CS 4375 Final Project Report

Kaggle Competition: House Prices - Advanced Regression Techniques

Mehedi Toufiqe (mht190001)

Dien Tran (dkt180000)

Carter Webb (cjw180000)

Kevin Tran (kxt170006)

Presentation Link

https://youtu.be/tx_vs_4ZiRo

Introduction

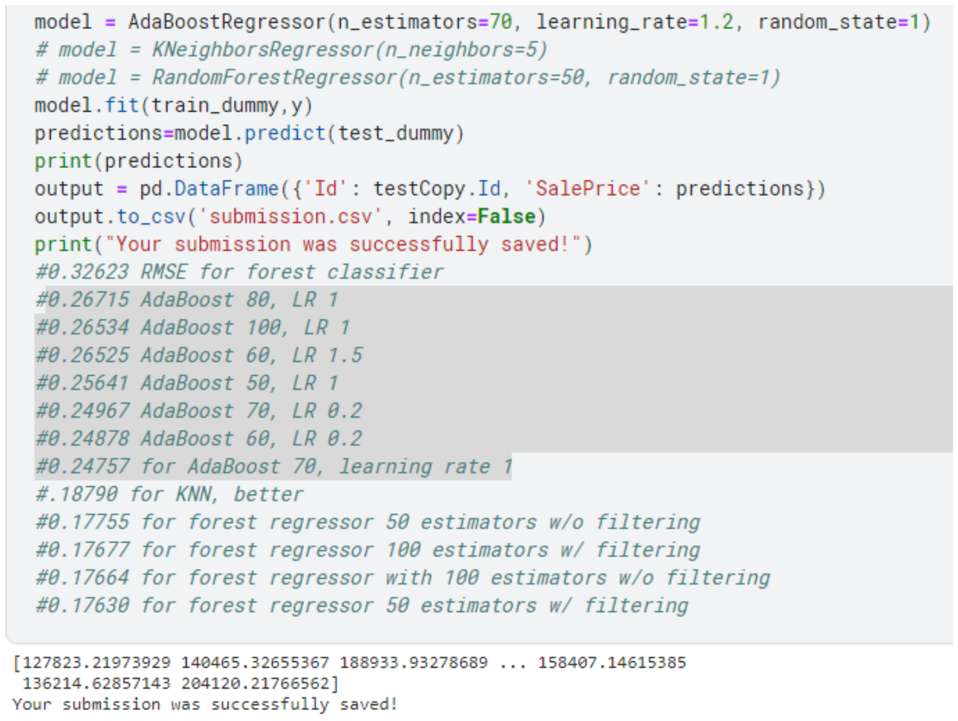
For our final project, we choose Kaggle Competition; House Prices - Advanced Regression Techniques (Predict sales prices and practice feature engineering, RFs, and gradient boosting). Link: https://www.kaggle.com/c/house-prices-advanced-regression-techniques . We choose this dataset because this dataset is pretty simple to understand and we don't have to spend much time cleaning the dataset. We only have one target to predict.

Our target is to implement four regressor methods; Random Forest Regressor, Ada Boost Regressor, KNeighbors Regressor, and Bagging Regressor. Our goal is to compare the prediction and decide which one is best for this competition.

AdaBoost Attempts

I tried using AdaBoost because it's a recent topic we learned, so it shouldn't hurt for me to try it out.

Results are the following:



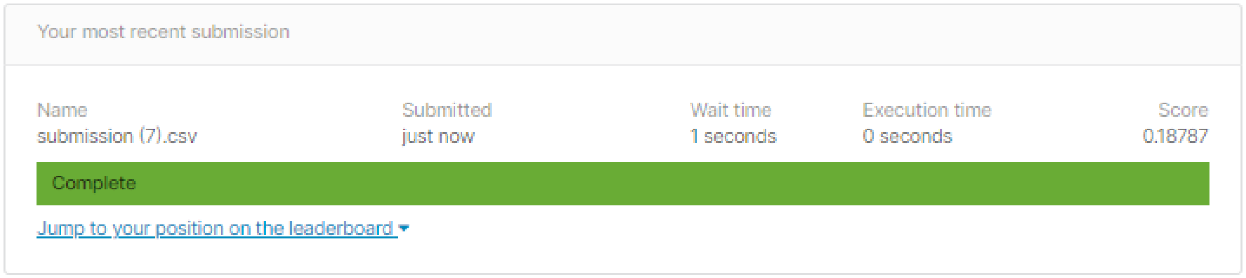
```
model = AdaBoostRegressor(n_estimators=70, learning_rate=1.2, random_state=1)
# model = KNeighborsRegressor(n_neighbors=5)
# model = RandomForestRegressor(n_estimators=50, random_state=1)
model.fit(train_dummy,y)
predictions=model.predict(test_dummy)
print(predictions)
output = pd.DataFrame({'Id': testCopy.Id, 'SalePrice': predictions})
output.to_csv('submission.csv', index=False)
print("Your submission was successfully saved!")
#0.32623 RMSE for forest classifier
#0.26715 AdaBoost 80, LR 1
#0.26534 AdaBoost 100, LR 1
#0.26525 AdaBoost 60, LR 1.5
#0.25641 AdaBoost 50, LR 1
#0.24967 AdaBoost 70, LR 0.2
#0.24878 AdaBoost 60, LR 0.2
#0.24757 for AdaBoost 70, learning rate 1
#.18790 for KNN, better
#0.17755 for forest regressor 50 estimators w/o filtering
#0.17677 for forest regressor 100 estimators w/ filtering
#0.17664 for forest regressor with 100 estimators w/o filtering
#0.17630 for forest regressor 50 estimators w/ filtering
```

```
[127823.21973929 140465.32655367 188933.93278689 ... 158407.14615385
 136214.62857143 204120.21766562]
Your submission was successfully saved!
```

I have tried various configurations with various numbers of estimators and learning rates. There was not a clear direction for me to go towards. It was not clear whether I should try tweaking the parameters more or use another method entirely.

K Nearest Neighbors

KNN is an algorithm that uses the K nearest points, calculated based on the values of the points, to classify an unclassified point. This can be used for regression by taking points with known values, such as the training set of data, and taking a weighted average based on distance of their values to fill in the missing values of an incomplete point, like those in a test set. The main variable in K nearest neighbors is the value of K, a value too low can lead to a model not being precise enough, but a value that is too large may lead to a model that is overfitted on the training data that is not effective at predicting values outside of the training set. KNN was chosen as a possible solution because it is one of the machine learning methods that has a history of being used for regression, and it was likely that similar houses would have correspondingly similar prices. One strange thing found in testing was that despite removing columns that contain mainly null values improving the performance of other algorithms, it worsens the performance of KNN. The results of KNN without dropping primarily null columns, with influence weighted by distance, and with K=10 was a RMSE of .18787, which was how this competition was scored.


Your most recent submission

Name	Submitted	Wait time	Execution time	Score
submission (7).csv	just now	1 seconds	0 seconds	0.18787

Complete

Jump to your position on the leaderboard

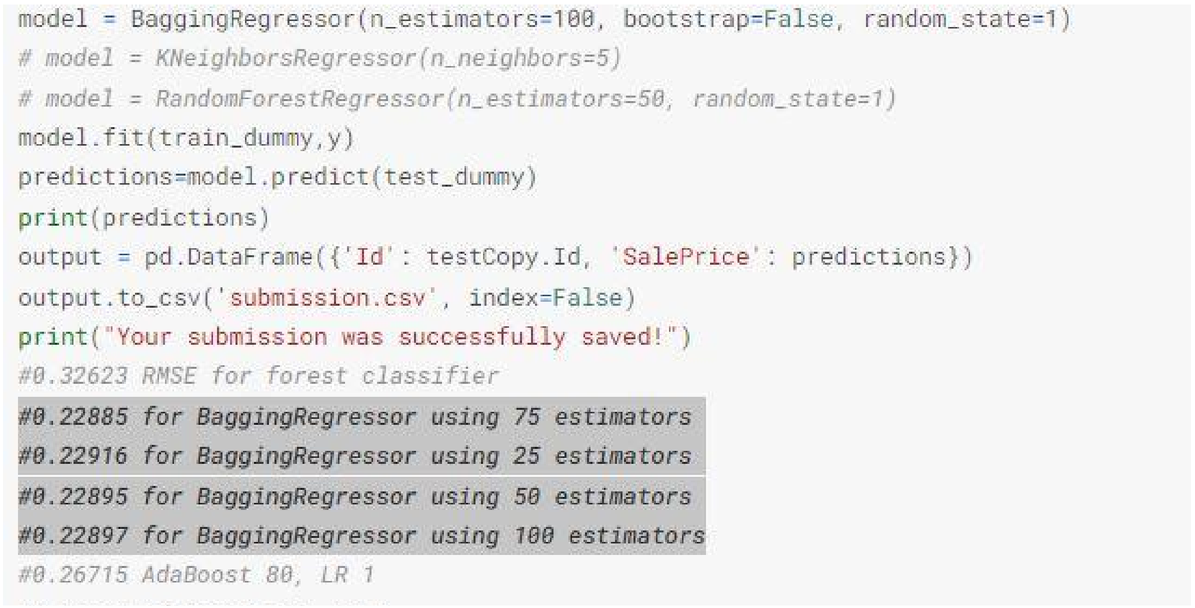
These results could likely be improved by further tuning the value of k to obtain a more accurate model. Tuning ability is limited by kaggle only allowing 10 submissions per day. Another way to try to improve the model is figuring out what attributes of the data have little to no impact on the house price to reduce the noise in the training set and therefore create a better model. A related method to improve the results would be finding out what attributes are stronger indicators of the house price and giving them more impact in calculating distance, ensuring the most relevant attributes shape the regression as much as possible.

```python
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.ensemble import BaggingRegressor
train=train.drop('Id',axis=1)
test=test.drop('Id',axis=1)
y=train["SalePrice"]
#print(features)
train['label']='train'
test['label']='test'
features=train.columns[1:]
combined= pd.concat([train,test])
x=pd.get_dummies(combined,columns=train.columns)
#print(x.columns)
#print(x['label_test'])
train_dummy=x[x['label_test']==0]
test_dummy=x[x['label_test']==1]
#print(train_dummy)
train_dummy=train_dummy.drop('label_test',axis=1)
train_dummy=train_dummy.drop('label_train',axis=1)
test_dummy=test_dummy.drop('label_test',axis=1)
test_dummy=test_dummy.drop('label_train',axis=1)

# model = AdaBoostRegressor(n_estimators=100, random_state=1)
model = KNeighborsRegressor(n_neighbors=10, weights='distance')
# model = RandomForestRegressor(n_estimators=50, random_state=1)
# model= BaggingRegressor(n_estimators=50, bootstrap=False, random_state=1)
model.fit(train_dummy,y)
predictions=model.predict(test_dummy)
print(predictions)
output = pd.DataFrame({'Id': testCopy.Id, 'SalePrice': predictions})
output.to_csv('submission.csv', index=False)
print("Your submission was successfully saved!")
#0.32623 RMSE for forest classifier
#0.24757 for AdaBoost 70
#0.22895 for BaggingRegressor
#.18790 for KNN, better. performed worse with filteriing
#0.17755 for forest regressor 50 estimators w/o filtering
#0.17677 for forest regressor 100 estimators w/ filtering
#0.17664 for forest regressor with 100 estimators w/o filtering
#0.17630 for forest regressor 50 estimators w/ filtering
```

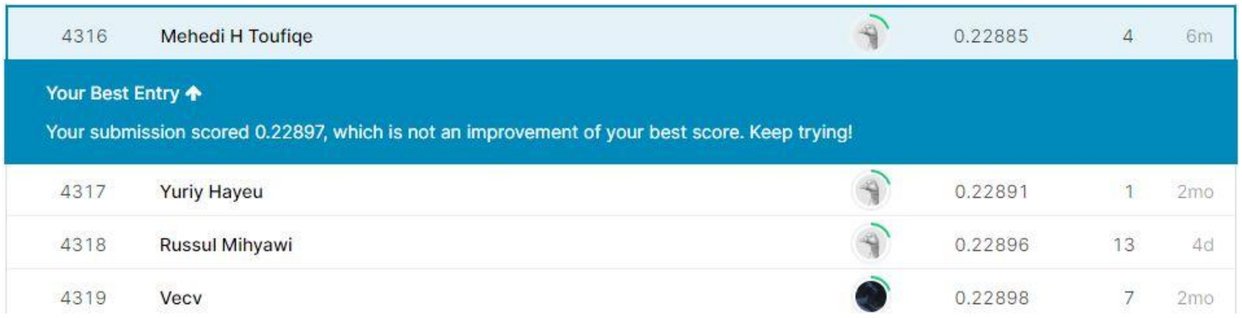
Bagging Regressor

The approach known as bagging induces a group of classifiers, each from a different subset of the training data. When presented with an example to be classified, the classifiers are all applied in parallel, each offering an opinion about the example's class. A master classifier then decides which label got more votes. A Bagging regressor is an ensemble meta-estimator that fits base regressors each on random subsets of the original dataset and then aggregates their individual predictions (either by voting or by averaging) to form a final prediction. Such a meta-estimator can typically be used as a way to reduce the variance of a black-box estimator (e.g., a decision tree), by introducing randomization into its construction procedure and then making an ensemble out of it. For this project, I'm using the bagging regressor to find a better rmse for our prediction.



```
model = BaggingRegressor(n_estimators=100, bootstrap=False, random_state=1)
# model = KNeighborsRegressor(n_neighbors=5)
# model = RandomForestRegressor(n_estimators=50, random_state=1)
model.fit(train_dummy,y)
predictions=model.predict(test_dummy)
print(predictions)
output = pd.DataFrame({'Id': testCopy.Id, 'SalePrice': predictions})
output.to_csv('submission.csv', index=False)
print("Your submission was successfully saved!")
#0.32623 RMSE for forest classifier
#0.22885 for BaggingRegressor using 75 estimators
#0.22916 for BaggingRegressor using 25 estimators
#0.22895 for BaggingRegressor using 50 estimators
#0.22897 for BaggingRegressor using 100 estimators
#0.26715 AdaBoost 80, LR 1
```

As we can see for the lowest estimator 25 we get rmse of 0.22916 and for the highest estimator, we get rmse of 0.22897. This BaggingRegressor method is not good for this prediction because when we are using a random forest regressor we can get a lower mean squared logarithmic error. Below my leaderboard position;



4316	Mehedi H Toufiqe	0.22885	4	6m

Your Best Entry
Your submission scored 0.22897, which is not an improvement of your best score. Keep trying!

4317	Yuriy Hayeu	0.22891	1	2mo
4318	Russul Mihyawi	0.22896	13	4d
4319	Vecv	0.22898	7	2mo

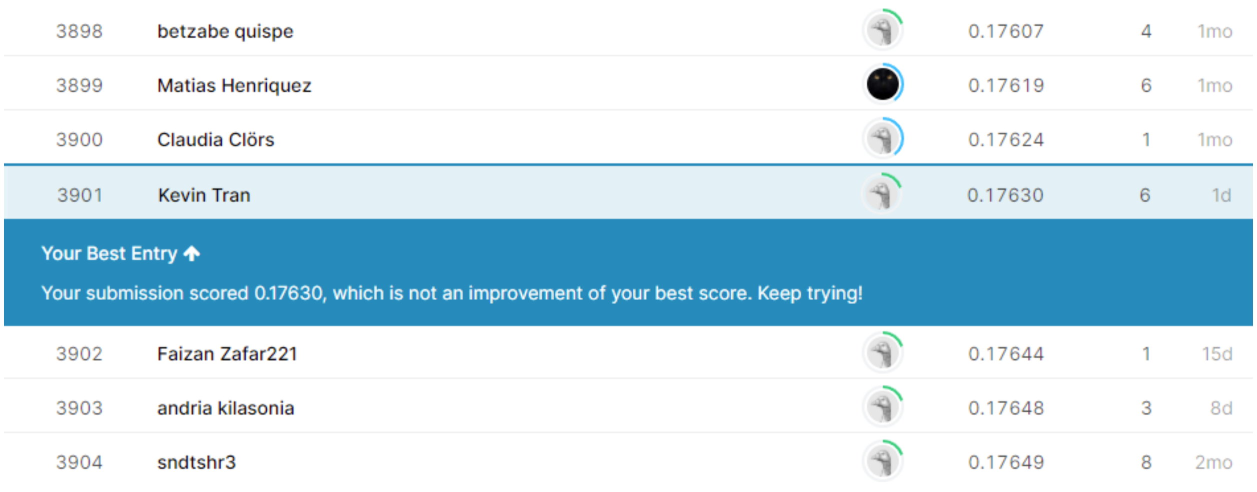
Random Forest Regressor

The random forest regressor is a great general model to use because it is very simple to use and requires little tuning to get a decently accurate score. It is also a widely used algorithm on Kaggle that is very effective. A random forest regressor is like a bagging decision tree except that it chooses a random set of input features to consider and would split each decision based on those random features. Against the other regressors, it seemed to have the most accuracy, or in this case, a lower root mean squared logarithmic error. Additionally, we tested multiple different configurations to see how it affected the overall score.

```
#0.17755 for forest regressor 50 estimators w/o filtering
#0.17677 for forest regressor 100 estimators w/ filtering
#0.17664 for forest regressor with 100 estimators w/o filtering
#0.17630 for forest regressor 50 estimators w/ filtering
```

We found it resulted in similar scores despite some tuning yet it still performed the best. The filtering that is occurring is one where we removed the features that had several NaN values, thus we can conclude that random forests are not sensitive to NaN variables, or resulting homogenization.

Leaderboard Position:



3898	betzabe quispe	0.17607	4	1mo
3899	Matias Henriquez	0.17619	6	1mo
3900	Claudia Clörs	0.17624	1	1mo
3901	Kevin Tran	0.17630	6	1d

Your Best Entry

Your submission scored 0.17630, which is not an improvement of your best score. Keep trying!

3902	Faizan Zafar221	0.17644	1	15d
3903	andria kilasonia	0.17648	3	8d
3904	sndtshr3	0.17649	8	2mo

Despite this, we can probably further improve the score if we managed to hypertune the parameters of the random forest like finding an optimal max depth or max features that the regressor should split on. There also might be some features that correlate that we can simplify in order to improve the score even further.