Classification Experiment

1. **(5 points) Pick some data suitable for classification learning that we haven't already used for this class. In your write-up briefly summarize the data set including the predictive columns and the target column(s). Briefly describe any data preparation you performed (and say why you did it).**

For the classification experiment, I picked UJIIndoorLoc from this [website](http://archive.ics.uci.edu/ml/datasets/UJIIndoorLoc). I chose this dataset because it was really close to what we are doing for the research project. In fact, this dataset has given me an idea of how I should be recording the Wi-Fi values for the machine learning piece of the project and how to deal with the missing values as well.

The data had no missing values, so there was no need for data cleaning. It has 21048 instances and 529 attributes. This data could be used for both classification and regression, but instead I only did classification for this assignment.

Among the attributes, 520 out of them which are Wi-Fi readings. The target columns for classification are ‘Floor’ and ‘BuildingID’. I only used the ‘Floor’ attribute for this experiment. The target for regression are ‘Longitude’ and ‘Latitude’. I did not do any experiment on these target columns.

In addition, there are a few other columns that are not necessary for machine learning but are there for a detailed information. For instance, the ‘UserID’, ‘PhoneID’, ‘Timestamp’, ‘SpaceID’, and ‘RelativePosition’.

Moreover, the data was already split into testing data (validationData.csv) and training data (trainingData.csv). They were both in their own CSV file.

1. **(6 points) Using either the scikit-learnSGDClassifier or Perceptron algorithm, perform an experiment to determine how well the classifier works on your data set. In your write-up, discuss the results. You will need to convince me that you have selected appropriate parameters for learning (e.g., learning rate, number of epochs). You will probably need to include a plot (or plots) of some sort to demonstrate this. Any data points you include should be averages over several runs (say 20?) with different training/testing splits to ensure that the results aren't simply due to luck.**

****

I used the SGDClassifier library to carry out this experiment. I chose n\_jobs to be -1 because given that there were 21048 instances, I wanted to make sure that computation is not a hindering factor in the algorithm. I chose Alpha to be 0.5. I experimented with the default Alpha (0.0001), and Alpha (0.5). The number of iterations was 5 at this point. The average accuracy for Alpha (0.5) was 0.6508010801083199, and for Alpha (0.0001) was 0.6508010801083199. So, I chose Alpha (0.5).

I ran the algorithm at 5, 10, 20 and 40 iterations. For each number of iterations, I ran the algorithm 25 times to get an average accuracy. I found that the algorithm performs best when the number of iterations is 20. The average accuracy at that point was about 0.6737, which was the highest for this algorithm.

1. **(6 points) Using a scikit-learn implementation of another classifier algorithm (e.g., k-Nearest-Neighbor, Random Forest, or Multilayer Perceptron), perform an experiment to determine how well the classifier works on your data set. In your write-up, discuss the results. You will need to convince me that you have selected appropriate parameters for learning (e.g., the number of neighbors, number of trees, etc.). You will probably need to include a plot (or plots) of some sort to demonstrate this. Any data points you include should be averages over several runs with different training/testing splits to ensure that the results aren't simply due to luck.**

****

I used RandomForestClassifier for this experiment. By using the default parameters, the classifier has an average accuracy of about 0.8356. This is already better than the SGDClassifier experiment, however, I wanted to see what it would do when I increase the estimators.

I increased the estimators to 20, then to 40, then to 80 and lastly to 200. I found that the accuracies start to saturate when the estimators are set to 200. So about 200 estimators are the best estimators to use for this algorithm. For this experiment I ran the algorithm about 20 times each, and then I calculated the average accuracy. The graph above shows the results.

1. **(3 points) Discuss conclusions in your write-up. Which algorithm performed better? Give reasons why you think that's the case.**

The Random Forest Classifier performed better for the classifier experiment. By using the default values, the algorithm gave a much better average accuracy than the SGDClassifier. The SGDClassifier reached 0.674 as its highest. However, the Random Forest Classifier reached about 0.874 as its highest.

Regression Experiment

1. **(5 points) Pick some data suitable for regression learning that we haven't already used for this class. In your write-up briefly summarize the data set including the predictive columns and the target column(s). Briefly describe any data preparation you performed (and say why you did it).**

For this experiment, I picked the Red Wine Quality from this [website](http://archive.ics.uci.edu/ml/datasets/Wine+Quality). I chose this dataset because wine tasting and wine quality fascinate me. It is hard to guess the quality of a wine when you are tasting it, except if you have tasted a lot of wines before. This dataset is fairly small. It has about 4898 instances.

This dataset has no missing values, so there was no need for data cleaning. However, when I downloaded the dataset, the dataset was in semicolon-separated values. I had to convert that into comma-separated values in Microsoft Excel so that I can read it from the Python code.

This dataset contains all real-value attributes. There are 12 attributes, namely ‘fixed acidity’, ‘volatile acidity’, ‘citric acid’, ‘residual sugar’, ‘chlorides’, ‘free sulfur dioxide’, ‘total sulfur dioxide’, ‘density’, ‘pH’, ‘sulphates’, ‘alcohol’ and ‘quality’. The ‘quality’ is the target column.

In the Python code, I had to split the dataset into training and testing data. I used 20 percent of the data for testing.

1. **(6 points) Using the scikit-learn SGDRegressor algorithm, perform an experiment to determine how well the classifier works on your data set. In your write-up, discuss the results. You will need to convince me that you have selected appropriate parameters for learning (e.g., learning rate, number of epochs). You will probably need to include a plot (or plots) of some sort to demonstrate this. Any data points you include should be averages over several runs with different training/testing splits to ensure that the results aren't simply due to luck.**

****

I used SGDRegressor for this experiment. I ran the experiment for 5, 10, 20, 40, 100, and 200 iterations. For each, I ran the code 20 times (aka runs) to get an average Mean Squared Error. I first calculated the mean squared error for each run using the mean\_squared\_error function in SKLearn. I then averaged the total mean squared error over 20.

For this experiment, I found that the least average Mean Squared Error occurs when the number of iterations is 100. If you use less than 100, you might end up with a really big Mean Squared Error, especially when the number of iterations is too small (see graph above). On the other hand, the more iterations you use, the more the Mean Squared Error increases. This is why 100 is the right number of iterations for this experiment.

1. **(6 points) Using a scikit-learn implementation of another regressor algorithm (e.g., k-Nearest-Neighbor, Random Forest, or MultilayerPerceptron), perform an experiment to determine how well the regressor works on your data set. In your write-up, discuss the results. You will need to convince me that you have selected appropriate parameters for learning (e.g., the number of neighbors, number of trees, etc.). You will probably need to include a plot (or plots) of some sort to demonstrate this. Any data points you include should be averages over several runs with different training/testing splits to ensure that the results aren't simply due to luck.**



I used RandomForestRegressor for this experiment. By using the default settings for this regressor, the average Mean Square fairly smaller than the one in SGDRegressor.

In the experiment, I ran the regressor for 10, 20, 40, 100, 200, and 400 estimators. As you can see in the graph, the regressor performs best when the number of estimators is larger than or equal to 400. The average Mean Squared Error decreases when you use more estimators. The least average Mean Squared Error occurred when number of estimators was 400 for this experiment.

Again, for this experiment, I ran the code 20 times for each number of estimators that I used. Then I calculated the average Mean Squared Error. Finally, I plotted the graph above.

1. **(3 points) Discuss conclusions in your write-up. Which algorithm performed better? Give reasons why you think that's the case.**

For the regressor, the Random Forest Regressor performed better than the SGDRegressor. The SGDRegressor has a large average Mean Squared Error (about 3.3997e+23) and the Random Forest Regressor has a very small average Mean Squared Error (about 3.575).