7.	AGGTANCSL
8.	AGGTANSLC

The residues which switch place are marked in blue. The procedure results in many more subsequences in the collections for each classification group. As mentioned earlier the parameters $p(a)$ and $p(a|b)$ are affected using this method, but by adding the subsequences in the way described above the same definition of the parameters may be used (only now they are used on more subsequences, which are not really in the training data – but generated prior to the training).

The method described as the *swap residue* method is tested using Test Case 16.

Test Case 16	Method	Possible Conclusion(s)
Doest the added subsequences (the swap residue method) increase the performance of the models?	Complete a training session using this method, by adding the subsequences generated from the original one actually in the training data.	The method may increase or decrease the performance of the models.

Test Case 16: Testing the new method, the *swap residue* method.

The test results indicate that the method increases the performance of the Markov models, leading to the possible conclusion that two residues may actually switch place without affecting the correct classification sequence.

3.6. Normalizing the Markov Models

This section treats the possibility to add up values of score functions for different window sizes (but for the same classification group). The main problem is that it is actually not possible to compare the values of two different score functions if they are calculated using a different window size. The number of terms in the score function depends on the window size. When the number of terms increases the value of the score function is getting smaller. So when comparing two score values based on different window sizes, then the score value obtained using the model with the smallest windows size will most likely dominate (because the associated score function contains fewer terms). To overcome this problem some sort of normalization scheme has to be used.

If the normalization problem is solved, the solution may be extended to combine several models and hopefully increase the performance of the combined models compared to one model having a specific window size. Instead of just having one Markov model for each group with a fixed window size, combining several models for the same group with different window sizes might make the overall model more flexible. Since the subsequences vary in length, it seems like a good idea to combine models that actually uses different window sizes.

Two such methods for normalizing the output of the Markov models have been implemented, making it possible to compare and add up score values calculated based on different windows sizes. The two different methods will be described next.

3.6.1. Basic Normalization

The first method denoted *basic normalization,* will be presented in the following (the method has been invented for this purpose only).

We have 3 different Markov models M_H, M_E and M_C, one for each group (H,E,C) and we want to normalize the output of the score functions associated with the models for different window sizes.

Assume that we want to normalize the score functions for window size l_1 to l_n. The following table (Table 2) can be constructed:

Window Size	l_1	l_2	**…**	l_n
	sc_{H1}	sc_{H2}	…	sc_{Hn}
	sc_{E1}	sc_{E2}	…	sc_{En}
	sc_{C1}	sc_{C2}	…	sc_{Cn}

Table 2: Normalizing models having windows sizes l_1 to l_n.

The values inside the matrix, sc_{H1} to $sc_{Cn,}$ is the returned score values calculated from the score functions associated with each Markov model for a particular window size. In this way sc_{H1} is the value of the score function that has a window size of l_1 and represents the model M_H.

If $l_1 < l_2 < \ldots < l_n$ then the l_n-column will most likely contain the smallest values (compared to the other values in the particular row) and the l_1-column will most likely contain the largest values (because of the number of terms in the score functions used to calculate the score values).

To be able to compare the values of the score functions directly all columns are scaled so the largest number in each column is set to 1.0. Column l_1 is scaled using the ratio, r:

$$r = \frac{1}{\max(sc_{H1}, sc_{E1}, sc_{C1})} \qquad (3.40$$

In this way the largest value will become 1.0 and the others will be relatively smaller. This procedure is applied to all columns.

Now an overall score value may be found for each Markov model representing one group. This new score value may be found by summing up all the scaled values in each row, resulting in three score values, sc'_H, sc'_E and sc'_C. The usual classification procedure can now be applied because the above system is reduced to contain only three score functions (values).

Using the same notation as in the above table (showing the normalization matrix) the formula for the combined score function for group X, sc'_X is given as:

$$sc'_X = \sum_{i=1}^{n} sc_{Xi} \cdot \frac{1}{\max\left(sc_{Hi}, sc_{Ei}, sc_{Ci}\right)} \qquad (3.41$$

where n is the number of windows sizes for which the normalization procedure should be applied (l_1 to l_n).

This normalization scheme is investigated further in Test Case 17.

Test Case 17	Method	Possible Conclusion(s)
Test the basic normalization procedure.	Run the test. Vary the window sizes and vary the number of models to be normalized.	Does the method increase or decrease the performance?

Test Case 17: Testing the basic normalization method.

The test shows that this method increases the performance of the models compared to using only one specific window size. The results are available in the section 7.Tests and Results.

3.6.2. Frequency Normalization

Another method for normalizing the outputs of models representing the same group but using different windows sizes is presented in this section. The normalization method described in this section both normalizes the values returned by the score functions and weights each value for each window size (using a certain scheme).

First the normalization part is described. As in the previous section the goal is to combine models representing the same group, but using different window sizes.

Again assume that we want to combine models using windows size l_1 to l_n where the following is true: $l_1 < l_2 < \ldots < l_n$. The score function using window size l_1 will most likely return values that are larger than the score function using window size l_2, since the window size is equal to the number of terms in the score function.

Let us recall the table depicting one possible matrix containing the score functions for several window sizes for each classification group (see Table 2). Now instead of scaling the columns as in the basic normalization method described in the previous section, each value is scaled using the nth root (where the window size determines the order of the root function).

Now the new combined score value for the group X, sc'_X may be found using the following formula:

$$sc'_X = \sum_{i=1}^{n} \sqrt[l_i]{sc_{Xi}} \qquad \textbf{(3.42}$$

where n is the number of window sizes for which the normalization procedure should be applied (l_1 to l_n). Now having this new method for normalizing the outputs of the models, the weighting of each term in the sum above is now explained.

The idea is to weight the normalized score value for each window size, according to some frequencies calculated based on the training data. Assume that the weights w_1 to w_n are defined for the helix group, then the above expression may be rewritten (for the helix group only) to incorporate the individual weights:

$$sc'_H = \sum_{i=1}^{n} w_i \cdot \sqrt[l_i]{sc_{Hi}} \qquad \textbf{(3.43}$$

As mentioned before the weight factors are calculated based on the training data. The description of how to obtain the weight factors will follow now.

Assume that we want to combine 3 Markov models having window sizes of 3, 4 and 5. For each group the training data is inspected to calculate the frequencies of the subsequences having length 3, 4 and 5 for the current group. The result is three frequencies for each classification group (H, E and C), which may be used as part of the weight factor.

The weight factor for the score function using window size *l*, is then defined as the frequency of subsequences having length *l* divided by the total number of subsequences (for one particular group).

The frequencies mentioned above may be calculated using several methods. This is illustrated with an example. Prior to the calculation the training data is investigated and collections of subsequences is generated, one collection for each classification group. For each classification group we loop through the collection of subsequences and calculate the frequencies needed (for the chosen window sizes).

Example:
This helical subsequence exists in the training data: PETILYFGKKR (length: 11).

Now we have (at least) three different options:

1. All frequency counters for subsequences length 1 to 11 is increased by one.
2. Only the frequency counter for subsequences of length 11 is increased by one.
3. The frequency counter for subsequences of length 1 is increased by 11 (the number of times a subsequence of length one may be generated from the above subsequence, keeping the same order of the residues), the counter for subsequences of length 2 is increased by 10 and so on.

This is repeated for all subsequences and for each classification group. The result is several frequencies for each classification group (one frequency for each group and window size combination).

Assume that we wish to normalize the models having windows size l_1 to l_n. The following table (Table 3) shows which values are calculated (sc_{Xi} denotes the score function (value) for the Markov model representing group X using window size l_i):

	Groups and frequencies					
Window size	**H**		**E**		**C**	
l_1	sc_{H1}	f_{H1}	sc_{E1}	f_{E1}	sc_{C1}	f_{C1}
l_2	sc_{H1}	f_{H2}	sc_{E2}	f_{E2}	sc_{C2}	f_{C2}
…	…	…	…	…	…	…
l_n	sc_{Hn}	f_{Hn}	sc_{En}	f_{En}	sc_{Cn}	f_{Cn}
		$F_H = \sum_{i=1}^{n} f_{Hi}$		$F_E = \sum_{i=1}^{n} f_{Ei}$		$F_C = \sum_{i=1}^{n} f_{Ci}$

Table 3: The values calculated for the frequency normalization method.

In Table 3 f_{H1} is the frequency counter for the helix group, using window size l_1 and so on.

The resulting score value for the combined (normalized) set of models may then be calculated as in:

$$sc'_X = \sum_{i=1}^{n} \frac{f_{Xi}}{F_X} \cdot \sqrt[l_i]{sc_{Xi}} \qquad \text{(3.44}$$

For each classification group (denoted X in the above formula), the normalized score value may be calculated and used as usual when classifying sequences. The difference is that the normalized score values are affected by several models using different window sizes and weighted by the frequency ratios determined from the training data.

The method just described and denoted as the frequency normalization is treated in Test Case 18.

Test Case 18	Method	Possible Conclusion(s)
Test the frequency normalization procedure.	Run the test. Vary the window sizes and vary the number of models to be normalized.	Does the method increase or decrease the performance?

Test Case 18: Testing the frequency normalization method.

This method appears to have a positive effect on the performance of the models tested.
The tests also show that the three options for calculating the frequencies obtain the same results.

The specific test results are again available in the section 7.Tests and Results.

3.6.3. Alternative Normalization Methods

The two normalization methods described in the previous sections are of course not the only ways to normalize the models (but they are the only ones implemented). Several other methods may be investigated.

One way is to scale the outputs of the different score function using another function than the nth root used in the frequency normalization scheme. Given the same conditions as for the frequency normalization the new normalized score function for some group X may be calculated as in the following (leaving out the individual weight factors):

$$sc'_X = \sum_{i=1}^{n} \frac{sc_{Xi}}{k^{l_i}} \quad \textbf{(3.45}$$

where the window sizes are l_i, $i \in [1;n]$. The constant k could be determined based on the mean value for each term in the score function. This is a direct substitute for the nth root used in the frequency normalization procedure, but other types of normalizations may be used.

Ignoring the size of the score values returned by score functions using different window sizes is possible. One possible way is to find the predicted classification group for some residue using several window sizes. Having these predictions (classification groups) for each window size, the most frequent group may be chosen.

Let us assume that we want to normalize the models using window sizes 3, 4, 5, 6 and 7. Each model is then used to get the predicted classification group. Suppose that for window size 3, 4 and 5 the helix (H) classification group is predicted, for window size 6 the coil group (C) is predicted and for window size 7 the beta sheet group (E) is predicted. Now the most frequent group is the helix group (3 out of 5) and this group is then assigned as the classification group for the current residue.

This scheme could be extended to incorporate some weight system for each window size (like the one used in the frequency normalization).

Due to the time period for this project we have not implemented the normalizations methods described in this section.

3.7. Combining Markov Models Having Different Orientations

Based on the analyses and methods described in the earlier sections we have found that the Markov models sometimes have trouble learning when one subsequence ends and the next begins. This section deals with the orientation of the Markov models and how this may be used to improve the general performance of the basic model.

The idea is to combine two Markov models, one for each direction of the sequence. When trying to predict the residue at position *i* in the sequence *S*, one Markov model may apply the score function on the residue x_i and the residues at position *i*+1 to *i*+*l*-1 where *l* is the window size. Another model may use the score function on the residues x_i to x_{i-l+1}.

Up until now whenever we have applied the score function to find the score value for the current window of residues, the residue at position *i* and the next *l*-1 residues have been considered (*l* is the window size):

Sequence, S_x	TPAFN$\boxed{\texttt{KPKVE}}$LHVHLDGAIKPETILYFGKKRGIALPADTVEELRNIIGMDKPLSLPGFLA
Predicted classification	CCCCC?

Now the normal window of residues and the reversed window of residues are taken into account.

Sequence, S_x	TPAFN$\boxed{\texttt{KPKVE}}$LHVHLDGAIKPETILYFGKKRGIALPADTVEELRNIIGMDKPLSLPGFLA
	T$\boxed{\texttt{PAFNK}}$PKVELHVHLDGAIKPETILYFGKKRGIALPADTVEELRNIIGMDKPLSLPGFLA
Predicted classification	----C?

The reversed window is treated in the reversed direction. Mathematically we may define the two score functions using the usual notation:

$$\begin{aligned} sc_{right} &= p(x_i)p(x_{i+1} \mid x_i)...p(x_{i+l-1} \mid x_{i+l-2}) \\ sc_{left} &= p(x_i)p(x_{i-1} \mid x_i)...p(x_{i-l+1} \mid x_{i-l+2}) \end{aligned} \quad \textbf{(3.46}$$

In practice this is accomplished having one model as usual and having another one for which the training sequences has been reversed (before anything else is done).

The question is now how to combine the two score values (for each classification group) obtained using the functions above. The simple way is to add the two results to get the new score value.

$$sc_{overall} = sc_{left}(\overline{x}_{left}) + sc_{right}(\overline{x}_{right}) \quad \textbf{(3.47}$$

Another variation is to use the minimum or maximum of the two returned results.

$$sc_{overall} = \min\left(sc_{left}(\overline{x}_{left}), sc_{right}(\overline{x}_{right})\right) \quad \textbf{(3.48}$$

Now having the combined score function, the classification process may be conducted as usual, now using the overall score function $sc_{overall.}$

One of the downsides of using this combined model, is that more residues can not be predicted, since a cutoff of *l*-1 (*l* being the window size) residues is necessary at each end of the sequence to be predicted (no rules for these residues have been defined, they are ignored during the classification process).

This combination of two Markov models, one for each direction, may be used for all earlier models presented. In other words, we may combine two of the basic Markov models (without any extensions) or we may combine two models which are several normalized models.

The argumentation for using this setup is that both the residues on the left side and on the right side of the residue to be predicted has an effect on the resulting classification.

The combined model using both orientations is tested using Test Case 19.

Test Case 19	Method	Possible Conclusion(s)
Test the combo model described above.	Run the test for various window sizes.	Does the method increase or decrease the performance?

Test Case 19: Testing the combo model (using normal and reversed orientation).

The tests showed some increase in performance using the method described above (see 7.Tests and Results).

3.8. Combining Different Extensions

Up until this point we have added standalone extensions meaning that one extension has been applied to the model at a time. A few methods work as simple combinations like the normalization methods and using $p'(a)$ combined with the reversed pair method. In this section we will discuss the idea of applying several extensions at the same time. This will result in a combined model consisting of known extensions.

There are many possibilities to combine the extensions presented in the previous sections. If all combinations should be applied it would require a lot of work and tests to analyze the models. Only a few of the presented models will be combined to see if it is possible to obtain higher prediction accuracies in this way. If this is possible there is a basis for further research.

We will intuitively expect that if a model with low performance is combined with another model of high performance the overall performance will be somewhere in between. We will also expect that it might be possible to gain performance when models with equal (or close to equal) performance are combined. These statements are only assumptions and the test result will show if it is actually possible to gain performance by combining different extended models.

Looking at the results obtained by running the tests for the different extensions makes it possible to handpick the better models. The following combinations are chosen:

- The new definition of the parameter denoted as $p'(a)$.
- The reversed pair method
- Frequency normalization
- The Momentum method

The following table shows the test case (Test Case 20):

Test Case 20	Method	Possible Conclusion(s)
Test the combination of $p'(a)$, the reversed pair method, frequency normalization and momentum.	Run the test using the best values for the each method (for extensions having parameters associated).	Is it possible to increase the performance by combining different extensions?

Test Case 20: Testing combinations of different extensions.

The test results show that the model consisting of combinations actually gains performance compared to the basic Markov model. The results also show that the combined model performs better than the extensions alone. This means that it is actually possible to combine models with a certain performance and thereby reach a higher level of performance. See the section 7.2.5.Combining Different Extensions for further details.

3.9. Summary

In the previous sections different modifications to the basic Markov model have been proposed. The modifications are based on observations and ideas, which became apparent during the preliminary test runs.

The modifications (extensions) proposed may be grouped as:

- Modifications of the individual terms in the score function (the parameters $p(a)$ and $p(a|b)$).
- Modifications of the overall score function.
- Modifications of the extracted subsequences (which also has an impact on the estimated parameters $p(a)$ and $p(a|b)$).
- Normalization methods to combine models predicting the same group but using different window sizes.
- Sequence orientation. Predicting both ways in a sequence.

Modifying how the parameters $p(a)$ and $p(a|b)$ are estimated by introducing $p'(a)$ and $p'(a|b)$ appears to increase the performance of the model. An example of a new score function only consisting of $p'(a)$ terms associated with a discount factor according to position has also been shown. This method also seems to increase the performance. The methods that take the order of the residues into account seem to be better than those that do not consider the order of the residues.

The modifications of the overall score function have also in general lead to better performances of the models. The methods using decision constants have increased the general performance. These methods are introduced to avoid that the model changes the predicted classification groups frequently yielding a predicted classification sequence with lots of small subsequences. This method seems to work. The momentum method has also increased the performance of the model. This method takes advantage of the peaks produced by the model. The method extends the peaks and thereby also avoids that the model changes the predicted classification group frequently.

The modification of the extracted subsequences (which serves as the training data for the model) has only increased the performance of the model using a window size of 3. For larger window sizes this method did not increase the performance. It is not possible to use Test Case 15 to conclude whether or not the Markov model has problems classifying whenever the window contains two or more subsequences. However altering the subsequences using the *swap residue* method did increase the performance of the models.

Normalizing different models has increased the performance. Also combining different model types has lead to an increase of performance.

The modifications that did not increase the overall performance of the model have given a better understanding of the Markov model – some of these modifications have also served as a basis for the ideas that have improved the model.

The specific test results obtained using the new models are presented in the section 7.Tests and Results. Each model is treated separately.

4. The GOR Classifier

4.1. Introduction

One of the more practically used schemes for predicting secondary protein structure is the GOR model. The model is somewhat different from the Markov models used in this project, but there are some similarities.

The model is named from the first letters of the inventors first names [1]. The model is based on calculating frequencies from the training data and using these frequencies (combined with statistics and information theory) to predict secondary structure of one or more test sequences.

The GOR model has been modified several times [1, 2, 3], resulting in several versions of the model. The idea behind the original model and the later versions will be described in more details in the following sections.

4.2. The GOR Idea in Theory

The GOR model is based on information theory (and Bayesian statistics), more specifically an information function described by Fano [2, 4]. Information functions may be used as a mathematical method of obtaining as much information as possible from some data. In the particular area, this function has been used:

$$I(S;R) = \log\left[p(S \mid R) / p(S)\right] \qquad \text{(4.1}$$

The function may be applied in several fields, but in this case the function is interpreted as follows: *S* is one of the classification groups to be predicted (H, E or C) and *R* is one of the residues. *p(S)* is then the probability that *S* is observed and *p(S|R)* is the conditional probability that *S* is observed having *R*. From the definition of conditional probabilities, *p(S|R) = p(S, R) / p(R)*, where *p(S,R)* is the joint probability of observing both *S* and *R*.

These basic rules are used to derive an approximation to the above information function
using the following definitions:

$$\begin{aligned} p(S,R) &= \frac{f_{S,R}}{N} \\ p(R) &= \frac{f_R}{N} \\ p(S) &= \frac{f_S}{N} \end{aligned} \qquad \text{(4.2}$$

where *N* equals the total number of residues in the current database, f_S is the frequency of the classification group *S*, f_R is the frequency of the residue *R* and $f_{S,R}$ is the frequency of having both classification group *S* and residue *R* in the training data.

Using the above stated rules we have:

$$I(S;R) = \log\left[(f_{S,R} / f_R)/(f_S / N)\right] \qquad \text{(4.3}$$

To include more information in the formula and thereby increase knowledge about the data the *information difference* is introduced:

$$I(\Delta S;R) = I(S;R) - I(nS;R) \tag{4.4}$$

where *nS* (*not S*) is the classification groups other than *S* (for the three different classification groups, this gives *nS* to be H and E, when *S* is C and so on). The above expression may be written as a function of the defined frequencies:

$$I(\Delta S;R) = \log(f_{S,R} / f_{nS,R}) + \log(f_{nS} / f_S) \tag{4.5}$$

This expression may be used as a simple predictor, where the information difference value is calculated for each possible classification group. The highest value (for some group) will then indicate the most probable classification group for that residue. This assumes that the frequencies have been calculated based on some training data, and that the actual values of the information difference above are calculated for each residue in one or more test sequences.

In practice the procedure of predicting residues is similar to the way Markov models predict – values representing the different groups are compared and the highest value dictates the classification. In this case the frequencies $f_{S,R}$, $f_{nS,R}$, f_{nS} and f_S are calculated for each possible value of *R* and *S*, where $R \in \Sigma$ and $S \in \Sigma_{classification}$. Having some test sequence A_{seq}, for which the classification sequence is to be predicted, each residue in A_{seq} must be investigated. For every residue the expression in (4.5 is calculated (for every group). The group associated with the highest value returned, may be assigned as the classification group for that residue. This procedure may be repeated until every residue has been classified.

The expression for the information difference in (4.5 may be extracted from the knowledge of the information function in (4.3 and the definition of the information difference in (4.4:

$$\begin{aligned}
I(\Delta S;R) &= I(S;R) - I(nS;R) \\
&= \log\left[(f_{S,R} / f_R)/(f_S / N)\right] - \log\left[(f_{nS,R} / f_R)/(f_{nS} / N)\right] \\
&= \log(f_{S,R} / f_R) - \log(f_S / N) - \log(f_{nS,R} / f_R) + \log(f_{nS} / N) \\
&= \log(f_{S,R}) - \log(f_R) - \log(f_S) + \log(N) - \log(f_{nS,R}) + \log(f_R) + \log(f_{nS}) - \log(N) \\
&= \log(f_{S,R}) - \log(f_S) - \log(f_{nS,R}) + \log(f_{nS}) \\
&= \log(f_{S,R} / f_{nS,R}) + \log(f_{nS} / f_S)
\end{aligned} \tag{4.6}$$

The information difference in (4.4 and (4.5 gives information about some state, using both the ordinary information *(S, R)* and the complementary information contained in *(nS, R)*.

One of the key elements in the GOR method is that a local sequence (or a window) of the current sequence is viewed and analyzed at some point in time. The information difference may be extended to such a local sequence, where the length of the local sequence is the current window size. The extended information difference may then be given as in:

$$I(\Delta S, R_1,..., R_n) = \log\left[p(S_j, R_1,..., R_n) / p(nS_j, R_1,..., R_n)\right] + \log\left[p(nS) / p(S)\right] \tag{4.7}$$

where the expression $p(S_j, R_1,...R_n)$ is the joint probability for the classification group *S* at position *j* in the sequence $R_1,...,R_n$.

All GOR versions are based on this expression, which is approximated differently for each version. This will be explained further in the next section, where each model version is described in more details.

4.3. The Different GOR Models

The GOR model has been modified several times from the basic form, which will be presented next. The different versions are based on different approximations to (4.7 and/or different databases.

The evolution of the GOR model has happened over several decades [2, 3, 16] and in that period of time the number of correctly classified protein sequences available for testing and training purposes has increased. This means that some of the GOR models that have been tested and documented to have some prediction accuracy (based on a specific database), in fact may have an even higher prediction accuracy (using a larger database), due to the increased number of (correctly) classified protein sequences. This is the case for GOR II, whose only difference to GOR I (according to [2]) is the database size.

Because of this fact the GOR models version I, III and IV will be presented, since these are definitely different. We have implemented and tested each of three different models on several databases and used the models as a benchmark tool for the Markov models developed in this project. Also the new version of the GOR model, GOR V [3], will be described, even though this model differs considerably from the other models due to the fact that it adds several new enhancements (based on biological background knowledge) to the GOR IV model.

4.3.1. GOR I

The GOR I model has taken into account the directional information contained in the 8 residues before and after the current residue (the one, whose classification group is to be predicted). This creates a window of 17 residues, where each residue (other than the center residue) is matched with the classification group for the center residue. In other words this approximation assumes that there is no correlation between the different residues in the current window.

This lead to a formal expression of GOR model version I:

GOR I

$$I(\Delta S_j; R_1,...,R_n) \approx I(\Delta S_j; R_J) + \sum_{m,m\neq 0} I(\Delta S_j, R_{j+m}) \quad (\ 4.8$$

where *j* is the position in the sequence of residues, where the classification group is to be predicted (i.e. the center residue in the window of 17 residues) and $m \in$ [-8,8], except 0.

The expression may be rewritten as a function of frequencies calculated from the training data:

$$\begin{aligned} I(\Delta S_j; R_1,...,R_n) &\approx I(\Delta S_j; R_J) + \sum_{m,m\neq 0} I(\Delta S_j, R_{j+m}) \\ &\approx \log(f_{S_j,R_j} / f_{nS_j,R_j}) + \log(f_{nS_j} / f_{S_j}) + \sum_{m,m\neq 0} \log(f_{S_j,R_{j+m}} / f_{nS_j,R_{j+m}}) + \log(f_{nS_j} / f_{S_j}) \end{aligned} \quad (\ 4.9$$

The expression above may be used directly when predicting secondary structure of proteins. The frequencies given in the above expression may be calculated (counted) easily from the training data and stored for further use. Predicting the actual test sequence may then be done using the calculated frequencies in the following manner:

For the current residue to be predicted, one calculates a value for each possible classification group. The group with the highest score is assigned as the predicted group for the current residue. This procedure continues until all possible residues are predicted.

As mentioned earlier the difference between GOR I and GOR II is the database size. In this text GOR I will be defined as above and GOR II will not be considered any further. Also GOR I and II were originally designed to predict structures having four different classification groups. In this text, GOR I will be used for predicting structures having 3 different classifications types (H, E and C).

4.3.2. GOR III

The next generation of the GOR model, GOR III introduced the so called pair information. Using the center residue in the window, one pair for every ordered combination of this residue with the other residues in the window is generated. Having a window size of 17 residues, the current residue (the one to be predicted) is matched with each of the other 16 residues in the window (8 on each side) generating 16 pairs. These pairs are taken into account in this model. This means that the correlation between the types of the residues in the current window and the type of the residue to be predicted is considered.

GOR III

$$I(\Delta S_j;R_1,...,R_n) \approx I(\Delta S_j;R_j) + \sum_{m,m\neq 0} I(\Delta S_j;R_{j+m} \mid R_j) \qquad \textbf{(4.10}$$

where *j* is the position in the sequence of residues, where the classification group is to be predicted and $m \in$ [-8,8], except 0.

The second term in (4.10 is a conditional expression, yielding that the information difference is now calculated based on both the current residue at position *j* and the residue at some position, *j + m*, in the current window. The classification of the residue at position *j + m* is still not considered.

The expression in (4.10 may again be rewritten as a function of frequencies calculated from the training data:

$$I(\Delta S_j;R_1,...,R_n) \approx I(\Delta S_j;R_j) + \sum_{m,m\neq 0} I(\Delta S_j;R_{j+m} \mid R_j) \qquad \textbf{(4.11}$$

where

$$I(\Delta S_j;R_j) = \log(f_{S_j,R_j} / f_{nS_j,R_j}) \qquad \textbf{(4.12}$$

and

$$\sum_{m,m\neq 0} I(\Delta S_j;R_{j+m} \mid R_j) = \sum_{m,m\neq 0} \log(f_{S_j,R_{j+m},R_j} / f_{nS_j,R_{j+m},R_j}) + \log(f_{nS_j,R_j} / f_{S_j,R_j}) \qquad \textbf{(4.13}$$

At the time this third version of the GOR model was proposed, the data available was somewhat limited to around 12.000 correctly classified residues in total. Because of this fact some of the frequency expressions in the above model (4.13 were impossible to calculate from the data available. Dummy frequencies were introduced to estimate the frequencies based on the current data. In this project GOR III has been used without using dummy frequencies, since it is believed that the database available now, is large enough for this (See the section 7.Test and Training Data).

4.3.3. GOR IV

The fourth version of the GOR model is in fact an expansion of GOR III. As mentioned, GOR III introduced the pair information where all possible pairs with the residue to be predicted were considered.

GOR IV expands this idea and looks at all possible pairs of residues in the current window (of 17 residues). There are (17 x 16) / 2 pairs to consider. In other words GOR IV takes into account that the residues from each pair are correlated. GOR IV is then expressed as in the following (using a slightly different notation that in [2]).

GOR IV

$$\log\frac{p(S_j,R_1,...,R_n)}{p(nS_j,R_1,...,R_n)} = \frac{2}{17}\sum_{m=-8,n>m}^{+8}\log\frac{p(S_j,R_{j+m},R_{j+n})}{p(nS_j,R_{j+m},R_{j+n})} - \frac{15}{17}\sum_{m=-8}^{+8}\log\frac{p(S_j,R_{j+m})}{p(nS_j,R_{j+m})} \quad (4.14$$

At first when GOR IV was proposed [2], there was no optimization of any kind on the above expression. The window size was constant (at 17) and nothing else was done to increase the performance of the algorithm. The expression is therefore shown for the window size of 17, although it is possible to vary the window size.

The expression above may be written with as a function of the window size.

$$\log\frac{p(S_j,R_1,...,R_n)}{p(nS_j,R_1,...,R_n)} = \frac{2}{2d+1}\sum_{n,m=-d,n>m}^{d}\log\frac{p(S_j,R_{j+m},R_{j+n})}{p(nS_j,R_{j+m},R_{j+n})} - \frac{2d-1}{2d+1}\sum_{m=-d}^{d}\log\frac{p(S_j,R_{j+m})}{p(nS_j,R_{j+m})} \quad (4.15$$

Here *d* is the window size variable, defined as the number of residues to be considered on each side of the center residue (the residue to be predicted). For the standard window size of 17, *d* equals 8 (the window size is equal to 2*d* + 1).

The above expression may be rewritten as a function of the frequencies calculated from the training data:

$$\log\frac{p(S_j,R_1,...,R_n)}{p(nS_j,R_1,...,R_n)} = \frac{2}{2d+1}\sum_{n,m=-d,n>m}^{d}\log\left(\frac{f_{S_j,R_{j+m},R_{j+n}}}{f_{nS_j,R_{j+m},R_{j+n}}}\right) - \frac{2d-1}{2d+1}\sum_{m=-d}^{d}\log\left(\frac{f_{S_j,R_{j+m}}}{f_{nS_j,R_{j+m}}}\right) \quad (4.16$$

4.3.4. GOR V

In 2002 another version of the GOR model was proposed [3]. This version was build directly on top of the above GOR IV model. The expression for GOR IV in (4.15 has been the basis for the new model, and on top of that several new adjustments have been implemented.

GOR V uses the reasonable results from the GOR IV model and adds several enhancements to the model. In the following the new steps in the GOR V model will be explained briefly. The following section refers to [3].

First of all another larger database has been used for the training and prediction procedure. The database assembled by Cuff and Barton [5, 6] containing 513 non-redundant sequences has been used. This database contains around 20.000 more residues than the database used with the original GOR IV model (This new database is the same database as we use in this project. More information is available in the section 7.Test and Training Data).

The second enhancement is decision constants. The decision constants are used in the final prediction part to ensure that the current classification group will only be predicted if the probability for that particular group is within some margin/threshold (compared to the probabilities for the other classification groups). This gives the possibility to adjust the prediction scheme for faults. If for example the model has a tendency to over-predict the coil (C) group, when in fact it is a helix (H) structure, the decision constants ensures that a coil will only be predicted if the probability for that group is larger than the probability for the helix group by some margin (i.e. the decision constant).

The third step was to include triplet statistics in the model. In the GOR IV model described above, only pair (and single) statistics are included (this will of course change the expression in (4.15).

The fourth step was to apply a resizable window for the model. When predicting sequences of smaller lengths, the window was decreased in size (according to some optimized parameters). This gives the advantage of being able to predict secondary structure on smaller proteins using a much smaller window (the GOR IV model had problems predicting sequences of small lengths, i.e. 20-30 residues).

The last step (which was the most successful and contributed to the largest increase in the prediction accuracy) was to use multiple sequence alignments for the prediction. This was implemented using the PSI-BLAST algorithm / program for generating the sequence alignments [20]. If no alignments where generated within five iterations, the original test sequence would be used for the prediction. In this case the prediction is very similar to the GOR IV model, except that the above 4 steps is completed. If the PSI-BLAST returned a set of alignments, these were used directly for the prediction of the secondary structure. Also some post-processing was conducted on the returned alignments. The idea was to exclude alignments that were too identical to the original sequence (the test sequence). The result was slightly better when excluding alignments that had an identity of 97% or greater to the test sequence.

The new GOR V model based on GOR IV and incorporating the above steps has shown a considerable increase in performance when predicting secondary structure. The results obtained with the various GOR models during the timeline of the GOR development period, will be described briefly in the next section.

4.4. Documented Results Using the Various GOR Models

One of the first GOR papers was published in the late seventies [1] and since then the work on GOR has continued resulting in several iterations of the model. The newest model GOR V has been described in a few papers published within the last few years [3].

The papers renew the success of the model as a reasonable secondary structure predictor compared to the prediction accuracies achieved by other known models. Although neural network models currently seems to have the highest prediction accuracies (closing in on the barrier at 80%), the GOR model in the newest version is capable of predicting secondary structure with a prediction accuracy of around 74%.

The people behind the GOR V model [3] claim that the GOR model in general has a considerable advantage compared to the neural network methods. This is based on the fact that the GOR model is very clear in the structure of which the model is based on (information theory combined with Bayesian statistics) and how the model works compared to the neural network models, which may seem like a black box.

Since this project deals with Markov models and the GOR models, this section will present an overview of the documented performances of the various GOR models.

GOR I was developed more than 25 years ago and started out predicting four different classification groups based on a rather small database. The second GOR model, GOR II, was updated with a new database in 1989 but the general algorithm was the same. In the subsequent iterations of the GOR model, the four-group setup, was decreased to a three-group setup (H, E and C). In 1996 Garnier et al. published a paper [2] presenting GOR III and GOR IV. The paper summed up the GOR models from version I to version IV.

In 2002 Kloczkowski et al. [3] published another paper describing the new GOR V model, which incorporated multiple sequence alignments (and several other extensions). In this paper the authors claim a 10% increase of the prediction accuracy, putting the GOR model in the group of the best current classifiers for secondary structure prediction.

The documented results for the different GOR models are shown in the following table (Table 4):

Model	**Q_3**	**Reference**
GOR I	55.0	[2]
GOR III	63.3	[2]
GOR IV	64.4	[2, 3]
GOR V	73.4	[3]

Table 4: Documented results for the different GOR models.

The results for GOR I has been obtained using a smaller database consisting of 67 proteins (later that performance was increased approximately 1% by using a larger database of 267 proteins).

The results for GOR III and IV in the above table have been obtained using the database described in [2] containing 267 proteins (around 63.500 residues) and using a full jackknife procedure (see the section 5.2.The Implemented Test and Training Procedure for a description of this method). Furthermore dummy frequencies were incorporated to obtain the above Q_3 for GOR III.

The result for the GOR V model has been obtained using the 513 non-redundant proteins (around 84.000 residues) assembled by Cuff and Barton (The DSSP Database) [5, 6].

We have implemented GOR I, III and IV mostly as benchmark tools for our own classifiers (the Markov models). The results obtained using the implemented GOR models and the Markov models will be presented in a later section (see 7.Tests and Results).

5. Evaluating Tests and Models

5.1. Introduction

Being able to compare different tests and evaluate tests in a standard way is valuable considering the number of tests possible and the number of other predictors developed.

This section will introduce several standard tools and/or methods which will be utilized for evaluation and comparison of the models implemented in this project (i.e. both the Markov models and GOR models). The standard notion of prediction accuracy will be defined and may be used to compare our results (with some caution) with the results from other studies.

Comparing and evaluating the results obtained using the implemented Markov models is a crucial part of determining which model is the better one. It is desirable to compare our results directly with results obtained in other studies, to get a clear picture of the real performance of the implemented models.

In this section we will describe what measurements and tools are used to make this evaluation and comparison. Also the implemented training procedure will be described in more detail.

5.2. The Implemented Test and Training Procedure

The actual training method for the training of the Markov models and the following testing, has been adopted from similar studies [2, 3, 8]. This approach is very convenient in that it makes it possible to compare our results directly with the results documented by other studies. In particular we are interested in comparing the results obtained using our Markov models with the results documented for the GOR models.

The training procedure used is a variant of the known jackknife method [2, 3] and may also be referred to under the name 'leave one out cross validation' (in short LOOCV).

The basic idea is described in the following:

- Assume that we have the complete database of which the model is to be evaluated. This database consists of a number of sequences, *n* (each sequence is associated with a correct classification sequence).
- For each training session, the Markov models are trained on every sequence in the database, leaving one out. The sequence which is left out (referred to as the test sequence) is then classified by the models trained on the rest of the sequences. This is repeated for every sequence in the database. That is, every sequence is left out once and the models are tested on that sequence (for every sequence a classification sequence will be predicted).

In this case the result is *n* similar tests (one for each sequence in the database).

As a measure of quality one prediction accuracy pr. sequence may be obtained using this procedure (the prediction accuracy pr. sequence is defined as the percentage of correctly predicted residues for that particular sequence).

The overall prediction accuracy (pr. residue) may be calculated after all test sequences have been predicted. This prediction accuracy is the percentage of all correctly classified residues for all residues predicted (these two different prediction accuracies will be treated in more later in this section).

It may be argued that the overall prediction accuracy may be too high using this jackknife procedure, since every sequence is tested once and does therefore have influence on the final result (every sequence is used both as a test sequence and a training sequence, although it is not in the same iteration). To counter this potential problem the database may be split into two parts, using one part for training and the other for predicting. This approach may yield results with lower prediction accuracies, but the results may be more independent of the database used.

5.3. Prediction Accuracy

It is important to be able to estimate the overall performance of the Markov models (or any other model) tested. The prediction accuracy (as mentioned several times before) may be defined for this usage. In general the prediction accuracy is defined as the percentage of correctly predicted residues for the current setup and is referred to as Q_3.

$$Q_3 = \frac{n_H + n_E + n_c}{N} 100 \qquad (5.1$$

where n_H, n_E and n_C is the number of correctly predicted residues for each type of classification group H, E and C. N is the total number of residues predicted (which is not necessarily the total number of residues in database, since residues may be ignored for several reasons already pointed out earlier).

There is however two ways to define this prediction accuracy when using the above jackknife method. We may calculate the prediction accuracy based on each iteration using the jackknife procedure (that is, for every tested sequence the prediction accuracy for that single sequence is calculated based on the numbers of correctly predicted residues).

This results in one prediction accuracy for each sequence in the database (while performing the jackknife procedure). This method has one downside though. In the case of a short test sequence even a few wrongly predicted residues, may result in a very low prediction accuracy (because of the definition of the prediction accuracy).

Therefore when evaluating the final result of the jackknife test and training procedure the prediction accuracy is measured as the percentage of correctly predicted residues of all predicted residues (for all tested sequences). This prediction accuracy may be referred to as the prediction accuracy pr. residue. The other method may be referred to as the prediction accuracy pr. sequence.

The pr. sequence prediction accuracy has the advantage that it is possible to calculate the mean standard deviation when using the jackknife procedure. One Q_3 value for each test sequence is obtained in the jackknife procedure, allowing the mean standard deviation to be calculated after the procedure has terminated. The pr. sequence Q_3 may also be used to identify sequences which are hard to predict, that is sequences resulting in a low pr. sequence Q_3.

Given n sequences, the corresponding prediction accuracies x_i for sequence i and the mean prediction accuracy x_{avr}, the mean standard deviation may be defined as:

$$S = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - x_{avr})^2} \qquad (5.2$$

The standard deviation holds information regarding the spread of the pr. sequence prediction accuracies. When observing or using the standard deviation, it is important to remember that the value is based on prediction accuracies calculated pr. sequence (which may result in a large standard deviation because of the fact that small sequences may result in low values for the pr. sequence Q_3). To use the formula of the standard deviation it is required that the data is normally distributed. This has been tested on several samples (from the test sessions) which actually show that the Q_3 value pr. sequence is normally distributed.

5.4. Accuracy Matrix

When evaluating the Markov models it is interesting to learn as much as possible from the current test and training session. The previous section described how the models are evaluated overall, that is how well the models predict sequences (overall), without differentiating between the different classification groups.

To learn more about the different classification groups, it is convenient to introduce an accuracy matrix. This method is adopted from the papers on the GOR model [2, 3].

The matrix is of the size 3×3 using the indices *i* and *j* for the relative position in the matrix. The *i* and *j* indices also denote the three classification groups (H, E and C). An element, A_{ij}, in the matrix, $[A_{ij}]$, is the number of residues predicted to be in state *j* that is actually in state *i* (according to the current database used).

Using this matrix definition, it is possible to derive valuable information of the overall prediction performance, but also more specifically on the individual subclasses, the different classification groups.

The sum of the elements in a column of the accuracy matrix, $[A_{ij}]$, is the total number of residues that are predicted to be in group *j*:

$$n_j = \sum_{i=1}^{3} A_{ij} \qquad \text{(5.3}$$

On the other hand the sum of the elements in a row of the accuracy matrix is the number of residues that are observed as classification group *i* in the database:

$$N_i = \sum_{j=1}^{3} A_{ij} \qquad \text{(5.4}$$

The correctly predicted residues are accounted for in the diagonal elements of the matrix, since these are the residues that are predicted to be in group *i* and are observed in group *j* (where $i = j$). The other elements of the matrix gives information of the incorrectly predicted residues, that is how many residues are predicted to be in group *i*, when they are in fact in group *j* ($i \neq j$).

It is now possible to calculate the overall prediction accuracy pr. residue, Q_3, which was defined in the previous section. The prediction accuracy is the percentage of all correctly predicted residues (defined by the sum of the diagonal elements) of the total observed residues and may be defined using the accuracy matrix, $[A_{ij}]$:

$$Q_3 = \frac{\sum_{i=1}^{3} A_{ii}}{N} 100 \qquad \text{(5.5}$$

N is defined as the sum of the all observed residues (which equals all predicted residues) in:

$$N = \sum_{i=1}^{3} N_i = \sum_{j=1}^{3} n_j \qquad (5.6$$

The matrix also makes it possible to measure the individual accuracies (noted as $Q_{3\ obs}$, since it is based on the observed residues in the group) within each classification group:

$$q_i = \frac{A_{ii}}{N_i} 100 \qquad (5.7$$

A similar measurement (noted as $Q_{3\ pred}$), may be calculated for each group:

$$q_j = \frac{A_{jj}}{n_j} 100 \qquad (5.8$$

This measurement is the probability that a residue predicted as some group X is in fact correctly classified as group X.

The accuracy matrix may then be depicted and expanded as in Table 5:

	Predicted			
Observed	H	E	C	Total
H	a_{11}	a_{12}	a_{13}	N_1
E	a_{21}	a_{22}	a_{23}	N_2
C	a_{31}	a_{32}	a_{33}	N_3
Total	n_1	n_2	n_3	N
Q_{pred}	$q_{H\ pred}$	$q_{E\ pred}$	$q_{C\ pred}$	
Q_{obs}	$q_{H\ obs}$	$q_{E\ obs}$	$q_{C\ obs}$	
Q_3				Q_3

Table 5: Accuracy matrix including the extra elements described above.

The darker area is the actual accuracy matrix. The other values in the table above are calculated based on these values. The matrix may also be referred to as a confusion matrix when used in the area of bioinformatics [21].

5.5. Matthews Correlation Coefficient

Considering that the best results in the area of secondary structure prediction are relatively high (with prediction accuracies just below 80% for the three classification group setup), it is interesting to look at a few very simple classifiers, that are easy to interpret (the classifiers have been mentioned before, under the term *trivial classification*).

The following examples may be uses as a few rules of thumb, when analyzing the resulting classifiers for the problem.

A classifier using a completely random assignment of the classification group for each residue in the test sequence, would result in a prediction accuracy of 33%. This type of classifier will only be used if one does not know anything about the populations of the groups. If the distribution of the groups is known an equal or higher performance is achieved by classifying all residues as the most frequent group (a prediction accuracy of 33% is achieved if the groups are equally populated).

In the case of the DSSP database (see the section 6.Test and Training Data) the most common classification group (using the structure class reduction scheme also described in section 6.Test and Training Data) is the coil (C) group with around 43% of the classified residues. This means that any prediction accuracy below 43% is worse than using the very simple classifier, which classifies every residue as a coil. It should be noted that a classifier that always predicts the same group is useless in the sense that it does not really supply any new information about the protein sequence.

To measure the quality of the prediction, Matthews's correlation coefficient (defined in [13], typically used for classification problems, for instance in [3, 14]) may be used for each group:

$$C_i = \frac{A_{ii}\left(\sum_{k\neq i}^{3}\sum_{j\neq i}^{3} A_{jk}\right) - \left(\sum_{j\neq i}^{3} A_{ij}\right)\left(\sum_{j\neq i}^{3} A_{ji}\right)}{\sqrt{\left(A_{ii} + \sum_{j\neq i}^{3} A_{ij}\right)\left(A_{ii} + \sum_{j\neq i}^{3} A_{ji}\right)\left(\sum_{k\neq i}^{3}\sum_{j\neq i}^{3} A_{jk} + \sum_{j\neq i}^{3} A_{ij}\right)\left(\sum_{k\neq i}^{3}\sum_{j\neq i}^{3} A_{jk} + \sum_{j\neq i}^{3} A_{ji}\right)}} \qquad (\ 5.9$$

The correlation coefficient makes it possible to compare the quality of the prediction with the random assignment (prediction accuracy of 33%), which yields $C_i = 0$. For the perfect prediction the coefficients are all 1, $C_i = 1$. If the correlation coefficient is negative, the prediction is worse than the random assignment.

Matthews correlation coefficient is undefined for the trivial classification for which every residue is classified as the most frequent group in the database (which is the coil group), since the expression above returns a division by zero.

5.6. Summary

The measurements described in the previous sections will be used when interpreting the results using the different Markov models (and also the GOR models) implemented in this project.

The main measure for the prediction precision accomplished using the implemented models will be the pr. residue prediction accuracy (Q_3), which is also the most common measure used in other studies [2, 3, 8, 16]. The other measures such as the mean standard deviation, the accuracy matrix and Matthews's correlation coefficient will be used whenever appropriate (when extra measures are needed).

The definitions above will be used without introduction in the section 7.Tests and Results.

6. Test and Training Data

6.1. Protein Databases and Classification Type Reduction

The two main databases used in this project are the database used by Garnier et al. [2] for the GOR IV classifier and the newest database assembled by Cuff and Barton [5, 6].

The first database used for the GOR IV classifier will be referred to as the *GOR database*. This database will not be used much, but since GOR IV is tested on this particular database we find it interesting to see if we can achieve the same performance using our own implementation of GOR IV (compared to the results documented for GOR IV [2]).

The second database assembled by Cuff and Barton [5, 6] is the primary database used for testing the classifiers developed in this project. This database will be referred to as the *DSSP database*[4]. The reasons for using this database instead of the one provided by the people behind GOR IV [2] are:

- The database contains more sequences than the GOR database.
- For the development of the GOR V model this database has been used.
- The database is newer than the GOR database (hopefully containing fewer errors).
- The database has been used in several studies.

However both databases could be used since each of them contain protein sequences and their associated correct classification sequences (the secondary structure). The contents of the databases are a bit different (as will be explained later) but both are usable.

A quick summary of the two databases is shown in Table 6:

Database	Sequences	Residues
GOR db	267	65230
DSSP db	513	84119

Table 6: A summary of the two databases used for this project.

6.1.1. The GOR Database

The GOR database is relatively straight forward since there are only four possible classification types: helix (H), beta sheet (or strand) (E), coil (C) and unknown (X). The unknown classification type is the result used when the current residue did not have any coordinates in the PDB file (The Protein Data Bank). The X type was omitted when using the GOR model as described in [2].

We have included the X type in the prediction procedure and tested the results when the X classification type is treated as each of the other classification groups (H, E, and C). The tests showed no large difference in the performances of the models and since it is not that frequent it is treated as a helix type.

According to [2] this database has been checked by V. Di Franscesco for homologous sequences and may therefore be used for secondary structure prediction. For secondary structure prediction it is custom to check the database for homologous sequences to avoid that the sequences are too identical within some margin (typically 30%). If some of the sequences are indeed identical (within the margin) there is a possibility that the results (prediction accuracies) may be biased towards the training data.

[4] The DSSP program is a program written to define the secondary structure of proteins in a standard way. The program may be used to extract the secondary structure of protein sequences given the three-dimensional atomic coordinates. The program does not predict secondary structure [7].

6.1.2. The DSSP Database

The DSSP database is a bit more informative than the GOR database. The DSSP assignments are available in the file, along with the assignments of other structural definition programs. Also multiple alignments for the sequence have been given (generated using the automated alignment procedure within *Jpred*, a prediction program developed by the Barton Group [12]).

Since the DSSP program returns the secondary structure in 8 different classification types, these types must be reduced/mapped to 3 different groups for the prediction (the three-state prediction of secondary structure is the most common). This however, may be done in several ways. Various papers have described how this is done for their classifier [2, 8]. The mapping of the DSSP database (and GOR database) is mentioned later in this section (see also Table 7 and Table 8 that contains these mappings).

The 8 structure types for the DSSP assignments are (as described in [7]):

- H = Alpha helix
- B = Residue in isolated beta-bridge
- E = Extended strand, participates in beta ladder
- G = 3-helix (3/10 helix)
- I = 5 helix (pi helix)
- T = Hydrogen bonded turn
- S = Bend
- _ = Gap

Furthermore there are some assignments of the type '?'. These are believed to be unknowns (and are treated as coils).

There are at least three different common ways to reduce the above classification types into groups. The methods are mentioned in [2, 8]. Usually the reduction results in three classification groups, one for helix, one for coil and the last classification group may (amongst others) be called (beta) strand or sheet.

The Critical Assessment of Structure Prediction, CASP [9], assumes that H, G and I are reduced to a helix group (H), B and E are reduced to a strand group (E) and the rest are reduced to at coil group (C). This is the method adopted in this project.
For the GOR V model described in [3], the following assignment was used. H was translated into a helix group (H), E was translated into strand (E) and the rest (B, G, I, S, T, and _) were translated into the coil group (C).

Other mappings and/or features have been used. The 'I'-classification type (π helix) may be included in the coil group (this does not change much, since it is very rare) and the B type may also be included in the coil (C) group.

Several people have made some pre-processing on the sequences before the actual training and prediction was done. This correction routine was based on the fact that helices rarely are shorter than 4-5 residues and therefore all helices of length 5 or shorter were treated as coils. Also the helical G types have been treated as alpha helices (H) if and only if they were neighbors to (alpha) helix types (H). Also corrections for bridges (classification type B) have often been used. The subsequence BC is then mapped to EE (strand/sheet type) and BCB is mapped to CCC (coil type) [11].

In one paper the authors [8] claim that the type reduction from the 8 different classification types returned by the DSSP program, may influence the overall prediction accuracy with up to 3%. This is of course a problem when comparing the results obtained using different classifiers, which are also based on different reductions schemes for the above classification types.

The DSSP database contains (as mentioned) 513 non redundant sequences. All the sequences has been compared pair-wise and are non redundant to a 5SD cut-off [12].

As mentioned before the DSSP database is the primary data source used in this project. The database is used as it is, without altering the data using any of the correction schemes mentioned previously. The distribution of the data is however examined. This is done to get information about the distribution of the classification groups, the length of the subsequences and so on. This kind of knowledge may be valuable when interpreting the results returned by the implemented Markov models.

6.2. Database Distribution

The two databases introduced above have been investigated to get an overview of the distribution of the classification groups and how each classification group is distributed according to the lengths of the subsequences.

The results are based on the two reduction schemes already presented in the previous section. The reduction scheme for using the GOR database is shown in Table 7.

Group	Group Name	Types
H	Helix group	HX
E	Beta sheet group	E
C	Coil group	C

Table 7: The reduction scheme used for the GOR database.

The reduction scheme showed in Table 7 shows how the original structure classes are reduced. This means that the original classes H,X are reduced to H, E is kept as E and C is kept as C.

The reduction scheme used for the DSSP database is shown in Table 8.

Group	Group Name	Types
H	Helix group	HGI
E	Beta bridge/ladder group	BE
C	Coil group	TS?_

Table 8: Reduction scheme used for the DSSP database.

Again this table shows how the structure classes in DSSP database are reduced. Structure classes H,G,I are reduced to H and B,E are reduced to E and T,S,?,_ are reduced to C.

Now investigating the basic distribution of the classification groups of both databases we find the results shown in Table 9.

Group	GOR db	Percentage	DSSP db	Percentage
H	21585	33.1%	29097	34.6%
E	13605	20.9%	19059	22.7%
C	28375	43.5%	35963	42.8%
Total	65230		84119	

Table 9: The distribution of the classification groups in the GOR and DSSP databases.

Table 9 shows that the distribution of the two database (according to the classification groups) are very similar. There are more helices and beta sheets in the DSSP database, but more coils in the GOR database. The table also shows that the most frequent group is the coil group with around 43% of the residues. The least frequent group is the beta sheet group with around 22% of the residues.

The trivial classification will therefore yield a prediction accuracy of 42.8% when using the DSSP database (and classifying every residue as coils).

Now since we are using the subsequences for the training of the Markov models it may be interesting to look at how these are distributed in each group according to the lengths of the subsequences. This is shown in Figure 12.

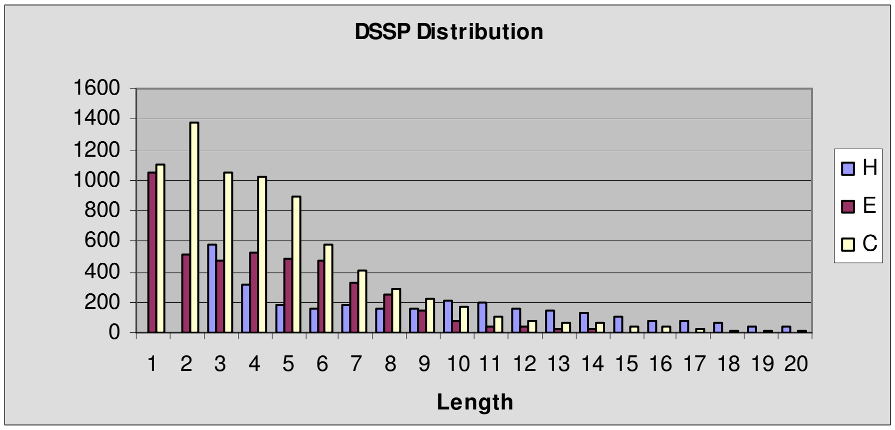


Figure 12: Distribution of the subsequences in the DSSP database according to the length.

The GOR database has a somewhat different distribution of the subsequences according to their length. This is shown in Figure 13.

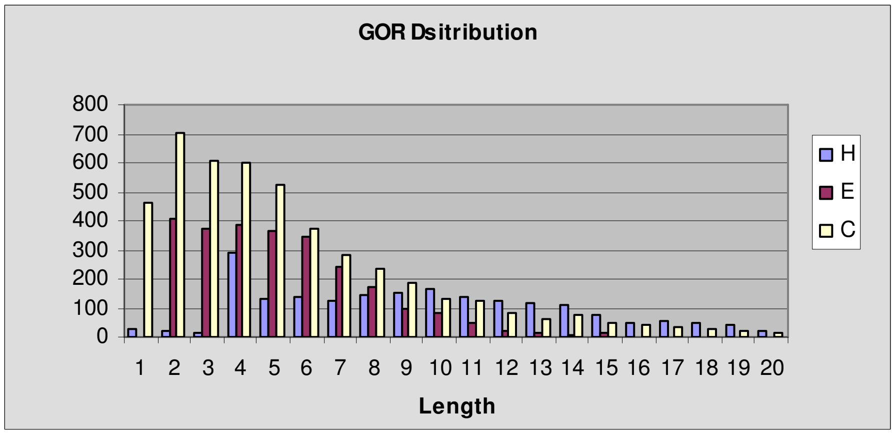


Figure 13: Distribution of the subsequences in the GOR database according to the length.

These figures are interesting since there are a few obvious differences. The helix groups for the two databases are almost alike although there are some helices of length 1 and 2 in the GOR database. On the other hand the DSSP database has quite a few helices of length 3. Since the GOR people [2, 3] uses correction algorithms for the data (treating helices of small lengths as coils and so on), this may be the explanation that there are almost no helices of length 1, 2 and 3. In fact further analysis of the GOR database shows that there are no helices of classification type H of length 1, 2 and 3 (but there is some of type X).

Also there are no beta sheets (E) of length 1 in the GOR database, while there are quite a few of them in the DSSP database. This may again be a result of the correction algorithm used by the GOR people [2, 3] where one idea is to replace the bridges (B type) with coils (C). The coil group is the most identical of the three classification groups.

Based on the distribution of the subsequences as shown above the mean subsequence length for each classification group may be calculated. These results are shown in Table 10.

Group	DSSP	GOR
H	9.3	10.9
E	4.3	5.3
C	4.7	6.0

Table 10: Average subsequence length for each group.

The table shows that the GOR database tends to have subsequences that are a bit longer (about one residue) than the subsequences in DSSP database. We see that the (H) subsequences with a mean length around ten residues are twice as long as the mean lengths for the (E) and (C) subsequences.

The mean subsequence length has been found by analyzing the DSSP and GOR databases. The number of subsequences for each group and for each length has been found. This is done by counting the number of helical subsequences of length = 1,2,3,… until all subsequences has been accounted for. The number of helical subsequences is stored for each length. This procedure is used for all groups. The information retrieved in this way is used to calculate the mean subsequence lengths.

These average lengths of the subsequences in the databases may be used as rules of thumb when analyzing the Markov models.

7. Tests and Results

This section will describe the tests completed for each developed Markov model. The tests are presented and discussed separately in the following subsections. Each test is intended to reveal possible interesting abilities of the particular Markov model. Also the tests are used to determine whether the particular extension is actually better than the basic Markov model or the model it has extended.

The section 5.Evaluating Tests and Models is the basis for the more formal treatment of the results obtained. As mentioned in that section the prediction accuracy, Q_3, will be the main measure to determine whether one model scheme is better than another (the pr. residue prediction accuracy is used unless otherwise mentioned). When analyzing the models more closely the other measures described may be applied.

The results are presented in the same order the models are originally presented in the section 3.Using Markov Models for Classification. Each test case described in that section will be presented again in this section and the obtained test results will be shown. It is assumed that the section 3.Using Markov Models for Classification has been read and the general ideas behind each extension are known.

The GOR model is treated in the later subsections of this section. The different versions of the GOR models which we have implemented are tested and treated separately. A comparison of the GOR models to the developed Markov models will be presented in a later subsection.

Finally the results will be summarized and discussed further giving an overview of the best Markov models developed and how these perform compared with the GOR models.

7.1. Time Consumption Running Tests

This section is meant as a very brief introduction to how time consuming the test sessions are.

For each iteration in the jackknife procedure the following tasks must be completed:

- Determine which sequences are used for training and add the corresponding sequence files to a collection.
- Read the training sequences and the correct classification sequences in the database and extract all subsequences for each classification group.
- Estimate the parameters $p(a)$ and $p(a|b)$ for each Markov model, M_H, M_E and M_C using the extracted subsequences.
- Read and classify the test sequence (the one left out of the current training procedure).

This procedure is repeated until every sequence has been classified.

Running a standard test using the basic Markov model and the DSSP database, the total time spent completing the full jackknife procedure is divided into each task as shown in Figure 14.

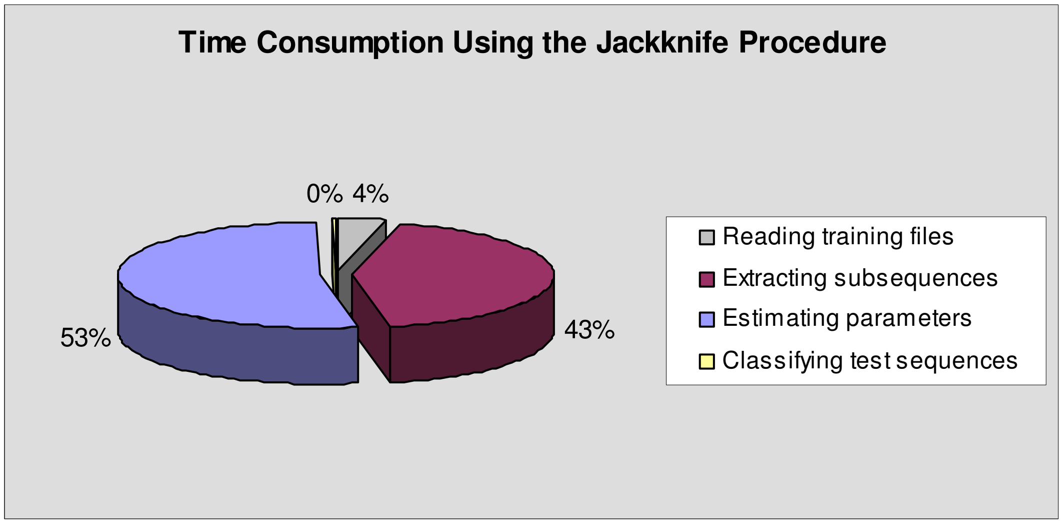


Figure 14: The time consumption completing the jackknife test and training procedure.

The figure shows that the process of estimating the parameters for each of the three Markov models, M_H, M_E and M_C is the most time consuming part of the jackknife procedure. It also shows that the part classifying the actual test sequence does not have a strong impact on the time consumption (increasing the window size does not increase this percentage much).

The percentages are calculated based on the overall time consumption for each task in the jackknife procedure. The results presented are shown to indicate the general time consumption in the whole classification process.

A few practical examples[5]:

- Using the basic Markov model (the pseudo count constant, c = 1 and the window size, l = 3) the total time used for the jackknife procedure classifying the DSSP database was 38 minutes.
- Using the basic Markov model (the pseudo count constant, c = 1 and the window size, l = 10) the total time used for the jackknife procedure classifying the DSSP database was 40 minutes.

Of course using some of the extensions of the basic Markov model do indeed increase the time consumption of the full jackknife process, since these extensions incorporate more programming logic. Some of the extensions also increase the number of subsequences (by adding new subsequences) and therefore the parameter estimation will be more time consuming.

No deeper analysis of the theoretical time consumption has been done.

7.2. Test Results Using Markov Models

This section and the following subsections present the tests and results obtained using the different Markov models and extensions presented in the section 3.Using Markov Models for Classification.

[5] These tests were conducted on a 2.0 GHz AMD machine running Windows XP.

7.2.1. The Basic Markov Model

The basic Markov model from which all other models are derived have been tested and analyzed in details. This section outlines and discusses the test results obtained using this model.

The section 3.4.The Basic Markov Model outlined two test cases that should be completed using this first model. The first test case is based on the very simple model which does not incorporate any scheme for pseudo counts. In this model the window size, *l*, may be varied as the only parameter (the parameters $p(a)$ and $p(a|b)$ defining each of the three Markov models M_H, M_E and M_C are determined during the training of the models, while the window size parameter is used when classifying test sequences).

The first test case, Test Case 1, is intended to reveal how the model performs for different values of the window size.

Test Case 1	Method	Possible Conclusion(s)
Explore the *window size* parameter.	Vary the *window size* and measure the overall performance of the model.	How the *window size* affects the overall performance of the basic Markov model.

Test Case 1: Exploring the window size parameter in the basic Markov model.

Test Case 2 is intended to analyze the effect of the pseudo count constant when using the basic model for classification. The pseudo count constant may be varied and for each value the models are tested.

Test Case 2	Method	Possible Conclusion(s)
Explore the *pseudo constant, c.*	Vary *c* and measure the overall performance of the model.	How *c* affects the overall performance of the basic Markov model.

Test Case 2: Investigating the effect of the overall performance of the models when introducing the pseudo count constant.

Remember that setting the pseudo count constant, $c = 0$, is the same as using the original model having no logic for handling pseudo counts. Due to this fact it is possible to combine Test Case 1 and Test Case 2 into one test.

Having the chosen values for the window size parameter, *l*:

$$\bar{l} = \{1,2,3,4,5,6,7,8,9,10\}$$

and having the chosen values for the pseudo count constant, *c*:

$$\bar{c} = \{0, 0.5, 1, 5, 10, 100, 1000\}$$

one test session may be completed for each possible combination of the windows size, *l*, and the pseudo count constant, *c*.

The values chosen for the two parameters in question are chosen based on observations from the training database (see the section 6.Test and Training Data) and analyses of the expressions for the parameters $p(a)$ and $p(a|b)$.

It is anticipated that having a window size of 1 is too small (remembering that the individual score functions for each Markov model would then consist of only one term, the $p(a)$ parameter). On the other hand a window size of 10 would in many cases mean that the current window overlaps several classification groups (several subsequences) since many subsequences are of smaller lengths (the mean lengths of the subsequences are shown in the section 6.2.Database Distribution). This may confuse the model and result in unwanted results (having low prediction accuracies).

The values for the pseudo count constant are chosen in a large interval ranging from 0 to 1000. Having a pseudo count constant equal to 0 reduces the model to the simple initial model, which is interesting to include in the test. The idea is to see if the pseudo count method actually changes the performance of the models. The other values for the pseudo count constant are chosen in a range from 0.5 to 1000 to make sure that a large interval is being tested. We assume that having a very large pseudo count constant may make the parameters $p(a)$ and $p(a|b)$ somewhat unreliable (the expressions for the parameters $p(a)$ and $p(a|b)$ would converge to $1/|\sum|$ when the value of the pseudo count constant is very large).

Running the above combined test, results in a graph showing the prediction accuracies (Q_3) for each combination of the window size and the pseudo count constant. The graph is shown in Figure 15.

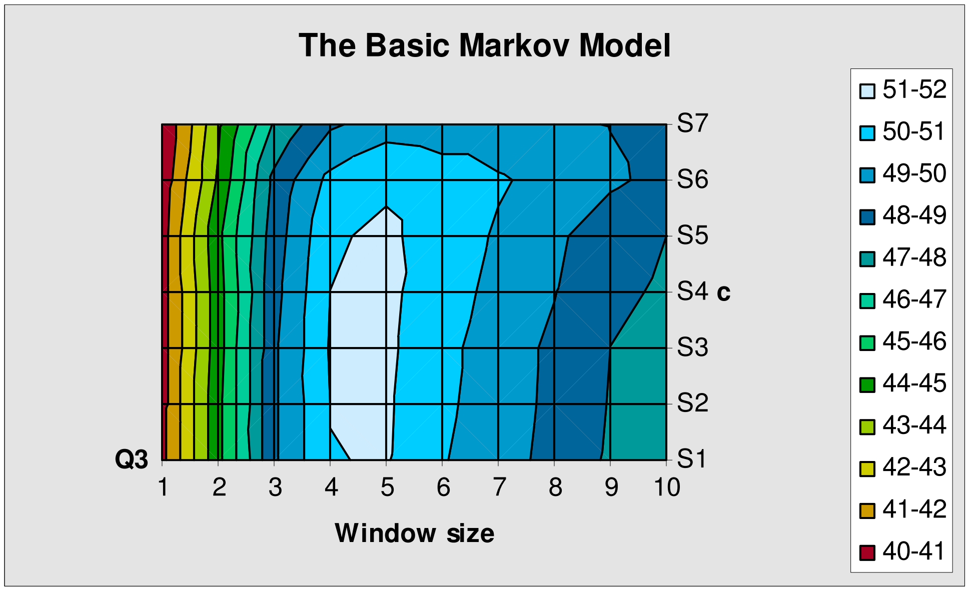


Figure 15: Results for the basic Markov model, varying the window size and the pseudo count constant.

The graph is based on the values found for each combination of the window size parameter and the pseudo count constant. To see the tendency in the obtained values (prediction accuracies) the graph is interpolated between the actual test results.

The results obtained combining Test Case 1 and Test Case 2 shows several interesting features about the models tested. First of all the highest prediction accuracy obtained, $Q_3 = 51.2\%$ is obtained using a window size of 5 and a pseudo count constant of 5 ($l = 5$, $c = 5$).

The trivial classification mentioned earlier yields a Q_3 of 42.8% which is lower than the best Q_3 found above. This indicates that the models trained are definitely better than the trivial classification and the Markov models are in fact able to infer some of the secondary structure from the one-dimensional protein sequences.

On the other hand the worst Q_3 value is 39.3% for the model scheme using a window size of 1 and a pseudo count constant of 1000. This Q_3 value is below the benchmark value for the trivial

classification and shows that the models trained are having prediction accuracies ranging over a relatively large interval (both beyond and above the Q_3 for the trivial classification).
Figure 15 also shows an optimal area of values for the window size and the pseudo count parameters having prediction accuracies above 51%. It seems that the window size is optimal around 4 and 5 and the pseudo count constant is optimal at values {0.5, 1, 5, 10}. This leads us to believe that we have found a set of values for the window size and the pseudo count constant which may be used in following tests exploring other features and models.

Another interesting thing which is evident from Figure 15 is that having a pseudo count constant, $c = 0$, does not seem like a major problem. The window size parameter has a much larger effect on the prediction accuracies for the tested models. Lets assume that the training database is large enough, so that every parameter $p(a)$ and $p(a|b)$ for each of the models M_H, M_E and M_C may be estimated to some value different from 0. If this is the case, the use for pseudo counts is not needed, since pseudo counts are implemented to prevent parameters being estimated as zero-probabilities.

Summing up the knowledge gained from Test Case 1 and Test Case 2 we find:

- The basic Markov model scheme is better than the trivial classification (for specific values of the window size and the pseudo count constant).
- Optimal values for the window size {4, 5} and for the pseudo count constant {0.5, 1.0, 5.0} have been found (using this particular model).
- The window size parameter has much more influence on the performance (Q_3) of the models than the pseudo count constant (which may be explained by the large database used for training).

7.2.1.1. Accuracy of the Individual Classification Groups

Since it seems that the pseudo count constant does not have a drastic effect on the performance of the basic Markov model, we now focus at the test results obtained using a pseudo count constant of 1. It is interesting to see how the individual prediction accuracies for each classification group behave for different window sizes.

Let us recall the accuracy matrix introduced in the section 5.Evaluating Tests and Models. Using the matrix, two types of individual Q_3 values ($Q_{3\ obs}$ and $Q_{3\ pred}$) were defined for each classification group. The observed Q_3 values $Q_{3\ obs}$ is the probability of predicting some residue as group X when the residue is in fact correctly classified as group X. The other individual Q_3 value $Q_{3\ pred}$ is the probability that some residue is actually correctly classified as group X when it has in fact been predicted as group X.

In the above test combining the window size with the values for the pseudo count constant, the according accuracy matrices have been found for each combination. Calculating the $Q_{3\ obs}$ and $Q_{3\ pred}$ for each classification group is then possible. The results are interesting in that they supply information regarding the individual groups and how they are affected when varying the window size.

The values for the individual $Q_{3\ obs}$ and $Q_{3\ pred}$ are shown in Figure 16 and Figure 17 respectively. The overall Q_3 values are also shown.

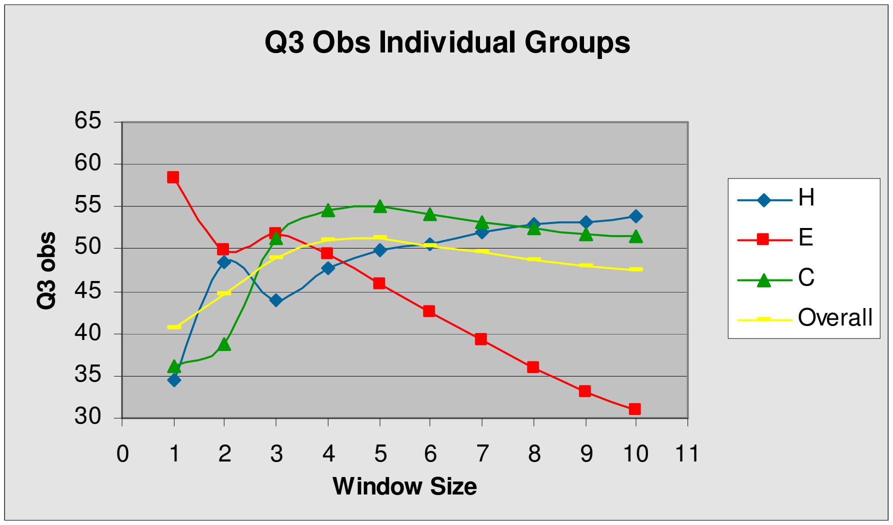


Figure 16: Graph showing the $Q_{3\ obs}$ values using the basic Markov model.

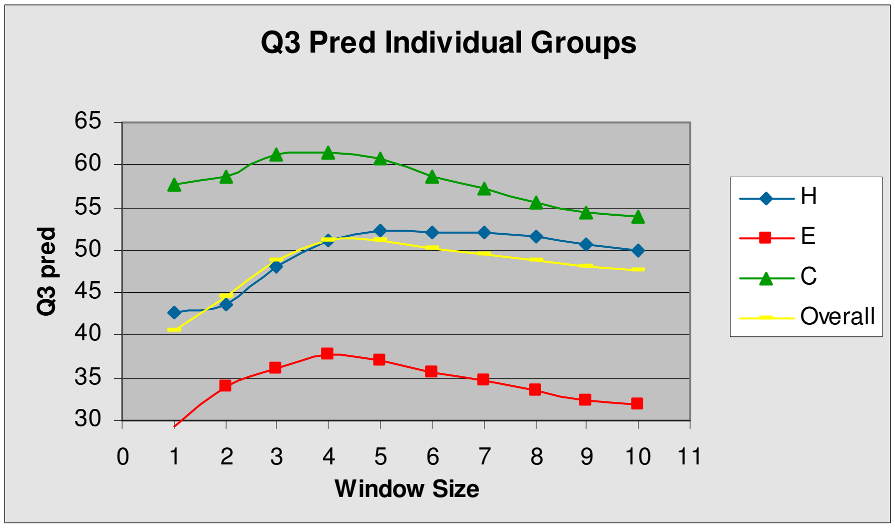


Figure 17: Graph showing the $Q_{3\ pred}$ values using the basic Markov model.

The two figures above shows several interesting features.

Looking at Figure 16 first, we see that the $Q_{3\ obs}$ for the beta sheet group (E) is lowered when increasing the window size. This may be partly explained by the fact that the beta sheets are the subsequences with the smallest mean length (of 4.3 residues closest to 4) and increasing the window size may mean that several subsequences overlap in the window. The consequence of this is possible misclassifications (incorrectly assigned classifications. This is explained in the section 3.Using Markov Models for Classification). The misclassifications occurs for a fixed number (depends on window size) of residues in a subsequence and since the beta sheet subsequences have the smallest mean length, a larger percentage of the beta sheet residues are incorrectly classified.

The $Q_{3\ obs}$ values for the helix group (H) and the coil group (C) are increased when increasing the window size. The helix group is the group having the longest mean subsequence length of 9.3 residues, which may explain why more helices are classified correctly when increasing the window size. The coil group has a mean subsequences length of 4.7 (closest to 5). At this window size we se that the performance starts to drop. The same argumentation may be used for the beta sheet group but the effect is not as clear as for the coil group.

Remember that having a $Q_{3\ obs}$ of 100% for some group does not necessarily mean that the classifier is perfect (this may be accomplished using the trivial classification which always predicts one group). The three $Q_{3\ obs}$ values must all be taken into account when assessing the overall performance.

The ideal situation would be to have $Q_{3\ obs} = Q_{3\ pred} = 100\%$ for every group (equal to the perfect prediction). This leads us to Figure 17 which shows the $Q_{3\ pred}$ values for each group. Here we see that the values for all groups increases using window size 4, which means that every individual group is predicted better, since more residues predicted as group X are in fact correctly classified as group X. An interesting feature here is that the curves formed by the individual values of $Q_{3\ pred}$ are very similar to the curve formed by the overall values of Q_3 (shown in yellow), meaning that if every individual $Q_{3\ pred}$ value is increased (or decreased) the overall prediction accuracy follows a similar pattern.

In general the graphs depicted in Figure 16 and Figure 17 showing $Q_{3\ obs}$ and $Q_{3\ pred}$ can be used to analyze a model. The graphs provide information about how the model performs internally (for each group) giving a more detailed view of the model.

Summing up the knowledge gained from this section:

- The accuracy matrix and the terms $Q_{3\ obs}$ and $Q_{3\ pred}$ may provide additional information when analyzing the models.

7.2.1.2. Reversing the Training Sequences

Originally when the basic Markov model was implemented we played around with the model to see how it actually worked. One of the things we tried out was to reverse the training sequences (and their corresponding correct classification sequences) before the parameters defining each of the models M_H, M_E and M_C were estimated. The effect is that the estimated parameters are based on the reversed subsequences.

Conducting a test and training session where the training sequences were reversed yielded some interesting results. It seemed that the overall prediction accuracy was larger when using the reversed sequences compared to the normal setup. The results obtained using a pseudo count constant of 1 and window sizes of 3, 4, 5 and 6 are shown in Table 11.

	Direction	
Window Size	**Normal**	**Reversed**
3	48.8	50.6
4	51.0	52.2
5	51.2	52.7
6	50.2	51.7

Table 11: Results obtained using reversed training sequences.

It is interesting to see that the overall Q_3 actually is increased by 0.8–1.5% when reversing the training sequences. These results were surprising at the time, since we anticipated the results to be (almost) the same as when using the normal direction. The explanation for this increase in Q_3 may be found in the way the Markov models are defined.

Reversing the sequences has an impact on the $p(a)$ and $p(a|b)$ parameters defining the models. These parameters are estimated on the training data (the exact method is described in the section 3.3.1.Definition of the Markov Models). When reversing the training sequences, the estimated parameters will probably end up having different values than when estimating the parameters as usual. At the time of this discovery we had difficulties explaining the phenomenon of the increased Q_3 values when reversing the training sequences.

Later on the new definition of the *p*(*a*) parameter denoted *p*'(*a*) was introduced and analyzed (this is addressed in section 3.5.1.1.Analysis of the Term *p*(*a*)). Using this new definition *p*'(*a*) we ran some additional tests to see if the reversed sequences still resulted in higher prediction accuracies. This was not the case though. The results for the normal sequences and the reversed sequences were very similar leading us to think that the original *p*(*a*) term was the main cause for this difference (the definition of the *p*'(*a*) parameters ensures that the values of the parameters values remain the same for the normal and reversed sequences, since *p*'(*a*) is the probability of observing *a* regardless of the direction of the sequences).

Analyzing the *p*(*a*) parameter and the *p*'(*a*) parameter using the extracted subsequences from the training database proved that there are in fact obvious differences between the estimated *p*(*a*) parameters using the normal sequences and the reversed ones, whereas the *p*'(*a*) parameters are almost the same. This leads to one possible explanation of the performance increase in the original model using reversed sequences. The large effect of the *p*(*a*) parameter on the resulting score values is causing these Q_3 differences (This is shown in more details in Appendix C – Additional Analyses and Tests, Reversing Training Sequences in the Basic Markov Model).

However we are well aware that reversing the sequences does not only affect the *p*(*a*) parameter but also the *p*(*a*|*b*) parameter. This parameter may also be investigated, but it is a bit more tedious considering the number of combinations of *a* and *b*.

As always it may be that the reason for the larger Q_3 values observed reversing the sequences may be explained using biological knowledge (the sequence may has a biological direction).

Summing up the knowledge gained from this section:

- Reversing the training sequences increases the performance (using the basic Markov model definition).

7.2.2. Extensions to the Basic Markov Model

For each extension (and for each test case) presented in the section 3.Using Markov Models for Classification the specific test results will be presented and analyzed. The results of every model scheme are compared to the results obtained in previous section where the basic Markov model was tested.

7.2.2.1. Extensions Based On the Individual Terms in the Score Function

The test cases concerning the individual terms in the score functions will now be presented again and the associated results will be commented.

7.2.2.1.1. Term *p*(*a*)

Test Case 3 explores the new definition of the first term in the score function, *p'*(*a*). First the parameter *p*(*a*) is redefined denoted as *p*'(*a*). The difference is that the parameter *p*(*a*) is the probability of observing the residue *a* as the first residue in a subsequence (for some group), whereas the parameter *p*'(*a*) is the probability of observing the residue *a* in any position.

Test Case 3	Method	Possible Conclusion(s)
Is *p*'(*a*) better than *p*(*a*) ?	Set the window size and pseudo count variables as fixed values. Produce two tests: a) Use the term *p*(*a*) and find the performance of the models. b) Use the term *p*'(*a*) and find the performance of the models.	*p*'(*a*) is better or worse than *p*(*a*). The test may be inconclusive showing no apparent difference between the two definitions.

Test Case 3: Asserting whether *p*'(*a*) is better or worse than *p*(*a*).

The results (prediction accuracies) obtained from running the above test described in Test Case 3 are shown in Table 12.

Window Size	**Using *p*(*a*)**	**Using *p*'(*a*)**
3	48.8	50.4
4	51.0	52.4
5	51.2	52.6
6	50.2	51.7

Table 12: Results obtained substituting *p*(*a*) with *p*'(*a*).

The results have been obtained using window sizes {3, 4, 5, 6} which include the optimal values found during the test of the basic Markov model (described in the previous section). The pseudo count constant has been set to one, *c* = 1 (this value for the pseudo count constant will be used unless otherwise mentioned).

The results show that the new definition of the first term in the score function, *p*'(*a*), does indeed increase the performance of the models (the prediction accuracy, Q_3, is higher). Using the *p*(*a*) term is the same as using the basic Markov model from the last section, for which the highest prediction accuracy obtained is 51.2% (using a window size, *l* =5). Using the same values for the window size and the pseudo count constant, result in an increase of 1.4% when substituting *p*(*a*) with *p*'(*a*).

These test results seem to backup the assumption that the Markov models may get confused when classifying sequences, for which the first residue in the window is not the first residue in a subsequence.

Having these results in mind, let us see how the models perform when excluding the first term, *p*(*a*) or *p*'(*a*), from the score function. This is treated in Test Case 4.

Test Case 4	**Method**	**Possible Conclusion(s)**
Is the terms *p*(*a*) or *p*'(*a*) necessary at all?	Modify the score functions for the individual Markov models (M_H, M_E and M_C) leaving out the *p*(*a*) / *p*'(*a*) term and measure the performance. The performance is compared to the results of the previous tests.	The terms *p*(*a*) or *p*'(*a*) is necessary in order to achieve good performances.

Test Case 4: Testing whether the *p(a)* or *p'(a)* term is necessary in the individual score functions for the three Markov models, *MH, ME* and *MC.*

The Q_3 values obtained running the above tests is depicted in Table 13.

Window Size	**Only *p*(*a*|*b*)**	**Normal, *p*(*a*)**	**Normal, *p*'(*a*)**
3	50.5	48.8	50.4
4	52.0	51.0	52.4
5	51.7	51.2	52.6
6	50.3	50.2	51.7

Table 13: Test results leaving out *p*(*a*) / *p*'(*a*) compared to when using one of the terms.

The first column in Table 13 (after the different values of the window size) is the test results using only the *p*(*a*|*b*) terms (leaving out *p*(*a*) or *p*'(*a*)). The two next columns show the results from the last tests, when using either *p*(*a*) or *p*'(*a*). Again the pseudo count constant, *c* equals 1.

The results indicate that when using the definition *p*(*a*) the term may be left out, yielding better results (although some of the results are very close). However using the *p*'(*a*) term instead of the *p*(*a*) term yields different results. The results are now (in most cases) better using both the terms *p*'(*a*) and *p*(*a*|*b*).

Summing up the knowledge gained about the first term in the score function:

- Substituting *p*(*a*) with *p*'(*a*) yields better results in general.
- Leaving out the terms *p*(*a*) or *p*'(*a*) leads to two conclusions:
  - The basic model (using *p*(*a*|*b*) and *p*(*a*)) is worse than only using *p*(*a*|*b*) which means that the *p*(*a*) term decreases performance.
  - Using only the *p*(*a*|*b*) term is worse than combining it with *p'*(*a*) which means that *p*'(*a*) is still needed in order to keep the performance up.

7.2.2.1.2. Term *p*(*a*|*b*)

Now the other parameter *p*(*a*|*b*) in the score function is investigated. First a basic test revealing whether or not it is necessary to include the term at all is conducted. Then possible modifications to the parameter is suggested and tested.

Test Case 5 describes the possible tests that may be conducted leaving out the *p*(*a*|*b*) parameter from the score function.

Test Case 5	Method	Possible Conclusion(s)
Does it make sense to include the *p*(*a*|*b*) terms in the score functions for each Markov model?	Set window size = 1and use *p*(*a*) as the only parameter defining the Markov models. The same test may be conducted using the *p*'(*a*) definition of the parameter.	Is it necessary to include the *p*(*a*|*b*) terms to get a reasonable classification?

Test Case 5: Investigating the performance of Markov models based on score functions where the *p(a|b)* terms are left out.

Running the two tests where the window size is 1 (see Test Case 5) yield the following results (shown in Table 14).

Window Size	Only *p*(*a*)	Only *p*'(*a*)
1	40.6	47.2

Table 14: The Q_3 values for running the tests without using the parameter *p*(*a*|*b*).

Both tests leaving out the parameter *p*(*a*|*b*) shows a drastic decrease in performance (Q_3). This is expected since the score function is reduced to one term only and the classifications are assigned using the probability of one residue only (the one in the window). There is no need to investigate this any further and we may continue with the assumption that the order of the residues (and thereby the parameter *p*(*a*|*b*)) is in fact connected with the correct classification sequence for some sequence, *S*.

Now knowing this, different variations of the parameter *p*(*a*|*b*) may be tested.

Summing up the conclusion from Test Case 5:

- The term *p*(*a*|*b*) is necessary in order to achieve a reasonable prediction accuracy.

7.2.2.1.3. The Reversed Pair Method

This method deals with a new definition of the parameter *p*(*a*|*b*). The parameter is redefined denoted as *p*'(*a*|*b*). The parameter is now estimated on the subsequences extracted (as usual) but now including the reversed subsequences for the group in question (the exact method is described in the section 3.5.1.3.The Reversed Pair Method).

Test Case 6 is used to explore the method.

Test Case 6	Method	Possible Conclusion(s)
Does the *reversed pair* method increase the performance (compared to the original model) ?	Substitute the $p(a\|b)$ parameter with the new definition $p'(a\|b)$ and conduct a test session as usual. The same test may be conducted using $p'(a)$ instead of $p(a)$.	The *reversed pair* method proves to increases or decreases the performance compared to the original model.

Test Case 6: Exploring the new definition of the $p(a|b)$ parameter denoted $p'(a|b)$.

The test results using this new definition of the parameter $p'(a|b)$ is shown in Table 15. Again the window size parameter is varied from 3 to 6 and the pseudo count constant is 1.

	Basic MM	Reversed Pair Method	
Window Size		Using $p(a)$	Using $p'(a)$
3	48.8	49.9	50.6
4	51.0	51.7	52.6
5	51.2	52.0	53.1
6	50.2	51.2	52.3

Table 15: Results obtained using the reversed pair method.

The results show that using the method denoted as the reversed pair method, the performance increases for both the model using the parameter $p(a)$ and the model using the parameter $p'(a)$. However using the parameter $p'(a)$ yields the best results (with the single best Q_3 at 53.1% for window size 5).

The results show that the definition of the parameter $p'(a|b)$ increases the performance of the models. This may be explained by the fact that more information (the reversed subsequences) is added to the process of estimating the $p'(a|b)$ parameter.

Summing up the conclusion from Test Case 6:

- The reversed pair method increases the performance in general.
- The highest value for Q_3 using the reversed pair method is 53.1% using the $p'(a)$ term and a window size of 5.

7.2.2.1.4. Using Only $p'(a)$ Terms in the Score Function

Now the results concerning Test Case 7 and Test Case 8 will be presented. The first methods are based on the idea that the order of the residues may not have a large effect on the associated correct classification sequence. The second method actually introduces a sort of order constraint again (discount factor) but uses the first method as a basis. This first method be tested by substituting the order terms $p(a|b)$ with the non-order terms $p'(a)$ in the score functions (the exact method is described in the section 3.5.1.4.Using Only $p'(a)$ Terms in the Score Function).

Test Case 7	Method	Possible Conclusion(s)
Does the new score function (3.28 affect the performance of the models?	Complete a training session using the new score function for each Markov model, M_H, M_E and M_C.	Is this method better or worse than the original basic model?

Test Case 7: Testing the models without the order constraints between the residues.

The results associated with the tests described in Test Case 7 are shown in Table 16.

Window Size	**Basic MM using *p*'(*a*) and *p*(*a*|*b*)**	**New method using only *p*'(*a*) terms**
3	50.4	52.3
4	52.4	52.1
5	52.6	51.1
6	51.7	49.8

Table 16: Prediction accuracies for the new method, using non-order terms in the score function.

The results show that the new method performs well on small window sizes but the performance decreases somewhat when using larger window sizes. The explanation for this phenomenon may be found in that using a small window of few residues, the order may not be that important (since few residues are present and the window size is the number of terms in the score function). When enlarging the window size, the order may have more and more impact on the correct classification sequence, which is shown in the decreasing prediction accuracy for larger window sizes.

Summing up the conclusion from Test Case 7:

- The new score function only increase performance for window size 3 where Q_3 = 52.3%.
- For higher values of the window size the model performs worse than the basic model using p'(a).

The method is altered to incorporate a discounting function on each of the non-order terms *p*'(*a*). This method means that each term in the score function is discounted dependent on the position in the current window. It is therefore not possible to change the order of the residues without changing the value of the score function. In other words a different order constraint has now been introduced (compared to when using the *p*(*a*|*b*) terms).

Test Case 8 investigates the performance of the models using the discounting function.

Test Case 8	**Method**	**Possible Conclusion(s)**
Does the new score function (3.29 affect the performance of the models?	Complete a training session using the new score function for each Markov model, M_H, M_E and M_C.	Is this method better or worse than the original basic model?

Test Case 8: Test the altered score functions impact on the performance of the models.

The results from this test are shown in Table 17.

Window Size	**Basic MM using *p*'(*a*) and *p*(*a*|*b*)**	**New method using only *p*'(*a*)^l_i terms**
3	50.4	52.7
4	52.4	53.3
5	52.6	53.3
6	51.7	53.1

Table 17: Introducing the discounting function on the non-order terms.

This method seems to yield better results than the basic Markov model (using the *p*'(*a*) parameter instead of the *p*(*a*) parameter). This method is an extended version of the previous order-less method. The previous method performed quite well and was actually better than the basic model using *p*'(*a*) for small window sizes. Using larger window sizes the previous order-less method was worse than the basic model using *p*'(*a*).

The new method uses a discount factor. This factor affects the score function. It is now impossible to rearrange the residues without changing the value of the score function. This means that a order constraint has been reintroduced but not in the usual way using $p(a|b)$ terms.

The conclusion is that the new method has increased performance by introducing new concepts. Only $p'(a)$ values are used for calculating the score function. A new order constraint has been introduced, the discount factor.

Summing up the knowledge gained from Test Case 8:

- Using $p'(a)$ terms only increase the Q_3 for small window sizes but decreases the Q_3 for larger window sizes.
- Using $p'(a)$ terms only incorporating the discounting function the Q_3 is increased (the highest Q_3 value is 53.3% using window size 4 and 5).

7.2.2.2. Extensions Based On the Overall Score Function

The following sections show the results of the tests from the corresponding sections in 3.Using Markov Models for Classification. All tests in this section are performed using window size 4 and pseudo count 1 if nothing else is stated. Each section presents the test case. Comments and results are presented for every test case.

7.2.2.2.1. Adding Noise

Test Case 9 explores adding noise to the models. The noise is added to the individual score values before they are compared to find the largest one. The value of the noise-amount has to be determined so it does not dominate the final score value completely. The chosen values in the test have been selected by analyzing the average of the score values for all models. Table 18 shows the average individual score value for specific window sizes.

Window size	Average score value
1	5.84E-2
2	4.10E-3
3	2.44E-4
4	1.45E-5
5	8.70E-7
6	5.21E-8
7	3.11E-9
8	1.86E-10
9	1.11E-11
10	6.70E-13

Table 18: Average score values for specific window sizes.

Table 18 shows that for a window size of 4 the average score function output for all models is 1.46E-5. To test how the noise affects the models, the amount will be varied in the tests. The chosen values are shown in Test Case 9.

Test Case 9	Method	Possible Conclusion(s)
How are the models affected when noise is added?	Test the models in iterations using different amounts of noise from $s=0$ to $s=1.0 \cdot 10^{-4}$.	This method is worse or better than the basic Markov model.

Test Case 9: Exploring the noise amount added to the models.

Using the new model, where noise is added to the score function, yields the results depicted in Figure 18.

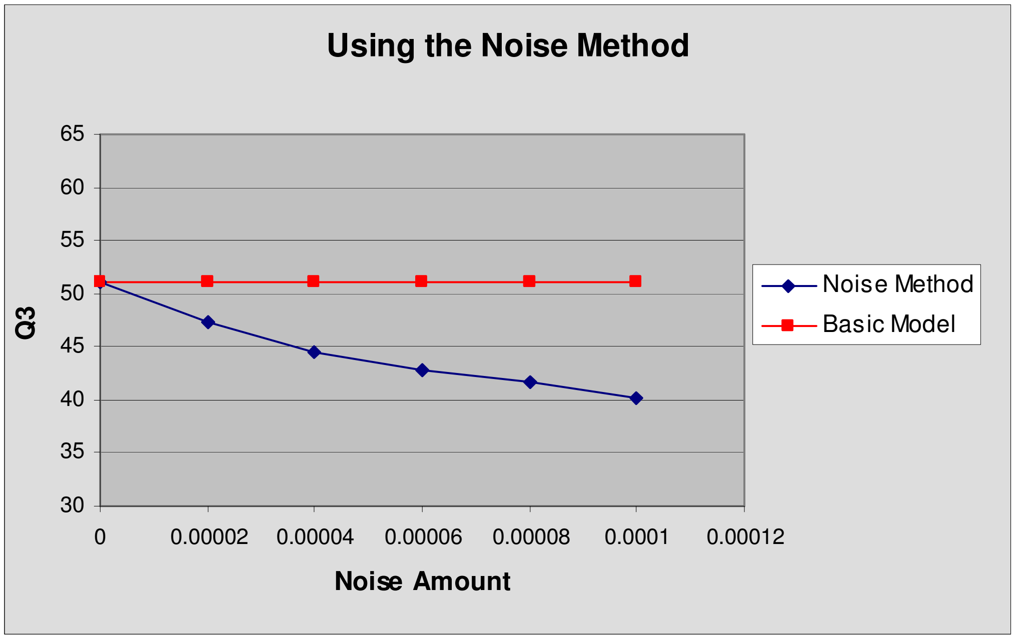


Figure 18: Adding noise to the output of the Markov models.

In Figure 18 we see that the performance of the model using the noise method decreases as the noise-amount is increased. The other curve (for the basic model) is shown as a reference and we see that the performance of the noise method is lower than the performance of the basic Markov model. The noise method may be considered worse than using the basic Markov model.

Taking a closer look at the graph we see that if the amount of noise equals zero then the model has a Q_3 of 51.1%. When the model has a zero noise amount the model is identical to the basic model using a pseudo count constant of 1 and a window size of 4.

The performance of the model converges towards a Q_3 of 33% when the amount of noise gradually increases. When the noise-amount is numerically large the random term will completely dominate the value of the score function. The score function will therefore return a random value. This means that the winning group will be completely random.

The probability of finding each of the three classification groups: helix, sheet and coil are given by p_H, p_E and p_C. The value of p_H, p_E and p_C is calculated by counting how often the group occurs in the database and dividing with the total number of residues in the database. The average performance of using a random guess can be calculated as:

$$Q_3 = \frac{1}{3}p_H + \frac{1}{3}p_E + \frac{1}{3}p_C = \frac{1}{3}(p_H + p_E + p_C) = \frac{1}{3} = 33\% \qquad (7.1$$

Summing up the conclusion from Test Case 9:

- Adding noise to the score functions does not seem to be useful.

7.2.2.2.2. Using Decision Constants

Test Case 10 explores the use of decision constants. Two tests are considered. The first test explores the difference method. This method requires knowledge of the numerical level of the score values returned by the individual score functions. The tests conducted while analyzing the noise method (from the previous section) provides information about this (see Table 18). The table shows the average numerical value of the score function when the window size is 4. This value is $1.4572{\cdot}10^{-5}$. The maximum value for the difference-limit is chosen to be $1{\cdot}10^{-5}$. In the test case the difference-limit (the decision constant) is varied in the interval $[0.0; 1{\cdot}10^{-5}]$.

Test Case 10	Method	Possible Conclusion(s)
Conduct a test using the difference method.	Complete several test sessions varying the decision constant r_1 and observe the results.	The difference method increases or decreases performance.
Conduct a test using the ratio method.	Complete several test sessions varying the decision constant r_2 and observe the results.	The ratio method increases or decreases performance.

Test Case 10: Using decision constants.

Figure 19 shows the test results obtained for the difference method.

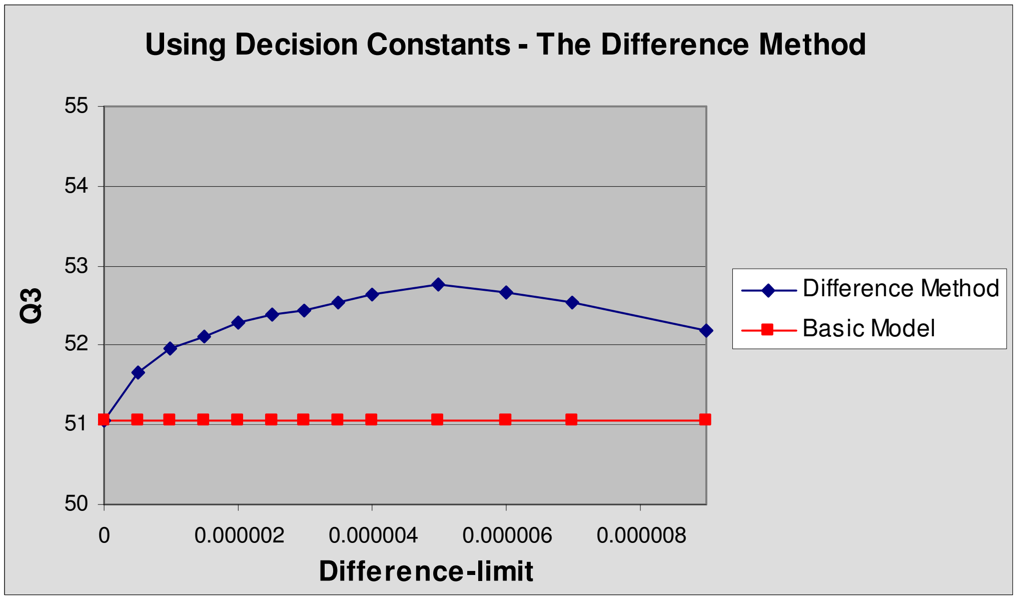


Figure 19: Q_3 as a function of the difference-limit using the difference method.

The graph shows that the difference method increases the performance of the model. The basic idea behind the model, that a new classification group is only assigned if the new score value exceeds the winning score value from the last round (last residue) by a certain amount seems to be a good idea.

When using this model we expect the predicted classification sequence to contain longer subsequences (residues classified as the same classification group). The method ensures that the model does not keep changing the predicted classification for every residue. By observing the training data (see the section 6.2.Database Distribution) we know that it is logically correct to have fewer longer subsequences instead of having many short subsequences only consisting of one or two residues. This point seems to be backed up by this test.

The problem with this method is that it requires knowledge about the numerical value of the score function. The difference limit is set with respect to this knowledge. This limit may be found experimentally by varying the parameter and observe the performances of the model. The limit may then be chosen from the model having the largest Q_3 value (as done in this section).

The ratio-I method is carried out the same way by varying the ratio-limit (the decision constant). In contrary to the difference method just described, the limit for this method is intuitively independent of the numerical level of the score values. The reason for this is that the ratio between two score values is calculated. Figure 20 shows the obtained prediction accuracies using this method.

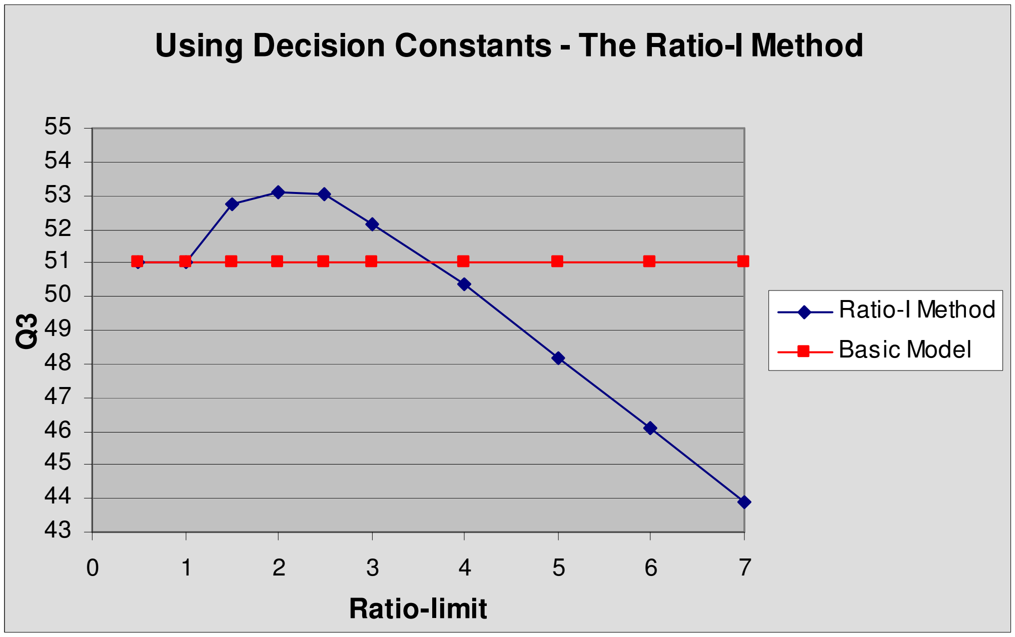


Figure 20: Shows the Q_3 as a function of the ratio-limit using the ratio-I method.

Figure 20 shows that for some values of the ratio limit the method increases the overall Q_3 and for some it decreases the Q_3. The curve seems to peak at the ratio limit of 2 which means that a given Markov model has to return a score value twice as large as the next greatest score value (from one of the other models) in order to change the classification group. If this is not the case the old classification (the classification group assigned for the previous residue) is used.

For a deeper analysis of this model see Ratio-I Method in Appendix C – Additional Analyses and Tests. The analysis carried out in the appendix shows how to use the accuracy matrix to analyze this model. The analysis provides an insight to the way the model works but does not find new information which might be used to improve the model.

Summing up the conclusion from Test Case 10:

- The difference method and ratio-I method increases the overall performance of the models.
- Using the difference method, the highest value of Q_3 = 52.8% is achieved having a difference-limit of $5.0 \cdot 10^{-6}$. The parameter value is very dependent of the window size.
- Using the ratio-I method, the highest value of Q_3 = 53.1% is achieved having a ratio-limit of 2.0.

7.2.2.2.3. The Ratio-II Method

Test Case 11 explores using the ratio-II method. The ratio limit is varied. As with the ratio-I method the limit does not depend on the window size, although it is possible that different ratio-limits lead to better results when the window size is varied. The ratio-limit is varied and the performance of the model is measured (the exact method is described in the section 3.5.2.2.1.The Ratio-II Method).

Test Case 11	Method	Possible conclusion(s)
Apply the *ratio-II method*	Vary the constant r_3 and observed the performance of the models.	The method increases or decreases the performance.

Test Case 11: Testing the ratio-II method.

The results of the test are shown the graph in Figure 21.

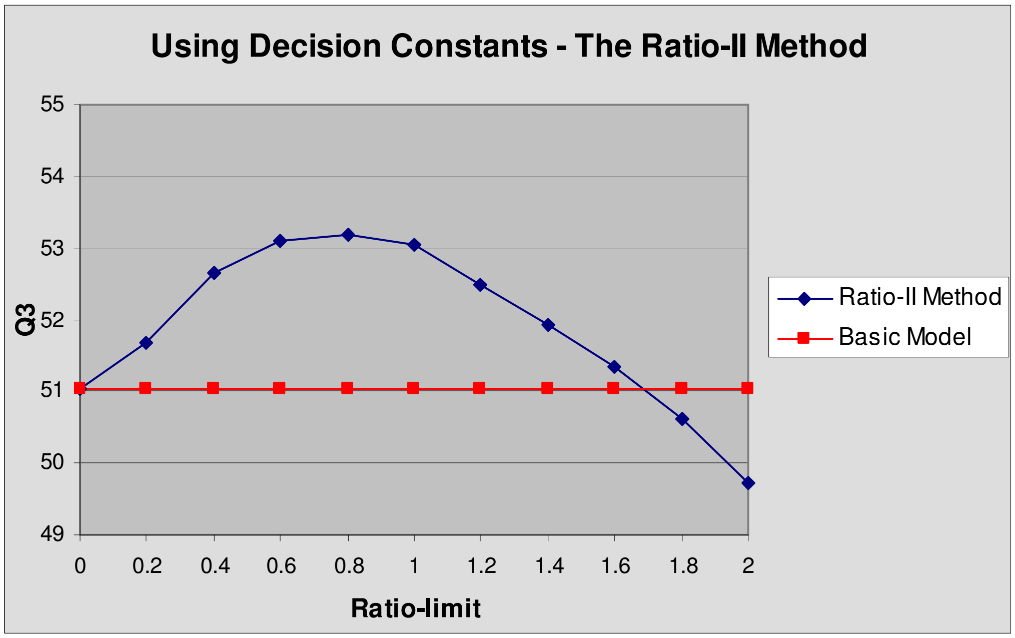


Figure 21: Performance of the Decision Ratio II method.

Figure 21 shows that the model using the ratio-II method has the highest prediction accuracy when the ratio value is between 0.6 and 1.0. When this limit is greater than 1.0 the performance drops and eventually (when the ratio is large) the model will perform worse than the basic model.

We will now try to explain what it means when the ratio-limit is in the interval [0.0; 1.0] and how come the performance drops drastically when the limit is increased above 1.0.

When the ratio-limit is 0 there is no restriction when choosing a new winning group (the model behaves as the basic Markov model). When the ratio-limit is less than 1.0 a restriction is added. The restriction ensures that the new highest score value needs to be closer to the previous winning score value, but it is not required to be the same value (or higher for that matter), to change the classification group. All values for the ratio-limit in the interval [0.0; 1.0] have the effects of a ratio-limit of 0.0 or 1.0 to some degree (the closer they are to one of them). Figure 21 shows that ratio-limits just below 1.0 seem to result in the highest prediction accuracies.

When the ratio-limit is greater than 1.0 it is a different situation. This situation means that the highest current score value needs to be greater than the previous score value in order to dictate a new winning group. The graph shows that larger ratio-limits make the model perform worse. In other words this means that if the restriction for changing the classification group is too strict, the performance drops (as in the case of a ratio-limit > 1.0).

Summing up the conclusion from Test Case 11:

- The ratio-II method increases the general performance of the model.
- The best Q_3 value (of 53.2%) is found using a ratio-limit of 0.8.

7.2.2.2.4. The Momentum Method

The momentum method is investigated in Test Case 12. The primitive momentum method does only have memory of the previous score value (for classifying the previous residue). The primitive momentum may be considered as a method which simply takes the previous classification into account before a new classification is made. If the momentum weight has a value of 0.5 it means that 50% of the previous score value will be added to the current. In other words the previous score value affects the current value by 50% of its own value. If the weight exceeds 1.0 it means that the previous value of the score function has a higher importance than the current when a residue is being classified.

Test Case 12	Method	Possible Conclusion(s)
Testing the primitive momentum method.	Run several test session varying the weight parameter and measure the performance of the models.	The primitive momentum method increases or decreases performance.

Test Case 12: Exploring the primitive momentum method by varying the weight, *w*.

Figure 22 shows the results obtained using the primitive momentum method.

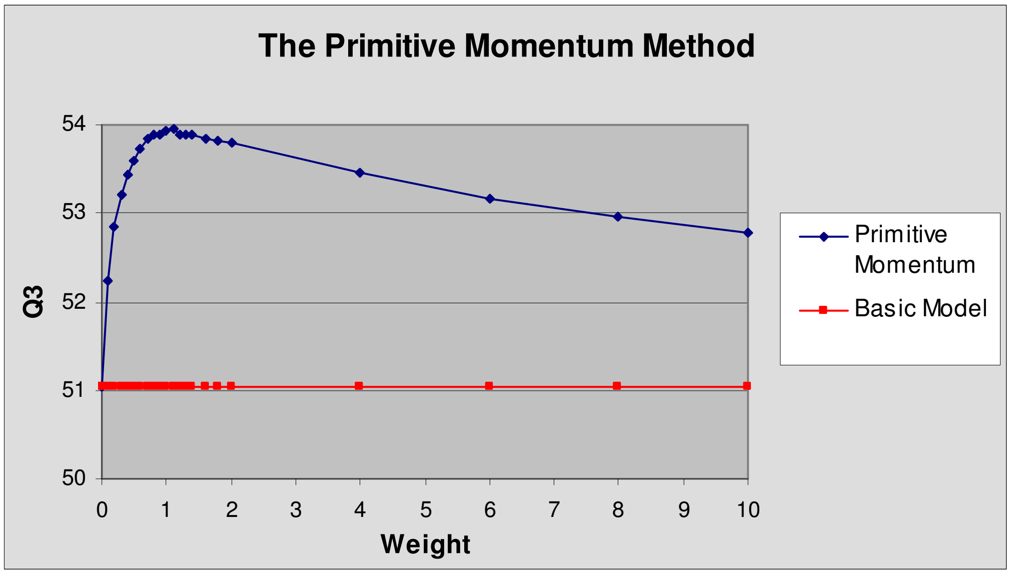


Figure 22: Performance of the primitive momentum method.

Figure 22 shows the performance of the primitive momentum method for window size = 4 when the weight is varied. The graph shows that the performance peaks at a weight value around 1.0. After this point the performance drops slowly.

It is interesting that the performance is relatively high when the value of the weight is greater than 1.0. Actually the performance for the weight value of 1.1 is slightly higher than for the weight value of 1.0. This means that the previous classification is weighted higher than the current classification.

We will now look at the meaning of a weight value of 1.0. If the value of the weight is 1.0 then the previous value of the score function is added to the current score value without any form of reduction. This means that if the previous residue is classified as a helix then the helix score value will be rewarded more than the other two score values for the current residue. We will therefore expect that if the helix group has a large score value for a residue then the next residue has a better chance of being classified as a helix (because the helix score value is rewarded more than the two other score values). If we look at the value of the score function we see that the previous and the current score value is weighted equally when the weight is 1.0.

Summing up the conclusion from Test Case 12:

- The primitive momentum method increases the general performance of the model.
- The highest performance (Q_3 = 53.9%) is obtained using a weight value of 1.1 (using a weight value of 1.0 yields the same result, only 0.02% points lower).

The following presents the real momentum method which not only takes the previous score value into account but all previous values.

The test of the real momentum method is contained in Test Case 13. This momentum method has memory back to the beginning of the sequence. We will therefore expect that weight values over 1.0 will cause the score value to explode (and the Q_3 to drop drastically) because weight values greater than 1.0 will cause the previous score values to be weighted higher than the current.

Test Case 13	**Method**	**Possible Conclusion(s)**
Testing the real momentum method.	Run several test session varying the weight parameter and measure the performance of the models.	The real momentum method increases or decreases performance.

Test Case 13: Exploring the real momentum method by varying the weight, *w* and incorporating more memory.

Figure 23 shows the outputted graph of the results obtained using the real momentum method.

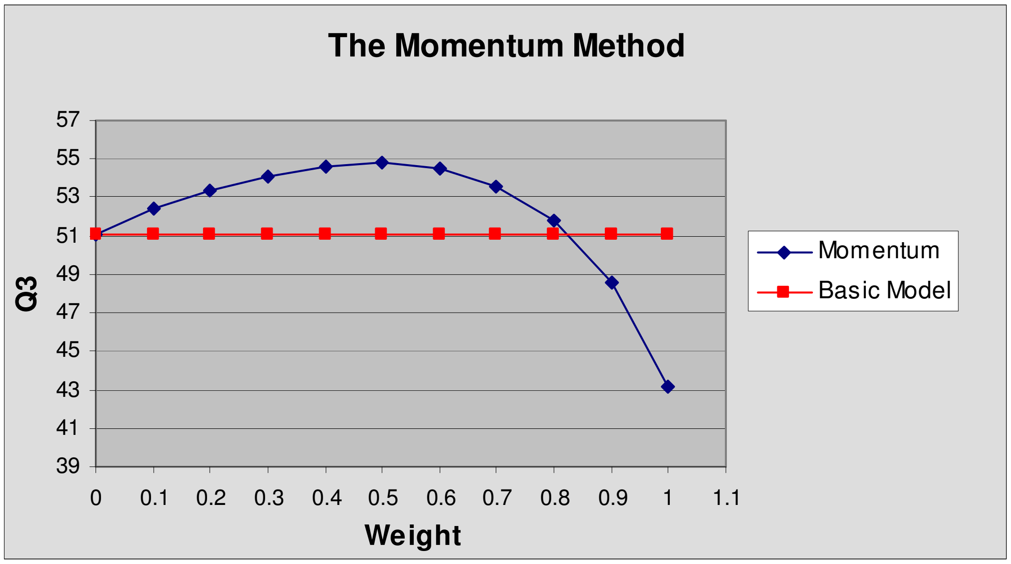


Figure 23: Performance of the momentum method.

The graph (in Figure 23) has the same tendency as the ratio-I method and the ratio-II method although the methods in theory are different. The graph shows that the momentum model performs better than the basic model in a large interval. The graph shows that the maximum Q_3=54.8% is obtained for a weight of 0.5. This means that the resulting score value is the current score value plus 50% (0.5) of the last score value for the same model and 25% (0.5^2) of the previous value and so on. We see that having a weight of 1.0 has decreased the performance of the model drastically (and is below the performance of the basic model).

The conclusion is that it is possible to gain performance using the momentum method. Using this method the frequent group changes in the predicted classification sequence are minimized, in that the current individual score value is dependent on the previous individual score value (and the previous group will be chosen more).

Summing up the conclusion from Test Case 13:

- The real momentum method increases performance of the model (compared to the performance of the basic Markov model).
- The highest performance (Q_3 = 54.8%) is obtained using a weight value of 0.5.

7.2.2.3. Extensions Based On Alternating the Training Procedure

This method is dealt with in Test Case 14. The method alters the subsequences that are used to train the Markov model. For each subsequence *l*-1 residues are appended where *l* is the window size. The idea is to avoid the problem that might arise when more than one subsequence is in the window (explained in the section 3.5.3. Extensions Based On Alternating the Training Procedure).

Test Case 14	Method	Possible Conclusion(s)
Test the new method of generating the subsequences for training of the models.	Conduct a test session using this new method.	The models may perform better, incorporating knowledge about the overlapping subsequences.

Test Case 14: Testing the new method of extracting the subsequences.

The following table shows the test results:

Window Size	Q_3
3	49.1
4	50.3
5	50.2
6	49.5

Table 19: Performance using the new method of extracting the subsequences.

The table shows the performance using the new method of extracting the subsequences with different window sizes. The performance is around 50% for window sizes from 3 to 6. This performance is worse than the basic model for window size 4,5,6 and slightly better than the basic model using window size 3.

The reason why we do not see any increase in the performance has been given in the corresponding section in 3.Using Markov Models for Classification. The explanation given there is that a new problem is introduced. The following section tries to deal with the original problem namely that the Markov model might have problems predicting the ends of the subsequences.

Summing up the conclusion from Test Case 14:

- The new score function does not increases the performance of the model and does therefore not solve the issue.

7.2.2.3.1. Classifying the Ends of Subsequences

Test Case 15 investigates the question about whether or not the Markov model has problems classifying whenever two or more subsequences overlap in the window.

Test Case 15	Method	Possible Conclusion(s)
Test whether or not the Markov model fails to predict overlapping subsequences in window.	Whenever the window contains residues of two or more different subsequences the window is shifted to the beginning of the next subsequence. All skipped residues are not classified. The window size is varied.	If performance increases the Markov models may have problems classifying small subsequences or ends of the subsequences.

Test Case 15: Skipping classification of ends of the subsequences.

The test has been performed varying the window size from 3 to 10. For every test the number of skipped residues and the total number of observed residues have been recorded. These numbers are used to calculate the percentage of how many residues are skipped during the classification process. The intention of this method is to make it clear whether or not the models have problems classifying the residues in a window that overlaps one or more subsequences. The results obtained (using the above method) are shown in Table 20.

Window Size	Skipped residues in %	Performance (Q_3) in %
3	32.8	50.2
4	45.9	54.8
5	56.6	57.8
6	65.2	59.8
7	72.0	62.2
8	77.3	64.3
9	81.6	65.9
10	85.1	67.8

Table 20: Results obtained using the test method. Skip % is the percent of all residues not classified. Performance is the Q_3 values obtained.

Table 20 shows that the performance increases when the window size is increased. It may appear as if the model is having problems classifying when two or more subsequences are present in the current window, but this conclusion can not be drawn. Table 21 shows why this is not possible.

(H) Distribution in %	(E) Distribution in %	(C) Distribution in %
40.9	20.0	38.9
44.2	18.6	37.1
48.3	16.5	35.0
52.8	14.0	33.0
57.0	11.3	31.5
60.7	8.9	30.3
63.7	6.8	29.3
66.0	5.3	28.5

Table 21: The observed distribution of the three groups H, E and C.

The distribution of the residues according to group they are correctly classified as, is presented for the DSSP database in section 6.2.Database Distribution Table 9. In this table we see that 34% the residues is of group H, 23% is of group E and 43% is of group C. Table 21 shows the distribution when we apply the new test method. The method involves a procedure where residues are skipped. This causes residues of certain groups to be skipped more than others. Groups that have a large fraction of subsequences of small size will often be skipped using this method. This is the case for the E and C group because a large fraction of these groups is small subsequences. When the window size is increased this effect becomes more and more obvious.

In Table 21 we see that already at window size 3, the helix group is the most frequently observed group. As the window size increases the percentage of helix groups observed is increased. For all window sizes greater than 3 the helix group is the most frequent (since the other groups are skipped). This means that the trivial classifier (always predicting the helix group) has a prediction accuracy equal to the distribution of the helix group. Figure 24 shows the performances obtained in Table 20 and Table 21 as graphs.

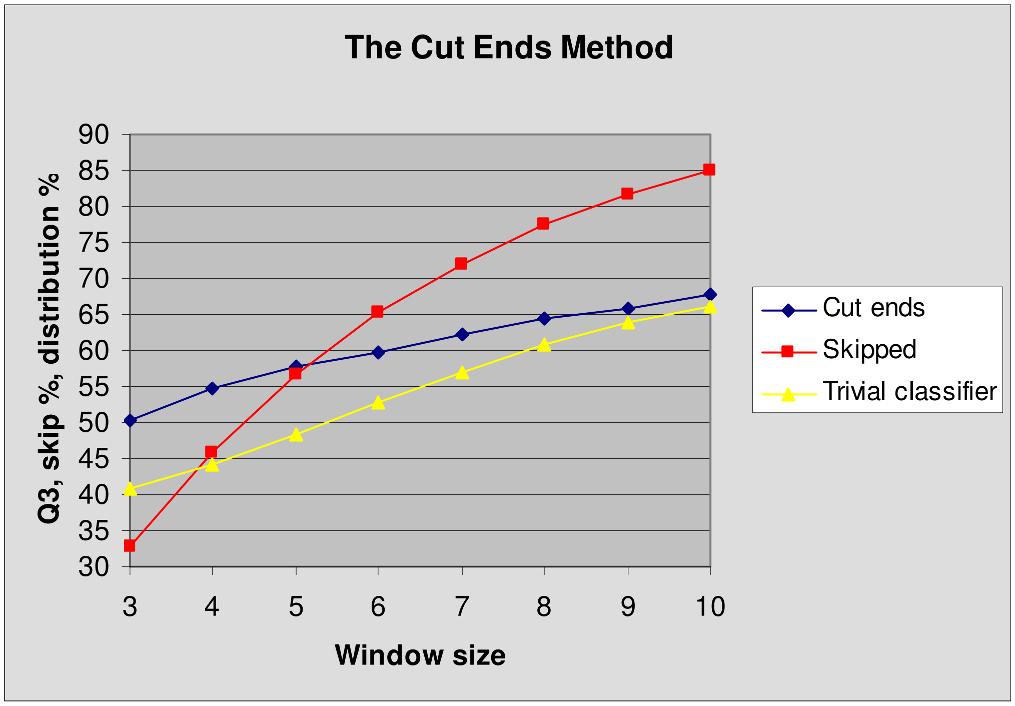


Figure 24: Graph showing both the % of skipped residues and the increasing performance as we increase the window size.

The graph on Figure 24 shows three curves, one showing the performance (Q_3) of the model as a function of the window size (Cut ends). Another curve shows the percentage of skipped residues as a function of the window size (Skipped). The last curve shows the distribution of helices as a function of the window size. This value is the same as the performance of the trivial classifier always predicting the helix group. For example using a window size of 7, between 70% and 75% of all residues are skipped using this method. The prediction accuracy for the model is between 60% and 65% and between 55% and 60% for the trivial classifier using a window size of 7.

At window size 3 the blue curve (Cut Ends) is 10% better than then yellow curve (the trivial classifier). This interval is almost the same as comparing the basic model to the trivial classifier (predicting the coil group). The difference between the blue and the yellow curve gets smaller as the window size is increased. This means that the model gets closer to the trivial classifier. The performance of the trivial classifier is an absolute minimum reference the performances are compared. The fact that the model performance gets closer to the trivial classifier shows that the model actually performs worse in the sense of how usable the model is.

The classifier introduced has changed the distribution of the groups. This fact causes the results to be inconclusive. The data has been changed and the model is therefore not tested in a reliable environment. The conclusion is therefore that the test scheme is invalid and that no real conclusion can be drawn.

Summing up the conclusion from Test Case 15:

- This test turned out to be invalid.
- The method changed the distribution of the groups. It is shown that the performance actually did not improve compared to the trivial classifier.

7.2.2.3.2. The Swap Residue Method

The next method tested is the one denoted as the swap residue method. The idea of this method is that a number of extra subsequences are added to the training data and therefore more data is available when the parameters are estimated. From each subsequence several new subsequences are generated. In each of the new subsequences two residues following each other (in the original subsequence) switch places.

The procedure generates more subsequences for the models to be trained on, but the subsequences are alike, except for the residues that have switched place.

This method is tested using Test Case 16.

Test Case 16	Method	Possible Conclusion(s)
Doest the added subsequences (the swap residue method) increase the performance of the models?	Complete a training session using this method, by adding the subsequences generated from the original one actually in the training data.	The method may increase or decrease the performance of the models.

Test Case 16: Testing the new method, the *swap residue* method.

The prediction accuracies obtained from this test are shown in Table 22.

Window size	Basic MM $p(a)$	Basic MM $p'(a)$	Swap Residue $p(a)$	Swap Residue $p'(a)$
3	48.8	50.4	50.0	50.5
4	51.0	52.4	51.8	52.7
5	51.2	52.6	52.2	52.8
6	50.2	51.7	51.4	52.2

Table 22: Prediction accuracies obtained using the swap residue method.

The tests show that this new method actually increases the performance compared to the original Markov models. The models using the swap residue method performs better than the corresponding basic Markov models using the parameters $p(a)$ and $p'(a)$. The swap residue method improves the Q_3 by 0.2-1.2% when using the $p(a)$ parameter for the different window sizes. Using the $p'(a)$ parameter the Q_3 is increased from 0.1-0.5%. The larger increase in Q_3 when the $p(a)$ parameter is used might be because the $p(a)$ parameter indirectly also changes when the subsequences are modified.

The conclusion is that it seems like it is possible to switch two residues in a pair positioned in a subsequence and still have the same correct classification for that sequence.

Summing up the conclusion from Test Case 16:

- This method increases the general performance of the model.
- The highest value of Q_3 is 52.8% using a window size of 5 and the $p'(a)$ term.

7.2.3. Normalizing the Markov Models

Two different normalization methods have been implemented for use with the Markov models. The basic idea is to combine models using different window sizes but classifying the same group, to incorporate more information (more models) when predicting the secondary structure. Hopefully this will lead to an increase in the performance of the models.

7.2.3.1. Basic Normalization

The exact normalization procedure for this method is described in the associated section 3.6.1.Basic Normalization. Basically it scales the outputs obtained from the different models using different window sizes and adds this up to a total (for each classification group). These values are then used to decide which classification group is assigned for the residue being predicted.

This normalization scheme is investigated further in Test Case 17.

Test Case 17	Method	Possible Conclusion(s)
Test the basic normalization procedure.	Run the test. Vary the window sizes and vary the number of models to be normalized.	Does the method increase or decrease the performance?

Test Case 17: Testing the basic normalization method.

The tests using the basic normalization scheme incorporates two basic variables, which is the number of different models that are normalized and the window sizes used for each model. The normalization scheme is built in such a way that a starting value for the window size is chosen and then the number of models to normalize is chosen.

Let us assume that we choose windows size, *l* = 3 and the number of normalized models to 4. This would mean that models using window size = {3, 4, 5, 6} are normalized. The window size given is the starting value for this variable. It is not possible to normalize two models using window size 5 and 10 in this way.

However, this could easily be implemented, but the above approach is chosen since the initial test results show that window sizes in the interval [3, 6] are the most promising.

To test the basic normalization scheme we have chosen the window size and the number of normalized models in an appropriate interval, using window sizes beginning from 2 to 5 and normalizing 3, 4 and 5 models. The overall test results using these values are shown in Table 23.

	Normalized Models		
Window Size	3	4	5
2	49.4	50.5	51.0
3	51.3	51.6	51.7
4	51.6	51.5	51.2
5	51.0	50.8	50.5

Table 23: Prediction accuracies using the basic normalization scheme.

Table 23 should be interpreted as explained in the following. The value available in the first row and the first column of the table is the overall Q_3 when normalizing three models using window sizes 2, 3 and 4. In this case the Q_3 value is 49.4%.

It is obvious that the scheme does not increase the overall Q_3 drastically. The best Q_3 obtained using the basic Markov model without the normalization scheme is 51.2%. Now using this scheme the best Q_3 is 51.7%, an increase of 0.5% (normalizing 5 models using window sizes 3, 4, 5, 6 and 7).

One reason that this scheme is not drastically better than the basic Markov model may be that each model which affects the normalized score value has as much influence as any other model. In other words the models using different window sizes (which are normalized yielding a normalized score value) are not weighted in any way. Including three models which performs equally well (having around the same overall Q_3) and including one model which is worse, may pull down the overall Q_3 of the normalized models.

However the normalization scheme does show a small increase in the Q_3 values. For instance normalizing the models using window sizes 4, 5 and 6 yields a Q_3 of 51.6%. Using the basic Markov models for each window size (4, 5 and 6) the Q_3 results are 51.0%, 51.2% and 50.2% respectively. The average is then 50.8% where as the normalized result is 51.6%, an increase of 0.8%. This seems to be the general case, that the normalized Q_3 is actually better than the average of using the models separately.

Summing up the conclusion from Test Case 17:

- This method increases the performance of the model slightly.
- The highest value of Q_3 is 51.7% normalizing the models using window sizes: 3, 4, 5, 6, 7.

7.2.3.2. Frequency Normalization

The other normalization method which we have implemented is denoted as the *frequency normalization* method, where the outputs of the score function is normalized and weighted using frequencies calculated based on the subsequences in the training data. The method is described in more details in the section 3.6.2.Frequency Normalization and is treated in Test Case 18.

Test Case 18	Method	Possible Conclusion(s)
Test the frequency normalization procedure.	Run the test. Vary the window sizes and vary the number of models to be normalized.	Does the method increase or decrease the performance?

Test Case 18: Testing the frequency normalization method.

In the section 3.6.2.Frequency Normalization it is described how the frequencies used for weighting may be calculated in three different ways. Each method has been tested separately and it seems that the second option for calculating the frequencies is yields the best results (although the difference is small).

The results presented here are based on the models using the second option for calculating the frequencies. The frequencies of subsequences having the exact lengths are calculated as described in 3.6.2.Frequency Normalization.

The frequency normalization procedure is tested the same way as the basic normalization method described in the previous section. The beginning window sizes are {2, 3, 4, 5} and the number of normalized models is {3, 4, 5}. This means that each combination of the values mentioned are tested. The first combination having a beginning window size of 2 and normalizing 3 models means that models using window sizes 2, 3 and 4 are normalized.

The results of this test are showed in Table 24.

	Normalized Models		
Window Size	3	4	5
2	49.1	49.8	50.1
3	50.7	50.9	51.2
4	51.5	51.7	51.7
5	51.3	51.2	51.2

Table 24: Prediction accuracies using the frequency normalization method.

This normalization method does not differ too much from the basic normalization method treated in the previous section. The results are not drastically different from using the basic Markov model (for a fixed window size) but nevertheless each result is better than the average of the Q_3 values for the models which are normalized. This means that we actually do gain some performance using the normalization scheme, although it is not a drastic change (the performance increase is around 0.5-1.5%).

Looking at the two normalization methods presented the frequency scheme has a tendency to be better when normalizing more models than the basic normalization scheme from the previous section. This may be due to the fact the each term is weighted (using the information obtained calculating the frequencies of the subsequences for each group).

The normalization method denoted as the frequency normalization may very well be used to increase the performance and incorporate more models into the prediction scheme, even though the prediction accuracy does not seem to increase drastically.

Summing up the conclusion from Test Case 18:

- The frequency normalization method increases the performance of the models slightly.
- The highest obtained value of Q_3 is 51.7% when normalizing window sizes: 4, 5, 6, 7 or window sizes: 4, 5, 6, 7, 8.

7.2.4. Combining Markov Models Having Different Orientations

This section deals with the combo model, which combines prediction of the residue at position *i* using both the residues at the positions before *i* and after *i*. The idea is to use two basic Markov models, which are trained using different orientations. The exact idea behind this model is described in details in section 3.7.Combining Markov Models Having Different Orientation.

Hopefully the overall prediction accuracies are higher using this combined model compared to the basic Markov model. The combined model using both orientations is tested using Test Case 19.

Test Case 19	**Method**	**Possible Conclusion(s)**
Test the combo model described above.	Run the test for various window sizes.	Does the method increase or decrease the performance?

Test Case 19: Testing the combo model (using normal and reversed orientation).

The results are shown in Table 25.

Window Size	**Func**: $sc_{forward} + sc_{backward}$	**Basic Markov Model**
3	51.0	48.8
4	53.3	51.0
5	53.9	51.2
6	53.2	50.2

Table 25: Results using the combined model.

The obtained prediction accuracies using this new model are obviously higher than using the basic Markov model (which corresponds to only using the $sc_{forward}$ value). In most cases the increase in Q_3 is larger than 2%. The largest increase is when using a window size of 5, where the basic Markov model has a Q_3 of 51.2% and the combined model has a Q_3 of 53.9%, an increase of 2.7%.

Other functions for using the $sc_{forward}$ and $sc_{backward}$ values have been tested. Taking the minimum or the maximum of the values and using this value as the overall score value did not show either an increase or a decrease of the overall performance of the model.

This combined model seems to be a good idea based on the overall performance of the model. The explanation for this increase of the overall Q_3 value may be found in the fact that $2l$-1 residues are taken into account for every residue that is classified (where *l* is the window size). Only *l* residues are taken into account when using the basic Markov model. The combined model predicts the classification sequence using more information about the sequence to be predicted than the basic Markov model normally does.

Summing up the conclusion from Test Case 19:

- This method of combining two Markov models with different orientation increases the performance of the model.
- The highest obtained value of Q_3 is 53.9% using a window size of 5.

7.2.5. Combining Different Extensions

This section presents the results obtained by running Test Case 20. The test reveals if it is possible to achieve higher prediction accuracies by combining several different model extensions.

Test Case 20	Method	Possible Conclusion(s)
Test the combination of *p*'(*a*), the reversed pair method, frequency normalization and momentum.	Run the test using the best values for the each method (for extensions having parameters associated).	Is it possible to increase the performance by combining different extensions?

Test Case 20: Testing combinations of different extensions.

The following extensions are chosen to be combined into one model:

- The new definition of the parameter denoted as *p*'(*a*).
- The reversed pair method
- Frequency normalization
- The momentum method

The resulting performance obtained by running the test is 57.2% (see Table 26). This performance is actually higher than all other performances obtained previously. This means that we have reached a higher level of performance by combining different extensions that as standalone extensions has a lower performance.

Window size	Basic Markov Model	Combination of extensions
4	51.0	57.2

Table 26: Prediction accuracies obtained using a combination of extensions.

To see how this model has improved compared to the original basic Markov model only using pseudo counts the following graph is presented (see Figure 25).

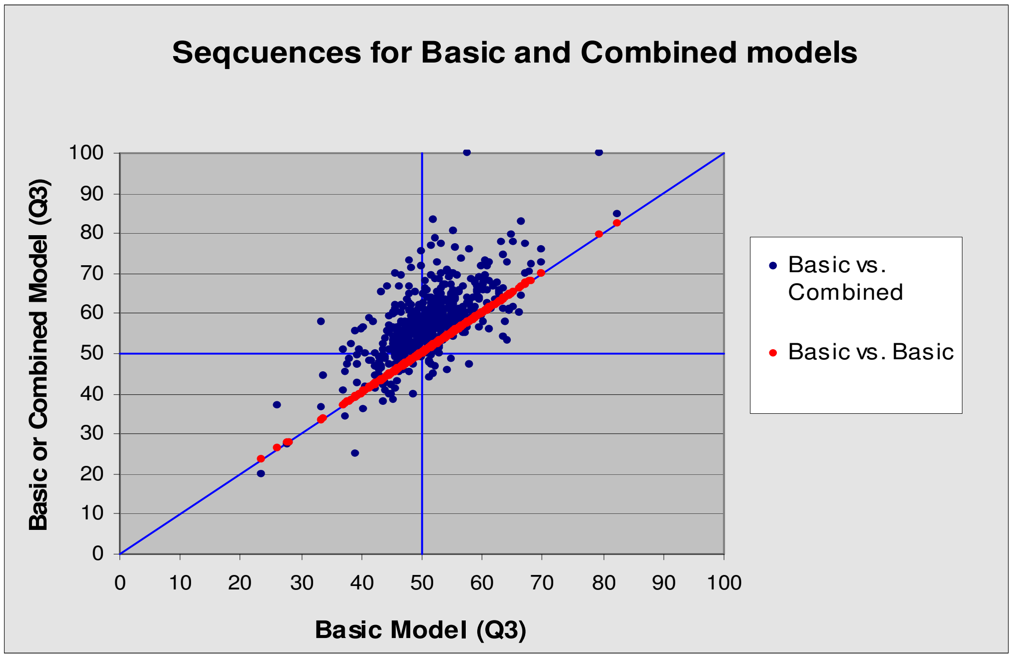


Figure 25: Comparison of the new combined model with the basic Markov model.

Figure 25 shows a comparison of the combined model and the basic model per sequences in the training data. If we only look at the red dots we see that they are placed at the diagonal. Each red dot represents a pr. sequence Q_3 obtained testing the basic Markov model on the DSSP database (it is the individual prediction accuracies obtained for each test sequence). A single red dot is plotted by measuring the Q_3 value pr. sequence for a single sequence in the training data using the basic model and then plot it at coordinates (x,y) = (Q_3, Q_3). Looking at the red dots we see that they seem to be concentrated in the interval from 0.4 to 0.6 which means that a sequence typically is classified with a Q_3 value between 0.4 and 0.6 using the basic model.

The pr. sequence Q_3 values obtained using the basic model will be denoted as Q_{3basic}. In the same way the pr. sequence Q_3 values obtained using the combined model will be denoted as $Q_{3combination}$.

The blue dots represent the pr. sequence prediction accuracies obtained for each sequence in the database using the basic model and the combined model respectively. The values are plotted using the coordinates (x,y) = (Q_{3basic}, $Q_{3combination}$). This means that if the first sequence in the database has $Q_{3basic} = 0.4$ and $Q_{3combination} = 0.6$ a point will be plotted at (x,y) = (0.4; 0.6).

The graph is useful for comparing how the two models classify every single sequence and how the new model has been improved. Table 27 describes how the graph may be interpreted.

Condition	**Explanation**	**Graph**
$Q_{3\ basic} = Q_{3\ combination}$	Both models classify the sequence with the same accuracy.	The point is plotted *on* the diagonal.
$Q_{3\ basic} < Q_{3\ combination}$	The combined model classifies with the highest accuracy.	The point is plotted *above* the diagonal
$Q_{3\ basic} > Q_{3\ combination}$	The basic model classifies with the highest accuracy.	The point is plotted *below* the diagonal.

Table 27: A description of how to interpret the graph in Figure 25.

Now using the table we see that most of the points are above the diagonal which means that the combined model is classifying most sequences more correctly than the basic model. Only a few points are placed below the diagonal which means that the combined model predicts those sequences worse than the basic model.

The graph shows that the high performance achieved using the new combined model is not the result of increasing the pr. sequence prediction accuracy for a few sequences and leaving the rest unaffected. On the contrary it seems that the new model does indeed increase the performance of almost every sequence in the database, resulting in a higher overall performance (Q_3).

The conclusion that may be drawn based on the graph above, is that overall ability to classify the sequences has been improved when a combination of extensions are applied. The main part of all sequences in the training data is classified with a higher accuracy, which is seen by the fact that most points are placed above the diagonal line.

Summing up the knowledge gained in this section:

- Combining different extensions may improve the overall Q_3 even more than when using the individual extensions.

7.3. Test Results Using the GOR Classifier

The GOR model has been implemented in different versions. GOR I, III and IV has been implemented following the definitions of the models described in the section 4.The GOR Classifier. However there are small differences in our implementations compared to the ones described in [2].

Every model is tested on the DSSP database presented by Cuff and Barton [5, 6] containing 513 non-redundant protein sequences and also on the GOR database which was used in [2] (the database was kindly provided by the authors of [2]).

Additionally the following applies:

- All models are used to predict the three different classification groups H, E and C. This includes GOR I which was originally developed to predict four different classification groups.
- GOR III has been used in the raw version (presented in 4.The GOR Classifier section) without using dummy frequencies (since the DSSP database contains more than enough information to do this).
- GOR IV has been implemented and used in the original form, without using any decision constants in the prediction process and without altering the sequences from the database (such as treating helical subsequences below length 5 as coils and so on [2]).

The GOR models are primarily used as benchmark tools for our own Markov models. Since we test the GOR models and the Markov models on the same data the results are directly comparable.

The results obtained using our implementation of the GOR models are summarized in Table 28:

Model version	**Database**	**Q_3**
GOR I	GOR	60.7
	DSSP	60.5
GOR III	GOR	59.6
	DSSP	60.0
GOR IV	GOR	63.4
	DSSP	63.4

Table 28: Results obtained using our implementation of the different GOR models.

Remember that the GOR database mentioned in Table 28 is the exact same database used in [2].

The table shows that our implementation of GOR I appears to have a higher prediction accuracy than the 55% reported in [2]. Again the 55% prediction accuracy may be on a much smaller database (and using four different classification groups). Also the table shows that there is almost no difference in using either of the two databases (the DSSP and the GOR database) even though the DSSP database contains around 20.000 more residues (and around 250 more sequences). One explanation for this may be that the sequences in the GOR database hold enough information to train the classifiers properly.

The GOR III model is documented to have a prediction accuracy of 63.3% using dummy frequencies and decision constants. Our implementation of GOR III reaches a prediction accuracy of 60.0% without using dummy frequencies or decision constants.

Also we see that our implementation of GOR IV has an overall prediction rate of 63.4% whereas it was reported to be 64.4% in [2] using the same database (the GOR database). This leads to the fact that there may be small differences in our implementation of the GOR models compared to the implementation described in [2] (this was mentioned earlier).

In the following section, some of the more specific results obtained using our implementation of the GOR models will be presented.

To get an understanding of the GOR models, we have investigated the models more closely. We have tried to vary the window size (which is normally fixed at 17, where the center residue is the residue to be predicted) and seen how this affected the models. These tests were carried out before we discovered that a resizable window had been implemented in the fifth version of GOR.

7.3.1. Test Results Using the GOR Model with Various Window Sizes

To get an idea about the GOR model, we have tried to vary the window size for GOR I and GOR III. The results for the DSSP database and the GOR database are shown in Figure 26.

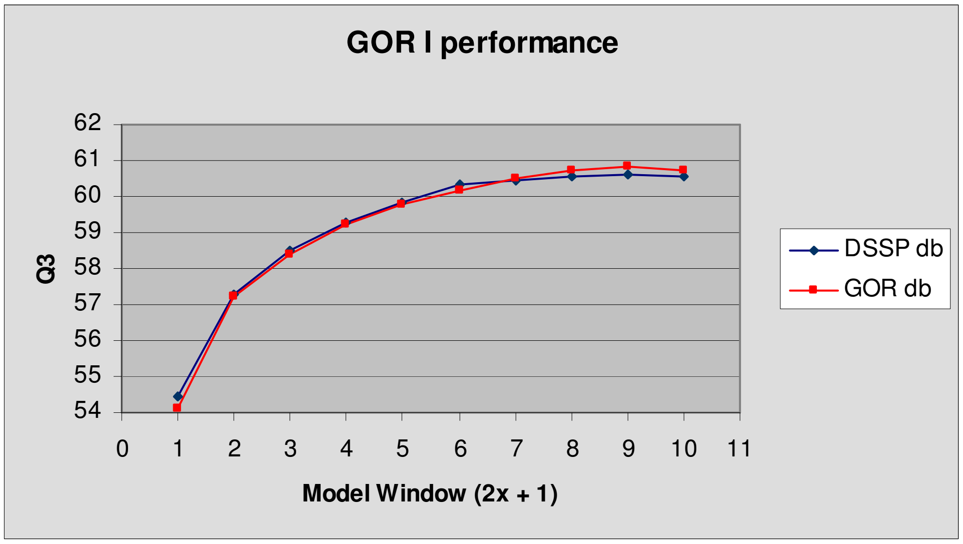


Figure 26: Performance of the GOR I model.

In the case of GOR I, the graph shows that using the larger DSSP database does not increase the prediction accuracy. In fact they look almost the same. Also it is worth noting that the window size of 17 (*x* = 8) gives in fact one of the best overall prediction accuracies. Increasing the window size seems to introduce some over fitting of the model.

In the case of GOR III the graph is a bit different (see Figure 27). First of all there is an obvious difference between the two databases, even though it is only around 0.5% point. A more interesting point is the fact that the performance begins to decrease for window sizes greater than 15 (x > 7). This shows that the window size of 17 used in the paper [2] may not be the window size producing the best overall results (a window size of 15, x = 7 appears to be even better).

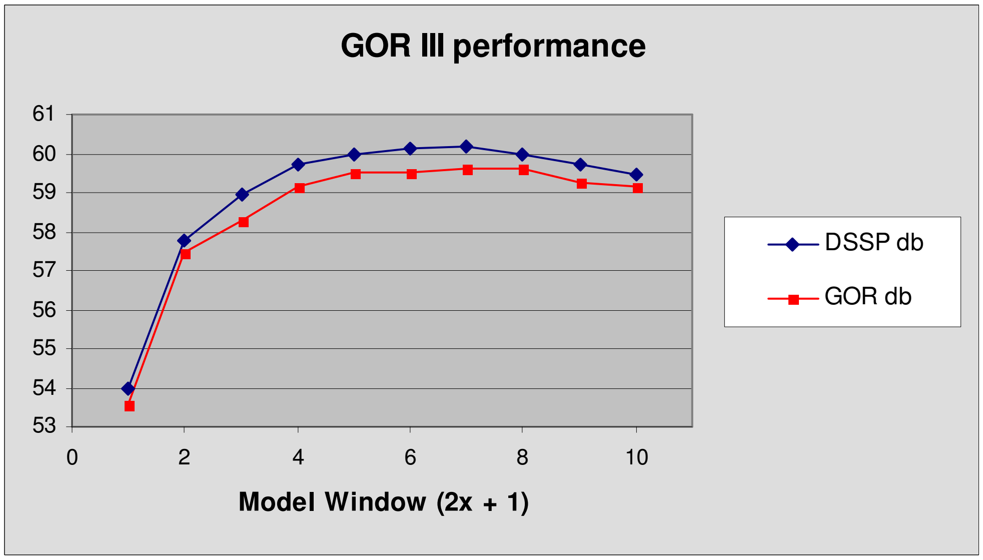


Figure 27: GOR III performance on the DSSP and GOR databases.

The GOR models have also been tested on smaller datasets (subsets of the databases) containing less than 100 sequences. Since GOR I uses less information than GOR III (which includes pair statistics), GOR I actually performs quite good on smaller datasets. The prediction accuracy was only decreased by a few percent compared to the overall result of around 60.5% when using the full databases.

Using GOR III on smaller subsets of the databases does decrease the performance a lot more than for the GOR I model. In this case the prediction accuracy drops drastically to around 50% and lower for datasets containing less than 50 sequences. The reason for this observation lies in the pair statistics incorporated in the GOR III model. The statistics (frequencies) calculated become a problem when the database is too small (since some of the combinations of residues are not in the training data).

The GOR IV model has not been tested using different window sizes.

7.4. Comparison of the GOR Models with the Markov Models

The conducted tests using both the GOR models and the different Markov models show that the GOR models are superior to the Markov models, in that higher prediction accuracies are achieved.

The first GOR model (GOR I) reaches a prediction accuracy of 60.5% using the DSSP database. This prediction accuracy is obtained using our own implementation of the model. The best Markov model developed, which is a combination of different extensions achieve a prediction accuracy of 57.3%.

It seems that the classification problem of assigning the different secondary structure classes to each residue is more successful using the GOR models. The fourth GOR model (GOR IV) has a prediction accuracy of 63.4% using our implementation of the model. Although this is around 1% less than the documented prediction accuracy it is a lot more than 57% (the Q_3 for the highest scoring Markov model).

In this section we will try to find a possible explanation to why the GOR models are superior to the Markov models when it comes to the overall prediction accuracy.

The problem of classifying the secondary structure of protein sequences may be thought of as capturing the necessary information from the training sequences and using this information to create a model, which is able to predict the structure for an independent test sequence (not in the training data).

One way of looking at how much information is captured in the models, is to look at the number of parameters in the different models. The parameters are used in the classification process when some group is assigned for every residue. The initial Markov models are based on the parameters $p(a)$ and $p(a|b)$, which have been redefined several times in the previous sections. The general case is that there are $g(s^2 + s)$ parameters for each Markov model predicting a particular group (g is the number of groups and $s = |\Sigma|$). If we consider each group H, E and C then $\underline{3\cdot(s^2 + s)}$ parameters are used.

The first GOR model (GOR I) does not use the same number of parameters as the Markov model. The GOR I model uses $2\cdot w\cdot g\cdot s+2\cdot g$ parameters where w denotes the number of residues in the window. The first term $2\cdot w\cdot g\cdot s$ is the parameters used for pairs consisting of a residue and a group for each position in the window these values are multiplied with two because of the information difference (see 4.2.The GOR Idea in Theory). The second term $2\cdot g$ is the parameters used for the frequency of each *group* and *not group* (see 4.3.1.GOR I for further details of GOR I model).

In general the GOR models uses a window size of 17, making w = 17. Having three different classification groups the number of parameters is $2 \cdot 17 \cdot 3 \cdot s + 2 \cdot 3 = \underline{102s+6}$.

Having the alphabet size $|\Sigma| = s = 20$ and $w = 17$, the GOR model has $102 \cdot 20+6 = 2046$ parameters. In comparison the Markov model has $3 \cdot (20^2 + 20) = 1260$ parameters (that is the total number of parameters used, when predicting three classification groups). The GOR I model does indeed capture more information, when measuring the number of parameters.

The parameters used in the Markov models tell something about when a given residue appears as the first residue in a subsequence (the $p(a)$ parameter) and what pairs of residues occur (the $p(a|b)$ parameter). These parameters are used to calculate the probability of a window occurring in a sequence. This method relies on the Markov assumption (see Appendix A – The Markov Assumption). It does not incorporate any scheme for the exact positions like the GOR model. The GOR model looks at the position of the residue to be predicted and the relative positions of the other residue in the window. That is the GOR model does indeed consider the location of the residues.

The windows W_1 = *BABGB* and W_2 = *BGBAB*, would result in the same score value using the basic Markov model. The reason is that the two windows produce the same pairs: *BA*, *AB*, *BG*, *GB*. The GOR I model will in this case be able to see the difference. The reason for this is that the model contains statistics based on the positions in the window and the model will therefore recognize that two residues have switched places.

Another interesting point when analyzing the parameters in the GOR I model and the Markov models is that the parameters in the Markov models are always independent of the window size used. The number of parameters for the GOR I model is dependent on the window size (previously called w). The Q_3 for the GOR I model drops whenever the window size is lowered (as seen in Figure 26), which means that the performance of the model depend on w and therefore also on the number of parameters.

The resulting remark may be that the GOR I model does indeed consider the exact positions of the residues in the local sequence (the window) while the specific pairs are considered in the Markov models. The Markov model uses the pairs to calculate the probability of the window and thereby only considers the position of the residues indirectly.

Looking at the GOR III and GOR IV models (see section 4.3.The Different GOR Models) we see that the number of parameters is increased. This model incorporates frequencies as

$f_{S_j,R_{j+m}}$, $f_{nS_j,R_{j+m}}$, $f_{S_j,R_{j+m},R_{j+n}}$, and $f_{nS_j,R_{j+m},R_{j+n}}$

The expressions are used in the definition of GOR IV, see the section 4.3.The Different GOR Models. These expressions are dependent on several positions in the local sequence. This means that the models include even more information about the exact positions of the residues in the window than GOR I. The tests show that these extensions increases performance compared to the original GOR I model.

It seems that the GOR models are in fact better to predict the secondary structures than the Markov models and the fact that the GOR models considers the exact positions of the residues may be an explanation for this.

7.5. Summary

The previous sections have presented the results for each of the treated models, both the different GOR models and the different Markov models. Summing up the results obtained using the different models we see that the GOR models are superior to the Markov models. The previous section gave a possible explanation to this.

Now focusing on the Markov models, we see that different extensions have been proposed and different results have been obtained. Using the basic Markov model with pseudo counts and a reasonable window size a prediction accuracy of 51.2% was achieved. We have seen that the window size has more influence on the general performance than the pseudo count constant.

Replacing $p(a)$ and $p(a|b)$ by introducing $p'(a)$ and $p'(a|b)$ has increased the performance (Q_3) of the model from 51% to 53%. The new score function only consisting of $p'(a)$ values and incorporating a discount factor according to the position also has a performance of 53%.

The difference, ratio-I and ratio-II methods have performances of around 53%. The momentum method almost achieved a prediction accuracy of 55%. The window size parameter has not been explored on these methods since it would require a lot of time to perform the tests. Instead the new parameter associated with the methods has been explored. However it is possible that the new parameter also depends on the window size. It is a fact that the difference-limit depends a lot on the window size (because the numerical values of the score function changes drastic when the window size is changed).

The test classification method where some of the residues were skipped according to the window size turned out to be inconclusive. It was not actually possible to conclude that the Markov model has problems predicting two or more subsequences in same window.

The normalization schemes used in this project did not increase the performance drastically (they achieved a prediction accuracy of 51.7% at their best). The combined model using both the forward and the reversed sequence direction achieved a prediction accuracy of almost 54%. Combining some of the better extensions resulted in prediction accuracies above 57%. This is an increase of more than 2% of the best individual extension. In total the basic Markov model has been improved from around 51% to around 57% (using the combined model of different extensions), an increase of 6%.

The first GOR model, GOR I, obtained a prediction accuracy of 60.5%, which is more than 3% higher than for the best Markov model scheme. It appears that the GOR models indeed capture the information in the one-dimensional protein sequences more than the implemented Markov models.

8. Implementation

This section contains an overview of the implementation. The general use of the Markov models is described and the corresponding classes are mentioned in order to get a picture of the structure of the program. The section is divided into several parts. The sections only give an overview of the implementation see Appendix E – CD and Source Code for actual details. The contents of these parts will now be explained.

Data and Data Handling describes the data and how data is managed in our program. We have two different formats for files containing sequence data. The method for reading these files into variables is described. This introduces the classes involved.

The Markov Model describes how the Markov models are implemented. The descriptions should make it possible to understand the concepts used. The concept of extending the model and how these extensions are applied in the source code are also described here.

Running Tests is the last section and contain information about how the models are tested and how the command line parameters are used to launch tests.

8.1. Data and Data Handling

This section contains information about the different data files we have used and the classes using them. In the following table the different data files are shown by their extension.

Type	**Description**
.dat	Our own data files. These files contain a list of the sequences (file names) that should be read.
.seq	Sequence data from the GOR database (all sequences in one file) [2].
.obs	Observation data from GOR database (correct classification sequences corresponding to the .seq file) [2].
.all	These files are earlier referred to as the DSSP database. Each file contains one sequence and the associated classification sequence [12]. There are 513 different files.
.csv	Microsoft Excel files (comma separated files) which are outputted by our program.

Table 29: Different file types used in this project.

Table 29 shows a brief description of each file. In the next sections the files will be described further together with the classes that use them. One of the file types (*.dat* files) is our own format. The rest are already defined.

The *.csv* file is used to output results. The results are outputted as comma separated values. This file type makes it possible to load the results obtained by running a test into Excel (or some other program able to read the csv files) and process the test data. In the following the other file types mentioned will be described.

8.1.1. Sequences and Classification Sequences

In order to train the Markov models it is necessary to load sequences and their associated correct classification sequences. The sequences used in this project are from the distribution material used in Jpred (the DSSP database) and has been downloaded from the internet [12]. Every file contains a protein sequence and the associated correct classification sequence, plus some extra information not used in this project.

These files have the extension *.all*.

```
1:      RES:Q,C,K,...
2:      DSSP:_,_,S,...
3:      DSSPACC:e,e,e,...
4:      STRIDE:C,C,C,...
5:      RsNo:4,5,6,...
6:      DEFINE:_,_,E,...
7:      align1:Q,C,K,...
8:      align2:T,S,A,...
9:      align3:R,Y,K,...
10:     align4:N,S,V,...
```

Listing 1: Listing of a *.all*-file containing a sequence and its associated correct classification sequence.

Listing 1 shows a part of a typical *.all* file. The first line contains the sequence. The keyword is "RES:" and after the colon the residues are listed separated by commas. In the second line the correct classification sequence is listed. The keyword for the classification sequence is "DSSP:" and the following structure groups are also separated by commas.

In order use the data in the *.all* file it is necessary to store the sequence and classification sequence in variables. The class *CSequence* is created for this purpose. *CSequence* can hold a sequence and the associated classification sequence. The class *CParser* is used to read the data from the *.all* file and store it in a *CSequence* class. The following diagram shows the dataflow of the sequence from raw data in *.all* file to the class *CSequence* containing the same data ready for use in the program.

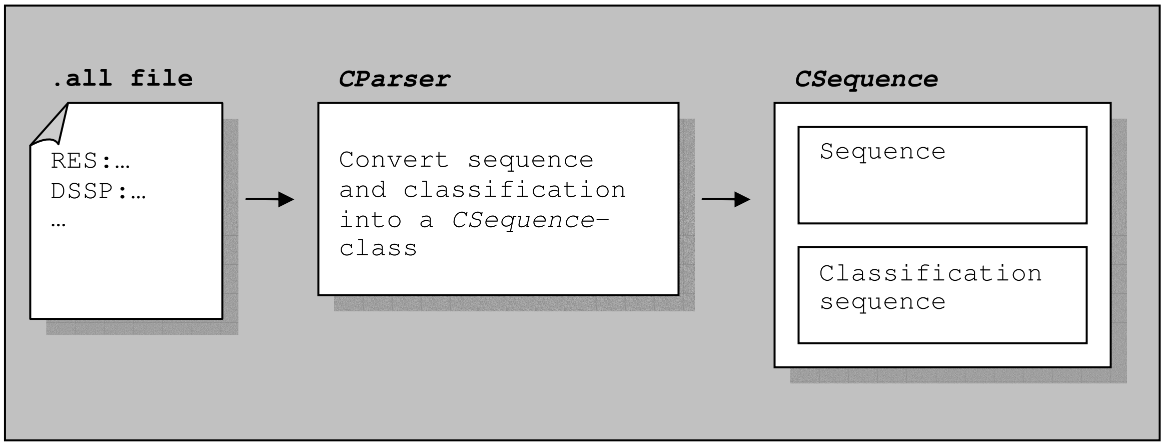


Figure 28: Dataflow from raw sequence data to the class *CSequence*.

Figure 28 shows the dataflow for only one sequence and therefore one file. In order to train the Markov models it is necessary to read all sequences (there are 513 files/sequences in the DSSP database). This is done by automating the process shown on the figure and iterating through all files. The class *CSequenceHandler* does this job.

In order to train the Markov model we also need to split the sequences up into subsequences. *CSequence* is able to supply the necessary information for this job. To split the sequence up into subsequences we need to know the sequence and the associated classification sequence. *CSequence* contains these sequences. When we iterate through all files we split every sequence up into subsequences and put them into three collections one collection for each group. Each collection of subsequences is held by the class *CSubSequenceCollection*. It is not possible to track the subsequences to their original sequences after they are extracted. Figure 29 shows the iteration through all files and how they are split into subsequences and sorted according to the different classification groups.

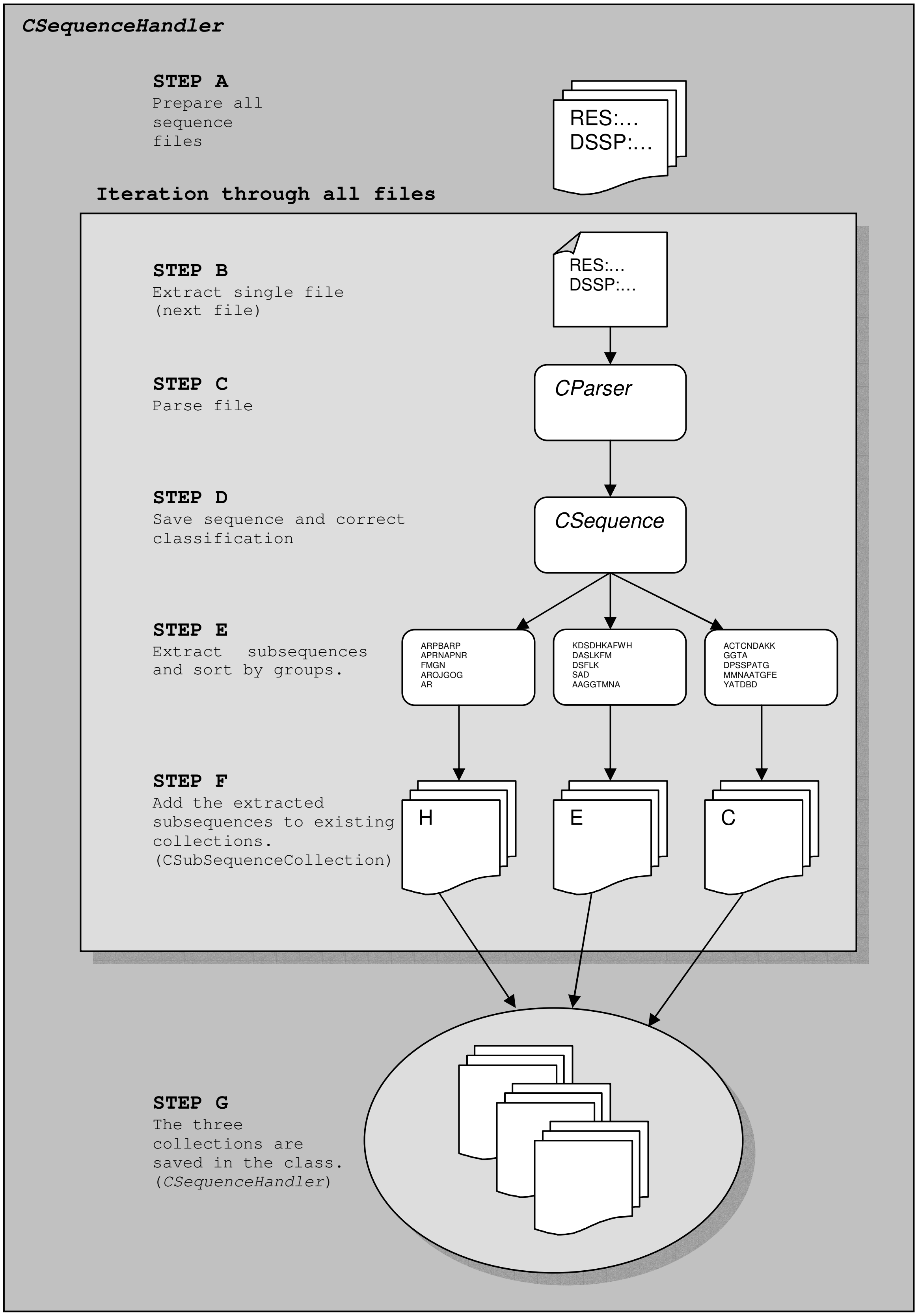

CSequenceHandler
STEP A
Prepare all sequence files
RES:…
DSSP:…
Iteration through all files
STEP B
Extract single file (next file)
RES:…
DSSP:…
STEP C
Parse file
CParser
STEP D
Save sequence and correct classification
CSequence
STEP E
Extract subsequences and sort by groups.
ARPBARP
APRNAPNR
FMGN
AROJGOG
AR
KDSDHKAFWH
DASLKFM
DSFLK
SAD
AAGGTMNA
ACTCNDAKK
GGTA
DPSSPATG
MMNAATGFE
YATDBD
STEP F
Add the extracted subsequences to existing collections. (CSubSequenceCollection)
H
E
C
STEP G
The three collections are saved in the class. (CSequenceHandler)

Figure 29: The sequence data is read by the class *CSequenceHandler*.

The procedure shown in Figure 29 is carried out by the class *CSequenceHandler*. A short description of the figure follows:

- Step A:
    - All files containing sequences are given to the class *CSequenceHandler*.
- Step B:
    - Here we enter the loop. Every iteration of the loop start by taking out a new file from the original collection. When the loop ends all files are handled.
- Step C:
    - When the file has been chosen (in Step B) it is passed on to the parser (the name of the class is *CParser*). The parser reads the data file and puts the sequence and its classification into a *CSequence* class.
- Step D:
    - The sequence and its correct classification sequence in the data file are stored in the class (*CSequence*)
- Step E:
    - Now the extracted sequences are split up into several subsequences. These subsequences are sorted by group meaning that if a subsequence is classified as a helix then it is saved in the helix subsequence collection and so on. In this way we have three collections. Each collection contains a number of subsequences which are all taken from the sequence. In this way the original sequence and classification is converted into collections of subsequences. The collections are temporary.
- Step F:
    - The temporary collections are added to permanent collections. Now we go to step B and continue as long as there are files left to read.
- Step G:
    - When all files are read the permanent collections of subsequences (now taken from all sequences from all files) are stored in the *CSequenceHandler* class. Now *CSequenceHandler* contains the necessary subsequences to train the Markov model.

The dataflow for reading the sequence data has now been presented. In the next section we explain the use of other files and formats.

8.1.2. Other Files

We have previously showed a table (Table 29) of the different file types used in this project. The previous section explained the *.all* file. This section will explain the *.seq* and *.obs* formats used by the provided GOR database [2]. The *.dat* file will also be explained which is our own format that contains a list of sequences that will be used in the training. The sequences (given by the filenames) in the *.dat*-file are also referred to as *datasets*.

8.1.3. The GOR Database

The previous section explained the file format of *.all* files which contains sequence data. We have also obtained sequence data in another format, namely the data used in the GOR database (provided by the authors of [2]). These files have extensions *.seq* and *.obs*. In order to read these sequences they are converted into *.all* files. The conversion is carried out by *CParser* class.

The GOR data consist of only two files. One has extension *.seq* and contains all sequences. The other file has extension *.obs* and contains the correct classifications for the corresponding sequences in the *.seq*-file.

Listing 2 shows a sample of a *.seq* file.

```
1:      !1AAJa APOAMICYANIN                     ELECTRON TRANSPORT            105
2:      DKATIPSESPFAAAEVADGAIVVDIAKMKYETPELHVKVGDTVTWINREA
3:      MPHNVHFVAGVLGEAALKGPMMKKEQAYSLTFTEAGTYDYHCTPHPFMRG
4:      KVVVE@
5:      !1AAKa UBIQUITIN CONJUGATING ENZYME   PRELIMINARY                   150
6:      MSTPARKRLMRDFKRLQQDPPAGISGAPQDNNIMLWNAVIFGPDDTPWDG
7:      GTFKLSLQFSEDYPNKPPTVRFVSRMFHPNIYADGSICLDILQNQWSPIY
8:      DVAAILTSIQSLLCDPNPNSPANSEAARMYSESKREYNRRVRDVVEQSWT
9:      AD@
```

Listing 2: Listing of a *.seq* file from GOR data.

First line in the *.seq* file starts with an exclamation mark, some sort of ID, the protein name and some description. In the end of the first line there is a number indicating how many residues the sequences consist of. In Listing 2 the first protein is *Apoamicyanin* and consists of 105 residues. The second line and the following 105 letters contain the residue in the sequence. The sequence is terminated using the '@' character. The first sequence is terminated in line four where the first '@' character is.

The classification sequences found in the *.obs* file is using the same format (see Listing 3).

```
1:      !1AAJa APOAMICYANIN                     ELECTRON TRANSPORT            105
2:      CCEECCCCCEECCCCCCCCCEEEEEECCEECCCEEEECCCCEEEEEECCC
3:      CCCCCEECCCCCCCCCEECCCCCCCEEEEEEECCCEEEEEEECCEEEEEE
4:      EEEEC@
5:      !1AAKa UBIQUITIN CONJUGATING ENZYME   PRELIMINARY                   150
6:      CCCCCCHHHHHHHHHHCCCCCCEEEEEEECEEEEEEEEEEECCCCCCCCC
7:      CEEEEEEECCCCCCCCCEEEEECCCCCCCCCCCCCCCCHHHHCCCCCCCC
8:      CHHHHHHHHHHHHHCCCCCCCCCHHHHHHHHHCHHHHHHHHHHHHHHCCC
9:      XX@
```

Listing 3: Listing of a *.obs* file from GOR data.

To convert the *.seq* and *.obs* files into several *.all* files we first of all need to know what name we will call every *.all* file (because we need one file for each sequence). Since the data does not contain two proteins with the same ID we use this as the unique file name. This means that a new file is created for every sequence parsed. Listing 4 shows the result of converting the first sequence in the GOR-data (from Listing 2 and Listing 3) to *.all*-file format:

```
1:      RES:D,K,A,T,I,P,S,E,S,P,F,A,A,A,E,V,A,D,G,A,I,V,V,D,I,…
2:      DSSP:C,C,E,E,C,C,C,C,C,C,E,E,C,C,C,C,C,C,C,C,E,E,E,E,E,…
```

Listing 4: File (*.all*) after GOR data is converted.

The filename of the new *.all* file is *1AAJa.all*. After the conversion the file (*1AAJa.all*) contains the residues from the first sequence in the *.seq* file. Each residue is now separated by a comma. The sequences are marked by the keyword "RES:". The second line is the classification sequence marked with the keyword "DSSP:" and the structure classes following according to the standard used in *.all*-files.

8.1.4. Sequence File List

The class *CSequenceHandler* has to know which files that contain the sequence data. This information is stored in a *.dat* file. A *.dat* file is a simple file that contains names of other files. These filenames are the names of the corresponding files containing the sequences and the correct classification sequences (*.all* files). It is possible to use comments in the .dat files by letting the first character of the line be ';'. A sample of a .dat file can be seen in Listing 5.

```
;
;This is a test .dat-file
;
1alkb-1-AS.all
1cbg-1-AS.all
1celb-1-AUTO.1.all
1lap.all
1qbb-3-AUTO.1.all
1reqc-2-AS.all
1rlr-2-JAC.all
1trh-1-AS.all
1tsp-1-AS.all
1vnc-1-JAC.all
2glsa.all
4gr1.all
6cpp.all
6cts.all
7cata.all
7icd.all
```

Listing 5: Example of a *.dat* file. The *.dat* file contains sequence-filenames.

If we use a .dat file like the one shown in Listing 5 we will train the Markov model using the sequences contained by the 16 *.all* files. The training procedure (the jackknife method) used in this project demands that it is possible to leave one of these file out and then train the Markov models using the remaining *.all* files. This is possible using the class *CDataHandler*. This class reads a *.dat* file and stores the information in the .dat file in a vector. The class can return how many filenames it contains and it can return the filename at index *i*. When we use this class it is possible to control which files that are added to the *CSequenceHandler* object. This means that it is possible to control which files are used to train the Markov models. The relation between *CSequenceHandler* and the Markov models is shown later in the section 8.2.The Markov Model - General.

8.1.5. Summary of Data and Data Handling

In the section called 8.1.Data and Data Handling the dataflow in our program is outlined. Listing 6 contains a brief description of how the sequences are read. Each step has been described in more details in the previous section. Listing 6 shows how sequences are loaded from *.all* files into variables in *CSequenceHandler*.

```
CDataHandler loads .dat file containing .all sequence-files
Using CDataHandler the necessary sequence files are added to CSequenceHandler
CSequenceHandler splits all sequences up into subsequences
       This is done by using:
              CParser to parse a single file
              CSequence to contain the sequence that is parsed and retrieve subsequences
              CSubsequenceCollection to hold the extracted subsequences
```

Listing 6: Scheme shows how to load sequences from files into the class *CSequenceHandler*.

The next section will give an overview of how the Markov models are implemented. In the section *General* we show how the Markov models interact with the data handling part described in this section.

8.2. The Markov Model

This section presents an overview of the implementation of the Markov model. The basic functionality is presented first. This includes estimating the model parameters and using these parameters to return the value of the score function.

After the general introduction the basic model is presented. This model is able to generate a classification sequence when a sequence is inputted.

The last sections explain how different extensions to the model are applied.

8.2.1. General

The class *CMarkovModel* contains the basic operations and parameters in the Markov model. The class contains the parameters $p(a)$ and $p(a|b)$ for each of the three Markov models (H,E and C). The parameters are estimated based on the subsequences supplied by *CSequenceHandler* (described in previous section). Figure 30 shows the classes used by *CMarkovModel* to retrieve the sequence data.

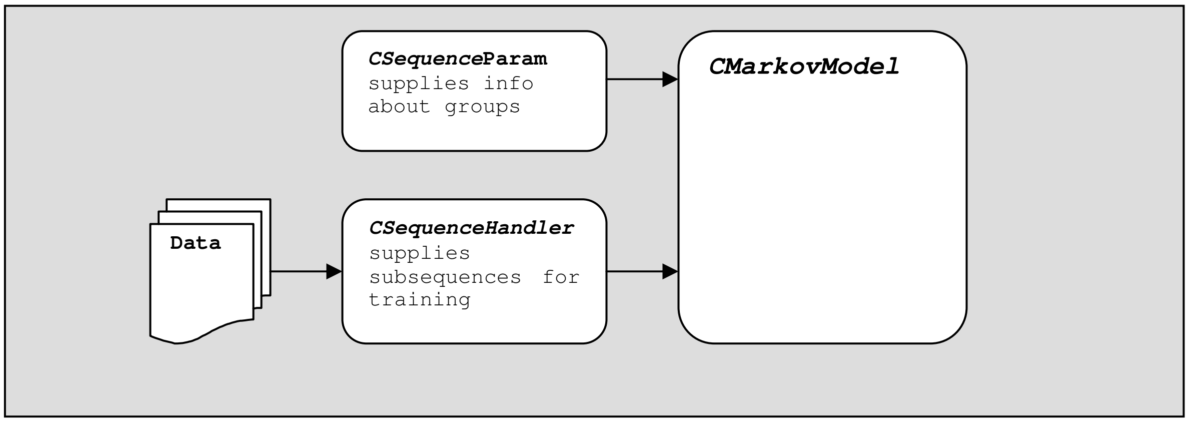

CSequenceParam supplies info about groups
CMarkovModel
Data
CSequenceHandler supplies subsequences for training

Figure 30: Classes used by *CMarkovModel* to retrieve sequences.

Figure 30 shows that *CSequenceHandler* uses the data files to extract the subsequences for the three groups. *CMarkovModel* uses the subsequences in *CSequenceHandler* to estimate the parameters defining the Markov model. The class *CSequenceParam* supplies information about the groups. This class is initialized by the user. The user stores which groups the sequences contains and includes an identifier for these groups. The identifier is used to retrieve information about subsequences later. The identifier is necessary for the *CMarkovModel* object to extract the subsequences belonging to some classification group.

CMarkovModel contains 3 sets of parameters. One set for every model. This is shown in Figure 31.

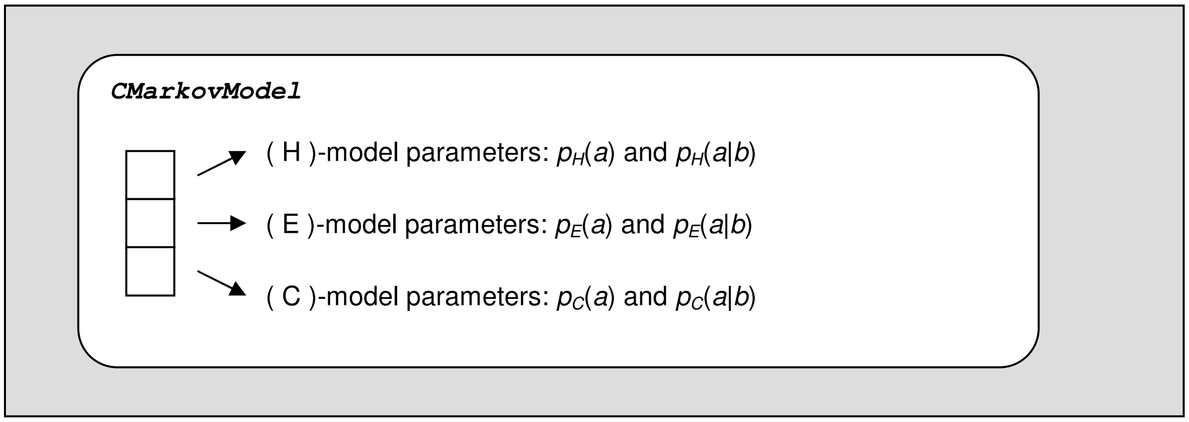

CMarkovModel
(H)-model parameters: $p_H(a)$ and $p_H(a|b)$
(E)-model parameters: $p_E(a)$ and $p_E(a|b)$
(C)-model parameters: $p_C(a)$ and $p_C(a|b)$

Figure 31: The structure of *CMarkovModel*

Figure 31 shows that *CMarkovModel* contains the parameters for all three models. The parameters for each model are estimated based on the input from *CSequenceHandler*. The function in *CMarkovModel* that estimates the parameters are called *calc*(…). This function takes the subsequences and the group information as input. Subsequences are supplied by *CSequenceHandler* and the group information is supplied by *CSequenceParam*.

CMarkovModel serve as a base class for the models used in this project. These models are explained in the following sections. The basic Markov model class, *CMarkovModelBasic*, extends this class. *CMarkovModelBasic* contains a function called *GetClassificationRate*(…) which takes a sequence and information about the groups as input. This function will classify a single sequence based on the parameters estimated using a given dataset. This will all be explained in the next section.

To summarize this section the following list is shown. The list shows the steps for estimating the parameters for the three Markov models (representing the groups H, E and C):

1) Load subsequences into *CSequenceHandler*.
2) Call *calc()* in *CMarkovModel* using subsequences and group information as input.
3) The estimated parameters are stored in *CMarkovModel*.

When the parameters are estimated for each model it is possible to calculate the score functions for each model and classify a sequence. This is explained in the following section.

8.2.2. Basic Implementation

CMarkovModelBasic extends the *CMarkovModel* class. The base class *CMarkovModel* contains the estimated parameters and a function that returns the value of the score function. This function takes a group and a window of residues as input. The return value is the value of the score function associated with the group. *CMarkovModelBasic*'s job is to compare the score functions for every model (H, E and C) and classify each residue according to the winning score function. Figure 32 shows this process.

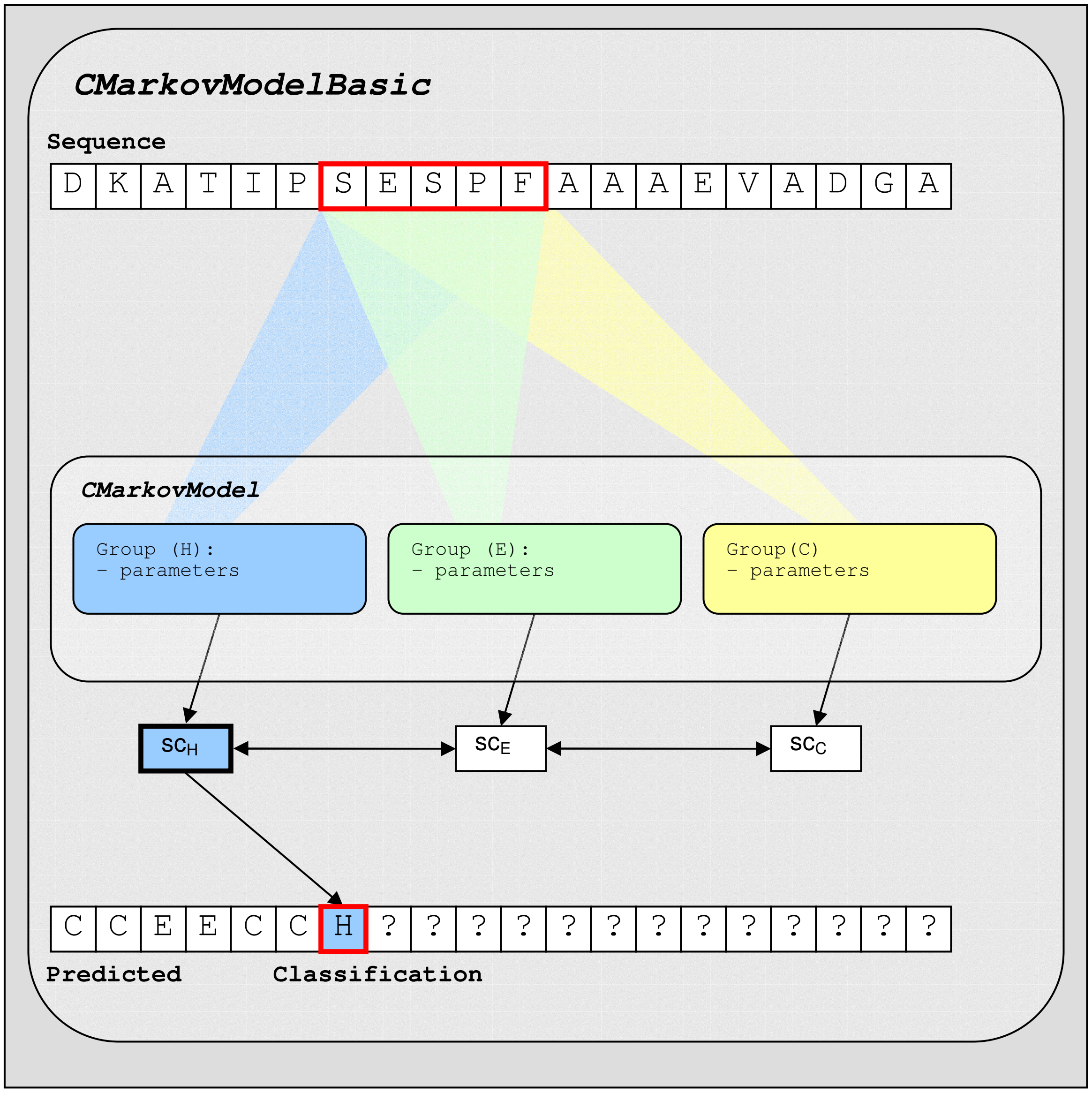


Figure 32: Markov models used for classifying a sequence

Figure 32 shows that the class *CMarkovModelBasic* has extended the functionality of *CMarkovModel*. *CMarkovModelBasic* uses the parameters from each Markov model to calculate the score functions. This is actually done in *CMarkovModel* that *CMarkovModelBasic* extends. The score functions are compared and the residue is predicted. The window marked with a red rectangle in the top of the figure is used as input for the score functions. As mentioned earlier *CMarkovModel* has a function that returns the value of the score function given the group and a window of residues. This function is used by *CMarkovModelBasic* to retrieve the values of the three score functions. *CMarkovModelBasic* contains the input sequence and its classification sequence.

The window of residues and the current position in the sequence and classification sequence is managed by *CMarkovModelBasic*. These positions are shifted by one to the right whenever a prediction has been made (this process is described in section 3.Using Markov Models for Classification).

The prediction iteration (predicting the residue and shifting the window) is performed until the end of the window has reached the end of the sequence. *CMarkovModelBasic* records the number of correctly predicted residues and the total number of observed residues (residues predicted). These numbers are used to calculate the prediction accuracy, Q_3. The numbers are saved as public variables in the *CMarkovModel* class and can be accessed by the user of the model (this makes it possible to store the Q_3 values for further data analysis).

8.2.3. Extensions to the Basic Model

The following sections describe the classes involved when the Markov model is extended. The structure of the sections is the same as earlier in the report. First extensions based on the individual terms in the score function are presented. After that the extensions based on the overall score function is presented. In the last part of the section the normalized and combined models are presented.

Every section will give an outline of where the actual changes are carried out and information about how it is done. The section ends with an overview of the stages that is visited during the use of a Markov model. The stages where the different extensions are applied are recalled here.

8.2.4. Extensions Based On the Individual Terms in the Score Function

The extensions based on the individual terms are made in *CMarkovModel*. The following figure (Figure 33) shows where these changes are applied:

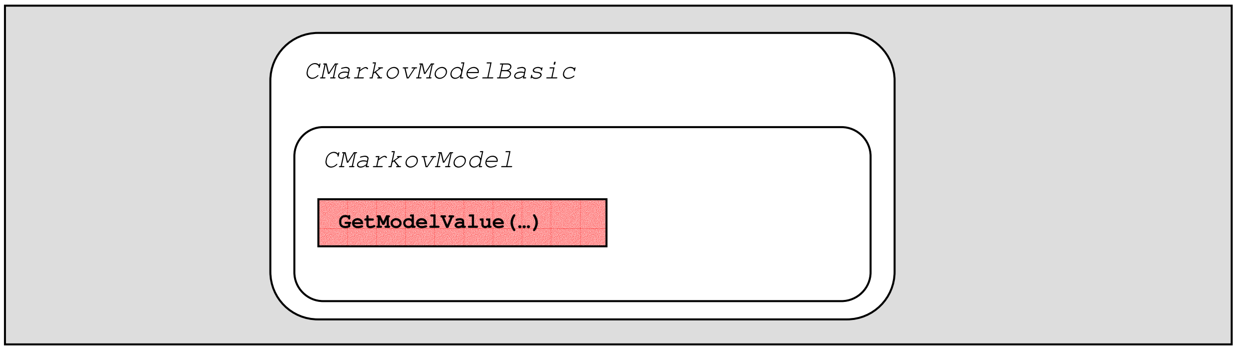


Figure 33: Extensions to the terms in the score function are made inside *CMarkovModel* more specific in a function called *GetModelValue*(…).

Figure 33 shows that the score function and its terms can be accessed in the function called *GetModelValue*. This function is inside the *CMarkovModel* class. *CMarkovModel* contains a number of private variables that affects the way that the score function is calculated. The variables in *CMarkovModel* that affect the score function are:

Variable	Description
`m_bUseSingleResidueTerm`	If true the $p(a)$ term is used in the score function, else its neglected
`m_bUsePairResidueTerm`	If true the $p(a\|b)$ terms are used in the score function, if false the terms are neglected.
`m_bAlternativeModelValue`	If true the alternative score function is used where we only use $p'(a)$ terms. If false the usual score function is used.
`m_bAlternativeModelValue_simpleversion`	If true the $p'(a)$ terms does not use the discounting factor. This variable only has an effect if `m_bAlternativeModelValue` is true.

Table 30: Boolean variables activating different code blocks.

The variables in Table 30 can be set by functions in *CMarkovModel*. This means that if *CMarkovModelBasic* needs to use a certain method for calculating the score function it needs to call the setup functions in *CMarkovModel* before *GetModelValue()* is called. The setup function will set one of the Boolean values in the table to true depending on which setup function is called. This method of enabling and disabling extensions is used in general.

8.2.5. Extensions Based On the Overall Score Function

Extensions based on the overall score function are made in *CMarkovModelBasic*. These extensions are typically applied after the value of the score function is retrieved. Some extensions alter the returned value (the momentum method), others alter the way that we find the winning score function (decision constants). Figure 34 shows an overview of *CMarkovModelBasic*.

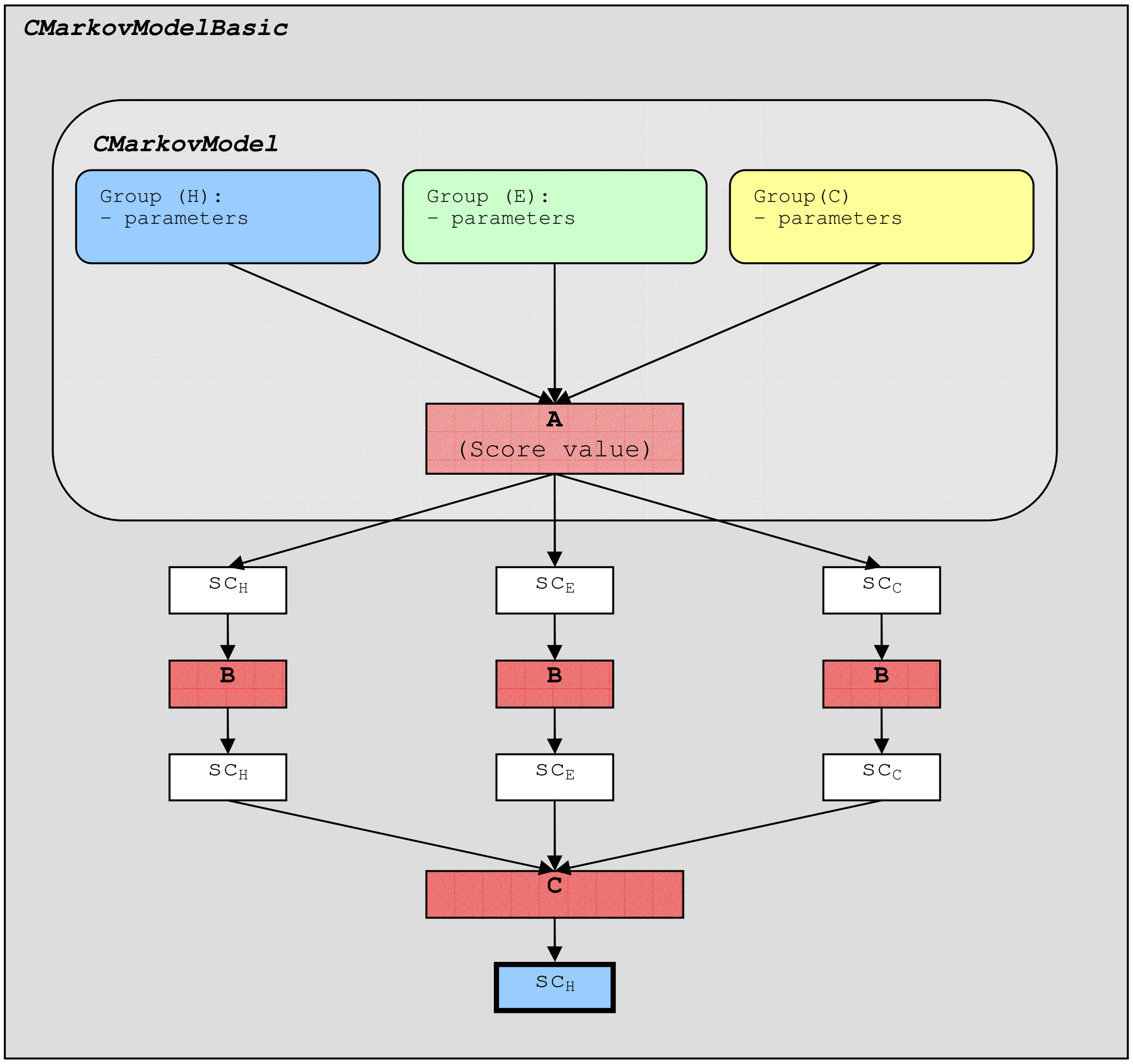


Figure 34: *CMarkovModelBasic* with extensions. The red boxes indicate where an extension has been applied in the source code.

Figure 34 shows a part of *CMarkovModelBasic*. It is the figure presented earlier but the extensions have now been filled in. Compared to the figure presented earlier the protein sequence and the correct classification sequence have been omitted.

The figure has a number of red boxes. These boxes represent the places where the extensions have been applied. The first extension (box A – the score function) is placed in *CMarkovModel*. This box applies extensions to the individual terms in the score function. The boxes marked with B apply changes to the overall score function. The box marked with C represents the rule for determining the winning score function (possibly by using decision constants).

Inside each box in the figure there are blocks of code. These blocks can be enabled and disabled. The code blocks are extensions to the model. If the blocks are disabled no extensions are used. In this way it is possible to apply different methods by activating the code blocks. It is possible to combine methods from different boxes. This is also explained later.

Adding noise, decision constants and ratio-II methods
These methods add restrictions for classifying a residue as a new group (the box marked with C in Figure 34). The noise method works differently but the code needed for this method is placed at the same location in the source code. A new comparison block is added if we enable some of the methods just mentioned. This means that new rules for picking a group for the current residue are used. When the three Markov models have returned the value of their score function the new comparison block will make the decision about the prediction. The standard method is to compare all values and let the winning model predict the group. The new rules are described separately in the section 3.5.2.Extensions Based On the Overall Score Function. The new comparison code is activated through Boolean variables which are declared in *CMarkovModel*. Whenever the user of the class (*CMarkovModelBasic*) wants a certain extension enabled calling a simple function will do the trick. The function will enable the Boolean variable and therefore the comparison block needed in order to active the necessary rule.

Momentum
Momentum alters the value of the score function (the boxes marked with B in Figure 34). This block of code is activated just after the score function values are retrieved. The previous values of the score functions are saved in an array and make it possible to retrieve the last value of the score function (step *i*-1 if the current step is *i*). It is therefore possible to apply the momentum methods. The values of the score functions are altered and used as usual in *CMarkovModelBasic*. This means that after the score values are altered they are compared as usual.

It is actually possible to enable both momentum and for instance the ratio-II method. The reason is that the code is independent of each other and is placed in two different places in the source code. The momentum code is added after retrieving the score function values and the ratio-II rule is applied after the values has been found for all models.

8.2.6. Extensions Based On Alternating the Training Procedure

These extensions are made in *CSequenceHandler* and *CSequence*. As mentioned earlier *CSequenceHandler* reads the sequences and the correct classification sequences and stores them in variables. *CSequence* is able to extract all subsequences in a single sequence. These two classes are therefore central when the training process is changed.

The altered training procedure described in this report is about appending extra residues to every subsequence that is extracted. The place to do this is in *CSequence* because it is the class that extracts the subsequences. *CSequenceHandler* is the class that uses *CSequence* so this class forwards the information about altering the training method to *CSequence*. Figure 35 shows how *CSequence* extracts subsequences normally.

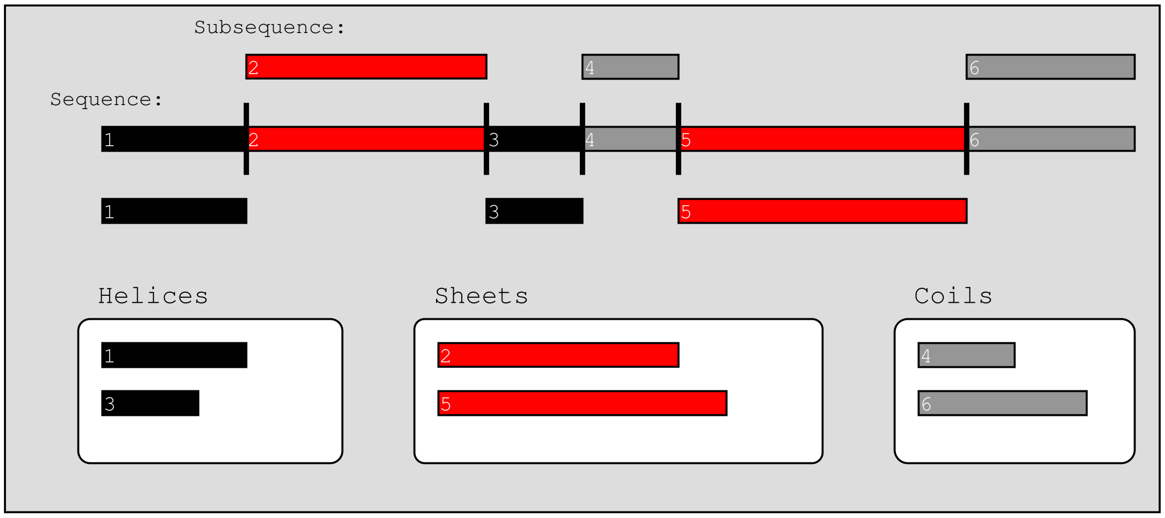


Figure 35: Normal subsequence extraction.

Figure 36 shows how extra residues are appended on every subsequence.

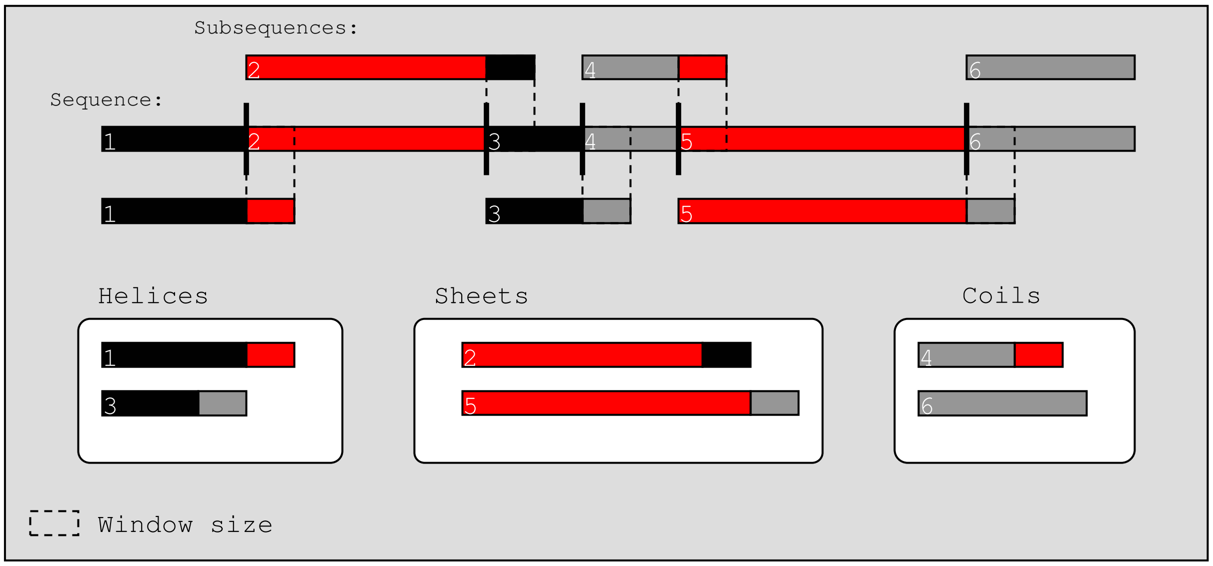


Figure 36: Alternative training method by concatenating residues to the extracted subsequences.

To implement this feature in *CSequence* a new block of code is inserted and is activated by a Boolean variable. The new code simply adds the next *l*-1 residues to the subsequences where *l* is the window size. If it is the last subsequence no extra residues are added. To enable the new code block the user calls *EnableExtendSubsequence(true)* in *CSequenceHandler*. *CSequenceHandler* will forward the information to *CSequence* and the training method will be altered because the new code block is activated in *CSequence*.

8.2.7. Normalized Models

The normalization methods extend *CMarkovModel* and works different than *CMarkovModelBasic*. In order to do some sort of normalization several score values for different window sizes are retrieved. These score values are *normalized* meaning that they are manipulated so they can be directly combined (without the normalization it would not make any sense to sum the values directly).

The first example of a normalizing model is the basic normalization model. This model scales (as described earlier) the score values obtained using different window sizes. This method makes it possible to add up the score values from the different window sizes. When the summed score values are calculated the result is replaced with the usual value of the score function associated with the group. From this point the model works as *CMarkovModelBasic*. Figure 37 illustrates where the normalization takes place (the red dotted rectangle).

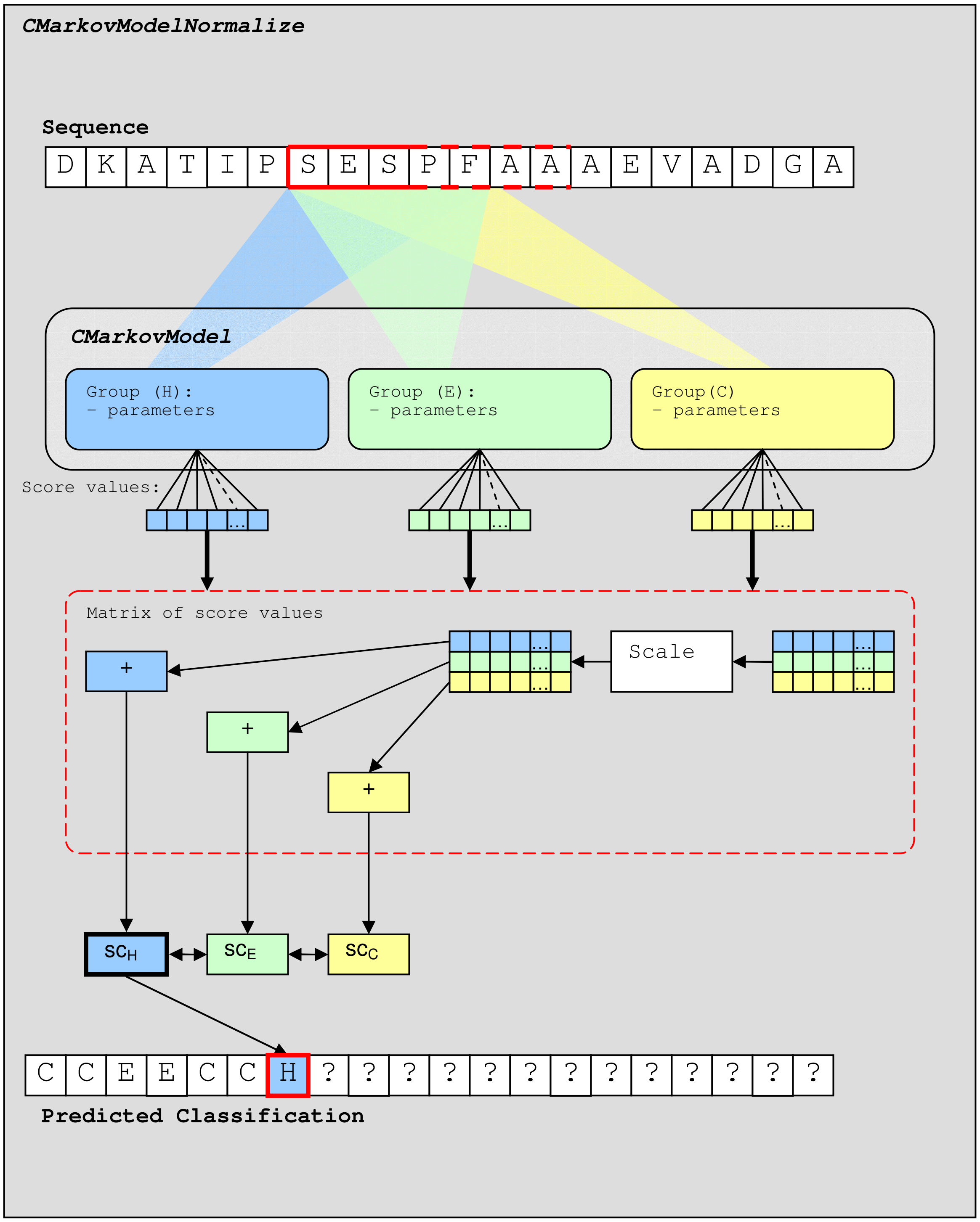


Figure 37: The structure of *CMarkovModelNormalize* that is able to normalize score values.

The figure shows that several different window sizes are fed into the Markov models. This produces a score value for each window size. This means that we have an array of score values outputted for every Markov model. The normalization is shown inside the red dotted rectangle.

In the red dotted rectangle we see that the three arrays of score values are representing a matrix. Each column in this matrix is scaled according to the description in section 3.6.1.Basic Normalization. After the scaling the rows are summed up and the three results are replaced as the usual score values. From here the model works as *CMarkovModelBasic*.

The normalization takes place between getting the score values and comparing these values. The normalization model retrieves several score values (because several window sizes are used) where *CMarkovModelBasic* only retrieves one score value (for each classification group). The normalization method works almost as the *CMarkovModelBasic* because an array of score values is summed up and only one score value is used in the end (like *CMarkovModelBasic*). The fact that *CMarkovModelNormalize* is very similar to *CMarkovModelBasic* makes it possible to apply the extensions presented in 3.5.Extensions to the Basic Markov Model such as the momentum method and the decision constants to the normalization model.

8.2.8. Combining Methods and Models

We now see that it is possible to combine some of the different methods presented earlier in the report. The reason is that the methods are applied in different stages of predicting the final group for a residue. The training stage is the first stage to apply the new code in. After this we retrieve the value of the score functions. Here we have different approaches of altering the score values as describe earlier. Right after this part we can apply the normalization and end up with the usual three score values (one for each model). As the last stage we have different ways of comparing these values.

The building blocks of the extensions for the Markov model are shown in Figure 38.

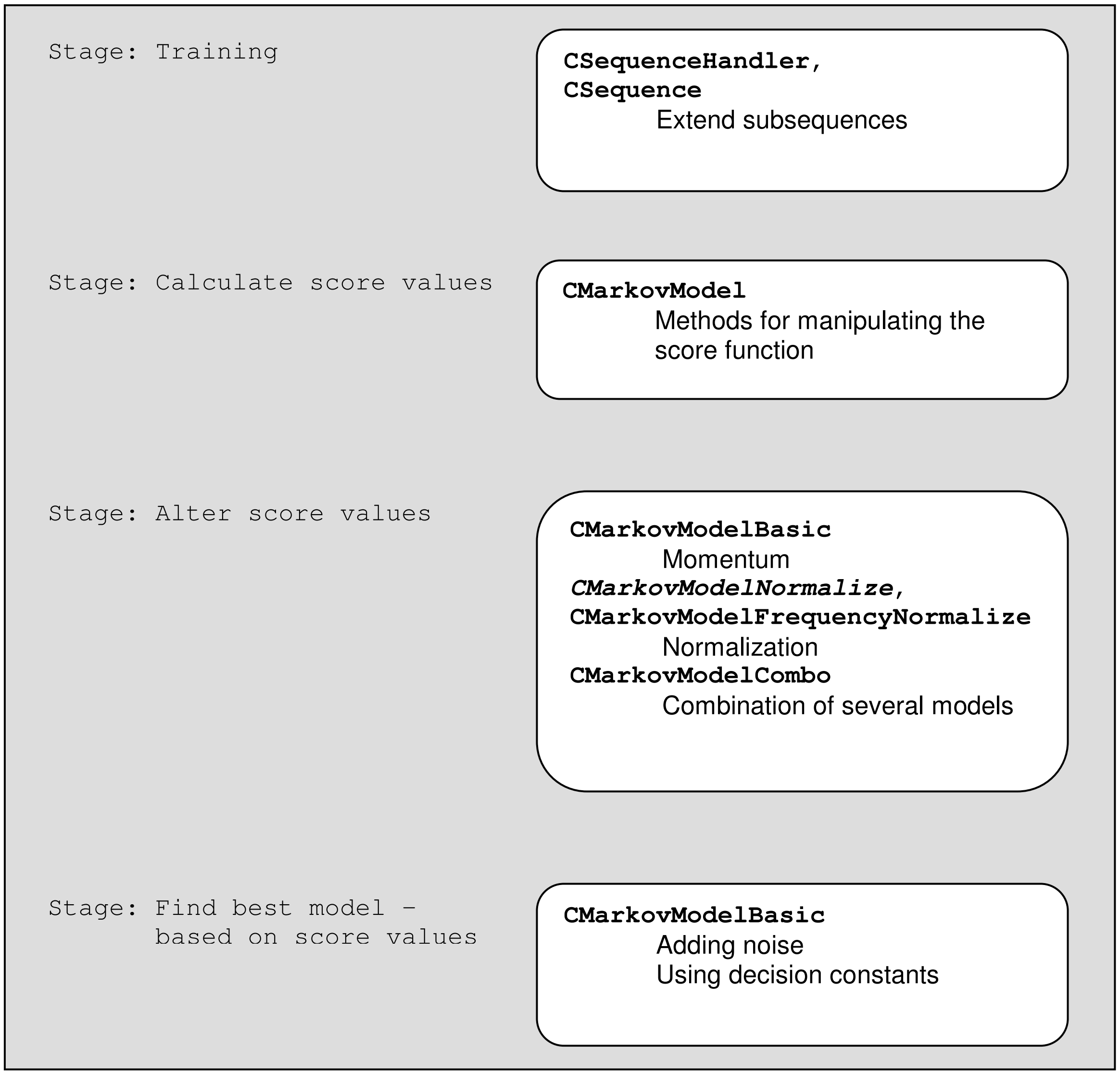

Stage: Training
CSequenceHandler, CSequence
Extend subsequences
Stage: Calculate score values
CMarkovModel
Methods for manipulating the score function
Stage: Alter score values
CMarkovModelBasic
Momentum
CMarkovModelNormalize, CMarkovModelFrequencyNormalize
Normalization
CMarkovModelCombo
Combination of several models
Stage: Find best model – based on score values
CMarkovModelBasic
Adding noise
Using decision constants

Figure 38: Figure of different stages in the process of using a Markov model and where the different methods of extending the Markov model are applied.

The figure shows the different stages of using a Markov model from the training of the model to predicting a sequence. In every box associated with a stage the affected classes and methods used are shown. When a new combined model is created it may consist of one method from each box (each stage) in Figure 38.

8.3. Tests classes

To be able to test the Markov models used in this project we have implemented several test classes. Every test class extends the base test class *CTestSettings*. *CTestSettings* contains the default test settings such as where to find the data files, the name of the csv output file and so on. *CTestSettings* also supply functions for setting these parameters. The test class that extends *CTestSettings* will therefore be flexible because it contains the test variables and possibility to change these by calling appropriate functions.

The class *CMarkovModelBasic* has an associated test class. This class is called *CTestBasicMarkovModel*. The test class uses the jackknife training method. The sequences used for training are given as a *.dat* file.

A single default test consists of a loop through each value of the window size and the pseudo count constant (meaning that these variables are altered and the Markov model is trained and tested using every possible combination of these parameters). The intervals for the parameters are set by the user, when calling the appropriate functions in *CTestBasicMarkovModel*.

To be able to use the extensions such as the momentum method or the decision constants it is necessary to enable a new loop. This is possible by calling the function *EnableExtraParameter*(…). This function will activate a new loop in addition to the two existing ones. The new loop will be used as the new parameter depending on the method used. If the momentum method is used the weight will be the loop parameter (called *extra_parameter* in Listing 7). If the difference method is used the difference-limit is the new parameter and so on. Enabling this extra loop will of cause make the test bigger.

The following pseudo code in Listing 7 will show how the test works.

```
Loop through a number of window sizes, l
   Loop through a number of pseudo count values, c
          Loop through a number of different values, extra_parameter
                 Use jackknife
                        Train a Markov model using a set of sequences

                        Test the model using jackknife method. (The variables l, c
                        and the extra_parameter are used when the Markov model
                        predicts the sequence.)
                 End of jackknife
                 Output results of jackknife procedure.
          End of loop extra_parameter
   End of loop c
End of loop l
```

Listing 7: Testing the basic Markov model.

Listing 7 shows the three loops iterating through the parameters that should be explored. For every iteration of the inner loop the results obtained are logged into a *.csv* file. This makes it possible to produces graphs and other useful test analysis after a test run.

In order to make it easier to test methods without changing the code the possibility of adding command line parameters has been implemented. This will be described in the following section.

8.4. Running Tests Using Command Line Parameters

Listing 8 shows the output when parameter `-h` is given.

```
Syntax for CTest
--------------------------
CTest <test type> <args>

<testtype>
  basicmodel                         use the simple basic Markov model
  combo                              combine orientation
  basicnorm                          basic normalization
  freqnorm                           frequency normalization
  gor                                gor model

type CTest <test type> -h for more help
```

Listing 8: Syntax of the command line utility (printed using parameter `-h`).

In Listing 8 we see that it is possible to launch 5 different test types using the command line utility. Each test type has some parameters associated. These parameters can be seen by typing:

```
java CTest <test type>
```

where the test type can be `basicmodel`, `combo`, `basicnorm`, `freqnorm` or `gor`. Listing 8 shows what the different test types mean. Each test type launches a different test class.

We start out by showing the arguments for the test type `basicmodel` (see Listing 9). The following text serves as an introduction of how to use the help function in the command line utility and to get a feeling of how different tests can be performed. In the last part we refer to Appendix D – Command Line Utility for examples of how to run tests using the command line utility.

Listing 9 shows the arguments that can be used with the `basicmodel` test type. The listing is the output we receive when the command: `java CTest basicmodel -h` is written.

```
Syntax for java CTest
--------------------------
java CTest <test type> <args>

<testtype>
  basicmodel                        use the simple basic Markov model
  combo                             combine orientation
  basicnorm                         basic normalization
  freqnorm                          frequency normalization
  gor                               gor model

<args> for basicmodel
  xp:<X1,X2,X3>                     extra param from X1 to X2 and stepsize X3
  i:<X1,X2,X3>                      pseudo count from X1 to X2 and stepsize X3
  l:<X1,X2>                         length from X1 to X2 (stepsize=1)
  csv:<filename>                    set excel output file
  datafile:<filename>               set data file name
  datapath:<path>                   set data path

  Following arguments are enabled with prefix + and disabled with prefix -
  <+|->sr                           single residue term (+)
  <+|->pr                           pair residue term (+)
  <+|->sr                           swap residues (-)
  <+|->am                           alternative model value (-)
  <+|->xs                           extend subsequence (-)
  <+|->sv                           simple version for model value (-)
  <+|->cu                           cut ends (classification) (-)
  <+|->dd                           decision difference (-)
  <+|->dr                           decision ratio-I (-)
  <+|->no                           noise (-)
  <+|->r2                           decision ratio-II (-)
  <+|->pm                           primitive momentum (-)
  <+|->ft                           first term method (-)
  <+|->mo                           momentum (-)
  <+|->rp                           reverse pair (-)
  <+|->d1                           sub: normal method (-)
  <+|->d2                           sub: reverse method (-)
  <+|->d3                           sub: Both method (-)
  <+|->s1                           direction: Forward (-)
  <+|->s2                           direction: Backwards (-)
  <+|->pc                           use pseudo count constant (-)
  <+|->gg                           use GOR groups (-)
```

Listing 9: Arguments for the test type basicmodel (printed using `java CTest basicmodel-h`).

In Listing 9 we see that we have the possibility to use many arguments for the test type `basicmodel`. Most of the arguments shown will extend the model in some way. We will shortly explain the arguments that are not self-explanatory and which arguments that have special relations. All arguments will not be explained because most arguments should be self-explanatory (when the user has read the report). As we see in Listing 9 the explanation for each argument ends with a set of parentheses. These parentheses represent the default value for the argument. If there is a '-' then the argument is disabled if there is a '+' then the argument is enabled.

The single residue term and the pair residue terms are enabled by default. This means that the score function works as the original definition. The '`i:`' argument iterates over a set of pseudo count values. If the '`i:`' argument is used the pseudo count is automatically enabled and the switch `+pc` is not needed. If the GOR database is to be used it is necessary to set the data file and data path using '`datafile:`' and '`datapath:`' and enable GOR groups with the argument `+gg`. (because the GOR database requires another mapping than the DSSP database)

To see the syntax for the other test types (other than `basicmodel`) write:

```
java CTest <test type> -h
```

where `<test type>` is the test type for which the arguments are shown.

Appendix D – Command Line Utility shows how to run some of the test cases (from the test section) using the command line – the examples should make it more clear how the command line arguments are actually used.

Appendix E – CD and Source Code shows how the source code is packed in a jar file, which may be used directly (without compiling the source files).

Discussion

The general goal in secondary structure prediction is to be able to classify protein sequences with such precision that some of the functional properties of proteins may be interpreted from the predicted classification sequences.

In this project Markov models have been used to develop several classifiers able to predict the secondary structure of protein sequences to some degree (having some precision). Markov models are chosen for this, hoping that they are able to infer the secondary structure of the proteins from the protein sequences only.

The definition of the simple Markov model proposed in the beginning of the report has been analyzed in details. The different parts of the model have been modified (and/or extended) and the performances of the new models have been obtained experimentally running tests. Each modification has been based on observations from other tests or on ideas originating from different analyses.

In the previous sections several different modifications to the basic Markov model has been proposed. Some of the modifications turned out to be more successful than others. In general the modifications have shown how the Markov models work and what problems there may be using these models for this specific classification problem.

Using the basic Markov model it was possible to get a prediction accuracy of 51.2% when optimizing the two parameters, the window size and the pseudo count constant. Using this Q_3 as a benchmark value when testing new modifications of the model, it was possible to quickly determine if a new extension of the basic model was better or worse.

When analyzing the basic Markov model it became apparent that this model has difficulties in certain situations. The basic model (in its original definition) appears to have problems classifying residues not being the first residue in a subsequence. This was discovered investigating the parameters defining the models. It seemed that a new definition of the $p(a)$ parameter solved some of these problems. Another problem is that the model has difficulties whenever the window of residues overlaps two or more subsequences. During the training of the models, the subsequences are extracted for each individual group, which means that the models do not actually have any knowledge about the boundaries of these subsequences (they are not trained on overlapping sections). This problem was dealt with in several ways, however no solution has been found. Some of the suggested extensions did increase the performance of the models meaning that some of the problems may have been removed.

Using the single extensions to the basic Markov model the prediction accuracy has been increased from around 51% to around 55% (an increase of 4%). The best single extension is the momentum method for which the best overall Q_3 is 54.8%. This method was introduced to incorporate memory of the previous predictions into the current one. The basis for this extension was found in analyses of single test sequences and the output from the individual score functions (for every model M_H, M_E and M_C). It seemed that the returned score value for the correct group was high at the beginning of a subsequence, but then decreased immediately after. The momentum method is a way to let the prediction process be dependent on the previous steps (predicting the previous residues in the sequence).

Tests have shown that it is possible to obtain higher prediction accuracies when combining some of the single extensions suggested. Combining some of the better combinations resulted in a prediction accuracy of 57.2%, meaning that it actually makes sense to combine different extensions. Suppose two extensions are combined which are in theory different from each other. The models using either of the extensions are improved which means that there is a good chance that the combined model may be improved as well.

Implementing the combinations of some of the extensions did increase the performance of the models but the prediction accuracies obtained were still around 3% lower than the Q_3 for the first GOR model.

The GOR I model yields a Q_3 of 60.5% using our implementation of the model. The GOR model is a statistical model similar to the Markov model. Both models use statistics calculated based on the training data to evaluate a window of residues for each classification group. The highest scoring classification group is assigned as the classification for the current residue.

The GOR I model is quite different from the implemented Markov models. It incorporates more direct information of the order of the residues. Information regarding the exact positions of the residues in a local sequence is incorporated into the parameters defining the model (the frequencies calculated based on the training data).

The Markov model does take the order of the residues into account, but it is in a more indirect way. The Markov models deals in general with pairs (not knowing the exact position).

Also the GOR model uses more parameters than the Markov model based on the first order Markov assumption. This fact means that the GOR I model incorporates more specific information than the Markov model.

The extensions developed based on the basic Markov model show that it is indeed possible to increase the performance of the models based on analyses and investigations of the different variables used in the model. The overall prediction accuracy is increased by 6% by changing the Markov model or extending it. There is reason to believe that it is possible to increase the performance even more, given the time needed for this. Also no biological knowledge has been incorporated or utilized for any of the extensions. The newest GOR model (GOR V) shows that it is possible to increase the general performance drastically when incorporating several different kinds of biological methods. This could be the source of several interesting analyses using the Markov models (and the extensions) presented and on top of these use biological background knowledge.

The test results have shown that it appears to be difficult to achieve a performance at the same level as the GOR I model using the current definition (and implemented extensions) of the Markov model.

Future Perspectives

The report has presented Markov models in the simple form and several extensions and/or modifications of this model. Some of these extensions have revealed several weaknesses in the Markov model. Given the time needed it could be interesting to explore these weaknesses more and try to improve the model in these specific areas.

The extensions presented are applied in different stages of the model. These stages may be described in short as:

- Training (including subsequence extraction and parameter estimation).
- Using the score functions.
- Manipulating the values returned from the score functions.
- Making the decision of which group to predict as the classification for some residue.
- Combining models and extensions.

These stages have all been explored to some degree but could all be explored even further. Extensions for each of these stages have been presented. Combining different extensions appeared to be very effective. This might mean that other model types (for instance neural networks) could be combined successfully with the Markov models, resulting in higher prediction accuracies.

The training stage where the subsequences are extracted from the correctly classified sequences leading to the parameter estimation is an interesting stage. The extensions presented (from this stage) have not resulted in remarkable results but it has not been researched that much either. This is a possible subject for further research and improvement.

If we look at the model type in general, a Markov model using the Markov assumption of first order has been used. It could be interesting to use Markov models based on the second (or third) order Markov assumption. These models will incorporate more parameters but it seems that there are sufficient data in the DSSP database for this (GOR III and IV also have many parameters). The whole normalization process (using models based on different window sizes) could also be explored in more details.

Also it could be interesting to incorporate biological knowledge into the whole prediction process. The GOR V model has been increased by around 10% using multiple alignments. The database could be inspected for errors by someone with a biological background and subsequences which do not make physical sense could be corrected.

There are plenty of things which may be explored in more details. It does not automatically mean that the performance of the models will increase drastically, but it is definitely possible.

Conclusion

The tests and analyses conducted in this report have shown that Markov models are able to predict the secondary structure of unknown protein sequences to some extent. The prediction accuracy obtained using the basic model is better than using the trivial classification where every residue is predicted as the most frequent classification group. The trivial classification has a Q_3 of 42.8%. The basic Markov model has a Q_3 of 51.2%.

The basic Markov model has been extended in different ways. The extensions are grouped according to where they are applied in the model. It would be fairly easy to think of new extensions using ideas based on methods presented in this report. Several extensions have been applied to the same model. This did indeed increase the performance of the model even more. The combined model consisted of 4 different extensions. The best of these extensions has Q_3 = 54.8%. When all extensions are combined the new model achieves a Q_3 of 57.2%.

The different GOR models have also been implemented. These models perform better than the Markov models. The Q_3 value of the best GOR model (GOR IV) is 63.4%. The GOR model using the fewest number of parameters (GOR I) has Q_3 = 60.5%.

The GOR models and Markov models have been compared. One of the differences of the two models is that the GOR model stores information about the exact position of the residues. The Markov model only contains indirect information about the position of the residues. The position of the residues is indirectly given by the probability of the current window of residues (using pair statistics).

As mentioned in the previous section there are several possible extensions and variations of the Markov model that could be interesting to explore in more detail. This project has presented some ideas of how the models may be analyzed and extended.

References

1. Garnier J, Osguthorpe DJ, Robson B. Analysis of the Accuracy and Implications of Simple Methods for Predicting the Secondary Structure of Globular Proteins. J Mol Biol 1978 Mar 25;120(1):97-120.
2. Garnier J, Gibrat, JF, Robson B. GOR Method for Predicting Protein Secondary Structure from Amino Acid Sequence. Methods in Enzymology, vol. 266, pg. 540-553.
3. Kloczkowski A, Ting K.L, Jernigan R.L, Garnier J. Combining the GOR V Algorithm With Evolutionary Information for Protein Secondary Structure Prediction From AminoAcid Sequence. Proteins 2002,49:154-166.
4. Fano R. Transmission of Information. Wiley, New York, 1961.
5. Cuff JA, Barton GJ. Evaluation and Improvement of Multiple Sequence Methods for Protein Secondary Structure Prediction. Proteins 1999;34:508-519.
6. Cuff JA, Barton GJ. Application of Multiple Sequence Alignment Profiles to Improve Protein Secondary Structure Prediction. Proteins 2000;40:502-511.
7. Centre for Molecular and Biomolecular Informatics, DSSP. http://www.cmbi.kun.nl/gv/dssp/. Last viewed 10/12-2003.
8. Petersen T.N, Lundegaard C, Nielsen M, Bohr H, Bohr J, Brunak S, Gippert G.P, Lund O. Prediction of Protein Structure at 80% Accuracy. Proteins 2000;41:17-20.
9. Moult J, Judson R, Fidelis K, Pedersen JT. A Large Scale Experiment to Assess Protein Structure Prediction. Proteins 1997;27:329-335.
10. Frishman D, Argos P. Seventy-five Percent Accuracy in Protein Secondary Structure Prediction: Combination of Three Different Methods. Protein Eng 1988;2:185-191.
11. Rost B, Sander C. Third Generation Prediction of Secondary Structure. Webster DM, editor. Protein Structure Prediction: Methods and Protocols. Totowa, NJ: Humana Press; 2000, page 71-95.
12. The Barton Group. Jpred Distribution Material. http://www.compbio.dundee.ac.uk/~www-jpred/data/. Last viewed 10/12-2003.
13. Matthews BB. Comparison of the Predicted and Observed Secondary Structure of T_4 Lysozyme. Biochim Biophys Acta 1975;405:442-451.
14. Zhang Z, William I W. A profile hidden Markov model for signal peptides generated by HMMER. Bioinformatics Applications Note 2003;19:307-308.
15. Yoshikawa H, Ikeguchi M, Nakamura S, Shimizu K, Doi J. Prediction of Protein Structure Classes and Secondary Structures by Means of Hidden Markov Models. Systems and Computers in Japan, 1999;30:13:13-22.
16. Arjunan S.N.V, Deris S, Illias R.M. Prediction of Secondary Structure. Jurnal Teknologi 35(C);2001;81-90.
17. The U.S. Department of Energy, http://www.ornl.gov/sci/techresources/Human_Genome/home.shtml. Last viewed 15/01-2004.
18. The University of Arizona. The Biology project. http://www.biology.arizona.edu/biochemistry/problem_sets/large_molecules/03t.html?. Last viewed 15/01-2004.
19. The U.S. Department of Energy, The Critical Assessment of Techniques for Protein Structure Prediction, CASP, http://predictioncenter.llnl.gov/casp[X]/ where [X] = {1, 2, 3, 4, 5}. Last viewed 3/2-2004.
20. NCBI, Blast Information, http://www.ncbi.nlm.nih.gov/BLAST/. Last viewed 17/2-2004.
21. Institute of Microbial Technology, Chandigarh, India, http://www.imtech.res.in/raghava/mhcbench/parameter.html. Last viewed 20/2-2004.
22. Fosler-Lussier, E. Markov Models and Hidden Markov Models: A Brief Tutorial. Berkeley, CA TR-98-041;1998;p1

Appendix A – The Markov Assumption

The parameters defining the Markov models (as described in 3.3.1.Definition of the Markov Models) are based on the so called Markov assumption [22]. The idea is that the probability of seeing a certain sequence is reduced to a product of simple terms, where each term is the probability of seeing some character *a* having the character *b*.

The following formula is known as the Markov assumption of first order:

$$p(x_n \mid x_{n-1},...x_2,x_1) \approx p(x_n \mid x_{n-1})$$

The assumption states that the probability of an observation at the time *n* only depends on the observation at the time *n*-1. If we use a Markov assumption of second order the observation at the time *n* depends on observations at the time *n*-1 and *n*-2.

In the following the term *Markov assumption* or just *assumption* is used instead of the *first order Markov assumption*. The Markov assumption makes it possible to reduce the complicated term stating the probability to see a certain sequence of observations:

$$\begin{aligned} p(x_1,x_2,x_3,...,x_n) &= p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_2,x_1)\cdots p(x_n \mid x_{n-1},x_{n-2},...,x_1) \\ &= p(x_1)\prod_{n=2}^{N} p(x_n \mid x_{1:n-1}) \end{aligned}$$

In the formula the term $x_{1:n-1}$ means $x_1,x_2,...,x_{n-1}$. Using the Markov assumption it is possible to rewrite this last equation to a more simple equation:

$$p(x_1)\prod_{n=2}^{N} p(x_n \mid x_{1:n-1}) \approx p(x_1)\prod_{n=2}^{N} p(x_n \mid x_{n-1})$$

This equation is the basis for the Markov models used in this project. The Markov models used will observe residues in a sequence. These residues are placed in some order in the training set. To store information about all combinations of residues (according to type and position) would require too much space and information from the databases. When we use the Markov assumption we reduce the problem to only two different parameters $p(a)$ and $p(a|b)$ which adds up to a total count of s^2+s parameters (where $s=|\sum|$). If the Markov assumption is not used the total number of parameters would be $s^n + s^{n-1} + s^{n-2} + \dots + s$ where n is the number of residues in our window. This number increases very fast when *n* is increased.

The Markov model is holding the values for $p(a)$ and $p(a|b)$. $p(a)$ is the probability that a sequence starts with *a* and $p(a|b)$ is the probability of seeing *a* when *b* is given (meaning that *a* follows immediately after *b* in a sequence).

Appendix B – Using Pseudo Counts

This appendix validates the parameter estimations when using pseudo counts in the Markov models. The two parameters should always respect the conditions given in (3.14. This appendix shows that this is still the case after the introduction of the pseudo count constant.

p(*a*) sums to 1 when *a* varies over the alphabet, ∑. *p*(*a*|*b*) sums to 1 for all possible *a*'s (in the alphabet) when *b* varies over the alphabet.

The following shows that *p*(*a*) sums to 1:

Given that :

a,b	are residues in the alphabet, Σ
y_1^j	is the first residue in subsequence j
k	is the total number of subsequences in the collection
i	is the position in the current subsequence, $i > 0$
c	is the pseudo count

$$\begin{aligned}
\sum_{a\in\Sigma} p(a) \quad &= \sum_{a\in\Sigma} \frac{c + \left|\{ j \mid y_1^j = a\}\right|}{|\Sigma| c + k} \\
&= \frac{1}{|\Sigma| c + k} \sum_{a\in\Sigma} \left(c + \left|\{ j \mid y_1^j = a\}\right| \right) \\
&= \frac{1}{|\Sigma| c + k} \sum_{a\in\Sigma} \left(c + \left|\{ j \mid y_1^j = a\}\right| \right) \\
&= \frac{1}{|\Sigma| c + k} \left(\sum_{a\in\Sigma} c + \sum_{a\in\Sigma} \left|\{ j \mid y_1^j = a\}\right| \right) \\
&= \frac{1}{|\Sigma| c + k} \left(|\Sigma| c + k \right) = 1
\end{aligned}$$

The following shows that *p*(*a*|*b*) also sums to 1:

$$
\begin{aligned}
\sum_{b\in\Sigma} p(a \mid b) \quad &= \sum_{b\in\Sigma} \frac{c + \left|\left\{(i,j) \mid i \geq 2 \wedge y_i^j = a \wedge y_{i-1}^j = b\right\}\right|}{|\Sigma|c + \left|\left\{(i,j) \mid i \geq 2 \wedge y_i^j = a\right\}\right|} \\
&= \left(\frac{1}{|\Sigma|c + \left|\left\{(i,j) \mid i \geq 2 \wedge y_i^j = a\right\}\right|}\right) \sum_{b\in\Sigma} \left(c + \left|\left\{(i,j) \mid i \geq 2 \wedge y_i^j = a \wedge y_{i-1}^j = b\right\}\right|\right) \\
&= \left(\frac{1}{|\Sigma|c + \left|\left\{(i,j) \mid i \geq 2 \wedge y_i^j = a\right\}\right|}\right) \sum_{b\in\Sigma} c + \sum_{b\in\Sigma} \left|\left\{(i,j) \mid i \geq 2 \wedge y_i^j = a \wedge y_{i-1}^j = b\right\}\right| \\
&= \left(\frac{1}{|\Sigma|c + \left|\left\{(i,j) \mid i \geq 2 \wedge y_i^j = a\right\}\right|}\right) \left(|\Sigma|c + \left|\left\{(i,j) \mid i \geq 2 \wedge y_i^j = a\right\}\right|\right) \\
&= 1
\end{aligned}
$$

Appendix C – Additional Analyses and Tests

Reversing Training Sequences in the Basic Markov Model

Investigating the estimated *p*(*a*) and *p*'(*a*) parameters on the DSSP database using both the normal sequence direction (as given in the database) and using reversed sequences the following graph (for the H classification group) is obtained.

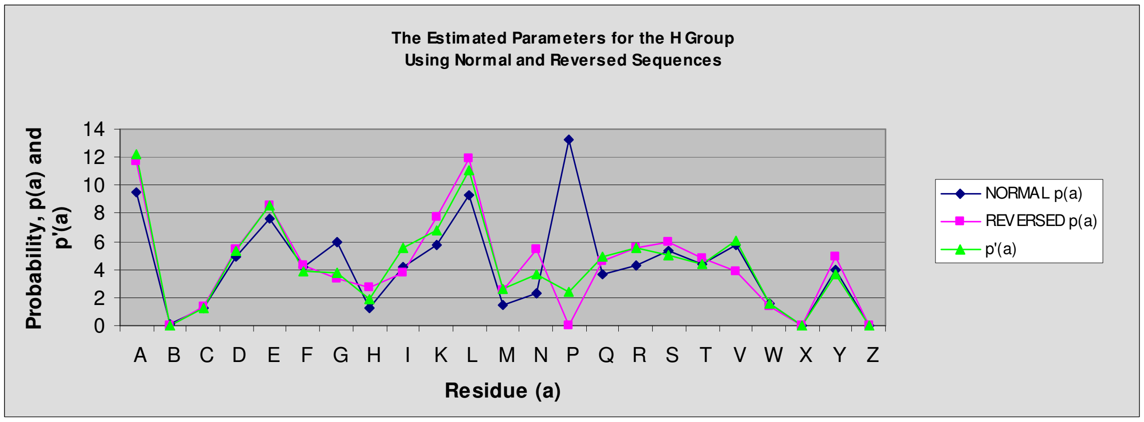


Figure 39: The Estimated parameters using both the normal sequences and the reversed ones.

Figure 39 shows how the parameter *p*(*a*) behaves when estimated on the normal sequences (as given in the database) and on the reversed sequences. The parameter *p*'(*a*) is also given. Tests show that training the model on reversed sequences lead to better results. The following will use Figure 39 to explain this.

The *p*(*a*) terms has an effect on the score function. We know from the tests that using *p*'(*a*) leads to higher performance. We will therefore use *p*'(*a*) as a guideline to a better performance on the graph on Figure 39. The closer the two *p*(*a*)-curves are to the *p*'(*a*)-curve the better the performance is.

In Figure 39 the parameter *p*(*R*) (the probability of observing residue R as the first residue) differs somewhat when using normal sequences and reversed sequences compared to the *p*'(*a*) graph. We see that the *p*(*a*) graph using normal sequences differ the most. This provides us with a possible explanation why the model using *p*(*a*) with normal sequences performs worse than *p*(*a*) using reversed sequences.

Similar places in the graph may be found (although not that extreme) and may contribute to the explanation that the estimated *p*(*a*) parameter has some impact on the score function and in the end on the overall Q_3 (partly causing a difference of 0.8-1.5% when using the basic Markov model). Using the parameter definition *p*'(*a*) eliminates (to some extent) this difference in Q_3.

Ratio-I Method

Plotting the individual Q_3 performances for each group H, E and C gives us the following graph:

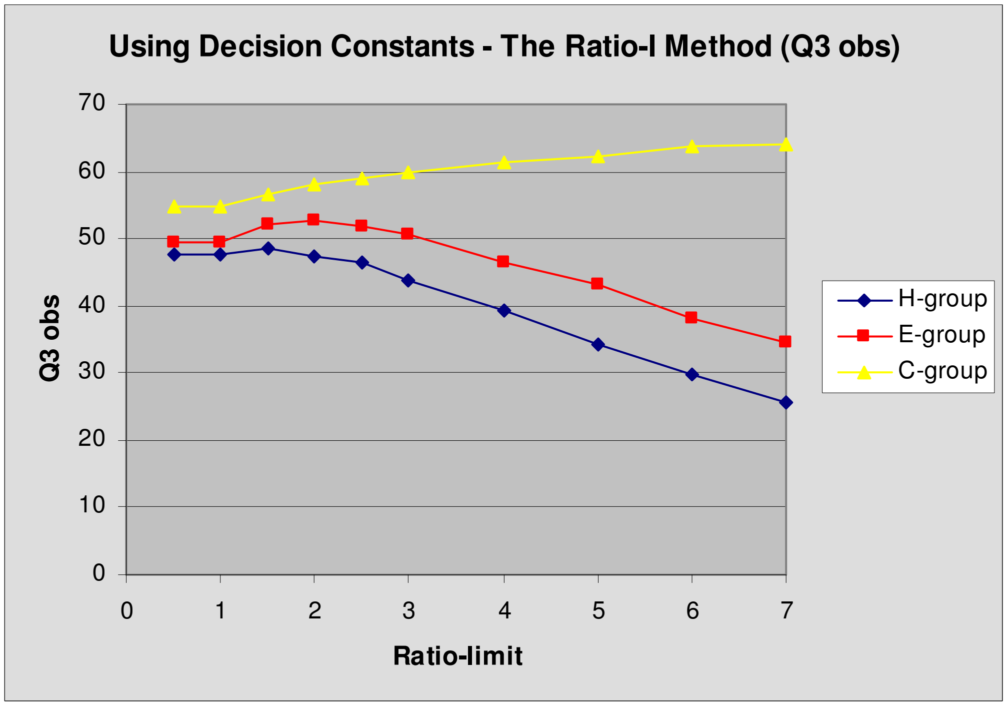


Figure 40: Individual Q_3 values for each group.

Figure 40 shows the individual Q_3 values for the three groups (using window size = 4). This gives us another picture of how the method Ratio-I affects the model. We can see that a difference-limit up to 2.0 improves the performance of the coil and beta sheet groups while the helix group is almost unaffected. Having a difference-limit above 2.0 another effect is seen.

When the difference limit is greater than 1.0 the Q_3 value for the coil group are superior compared to the two other values. In this interval the overall value of Q_3 drops (as seen in Figure 20). The interesting part here is that the method introduced affects the individual models differently. The coil model improves but the other models get worse. We will now look closer at the accuracy matrix generated at the end of the jackknife procedure using the above method (and having a difference-limit of 7).

	H	E	C	Total
H	7429	6383	15051	28863
E	2448	6487	9887	18822
C	5704	6884	22307	34895
Total	15581	19754	47245	82580
Q pred	47.7	32.8	47.2	
Q obs	25.7	34.5	63.9	

Table 31: Statistics for the Ratio-I method when the ratio-limit is 7.

If we look at the table we see that the models classify the residues as coils very often. The total number of residues classified by the models as coils is 47245 out of 82580 possible. This means that the models classify 57% of all residues as coils. The actual distribution of coils is 34895/82580 = 43%. This means that the model is classifying too many residues as coils. Some of these guesses have to be wrong because there are only 43% residues which are coils. This could explain why the individual Q_3 value is high. The model classifies frequently as coil and therefore we have more correctly classified coils.

The fact that the model over classifies the coil group causes the prediction accuracy for the two other groups to drop. The reason for this is that when the model classifies residues that actually are beta sheets or helices it may classify them as coils. These residues will be incorrectly classified and the prediction accuracy will drop for these two groups individually.

The expression Q_{pred} also shows us that of all the residues we predict as coils only 47% of them are predicted correctly. This is another way of saying that the model predicts coils *too often* and as a result misclassifies the other groups.

There is still an open question of why the coil group is predicted frequently. If we recall how the method works we know that a new classification group is only chosen if the score value is bigger than the second greatest value of a factor given by the ratio-limit. It could be a fact that the coil model outputs high values for the score function whenever it predicts a residue as coil. This will cause a change in the classification (the model classifies a residue as a coil). After this classification has been made the other model has to be large enough in order to avoid that the following residue will be classified as coil. If the helix and beta sheet groups are not able to do this we would observe the effect seen in the graph.

In this analysis we have not found any information that makes us able to improve this model further, but we have seen that the accuracy matrix makes it possible to further analyze the results we have obtained. This could be a key element to figure out what makes the model better and therefore a deeper insight to which elements that have an impact on the model.

Appendix D – Command Line Utility

This appendix will show examples of how to run some of the test cases using the command line utility. In the following we assume that the folder structure showed in Figure 41 is used.

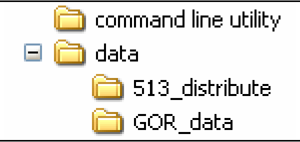


Figure 41: Folder structure used in the examples.

Figure 41 shows the folder structure used for all examples in this appendix. The folder `command line utility` contains the `CTest.class` file (which is the utility). When we run the utility we are in the `command line utility` folder. The folder `data` contains the DSSP database and the GOR database each database has its own folder. The default values for the paths (in the utility) also assume that the folder structure is as shown in Figure 41. If another folder structure is used the data path should be given as an argument to the utility.

In the following we present the command lines for running test cases using each of the possible test types.

The commands shown in Listing 10 run the combination of Test Case 1 and Test Case 2.

```
1:      java CTest basicmodel csv:test1.csv i:0.0,1.0,0.5 l:1,10
2:      java CTest basicmodel csv:test2.csv i:5,10,5 l:1,10
3:      java CTest basicmodel csv:test3.csv i:100,1000,900 l:1,10
```

Listing 10: Three command lines are used for running the combination of test case 1 and 2.

Listing 10 shows the three commands used to perform the combination of Test Case 1 and Test Case 2. The first line loops over the pseudo count values 0.0 and 1.0 with step size 0.5 this is specified using the argument ‘`i:0.0,1.0,0.5`’. This means that the values: 0.0, 0.5, 1.0 are tested. At the same time we also test the values: {1,2,3,4,5,6,7,8,9,10} this is specified using the argument ‘`l:1,10,1`’. The ‘`l:`’ argument specifies the start, end and step size for the window size.

In Test Case 1 and Test Case 2 the values 0.0, 0.5, 1.0, 5.0, 10.0, 100.0 and 1000.0 are used. The values 5.0 and 10.0 are tested using the command in line 2. The values 100 and 1000 are tested using the command in line 3. Each command generates a new excel file (in csv format). This is specified using the argument ‘`csv:`’. The files generated in Listing 10 are therefore `test1.csv`, `test2.csv` and `test3.csv`. The merging of these output files are up to the user.

Listing 11 shows how a part of Test Case 6 is started. The part we are interested in is where we use reversed pair and *p*’(*a*) in the same model.

```
1:      java CTest basicmodel i:1,1,1 l:3,6 +ft +rp
```

Listing 11: Testing the use of reversed pair (`+rp`) and *p*’(*a*) (`+ft`) for window sizes 3,4,5,6 using pseudo count 1.0.

Listing 11 shows the use of the extension arguments. We see two new arguments `+ft` and `+rp`. The first argument `+ft` enables the use of *p*’(*a*). The second argument `+rp` enables the use of the reversed pair method. We also see that no output file is specified which means that the default filename is used (`output.csv`).

```
java CTest gor datapath:.\..\Data\GOR_data datafile:GOR_data.dat +g1
```

Listing 12: Testing GOR I algorithm using the GOR-database.

Listing 12 shows how to test the GOR I algorithm using the GOR database. The test type `gor` makes sure that the GOR algorithms are used. The argument `+g1` enables GOR I algorithm (arguments `+g3` and `+g4` will enable GOR III and GOR IV algorithm). We also see two new arguments `datapath` and `datafile`. These new arguments make sure that we point to the GOR database in the correct folder and that we use the GOR sequences (supplied in GOR_data.dat).

```
java CTest combo l:3,6 i:1,1,1
```

Listing 13: A command line tests the combination of models having different orientation. Window sizes 3 to 6 and pseudo count 1.0 is used.

Listing 13 shows how to run Test Case 19. In this test case the combination of two models having different orientation is tested on window sizes from 3 to 6 and pseudo count 1.0. The new test type `combo` can also be seen in Listing 13.

```
java CTest basicnorm l:2,5 n:3,5 i:1,1,1
```

Listing 14: Testing the basic normalization method looping through window size values from 2 to 5 and normalizing models from 3 to 5 using pseudo count 1.0.

Listing 14 shows how to run Test Case 17. The new argument here is ‘`n:`’. This argument specifies how many models that should be normalized. The first test normalizes 3 models starting at window size 2 meaning that window sizes 2, 3 and 4 are normalized. The new test type is `basicnorm`.

```
java CTest freqnorm l:2,5 n:3,5 i:1,1,1 +f1
java CTest freqnorm l:2,5 n:3,5 i:1,1,1 +f2
java CTest freqnorm l:2,5 n:3,5 i:1,1,1 +f3
```

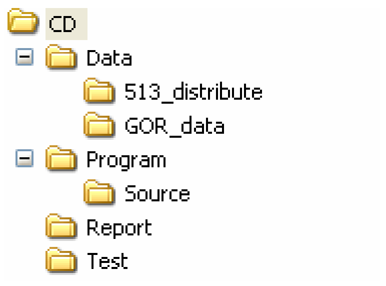
Listing 15: Testing the frequency normalization method looping through window size values of 2 to 5 and normalized models of 3 to 5 using pseudo count 1.0.

Listing 15 shows how to run Test Case 18. The new arguments here are `+f1`, `+f2` and `+f3`. These arguments specify what method to use when counting the frequencies (see section 3.6.2 Frequency Normalization). The new test type introduced here is `freqnorm`.

Appendix E – CD and Source Code

All source code, test files and so on are available on the associated cd.

The folder structure on the CD is given in the following figure:



The content of every dir is now explained.

Data:	This directory contains the two directories 513_Distribute and GOR_data, which holds the DSSP and GOR databases respectively.
Program:	This directory holds the implemented program. The jar executable is located in the root of this directory along with the database definition files (.dat). In the subdirectory Source, the actual source files are kept (.java), which may be compiled using the javac command.
Report:	This directory hold the final report file.
Test:	This directory holds the test files in Excel format (.xls). All results are available in these files.

Now running a specific test using the jar executable may be done as usual. A few examples are given (assuming the current directory is the program directory mentioned above):

Display help:

```
java -jar RunTest.jar -h
```

Run a test using the basic Markov model, the pseudo count constant of 1 and the window size of 5:

```
java -jar RunTest.jar basicmodel i:1,1,1 l:5,5 csv:c:\output.csv
```

The program has been implemented using Java version 1.4.0_02.

Resulting output running `java -version`:

```
> java version "1.4.0_02"
> Java(TM) 2 Runtime Environment, Standard Edition (build 1.4.0_02-b02)
> Java HotSpot(TM) Client VM (build 1.4.0_02-b02, mixed mode)
```

Listing 16: Output running the java –version command.