UNIVERSITY OF OXFORD

Machine Learning
Michaelmas Term 2017
Week 6

Practical 2 : Generative and Discriminative Models

In this practical, we'll compare the Naïve Bayes Classifier and Logistic Regression on two different datasets. As part of the practical you should briefly read the following paper:

On Discriminative vs. Generative classifiers: A comparison of logistic regression and naive Bayes
Andrew Y. Ng and Michael I. Jordan
Advances in Neural Information Processing Systems (NIPS) 2001.
Paper available *here*

You should read the Introduction and the Experiments section. The goal in this practical is to qualitatively reproduce some of the experimental results in this paper. You are strongly encouraged to read the rest of the paper, which is rather short and straightforward to read, though some of you may want to skip the formal proofs.

Naïve Bayes Classifier

You should implement a Naïve Bayes Classifier directy in python. To keep your code tidy, we recommend implementing it as a class. Make sure that your classifier can handle at least binary and real-valued features and an arbitrary number of class labels. Suppose the data has 3 different features, the first and third being binary and the second real-valued, and that there are 4 classes. Write an implementation that you can initialise as follows:

```
nbc = NBC(feature_types=['b', 'r', 'b'], num_classes=4)
```

Optional: Also add support to use multivariate featurs, i.e., features that make take some K different values. Note that K may be different for different features.

Along the lines of classifiers provided in `sklearn`, you want to implement two more functions, `fit` and `predict`. The `fit` function should estimate all the parameters of the NBC. You should model all binary features as Bernoulli random variables and all real-valued features as univariate Gaussians. You also need to estimate the class probabilities $\pi_c = p(y = c)$. The `predict` function should compute the class conditional probabilities for the new inputs on all classes and then return the class that has the largest one.

Implementation Issues

- Remember to do all the calculations in *log space* to avoid running into underflow issues.
- Although for the two datasets required for this practical, we've used class labels $0, 1, \ldots, C-1$, try to make your implementation general enough to handle arbitrary class labels (not even necessarily integers)

- As far as possible use matrix operations. So assume that `Xtrain`, `ytrain`, `Xtest` will all be `numpy arrays`. Try and minimise your use of python loops. (In general, looping over classes is OK, but looping over data is probably not a good idea.)
- You want to ensure that the estimates for the parameter for the Bernoulli random variables is never exactly 0 or 1. For this reason you should consider using Laplace smoothing (wiki). Also, the variance parameter for Gaussian distributions should never be exactly 0, so in case your calculated variance is 0, you may want to set it to a small value such as $1e-6$. Note that this is essential to ensure that your code never encounters *division by zero* or *taking logarithms of 0* errors.

For training data `Xtrain, ytrain` and test data `Xtest` you should be able to run:

```
nbc.fit(Xtrain, ytrain)
yhat = nbc.predict(Xtest)
test_accuracy = np.mean(yhat == ytest)
```

Logistic Regression

For logistic regression, you should use the implementation in `sklearn`. Adding the following line will import the LR model.

```
from sklearn.linear_model import LogisticRegression
```

Read the information provided on the following links to understand some details about how the logistic regression model is implemented in scikit-learn.

- http://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
- http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.LogisticRegression

Handin 1: In the lectures, we only formulated the negative log-likelihood for logistic regression without adding any regularization term. As per the formulation used in the lectures, if you wanted to add $\lambda \mathbf{w}^\mathsf{T}\mathbf{w}$ as a regularization to the negative log-likelihood of observing the data, and set $\lambda = 0.1$, what value of C would you set in the sklearn implementation?

Comparing NBC and LR

Iris Dataset

You will now compare the performance of NBC and LR on two different datasets. The first is the iris dataset. You can obtain this as follows:

```
from sklearn.datasets import load_iris
iris = load_iris()
X, y = iris['data'], iris['target']
```

There are three classes denoted by $0, 1, 2$, which stand for setosa, versicolour and virginica, three varieties of iris. There are four features, all real-valued, measurements of sepal length and width, and petal length and width.

Congressional Voting Records Dataset

The second dataset you will use is voting records in the US House of Representatives in 1984. The goal is to predict whether the representative is a Republican or Democrat. The original dataset is available here:

http://archive.ics.uci.edu/ml/datasets/Congressional+Voting+Records

For the purpose of this practical, we have put the data in numpy array format, as well as deleted those records that had missing entries. This dataset is available on the course homepage (as `voting`, not `voting-full`) and can be loaded as follows:

```
import cPickle as cp
import numpy as np

X, y = cp.load(open('voting.cPikcle', 'rb'))
```

A vote of yes is encoded as 1, no as 0. A Republican is 0 and a Democrat 1. However, these particular encodings are only important to the extent they tell us that we need to use binary features in NBC, don't need to perform a one-hot encoding for LR, and are solving a binary classification problem.

Experiments

For both datasets you'll compare the classification error of the NBC and LR trained on increasingly large training datasets. Because the datasets are so small, you should do this multiple times and average the classification error. One run should look as follows:

- Shuffle the data, put 20% aside for testing.

  ```
  N, D = X.shape
  Ntrain = int(0.8 * N)
  shuffler = np.random.permutation(N)
  Xtrain = X[shuffler[:Ntrain]]
  ytrain = y[shuffler[:Ntrain]]
  Xtest = X[shuffler[Ntrain:]]
  ytest = y[shuffler[Ntrain:]]
  ```

- Train 10 classifiers, where the k^{th} classifier is trained using $10k\%$ of the training data. For each classifier store the classification error on the test set.

You may want to repeat this with at least 200 random permutations (possibly as large as 1000) to average out the test error across the runs. In the end, you'll get average test errors as a function of the size of the training data. Plot these curves for NBC and LR on the two datasets.

Handin 2 Include the plots of the two curves in your report.

Optional

Missing Values for Voting Dataset

You may want to use the full voting dataset and repeat the above experiments. The full dataset is available in numpy array format on the course webpage as `voting-full`. Any missing value in the voting record is indicated by 2. For Naïve Bayes, we have seen how missing values can be handled in a principled manner. For LR you may experiment with different options, such as using the mean value of the feature, always using 0 (for missing values), or allowing 'unknown' to be a legitimate feature value. In the last case, you will have to use one-hot encoding to encode the three possible values for each feature.

Other Datasets

There are other datasets used in the paper. Some with missing values, some without. Download them from the UCI website and run the same procedure on them. You will also have to put them in numpy array format. The code for doing this in the case of voting is provided on the course website in the hope that it may be useful for other datasets. The experiments reported in the paper never combine discrete and real-valued features, but you may want to test what happens when you do.