Tyler Apostolico
GATECH - CS-7641 - Assignment 1

Supervised Learning

Data Set 1 Info:

Data set 1 deals with voter information and whether that user is classified as a democrat or republican. This data set is a difficult classification problem because there are multiple factors that could determine if someone is one or the other. In some classification examples I see a large lean toward one or the other political party (crime enforcement is highly a republican trait, aid to el salvador is a highly democratic, where immigration is fairly equal). Therefore, there is a lot of overlap in determinants. I believe this data set will be harder to classify because I can't even tell you concretely what makes someone a republican or democrat, since a democrat could be a large believe in higher crime enforcement, or a republican could believe in aid to el salvador. Data set 2 has more of a set relationship between data. Overall, I'm hoping there will be some examples here that will be very hard to classify because of this.

Data Set 2 info:

In contrast, data set 2 deals with a lower number of classifiers (9 vs 1). I chose one set of data that would be hard to classify (data set 1) and another that would hopefully be easier to classify (this one). This data set is about tic-tac-toe and in which situationsIshould win or lose. This is only the set of combinations where x goes first. It's an interesting data set because I are able to use it in order to develop a system of decisions to hopefully win tic-tac-toe every time. After analyzing this data, it seems that this one will likely be easier to classify because all of the attributes are split ~50/50 between positive and negative outcomes. In a tic-tac-toe game it's not necessarily consistent in which cases you win. As the data is separated into which player own which block of the square, it makes me curious how the algorithms will be able to figure out that three pieces in a row is the key to winning.

Splitting Data sets

I split these data sets into both training and testing sets manually. My methodology was to remove some unique/varying values from the data retrieved and see how well the algorithms performed given varying input and various classification, regression, and situations. I therefore took out data from the original data set and placed it in a separate file to load in to each algorithm. These files can be found in my code repository. In some cases it made more sense to do cross-validation in order to get data sets quicker and to get variance.

Decision Tree

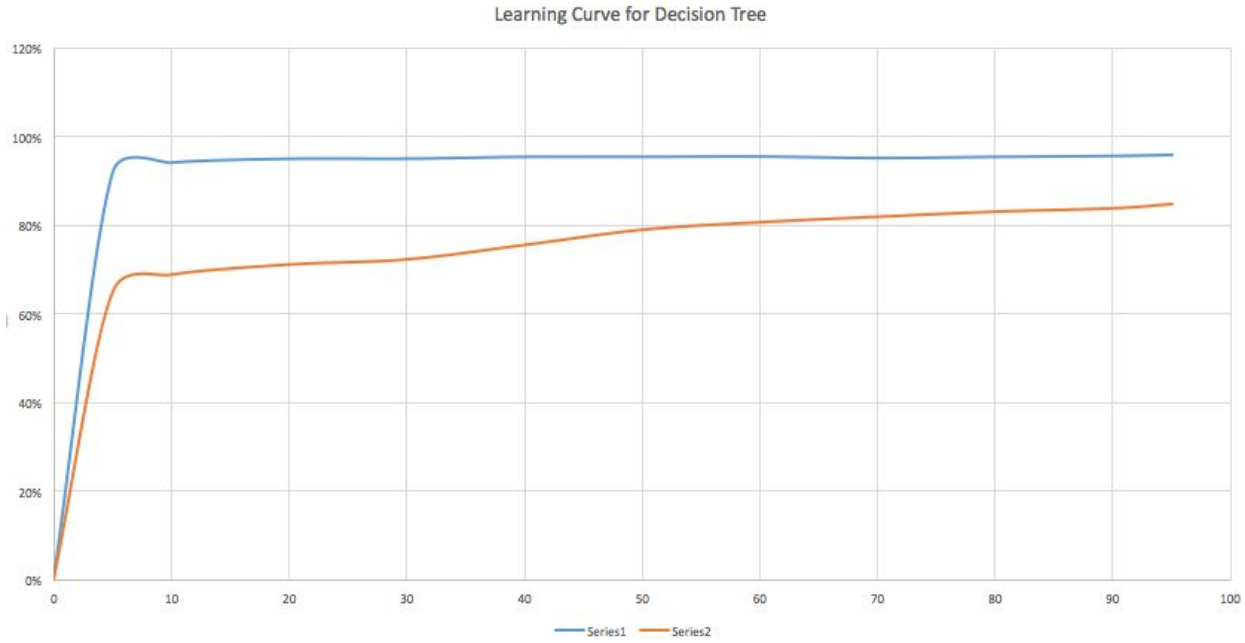
My decision Tree algorithm was also done through Weka. I have both J48 and ID3 algorithms. C4.5 goes back through the tree once it's been created and attempts to remove branches that do not help by replacing them with leaf nodes - this is a form of pruning. However, all testing was only done with J48 since it is simpler to prune and (generally) is considered a better algorithm.

Table 1: Decision Tree Results

Percent of training set used	Data Set 1 Correctness	Data Set 1 Failure	Data Set 2 Correctness	Data Set 2 Failure
5	91.72%	8.28%	64.70%	35.30%
10	94.11%	5.89%	68.75%	31.25%
20	94.98%	5.02%	71.04%	28.96%
30	95.00%	5.00%	72.20%	27.80%
40	95.41%	4.59%	75.46%	24.54%
50	95.43%	4.57%	78.68%	21.32%
60	95.48%	4.52%	80.59%	19.41%
70	95.16%	4.54%	81.82%	18.18%
80	95.41%	4.59%	82.97%	17.03%
90	95.62%	4.38%	83.73%	16.27%
95	95.87%	4.13%	84.69%	15.31%

Since decision trees are generally white box algorithms, I was expecting it to generally perform worse on these data sets in comparison to the other algorithms. I did notice that although the algorithm took longer when dealing with data set 1 (with 17 attributes, probably creating a longer decision tree), it is much more accurate, especially at the low levels. I see about a 27% difference in the data sets when using 5% of the data to train. Data set 2 better exemplifies my goals when picking data sets. In initial trials, i did not expect the asymptotic relationship to begin so quickly. It would be much more interesting to go from 0-20 instead of 5-95, as that is where most of the fluctuation occurs. However, a line of best fit should be fairly close either way. As I can see from graph 1 (below) more data points in the lower end of the data table would have been sufficient. Both data sets became easy to train fairly quickly, so the complexity of the problems were not as high as I had expected. In retrospect, data sets with more chances of outliers would be a lot harder to classify. In both of these examples I don't have many outliers, although I would expect the vote data to contain more. However, the asymptotic line at approximately 4% error and 14% error (data 1 and data 2 respectively) show that there are cases that J48 is having trouble classifying. Data set 2 is a much better example of what I was hoping to see, although the jump at each percentage of the training data early on is quite rapid. I do see a slight bump just before 10% indicating more data points would be useful.

Graph 1: Learning Curve for Decision Tree



Boosting

I used AdaBoost M1 from weka's library to do my boosting algorithm. I found an implementation in which I can set the algorithm to pair with different classifiers. Here are the results of the different classifiers at 100 iterations:

Table 2: Boosting with Classifier Comparison

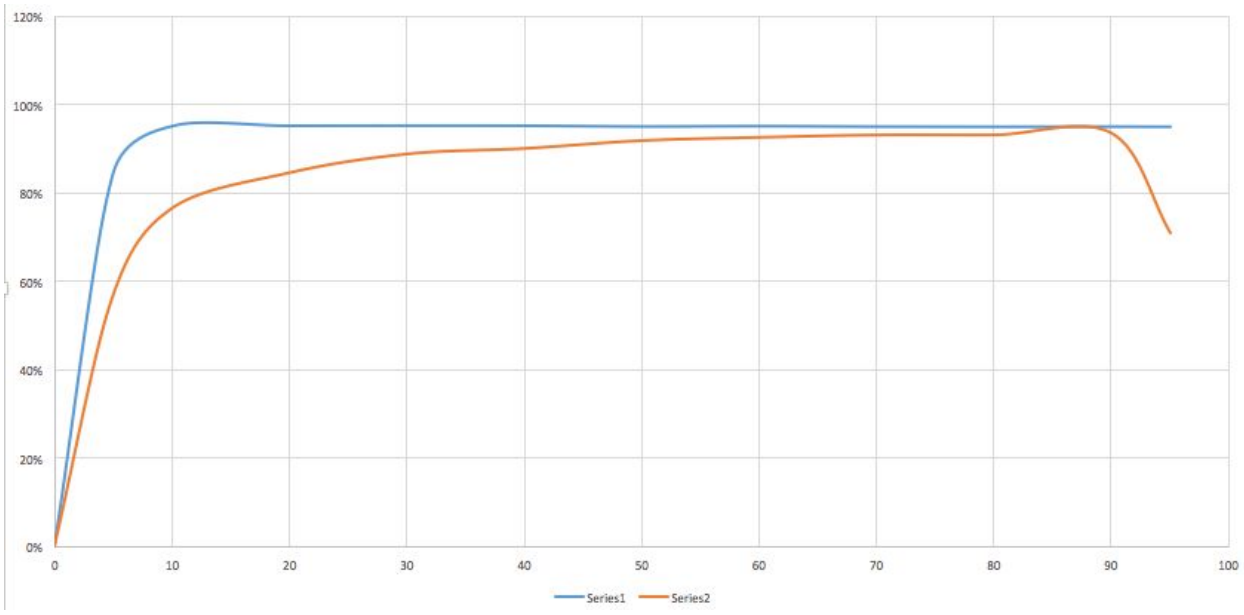
Classifier (paired with adaboostM1 algorithm)	Data Set 1 Correctness	Data Set 1 failure	Data Set 1 Relative Absolute error	Data Set 2 correctness	Data Set 2 failure	Data set 2 Relative absolute error
J48	95.12%	4.88%	14.90%	89.09%	10.91%	22.03%
Naive Bayes	90.25%	9.75%	19.9%	70.90%	19.10%	79.09%
NBTree	97.56%	2.44%	8.47%	85.45%	14.55%	44.96%

It is clear in this case that Data set 1 is easier to classify via these boosting methods. Data set 2 is significantly more difficult. However, Now I will pick one of these algorithms to stick with and modify the amount of training data adaboost uses in order to see how that affects the performance of the algorithm, as I did with the decision tree algorithm.

Table 3: Boosting Results

Percent of training set used	Data Set 1 Correctness	Data Set 1 Failure	Data Set 2 Correctness	Data Set 2 Failure
5	84.39%	15.71%	56.87%	42.13%
10	95.04%	4.96%	76.52%	23.48%
20	95.15%	4.85%	84.58%	15.42%
30	95.18%	4.82%	88.83%	11.17%
40	95.16%	4.84%	90.08%	9.92%
50	94.99% *	5.01%	91.86%	8.14%
60	95.07%	4.93%	92.60%	7.40%
70	94.96%	5.04%	93.14%	6.86%
80	94.92%	5.08%	93.16%	6.84%
90	94.95%	5.05%	93.53%	6.47%
95	94.92%	5.08%	70.92%	29.08% -**overfit**

Graph 2: Learning Curve for AdaBoostM1:



The data results from my boosting learning curve are much more what I was hoping for, especially for data set 2 where the curve is gradual through the 5-20 range. I believe at the end

of my trial I did witness some overfitting of the data. After allowing the algorithm to train at 95% of the training data, I see it is not nearly as successful as it was with 90%. However, AdaBoost does have a tendency to overfit, particularly when noise is present in the data. Otherwise the data reacts as expected - as I increase the amount of training data, the algorithm gets more accurate in predicting the class.

KNN

In this algorithm I ran a loop to test how efficient the algorithm was when considering 1-5 neighbors (k=1, k= 2, … k = 5). It's important to note for this implementation that only one file is accepted and the testing set is obtained through the code. All of the other algorithms accept both test and training sets in order to split the data and learn. The following testing done was doing Euclidean Distance along with Weka's IBk algorithm.

Table 4: KNN Results - Varying K:

KNN IBK algorithm (k = 1-5, 10, 20)	Data Set 1 Training Set Correctness	Data Set 2 Testing set Correctness	Data Set 2 Training set Correctness	Data Set 2 Testing set Correctness
1	99.75%	92.68%	100%	98.18%
2	95.17%	92.68%	99.33%	98.18%
3	94.41%	95.12%	99.33%	98.18%
4	93.91%	95.12%	99.33%	98.18%
5	93.91%	95.12%	99.33%	98.18%
10	93.91%	95.12%	98.18%	98.18%
20	91.87%	92.68%	83.52%	81.82%

Particularly with the data set, as I increased the number of neighbors, it was able to classify democrats much better, but there were not large differences for the democratic data. In general, IBk did an excellent job classifying each of these data sets. Making a graph would be a but redundant in this case. I later added the more outlier cases of 10 and 20 nearest neighbors in order to see around what point the algorithm would begin to overfit. Again, data set 1 seems to be easier to classify here in general. I don't believe a graph would add much value. As we see the results consistently gets worse except for with data set 2 testing set.

Neural Networks

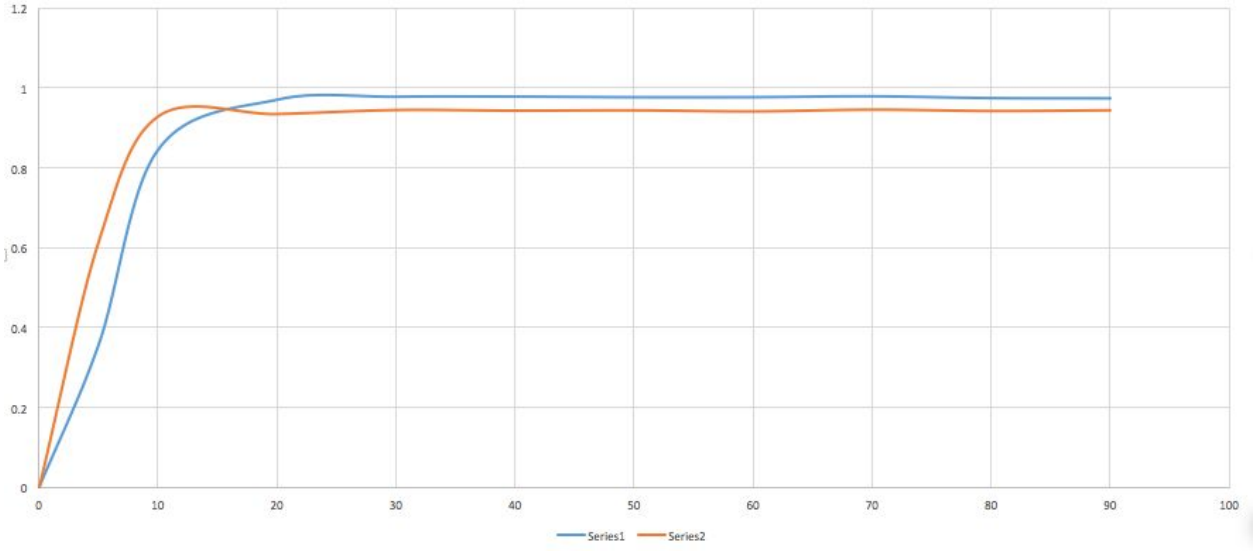
My neural network implementation is also from Weka's library, using the MLP (multi-layer-perceptron) algorithm. This classifier uses backpropagation to learn a multi-layer

perceptron to classify instances. My code is modified to test both training data and testing data. All tests were also done with 3 hidden layers, as is the default for this algorithm.

Table 5: Neural Network Results - MLP

Percent of training set used	Data Set 1 Correctness	Data Set 1 Failure	Data Set 2 Correctness	Data Set 2 Failure
5	35.80%	65.20%	61.57%	38.43%
10	84.62%	15.38%	92.99%	7.01%
20	97.12%	2.88%	93.57%	6.43%
30	97.86%	2.14%	94.58%	5.02%
40	97.89%	2.11%	94.42%	5.58%
50	97.74%	2.26%	94.50%	5.50%
60	97.75%	2.25%	94.24%	5.78%
70	97.97%	2.03%	94.67%	5.33%
80	97.52%	2.48%	94.34%	5.66%
90	97.45%	2.55%	94.51%	5.49%

Graph 4: Neural Network Results - MLP algorithm Training curves



This was the only algorithm that was able to identify data set 2 faster than data set 1. Although the asymptote for data set 2 is lower than the asymptote for data set 1, it was able to classify better in percentages (of training data) below 15. I'd argue that this algorithm is the first that

actually struggled in classifying data set 1 in low percentages, and that is the cause of this difference, since it eventually did end up classifying data set 2 better than data set 1. I believe that somewhere in the backpropagation process, the algorithm was struggling to come up with a linear perceptron as it ran into some special cases. After finding error in certain area of voters statistics, it weighted those areas a bit too heavily in it's initial non-linear regression decisions. A possible solution to this would be to change the backpropogation process to change those variables that constituted a wrong decision and weigh them less in that decision. Then, make the algorithm able to identify outliers based on previous weighting.

SVM

I used Weka's SVM implementation for the Support Vector algorithm as well. My implementation both a training set and test set. I used the CMO implementation in order to run all of my tests.

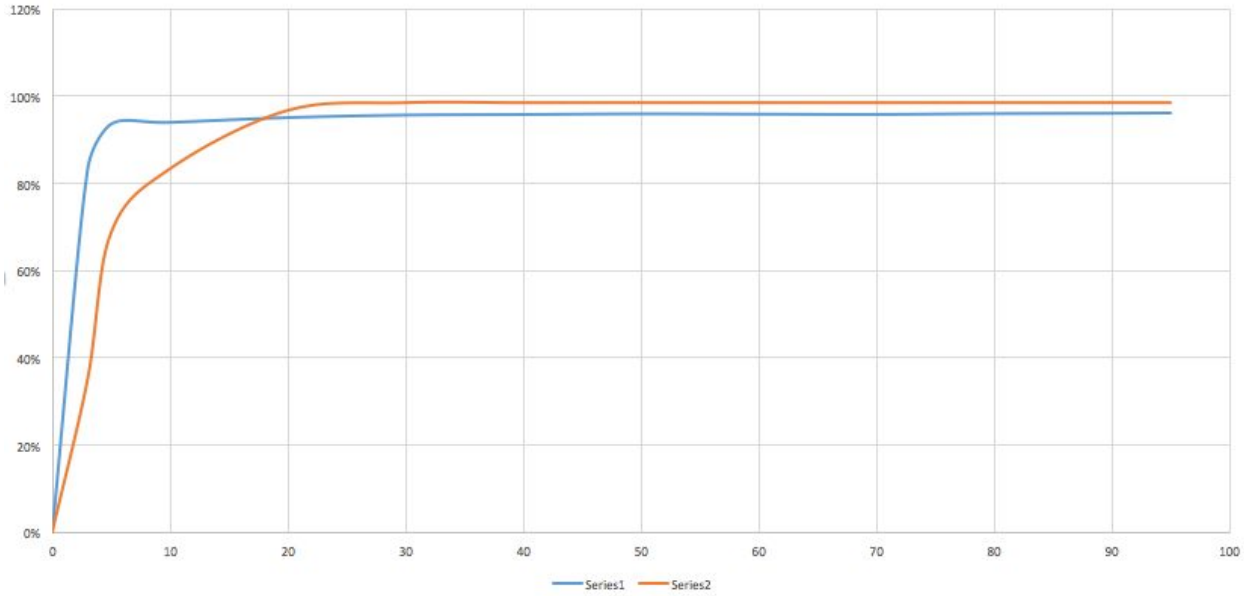
Table 6: SVM Results - CMO algorithm

Percent of training set used	Data Set 1 Correctness	Data Set 1 Failure	Data Set 2 Correctness	Data Set 2 Failure
5	91.13%	8.87%bg	68.64%	31.36%
10	93.45%	6.55%	83.29%	16.71%
15	93.83%	6.17%	96.68%	3.32%
20	94.93%	5.07%	98.45%	1.55%
30	95.51%	4.49%	98.45%	1.55%
40	95.64%	4.36%	98.45%	1.55%
50	95.79%	4.21%	98.45%	1.55%
60	95.72%	4.28%	98.45%	1.55%
70	95.66%	4.44%	98.45%	1.55%
80	95.84%	4.26%	98.45%	1.55%
90	95.92%	4.08%	98.45%	1.55%

Again, data set 1 proves extremely easy to classify, which is surprising to me. On initial analysis of the data and the distribution of republican or democrat on each matter, I assumed the algorithms would struggle in cases where certain attributes were unexpected. However, I believe data set 2 is much harder to classify because of the relationship not only between the

class and the other attributes, but the relationship between multiple attributes and the result class (3 in a row, positive).

Graph 5: SVM Results - CMO algorithm



In this case, we see that data set 1 was classified almost immediately. In accordance with almost every other algorithm, data set 2 was much harder to classify. In retrospect I would have chosen a more difficult problem, but I did learn a lot in that even though few outliers may occur in voting classification, it does not necessarily mean the entire sample is hard to classify.

Conclusion

In order to analyze how well each algorithm actually performed, I think it's only fair to compare how well the algorithms performed on the same datasets with the same amount of training data used. See table:

Table 7: Comparison at 10% of training data

Percent of training set used	Data Set 1 Correctness	Data Set 2 Correctness	Total Correctness (average)
SVM - CMO	93.45%	83.29%	88.37%
NN - MLP	84.62%	92.99%	88.81%
Boosting - AdaBoost	95.04%	76.52%	85.78%
Tree - J48	94.11%	68.75%	81.43%

KNN - IBk	94.40%	99.33%	96.96%

We can see that the average of these two datasets shows that MLP is the most effective algorithm for these datasets, while J48 is the least effective algorithm. This is primarily because J48 struggled so much with classifying data set 2 (tic-tac-toe). In general, data set 1 was classified much better by all algorithms except for MLP, which performed better on data set 2. IBk was significantly better at classifying all of the data sets.

Code Repository: https://github.com/tylerapost/SupervisedLearning