Aaron Feldman

10/26/16

Project Three

1. Pick some data suitable for classification that we haven’t used in class yet (5/5): For this experiment, I chose soccer statistics on the players from the top five teams from the English Premier League. I believe it is suitable because it has classification categories like Team (which says either Leicester City, Arsenal, Manchester City, Manchester United, Tottenham), and On\_National\_Team (which either says “Yes” or “No” to indicate whether the player plays for their country as well), Nationality (which states which country they are originally from, regardless of whether they made their national team), and their position (Defense, Midfield, or Forward). It also has numeric categories suitable for predicting classification. They are as follows:

AGE: the player’s age

GS: Games started

SB: Subs in

G: Total goals

SH: Total shots

SG: shots on target

A: Goal Assists

FC: Fouls Committed

FS: Fouls suffered

YC: Yellow Cards

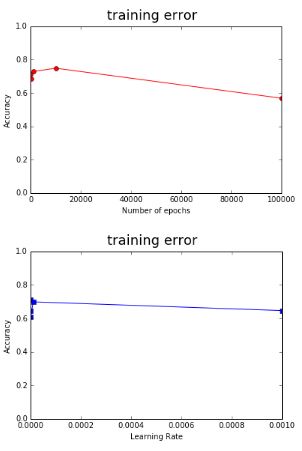
RC: Red Cards

The categories PLAYER (the player’s name) and NO. (the player’s number) are simply columns for identification. I did not do any data preparation.

Use a Perceptron algorithm to see how well the classifier works on your dataset (6/6 points): For part one, I chose to do a perceptron algorithm. I used all the numeric data above to predict whether the players made their national team. I played with the learning rate and number of epochs to determine what got me the highest accuracy. The accuracy seemed to be sensitive to large changes in number of epochs and learning rate, so I tried out different powers of ten as the x-values. See the graphs below:

|  |  |
| --- | --- |
| # of Epochs | Accuracy |
| 1 | .712 |
| 100 | .684 |
| 1,000 | .728 |
| 10,000 | .748 |
| 100,000 | .568 |
| 1,000,000 | .596 |

|  |  |
| --- | --- |
| Learning Rate | Accuracy |
| .00000000001 | .71 |
| .000000001 | .646 |
| .0000001 | .61 |
| .00001 | .698 |
| .001 | .646 |
| 1 | .712 |



Each one of the accuracies was determined by running the program twenty times with the given learning rate and number of epochs and taking the average. As you can see, the optimal number of epochs was around 10,000, which gave me an accuracy of .748, which is pretty decent. Generally, the graph seems to increase until that point and then drastically decreases.

The highest learning rate value I got was actually 1, which is the default value in scikit-learn. However, I believe that the learning rate has little to zero effect on this dataset. If you look at the graph, you can see that the accuracies increase and decrease randomly as the learning rate increases. To illustrate, the accuracies of 1 and .00000000001 are almost exactly the same.

Use kNN to determine how well the classifier works on your dataset (6/6 points): Plugging in different k-values, I produced the following graph:

|  |  |
| --- | --- |
| K-Value | Accuracy |
| 2 | .622 |
| 4 | .71 |
| 5 | .766 |
| 7 | .752 |
| 10 | .728 |



I found the most accurate value of k to be 5. I derived the accuracy by running the program twenty times each for the respective values of K so I knew it wasn’t just luck. After 5, the accuracy seems to decrease slightly.

Compare kNN and Perceptron: Of the two algorithms, kNN performed better. kNN gave me the highest overall accuracy of .766 with the best k-value I derived. In addition, I averaged the accuracies each of the three tables above with different epochs, learning rates and neighbors and kNN had the most consistently high accuracy. The first table where I altered the number of epochs had an average accuracy of .672, the second table where I altered the learning rate had an accuracy of .6703, and kNN had an accuracy of .7156. I believe that Perceptron is less accurate because it is harder to draw a perfectly linearly separable line on this dataset. A national soccer team, more or less, selects a diversity of offensive and defensive players. Offenders and defenders should have different statistics (ie forwards score more goals but defenders commit more fouls), however, different types of players are selected based on different criteria. For example, defenders and goalkeepers who do not score many goals still often make their national teams but forwards who fail to score many goals are probably not selected, as there is probably a better forward elsewhere. I think if I had included the column POS (position), the data would have been more linearly separable and therefore more accurate. I think kNN was more accurate because players with similar statistics (regardless of position) are more likely to not make or make their national team and these groupings allowed the program to make a slightly more accurate guess based on similar players.

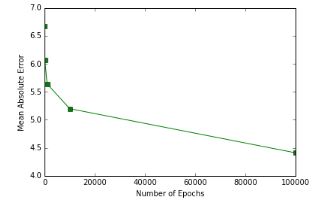
Regression

Pick some data suitable for Regression (5/5): For the regression algorithm, I chose the Housing dataset from the Machine Learning Repository from UCI. I believe it is suitable for regression because it has all numeric data, and I plan to use it to predict the value of MEDV (which is also numeric). I did not do any data preparation. The columns in the dataset are described as follows:

1. CRIM: per capita crime rate by town   
2. ZN: proportion of residential land zoned for lots over 25,000 sq.ft.   
3. INDUS: proportion of non-retail business acres per town   
4. CHAS: Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)   
5. NOX: nitric oxides concentration (parts per 10 million)   
6. RM: average number of rooms per dwelling   
7. AGE: proportion of owner-occupied units built prior to 1940   
8. DIS: weighted distances to five Boston employment centres   
9. RAD: index of accessibility to radial highways   
10. TAX: full-value property-tax rate per $10,000   
11. PTRATIO: pupil-teacher ratio by town   
12. B: 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town   
13. LSTAT: % lower status of the population   
14. MEDV: Median value of owner-occupied homes in $1000's

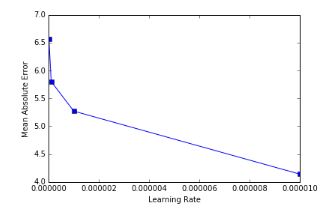
Use a scikit-learn SGDRegressor algorithm to determine how well the classifier works (6/6 points): For this section, I plugged in different values for learning rate and epochs and produced the following graphs. I chose to use mean\_absolute\_error instead of mean\_squared\_error because it gave me more accurate results.

|  |  |
| --- | --- |
| Number of Epochs | Mean Absolute Error |
| 10 | 6.67 |
| 100 | 6.07 |
| 1000 | 5.64 |
| 10000 | 5.198 |
| 100000 | 4.41 |
| 1000000 | Program Crashed |



Looking at the above graph, it seems that the higher the number of epochs, the higher the accuracy. I tried plugging in one million but the program would not run. Based on this model, one could select an even higher learning rate than 100,000 up until the point it crashes.

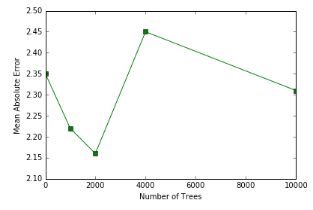
|  |  |
| --- | --- |
| Learning Rate | Mean Absolute Error |
| .00000001 | 6.56 |
| .0000001 | 5.80 |
| .000001 | 5.27 |
| .00001 | 4.14 |
| .001 | In the millions |



Looking at the graph above, it appears the ideal learning rate is .00001, which gives me a mean absolute error of 4.14. Adding another zero only seems to increase the mean absolute error.

Use a scikit-learn implementation of Random Forest Trees (6/6 points): Using different numbers of random forest trees, I produced the following graph. It appears that 2000 is a good number to use. After that, the mean absolute-error increases slightly and takes a very long time to run. I produced these numbers by running the program with the given number of trees several times.

|  |  |
| --- | --- |
| Number of Trees | Mean Absolute Error |
| 10 | 2.35 |
| 1000 | 2.22 |
| 2000 | 2.16 |
| 4000 | 2.45 |
| 10000 | 2.31 |



Which algorithm performed better: The Random Forest Tree algorithm performed better than SGDRegressor. SGDRegressor had mean absolute errors in the 3-5 range, whereas Random Forest Trees were only off by about 2-2.5. Also, I think that in this dataset, the algorithm was able to consistently choose strong attributes for the roots of the trees and the subsequent branches. I think, therefore, that perhaps if an attribute is between a certain set of numbers then that is a very clear indication that its MEDV value will be in a certain range. Because we were using multiple trees, it was even less sensitive to noise and variance was decreased, making this algorithm even more accurate. I think one reason that SGDRegressor is less accurate is because it is possibly subject to feature scaling. For example, everything in the TAX and B attributes are around 300 and 400 respectively, but the other entries are significantly lower (either decimal places or less than fifty).