ML 4/M Naive Bayes Classification Lab

Simon Rogers

3rd March 2016

Aims

To build a Naive Bayes classifier with Gaussian class conditional distributions. Note: I've given quite a few code snippets below. There are more concise python-ey ways of implementing this, but I think the ones I've given are a bit easier for you to parse and understand. You don't need to do it this way if you can find a neater way of doing it.

Tasks

- Load the train and test data used in the previous (KNN) lab
- Step 1: *Learning* (fitting the class conditional distributions)
  - You need to fit a Gaussian distribution to each feature (column) for each class.
  - This requires computing the mean and variance of each column for examples of a particular class
  - The following code snipped defines a set of targets (t; using class labels 0 and 1) and then uses this to compute the mean and variance:

```
import numpy as np

# Assumes X is an N x D matrix (D is the number of features)
trainx = np.loadtxt('trainx.csv',delimiter=',')
testx = np.loadtxt('testx.csv',delimiter=',')
traint = np.hstack((np.zeros(50,),np.ones(50,)))
testt = np.hstack((np.zeros(200,),np.ones(200,)))

class0_pos = np.where(traint==0)[0]
class0_mean = trainx[class0_pos,:].mean(axis=0)
class0_var = trainx[class0_pos,:].var(axis=0)
```

```
class1_pos = np.where(traint==1)[0]
class1_mean = trainx[class1_pos,:].mean(axis=0)
class1_var = trainx[class1_pos,:].var(axis=0)
```

- Continued…
  - Define the prior probability to be the proportion of points in each class (you could use `1.0*len(class0_pos)/len(traint)` for example)
- Step 2: *Testing* (evaluating the predictive probability for a test point):
  - Any test point has one value for each feature. For each class, compute the value of the Gaussian pdf for each feature and multiply these values together (and multiply by the prior). Recall that the Gaussian pdf for a Gaussian with mean μ and variance σ^2 is given by:

$$p(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{1}{2\sigma^2}(x-\mu)^2\right\}$$

- Continued…
  - The following code snippet computes the likelihood of a single test point in class 1:

```
test_point = testx[5,:] # random choice of the 6th data point - you should loop
total_like1 = 1.0
# Loop over the features
for i,x in enumerate(test_point):
    total_like1 *= 1.0/(np.sqrt(2.0*np.pi))
    total_like1 *= 1.0/np.sqrt(class1_var[i])
    total_like1 *= np.exp((-1.0/(2.0*class1_var[i]))*(x - class1_mean[i])**2)

# multiply by prior (assuming you've defined this somewhere)
total_like1 *= prior1
```

- Continued…
  - Finally, normalise the likelihoods into probabilities for the two classes. This is as simple as `prob0 = total_like0/(total_like0 + total_like1)`. (assuming `total_like0` is the likelihood for class 0)
  - You can now assign the test point to the class with the highest probability and then compare that with the true class label.