Inf 2B Coursework-2
Task-2 Report
Bernoulli naive Bayes classification

UUN: s1616497

April 13, 2018

Abstract

The aim of task one is to use Bernoulli naive Bayes classification in order to classify a dataset of images into one of 26 different classes: representing the letters of the English alphabet.

1 Code Description

1.1 File 1: $my\_bnb\_classify.py$

Parameter list:
$Xtrn$, $Ctrn$, $Xtst$, $threshold$

Return value:
$Cpreds$: Nx1 matrix of predicted classes of $Xtst$ matrix feature vectors

In order to proceed with the Bernoulli naive Bayes classification process, the first step is to *Binarize* the training and test vectors. *np.where*(), a *vectorized* function was used to do so.
Next, more *vectorization* was done through the use of *np.compress*() for extracting the feature vectors belonging to a particular class. Also, *np.sum*() was used to calculate the *mean vectors*. *np.log*() was used to calculate the log *probabilities*.

The main area where the code was sped up is in the calculation of the *likelihood* for each test vector for each class. This was done through the use of matrix multiplication, as the *vectorized* version of *log likelihood* is literally:

Formulae Used:

Matrix of log likelihood $= Btst$ x $probs1.T + (1 - Btst)$ x $probs0.T$

Where:

- $Btst = Binarized$ matrix of test vectors
- $probs1$ = matrix of probability of one occurring at a particular dimension of a particular class of feature vectors.
- $probs0 = 1$ - $probs1$

```
probs1[probs1 == 0] = 1.0e-10
probs0[probs0 == 0] = 1.0e-10
probs1 = np.log(probs1)
probs0 = np.log(probs0)
ci = np.dot(Btst, probs1.T) + np.dot((1-Btst), probs0.T)
Cpreds = np.argmax(ci, axis=1).reshape((test_size, 1))
```

Figure 1: Snippet of log posterior probability calculation.

The datatypes of the elements in the matrices were changed to float 32 which also helped speed up the calculation procedure.
argmax() was *vectorized* to reduce time taken to calculate the predicted classes of the test feature vectors.

1.2 File 2: $my\_confusion.py$

Parameter list:

- $Ctrues$: The correct classes of the test vectors(Shape: Nx1)
- $Cpreds$: The predicted classes of the test vectors(Shape Nx1)

Return value:

- CM : K-by-K confusion matrix, where $CM[i][j]$ is the number of samples whose target is the $i'th$ class that was classified as j. K is the number of classes, which is 26 for the data set.
- acc : The ratio of right predictions to total predications

There was no need of vectorization in this case. Just a for-loop iterating over all the predications was used.

1.3 File 3: $my\_bnb\_system.py$

The function *time.clock*() was used which is a part of the package *time*.
Used *scipy.io.savemat*() to store the $Confusion\ Matrix$ in the respective files.
matplotlib.pyplot was used to create a graph showing how the accuracy varies with increase in threshold value.

2 Values Obtained

Prediction Size	Number of Errors	Accuracy
7800	2890	0.629487179487

.

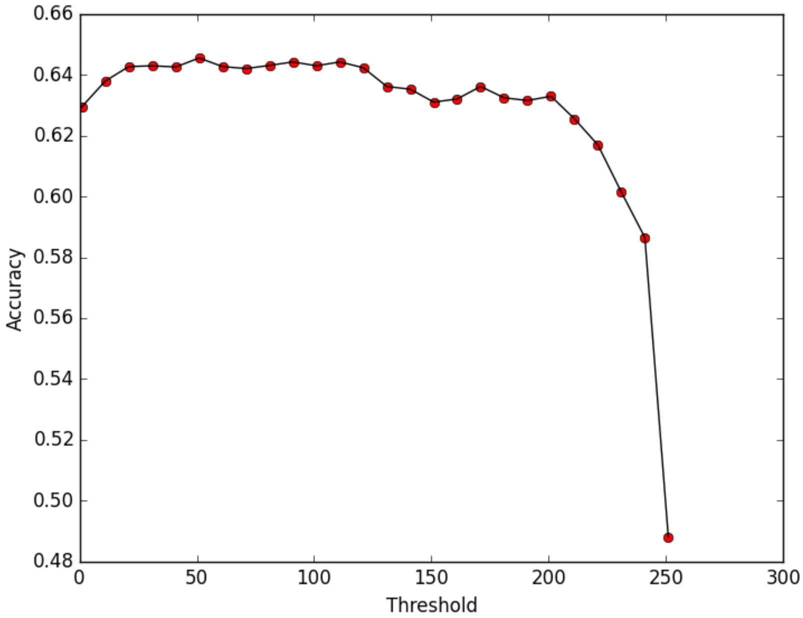

Figure 2: Graph showing the varying of accuracy with increase in threshold value.

Time taken for $my\_knn\_classify.py$: 4.87 seconds for threshold value of 1.0 .
At threshold = 51, we get a maximum accuracy of 0.645512820513.
It is quite clear that the accuracy value first increases to a peak and eventually starts to drop with the increase in threshold. There is a steep drop as threshold increases above 200. At this point, the binarized vectors from the training data might have lost a lot of information specific to the class that the feature vector belongs to. It may also be the case that a lighter shade of an image may be classified as something else simply because the threshold value is too high.