Project overview:

Leader: Kevin Fortier

Team: Ryan Ferrin, Brian Valentino

Summary:

Our work strategy was as follows- we assigned one problem per teammate and each teammate was also given a problem to review. This was when we had four people. Then one member dropped the class, apparently. So we modified this strategy slightly, Kevin did problems 1.1 and 1.4, and Ryan did problem 1.2 and Brian did problem 1.3. We pushed our work to a single github repo with our own development branch. Then we make pull requests with code that was ready for review. The assigned reviewer would then look over the code and merge it to the master branch if all looked good.

All in all, we each contributed fairly and performed our assigned tasks- we looked over each others code and actively communicated when issues came up.

Problem 1.1:

An end-to-end machine learning pipeline was developed for analyzing the Boston housing dataset. This involved formatting the data (including splitting into a test and training set), performing a rough analysis of the data, transforming the data, fitting a few machine learning models to the data, performing predictions with each model, selecting the best model and finally evaluating the final model on test data that the model had not yet seen.

I explain each of these steps in more detail in the jupyter notebook.

Problem 1.2:  
  
In problem 1.2 several methods and models were used to predict handwritten digits in the mnist data set. The first method used is the “K Nearest Neighbors”(KNN), this method finds the data points of the training data set that are closest to a test data point using euclidean distance calculations. We were asked to run mnistKNNdemo (from pmtk3) the demo was written in MATLAB, it was recreated in python using sci-kit learn libraries. In the demo the first 10000 images of the mnist training data were used to classify the first 1000 images of the test set. We selected k=5 to find 5 data points for classification. The resulting miss-classifications rate was 8.4% and a computation time of 14.13 seconds.

For part 2 we created an approximate nearest neighbors method using the FLANN libraries provided. The “Approximate Nearest Neighbors” method is a variation of the KNN method but attempts to reduce the computation time to predict by using a “kmeans” cluster for classification instead of the entire data set. The ‘kmeans’ establish k clusters by minimizing the variances of the k clusters. The expected result is a faster algorithm that might sacrifice an acceptable amount of accuracy. In our implementation we kept the data set sizes and the number of neighbors the same as those used for KNN. The resulting miss-classifications rate was 8.0% and a computation time of 3.5 seconds. As expected the computation time was much faster than KNN however the accuracy was slightly improved.

On part 3 we were asked to test a multinomial logistic regression model. This model fits a logistic regression curve for each case, in the case of the mnist data sets there are 10 possible cases. For each case the probability of that being the right classification is computed the predictor selects the highest probability as the prediction for a new data point. The resulting miss-classifications rate was 15.2% and a computation time of 321.84 seconds

|  |  |  |  |
| --- | --- | --- | --- |
|  | KNN | FLANN | Logistic Regression |
| misclassification rate | 8.4% | 8.0% | 15.2% |
| computation time | 14.13 sec | 3.5 sec | 321.84 sec |

Problem 1.3:  
Running the python script for this problem (python problem\_1\_3.py) in this repository a number of times we get the following results:

|  |  |
| --- | --- |
| Run 1 | Run 2 |
| Linear Regression  MSE without CV: 3449.19  MSE with CV: 3000.38 | Linear Regression  MSE without CV: 2869.01  MSE with CV: 3000.38 |
| Ridge Regression  MSE without CV: 3831.89  MSE with CV: 3364.53 | Ridge Regression  MSE without CV: 3689.43  MSE with CV: 3364.53 |
| Bayesian Ridge Regression  MSE without CV: 3831.89  MSE with CV: 2998.91 | Bayesian Ridge Regression  MSE without CV: 3689.43  MSE with CV: 2998.91 |
| K-Nearest Neighbors  MSE without CV: 4348.41  MSE with CV: 3764.74 | K-Nearest Neighbors  MSE without CV: 3287.32  MSE with CV: 3764.74 |

There are a few interesting results to discuss with these runs. The first thing to note is that all of the models without cross validation perform differently in run 1 vs run 2. This is expected since only splitting the data once, it could be more or less predictive of the testing data depending on how the data is divided up. The second thing to note is that the cross-validation models are consistent between runs. This is expected since regardless of how we split the data, by iterating over each split and randomly assigning it test or train, we would converge to the same model each time. This leads to the next point, in general we would expect the cross-validated models to perform better on any subsequent data, even if it has a higher error rate for a particular test data set. This is because while one particular split may be predictive for the test data set, it does not mean that it would be a good fit for the subsequent data.

In terms of the individual model type performance we see that all of the linear models perform better than the KNN model, and within the linear models the Bayesian model performs the best of the three. This is unsurprising. Plotting the individual features of dataset we can see that many of the features have a linear relationship. So we would exect that the linear models would predict better than the clustering the KNN model performs. It is also unsurprising that the Bayesian model performs better than both the Ridge and OLS regressors. This is due to the fact that not all of the features have a linear relationship. It is also interesting to note that the OLS model performs better than the Ridge model, but again this seems to be because the data is already normalized and centered around 0. There are not any ridges present in the data, so the additional penalty makes the model perform slightly worse than the OLS model.

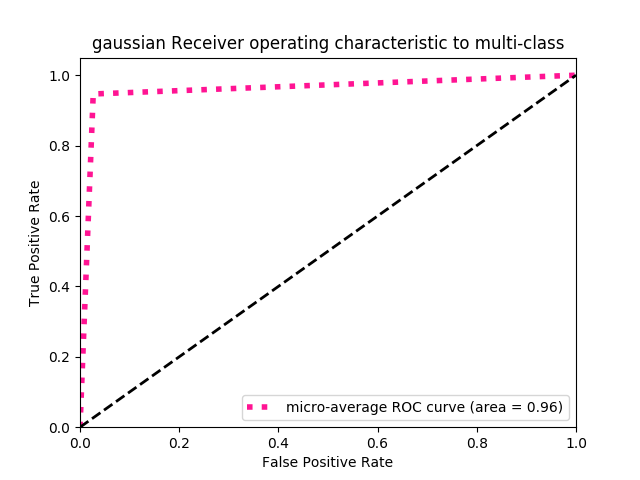
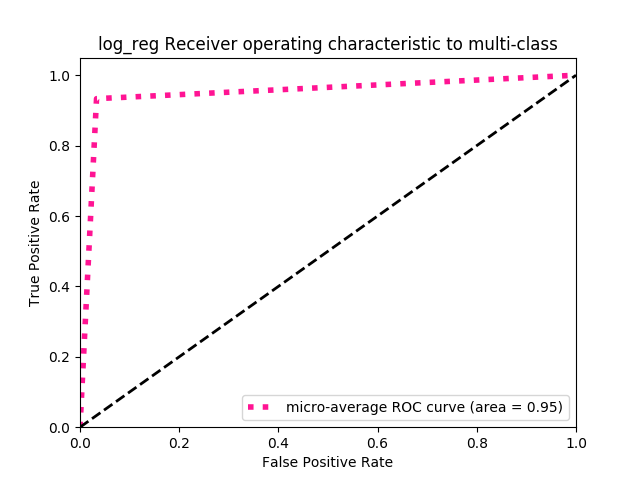
Problem 1.4:

In problem 1.4 we are going to train three different models on the iris dataset and evaluate their performance using micro averaged precision recall and a receiver operating characteristic curve. The models that we will be fitting the data to are a gaussian classifier, linear discriminant analysis and logistic regression classifier. After training each of these models, we will perform predictions with the resulting model and analyze the results of those predictions when compared to the actual target value.

First, we compute a confusion matrix for each of the models. This shows the number of correct and incorrect classifications per class. These matrices are printed in the console when the program is run. All of the models perform within 3-5% error, and each model is able to predict the first class with one hundred percent accuracy and the third class with the least accuracy. All in all, the linear discriminant analysis performs the best, but it is not clear if this information is meaningful in anyway, after all, the model only slightly outperforms the other models.

Next, we look at the roc and micro averaged precision curves. In general, the roc curve is going to allow us to compare the rates of true positive to false positive. This curve gives us a way to make comparisons between models. The essence of this curve is the model’s skill, or the degree to which the model classifies a false answer as correct and a correct answer as false. Looking at the three curves for the models we are comparing, they all display a curve that steeply rises to the top left, this is an indicator of a high skil model- high true positive and low false positive. The three models all look very similar and this is an indicator the the high scores seen on the confusion matrix is in fact telling us that the models are able to learn the feature to label mapping well.

See next page for charts.



Next, we analyze our models based on their precision. The micro averaged precision recall curve is going to show how well our models select the positive label. This curve is especially useful when there are imbalances in the dataset. In general, the larger the area is under this curve, the more accurate the model is. Looking below at our three curves, we can see that the models have high precision and therefore we have even greater confidence in the ability of our models to map features to labels for this particular dataset. It is still unclear which model is exactly the best for the job, that also has to do with how well our models generalize beyond this dataset.

See Next Page for charts.

In conclusion, each of the models performs well on the dataset. Further analysis of the models could help determine a best model. In particular, putting multiple model into a production environment and seeing which performs best over time could help determine a model to focus on in the future.

