MC3-Project-1 - Evaluate Learning Algorithms

Magahet Mendiola

November, 2015

Overview

We will evaluate the use of linear regression and KNN learning algorithms on various datasets. We will also explore the effect of various algorithm parameters on performance and overfitting.

Custom Datasets

Best for Linear Regression

Create your own dataset generating code (call it best4linreg.py) that creates data that performs significantly better with LinRegLearner than KNNLearner.

Explain your data generating algorithm, and explain why LinRegLearner performs better. Your data should include at least 2 dimensions in X, and at least 1000 points. (Don't use bagging for this section).

The following is a simple algorithm for generating a dataset that performs better with linear regression:

```
for x in xrange(1000):
    point = ([x, x], 1000 * x)
```

The points follow a straight diagonal line with a high slope. The reason this works well with linear regression is that it exactly follows a linear model. The learner will find precise parameters for

$$y = mx + b$$

, and will able to interpolate any point along this line with high accuracy.

KNN, on the other hand, does not attempt to learn an underlaying model. It simply finds the average value of the closest training points. If the closest training points lay to one side of the test point, KNN will not be able to extrapolate from a model. Rather, it will return the average of the closer points.

Here is an example with 1d x:

```
training points = [0, 1, 2]
test points = [3, 4, 5]
```

In this case, KNN will return the average of k training points when queried with the test points. In every case, the estimate will be very far from the actual. In the case of k = 3 and using the data generating model of

$$y = 1000 * x$$

, we get the following error:

```
actual, estimate
3000, 1000
4000, 1000
4000, 1000
```

On randomly distributed data, this limitation of KNN presents itself as under and over estimates of any points that need to be interpolated. As we see in Figure x, which shows a zoomed plot of testing data against both KNN and linear regression.

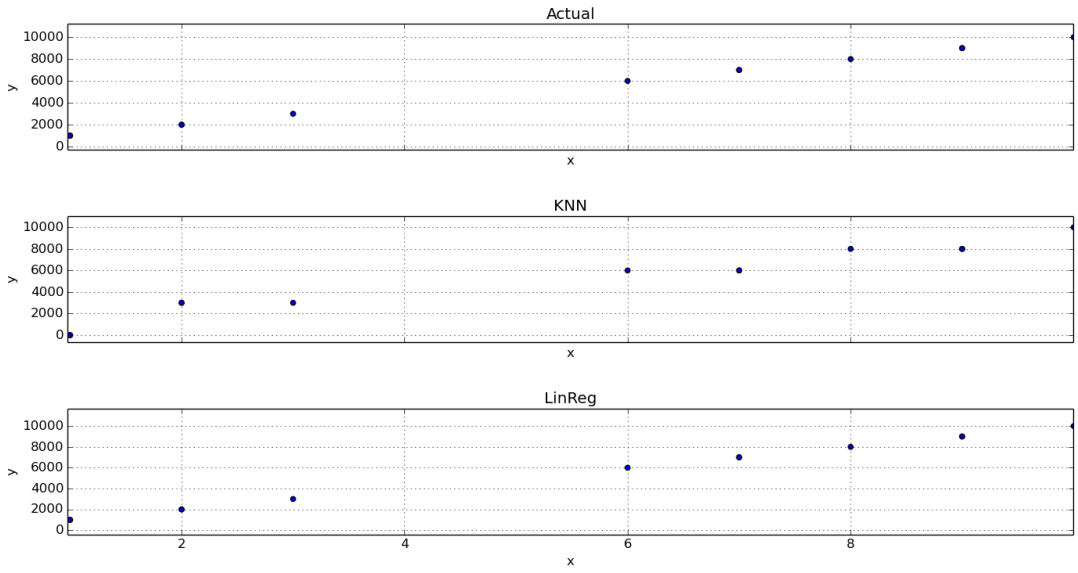

Figure 1: Linear Regression Suited Data

LinReg results

```
In sample results
RMSE:  2.79325142747
corr:  0.999999999952

Out of sample results
RMSE:  2.97142933715
corr:  0.999999999949
```

KNN results

```
In sample results
RMSE:  718.366173592
corr:  0.999996702238

Out of sample results
RMSE:  1202.60984329
corr:  0.99999203473
```

Best for KNN

Create your own dataset generating code (call it best4KNN.py) that creates data that performs significantly better with KNNLearner than LinRegLearner.

Explain your data generating algorithm, and explain why KNNLearner performs better. Your data should include at least 2 dimensions in X, and at least 1000 points. (Don't use bagging for this section).

The following is a simple algorithm for generating a dataset that performs better with knn than linear regression:

```
for x in x_list:
    print '{},{},{}'.format(x, x, add_noise(x % 200))
```

It generates a non-linear repeating pattern that can be seen in Figure X.

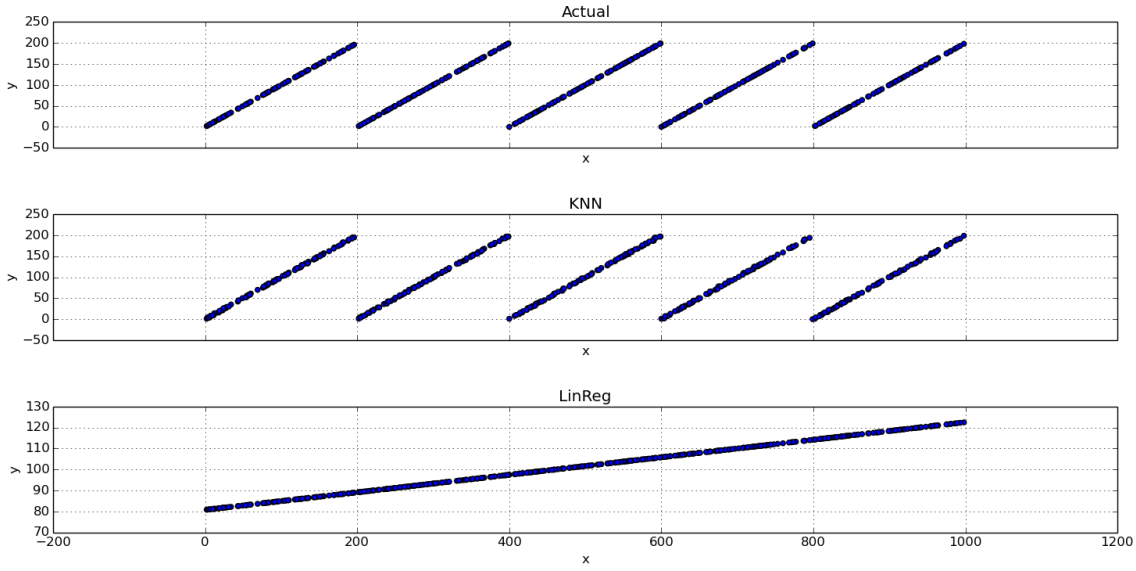

Figure 2: KNN Suited Data

You can see that KNN is able to adapt to the steps from high Y values to low. Some queries will have issues when the nearest neighbor includes both high and low points. However, compared to the overall performance of linear regression, this accounts for a minor increase in error. The following shows the error results for each learner:

KNN results

```
In sample results
RMSE:  7.19196543288
corr:  0.992129724033

Out of sample results
RMSE:  11.0406799595
corr:  0.981941492455
```

LinReg results

```
In sample results
RMSE:  55.9885148725
corr:  0.212757185305

Out of sample results
RMSE:  57.5249782041
corr:  0.177523761057
```

KNN Evaluation

Effect of K on Overfitting

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

Figure X shows that overfitting begins to occur at K = 2. Before this point, error on the test dataset decreases with smaller values of K. At K = 2, the error on testing data increases while it continues to decrease for the training set.

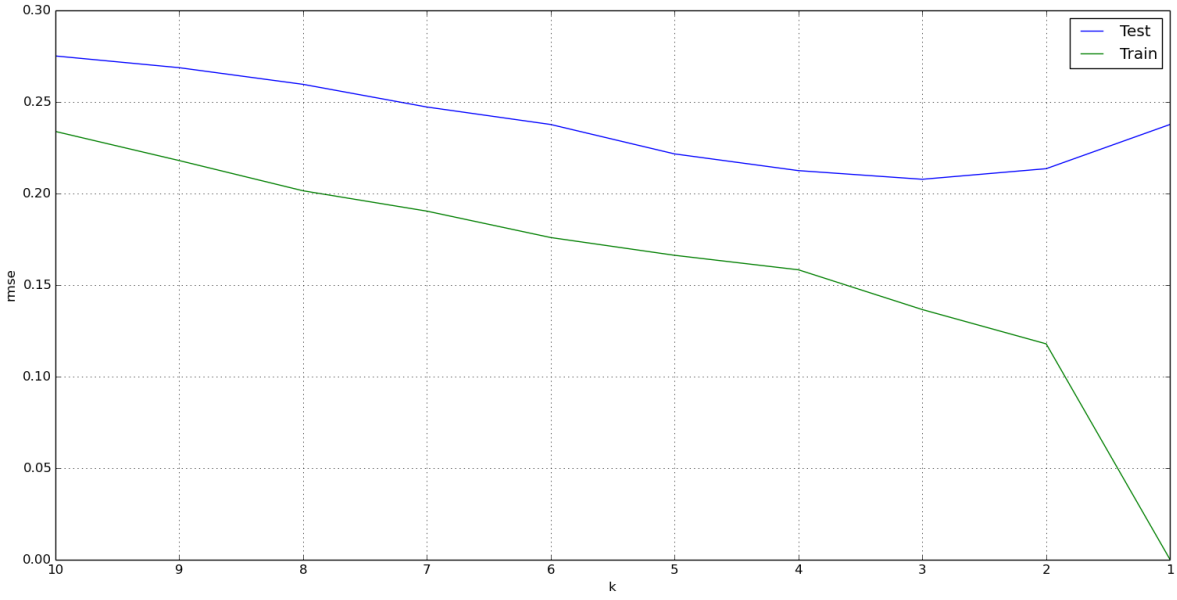

Figure 3: Test and Train error for various K

Effect of Bag Count on Performance

Now use bagging in conjunction with KNN with the ripple dataset.

How does performance vary as you increase the number of bags?

Runtime performance of the bagging learner increases linearly with the number of bags. We see this in Figure X. This result is expected, as each KNN learner requires an equal amount of time to train and query.

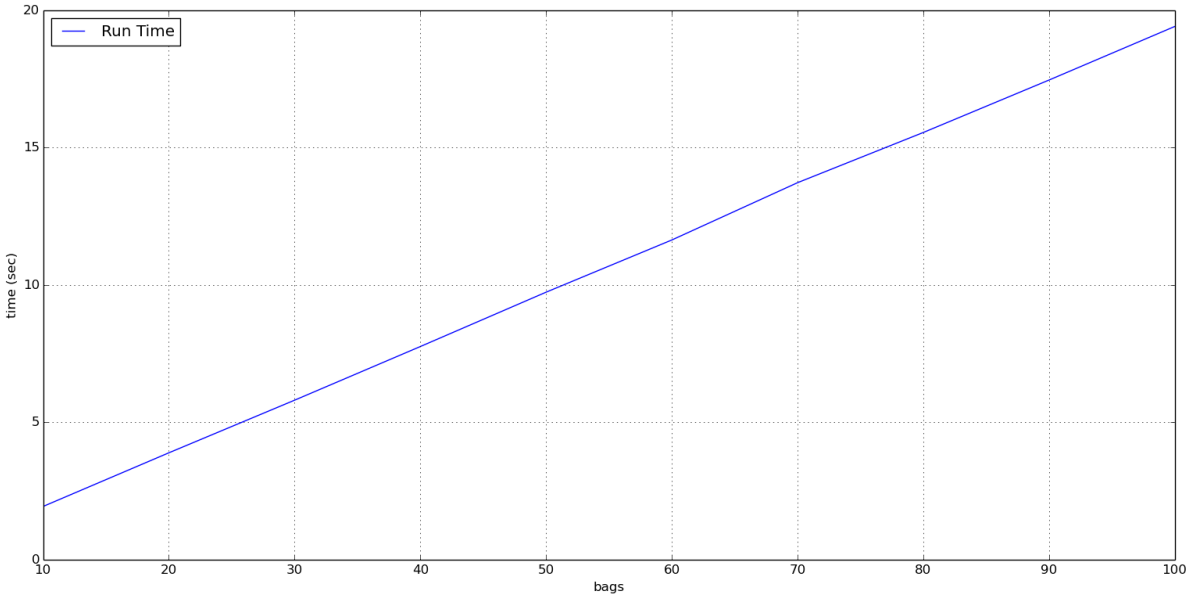

Figure 4: Runtime Performance for Various # of Bags

Effect of Bag Count on Overfitting

Does overfitting occur with respect to the number of bags?

The number of bags does not not seem to contribute to overfitting. Figure X shows that error decreases identically for both the training and testing datasets as the number of bags increase. We also see that error does not drop significantly after a reaching 20-30 bags.

Effect of Bagging with respect to K

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

Figure X shows the error for various number of bags with a KNN learner set with K = 1. As we observed earlier, that value of K causes the worst overfitting. Training error will be zero, and testing error will be higher due to the loss of generalization. Bagging averages various learners' results, thus adding generalization. The effect is that bagging does, indeed, reduce overfitting.

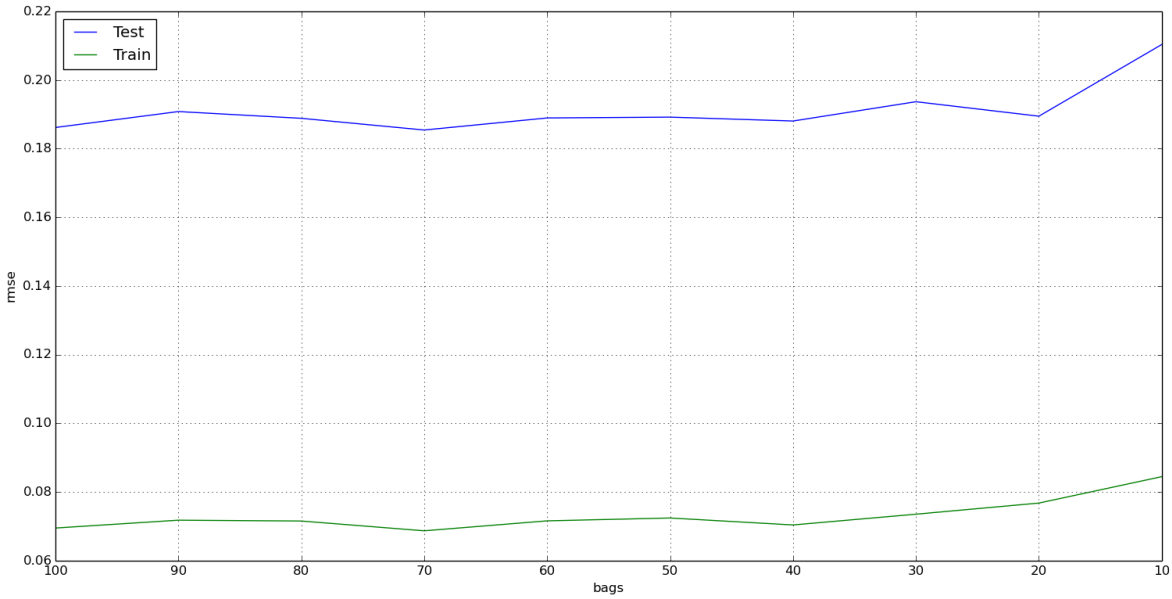

Figure 5: Test and Train Error for Various # of Bags

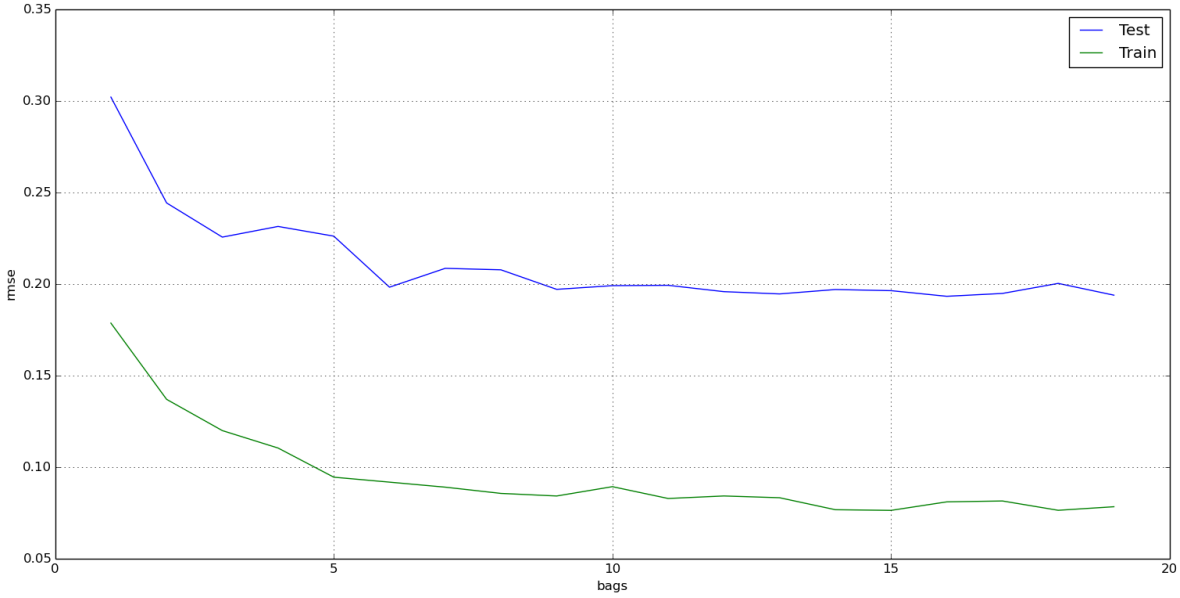

Figure 6: Test and Train Error for Various # of Bags with K = 1