CLASSIFICATION PROBLEM

Stock market prediction is the act of trying, to determine the future value of a company's stock or other financial instrument traded on a financial exchange .Trading in the stock market has gained huge attractiveness to me as a means through which, one can obtain vast profits while applying big data algorithm analysis. Attempting to profitably and precisely predict the financial market has long engrossed the interests of me and using the 5 supervised learning algorithm to accurate stock market predictions are important.

STOCK MAKET DATA

In the proposed trend prediction system, data from stock market is considered for the study. Datasets from four companies are considered for the study: GOOG,AAPL data time from 01/01/2010 to 12/31/2010 is considered for the purpose. All the datasets are tested against proposed system and make predictions in the last step of the system.

RESULTS

A decision tree learning algorithm analyzes variations in input variables of training data in relation to what is chosen as the dependent variable, which is the variable to be classified for the test object, to predict the value of the dependent variable in each of the test objects given their input variables. Justly, the way the variations in input variables interact to approximate a value for the desired classification can be visualized in terms of a biological tree (though a flow chart also works for this purpose). Here, the trunk and branches the tree refer to input variables and the branches end with a leaf that represents the dependent variable. The algorithm chooses one of the input variables a the trunk of the tree and, based on this value, chooses to go down one of a number of branches. This split into branches is known as a node. One of these branches may lead directly to classification of the dependent variable in the case where this input variable has been found to consistently to lead to that specific classification independent of the other input variables. If this is not the case, the branch will split and the branch to be continued down will depend on one of the other input variables. This splitting of branches continues based on values of input variables that are found to best split the data until a path is found that leads to the dependent variable with statistical significance across the training data. It is this tree that is then used in predicting the classification (dependent variable) of the test objects.

It should be noted that there is an issue with producing an optimal tree. The input variable rules upon which branching decisions are made may be locally optimal but not necessarily optimal in producing the most consistently accurate classification for the dependent variable. A decision tree may also be overly complex, over-fitting itself to the training data such that the relationships between input variables is not general enough to function well for data outside of the training data. To solve this problem, pruning is implemented, which reduces the size of the trees by removing sections found to not to provide much benefit in classification. A programmer does this by attempting to choose the optimal number of nodes for the tree that results in the least amount of error. The problem is that there is no way to know if the addition of another node will have a notable impact in reducing error. Consequently, a common strategy is to increase the number of nodes until the number of data sets falling within each node is small; then use pruning to remove nodes that do not produce additional predictive power.

The artificial neural network (ANN) algorithm is inspired by the workings of biological neural networks. The algorithm works through systems of interconnected "neurons" (also referred to as nodes) which exchange information amongst each other. The information being exchanged from one neuron to another is information that has been weighted by the "synapse" (interconnect) between them. Consequently, the first layer of "neurons" is the input information and the output from the last layer of

neurons is that information after it has passed through a series of interconnected weighing functions. These weights are improved through experience through a learning algorithm, meaning the algorithm is adaptive to input information and capable of learning. The output can be seen as a non-linear weighted sum that has been but through some predetermined function (the activation function). A multitude of ANNs can be produced based on different neuron interconnection patterns, learning algorithms for updating weights of interconnects, and activation function. The activation function is the function chosen for the algorithm that converts a neuron's weighted input to its output. To put it mathematically, the output is a function of the non-linear sum of weighted outputs from each neuron layer. This sum is non-linear because the weighing functions of each interconnection are independent of each other. The function in which the sum of these weighted outputs fit is predefined by the programmer and is chosen based on what is seen as most appropriate for the desired task of the algorithm.

The methods in which the weighing functions of each interconnect update/learn fall into three classes: supervised learning, unsupervised learning, and reinforcement learning. All of these function by attempting to reduce some cost function that relates the difference between outputs generated by the neural network and the actual values, thereby reducing error and improving he predictive power of the system. The difference between these methods of learning is where this cost function comes from and how training is performed. In supervised learning, input and output values representative of the entire system are given. The algorithm produces a function of the input variables then compares the result to the known output variables. The cost function in this method is the error between the predicted result and the actual result. This error is then used to update the weighting functions in a cyclical manner until the result is optimized, meaning the cost function is minimized. This method of learning is useful when the programmer desires the predicting function to replicate the relationship between the training data for the test data. This method is specifically useful in classification and regression type problems. In unsupervised learning, data is provided as well as a cost function which is a function of the system inputs and outputs. The algorithm learns by optimizing a function of the input variables which minimizes the specified cost function. The costs produced from each iteration of training is used to update the weights, which are continually updated until minimum cost is achieved. This method of learning is useful when the cost function is known but the training data available may not be representative of the entire system. This is useful in estimation type problems. In reinforcement learning, no previous training data is given. Instead, training occurs through interaction. For each input, the algorithm produces an output and a cost based on arbitrary functions. This cost is then used to optimize the function used to produce the output for the next iteration. With increasing iterations, the predicting capability works to improve itself continually. This form of learning is useful in sequential decision making tasks such as games and control problems.

Over-training can be an issue with ANNs when the desire that it is general enough to function for unseen data. This occurs when the capacity (information stored in the network and its complexity) notably surpasses the parameters needed to produce an accurate prediction. One method to inhibit this is cross-validation. For this purpose, known data is partitioned into a training and a test data set. This is performed by taking a model based on a training data set and using it to make predictions on an independent test data set. This is performed multiple times on varied partitions of the data into training and test data. The validation of the ANN is determined from the average predicting accuracy over these cycles of testing.

In essence, boosting is an algorithm based upon the combination of a set of weak learner algorithms to produce a strong learner. Each weak learner is arbitrarily better than random guessing when attempting to produce an accurate prediction. Each weak learner is first optimized to its capacity for accuracy before being added to the boosting algorithm. Once they are added, each is typically weighted

according to its capacity for for accuracy. The data is then re-weighted with data producing accurate classification losing weight and data with miss-classification gaining weight. Consequently, the subsequent weak learner to be added will concentrate more on its ability to predict the classifications the previous weak learners failed to predict. Various weak learners can be used in this algorithm, including decision branch algorithms. If the boosting algorithm is not adaptive with future data, it does not make full use of the weak learners. Many adaptive boosting algorithms make use of learning by adapting the weights to minimize a convex type cost function. However, this method of learning has been shown to not work well in real life examples in which there is random classification noise. This is because the learning part of the algorithm puts a lot of effort into rectifying this noise with the surrounding data, drastically decreasing the predicting capability of the model that is consequently used. This is not found to be an issue when the boosting algorithm is based upon branch learning algorithms.

Support Vector Machines (SVM) algorithms are designed specifically for classification and regression. They are supervised learning algorithms that make use of contained learning algorithms to analyze data a recognize patterns. It is a supervised learning algorithm in that data is separated according to programmer defined categories. The algorithm then outputs an optimal hyperplane or set of hyperplanes in high to infinite dimensions which categorize new data. A hyperplane is a dividing plane made up of one dimension less than the dimensions of the classification. For example, if categorization is performed in three dimensions, the hyperplane would divide it with a 2 dimensional plane. If categorization is performed in two dimensions, the hyperplane would be a line dividing that space. An optimal hyperplane is one which divides the categories while also being furthest from any training data point. The larger the margin between hyperplanes and training data, the lower the generalization error of the algorithm. Often, classification categories are not linearly separable. To make separation easier, the hyperplanes are then mapped in higher dimensional space. The hyperplanes in these higher dimensional spaces are defined by points within that space whose dot product with a vector in that space is constant. The computational power taken up by these dot products is reduced through the use of kernel functions based on the variables in the original space. There are various kernels and it is the programmer's responsibility to pick one which works best for the system. When there exists no clear division between all data points, a soft margin is used to divide the data as much as possible by dividing it such that it maintains the largest margin between as many data points as possible. This, in turn, allows the algorithm to function well even when outliers and mislabeled data may be present.

The effectiveness of an SVM depends on the kernel, the kernel's parameters, and the soft margin factor chosen by the programmer to best suit the problem. Optimization of these parameters is typically performed using cross-validation, finally picking the parameters which produce the highest cross-validation accuracy. What these parameters actually mean in how they relate to the problem is difficult to understand. Another issue is that a solo SVM algorithm can only directly differentiate between two classes. Consequently, problems which constitute multiple classes must be divided into separate binary problems which are computable by SVMs. This can be done using binary classifiers that either analyze a class compared to the combination of other classes or compare pairs of classes.

The k-Nearest Neighbors (kNN) algorithm predicts a value/class of an object by analyzing those properties of a specified number of nearest neighbors. In other words, the algorithm analyzes the values of k number of values nearest the object. In classification, the class assigned for an object is based on a majority vote of the k number of neighbors specified; this effectively assigns a class based on a localized average. The training portion of this algorithm works such that data will be available which is called upon during testing to determine what the nearest neighbors are to an object. The computation becomes more intensive with larger amount of stored data that will be analyzed for nearness to objects

during testing. A higher number of neighbors (k) being analyzed means the class assigned to the object takes larger amount of data to produce its average classification, which is useful in that it reduces the effect of any outliers (noise) in the data. On the other hand, choosing a high number of neighbors for analysis may also diminish any localized trends in classification. Consequently, choosing an appropriate value for k that produces the highest level of accuracy depends on the nature of the data and may not be the same for different classes for the same object. The ideal value of k giving the highest accuracy for a classification increases as a function of increasing number of data points.

For training and testing, the dependent variables in my data were the "Open," "High," "Low," and "Close" values for the stocks of AAPL and GOOG for the 2010 year with "Adjusted Close" as the dependent variable to be predicted during testing. As can be seen from Figure 1 below, the GOOG stock prices are much more prone to erratic/noisy behavior than the APPL stock. Consequently, the learning algorithms used on these two stocks will be attempting to predict two very different types of behavior. I randomly partitioned the data from this year into training and testing data.

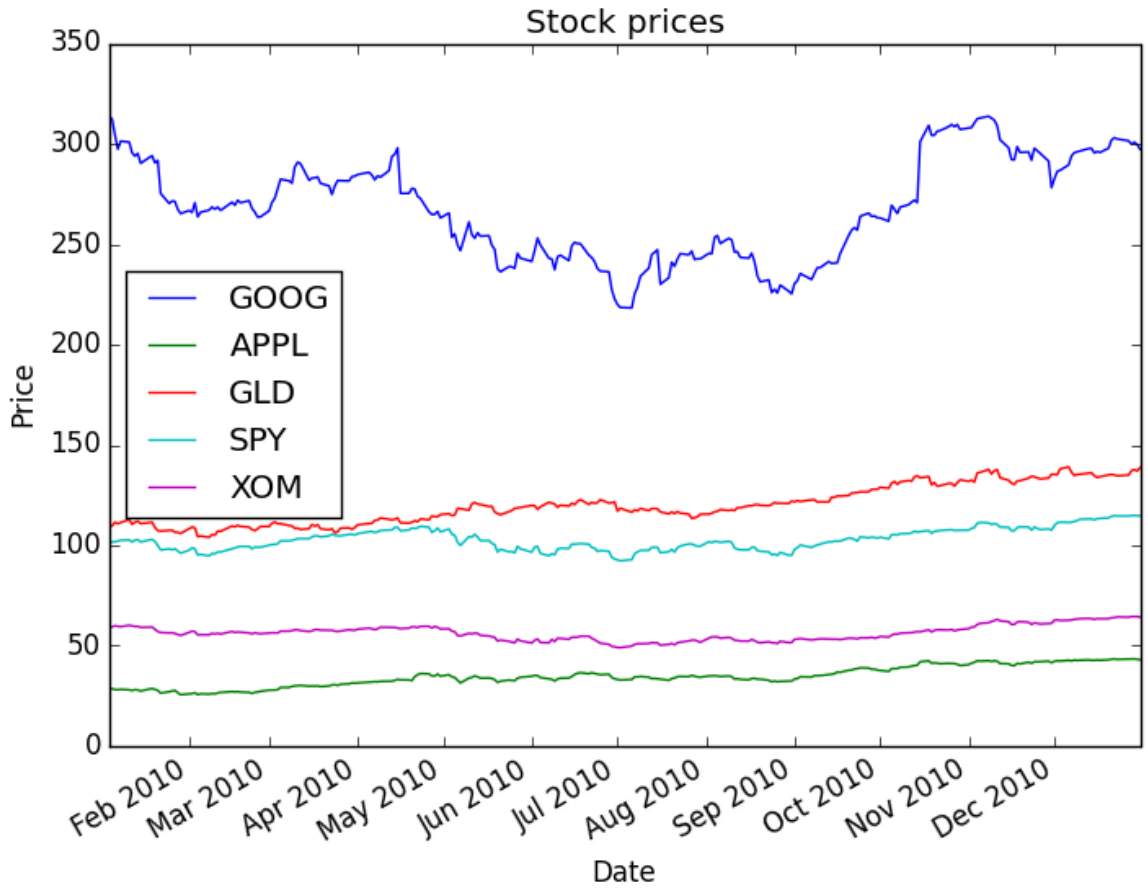

Figure 1: Sample stock prices during the 2010 year

For my ANN algorithm, I used some cross-validation to find that 50 nodes with 1000 iterations gave improved results. The algorithm I used utilizes a sigmoid function as its activation function.

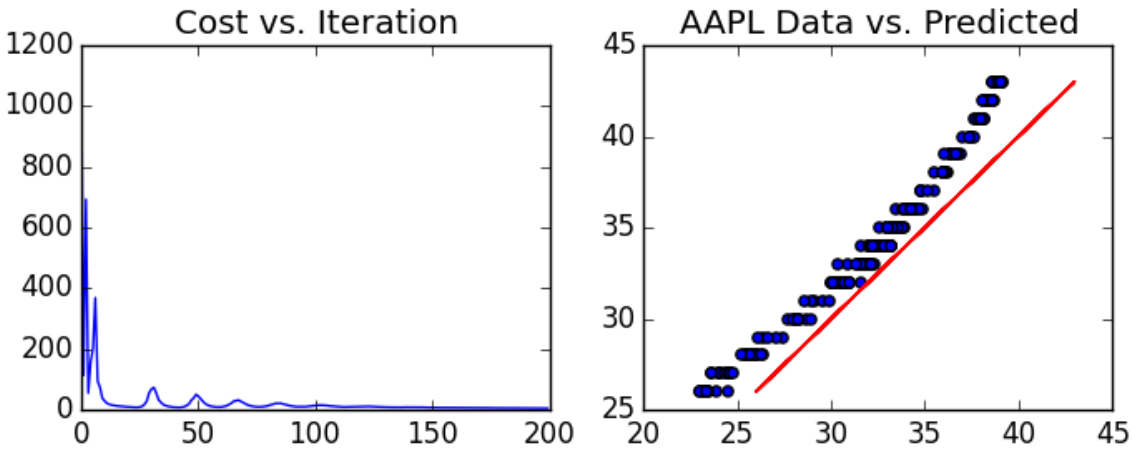

Figure 2: Decrease of cost with iterations and the predicted values of Adjusted Close compares to actual data for APPL stock

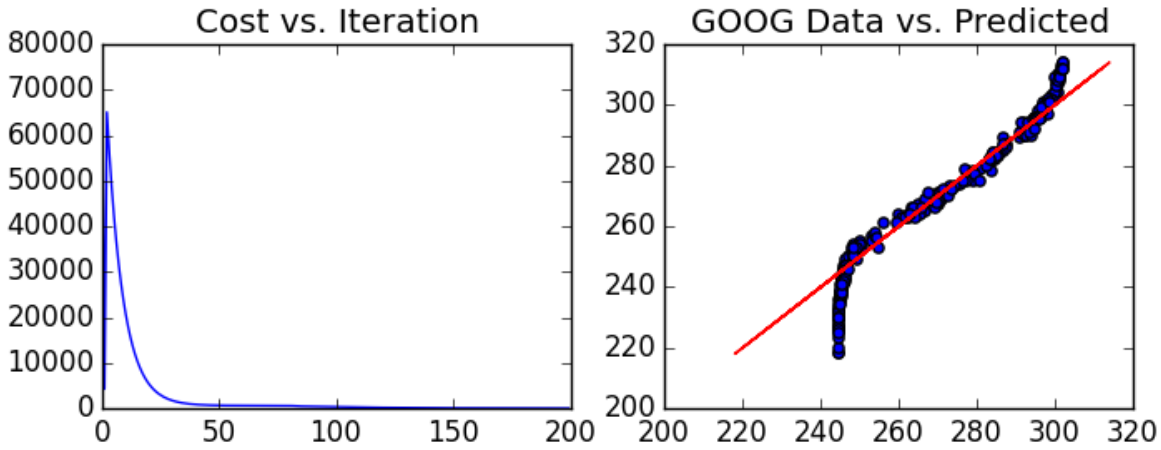

Figure 3: Decrease of cost with iterations and the predicted values of Adjusted Close compares to actual data for GOOG stock

```
>>>
AAPL ANN clock time 3.842954470842641
GOOG ANN clock time 3.8733027819210806
>>>
```
Figure 4: Clock time for ANN algorithm to run testing

From the predicted values compared to actual values, it can be seen that the algorithm does a fairly at predicting values close to the actual values of the dependent variable. For the APPL stock, it can be seen that accuracy is equally spread across the data. For the GOOG stock, the algorithm does very well at predicting values over a majority of the range of values but loses some of its predictive capability for the outlying data outside of the range of this majority. The Clock Times are fairly long due to the higher computational power needed to run this algorithm.

For my kNN algorithm, I chose to test k = 3, 5, and 7 to determine which worked to capture the different behavior seen between the APPL and GOOG stocks. See Figure 5 for results from running this algorithm. It can be seen that k = 5 produces the highest accuracy for the APPL stock; whereas, a k = 3 produces the highest accuracy for GOOG. This can be explained because a higher value of k brings in more nearest neighbors for analysis, which is useful for reducing noise but has the downfall of possibly

ignoring local trends. From these results, it can be inferred that the GOOG stock is more highly influenced by local trends than the AAPL stock. The accuracy values of the AAPL stock are all much higher than those found for the GOOG stock. This is likely to the difference between the stock behaviors, with GOOG behaving much more erratically than the APPL stock. The Clock Times for both are much faster than those seen for my ANN algorithm due to the kNN being much more simplistic and requiring less computational power. However, the ANN algorithm proved to be more accurate for predicting in both stocks.

```
>>>
AAPL
k = 3 Accuracy: 75.75757575757575%
k = 5 Accuracy: 83.544303797468 36%
AAPL kNN clock time 1.7841977176542795e-06
k = 7 Accuracy: 77.21518987341773%
GOOG
k = 3 Accuracy: 24.675324675324674%
k = 5 Accuracy: 19.696969696969695%
GOOG kNN clock time 2.2302471470725393e-06
k = 7 Accuracy: 18.30985915492958%
>>>
```

Figure 5: Accuracy and clock time of using kNN for determining the "Adjusted Close" for APPL and GOOG stocks

References

http://docs.opencv.org/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html
http://stackoverflow.com/questions/9014416/trying-to-write-my-own-neural-network-in-python
http://www.support-vector-machines.org/
http://scikit-learn.org/stable/modules/neighbors.html
http://scikit-learn.org/stable/modules/svm.html
http://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html#example-classification-plot-classifier-comparison-py
http://scikit-learn.org/stable/auto_examples/classification/plot_digits_classification.html#example-classification-plot-digits-classification-py
http://machinelearningmastery.com/tutorial-to-implement-k-nearest-neighbors-in-python-from-scratch/
http://scipy-lectures.github.io/advanced/scikit-learn/#k-nearest-neighbors-classifier
http://www.analyticsvidhya.com/blog/2015/08/common-machine-learning-algorithms/
https://www.quora.com/How-do-I-implement-a-simple-neural-network-from-scratch-in-Python
http://www.heatonresearch.com/node/1823
http://www.cs.cmu.edu/~schneide/tut5/node42.html
http://www.onlamp.com/lpt/a/6464
http://www.dkriesel.com/_media/science/neuronalenetze-en-zeta2-2col-dkrieselcom.pdf
http://www.analyticsvidhya.com/blog/2015/08/common-machine-learning-algorithms/
https://gist.github.com/mblondel/586753
http://pythonprogramming.net/support-vector-machine-svm-example-tutorial-scikit-learn-python/