Springboard

DATA SCIENCE CAPSTONE PROJECT 1

CAPSTONE PROJECT: IN-DEPTH ANALYSIS (Machine Learning)

-Mehreteab Kidane

INTRODUCTION AND PROBLEM STATEMENT

The problem I am trying to solve in this project is the house price prediction problem. The relationship between house prices and the economy is an important motivating factor for predicting house prices. Based on certain features of the house, such as the area in square feet, the condition of the house, number of bedrooms, number of bathrooms, number of floors, year of built; we have to predict the estimated price of the house.

EVALUATING THE ALGORITHMS

There are different machine learning algorithms to predict the house prices. This project will use linear regression to predict house prices in King County, USA. The motivation for choosing linear regression is it can accurately predict the trends when the underlying process is linear. There are many factors affect house prices, such as numbers of bedrooms and bathrooms. In addition, choosing different combinations of parameters in Linear Regression will also affect the predictions greatly.

```
Load the dataset
Kaggle- House Sales in King County, USA

dataset = read.csv('../input/kc_house_data.csv')
```

I used two regression algorithms to train machine learning models. The models used are Regression and Random Forest Regression.

SPLITTING DATA INTO TRAINING AND TESTING SET

The training dataset and test dataset must be similar, usually have the same predictors or variables. They differ on the observations and specific values in the variables. If you fit the model on the training dataset, then you implicitly minimize error or find correct responses. The fitted model provides a good prediction on the training dataset. Then you test the model on the test dataset. If the model predicts good also on the test dataset, you have more confidence. You have more confidence since the test dataset is similar to the training dataset, but not the same nor seen by the model. It means the model transfers prediction or learning in real sense.

So, by splitting dataset into training and testing subset, I can efficiently measure my trained model since it never sees testing data before. Thus it's possible to prevent overfitting.

I am just splitting dataset into 20% of test data and remaining 80% will used for training the model.

```
X_train, X_test, y_train, y_test = cross_validation.train_test_split(X, y
,test_size = 0.2)
```

FEATURE IMPORTANCE

After splitting let's identify the most important fractures in our dataset that contribute most to my prediction variable. Irrelevant features would decrease accuracy especially for linear algorithms like linear regression.

- Reduce Overfitting : Less redundant data means less opportunity to make decisions based on noise
- Improve accuracy : Less misleading data means modeling accuracy improves
- Reduce Training Time : Less data means that algorithms train faster

APPLYING MACHINE LEARNING MODEL

I. Linear Regression

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression ()
regressor.fit (train_features, train_labels)
```

I trained my model on the training set using the "fit()" method of the LinearRegression class from the sklearn.linear_model module.
Next, I need to make predictions on the test set. To do so, execute the following script:

```
predicted_price = regressor.predict(test_features)
```

Now my model has been trained, the next step is to evaluate the performance of the model. The metrics used for the evaluation of linear regression model are root-mean-square error **(*RMSE*)**, mean squared error **(*MSE*)**, and mean absolute error **(*MAE*)**.

The following script finds the value for these metrics for the linear regression algorithm:

```
from sklearn import metrics
```

print('Mean sAbsolute Error:', metrics.mean_absolute_error(test_labels, predicted_price))

print('Mean Squared Error:', metrics.mean_squared_error(test_labels,predicted_price))

print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(test_labels, predicted_price)))

The output looks like this:

```
Mean Absolute Error: 126522.39741578815
Mean Squared Error: 39089803087.07735
Root Mean Squared Error: 197711.41364897817
```

The lesser the values for these metrics, the better is the performance of the algorithms. Let's see how random forest regressor performs on this data and compare the two algorithms.

II. Random Forest Regressor

Different kinds of models have different advantages. The random forest model is very good at handling tabular data with numerical features, or categorical features with fewer than hundreds of categories. Unlike linear models, random forests are able to capture non-linear interaction between the features and the target.
Execute the following script to train the machine learning model using Random Forest Regressor.

```
from sklearn.ensemble import RandomForestRegressor
regressor = RandomForestRegressor(n_estimators=200, random_state=0)
regressor.fit(train_features, train_labels)
```

The next step is to predict the output values:

```
predicted_price = regressor.predict(test_features)
```

And finally, the script below finds the performance metrics values for the random forest algorithm:

```
from sklearn import metrics
```

print('Mean Absolute Error:', metrics.mean_absolute_error(test_labels, predicted_price))
print('Mean Squared Error:', metrics.mean_squared_error(test_labels,predicted_price))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(test_labels, predicted_price)))

The output of the script above looks like this:

```
Mean Absolute Error: 70467.01934555387
Mean Squared Error: 16251802750.754648
Root Mean Squared Error: 127482.55861393215
```

Conclusion

From the output, it is visible that the random forest algorithm is better at predicting house prices for the Kings County housing dataset, since the values of ***MAE***, ***RMSE***, and ***MSE*** for random forest algorithm are far less compared to the linear regression algorithm.