Homework 1 Problem 1

Part A:

The following requirement asked us to implement a naive Bayes algorithm from scratch. The data set PIMA Indian dataset has been used to carry out the analysis.

The data set was imported in R and sliced into feature set and class set. Following libraries were use:

-library(klaR)

-library(caret)

-library(e1071)

A function "do_bayes_classification" contains the actual logic of doing naive Bayes classification. We start of by creating data partition with 80% training data and rest 20% as test set. We go forward by splitting the training data further into positive subset and negative subset using the training labels we had. The same sub setting was done for the test data as well. We now calculate the mean and standard deviation of both positive and negative training subset, which was used to perform normalization by subtracting each data point with its mean and then dividing by its standard deviation. After we are done with this we now calculate the posterior probability by multiplying density likelihood function and the prior probability. The prior was calculated for both positive and negative by taking the no of samples in respective class over total no of observation.

After calculating posterior for both classes we compare their values and see which one is greater. We then check these predicted label with the actual label and calculate the accuracy based on that.

For a tenfold test following is the accuracy reported on the 20% that was held out for evaluation.:

Output:

[1] 0.7647059, 0.7712418 ,0.7712418 ,0.7385621 ,0.7973856 ,0.7581699 ,0.7450980 ,0.7516340 , 0.7843137 ,0.8104575

with the maximum being 81.04%

Part B

The 0 values for columns 3,4,6,8 were replaced by NA and the data was fed back to the above method implemented in part A. For a tenfold test following is the accuracy reported on the 20% that was held out for evaluation:

Output:

[1] 0.7320261 ,0.7647059 ,0.7647059 ,0.7516340 ,0.7777778 ,0.7124183 ,0.7843137 ,0.7385621 ,0.7385621 ,0.7320261

with the maximum being 78.4%

There might not be a noticeable change and the reason being that the Bayes classifier will simply ignore the na values and evaluate on the features that have values.

Part C:

Here we used the caret and klaR packages to build a naive Bayes classifier assuming that no attribute has a missing value. The data was sliced into 80% training and 20% test. Using cross-validation with 10 folds following accuracy was reported on the 20% that was held out for evaluation:

Confusion Matrix and Statistics

		Reference	
Prediction		0	1
	0	93	13
	1	19	28

Accuracy : 0.7908

95% CI : (0.7178, 0.8523)

No Information Rate : 0.732

P-Value [Acc > NIR] : 0.05769

Kappa : 0.4905

Mcnemar's Test P-Value : 0.37676

Sensitivity : 0.8304

Specificity : 0.6829

Pos Pred Value : 0.8774

Neg Pred Value : 0.5957

Prevalence : 0.7320

Detection Rate : 0.6078

Detection Prevalence : 0.6928

Balanced Accuracy : 0.7566

'Positive' Class : 0

Part D:

Here we used the svmLight package to perform Support vector machine classification on the same data set with 80-20 split. The predictions were compared with the actual label and following accuracy was reported on the 20% that was held out for evaluation:

Output:

[1] 0.7908497

Note in order to run the svm classifier the binaries for svmlight had to be downloaded and placed in the correct path.

Note

In order to run the script, you need to install the above mentioned libraries and specify the path to the data and the svmlight binaries.

Citation

Part of the script has been referenced from the code that Professor shared with the class. The code was adapted and improvised further to achieve a high accuracy, Which can be seen from the output of the classifier created from scratch giving me accuracy of (81.04%) whereas the caret package model gave me 79.8% accuracy.

****

Homework 1 Problem 2

Part A:

Python has been used for this part of assignment. The data used was MNIST image dataset and it was imported into the code using the sklearn utility dataset = datasets.fetch_mldata("MNIST Original").

Following libraries are needed for the script to run:

-pandas, numpy, scipy, matplotlib, sklearn, PIL

-The data is split in 80-20 ratio.

-The images in the original dataset were represented by 28*28 pixel.

-"rescale_strech_image" methods does the job of identifying the non-zero pixels in the image, cropping all the background zero pixels, resizing it to 20*20 and doing the thresholding by converting every pixel to either 0 or 1. The output of this method is an numpy array that contains each image in a 20*20 bounded stretched form.

-PIL library was used to view the original and modified images by converting them back and forth to Image array object.

-The way stretching was done was that I first represented each image in 28*28 numpy array and tried finding out the indexes of non-zero rows and column. Using the min, max indexes of rows and column I represented the image and resizes it to 20*20 numpy array.

- I then initialized the naive Bayes model using sklearn for both Bernoulli and Gaussian distribution and passed in my respective training and test data:

Following are the output from my script:

Accuracy	**Bernoulli**	**Gaussian**
untouched images	54.82%	83.31%
stretched bounding box	82.12%	80.9%

Reasoning: - In case of untouched images the Gaussian gave a good score as the distribution of pixels nearly follow a normal distribution and worked well with this dataset on the other hand Bernoulli expects the features to be binary but in this case it's not hence it shows up with low score.

For Stretched and bounded images we have throws away the background pixels and threshold each pixel to be between 0 and 1 and with less no of features Bernoulli perform good as compared to Gaussian but Gaussian still has a reasonably high score as the underlying distribution of pixels doesn't change with thresholding.

Part B:

The same setup was used to perform random forest by using the methods created above to create a bounded stretched image.

Following are the output from my script:

Untouched raw pixels:

	depth = 4	depth = 8	depth =16
#trees = 10	74.85%	89.98%	94.65%
#trees = 20	77.37%	91.62%	95.97%
#trees = 30	78.44%	92.13%	96.38%

Stretched bounding box:

	depth = 4	depth = 8	depth = 16
#trees = 10	74.4%	91.14%	95.18%
#trees = 20	78.71%	92.11%	96.23%
#trees = 30	80.01%	92.4%	96.53%

Reasoning: - For untouched images decision tree will work best with maximum depth as the no of decision to be taken will be more helping each prominent features to be selected.

For untouched images we see a slight increase at different levels but not major changes since the random forest would have already identified the promising features from the untouched so giving a subset of those promising feature will not help much.

Citation

Some of the useful numpy arrays operations have been looked on for in stack overflow.