Python programming — machine learning

Finn Årup Nielsen

DTU Compute
Technical University of Denmark

October 28, 2013

Overview

Machine learning: Unsupervized and supervized

The concept of generalization

Overview of Python machine learning packages

Computations in a naïve Bayes classifier

Example with Pima data set: Baseline, linear, random forest.

Machine learning

Unsupervized learning

You have multiple data items each with (usually) multiple features

Example: Topic mining

Supervized learning

You have multiple data items each with (usually) multiple features and an associated label (or continuous values) to each data item.

You usually want to predict unknown labels

Example: spam detection, sentiment analysis

Supervized learning

Regression (usually when labels and classification are continuous)

"Classical statistics": Linear regression, multiple regression.

(Artificial) neural networks: Nonlinear model

Naïve Bayes Classifier: A model with assumption of independence between features.

Others: "Support vector machine", "Random forest", . . .

Generalization

Generalization = How the (trained) model perform on new unseen data.

When you estimate/train a machine learning model on (training) data you will likely "overfit", i.e., the model parameters are fitted to the data "too much".

To get a unbiased estimate of the performance of the machine learning model you need to split the data and train on one part and test on the other.

Important concept: Cardinal sin to evaluate the performance on the training set.

Python machine learning package

Name	KLoC	GS-cites	Reference
Scipy.linalg			
Statsmodels	92	2	(Seabold and Perktold, 2010)
Scikit-learn	398	278	(Pedregosa et al., 2011)
Orange	286	0 (yet!)	(Demšar et al., 2013)
mlpy	75	8	(Albanese et al., 2012)
MDP	31	58	(Zito et al., 2008)
Gensim	9	62	(Řehůřek and Sojka, 2010)
NLTK	214	527	(Bird et al., 2009)
. . .			

Python supervized learning

Linear regression: `scipy.linalg.lstsq`

scikit-learn: Lots of algorightms for supervized learning

NLTK has its own naïve Bayes classifier (`nltk.NaiveBayesClassifier`) and access to external classifiers, e.g., scikit-learn (if setup)

Naïve Bayes classifier (NBC)

Described in Python books with full implementation (Segaran, 2007, pp. 123–127, 277–281) or just the functional interface to the NTLK implementation (Bird et al., 2009, chapter 6)

"Bayes" because the Bayes theorem is used on the features/labels:

$$P(\text{label}|\text{features}) = \frac{P(\text{features}|\text{label})\, P(\text{label})}{P(\text{features})} \tag{1}$$

"Naïve" because of a strong (usually wrong) assumption about independence of the features:

$$P(\text{features}|\text{label}) = \prod_n P(\text{feature}_n|\text{label}) \tag{2}$$

Note that the features in the NBC might be both categorical (binary, multiple classes) or continuous, but that the Python books present the NBC for binary features.

Example with NLTK

Small data set with two documents, one spam and one "ok":

```
Buy Viagra
Hello dear
```

NLTK NBC trained with features (here: words) extracted from the data set and then classifying a new document:

```
import nltk
labeled_featuresets = [({'viagra': True, 'hello': False}, 'spam'),
                       ({'viagra': False, 'hello': True}, 'ok')]
classifier = nltk.NaiveBayesClassifier.train(labeled_featuresets)
classifier.prob_classify({'viagra': True, 'hello': False}).prob('spam')
0.9
```

So what does "`0.9`" mean? An estimate of the probability that the new document is spam!

Contingency table of documents

NBC makes estimation from data represented in a contingency table:

Features/Labels	Spam	Ok	...	Totals
Viagra (`True`)	1	0	0	1
¬ Viagra (`False`)	0	1	0	1
...	0	0	0	0
Totals	1	1	0	2

Table body:

```
>>> [ (k,v.freqdist().items()) for k,v in
                                  classifier._feature_probdist.items() ]
[(('ok', 'hello'), [(True, 1)]), (('ok', 'viagra'), [(False, 1)]),
 (('spam', 'hello'), [(False, 1)]), (('spam', 'viagra'), [(True, 1)])]
```

Last row:

```
>>> [ (k,v) for k,v in classifier._label_probdist.freqdist().items() ]
[('ok', 1), ('spam', 1)]
```

Naïve naïve Bayes classifier

"Naïve naïve Bayes classifier": estimation of probabilies with maximum likelihood:

$$P(\text{viagra}|\text{spam}) = \frac{P(\text{viagra}, \text{spam})}{P(\text{spam})} \approx \frac{\text{count(viagra, spam)}}{\text{count(spam)}} = \frac{1}{1} = 1 \quad (3)$$

$$P(\text{hello}|\text{spam}) = \frac{P(\text{hello}, \text{spam})}{P(\text{spam})} \approx \frac{0}{1} = 0 \quad (4)$$

But "hello" might occure in some spam documents, so it might not be a good idea to estimate the probability as zero.

To get around that problem a technique is used that in text mining is called **smoothing**.

Smoothing

NLTK implements smoothing in `nltk.probability`: "Lidstone estimate", "expected likelihood estimate" (ELE), . . .

$$P(\text{feature}|\text{label}) \approx \frac{\text{count}(\text{feature, label}) + \gamma}{\text{Total} + B\gamma}, \tag{5}$$

where γ is the amount of smoothing and B is the number of labels.

For ELE $\gamma = 0.5$.

Wikipedia refers to this as "additive smoothing".

This kind of smoothing is also closely associated with what in statistics/machine learning is called the Dirichlet prior for the multinomial distribution.

Prior

Prior probabilities is unknown, but usually estimated/set to the relative frequency the different labels occur in the full data set:

$$P(\text{spam}) = \frac{\text{count(spam)}}{\text{count(spam)} + \text{count(ok)}} = \frac{1}{1+1}$$

This is the maximum likelihood estimate.

Default estimation in NLTK is ELE ($\gamma = 0.5$) and the estimated probability distribution is available in `classifier._label_probdist`

Example with two spam document and one ok document the priors are computed as:

$$\begin{aligned} P(\text{spam}) &= \frac{2+0.5}{3+1} = 0.675 \\ P(\text{ok}) &= \frac{1+0.5}{3+1} = 0.375 \end{aligned}$$

Probabilistic prediction computations

$$
\begin{aligned}
P(\text{spam}) &= (1 + 0.5)/(2 + 2 \times 0.5) = 0.5 \\
P(\text{ok}) &= (1 + 0.5)/(2 + 2 \times 0.5) = 0.5 \\
P(\text{viagra}|\text{spam}) &= (1 + 0.5)/(1 + 1) = 0.75 \\
P(\text{viagra}|\text{ok}) &= (0 + 0.5)/(1 + 1) = 0.25 \\
P(\text{hello}|\text{spam}) &= (0 + 0.5)/(1 + 1) = 0.25 \\
P(\text{hello}|\text{ok}) &= (1 + 0.5)/(1 + 1) = 0.75 \\
P(\text{features}) &= \sum_m \prod_n P(\text{feature}_n|\text{label}_m)\, P(\text{label}_m) \\
P(\text{spam}|\text{viagra}, \neg\text{hello}) &= \frac{\prod_n P(\text{feature}_n|\text{spam})\, P(\text{spam})}{P(\text{features})} \\
&= \frac{0.75 \times 0.75 \times 0.5}{0.75 \times 0.75 \times 0.5 + 0.25 \times 0.25 \times 0.5} \\
&= 0.28125/(0.28125 + 0.03125) \\
&= 0.9
\end{aligned}
$$

Another dataset

Now with three features:

```
>>> labeled_featuresets = [
      ({'viagra': True, 'buy': True, 'hi': False}, 'spam'),
      ({'viagra': True, 'buy': False, 'hi': False}, 'spam'),
      ({'viagra': False, 'buy': False, 'hi': True}, 'ok')]
>>> classifier = nltk.NaiveBayesClassifier.train(labeled_featuresets)
>>> print(classifier._label_probdist.freqdist())
<FreqDist: 'spam': 2, 'ok': 1>
>>> classifier.prob_classify({}).prob('spam')
0.625
```

Prior on 0.625 equal to:

$$(2 + 0.5)/(3 + 2 \times 0.5) = 0.625$$

Most informative feature

The NLTK NBC has a convenient method to display “informative” features (`most_informative_features/show_most_informative_features`).

How “informative” a feature n is, is computed as separated for each feature n

$$I_n = \max_{\text{label}}(P(\text{feature}_n|\text{label})/\min_{\text{label}}(P(\text{feature}_n|\text{label}))$$

For the “viagra” feature:

$$\begin{aligned} I_{\text{viagra}} &= 0.71/0.2 = 3.57 \\ I_{\neg\text{viagra}} &= 0.6/0.14 = 4.2 \end{aligned}$$

Note that there is a bug/feature in the present version (naivebayes.py 2063, 2004-07-17) of the `most_informative_features` and `show_most_informative_features` methods so that not all features are show for the present example. :-(

Manual computations

$$
\begin{aligned}
P(\text{spam}) &= 0.675 \\
P(\text{viagra}|\text{spam}) &= (2+0.5)/(2+1) = 0.833 \\
P(\text{viagra}|\text{ok}) &= (0+0.5)/(1+1) = 0.25 \\
P(\neg\text{viagra}|\text{spam}) &= (0+0.5)/(2+1) = 0.166 \\
P(\text{buy}|\text{spam}) &= (1+0.5)/(2+1) = 0.5 \\
P(\neg\text{buy}|\text{spam}) &= (1+0.5)/(2+1) = 0.5 \\
P(\neg\text{buy}|\text{ok}) &= (1+0.5)/(1+1) = 0.75 \\
P(\text{hi}|\text{spam}) &= (0+0.5)/(2+1) = 0.166 \\
P(\neg\text{hi}|\text{spam}) &= (2+0.5)/(2+1) = 0.833 \\
P(\neg\text{hi}|\text{ok}) &= (0+0.5)/(1+1) = 0.25 \\
P(\text{features}) &= \sum_m \prod_n P(\text{feature}_n|\text{label}_m)\, P(\text{label}_m) \\
P(\text{features}, \text{spam}) &= 0.833 \times 0.5 \times 0.833 \times 0.675 = 0.234375 \\
P(\text{features}, \text{ok}) &= 0.25 \times 0.75 \times 0.25 \times 0.325 = 0.015234375 \\
P(\text{spam}|\text{viagra}, \neg\text{buy}, \neg\text{hi}) &= \frac{\prod_n P(\text{feature}_n|\text{spam})\, P(\text{spam})}{P(\text{features})} \\
&= \frac{0.234375}{0.234375 + 0.015234375} = 0.938967
\end{aligned}
$$

Probabilistic prediction computation

```
>>> classifier.prob_classify({'viagra': True,
                              'hi': False,
                              'buy': False}).prob('spam')
0.938213041911853
```

Notice the small difference between the manual and the NLTK results! :-o

The present version of NLTK handles feature/label combinations it hasn't seen (e.g., viagra=True, ok) in a special way!

Pima data set machine learning

Get a data set

Get a data set consisting of training and testing parts from *R*:

```
$ R
> library(MASS)
> write.csv(Pima.tr, "Pima.tr.csv")
> write.csv(Pima.te, "Pima.te.csv")
```

We want to predict the “type” column of each row from Its either “yes” or “no” indicate whether a subject has diabetes or not.

This data set is not sparse.

First performance measure function

```
def accuracy(truth, predicted):
    if len(truth) != len(predicted):
        raise Exception("Wrong sizes ...")
    total = len(truth)
    if total == 0:
        return 0
    hits = len(filter(lambda (x, y): x == y, zip(truth, predicted)))
    return float(hits)/total


def test_accuracy():
    assert accuracy([], []) == 0
    assert accuracy([1], [1]) == 1.0
    assert accuracy([1], [0]) == 0.0
    assert accuracy([1, 2, 3, 4], [1, 5, 3, 7]) == 0.5
```

Baseline model

"Training" a baseline model:

```
>>> import pandas as pd
>>> Pima_tr = pd.read_csv("Pima.tr.csv", index_col=0)
>>> Pima_te = pd.read_csv("Pima.te.csv", index_col=0)
>>> Pima_tr.groupby("type").count()
      npreg  glu   bp  skin  bmi  ped  age  type
type
No      132  132  132   132  132  132  132   132
Yes      68   68   68    68   68   68   68    68
```

There are more "type"= "no" in the training data set.

Lets just predict "no".

“No” classifier

```
class NoClassifier():
    """Classifier that predict all data as "No". """
    def predict(self, x):
        return pd.Series(["No"] * x.shape[0])


no_classifier = NoClassifier()
predicted = no_classifier.predict(Pima_te)

>>> accuracy(Pima_te.type, predicted)
0.6716867469879518
```

Around 67% prediction accuracy on the test set.

Linear model with `scipy.linalg` ...

```
from scipy.linalg import pinv
from numpy import asarray, hstack, mat, ones, where

class LinearClassifier():
    """ y = X*b and b = pinv(X) * y """
    def __init__(self):
        self._parameters = None

    def from_labels(self, y):
        return mat(where(y=="No", -1, 1)).T

    def to_labels(self, y):
        return pd.Series(asarray(where(y<0, "No", "Yes")).flatten())

    def train(self, x, y):
        intercept = ones((x.shape[0], 1))
        self._parameters = pinv(hstack((mat(x), intercept))) * self.from_labels(y)

    def predict(self, x):
        intercept = ones((x.shape[0], 1))
        y_estimated = hstack((mat(x), intercept)) * self._parameters
        return self.to_labels(y_estimated)
```

. . . Linear model

```
lc = LinearClassifier()
lc.train(Pima_tr.ix[:,:7], Pima_tr.type)

predicted = lc.predict(Pima_te.ix[:,:7])

accuracy(Pima_te.type, predicted)
```

Now we get 0.7981927710843374. Somewhat better.

Note the hassle with conversion from and to labels "no" and "yes" and between matrices, arrays and Pandas Series.

Scikit-learn Random Forest

Here the data is converted to numerical arrays and vectors

```
from numpy import where

X_tr = Pima_tr.ix[:,:7]
y_tr = where(Pima_tr.type=="No", -1, 1)
X_te = Pima_te.ix[:,:7]
y_te = where(Pima_te.type=="No", -1, 1)

from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()
rfc.fit(X_tr, y_tr)
>>> predicted = where(rfc.predict(X_te)==-1, "No", "Yes")
0.7228915662650602
>>> rfc.score(X_te, y_te)
0.72289156626506024
```

Ups, not better. Note that the result may vary

ROC curve

See Receiver operating characteristic example using functions from the `sklearn.metrics` module

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

fpr, tpr, thresholds = roc_curve(y_te, rfc.predict_proba(X_te)[:,1])
roc_auc = auc(fpr, tpr)

plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc_auc)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc="lower right")
plt.show()
```

Summary

There is a number of different packages for machine learning and prediction.

Orange and Scikit-learn has a very high developer activity

Note memory issues: The NLTK classifier might use a lot of memory

References

Albanese, D., Visintainer, R., Merler, S., Riccadonna, S., Jurman, G., and Furlanello, C. (2012). mlpy: machine learning python. ArXiv.

Bird, S., Klein, E., and Loper, E. (2009). *Natural Language Processing with Python*. O'Reilly, Sebastopol, California. ISBN 9780596516499.

Demšar, J., Curk, T., Erjavec, A., Črt Gorup, Hočevar, T., Milutinovi, M., Možina, M., Polajnar, M., Toplak, M., Stari, A., Štajdohar, M., Umek, L., Žagar, L., Žbontar, J., Žitnik, M., and Zupan, B. (2013). Orange: data mining toolbox in Python. *Journal of Machine Learning Research*, . Link.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Édouard Duchesnay (2011). Scikit-learn: machine learning in Python. *Journal of Machine Learning Research*, . Link.

Seabold, S. and Perktold, J. (2010). Statsmodels: econometric and statistical modeling with python. In *Proceedings of the 9th Python in Science Conference*. Link.

Segaran, T. (2007). *Programming Collective Intelligence*. O'Reilly, Sebastopol, California.

Zito, T., Wilbert, N., Wiskott, L., and Berkes, P. (2008). Modular toolkit for data processing (mdp): a python data processing framework. *Frontiers in Neuroinformatics*, . DOI: 10.3389/neuro.11.008.2008.

Řehůřek, R. and Sojka, P. (2010). Software framework for topic modelling with large corpora. In *Proceedings of LREC 2010 workshop New Challenges for NLP Frameworks*. Link.