Introduction to data analysis

Housing price predictions using SciKit-Learn

Introduction

This Project is about using Machine Learning techniques to develop a model which can make predictions of the price of the house given the respective features. This process of making predictions usually comes under the **Regression** : which means to predict a continuous number for a given input features. In this project we have used ML models such as Linear Regression, Lasso Regression, Decision Tree and Random forest methods.

Data Description, Visualization and Cleaning:

Before feeding our data to the Machine Learning Model we would like to see the content of our data. So, we see below that our data has 1460 instances and 81 Features.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
```

Now, let's try to see which Features/ Columns have null/nan values. From the figure below we see that 19 Features having none/nan values.

```
# finding features containing null values
i = df.columns[df.isnull().any()].tolist()
i
```

```
['LotFrontage',
 'Alley',
 'MasVnrType',
 'MasVnrArea',
 'BsmtQual',
 'BsmtCond',
 'BsmtExposure',
 'BsmtFinType1',
 'BsmtFinType2',
 'Electrical',
 'FireplaceQu',
 'GarageType',
 'GarageYrBlt',
 'GarageFinish',
 'GarageQual',
 'GarageCond',
 'PoolQC',
 'Fence',
 'MiscFeature']
```

Dropping Features: now we drop those features which have more than 50% of the values as None/Nan.

```
In [5]: #dropping features like MiscFeature, Fence, PoolQC, FireplaceQu and Alley bc more than 50% of the instances are
        # are missing from these features
        df.drop(columns=["MiscFeature","PoolQC","Alley", "Fence","FireplaceQu" ],axis=1, inplace=True)
```

Now, we'll try to fill the nan/none values with the mean of the column value.

```
In [9]: # LotFrontage
        mean_lf = df["LotFrontage"].mean()
        df["LotFrontage"] = df["LotFrontage"].fillna(mean_lf)
```

And, then we'll drop all those instance/ rows which have null/nans. And we now have a clean data with no nan values but the price we have to pay was loosing around 120 instances, but that's fine since we are just using trial and error and and we'll see if that affects our predictions. For now let's just move on bc life's always easy with less stuff to deal with "less is more"!

```
df.dropna(inplace=True)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1338 entries, 0 to 1459
Data columns (total 76 columns):
```

<u>Checking the Skewness of the data:</u>

```
In [15]: skewed_features = df.skew().sort_values(ascending=False)

In [16]: skewness_df = pd.DataFrame({'Skewed Features' :skewed_features})

In [92]: skewness_df[:10]
Out[92]:
```

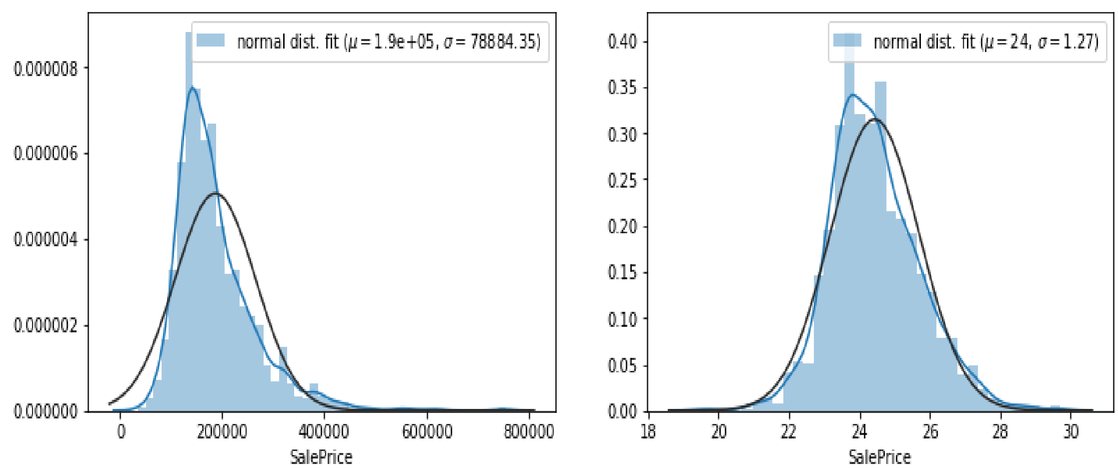
	Skewed Features
MiscVal	24.632578
PoolArea	14.187832
LotArea	11.938124
LowQualFinSF	10.566815
3SsnPorch	10.096553
KitchenAbvGr	5.943561
BsmtFinSF2	4.146519
ScreenPorch	3.916848
BsmtHalfBath	3.847909
EnclosedPorch	3.205286

So, we do see that our features are skewed to the right. So we used the boxcox1p module from the scipy package to reduce the skewness of the data.

```
In [19]: #selecting a thrshold for skewness say 1.0 and only working on apply box cox to those features

In [20]: from scipy import stats

In [21]: skewness_df = skewness_df[abs(skewness_df) > 0.6]

In [22]: skewed_feature_cut = skewness_df.index

In [23]: from scipy.special import boxcox1p
```

```
y = ((1+x)**lmbda - 1) / lmbda  if lmbda != 0
    log(1+x)                    if lmbda == 0
    boxcox(x,lam) returns y
```

```
In [24]: lam = 0.1
         for feat in skewed_feature_cut:
             #print(feat)
             df[feat] = boxcox1p(df[feat],lam)
             df[feat] += 1
```



In the above figure, left is before skewness removed and right is after skewness is removed from the SalePrice, Similar plots can be done for the other features as well.

<u>Encoding the Categorical Features:</u> Since a Machine Learning algorithm understands only numerics, we have convert out categorical features to ML algo understandable form so, we use one-hot encoding.

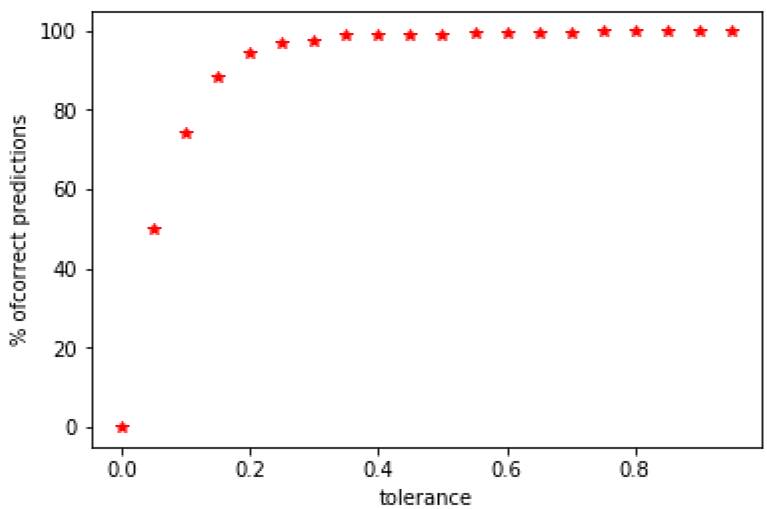
Linear Regression:

```
from sklearn.model_selection import train_test_split
```

```
y = df["SalePrice"]
df2 = df.drop(["SalePrice"],axis=1)
X_train, X_test, y_train, y_test = train_test_split(df2, y, test_size=0.2, shuffle=True, random_state=1)
```

```
from sklearn import linear_model
lm = linear_model.LinearRegression() #regularized linear model lasso because after one hot encode i was underfitting

model = lm.fit(X_train, y_train)
predictions = lm.predict(X_test)
```

Scoring Metric Plot:



r2Score= 0.8579093338230042

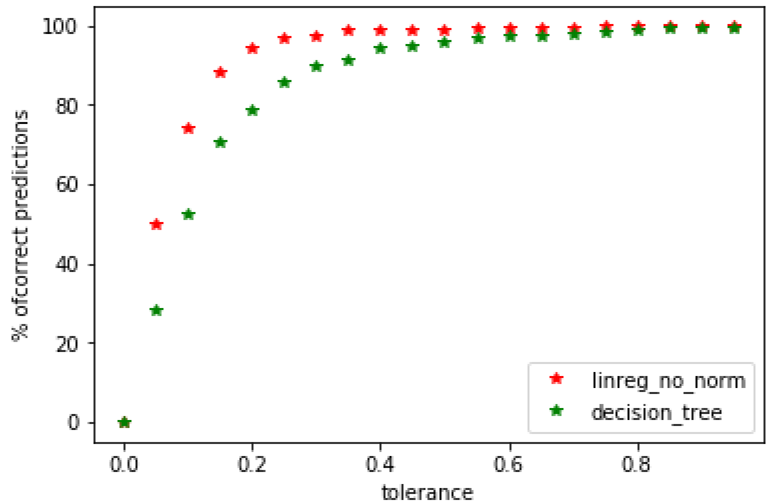
Decision Tree:

```
In [74]: from sklearn.tree import DecisionTreeRegressor
```

```
In [75]: y = df["SalePrice"]
         df2 = df.drop(["SalePrice"],axis=1)
         X_train, X_test, y_train, y_test = train_test_split(df2, y, test_size=0.2, shuffle=True, random_state=1)
```

```
y = scaled_features_df["SalePrice"]
df2 = scaled_features_df.drop(["SalePrice"],axis=1)
X_train, X_test, y_train, y_test = train_test_split(df2, y, test_size=0.2, shuffle=True, random_state=1)
```

Scoring Metric Plot:



r2 Score= 0.6790908937681206

Random Forest:

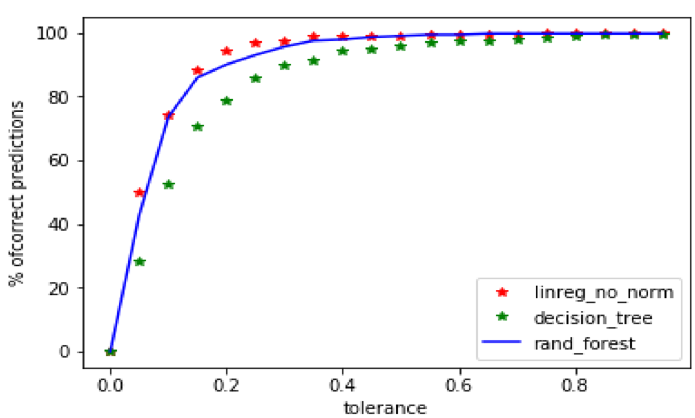
```
from sklearn.ensemble import RandomForestRegressor
```

```
rfgr = RandomForestRegressor(max_depth=10,random_state=0,n_estimators=150)
```

```
rfgr.fit(X_train,y_train)
```

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=10,
           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=150, n_jobs=None,
           oob_score=False, random_state=0, verbose=0, warm_start=False)
```

Scoring Metric Plot:



r2 Score = 0.8448056983954947

Conclusion:

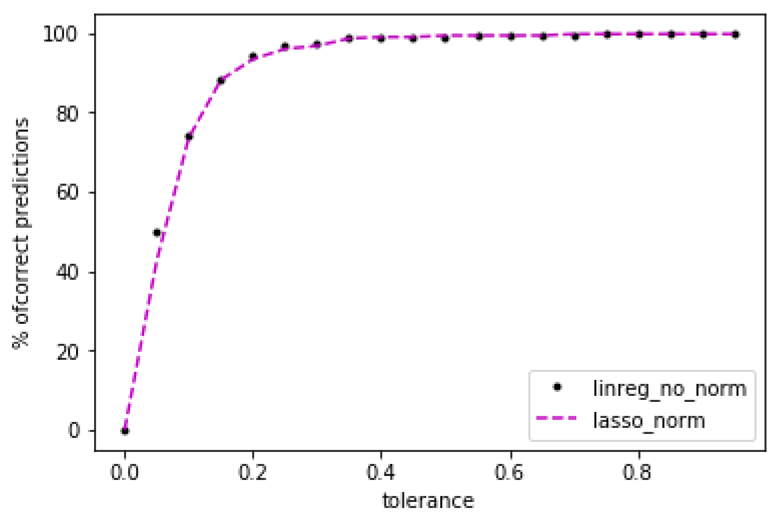
Looks like the Linear Regression model is performing really good prediction with a mean difference between the test and prediction of $ -673.544086965215.

But, I read in an article https://nycdatascience.com/blog/student-works/improving-model-accuracy-kaggle-competition/

And ran the test using GridSearchCV and the training score was very less compared to the test score which according to the article meant that probably my model is overfitting.

So, I tried using Lasso(l1 norm regression technique) but then my r2Score improved a liitle but the overall performance was still the same.

This might need a more inspection and I plan to do this in the coming days as and when i get enough time. Below is the plot for Lasso regression



R2 score Lass0 0.8645042477801722

Model	Lin_reg	Lasso	Deci Tree	Ran For
r2_score	0.857909333	0.86450	0.67909	0.84480

References and Sources:

1.SciKit-Learn https://scikit-learn.org/stable/about.html#citing-scikit-learn

2.Pandas, Matplotlib, seaborn, scipy, python

3.https://nycdatascience.com/blog/student-works/improving-model-accuracy-kaggle-competition/

4.https://www.kaggle.com/erick5/predicting-house-prices-with-machine-learning/comments

5.https://www.kaggle.com/juliencs/a-study-on-regression-applied-to-the-ames-dataset