**Nguyen Phu huy**

**18rd June,2020**

# **Capstone Project**

# House Price Prediction

# 1. Introduction

Competition Description:  
Ask a home buyer to describe their dream house, and they probably won't begin with the height of the basement ceiling or the proximity to an east-west railroad. But this playground competition's dataset proves that much more influences price negotiations than the number of bedrooms or a white-picket fence. With 79 explanatory variables describing (almost) every aspect of residential homes in Ames, Iowa, this competition challenges you to predict the final price of each home.

Acknowledgments:  
The Ames Housing dataset was compiled by Dean De Cock for use in data science education. It's an incredible alternative for data scientists looking for a modernized and expanded version of the often cited Boston Housing dataset.

# 2. Data

Kaggle provide the script to pull data from given path.

In [18]:

*#Load data*

**import** **numpy** **as** **np** *# linear algebra*

**import** **pandas** **as** **pd** *# data processing, CSV file I/O (e.g. pd.read\_csv)*

**import** **os**

**for** dirname, \_, filenames **in** os.walk('/kaggle/input'):

**for** filename **in** filenames:

print(os.path.join(dirname, filename))

*#get training data and testing data*

train\_data = pd.read\_csv('https://raw.githubusercontent.com/huynguyenphu/documents/master/train.csv')

test\_data = pd.read\_csv('https://raw.githubusercontent.com/huynguyenphu/documents/master/test.csv')

## 2.1 Load Data

In [19]:

*#check the numbers of samples and features*

print("The train data size before dropping Id feature is : **{}** ".format(train\_data.shape))

print("The test data size before dropping Id feature is : **{}** ".format(test\_data.shape))

*#Save the 'Id' column*

train\_ID = train\_data['Id']

test\_ID = test\_data['Id']

*#Now drop the 'Id' colum since it's unnecessary for the prediction process.*

train\_data.drop("Id", axis = 1, inplace = **True**)

test\_data.drop("Id", axis = 1, inplace = **True**)

*#check again the data size after dropping the 'Id' variable*

print("**\n**The train data size after dropping Id feature is : **{}** ".format(train\_data.shape))

print("The test data size after dropping Id feature is : **{}** ".format(test\_data.shape))

The train data size before dropping Id feature is : (1460, 81)

The test data size before dropping Id feature is : (1459, 80)

The train data size after dropping Id feature is : (1460, 80)

The test data size after dropping Id feature is : (1459, 79)

## 2.2 Data Cleaning

### Outliers

In [37]:

*#handling outlier in the training data*

*#We do a pair scatter plot between the reponds and its highly correlated predictors and find possible outliers.*

**import** **matplotlib.pyplot** **as** **plt**

**import** **seaborn** **as** **sns**

*#Setting style to 'darkgrid'*

sns.set\_style('darkgrid')

corrmat = train\_data.corr()

top\_corr\_features = corrmat.index[abs(corrmat["SalePrice"])>0.6]

sns.pairplot(train\_data[top\_corr\_features], diag\_kind='kde')

*#Based on the pairplot, we remove the outlier that "GrLivArea" is larger than 4000*

train\_data = train\_data.drop(train\_data[(train\_data['GrLivArea']>4000)].index)

### Missing Values

In [21]:

*#processing the train and test data simulatously.*

ntrain = train\_data.shape[0]

ntest = test\_data.shape[0]

y\_train = train\_data.SalePrice.values

all\_data = pd.concat((train\_data, test\_data),sort=**False**).reset\_index(drop=**True**)

all\_data.drop(['SalePrice'], axis=1, inplace=**True**)

print("all\_data size is : **{}**".format(all\_data.shape))

all\_data size is : (2915, 79)

In [22]:

*#handleing missing value*

*# miss\_number=all\_data.isnull().sum()*

*# miss\_ratio=all\_data.isnull().sum()/len(all\_data)*

*# miss\_info=pd.DataFrame({'Number of miss':miss\_number,'Proportion of miss':miss\_ratio},)*

*# miss\_info=miss\_info.loc[miss\_info['Number of miss']>0]*

*# miss\_info=miss\_info.sort\_values(by='Number of miss',ascending=0)*

*# print(miss\_info)*

*#fill missing values*

**import** **copy**

all\_data2=copy.copy(all\_data)

*#By description, the following missing data are replaced by "None"*

**for** col **in** ('PoolQC', 'MiscFeature', 'Alley', 'Fence','FireplaceQu','GarageType',

'GarageFinish', 'GarageQual', 'GarageCond','BsmtQual', 'BsmtCond',

'BsmtExposure', 'BsmtFinType1', 'BsmtFinType2',"MasVnrType"):

all\_data2[col] = all\_data2[col].fillna('None')

*#By descriotion, the following missing data are replaced by number 0*

**for** col **in** ('GarageYrBlt', 'GarageArea', 'GarageCars','BsmtFinSF1', 'BsmtFinSF2',

'BsmtUnfSF','TotalBsmtSF', 'BsmtFullBath', 'BsmtHalfBath',"MasVnrArea"):

all\_data2[col] = all\_data2[col].fillna(0)

*#For "LotFrontage", we fill in missing values by the median LotFrontage of the neighborhood.*

all\_data2["LotFrontage"] = all\_data2.groupby("Neighborhood")["LotFrontage"].transform(**lambda** x: x.fillna(x.median()))

*#there is only one missing value for the following variable, just replace it by the mode.*

all\_data2['Electrical'] = all\_data2['Electrical'].fillna(all\_data2['Electrical'].mode()[0])

*# miss\_number=all\_data2.isnull().sum()*

*# miss\_ratio=all\_data2.isnull().sum()/len(all\_data2)*

*# miss\_info=pd.DataFrame({'Number of miss':miss\_number,'Proportion of miss':miss\_ratio},)*

*# miss\_info=miss\_info.loc[miss\_info['Number of miss']>0]*

*# miss\_info=miss\_info.sort\_values(by='Number of miss',ascending=0)*

*# miss\_info*

### Transforming some Numerical Variables that are Really Categorical

In [23]:

*#Transforming some numerical variables that are really categorical*

*#MSSubClass=The building class*

all\_data2['MSSubClass'] = all\_data2['MSSubClass'].astype(str)

*#Changing OverallCond into a categorical variable*

all\_data2['OverallCond'] = all\_data2['OverallCond'].astype(str)

*#Year and month sold are transformed into categorical features.*

all\_data2['YrSold'] = all\_data2['YrSold'].astype(str)

all\_data2['MoSold'] = all\_data2['MoSold'].astype(str)

### Label Encoding the Categorical Variables

In [24]:

**from** **sklearn.preprocessing** **import** LabelEncoder

cols = ('FireplaceQu', 'BsmtQual', 'BsmtCond', 'GarageQual', 'GarageCond',

'ExterQual', 'ExterCond','HeatingQC', 'PoolQC', 'KitchenQual', 'BsmtFinType1',

'BsmtFinType2', 'Functional', 'Fence', 'BsmtExposure', 'GarageFinish', 'LandSlope',

'LotShape', 'PavedDrive', 'Street', 'Alley', 'CentralAir', 'MSSubClass', 'OverallCond',

'YrSold', 'MoSold')

*# process columns, apply LabelEncoder to categorical features*

**for** c **in** cols:

lbl = LabelEncoder()

lbl.fit(list(all\_data2[c].values))

all\_data2[c] = lbl.transform(list(all\_data2[c].values))

*# shape*

print('Shape all\_data: **{}**'.format(all\_data2.shape))

Shape all\_data: (2915, 79)

### Feature engineering

In [25]:

*# Deep feature engineer*

all\_data2['YrBltAndRemod']=all\_data2['YearBuilt']+all\_data2['YearRemodAdd']

all\_data2['TotalSF']=all\_data2['TotalBsmtSF'] + all\_data2['1stFlrSF'] + all\_data2['2ndFlrSF']

all\_data2['Total\_sqr\_footage'] = (all\_data2['BsmtFinSF1'] + all\_data2['BsmtFinSF2'] +

all\_data2['1stFlrSF'] + all\_data2['2ndFlrSF'])

all\_data2['Total\_Bathrooms'] = (all\_data2['FullBath'] + (0.5 \* all\_data2['HalfBath']) +

all\_data2['BsmtFullBath'] + (0.5 \* all\_data2['BsmtHalfBath']))

all\_data2['Total\_porch\_sf'] = (all\_data2['OpenPorchSF'] + all\_data2['3SsnPorch'] +

all\_data2['EnclosedPorch'] + all\_data2['ScreenPorch'] +

all\_data2['WoodDeckSF'])

*# simplified features*

all\_data2['haspool'] = all\_data2['PoolArea'].apply(**lambda** x: 1 **if** x > 0 **else** 0)

all\_data2['has2ndfloor'] = all\_data2['2ndFlrSF'].apply(**lambda** x: 1 **if** x > 0 **else** 0)

all\_data2['hasgarage'] = all\_data2['GarageArea'].apply(**lambda** x: 1 **if** x > 0 **else** 0)

all\_data2['hasbsmt'] = all\_data2['TotalBsmtSF'].apply(**lambda** x: 1 **if** x > 0 **else** 0)

all\_data2['hasfireplace'] = all\_data2['Fireplaces'].apply(**lambda** x: 1 **if** x > 0 **else** 0)

### Skewed features

In [26]:

*#skew data*

**from** **scipy.stats** **import** skew

numeric\_feats = all\_data2.dtypes[all\_data2.dtypes != "object"].index

*# Check the skew of all numerical features*

skewed\_feats = all\_data2[numeric\_feats].apply(**lambda** x: skew(x.dropna())).sort\_values(ascending=**False**)

print("**\n**Skew in numerical features: **\n**")

skewness = pd.DataFrame({'Skew' :skewed\_feats})

print(skewness.head(10))

skewness = skewness[abs(skewness) > 0.5]

print("There are **{}** skewed numerical features to Box Cox transform".format(skewness.shape[0]))

**from** **scipy.special** **import** boxcox1p

skewed\_features = skewness.index

lam = 0.15

**for** feat **in** skewed\_features:

all\_data2[feat] = boxcox1p(all\_data2[feat], lam)

Skew in numerical features:

Skew

MiscVal 21.932147

PoolArea 18.701829

haspool 16.186531

LotArea 13.123758

LowQualFinSF 12.080315

3SsnPorch 11.368094

LandSlope 4.971350

KitchenAbvGr 4.298845

BsmtFinSF2 4.142863

EnclosedPorch 4.000796

There are 68 skewed numerical features to Box Cox transform

### Get dummies for Catigory Variables.

In [27]:

*#Get dummies for catigory variables.*

all\_data3=pd.get\_dummies(all\_data2)*#train2:after missing value, outlier; train3:get dummies for category variable.*

### Check the Normality of the Respond Variable (SalePrice)

In [28]:

*#check the normality of the responds variable*

**import** **seaborn** **as** **sns**

**from** **scipy.stats** **import** norm *#for some statistics*

**import** **matplotlib.pyplot** **as** **plt** *# Matlab-style plotting*

*#histogram plot*

sns.distplot(y\_train, fit=norm);

*#add title axis*

(mu, sigma) = norm.fit(y\_train)

plt.legend(['Normal dist. ($\mu=$ **{:.2f}** and $\sigma=$ **{:.2f}** )'.format(mu, sigma)],loc='best')

plt.ylabel('Frequency')

plt.title('SalePrice distribution')

*#use QQ-plot to see the normality*

**from** **scipy** **import** stats

fig = plt.figure()

res = stats.probplot(y\_train, plot=plt)

plt.show()

In [29]:

*#the respond variable is right skew, we use log transformation to make it more normally.*

y\_train\_log = np.log1p(y\_train)*#use np.log1p which applies log(1+x) when the data is close or equal to zero*

*#Check the new distribution*

sns.distplot(y\_train\_log , fit=norm);

*#add title axis*

(mu, sigma) = norm.fit(y\_train\_log)

plt.legend(['Normal dist. ($\mu=$ **{:.2f}** and $\sigma=$ **{:.2f}** )'.format(mu, sigma)],loc='best')

plt.ylabel('Frequency')

plt.title('log-SalePrice distribution')

*#Get also the QQ-plot*

fig = plt.figure()

res = stats.probplot(y\_train\_log, plot=plt)

plt.show()

The skew seems now corrected and the data appears more normally distributed.

# 3. Models

In [30]:

*#separate the training and testing data.*

x\_train = all\_data3[:ntrain]

y\_train\_log= np.log1p(y\_train)

x\_test = all\_data3[ntrain:]

In [31]:

!pip install lightgbm

Requirement already satisfied: lightgbm in /opt/conda/envs/Python36/lib/python3.6/site-packages (2.3.1)

Requirement already satisfied: scipy in /opt/conda/envs/Python36/lib/python3.6/site-packages (from lightgbm) (1.2.0)

Requirement already satisfied: numpy in /opt/conda/envs/Python36/lib/python3.6/site-packages (from lightgbm) (1.15.4)

Requirement already satisfied: scikit-learn in /opt/conda/envs/Python36/lib/python3.6/site-packages (from lightgbm) (0.20.3)

In [32]:

*#load packages*

**from** **sklearn.linear\_model** **import** Lasso, ElasticNet

**from** **sklearn.ensemble** **import** RandomForestRegressor, GradientBoostingRegressor

**from** **sklearn.kernel\_ridge** **import** KernelRidge

**from** **sklearn.pipeline** **import** make\_pipeline

**from** **sklearn.preprocessing** **import** RobustScaler

**from** **sklearn.base** **import** BaseEstimator, TransformerMixin, RegressorMixin, clone

**from** **sklearn.model\_selection** **import** KFold, cross\_val\_score, train\_test\_split

**from** **sklearn.metrics** **import** mean\_squared\_error

**import** **xgboost** **as** **xgb**

**import** **lightgbm** **as** **lgb**

## 3.1 Linear regression

In [33]:

*#lasso*

model\_lasso = make\_pipeline(RobustScaler(), Lasso(alpha =0.0005, random\_state=1))

*#Elastic Net Regression*

model\_ENet = make\_pipeline(RobustScaler(), ElasticNet(alpha=0.0005, l1\_ratio=.9, random\_state=3))

## 3.2 Xgboost with Hyper-parameter Tuning

In [34]:

*#xgboost with parameter tuning*

**from** **sklearn.model\_selection** **import** RandomizedSearchCV, GridSearchCV

**import** **scipy.stats** **as** **st**

**import** **datetime**

print('####################################################**\n{}**\start\_time'

.format(datetime.datetime.now().strftime('%H:%M')))

params = {

'colsample\_bytree': [0.4],

'gamma': st.uniform(0.0,0.05),

'learning\_rate': [0.05],

'max\_depth':[3],

'min\_child\_weight': [2],

'n\_estimators': st.randint(2000,3000),

'subsample': st.uniform(0.4,0.6),

'objective':['reg:squarederror'],

'reg\_alpha':st.uniform(0,0.5),

}

xgb\_temp = xgb.XGBRegressor()

model\_xgb\_tuned = RandomizedSearchCV(xgb\_temp, params, n\_iter=3,n\_jobs=-1)

model\_xgb\_tuned.fit(x\_train,y\_train\_log)

model\_xgb = xgb.XGBRegressor(\*\*model\_xgb\_tuned.best\_params\_)

print(model\_xgb)

print('**{}\t**End\_time**\n**####################################################'

.format(datetime.datetime.now().strftime('%H:%M')))

####################################################

07:56\start\_time

---------------------------------------------------------------------------

\_RemoteTraceback Traceback (most recent call last)

\_RemoteTraceback:

"""

Traceback (most recent call last):

File "/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/externals/joblib/externals/loky/process\_executor.py", line 418, in \_process\_worker

r = call\_item()

File "/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/externals/joblib/externals/loky/process\_executor.py", line 272, in \_\_call\_\_

return self.fn(\*self.args, \*\*self.kwargs)

File "/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/externals/joblib/\_parallel\_backends.py", line 567, in \_\_call\_\_

return self.func(\*args, \*\*kwargs)

File "/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/externals/joblib/parallel.py", line 225, in \_\_call\_\_

for func, args, kwargs in self.items]

File "/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/externals/joblib/parallel.py", line 225, in <listcomp>

for func, args, kwargs in self.items]

File "/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/model\_selection/\_validation.py", line 528, in \_fit\_and\_score

estimator.fit(X\_train, y\_train, \*\*fit\_params)

File "/opt/conda/envs/Python36/lib/python3.6/site-packages/xgboost/sklearn.py", line 328, in fit

verbose\_eval=verbose, xgb\_model=xgb\_model)

File "/opt/conda/envs/Python36/lib/python3.6/site-packages/xgboost/training.py", line 210, in train

xgb\_model=xgb\_model, callbacks=callbacks)

File "/opt/conda/envs/Python36/lib/python3.6/site-packages/xgboost/training.py", line 74, in \_train\_internal

bst.update(dtrain, i, obj)

File "/opt/conda/envs/Python36/lib/python3.6/site-packages/xgboost/core.py", line 1021, in update

dtrain.handle))

File "/opt/conda/envs/Python36/lib/python3.6/site-packages/xgboost/core.py", line 151, in \_check\_call

raise XGBoostError(\_LIB.XGBGetLastError())

xgboost.core.XGBoostError: b'[07:56:07] src/objective/objective.cc:23: Unknown objective function reg:squarederror\n\nStack trace returned 10 entries:\n[bt] (0) /opt/conda/envs/Python36/lib/libxgboost.so(dmlc::StackTrace[abi:cxx11]()+0x55) [0x7f21dc7667a5]\n[bt] (1) /opt/conda/envs/Python36/lib/libxgboost.so(xgboost::ObjFunction::Create(std::\_\_cxx11::basic\_string<char, std::char\_traits<char>, std::allocator<char> > const&)+0x859) [0x7f21dc804c49]\n[bt] (2) /opt/conda/envs/Python36/lib/libxgboost.so(xgboost::LearnerImpl::LazyInitModel()+0x25c) [0x7f21dc773d9c]\n[bt] (3) /opt/conda/envs/Python36/lib/libxgboost.so(XGBoosterUpdateOneIter+0x73) [0x7f21dc8e77c3]\n[bt] (4) /opt/conda/envs/Python36/lib/python3.6/lib-dynload/../../libffi.so.6(ffi\_call\_unix64+0x4c) [0x7f21eee6cec0]\n[bt] (5) /opt/conda/envs/Python36/lib/python3.6/lib-dynload/../../libffi.so.6(ffi\_call+0x22d) [0x7f21eee6c87d]\n[bt] (6) /opt/conda/envs/Python36/lib/python3.6/lib-dynload/\_ctypes.cpython-36m-x86\_64-linux-gnu.so(\_ctypes\_callproc+0x2ce) [0x7f21ef082ede]\n[bt] (7) /opt/conda/envs/Python36/lib/python3.6/lib-dynload/\_ctypes.cpython-36m-x86\_64-linux-gnu.so(+0x13915) [0x7f21ef083915]\n[bt] (8) /opt/conda/envs/Python36/bin/python(\_PyObject\_FastCallDict+0x8b) [0x55dec4d45e3b]\n[bt] (9) /opt/conda/envs/Python36/bin/python(+0x199c0e) [0x55dec4dcdc0e]\n\n'

"""

The above exception was the direct cause of the following exception:

XGBoostError Traceback (most recent call last)

<ipython-input-34-1554eae9bb37> in <module>

**21** xgb\_temp = xgb.XGBRegressor()

**22** model\_xgb\_tuned = RandomizedSearchCV(xgb\_temp, params, n\_iter=3,n\_jobs=-1)

---> 23 model\_xgb\_tuned.fit(x\_train,y\_train\_log)

**24** model\_xgb = xgb.XGBRegressor(\*\*model\_xgb\_tuned.best\_params\_)

**25**

/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/model\_selection/\_search.py in fit(self, X, y, groups, \*\*fit\_params)

**720** return results\_container[0]

**721**

--> 722 self.\_run\_search(evaluate\_candidates)

**723**

**724** results = results\_container[0]

/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/model\_selection/\_search.py in \_run\_search(self, evaluate\_candidates)

**1513** evaluate\_candidates(ParameterSampler(

**1514** self.param\_distributions, self.n\_iter,

-> 1515 random\_state=self.random\_state))

/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/model\_selection/\_search.py in evaluate\_candidates(candidate\_params)

**709** for parameters, (train, test)

**710** in product(candidate\_params,

--> 711 cv.split(X, y, groups)))

**712**

**713** all\_candidate\_params.extend(candidate\_params)

/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/externals/joblib/parallel.py in \_\_call\_\_(self, iterable)

**928**

**929** with self.\_backend.retrieval\_context():

--> 930 self.retrieve()

**931** # Make sure that we get a last message telling us we are done

**932** elapsed\_time = time.time() - self.\_start\_time

/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/externals/joblib/parallel.py in retrieve(self)

**831** try:

**832** if getattr(self.\_backend, 'supports\_timeout', False):

--> 833 self.\_output.extend(job.get(timeout=self.timeout))

**834** else:

**835** self.\_output.extend(job.get())

/opt/conda/envs/Python36/lib/python3.6/site-packages/sklearn/externals/joblib/\_parallel\_backends.py in wrap\_future\_result(future, timeout)

**519** AsyncResults.get from multiprocessing."""

**520** try:

--> 521 return future.result(timeout=timeout)

**522** except LokyTimeoutError:

**523** raise TimeoutError()

/opt/conda/envs/Python36/lib/python3.6/concurrent/futures/\_base.py in result(self, timeout)

**430** raise CancelledError()

**431** elif self.\_state == FINISHED:

--> 432 return self.\_\_get\_result()

**433** else:

**434** raise TimeoutError()

/opt/conda/envs/Python36/lib/python3.6/concurrent/futures/\_base.py in \_\_get\_result(self)

**382** def \_\_get\_result(self):

**383** if self.\_exception:

--> 384 raise self.\_exception

**385** else:

**386** return self.\_result

XGBoostError: b'[07:56:07] src/objective/objective.cc:23: Unknown objective function reg:squarederror\n\nStack trace returned 10 entries:\n[bt] (0) /opt/conda/envs/Python36/lib/libxgboost.so(dmlc::StackTrace[abi:cxx11]()+0x55) [0x7f21dc7667a5]\n[bt] (1) /opt/conda/envs/Python36/lib/libxgboost.so(xgboost::ObjFunction::Create(std::\_\_cxx11::basic\_string<char, std::char\_traits<char>, std::allocator<char> > const&)+0x859) [0x7f21dc804c49]\n[bt] (2) /opt/conda/envs/Python36/lib/libxgboost.so(xgboost::LearnerImpl::LazyInitModel()+0x25c) [0x7f21dc773d9c]\n[bt] (3) /opt/conda/envs/Python36/lib/libxgboost.so(XGBoosterUpdateOneIter+0x73) [0x7f21dc8e77c3]\n[bt] (4) /opt/conda/envs/Python36/lib/python3.6/lib-dynload/../../libffi.so.6(ffi\_call\_unix64+0x4c) [0x7f21eee6cec0]\n[bt] (5) /opt/conda/envs/Python36/lib/python3.6/lib-dynload/../../libffi.so.6(ffi\_call+0x22d) [0x7f21eee6c87d]\n[bt] (6) /opt/conda/envs/Python36/lib/python3.6/lib-dynload/\_ctypes.cpython-36m-x86\_64-linux-gnu.so(\_ctypes\_callproc+0x2ce) [0x7f21ef082ede]\n[bt] (7) /opt/conda/envs/Python36/lib/python3.6/lib-dynload/\_ctypes.cpython-36m-x86\_64-linux-gnu.so(+0x13915) [0x7f21ef083915]\n[bt] (8) /opt/conda/envs/Python36/bin/python(\_PyObject\_FastCallDict+0x8b) [0x55dec4d45e3b]\n[bt] (9) /opt/conda/envs/Python36/bin/python(+0x199c0e) [0x55dec4dcdc0e]\n\n'

## 3.3 LightGBM with Hyper-parameter Tuning

In [ ]:

*#LightGBM: another implementation of grandient boosting*

**import** **lightgbm** **as** **lgb**

print('####################################################**\n{}**\start\_time'

.format(datetime.datetime.now().strftime('%H:%M')))

params = {

'objective':['regression'],

'num\_leaves':[4,5],

'learning\_rate':[0.05],

'n\_estimators': [700,5000],

'max\_bin': [50,200],

'bagging\_fraction':[0.75],

'bagging\_freq':[5],

'bagging\_seed':[7],

'feature\_fraction':[0.2],

'feature\_fraction\_seed':[7]

}

light\_temp = lgb.LGBMRegressor()

model\_lgb\_tuned = GridSearchCV(light\_temp, params, n\_jobs=-1)

model\_lgb\_tuned.fit(x\_train,y\_train\_log)

model\_lgb = lgb.LGBMRegressor(\*\*model\_lgb\_tuned.best\_params\_)

print(model\_lgb)

print('**{}\t**End\_time**\n**####################################################'

.format(datetime.datetime.now().strftime('%H:%M')))

####################################################

07:59\start\_time

## 3.4 Random Forest with Hyper-parameter Tuning

In [ ]:

*#random forest*

**from** **sklearn.ensemble** **import** RandomForestRegressor

print('####################################################**\n{}**\start\_time'

.format(datetime.datetime.now().strftime('%H:%M')))

params = {

'max\_depth': [20,**None**],

'min\_samples\_leaf': [2],

'min\_samples\_split': [4],

'n\_estimators': [200,500],

}

rf\_temp = RandomForestRegressor()

rf\_temp\_tuned = GridSearchCV(rf\_temp, params, n\_jobs=-1)

rf\_temp\_tuned.fit(x\_train,y\_train\_log)

model\_randomforest = RandomForestRegressor(\*\*rf\_temp\_tuned.best\_params\_)

print(model\_randomforest)

print('**{}\t**End\_time**\n**####################################################'

.format(datetime.datetime.now().strftime('%H:%M')))

## 3.5 Use Cross Validation to Compare the Performance and Stacking the Models

In [ ]:

*#Use cross validation to compare the performance*

**from** **sklearn.model\_selection** **import** KFold

*#Validation function*

n\_folds = 5

**def** rmsle\_cv(model):

kf = KFold(n\_folds, shuffle=**True**, random\_state=42).get\_n\_splits(x\_train)

rmse= np.sqrt(-cross\_val\_score(model, x\_train, y\_train\_log, scoring="neg\_mean\_squared\_error", cv = kf))

**return**(rmse.mean())

models = {

'Lightgbm':model\_lgb,

'XGBoost':model\_xgb,

'Lasso':model\_lasso,

'Random forest':model\_randomforest,

'Elastic Net':model\_ENet

}

**for** model\_ind, model\_fn **in** models.items():

print('Fitting:**\t{}**'.format(model\_ind))

model\_fn.fit(x\_train, y\_train\_log)

print('Done! Error:**\t{}\n**'.format(rmsle\_cv(model\_fn)))

*#combine the models*

**class** **AveragingModels**(BaseEstimator, RegressorMixin, TransformerMixin):

**def** \_\_init\_\_(self, models):

self.models = models

*# we define clones of the original models to fit the data in*

**def** fit(self, X, y):

self.models\_ = [clone(x) **for** x **in** self.models]

*# Train cloned base models*

**for** model **in** self.models\_:

model.fit(X, y)

**return** self

*#Now we do the predictions for cloned models and average them*

**def** predict(self, X):

predictions = np.column\_stack([model.predict(X) **for** model **in** self.models\_])

**return** np.mean(predictions, axis=1)

*#combine the model together(stacking)*

averaged\_models = AveragingModels(models = (model\_lgb, model\_xgb,model\_lasso,model\_ENet))

score = rmsle\_cv(averaged\_models)

print(" Averaged base models score: **\t{}\n**".format(score))

## 4. Predictions and Submit the Results

In [ ]:

*#We use the stacked model for our final predictions.*

averaged\_models.fit(x\_train, y\_train\_log)

y\_pred=averaged\_models.predict(x\_test)

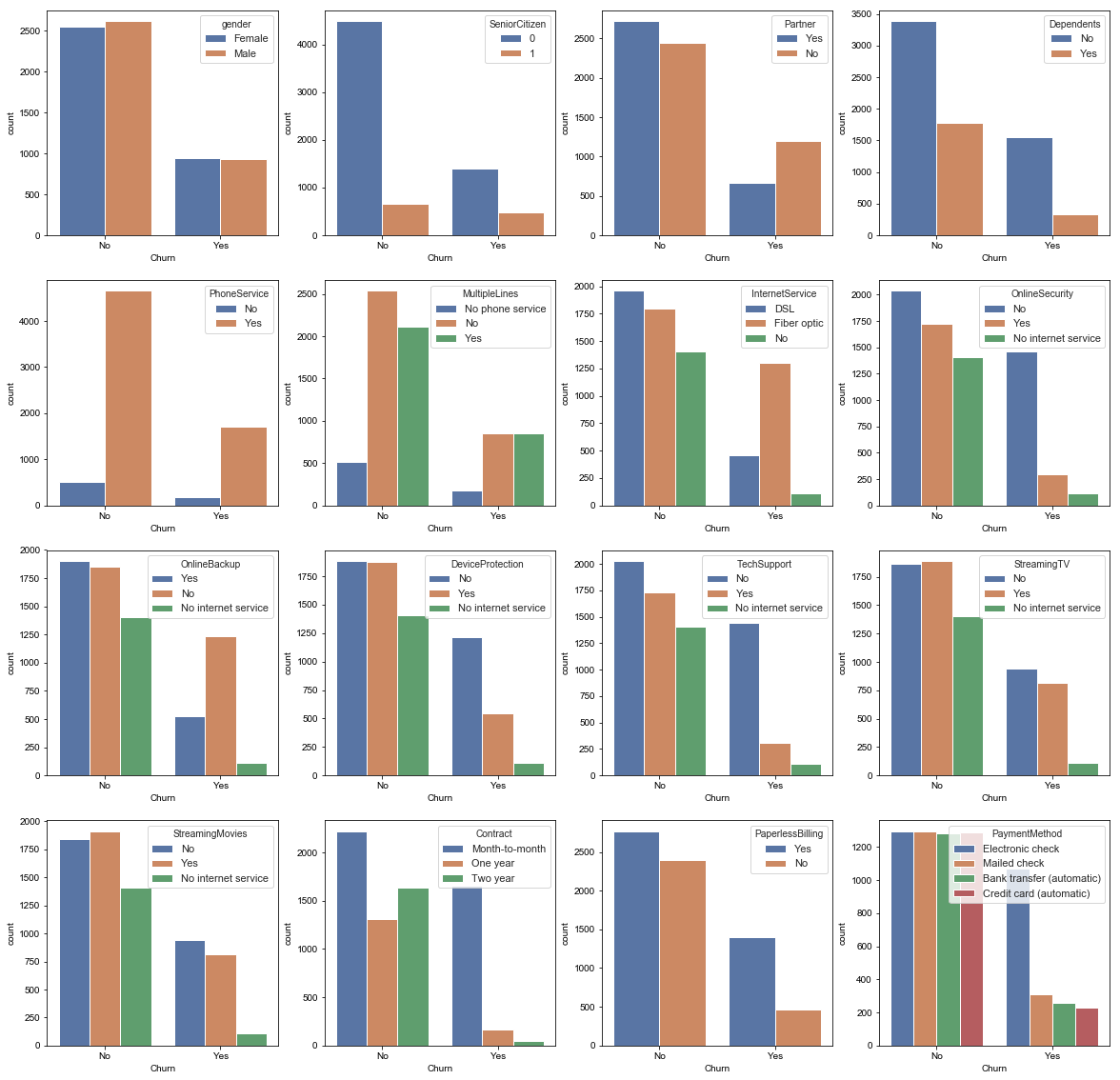
sub = pd.DataFrame()

sub['Id'] = test\_ID

sub['SalePrice'] = np.expm1(y\_pred)

sub.to\_csv('https://raw.githubusercontent.com/huynguyenphu/Advance-Data-Scientist/Aaron\_submission.csv',index=**False**)

In [ ]:



The mean values for numerical columns are given here.

|  | **SeniorCitizen** | **tenure** | **MonthlyCharges** |
| --- | --- | --- | --- |
| **count** | 7043.000000 | 7043.000000 | 7043.000000 |
| **mean** | 0.162147 | 32.371149 | 64.761692 |
| **std** | 0.368612 | 24.559481 | 30.090047 |
| **min** | 0.000000 | 0.000000 | 18.250000 |
| **25%** | 0.000000 | 9.000000 | 35.500000 |
| **50%** | 0.000000 | 29.000000 | 70.350000 |
| **75%** | 0.000000 | 55.000000 | 89.850000 |
| **max** | 1.000000 | 72.000000 | 118.750000 |

As most of the data was categorical, bar plots were used to plot relation between the target variable and a particular column

# Algorithms and Techniques

# Logistic Regression

* Logistic Regression is primarily used for classification problems. The most common examples would be classifying whether an email is spam or not spam, and in this case, classifying whether a customer may churn or no. Logistic Regression makes use of a sigmoid function to map the data into two categories, of 0 and 1 in terms of a binary classification. Logistic Regression makes use of Maximum Likelihood Regression to estimation of regression coefficients. The coefficients obtained are that set of coefficients for which the probability of getting the data we have observed is maximum
* This is used as the benchmark model.
* Logistic Regression was chosen here as it is one of the most common regression algorithms for binary classification in Supervised learning.
* Logistic Regression has shown an accuracy of around 78.60sss percent, which was sent as the benchmark metric to beat. Please look at Data Pre-processing section to understand how the data was encoded for this particular task.

1. XGBOOST

* XGBOOST stands for Xtreme Gradient Boosting. The idea of boosting means, you use multiple weak learners to build a stronger model. Each weak learner is a decision tree with a single split. You use multiple learners to build a stronger and robust model, as each learner qualifies and corrects a mistake made by a previous learner.

For the basics of Boosting, please refer

* Gradient Boosting is a method where new models are created to rectify the residual errors of the previous learner to ultimately make a strong learner. It is called gradient boosting because it uses gradient descent to minimize the loss function.
* XGBoost algorithm is an implementation of gradient boosted decision trees for performance and speed.
* This model is well suited for structured and tabular data and this was used to beat the performance of Logistic Regression.

1. Artificial Neural Network

* Neural networks are built from neurons, which are trained on certain input data, and using activation functions, predict which class a particular data point falls. Each input connected to the layer has certain weights attached to it, which signify how important or how much of weight that particular feature has in predicting the target variable. The activation function, sums up these weights and functions, and based on a certain criteria, that neuron gets activated and allows the input to pass from that node. Cross entropy is then used to determine, how far the prediction is from the actual values and based on this, the weights for each node or input feature are adjusted in order to minimize the error. The neurons thus learn which features actually contribute and how much for making a prediction.
* An artificial Neural network was built to understand patterns between the data, and was used this particular problem is a good fit for the neural network.
* The training data isn’t much, and can will not take that long to train. Neural networks update the weights for each node according the accuracy obtained during forward propagation, and learn which features provide more accuracy.

# Benchmark model

The benchmark model whose performance we’re looking to beat is Logistic Regression model. For performance of this model please refer to the Implementation

# Methodology

# Data Pre-processing

* First, the Total charges column was converted to a numeric value, and then the rows which had missing values for that column were dropped. The number of missing columns here were 11, which is a small number, and were hence dropped from the data frame.
* All the categorical variables were encoded and converted to numerical values. Pandas getdummies method was used to encode the data.
* Standard scaling was used to scale to balance the difference in values of the various columns
* The data was split into training and testing sets

# Implementation

A Jupiter Notebook and the editor Spyder have been used for the implementation.

# Logistic Regression

The following steps were performed for Logistic Regression.

* The data was loaded into a dataframe using pandas
* Pandas methods info() and describe were used to display statistics and general information about the dataset.
* Data Pre-processing was performed as mentioned above using libraries from sklearn.
* Data was split into training and testing as mentioned above.
* The module Logistic Regression was imported from sklearn.linear\_models
* The training data was fit to the classifier and then a prediction was made on the testing set.
* Performance: This model gave an accuracy of 78.60 on unseen data.

XGBOOST

* Xgboost model was first imported.
* The XGBOOST classifier was fit onto the training set.
* A prediction was made on the test set.
* On the first prediction it had an accuracy of only over 78.89 percent, which is more than Logistic Regression, but only marginally
* Parameter tuning was then performed on it, after which the accuracy improved slightly to above 80 percent.

Artificial Neural Networks

* Installation of tensorflow and keras
* Importing classes like Sequential and Dense from keras modules
* Built a deep neural network using Sequential and Dense
* Set appropriate network parameters, loss function and estimator.
* Trained the network on training set
* Made a prediction on the testing data. Gave an accuracy of more than 79 percent, around 79.30

# Refinement

# XGBOOST

* For XGBoost , GridSearch CV was used to find out best values for n\_Estimators,max\_depth,

Min\_child\_weight.

* Grid search cv was used to find out optimal parameters for XGBOOST
* After finidng out optimal parameters for them, the values were plugged into the model and the model was trained on the testing set.
* The grid search CV takes a bit of time to execute.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Max\_Depth | Min\_child\_weight | Learningrate | Subsample | Colsamplebytree | Reg\_alpha |
| Range(3,10,2) | Range(1,6,2) | 0.1 | 0.8 | 0.8 |  |
| 3 | 5,6,8,10,12 | 0.1 | 0.8 | 0.8 |  |
| 3 | 12 | 0.1 | Range(0.6,1) | Range( |  |
| 3 | 12 | 0.1 | 0.8 | 0.9 | 0.01 |

The accuracy was improved to about 80.02 percent on unseen data, which is more than the one obtained on Logistic Regression

# Artificial Neural Network

* Dropout layers were added after the input layer and hidden layers as well, with a dropout rate of 0.2
* Grid search cv was conducted and the values for n\_epoch and batch size was optimized.

# Results

# Model Evaluation and Validation

* Grid Search cv and k fold validation was used to test the model.
* The final parameters for XGBOOST were chosen because they gave the best accuracy, of around 80.5 percent. This was obtained by tuning each of the parameters and using grid search for each of the parameters.
* However, when a new model was created, its accuracy was only slightly better than that given by a logistic regression.
* For neural networks as well, accuracy is only more with a very tiny difference.

|  |  |
| --- | --- |
| Model | Accuracy(in percent) |
| Logistic Regression | 78.89 |
| XGBOOST | 80.02s |
| Artifical Neural Network | 79.38 |

# Conclusion

# Reflection

The steps followed in this project can be summarized as follows

* Relevant dataset was found for the dataset.
* The data was loaded into dataframes in pandas and appropriate processing was performed.
* The data was split into training and testing sets
* Logistic Regression was chosen as a benchmark model whose performance XGBoost and am artificial neural network is supposed to beat
* The accuracy was found out both by using sklearn’s function and generating the confusion matrix.
* XGBoost was imported and some further processing was done.
* Training data was fit to this classifier, and then tested on test data.
* GridSearch CV was used to find out the best parameters.
* Grid search CV here was a little slow
* Building a neural net for this problem once the pre-processing was done was fairly simple
* For the ANNS, grid search CV took a lot of time. Increasing the accuracy of the ANN was the most difficult part of the project.
* Dropout layers were added to try and tune the neural network.
* Before actually building an algorithm, lot of thinking had to go into the cleaning and pre-processing of data.
* I tried both one hot encoder and the getdummies method from pandas.
* I had to use predict\_classes instead of predict as the final activation function in my neural network was sigmoid function.

# Challenges faced

* The numpy installation had to be upgraded, which was giving a fatal error otherwise. The error is mentioned below
* ModuleNotFoundError: No module named 'numpy.core.\_multiarray\_umath'
* Another error encountered because of the issue of the numpy version was.

An error ocurred while starting the kernel 2019󈚦󈚸 01:46:35.136525: F tensorflow/python/lib/core/bfloat16.cc:675] Check failed: PyBfloat16\_Type.tp\_base != nullptr

* The Dense API had to be updated to the latest API.
* As the data had a lot of categorical variables, I tried both One Hot encoding and pandas.getdummies for this purpose.

# Improvement

* Time for grid search cv of the neural network needs to be improved. Maybe better hardware or using a GPU should improve it.
* The models XGBOOST and ANN’S have not beaten the Logistic Regression model by a huge margin in terms of the accuracy score.