**Part 1: KNN Learner**

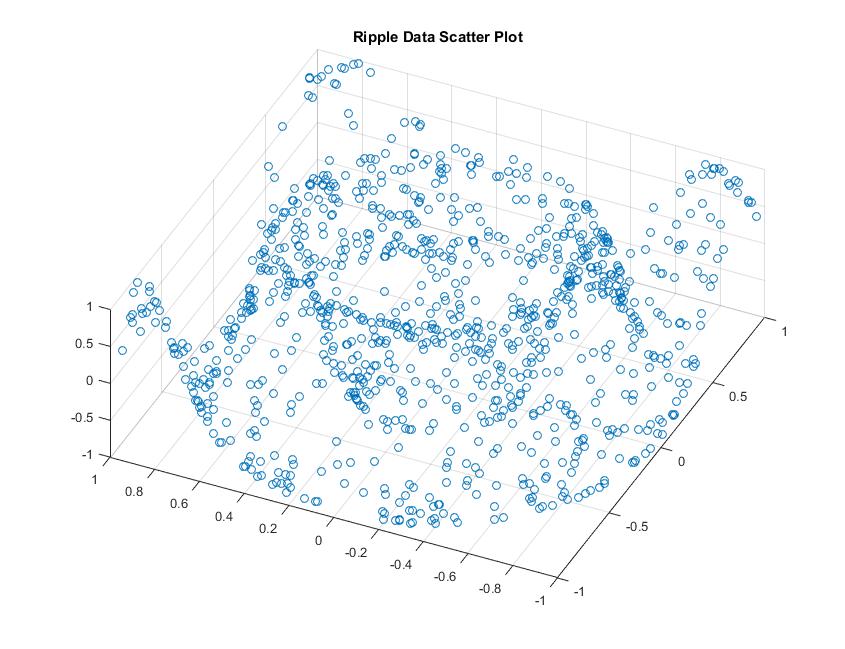
KNN learner uses the k-nearest neighbor is a method to estimate new observations based on previously recorded features and labels. Given a new sample (could have multiple features), the learner will compare that to all previously recorded features/label pairs and select the ‘k’ points that have the least error. The following illustrates how knn-learner works when given a new point (p1):

1. Let us call training observations X(i)j with associated Y(i) where (i) is the observation index and ‘j’ is the feature index
2. Compare p1 to all X(i)j using RMS error
3. Choose the ‘k’ Y(i)’s associated with X(i)j with least error
4. Return mean of the chosen ‘k’ Y(i)s

To test the accuracy of my classifier it was ran on ‘ripple.csv’ and the following results were obtained using the default value of ‘k’ (k=3):

|  |  |
| --- | --- |
| **KNN Learner** | **Linear Regression Learner** |
| In sample results  RMSE: 0.136590187312  corr: 0.981360326901 | In sample results  RMSE: 0.705375081106  corr: 0.040607011093 |
| Out of sample results  RMSE: 0.207762150054  corr: 0.955537498166 | Out of sample results  RMSE: 0.70409341685  corr: 0.0162663457175 |
| Runtime: 0.092 seconds | Runtime: 0.004 seconds |

It can be seen from the table above that the KNN learner outperforms the linear regression learner in the ‘ripple’ data and this is because the data is non-linear as can be seen in the plot below. The knn-learner performs better since it is non-parametric where it predicts each point based on the neighboring points with out forcing it on all future points. Where as the lin-reg learner comes up with parameters that are based on the train set and independent of the new observed point where we want to predict its value.



It can be seen above that the data follows a non-linear behavior which explains the poor performance of the linear regression learner.

**Part2: BagLearner**

Bagging is a way to refine the accuracy of the learner algorithm as mentioned in [1]. It draws ‘n’ bags of samples from the training data and applies the learner to those ‘n’ bags and then it averages the results of all of the bags into an aggregated value that usually is more accurate and stable. The results for the bag learner compared to the regular KNN learner for the ‘ripple’ data is as follows. This is based on k=3 for both learners and 20 bags for the bag learner.

|  |  |
| --- | --- |
| **Bag Learner** | **KNN Learner** |
| In sample results  RMSE: 0.12516014302  corr: 0.985542263338 | In sample results  RMSE: 0.136590187312  corr: 0.981360326901 |
| Out of sample results  RMSE: 0.198093185311  corr: 0.961210370576 | Out of sample results  RMSE: 0.207762150054  corr: 0.955537498166 |
| Runtime: 1.555 seconds | Runtime: 0.092 seconds |

It can be seen how bag learner performs slightly better than the KNN-learner. However, it has a higher runtime due to it running ‘n’ times more than the KNN-learner.

**Part3: Experiments**

1. Best4linreg

The data I have created for this part is linear data in 3-dimensional space where two dimensions are needed to get the output. The reason I only chose 2-dimensions in X is that if were to use more dimensions, it would be difficult to visualize. The main idea of my function is linearity in three dimensional space which follows the following equation:

Y = x1 + x2

Since it is a linear equation, it is expected that the linear-regression learner would outperform the Knn-learner. The performance is as follows which were obtained using the generated csv file called by the testlearner function for both the linear and knn learners:

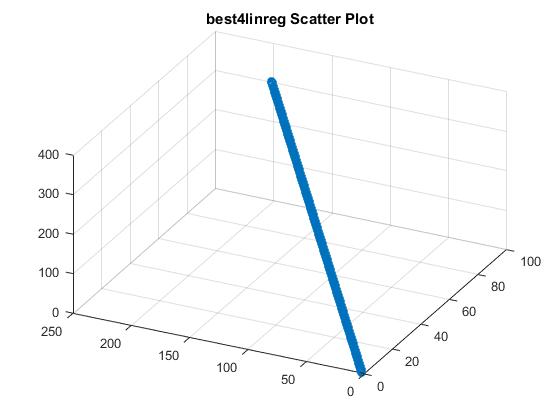
|  |  |
| --- | --- |
| **KNN Learner** | **Linear Regression Learner** |
| In sample results  RMSE: 0.204979673789  corr: 0.999997223759 | In sample results  RMSE: 2.40553566525e-13  corr: 1.0 |
| Out of sample results  RMSE: 0.297363414024  corr: 0.999994003255 | Out of sample results  RMSE: 2.32182694885e-13  corr: 1.0 |
| Runtime: 0.090 seconds | Runtime: 0.005 seconds |

The way the data was generated, points from 0-100 with a 0.1 step were created for x1 and points from 2\*range(1-100) with 0.1 step were created for x2. Where the value of x1 and x2 for each observation is different. So a subset of x1 and x2 combinations were used to form a line. If all the x1 and x2 combinations were to be included then a plane would be obtained.

Then Y was obtained using the above equation. If I were to leave the data un-scrambled, then I would create a bias for the linear regression learner where the KNN-learner will perform really poorly, since all of the test data will point in the outer boundary of the line created from the testing data. To solve this I decided to scramble the data so that training data will not be condensed in the center but would be all over.

It can be seen in the table above that the linear regression outperforms the Knn-learner and it also has a faster execution time. This however is quite understandable because if we examine the equation above we would find that the derivative of the equation is a constant such that and which clearly shows the linear behavior of (Y).

To further illustrate the linearity of Y, the following scatter plot shows the shape of the data in 3-dimensional space.



1. **Best4KNN**

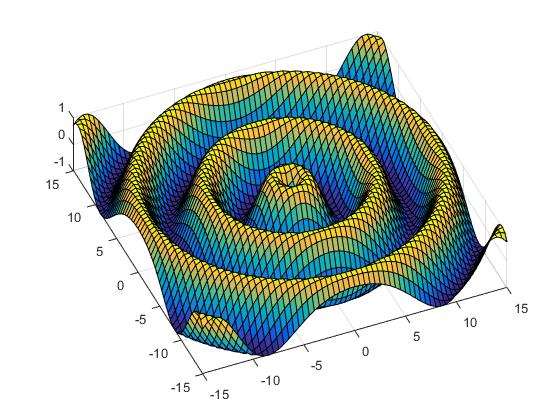
The data I have created for this part is data in 3-dimensional space where two dimensions are needed to get the output. I generated the data based on the following equation:

The values of x1 were varied from -20 to 20 with a 0.04 step size and the x2 is twice x1 to ensure that all (x1, x2) value pairs have different values and then the data is shuffled resulting in a slice in the ripple. The results are as follows which were obtained using the generated csv file called by the testlearner function for both the knn and linear regression learners:

|  |  |
| --- | --- |
| **KNN Learner** | **Linear Regression Learner** |
| In sample results  RMSE: 0.0476335820357  corr: 0.997753232477 | In sample results  RMSE: 0.708582307705  corr: 0.0250300210444 |
| Out of sample results  RMSE: 0.0658878296558  corr: 0.995546503543 | Out of sample results  RMSE: 0.695770983909  corr: -0.0410521002959 |
| Runtime: 0.092 seconds | Runtime: 0.006 seconds |

It can be seen from the table above that the KNN-learner outperforms the linear regression learner quite significantly. This is quite understandable since if we were to examine the derivative of the above equation, we would find it exhibiting a nonlinear behavior. The derivatives with respect to the variable are as follows:

and

Which clearly exhibits a non-linear behavior since the rate of change is dependent on the input values. The plot of the above equation spanning all the x1 and x2 combinations between -15 and 15 (a mesh grid) looks like:

1. **Consider the dataset ripple with KNN. For which values of K does overfitting occur? (Don't use bagging).**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | | K = 1 | 2 | **3** | 4 | 5 | 6 |
| In sample | RMSE | 0.0 | 0.1178 | **0.1365** | 0.1583 | 0.1662 | 0.1759 |
| Corr | 1.0 | 0.9859 | **0.9813** | 0.9753 | 0.9734 | 0.9716 |
| Out sample | RMSE | 0.237 | 0.2135 | **0.2077** | 0.212 | 0.2216 | 0.2376 |
| Corr | 0.944 | 0.9529 | **0.9555** | 0.9546 | 0.9517 | 0.9465 |

We can see from the table above that from k=1 to k=3, the in sample error increases while the out sample error decreases. This behavior continues until we reach k=3 after which the out sample stops decreasing and both the in sample and out sample increase. So the it occurs for k>3.

The same can be concluded from correlation where it keeps on increasing for the out of sample until we reach k=3 and then it stops and starts decreasing.

1. **Now use bagging in conjunction with KNN with the ripple dataset. How does performance vary as you increase the number of bags? Does overfitting occur with respect to the number of bags?**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | bags = 1 | 5 | 10 | 15 | 20 | 25 | 30 | 60 |
| In sample | RMSE | 0.2210 | 0.1472 | 0.1358 | 0.1298 | 0.1242 | 0.1248 | 0.1228 | 0.1228 |
| Corr | 0.9499 | 0.9791 | 0.9826 | 0.9847 | 0.9859 | 0.9858 | 0.9862 | 0.9863 |
| Out sample | RMSE | 0.3003 | 0.2146 | 0.2048 | 0.1930 | 0.1882 | 0.1896 | 0.1930 | 0.1900 |
| Corr | 0.9049 | 0.9535 | 0.9581 | 0.9639 | 0.9657 | 0.9654 | 0.9638 | 0.9652 |

It can be seen from the table above that the increase in number of bags when numBags is small increases the performance this can be seen between numBags = 1 and numBags = 5 where the rmse decreased significantly and the correlation increased. Beyond numBags = 5, we start seeing the performance reaching a steady state is that stable with out much change in performance. This is especially evident when we compare numBags = 30 to numBags = 60 where you would think that the performance would change but in reality it doesn’t since the learner reached its steady state at a smaller value of bags.

So we can say that bagging did not cause overfitting rather it stabilizes the output.

1. **Can bagging reduce or eliminate overfitting with respect to K for the ripple dataset?**

Based on the table in D, I would assume that bagging performs well for numBags = 20. Let us then re-examine the table in C and see the effect of bagging on overfitting

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | | K = 1 | 2 | 3 | 4 | 5 | 6 |
| In sample | RMSE | 0.0755 | 0.1083 | 0.1293 | 0.1454 | 0.1648 | 0.1807 |
| Corr | 0.9945 | 0.9889 | 0.9849 | 0.9810 | 0.9765 | 0.9728 |
| Out sample | RMSE | 0.1962 | 0.1945 | 0.1927 | 0.2035 | 0.2216 | 0.2396 |
| Corr | 0.9607 | 0.9619 | 0.9637 | 0.9607 | 0.9555 | 0.9490 |

We can see from the table above that bagging had the most effect on the under fitted learners with values of (k<=3). Where it pushed the under fitted learners to be much closer to the optimum correlation and RMSE values. It also improved some of the over fitted learners but that improvement starts degrading as the value of k increases where in the case of k=6, it looks as if it has no much effect.

**References:**

1. <https://en.wikipedia.org/wiki/Bootstrap_aggregating>
2. <https://www.wolframalpha.com/>

Wolfram Alpha to find the derivative of the equation.

1. Code provided in hints and resources that was used in the bagging algorithm.
2. CS 7646 Video Lectures
3. The line of method1 to optimize the calculation of the RMS error as posted on piazza <https://piazza.com/class/ij9yiif53l27fs?cid=1143>