In [3]:

```
##########Code for the KNN Learner:##########
from sklearn.neighbors import KNeighborsRegressor  #Imported Library for KNN
Regressor
from sklearn.ensemble import BaggingRegressor

#print Xtr.shape, Ytr.shape

#Create a list to store the top KnnRegressor Performers
topKnnPerformers = [None]*10

#Predict and print the MSE for training and test data on knnRegressors from 1
to 20 Nearest Neighbors
for K in range(1, 21):
    knnRegressor = KNeighborsRegressor(n_neighbors = K)
    yhatTr = knnRegressor.fit(Xtr, Ytr).predict(Xtr)
    yhatVa = knnRegressor.fit(Xte, Yte).predict(Xte)
    print "MSE for K = {:>2} Neighbors     Error(Training) = {:>7.4f}     Err
or(Validation) = {:>7.4f}".format(
            K, np.mean((Ytr - yhatTr)**2), np.mean((Yte - yhatVa)**2))

    #Since the first 10 regressors
    if (K <= 10):
        topKnnPerformers[K-1] = knnRegressor
```

```
MSE for K =  1 Neighbors     Error(Training) =  0.0000     Error(Validation)
=  0.0000
MSE for K =  2 Neighbors     Error(Training) =  0.1863     Error(Validation)
=  0.2122
MSE for K =  3 Neighbors     Error(Training) =  0.2555     Error(Validation)
=  0.2805
MSE for K =  4 Neighbors     Error(Training) =  0.2932     Error(Validation)
=  0.3197
MSE for K =  5 Neighbors     Error(Training) =  0.3175     Error(Validation)
=  0.3458
MSE for K =  6 Neighbors     Error(Training) =  0.3352     Error(Validation)
=  0.3676
MSE for K =  7 Neighbors     Error(Training) =  0.3486     Error(Validation)
=  0.3826
MSE for K =  8 Neighbors     Error(Training) =  0.3578     Error(Validation)
=  0.3965
MSE for K =  9 Neighbors     Error(Training) =  0.3655     Error(Validation)
=  0.4033
MSE for K = 10 Neighbors     Error(Training) =  0.3719     Error(Validation)
=  0.4087
MSE for K = 11 Neighbors     Error(Training) =  0.3771     Error(Validation)
=  0.4145
MSE for K = 12 Neighbors     Error(Training) =  0.3812     Error(Validation)
=  0.4190
MSE for K = 13 Neighbors     Error(Training) =  0.3855     Error(Validation)
=  0.4221
MSE for K = 14 Neighbors     Error(Training) =  0.3895     Error(Validation)
=  0.4259
MSE for K = 15 Neighbors     Error(Training) =  0.3928     Error(Validation)
=  0.4289
MSE for K = 16 Neighbors     Error(Training) =  0.3964     Error(Validation)
=  0.4311
MSE for K = 17 Neighbors     Error(Training) =  0.3993     Error(Validation)
```

```
=  0.4340
MSE for K = 18 Neighbors     Error(Training) =  0.4020     Error(Validation)
=  0.4370
MSE for K = 19 Neighbors     Error(Training) =  0.4041     Error(Validation)
=  0.4393
MSE for K = 20 Neighbors     Error(Training) =  0.4062     Error(Validation)
=  0.4417
```

In [7]:
```
print(topKnnPerformers) #checking to see if the first 10 knn neighbors were s
tored in the list
```

```
[KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
          metric_params=None, n_jobs=1, n_neighbors=1, p=2,
          weights='uniform'), KNeighborsRegressor(algorithm='auto', leaf_size
=30, metric='minkowski',
          metric_params=None, n_jobs=1, n_neighbors=2, p=2,
          weights='uniform'), KNeighborsRegressor(algorithm='auto', leaf_size
=30, metric='minkowski',
          metric_params=None, n_jobs=1, n_neighbors=3, p=2,
          weights='uniform'), KNeighborsRegressor(algorithm='auto', leaf_size
=30, metric='minkowski',
          metric_params=None, n_jobs=1, n_neighbors=4, p=2,
          weights='uniform'), KNeighborsRegressor(algorithm='auto', leaf_size
=30, metric='minkowski',
          metric_params=None, n_jobs=1, n_neighbors=5, p=2,
          weights='uniform'), KNeighborsRegressor(algorithm='auto', leaf_size
=30, metric='minkowski',
          metric_params=None, n_jobs=1, n_neighbors=6, p=2,
          weights='uniform'), KNeighborsRegressor(algorithm='auto', leaf_size
=30, metric='minkowski',
          metric_params=None, n_jobs=1, n_neighbors=7, p=2,
          weights='uniform'), KNeighborsRegressor(algorithm='auto', leaf_size
=30, metric='minkowski',
          metric_params=None, n_jobs=1, n_neighbors=8, p=2,
          weights='uniform'), KNeighborsRegressor(algorithm='auto', leaf_size
=30, metric='minkowski',
          metric_params=None, n_jobs=1, n_neighbors=9, p=2,
          weights='uniform'), KNeighborsRegressor(algorithm='auto', leaf_size
=30, metric='minkowski',
          metric_params=None, n_jobs=1, n_neighbors=10, p=2,
          weights='uniform')]
```

Using the KNeighborsRegressor from the sklearn library a knnregressor learner was constructed

with neigbors ranging from 1 to 20. It was expected, as the commentary mentioned on the project writeup that the KNN learners would take some time to train and predict. The process was ended up being fast (30 minutes - 1 hour) and the resulting MSE was very low giving us good results for the FULL data set. As the results are seen above, the error on the training data was < ~.41 and < ~.44 for the test data. Because the model starts underfitting for larger values of K we decided to move forward with neighbors less than K = 10, meaning taking the top 10 performers to store into the ensemble.

In [ ]:
```
topEnsemblePerformers = [None]*10

for i, regressor in enumerate(topKnnPerformers):

    knnEnsemble = BaggingRegressor(base_estimator = topKnnPerformers[i], n_es
timators = 10)
    yhatTr = knnEnsemble.fit(Xtr, Ytr).predict(Xtr)
    yhatVa = knnEnsemble.fit(Xte, Yte).predict(Xte)

    print "MSE for Ensemble with a K = {:>2} Neighbors Regressor     Error(Tr
aining) = {:>7.4f}     Error(Validation) = {:>7.4f}".format(
            regressor.n_neighbors, np.mean((Ytr - yhatTr)**2), np.mean((Yte -
yhatVa)**2))
```

MSE for Ensemble with a K = 1 Neighbors Regressor Error(Training) = 0.0955 Error(Validation) = 0.1055
MSE for Ensemble with a K = 2 Neighbors Regressor Error(Training) = 0.1729 Error(Validation) = 0.1936
MSE for Ensemble with a K = 3 Neighbors Regressor Error(Training) = 0.2359 Error(Validation) = 0.2596

Playing around with the BaggingRegressor library, there was no way to store the 10

topKnnPerformers into one Ensemble so we tested storing 10 of each topKnnPerformers and train/predicted to see the results. We found that since it was taking too long to output the MSE and this approach just did not make sense, so we did not move forward with it. However, it is worth noting the drop in MSE for the 3 outputs that did end up printing as shown above. Afterwards, reading through library documentation we moved forward with another approach below.

In [18]:
```
knnEnsemble = BaggingRegressor(n_estimators = len(topKnnPerformers))
knnEnsemble.estimators_ = topKnnPerformers
yhatTr = knnEnsemble.fit(Xtr, Ytr).predict(Xtr)
yhatVa = knnEnsemble.fit(Xte, Yte).predict(Xte)

print "MSE for Ensemble with top 10 KnnRegressors     Error(Training) = {:>7.
4f}     Error(Validation) = {:>7.4f}".format(
       np.mean((Ytr - yhatTr)**2), np.mean((Yte - yhatVa)**2))
```

```
MSE for Ensemble with top 10 KnnRegressors     Error(Training) =  0.0719
Error(Validation) =  0.0773
```

MSE for Ensemble with top 10 KnnRegressors Error(Training) = 0.0719 Error(Validation) = 0.0773

Since BaggingRegressors library had an attribute to store subset of estimators, we decided to take our list of topPerformers trained previously and store this list to that attribute. This allowed us to train and predicted on the entire list of learners rather than having to indivudally store them in a BaggingRegressor with 10 copies of each knnRegressor. After training and predicting on this entire list of already well performing learners, the result was an even lower MSE!

In [9]:
```
#Combining the Ensemble

#To output to Kaggle
#Based off of our results above we will use the optimum values given out by t
he knnEnsemble above as well as
#store the optimum Decision Tree leaf value of 2^8 into its own Ensemble repr
edict and average it with knnEnsemble prediction

avgPredTeList = []

optimumKnnEnsemble = BaggingRegressor(n_estimators = len(topKnnPerformers))
optimumDtEnemble = BaggingRegressor(
    base_estimator = tree.DecisionTreeRegressor(max_depth = 20, min_samples_l
eaf = 2**8), n_estimators = 10)

KnnTr = optimumKnnEnsemble.fit(Xtr, Ytr)
DtTr = optimumDtEnemble.fit(Xtr, Ytr)
```

The code above takes the Knn values based on our previous test runs earlier of KNearestNieghbors of up to 10 as well as the optimum Decsion Tree Ensemble based on our previous findings of min_samples_leafs @ 2^8 and retrained aftering storing each to their own Ensemble.

In [15]:
```
yhatKnnTr = KnnTr.predict(Xe1)
yhatDtTr = DtTr.predict(Xe1)

avgPredTeList = (yhatKnnTr + yhatDtTr) / 2
print(len(avgPredTeList))

fh = open('predictions.csv','w') # open file for upload
fh.write('ID,Prediction\n') # output header line
for i,yi in enumerate(avgPredTeList):
    fh.write('{},{}\n'.format(i+1,yi)) # output each prediction

fh.close() # close the file
```

```
40000
```

Because these were our top learners we decided to take the predictions of both the Decision Tree Ensemble and the K nearest Ensemble and average these values to upload to Kaggle. Because of the large amount of time it takes to build these learners and the Ensemble version of these learners, we were only able to upload our predictions once and not reupload with improved prediction quality.

Conclusion

In conclusion, since our document ipython notebook size was exceeding far beyond 6 pages and we were not sure if our writeup was suppose to be in a seperate document, we decided to do our write up as we were writing and testing our code (as shown above). At the beginning we had decided to go more into detail with Ensembles by using learners that we did not go into detail with in class. This enabled us to use 3 different types of learners to equally balance the workload. Rodrigo, started with the intent of using Neural Networks until he realized that they only work with

classification models. He decide to switch to Support Vector Machines, also a learner we did not apply in our homeworks. Because the Support Vector Machine took too long to train on the entire dataset, a smaller subset was used. The result of predicting on these smaller subset were very high MSEs for both the training and test data. Because the SVM was constantly underfitting and unable to improve we came to the decision of removing it from the final ensemble. The decision tree, the learner tested by Mayra, had mixed performances. Aftering training and prediction on the decision tree using the library it looked as the depth increased the MSE decreased. Iterating thru different values of the leaf_node parameter, Mayra found that the decision tree at leaf node 2^8 had the optimum learner so we decided to store this learner into the final ensemble. The third learner, explored by Muzamil, was going into detail with K Nearest Nieghbors, but as apposed to using classifiers as in the previous homework, he used the library's knnRegressor model. He decided to train the KNN mode of up to 20 neighbors and the performance, as seen above, turned out to be very good. It was noticed, as expected that the more neighbors you have the more the model was starting to underfit as seen in the increase in the MSEs. He decided to take knnRegressors from 2-10 neighbors to store into the Ensemble. At first an attempt was made to try to use these 10 different learners in the baggingRegressor library but the only way to do this was store multiple of the same learners into baggingRegressor meaning he would have had to create 10 different ensembles to train on. After looking more into the library an attribute in the baggingRegressor was discovered where a list of sublearners can be set. He decided to take this approach and train the ensemble on the list of 10 KnnRegressors of up to 10 neighbors. The training and predicted process ended up being quick and the MSE was even less! We decided to take the predictions along with the optimum Decision Tree learner to upload to Kaggle. Overall, our final goal of creating an emsemble containing 7 or so of each our learners did not go as planned so we decided to choose the 2 learners and use 10 of each of those. There were many times that we could not see the results of our trainers due to very long run times. If something were to be changed in order to get the results we intended, we could probably use partitions of the dataset whenever possible.