CSC665 Artificial Section 01 Spring 2019

Kaggle Project – Regression Analysis

Team #27

Prof. Alex Kalinin

Team Members:

Ratna Lama Team Leader *rlama7@mail.sfsu.edu*

Rohit Nair

Michael Winata

Alexey Sergeev

Version History:

05/08/19: document created

05/21/19: document submitted for review

Content and Structure for the Kaggle Term Project

1. Dataset and Metrics
2. Model Selection
3. Training & Validation
4. Code
5. Acknowledgements

Content and Structure for the Kaggle Term Project

1. Dataset and Metrics

A. Tabular dataset

B. What metrics does Kaggle use to evaluate this competition?

➔

Kaggle submissions are evaluated on Root-Mean-Squared-Error (RMSE) between the logarithm of the predicted value and the logarithm of the observed sales price. (Taking logs means that errors in predicting expensive houses and cheap houses will affect the result equally.)

C. Provide exact formulas, and your implementation, or refer to the existing API (NumPy, Pytorch, etc.)

➔

a) Mean Squared Error (MSE)

MSE measures average squared error of predictions. For each point, it calculates square difference between the predictions and the target and then average those values.

The higher the MSE value, the worse the model is. MSE value is non-negative, since we are squaring the individual prediction-wise errors before summing them, but would be zero for an ideally perfect model.

Mathematically, MSE is expressed as below:

$$\boxed{\mathrm{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}$$

b) Root Mean Square Error (RMSE)

RMSE is the square root of MSE. RMSE makes the scale of the errors to be the same as the scale of targets. It represents the sample standard deviation of the difference between predicted values and observed values.

Since the square root is a non-decreasing function, every minimizer of MSE is also a minimizer of RMSE and vice-versa.

Mathematically, RMSE can be expressed as:

$$\mathrm{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2} = \sqrt{\mathrm{MSE}}$$

c) R Squared (R^2)

R squared also known as the coefficient of determination has the advantage of being scale-free. The R^2 is always going to be between -infinity and 1.

If R^2 value is negative then the model is worse than the predicting the mean.

For a good model the desired result of R^2 is closer to 1.

Mathematically, R^2 can be expressed as:

$$R^2 = 1 - \frac{\text{MSE(model)}}{\text{MSE(baseline)}}$$

$$\text{MSE(baseline)} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \bar{y})^2$$

MSE, RMSE, Score calculations

In [71]:
```
mse = ((y_hat - y_test) ** 2).mean()
```

In [72]:
```
rmse = np.sqrt(mse)
```

In [73]:
```
v = ((y_test - y_test.mean()) ** 2).mean()
```

In [74]:
```
mse, rmse, v
```

Out[74]:
```
(726231060.1455075, 26948.67455266599, 7005309052.339321)
```

In [75]:
```
score = (1 - (mse/v))
score
```

Out[75]:
```
0.8963313317485981
```

Score Calculation using Python Sci-kit library

Random Forest Regressor

```
In [19]:  rf = RandomForestRegressor(n_estimators=100, random_state=17)

          %time rf.fit(X_train, y_train)

          Wall time: 2.46 s

Out[19]:  RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                     max_features='auto', max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=None,
                     oob_score=False, random_state=17, verbose=0, warm_start=False)

In [20]:  rf.score(X_train, y_train)

Out[20]:  0.9787190039874757

In [21]:  rf.score(X_test, y_test)

Out[21]:  0.896331331748598
```

D. How many samples does the entire dataset have?

➔The entire dataset has 1460 samples

read CSV

```
In [49]:  df = pd.read_csv("train.csv")

In [50]:  df.shape

Out[50]:  (1460, 81)

In [51]:  df.head()

Out[51]:
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0

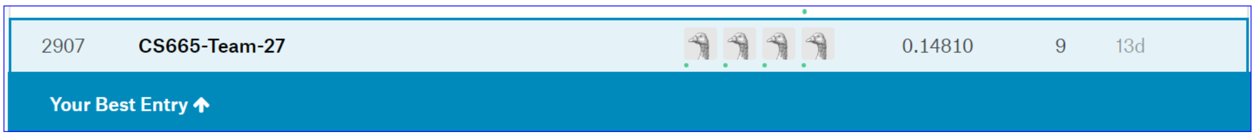
5 rows × 81 columns

E. How many features?

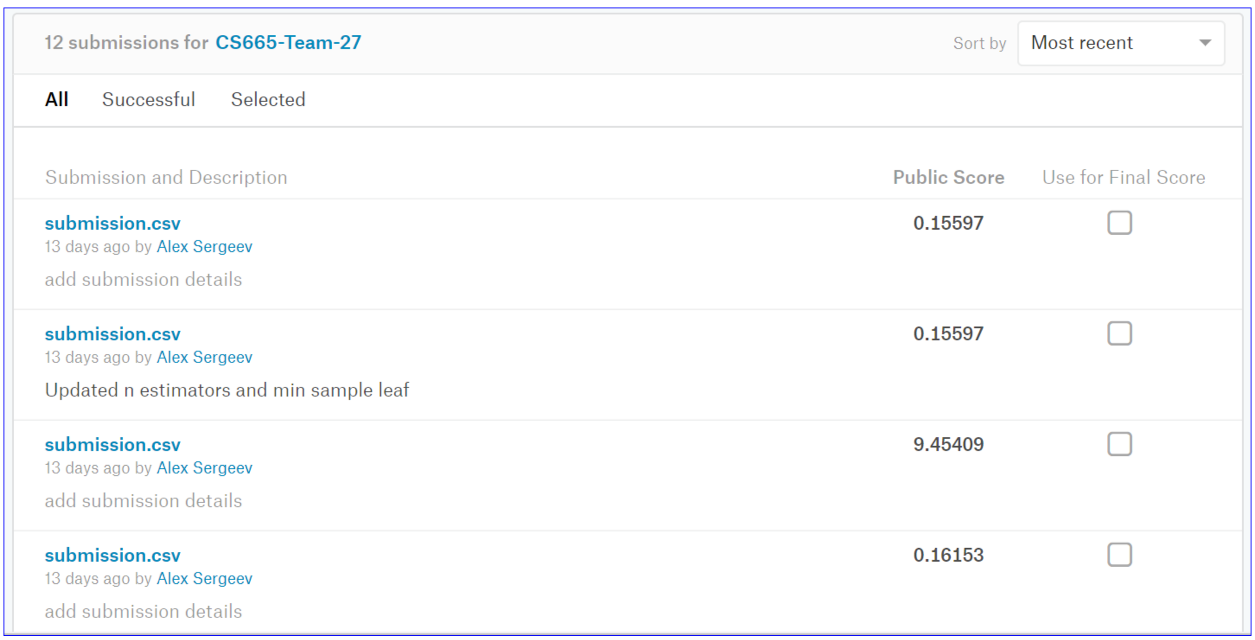
➔The entire dataset has 80 features.

F. What's the highest test score your achieved on Kaggle (the actual score value**, not your leaderboard position)?**

➔Each team members made an individual submission to the Kaggle competition. The highest score after all four submission is 0.14810 as of 05/18/2019.



Kaggle Submission History



Submission	Score
Kaggle Housing _1.zip 14 days ago by Rohitrna Contains CSV file with my predictions	0.31112
Kaggle Housing_.zip 14 days ago by Rohitrna My submission contains the CSV file with the predictions	Error
Kaggle Housing.zip 14 days ago by Rohitrna My submission contains my notebook and a CSV of my predictions	Error
mwinata rev3.csv 16 days ago by Michael Winata categorized columns, and set remainder null to mean values	0.17419
mwinata-rfr-result-rev-2.csv 16 days ago by Michael Winata split train data based on nan columns, and drop any nan values	0.17987
mwinata-rfr-result - result.csv 16 days ago by Michael Winata done by randomforestregressor - categorized, and then set nan to 0	0.14810

Submission	Score
Kaggle Housing.zip 5 days ago by Rohitrna My submission contains my notebook and a CSV of my predictions	Error
mwinata rev3.csv 7 days ago by Michael Winata categorized columns, and set remainder null to mean values	0.17419
mwinata-rfr-result-rev-2.csv 7 days ago by Michael Winata split train data based on nan columns, and drop any nan values	0.17987
mwinata-rfr-result - result.csv 7 days ago by Michael Winata done by randomforestregressor - categorized, and then set nan to 0	0.14810
submission-rlama-02.csv 7 days ago by R Lama Regression Analysis of Housing Sales Prices in Ames Iowa using Random Forest Regression Analysis.	0.15501
submission-rlama-01.csv 7 days ago by R Lama Regression Analysis of Housing prices in Ames Iowa using Random Forest Regression.	Error

mwinata-rfr-result-rev-2.csv 16 days ago by Michael Winata split train data based on nan columns, and drop any nan values	0.17987	☐
mwinata-rfr-result - result.csv 16 days ago by Michael Winata done by randomforestregressor - categorized, and then set nan to 0	0.14810	☐
submission-rlama-02.csv 16 days ago by R Lama Regression Analysis of Housing Sales Prices in Ames Iowa using Random Forest Regression Analysis.	0.15501	☐
submission-rlama-01.csv 16 days ago by R Lama Regression Analysis of Housing prices in Ames Iowa using Random Forest Regression.	Error	☐

No more submissions to show

2. Model Selection

A. What algorithms have you chosen? If you've tried multiple models, list them all. Describe your reasoning for choosing these particular models.

➔

Each team members made an individual submission to the Kaggle competition. However, since we were all trying to predict the sale price of a house, a continuous real value, we used linear regression in our analysis.
More specifically we used Random Forest Model in our analysis.

EXPLAIN(PARAMETERS)

Lists of models:

a. Random Forest Regression

   Random Forest are made of many decision trees. They are ensembles of decision trees, each of which vote on how to classify or predict a given instance of input data, and the random forest bootstraps those votes to choose the best prediction. This is done to prevent overfitting, a common flow a decision tree.

B. What are the best hyper-parameter settings you've found (e.g. the number of trees, any regularization, sampling ratio, learning rate, the size of the NN, etc.)?

➔Each team members made an individual submission to the Kaggle competition. Below is the breakdown of hyper parameter settings for each member.

NOTE: I (Ratna Lama) used random search grid validation to find the best parameter combinations. It is computationally expensive, took about **12** minutes with full core on my laptop. So I have included the code however have commented it out for the submission purposes. The result is detailed below.

Hyper-parameter settings:

```
In [33]:  rf_random.best_params_

Out[33]: {'n_estimators': 800,
          'min_samples_split': 2,
          'min_samples_leaf': 2,
          'max_features': 'sqrt',
          'max_depth': 50,
          'bootstrap': False}
```

a. **Ratna Lama**

i. n_estimators

N_estimators represent the number of trees in the forest. Usually the higher the number of trees the better to learn the data. However, adding a lot trees can slow down the training process considerably. Initially, I chose n_estimators =100 however I used grid search to find the best parameters which logged n_estimators = 800

ii. max_depth = 50

max_depth represents the depth of each tree in the forest. The deeper the tree, the more splits it has and it captures more information about the data.

iii. min_samples_split = 2

min_samples_split represents the minimum number of samples required to split an internal node. This can vary between considering at least one sample to each node to considering all of the samples at each node. When we increase this parameter, each tree in the forest become more constrained as it has to consider more samples at each node.

iv. min_samples_leaf = 2

min_samples_leaf is the minimum number of samples required to be at a leaf node. This parameter is similar to min_samples_splits, however, this describe the minimum number of samples of samples at the leaves, the base of the tree.

v. max_features = sqrt

max_features represents the number of features to consider when looking for the best split.

vi. Bootstrap = false

method of sampling data points (with or without replacement)

Fine Tunning

In [31]:

```
from sklearn.model_selection import RandomizedSearchCV
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
print(random_grid)
```

```
{'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000], 'max_features': ['auto', 'sqrt'], 'max_depth': [1
0, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None], 'min_samples_split': [2, 5, 10], 'min_samples_leaf': [1, 2, 4], 'bootstr
ap': [True, False]}
```

Random Search Traning

In [32]:

```
# Use the random grid to search for best hyperparameters
# First create the base model to tune
rf = RandomForestRegressor()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations, and use all available cores
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid, n_iter = 100, cv = 3, verbose=2, random_sta
# Fit the random search model
rf_random.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 100 candidates, totalling 300 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  33 tasks      | elapsed:  1.6min
[Parallel(n_jobs=-1)]: Done 154 tasks      | elapsed:  6.0min
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed: 11.5min finished
```

Out[32]:

```
RandomizedSearchCV(cv=3, error_score='raise-deprecating',
          estimator=RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators='warn', n_jobs=None,
           oob_score=False, random_state=None, verbose=0, warm_start=False),
          fit_params=None, iid='warn', n_iter=100, n_jobs=-1,
          param_distributions={'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 2000], 'max_features': ['a
uto', 'sqrt'], 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None], 'min_samples_split': [2, 5, 10], 'min_samp
les_leaf': [1, 2, 4], 'bootstrap': [True, False]},
          pre_dispatch='2*n_jobs', random_state=42, refit=True,
          return_train_score='warn', scoring=None, verbose=2)
```

return_train_score='warn', scoring=None, verbose=2)

In [33]:

```
rf_random.best_params_
```

Out[33]:

```
{'n_estimators': 800,
 'min_samples_split': 2,
 'min_samples_leaf': 2,
 'max_features': 'sqrt',
 'max_depth': 50,
 'bootstrap': False}
```

Evaluate Random Search

```
In [34]:  # Determine if random search yielded a better model, we compare base model with the best random search model.

          def evaluate(model, X_test, y_test):
              predictions = model.predict(X_test)
              errors = abs(predictions - y_test)
              mape = 100 * np.mean(errors / y_test)
              accuracy = 100 - mape
              print('Model Performance')
              print('Average Error: {:0.4f} degrees.'.format(np.mean(errors)))
              print('Accuracy = {:0.2f}%.'.format(accuracy))

              return accuracy
```

```
In [35]:  # Base Model Performance
          base_model = RandomForestRegressor(n_estimators = 10, random_state = 42)
          base_model.fit(X_train, y_train)
          base_accuracy = evaluate(base_model, X_test, y_test)

          Model Performance
          Average Error: 18511.3616 degrees.
          Accuracy = 89.02%.
```

```
In [36]:  # Best Random Model Performance
          best_random = rf_random.best_estimator_
          random_accuracy = evaluate(best_random, X_test, y_test)

          Model Performance
          Average Error: 16050.1202 degrees.
          Accuracy = 90.02%.
```


```
In [36]:  # Best Random Model Performance
          best_random = rf_random.best_estimator_
          random_accuracy = evaluate(best_random, X_test, y_test)

          Model Performance
          Average Error: 16050.1202 degrees.
          Accuracy = 90.02%.
```


Random Forest Regressor

```
In [19]:  rf = RandomForestRegressor(n_estimators=100, random_state=17)

          %time rf.fit(X_train, y_train)

          Wall time: 2.46 s

Out[19]:  RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                     max_features='auto', max_leaf_nodes=None,
                     min_impurity_decrease=0.0, min_impurity_split=None,
                     min_samples_leaf=1, min_samples_split=2,
                     min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=None,
                     oob_score=False, random_state=17, verbose=0, warm_start=False)
```

3. Training & Validation

A. What is your training and test split approach?

➔

a. Ratna Lama

i. Train data set: 75%

ii. Test data set: 25%

Split the Data into Training and Testing Unit

```
In [16]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)

In [17]: # train_test_split

In [18]: X.shape, X_train.shape, X_test.shape, y_train.shape, y_test.shape
Out[18]: ((1460, 79), (1095, 79), (365, 79), (1095,), (365,))
```

B. What methods did you use to evaluate your performance on your datasets? Provide the exact formula and the implementation, or refer to an existing API.

a) What is the best score you've achieved?

Random Forest Regressor

```
In [19]: rf = RandomForestRegressor(n_estimators=100, random_state=17)

         %time rf.fit(X_train, y_train)

         Wall time: 2.46 s

Out[19]: RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                    max_features='auto', max_leaf_nodes=None,
                    min_impurity_decrease=0.0, min_impurity_split=None,
                    min_samples_leaf=1, min_samples_split=2,
                    min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=None,
                    oob_score=False, random_state=17, verbose=0, warm_start=False)

In [20]: rf.score(X_train, y_train)

Out[20]: 0.9787190039874757

In [21]: rf.score(X_test, y_test)

Out[21]: 0.896331331748598
```

MSE, RMSE, Score calculations

```
In [71]: mse = ((y_hat - y_test) ** 2).mean()

In [72]: rmse = np.sqrt(mse)

In [73]: v = ((y_test - y_test.mean()) ** 2).mean()

In [74]: mse, rmse, v

Out[74]: (726231060.1455075, 26948.67455266599, 7005309052.339321)

In [75]: score = (1 - (mse/v))
         score

Out[75]: 0.8963313317485981
```

4. Your Code

A. Include your entire training and testing pipeline. **It should run without any errors**.

➔ Please see attached file: **finalProject_v03.ipynb**

B. Links to the data to be downloaded

   a. https://www.kaggle.com/c/house-prices-advanced-regression-techniques/data

C. Assume your dataset is stored in the **data** subfolder (relative to the notebook). File name should match the downloaded files.

➔

read CSV

```
In [49]: df = pd.read_csv("train.csv")
```

```
In [50]: df.shape
Out[50]: (1460, 81)
```

```
In [51]: df.head()
Out[51]:
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	PoolArea	PoolQC	Fence	MiscFeature	MiscVal
0	1	60	RL	65.0	8450	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0
1	2	20	RL	80.0	9600	Pave	NaN	Reg	Lvl	AllPub	...	0	NaN	NaN	NaN	0
2	3	60	RL	68.0	11250	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0
3	4	70	RL	60.0	9550	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0
4	5	60	RL	84.0	14260	Pave	NaN	IR1	Lvl	AllPub	...	0	NaN	NaN	NaN	0

5 rows × 81 columns

Test DATA Set

```
In [30]: test_dataset = pd.read_csv('test.csv')
         test_dataset.head()
Out[30]:
```

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContour	Utilities	...	ScreenPorch	PoolArea	PoolQC	Fence	Misc
0	1461	20	RH	80.0	11622	Pave	NaN	Reg	Lvl	AllPub	...	120	0	NaN	MnPrv	
1	1462	20	RL	81.0	14267	Pave	NaN	IR1	Lvl	AllPub	...	0	0	NaN	NaN	
2	1463	60	RL	74.0	13830	Pave	NaN	IR1	Lvl	AllPub	...	0	0	NaN	MnPrv	
3	1464	60	RL	78.0	9978	Pave	NaN	IR1	Lvl	AllPub	...	0	0	NaN	NaN	
4	1465	120	RL	43.0	5005	Pave	NaN	IR1	HLS	AllPub	...	144	0	NaN	NaN	

5 rows × 80 columns

D. Print your **(train score, test score)** as the last output of your notebook. This score formula should match what you've described in #3.2.

➔

a. Score using Python Sci-kit library

Random Forest Regressor

```
In [19]:  rf = RandomForestRegressor(n_estimators=100, random_state=17)

           %time rf.fit(X_train, y_train)

           Wall time: 2.46 s

Out[19]:   RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                      max_features='auto', max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=None,
                      oob_score=False, random_state=17, verbose=0, warm_start=False)

In [20]:  rf.score(X_train, y_train)

Out[20]:  0.9787190039874757

In [21]:  rf.score(X_test, y_test)

Out[21]:  0.896331331748598
```

b. Score calculation

MSE, RMSE, Score calculations

```
In [71]:  mse = ((y_hat - y_test) ** 2).mean()

In [72]:  rmse = np.sqrt(mse)

In [73]:  v = ((y_test - y_test.mean()) ** 2).mean()

In [74]:  mse, rmse, v

Out[74]:  (726231060.1455075, 26948.67455266599, 7005309052.339321)

In [75]:  score = (1 - (mse/v))
          score

Out[75]:  0.8963313317485981
```

5. Acknowledgements

We thank Dean De Cock for compiling the Ames Housing dataset and making it available for the public general to use in data science education.