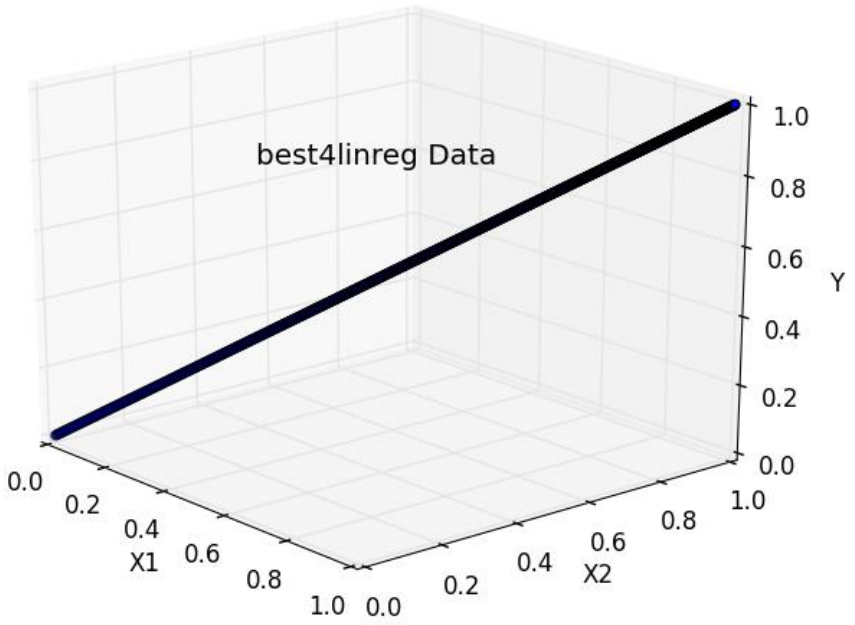
Daniel Rozen Report.pdf drozen3

Part 3: Experiments and report (50%)

Create a report that addresses the following issues/questions. The report should be submitted as `report.pdf` in PDF format. Do not submit word docs or latex files. Include data as tables or charts to support each your answers. I expect that this report will be 4 to 10 pages.

- 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).

My dataset uses a 3d linear diagonal line in 1000 data points that goes from (x1,x2,y)=(0,0,0) to (1,1,1)

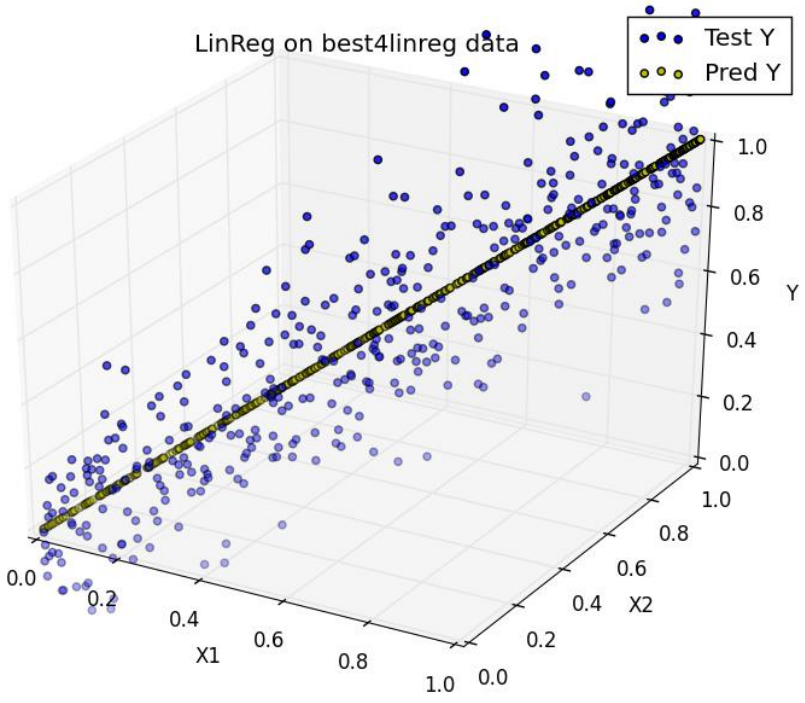


But then I add noise to it with `noise = np.random.normal(0,.2,1000)`
with a mean of 0 and std dev = .2

I randomly sampled 60% of the data without replacement for the training data and 40% for testing data.

As you can see above, the data set runs through 1000 points and (x1,x2,y) start at (0,0,0) and each increase by 1/1000 every step until they all reach (1,1,1).

Running this data on LinReg Learner produces the following results:



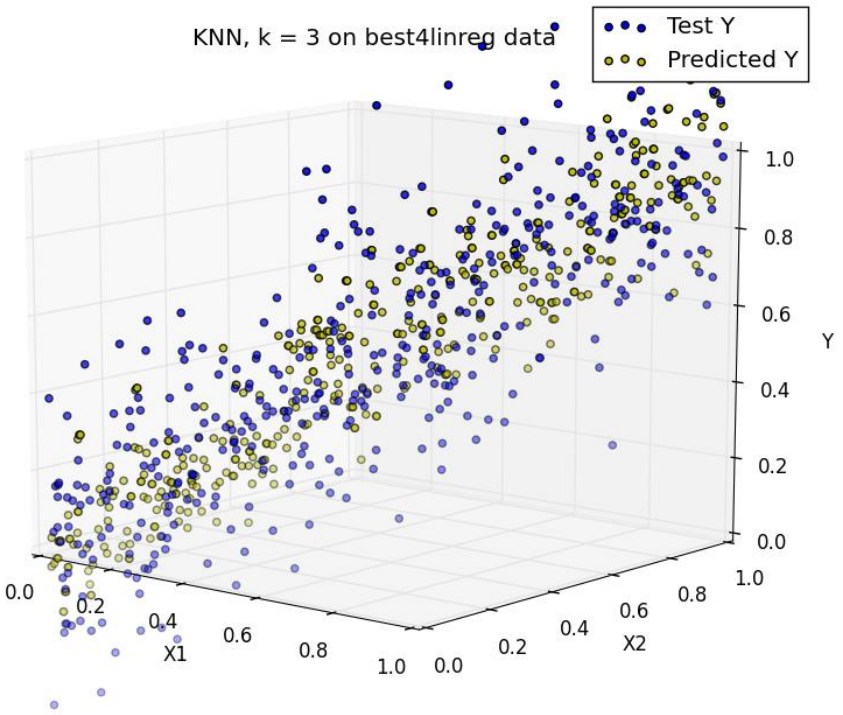
As you can see, it fits the noiseless data almost perfectly, and seems to ignore the noise.

LinReg Learner
Data/best4linreg.csv

In sample results
RMSE: 0.19830995698
corr: 0.82220810182

Out of sample results
RMSE: 0.197995756921
corr: 0.838960291973

However, KNN Learner doesn't perform as well as the following figure and data indicates:

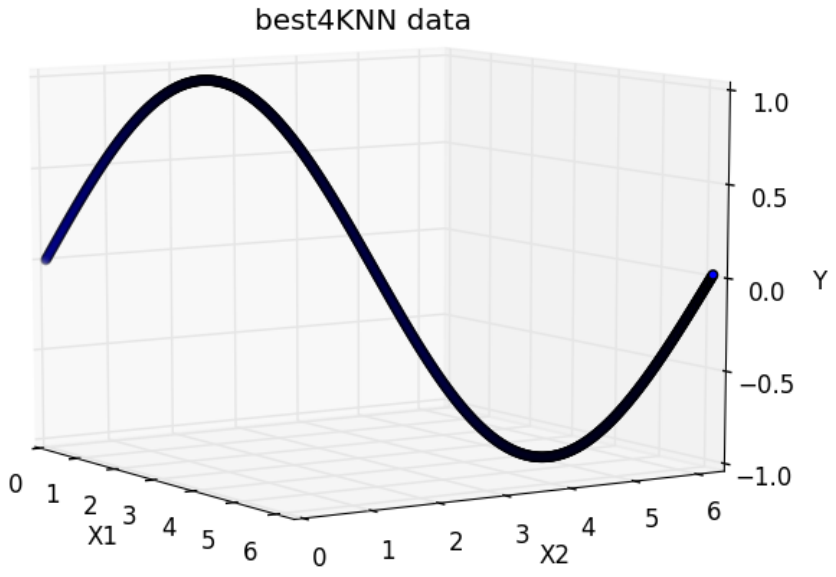


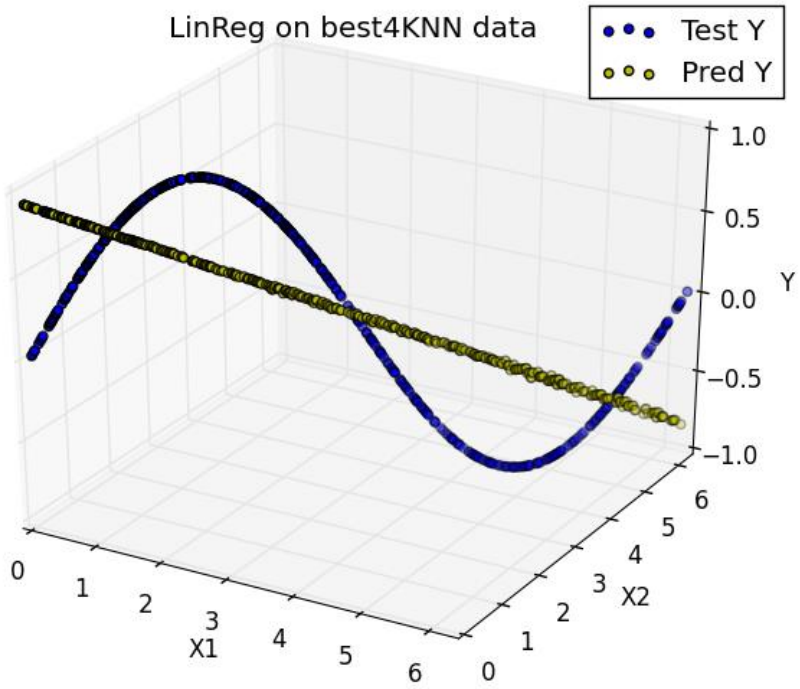
We see that KNN at k = 3 gets thrown off the line by the noise as it tried to model the noise and doesn't sufficiently get rid of the noise. The KNN predicted values are dispersed quite a bit around the real values of the straight line.

- 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).

  I used a 2d sine function in y. with x1 = x2 = (0, 6.3 (around 2*pi)) and y = sin(x1)

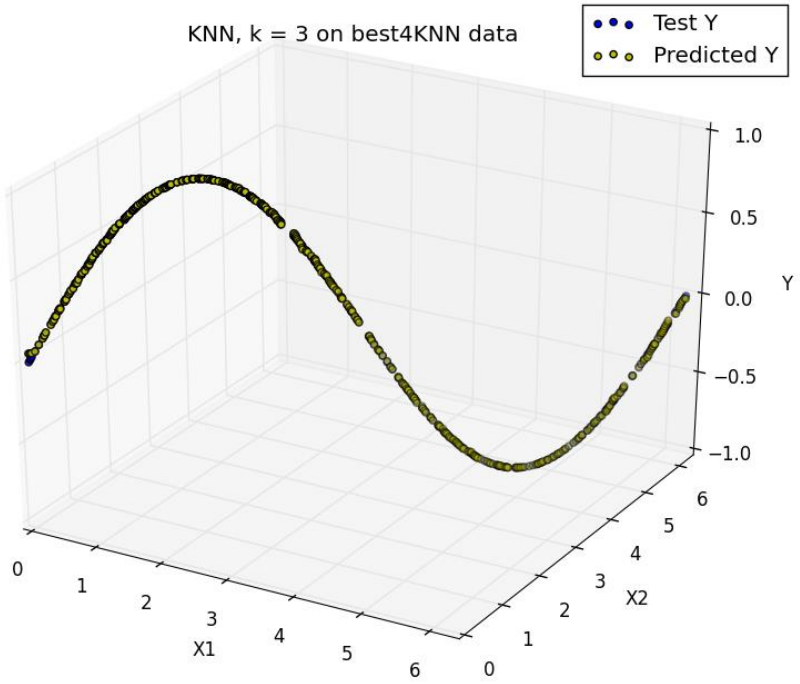
I randomly sampled 60% of the data without replacement for the training data and 40% for testing data.





We see Lin Reg does a poor job with fitting only a straight line estimating a sine curve. Lin reg can only estimate linear data, and therefore does poorly on exact curves.

For KNN we see much better results as KNN fits the Sine curve much better.



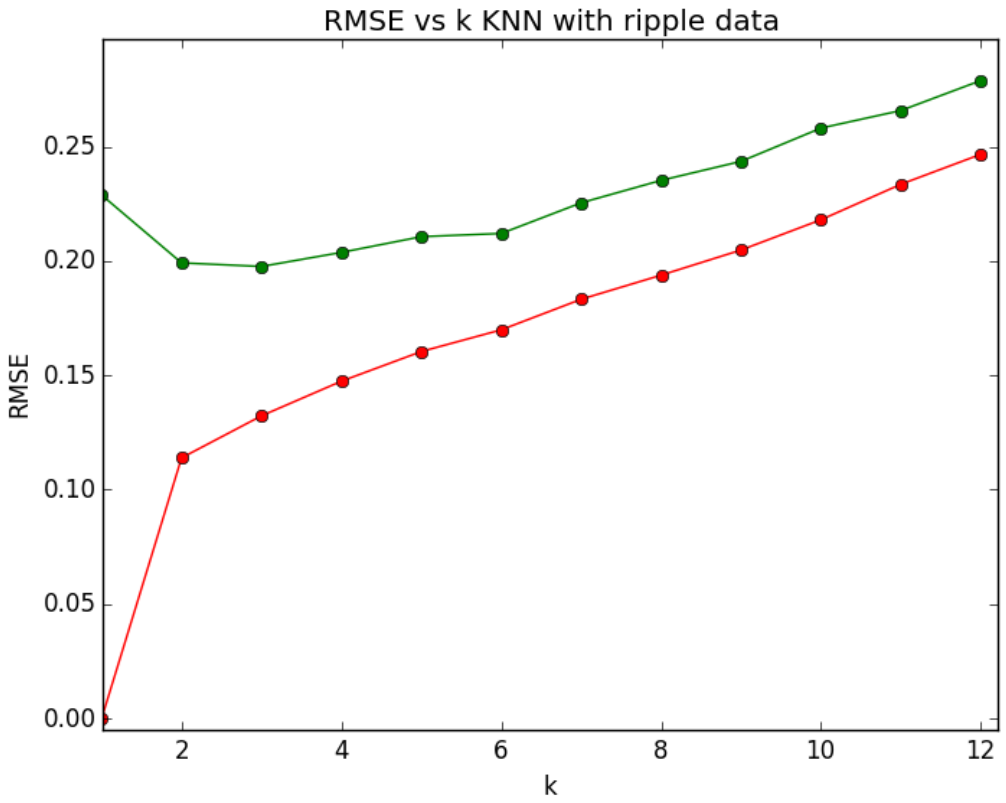
This is because KNN is based on the sample and not limited to a model, and tries to fit all of the data at the data points. And it also interpolates nicely and smoothly between data points.

Comparison results:

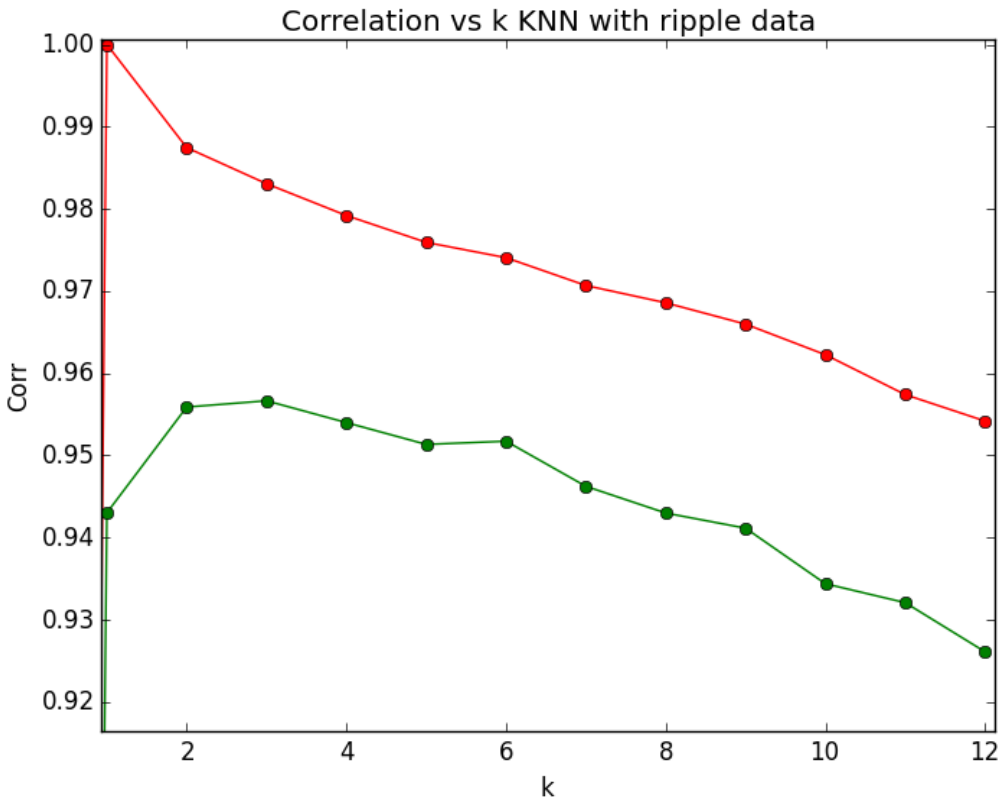
Data/best4KNN.csv	
LinReg Learner	KNN k = 3
In sample results	In sample results
RMSE: 0.454289654935	RMSE: 0.00311563536655
corr: 0.762939180835	corr: 0.999990173373
Out of sample results	Out of sample results
RMSE: 0.446261442069	RMSE: 0.00506943883105
corr: 0.792712330382	corr: 0.999974592019

We see almost perfect out of sample correlation with KNN, and an extremely lower out of sample RMSE 0.005 < 0.446 compared with LinReg.

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



We see that overfitting occurs at $k < 3$, since the out of sample RMSE begins diverging upwards, while the in sample RMSE diverges downwards away from the in sample RMSE at $k < 3$.



Additionally out of sample correlation decreases as k < 3 or k > 3. Highest correlation is found at k = 3.

Summary results:

k= 1 inRMSE= 0.0 inCorr= 1.0 outRMSE= 0.228847800228 outCorr= 0.943013029278

k= 2 inRMSE= 0.11395500116 inCorr= 0.987449575348 outRMSE= 0.19920248417 outCorr= 0.955882993726

k= 3 inRMSE= 0.132321054655 inCorr= 0.983057820753 outRMSE= 0.197630564718 outCorr= 0.956643598731

k= 4 inRMSE= 0.1474702198 inCorr= 0.979189869184 outRMSE= 0.203781242809 outCorr= 0.953992001699

k= 5 inRMSE= 0.160444342424 inCorr= 0.975922066871 outRMSE= 0.210663503719 outCorr= 0.951337098067

k= 6 inRMSE= 0.169939053988 inCorr= 0.974056727664 outRMSE= 0.212063456524 outCorr= 0.951727762039

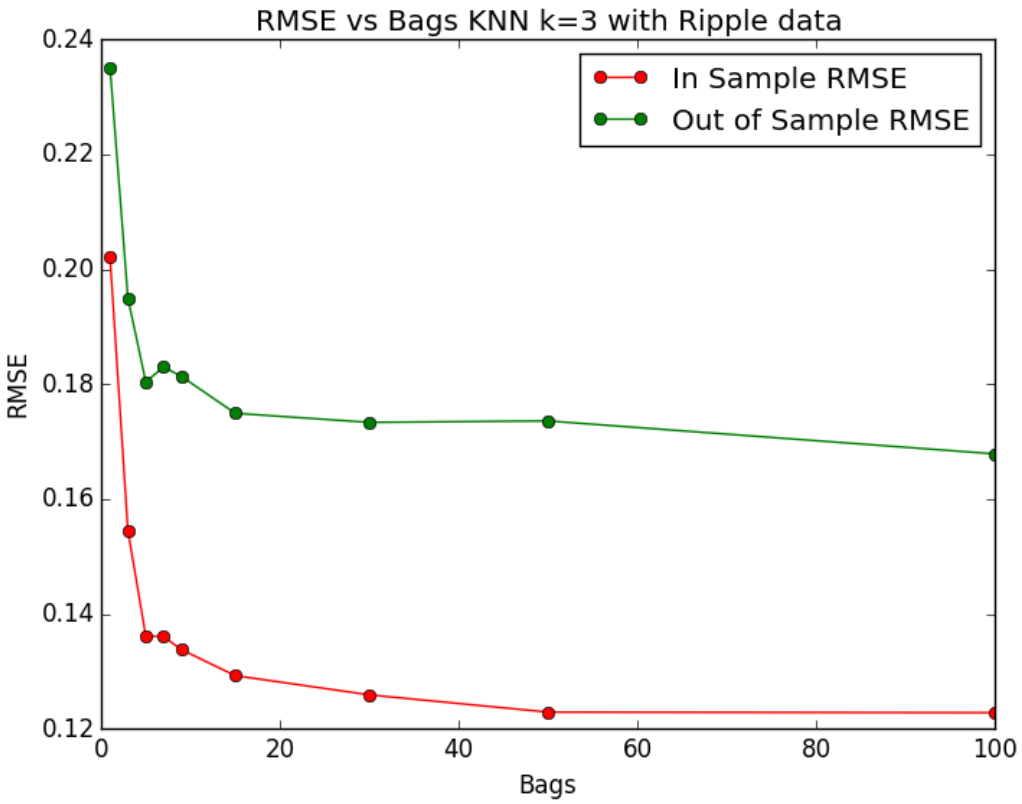
k= 7 inRMSE= 0.183304022472 inCorr= 0.970674155571 outRMSE= 0.225471806249 outCorr= 0.946238588467

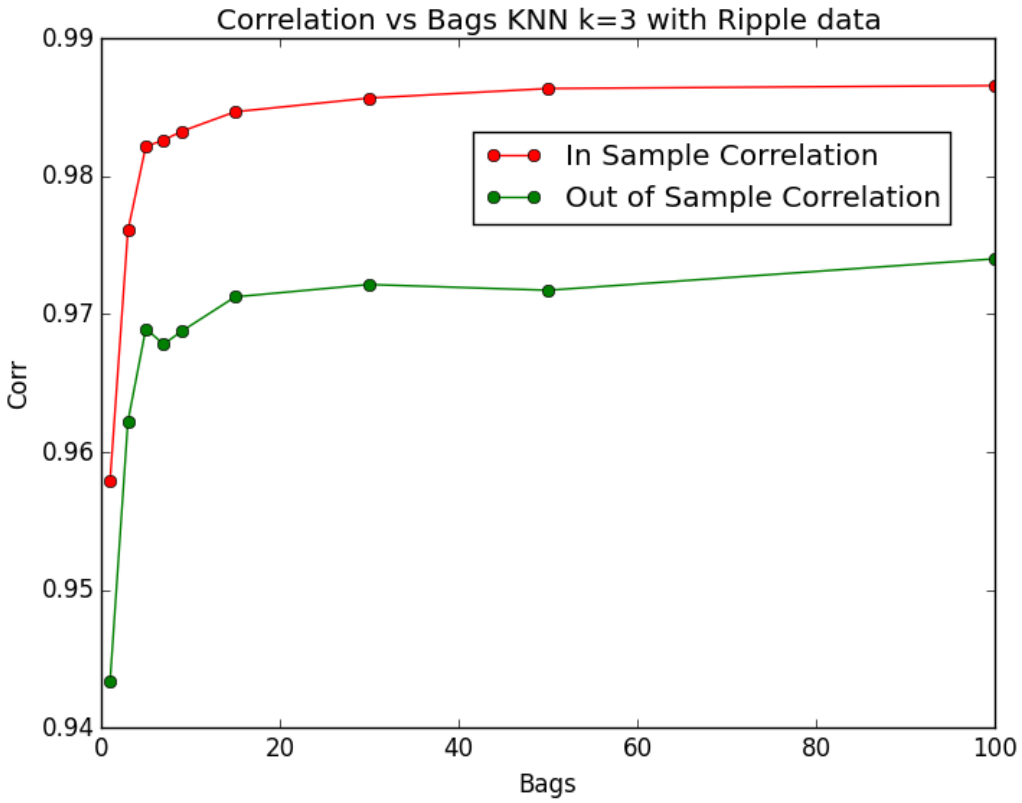
k= 8 inRMSE= 0.193830855874 inCorr= 0.968569053586 outRMSE= 0.235281885331 outCorr= 0.942986208734

k= 9 inRMSE= 0.204825827729 inCorr= 0.965972557716 outRMSE= 0.243597597143 outCorr= 0.941142610928

k= 10 inRMSE= 0.218067714011 inCorr= 0.96224802042 outRMSE= 0.258140912575 outCorr= 0.934383079854
k= 11 inRMSE= 0.233585798761 inCorr= 0.957395811705 outRMSE= 0.265820660639 outCorr= 0.932077426854
k= 12 inRMSE= 0.246693801167 inCorr= 0.954177946355 outRMSE= 0.278943451074 outCorr= 0.926100869114

- 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?





As we see in the above charts, there's a great increase in performances (due to a decrease in Out of sample RMSE and increase in Out of sample Correlation) from bags numbers of 1 to 5, a moderate increase from 5 to 15, and only a very slight increase from 15 to 100

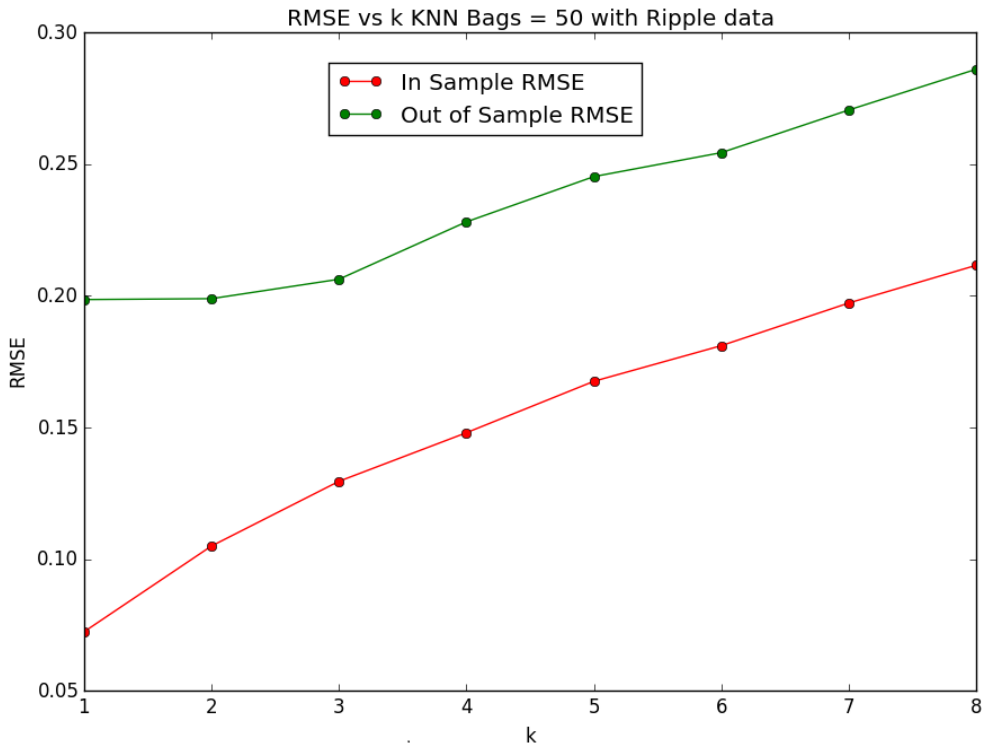
We find that increasing number of bags increases performance since we expect that an ensemble of learners trained on slightly different datasets will at least be able to provide some generalization and usually lower out of sample RMSE error.

Overfitting does not occur with respect to bagging as we would expect, as the greater the number of the bags, the less overfitting as the greater the generalization. We see this from the fact that we don't see a divergence of in sample versus out of sample RMSE or correlations away from each other as was the case in the above graphs of RMSE/Corr versus k.

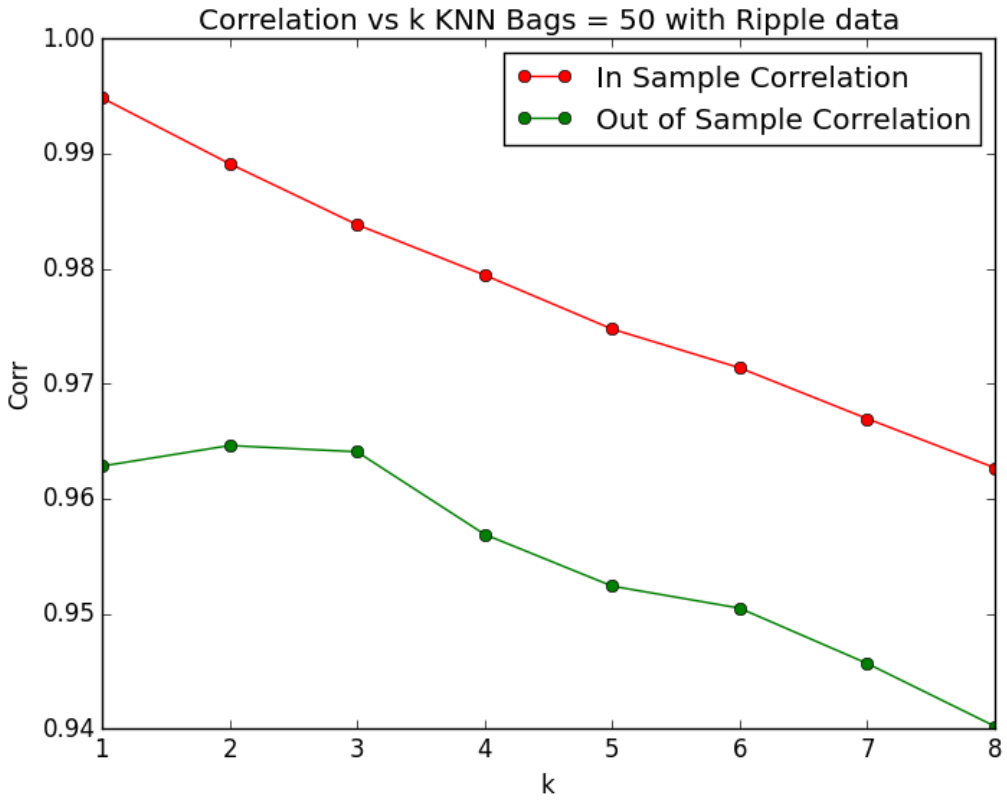
k = 3

Summary results:
bags = 1 inRMSE= 0.202074072047 inCorr= 0.957891126263 outRMSE= 0.235134728302 outCorr= 0.94333164922
bags = 3 inRMSE= 0.154481060986 inCorr= 0.976077768173 outRMSE= 0.194955231352 outCorr= 0.962221123866
bags = 5 inRMSE= 0.136160202115 inCorr= 0.98216316042 outRMSE= 0.180406455911 outCorr= 0.968937192142
bags = 7 inRMSE= 0.136041647384 inCorr= 0.982629568211 outRMSE= 0.182976156672 outCorr= 0.967826543668
bags = 9 inRMSE= 0.133773998454 inCorr= 0.983259368714 outRMSE= 0.181336990535 outCorr= 0.968758150689
bags = 15 inRMSE= 0.129264391448 inCorr= 0.984681638999 outRMSE= 0.174928940695 outCorr= 0.971257068106
bags = 30 inRMSE= 0.125916110271 inCorr= 0.985676678186 outRMSE= 0.173344254156 outCorr= 0.97215484928
bags = 50 inRMSE= 0.12290661847 inCorr= 0.98636768572 outRMSE= 0.173595660927 outCorr= 0.971736101352
bags = 100 inRMSE= 0.122813267807 inCorr= 0.986581537882 outRMSE= 0.167864248352 outCorr= 0.974014987821

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



As we see from the RMSE vs k graph above, bagging greatly reduces overfitting with respect to K for the ripple dataset, as bagging will compensate for overfitting for low k. It appears that bagging is completely eliminated as when k moves to less than 3, the out of sample RMSE actually slightly decreases instead of increasing when bagging wasn't used.



Correlation slightly decreases by about 1% which may be negligible.

Summary results:
Number of bags: 50
k = 1 inRMSE= 0.0723566318356 inCorr= 0.99483902148 outRMSE= 0.198546942473 outCorr= 0.96283099935
k = 2 inRMSE= 0.104943173811 inCorr= 0.989123307945 outRMSE= 0.198896104334 outCorr= 0.964618664897
k = 3 inRMSE= 0.129525583607 inCorr= 0.983819689854 outRMSE= 0.20626428804 outCorr= 0.964073082818
k = 4 inRMSE= 0.147984039857 inCorr= 0.97943572872 outRMSE= 0.228058971662 outCorr= 0.956898094932
k = 5 inRMSE= 0.1675179645 inCorr= 0.974744184974 outRMSE= 0.245263380373 outCorr= 0.952398331189
k = 6 inRMSE= 0.181055268527 inCorr= 0.971370427606 outRMSE= 0.254354282721 outCorr= 0.950478959485
k = 7 inRMSE= 0.197262416073 inCorr= 0.966951719036 outRMSE= 0.270560862423 outCorr= 0.945671668702
k = 8 inRMSE= 0.211585234043 inCorr= 0.96268432519 outRMSE= 0.286046442116 outCorr= 0.94021617475