# MC3-Project-1

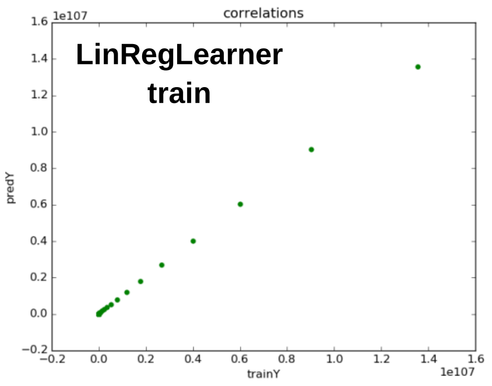
Xiaodong (Sheldon) Gu

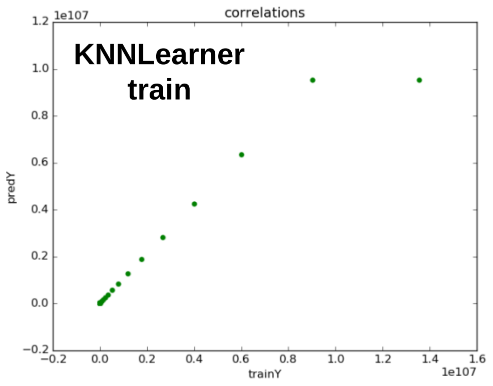
- Create your own dataset generating code (call it best4linreg.py) that creates data that performs significantly better with LinRegLearner than KNNLearner. Explain your data generating algorithm, and explain why LinRegLearner performs better. Your data should include at least 2 dimensions in X, and at least 1000 points. (Don't use bagging for this section).

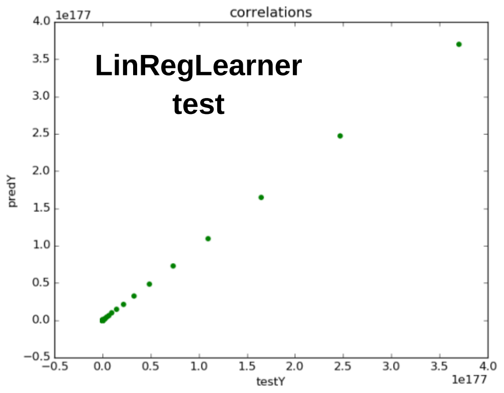
In my dataset, all points locate on a three dimensional line (X1, X2, Y). X1, X2 and Y are highly correlated (Y = X1 + X2), while the distances between points are keep increasing (1.5 fold each time), which means that points on the three dimensional line are not evenly distributed, but are more sparse as the line extends. The code is very simple

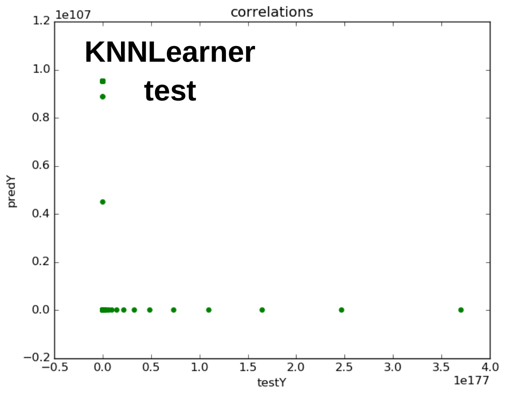
```
p = 0.0
rd = np.random.randint(0, 10, 1)
rd = rd[0]
for i in range(1000):
    p += rd
    out.writerow((p, p*2, p*3))
    rd *=1.5
```

LinRegLearner performs significantly better in training dataset and in testing dataset compared with KNNLearner, as demonstrated in the following four figures. As shown in my code, X1, X2 and Y are highly correlated (Y = X1 + X2), thus LinRegLearner works perfect for my dataset. While for KNNLearner, since the points are sparsely distributed, KNNLearner will more likely to pick neighbors at the left side, thus the predicted Y value will always smaller than the true value (as demonstrated in KNNLearner train figure). (Professor's testlearner function used the first 60% as training set and the last 40% as testing set, which make the performance of KNNLearner worse, since for the points in testing set, the neighbors they can find are the last k points in training set, but even the training, testing sets are chosen randomly (not the first 60%, last 40%), LinRegLearner will still perform much better than KNNLearner and the difference will be more like the difference shown in training set, not as dramatic as shown in testing set.





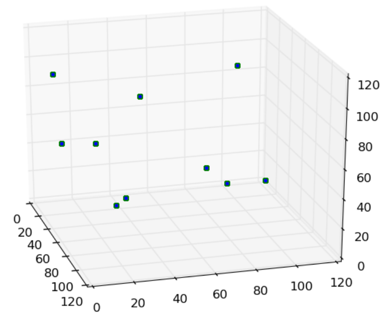


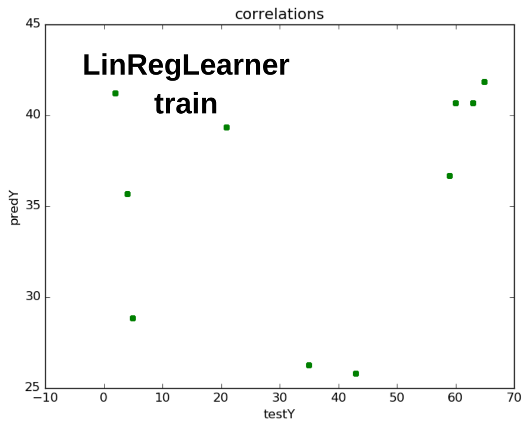


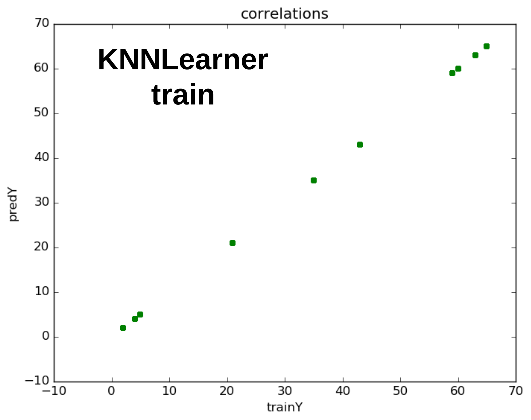
- Create your own dataset generating code (call it best4KNN.py) that creates data that performs significantly better with KNNLearner than LinRegLearner. Explain your data generating algorithm, and explain why KNNLearner performs better. Your data should include at least 2 dimensions in X, and at least 1000 points. (Don't use bagging for this section).

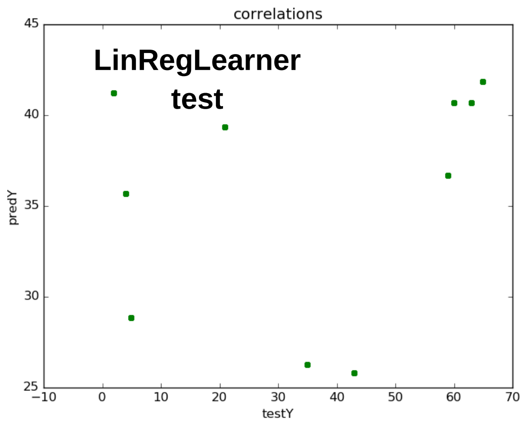
In my dataset, 1000 points locate in 10 randomly distributed clusters (each cluster has 100 points). The train set contains 60 points of each cluster, and the test set contains the left 40 points of each cluster. KNNLearner performs significantly better in testing dataset compared with LinRegLearner, as demonstrated in the following figures. The code is also quite simple:

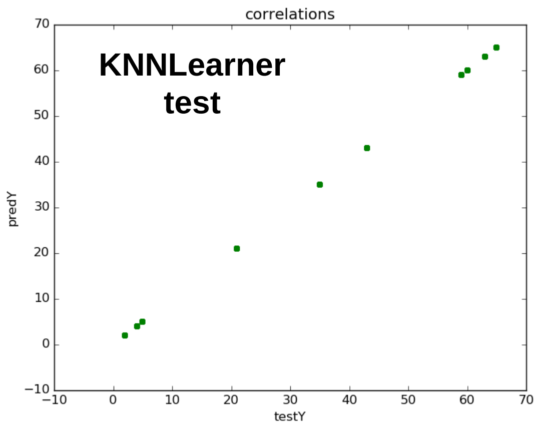
```
a = np.random.randint(0, 100, 10)
b = np.random.randint(0, 100, 10)
c = np.random.randint(0, 100, 10)
for i in range(100):
    for j in range(10):
        out.writerow((a[j], b[j], c[j]))
```







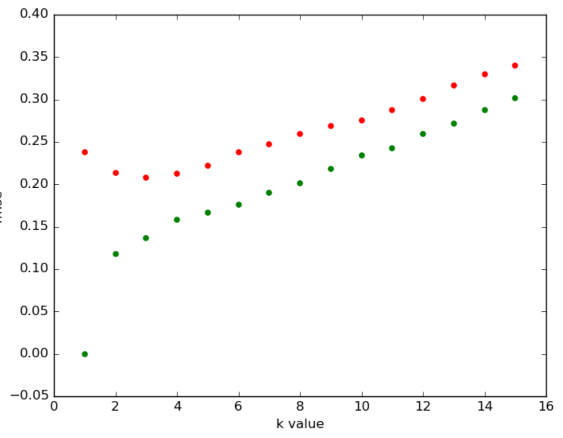




Since the clusters randomly distributed in the three dimensional space (X1, X2, Y), the correlations between X1, X2 and Y are very weak, thus LinRegLearner's performance is poor. When using KNNLearner, each point in test set can find 60 neighbors (distance = 0) in the same cluster, thus the prediction is very accurate (rmse = 0, when $k < 60$).

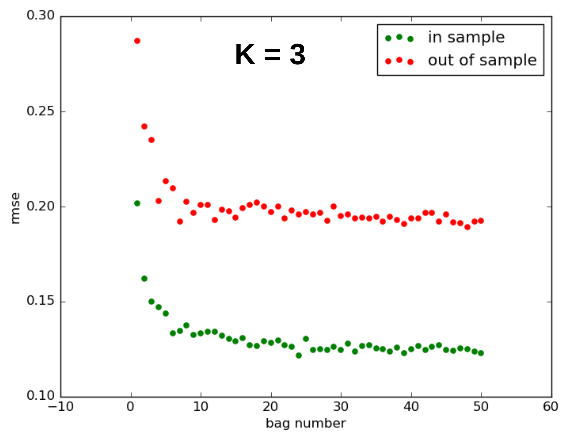
- Consider the dataset ripple with KNN. For which values of K does overfitting occur? (Don't use bagging).

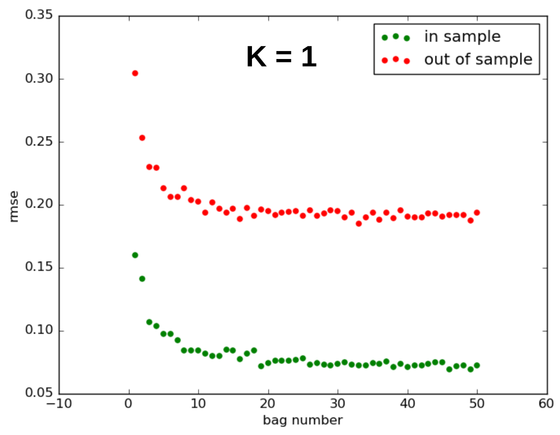
  For KNN learner, in the train set, rmse reduces when k decreases, and rmse reaches minima (rmse =0) when k equals 1. However, in the test set, rmse does not always decrease when k decreases, and when k decreases to a certain value, the further decrease of k leads the increase of rmse (the sign of overfitting). Based on my experiment on ripple dataset (as shown in the following figure), when k < 3, the overfitting occurs.



- 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?

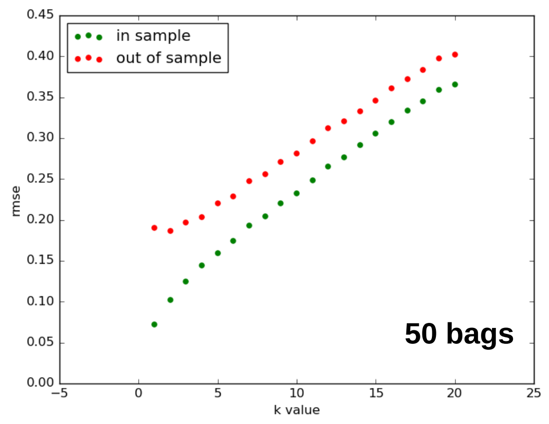
  Based on the experiment on ripple dataset ( k = 1, 3, bags = 1 ~ 50), rmse for train set and test set both decrease significantly at the beginning, then almost keep flat with further increase of bag number. No overfitting occurs during the increase of bag number.

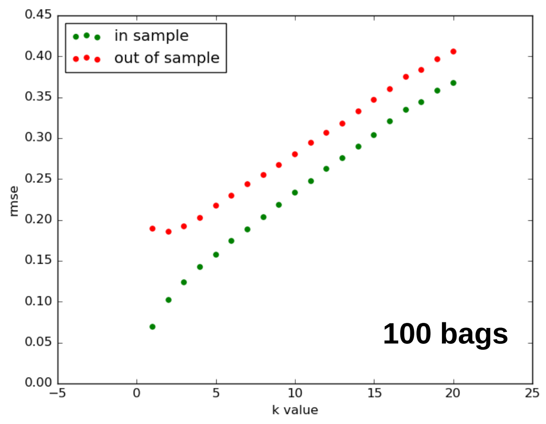


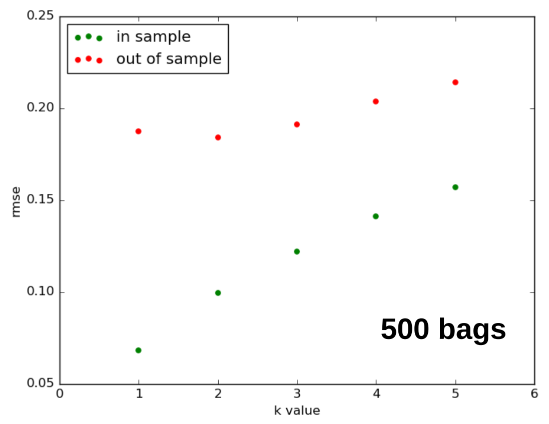


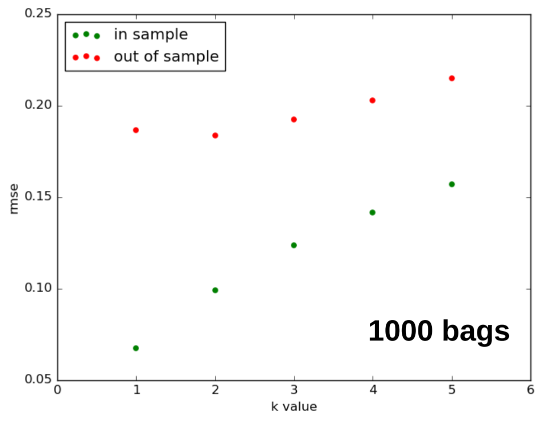
- Can bagging reduce or eliminate overfitting with respect to K for the ripple dataset?

  Base on the experiment using ripple data (bags = 10, 20, 50, 100, 500, 1000), bagging reduce overfitting, but cannot eliminate overfitting. For KNNLearner, overfitting happens when k < 3 (k= 1, 2). For bag learner, there is still an overfitting when k = 1.


in sample
out of sample
50 bags
rmse
k value


in sample
out of sample
100 bags
rmse
k value


in sample
out of sample
500 bags
rmse
k value


in sample
out of sample
1000 bags
rmse
k value