Model Selection Based on KPIs

Ananth Subramanian, Vitaly Briker, Jackson Au & Richard Farrow

December 06, 2019

1. Introduction

"Sigmoid" consulting group has been tasked to arrive at an optimal model that will save cost to "Adam Corporation" (here onwards referred as AC). Sigmoid has been provided with a dataset with a mixture of different features. The Key Performance Indicators are better "Recall" and "Precision". Sigmoid loses $100 for every false positive detected. Hence more importance should be given to "Precision" score. At the same time, it is mandatory to have good "Recall" as well as Sigmoid tends to lose $10 for every false negative.

2. Methods

Sigmoid went about the analysis of dataset and started with an Exploratory Data Analysis (EDA) of numerical variables. The EDA results showed that all the variables have close to normal distribution.
The numerical missing values that compose between 0.1-0.3% were imputed by mean value.
The observations with categorical missing data that compose between 0.1-0.3% were deleted from the dataset. One-hot-encoding was used to convert the categorical features.

On the reduction of correlated features, two features that had correlation greater than 0.5 were removed from the dataset.

Sigmoid also developer a utility function for reporting. This function reports the type of classifier used, the accuracy score of the classifier, Grid Search parameters, recall, precision, etc. This report function helped in decision making, ie. choosing the best model.

A new data set was created, and train and test data were created out of it. The train data set was used to train the models in 1st stage and the test was used to perform stage 2 modeling.

As a next step with the test set, and using K-fold (cv=5), the following classifiers were used to identify best estimators and create a baseline score :

1. Logistic Regression
2. SGD (Stochastic Gradient Descent) Classifier
3. Bernoulli Naïve Bayes
4. Gaussian Naïve Bayes
5. Decision Tree
6. ADA Boost
7. Random Forest
8. XG Boost

The metrics that were reported from these classifiers is “AUC score” (more information on this Results Section)

Random Forest provided a better result compared to other classifiers, in addition XGB boost was selected as second classifier to create and ensemble model. RF uses bagging technique while XGB is a boosting type.

Sigmoid chose Random Forest recursive feature elimination and cross-validated selection of the best number of features. The feature removal finally resulted in 15 explanatory variables.

Post using Random Forest recursive functionality, a new data set was created with reduced features. With this new data set, metrics like Accuracy and AUC along with confusion matrix were calculated.

Following the Random Forest and XGB Boos Classifiers, fine-tuned using Random Grid search function and cross validation.

Ensemble: Best practice as per Data Scientists are “Ensembles”. Ensemble refers to combining different models ie. the outputs from the various Level 1 models are then input to Level 2 model. Sigmoid used Random Forest Classifier and XG Boost as Level 1 models and Logistic Regression as Level 2 model Probabilities are predicted from both the Level 1 models and the obtained probabilities are then fed to a Level 2 model. The process of fitting the entire “Training” dataset using Random Forest and XG Boost was carried out. The output is then fed to the Logistic Regression model and fitted. The outcome are discussed in the Results Section.

Model Tuning: The primary objective for Sigmoid is to minimize losses ie. to arrive at a model with optimized “Threshold value”. The optimization of threshold value is to find a threshold that has minimum “Dollar Loss”.

A list of threshold values (range of 0.5 – 1) was defined and a utility function was developed. The function calculates Precision and Recall at various thresholds and chooses the threshold that strikes a balance between these parameters to get the minimum loss.

The model that fits the objective and the corresponding outcomes are discussed in the Results section.

3. Results

- **Exploratory data analysis outcome:**

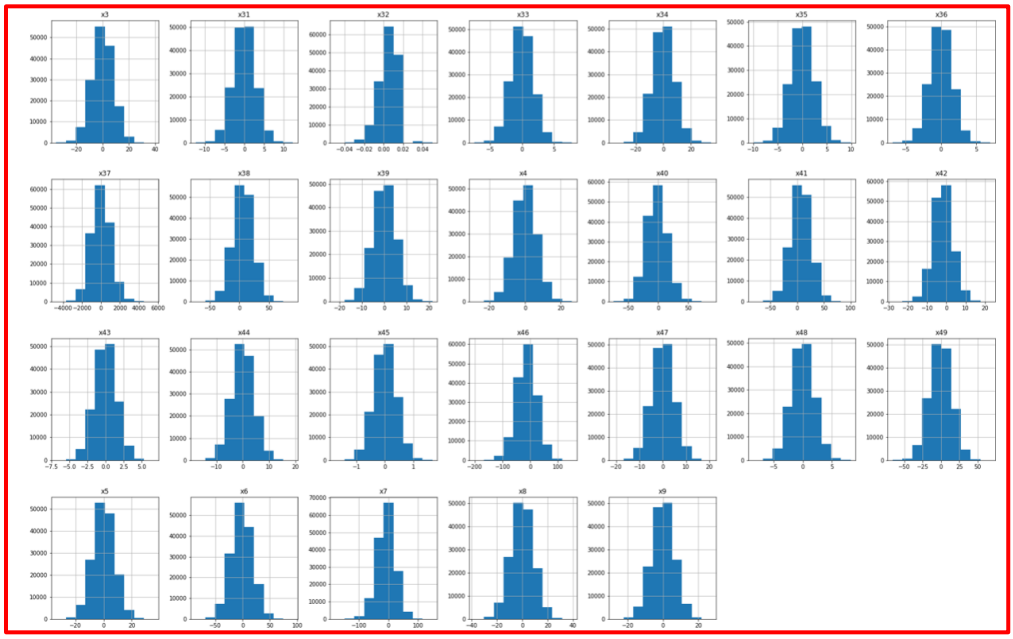


Figure 1 : Numerical Features Histogram

✓ **The numerical are normally distributed**

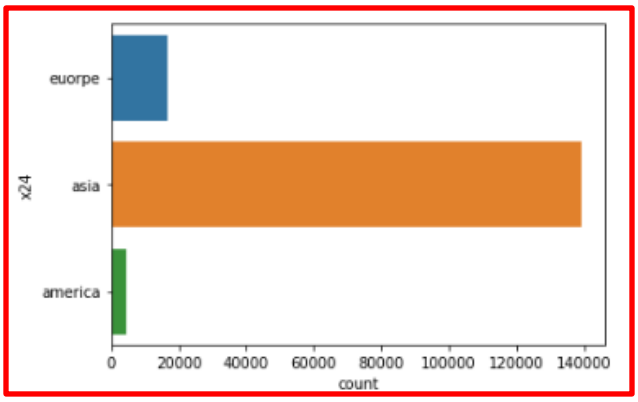


Figure 2 : Categorical feature distribution – Name of the continent

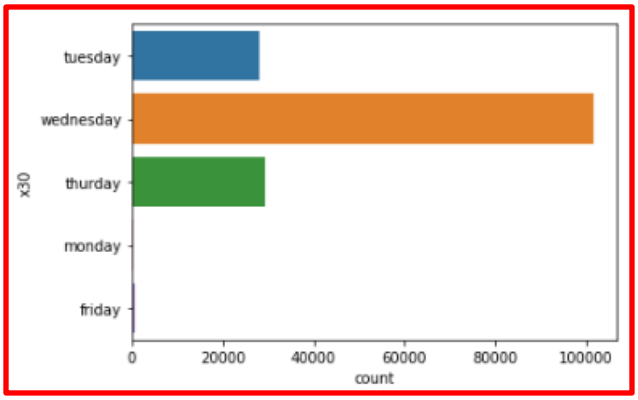


Figure 3 : Categorical feature distribution – Day of the week

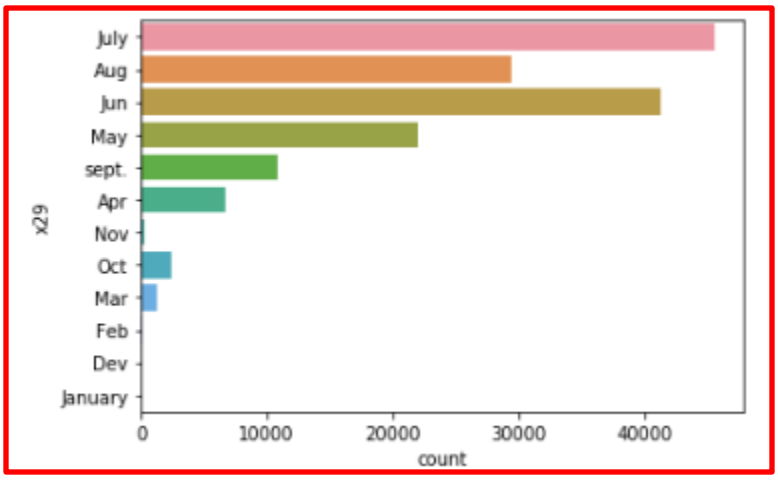


Figure 4 : Categorical feature distribution - Month

- **Correlation explanatory variables analysis:**

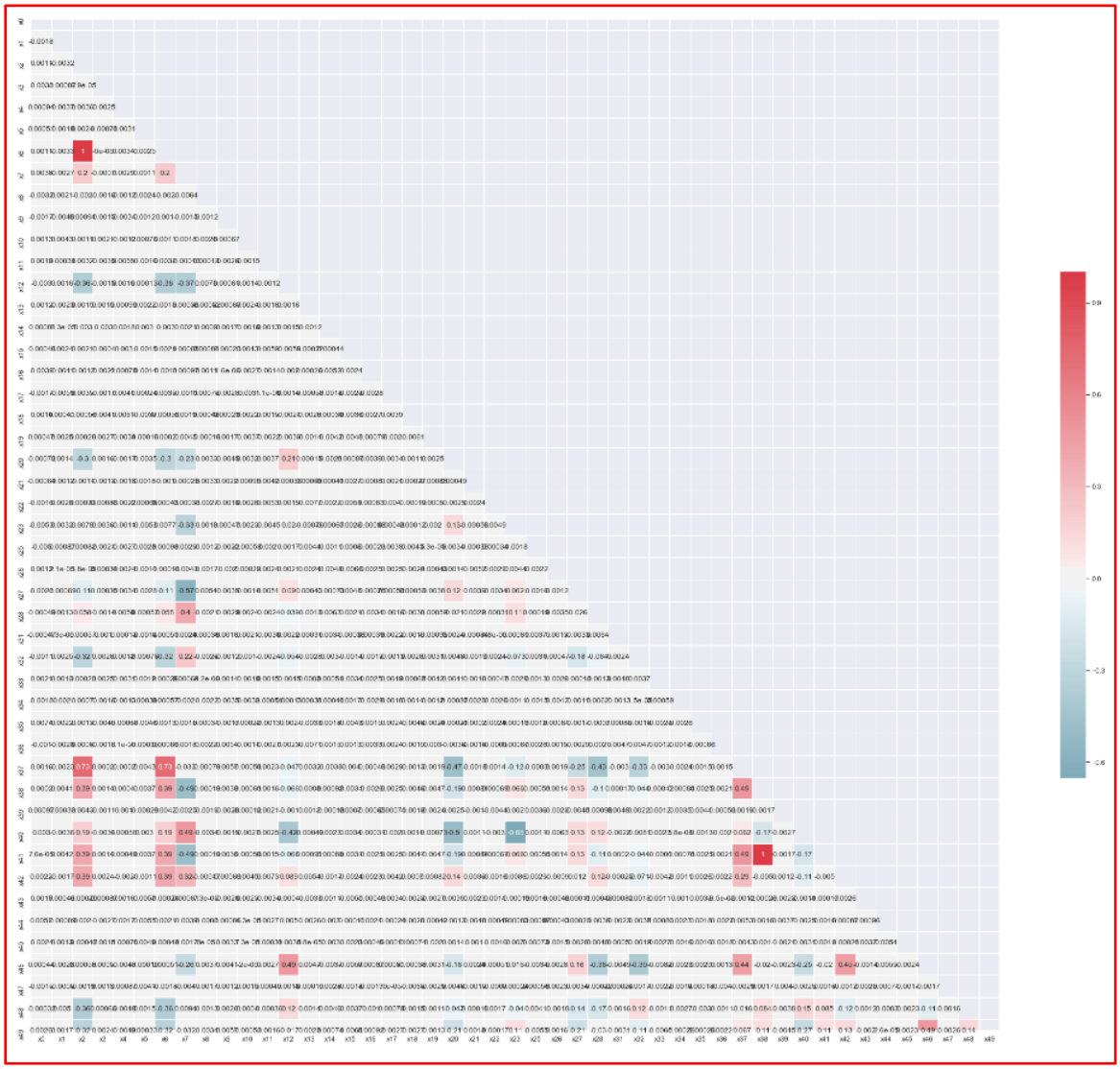


Figure 5 : Correlation matrix

	Field1	Field2	Correlation
0	x6	x2	0.999731
1	x41	x38	0.999755

Figure 6 : Highly correlated features (>95%)

✓ **The variables above were removed during further analyses.**

➢ **Data splitting into train and test sets :**

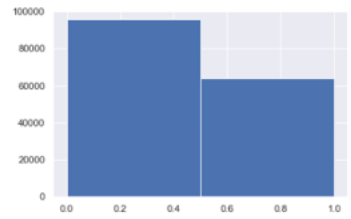
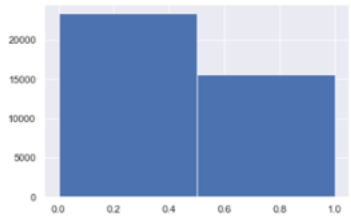
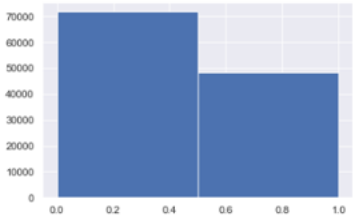
	Entire Data set	Test Data set	Train Data set
			
Class Ratio	1.49	1.5	1.49

Figure 7 : Response variable Class Ratio

✓ **The ratio of response classifier stayed the same after splitting the data into train and test sets.**

➢ **Results from various classifiers (k-fold splitting was used):**

	LR	SGD	BNB	GNB	DT	ADA	RF	XGB
Score	0.7	0.69	0.6	0.69	0.84	0.74	0.89	0.83
Time (sec)	9.17	3.51	1.06	1.33	33.08	133.02	25.9	161.63

Table 1 : Classifiers AUC score comparison

✓ **As seen above, Random Forest Classifier gives the best accuracy among all models.**

- **<u>Random Forest Recursive Outcome</u>:**

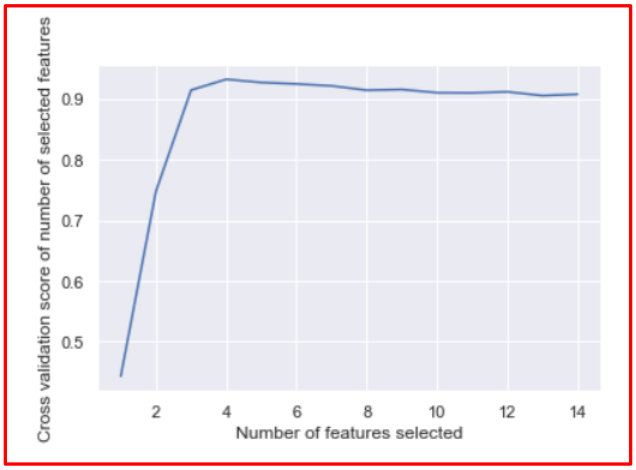


Figure 8 : Features importance using recursive method

- ✓ **As seen in the plot above, the cross validated score is mostly flat after 15 features with the same score. So only those features were carried forward in further analyses.**

- **Hyper parameters fine tuning :**


```
{'Classifier': 'Random Forest',
 'Parameters': "{'n_estimators': 200, 'min_samples_split': 12, 'min_samples_leaf': 25, 'max_features': 0.2}",
 'Accuracy_Mean': 0.91575,
 'Accuracy_Std': 0.001389194410840749,
 'AUC_Mean': 0.9716061035484408,
 'AUC_Std': 0.0011637065071580046,
 'MSE_Mean': -0.08425,
 'MSE_Std': 0.0013891944108407248,
 'Precision_Mean': 0.9242607376961871,
 'Precision_Std': 0.0014816856668531133,
 'Recall_Mean': 0.8606472341250655,
 'Recall_Std': 0.003153850651653802,
 'R2_Mean': 0.6493614278110206,
 'R2_Std': 0.005489883281733824}
```


Figure 9 : Random Forest Random grid search result


```
{'Classifier': 'XGB_Booster',
 'Parameters': "{'subsample': 0.6, 'min_child_weight': 5, 'max_depth': 5, 'gamma': 2, 'colsample_bytree': 0.8}",
 'Accuracy_Mean': 0.8947,
 'Accuracy_Std': 0.001952633549281031,
 'AUC_Mean': 0.958297950488861,
 'AUC_Std': 0.0016531020787175447,
 'MSE_Mean': -0.1053,
 'MSE_Std': 0.0019526335492810129,
 'Precision_Mean': 0.8922080394795255,
 'Precision_Std': 0.0032305087812282006,
 'Recall_Mean': 0.839022733615824,
 'Recall_Std': 0.0037367643260248218,
 'R2_Mean': 0.5617390848728934,
 'R2_Std': 0.00860474416557416}
```


Figure 10 : XGB Boost Random grid search result

- **Train data set - AUC Score on out of fold data set after grid search using KFold (K=5) :**

	Random Forest	XGB Boost
Baseline Score	0.89	0.83
Post Grid Search	0.91	0.89

Table 2 : Random Forest and XGB Boost Pre and Post Grid Search Result

Note : * AUC score calculated using “Predict” function.

- ✓ **Some improvement in Random Forest result and significant improvement in XGB result**

- **Ensemble Model Result Level1: Random Forest, XG Boost, Level2: Logistic Regression:**
  - **Test data set - AUC Score comparison:**

	Random Forest	XGB Boost	Ensemble Model
AUC Score	0.91	0.89	0.89

Table 3 : Random Forest, XGB Boost and Ensemble models

Note : * AUC score calculated using “Predict” function.

```
=== Confusion Matrix Ensemble Model ===
[[21371  1977]
 [ 2255 13309]]


=== Classification Report Ensemble Model ===
              precision    recall  f1-score   support

           0       0.90      0.92      0.91     23348
           1       0.87      0.86      0.86     15564

    accuracy                           0.89     38912
   macro avg       0.89      0.89      0.89     38912
weighted avg       0.89      0.89      0.89     38912

============================================================
```

Figure 12: Ensemble model confusion matrix with default threshold

- **Confusion matrix Threshold optimization outcome:**

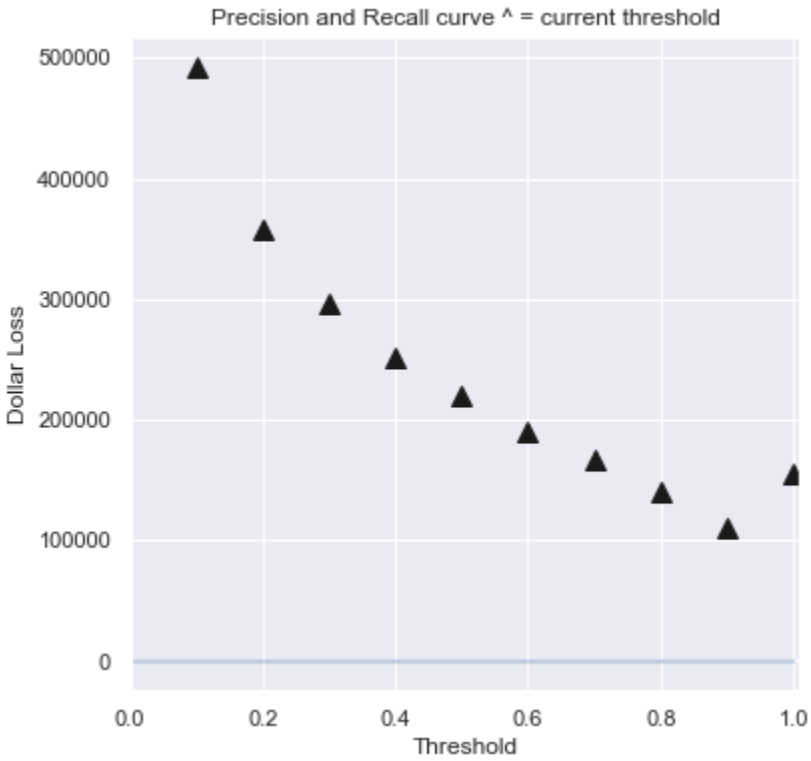


Figure 11: Threshold optimization skew test

- ✓ **Best threshold to minimize the loss is 0.9.**

	Correct Classification TP + TN	Dollar loss
Ensemble Model with default threshold	34681	$219970
Ensemble Model with Optimized threshold (0.9)	33857	$110580
Change	2.5% Reduction	50% Reduction

Table 4: Confusion matrix comparison

The utility function created for the purpose was used to take the inputs from the Ensemble model. Based on the outcome, the loss is reduced from $219970 to $110580 that is about 50% improvement compare to baseline model.

Upon using the threshold as recommended by the function, recalculation of metrics using the ensemble model yields the following results:

```
=== Confusion Matrix ===
[[22681   667]
 [ 4388 11176]]


=== Classification Report ===
              precision    recall  f1-score   support

           0       0.84      0.97      0.90     23348
           1       0.94      0.72      0.82     15564

    accuracy                           0.87     38912
   macro avg       0.89      0.84      0.86     38912
weighted avg       0.88      0.87      0.87     38912



Ensemble model auc Score with default threshold using predict is 0.885220
Ensemble model auc Score with optimized threshold using predict is 0.844750
```

Figure 13 : Confusion matrix post threshold optimization

The best threshold to minimize the loss is 0.9.
The AUC score decreased from 0.88 to 0.84

There is a reduction in accuracy score, but the most important objective of reducing the losses has been achieved.

4. Conclusion

Sigmoid Consulting went on their task of creating the model lowering accuracy by optimizing company money loss in case of misclassification prediction. It is learnt that the fruits of success lie in feature reduction, ensemble technique and defining the functions for model tuning.

5. Appendix – Code

```
#Python :
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import Imputer
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report,
confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn import metrics
from sklearn.metrics import roc_auc_score
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import RFECV
from sklearn.model_selection import train_test_split
import sklearn
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import SGDClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier,
GradientBoostingClassifier, AdaBoostClassifier
from sklearn.naive_bayes import BernoulliNB, GaussianNB
from xgboost import XGBClassifier
from time import time
from sklearn.metrics import precision_recall_curve
import warnings
warnings.simplefilter(action='ignore',
category=FutureWarning)

#load the data set
%time finalProjectDF = pd.read_csv("final_project.csv")

finalProjectDF.head()

finalProjectDF.shape

UniqueValueCounts = finalProjectDF.nunique(dropna=False)
UniqueValueCounts
#rows 24,29 30,32,37 are need check
#missing values are exist

#Remove percent sign
for i, row in enumerate(finalProjectDF['x32']):
    if "%" in str(row):
        cell = float(row.rstrip("%"))
        finalProjectDF.set_value(i,'x32',cell)
finalProjectDF['x32'] = finalProjectDF['x32'] .astype(float)
```

```
#Remove dollar sign
for i, row in enumerate(finalProjectDF['x37']):
    if "$" in str(row):
        cell = float(row.replace("$"," "))
        print(cell)
        finalProjectDF.set_value(i,'x37',cell)
finalProjectDF['x37'] = finalProjectDF['x37'] .astype(float)

#check missing values
NullValueCounts = finalProjectDF.isnull().sum()
print ("Missing values")
print(NullValueCounts/len(finalProjectDF)*100)

#response variable
response=finalProjectDF['y']
response.hist(bins=2)

print ("Ratio between response classes")
sum(finalProjectDF['y']==0)/sum(finalProjectDF['y']==1)

#selct only numeric columns
only_float=finalProjectDF.select_dtypes(include ='float64')
only_float.shape

#plot histogram only numerical columns
#looks like all data has close to normal distribution
fig = plt.figure(figsize = (30,35))
ax = fig.gca()
only_float.hist(ax = ax)

#fill with mean values
only_float.fillna(only_float.mean(), inplace=True)
NullValueCounts = only_float.isnull().sum()
NullValueCounts

#no missing values
only_float.describe()

#scale numerical values from 0 to 1
scaler = MinMaxScaler()
scaler.fit(only_float)
only_float_scaled =
pd.DataFrame(scaler.transform(only_float),
index=only_float.index, columns=only_float.columns)
only_float_scaled.describe()

#plot categorical variables
ax = sns.countplot(y="x24", data=finalProjectDF)

ax = sns.countplot(y="x29", data=finalProjectDF)

ax = sns.countplot(y="x30", data=finalProjectDF)
```

```
# Plot correlation between numerical features
#Few features have high correlation, we will delete >0.95
corr = only_float_scaled.corr()

# Generate a mask for the upper triangle
sns.set(font_scale=1)
mask = np.zeros_like(corr, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(35, 30))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(220, 10, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
ax = sns.heatmap(corr, mask=mask, cmap=cmap, vmax=1,
center=0,
            square=False, linewidths=.5, cbar_kws={"shrink":
.5},annot=True)

# Utility function to report best scores
def report(grid_clf,    n_top=3, clf_name = "",
verbose=True):

    if(verbose):
        for i in range(1, n_top + 1):
            candidates =
np.flatnonzero(grid_clf.cv_results_['rank_test_auc'] == i)
            for candidate in candidates:
                print("Model with rank: {0}".format(i))
                print("Mean validation score (AUC on Test):
{0:.3f} (std: {1:.3f})".format(

grid_clf.cv_results_['mean_test_auc'][candidate],

grid_clf.cv_results_['std_test_auc'][candidate]))
                print("Parameters:
{0}".format(grid_clf.cv_results_['params'][candidate]))
                print("")
    return {
        "Classifier":clf_name,
        "Parameters":str(grid_clf.best_params_),

"Accuracy_Mean":grid_clf.cv_results_["mean_test_acc"][grid_c
lf.best_index_],

"Accuracy_Std":grid_clf.cv_results_["std_test_acc"][grid_clf
.best_index_],

"AUC_Mean":grid_clf.cv_results_["mean_test_auc"][grid_clf.be
st_index_],

"AUC_Std":grid_clf.cv_results_["std_test_auc"][grid_clf.best
_index_],

"MSE_Mean":grid_clf.cv_results_["mean_test_mse"][grid_clf.be
st_index_],

"MSE_Std":grid_clf.cv_results_["std_test_mse"][grid_clf.best
```

```
_index_],

"Precision_Mean":grid_clf.cv_results_["mean_test_precision"]
[grid_clf.best_index_],

"Precision_Std":grid_clf.cv_results_["std_test_precision"][g
rid_clf.best_index_],

"Recall_Mean":grid_clf.cv_results_["mean_test_recall"][grid_
clf.best_index_],

"Recall_Std":grid_clf.cv_results_["std_test_recall"][grid_cl
f.best_index_],

"R2_Mean":grid_clf.cv_results_["mean_test_r2"][grid_clf.best
_index_],

"R2_Std":grid_clf.cv_results_["std_test_r2"][grid_clf.best_i
ndex_],
    }


#utility function to remove correlated features

def reduce_features(df, verbose = False):
    # calculate the correlation matrix
    corr_matrix  = df.corr().abs()

    # Select upper triangle of correlation matrix
    upper =
corr_matrix.where(np.triu(np.ones(corr_matrix.shape),
k=1).astype(np.bool))

    # Find index of feature columns with correlation greater
than 0.95
    to_drop = [column for column in upper.columns if
any(upper[column] > 0.95)]

    #Get all of the correlation values > 95%
    x = np.where(upper > 0.95)

    #Display all field combinations with > 95% correlation
    cf = pd.DataFrame()
    cf['Field1'] = upper.columns[x[1]]
    cf['Field2'] = upper.index[x[0]]

    #Get the correlation values for every field combination.
(There must be a more pythonic way to do this!)
    corr = [0] * len(cf)
    for i in range(0, len(cf)):
        corr[i] =  upper[cf['Field1'][i]][cf['Field2'][i]]

    cf['Correlation'] = corr

    if( verbose ):
        print('There are ', str(len(cf['Field1'])), ' field
correlations > 95%.')
        display(cf)

        print('Dropping the following ', str(len(to_drop)),
```

```
' highly correlated fields.')
        to_drop

    #Check columns before drop
    if( verbose ):
        print('\r\n*********Before: Dropping Highly
Correlated Fields***********************************')
        display(df.info(verbose=False))

    # Drop the highly correlated features from our training
data
    df = df.drop(to_drop, axis=1)

    #Check columns after drop
    if( verbose ):
        print('\r\n*********After: Dropping Highly
Correlated Fields***********************************')
        df.info(verbose=False)

    return df


#one hot encoding function
def get_one_hot_encodings(df, cols):
    result = pd.DataFrame()
    i = 0
    for col in cols:
        dummies = pd.get_dummies(df[col],prefix=col)
        if( i == 0 ):
            result = dummies.copy()
        else:
            result = pd.concat((result, dummies), axis=1)
        i+=1
    return result

only_cat=finalProjectDF.select_dtypes(include ='object')
only_cat.describe()


#create temprorary dataframe
temp = pd.concat(
    (
        only_float_scaled,
        only_cat,
        response_data
    ), axis = 1)
temp.describe()


#Remove records wih missing data of categorical variables,
the percentage of missing data is very low
temp.dropna(inplace=True)
temp.describe()

#Reduce correlated features
print("Reduce Features based on highly correlated values >
.95")
finalProjectDF_corr = reduce_features(temp, verbose=True)
pd.reset_option('max_row')
```

```
# Save csv
finalProjectDF_corr.to_csv("VB_Clean.csv",index=False)


#split into train and test set
# thest set will be used for stage 2

train_data=finalProjectDF_corr.iloc[0:120000]
test_data=finalProjectDF_corr.iloc[121000:len(finalProjectDF
)]

#response variable of test chunk
response=test_data['y']
response.hist(bins=2)
sum(test_data['y']==0)/sum(test_data['y']==1)

#response variable of train chunk
response=train_data['y']
response.hist(bins=2)
sum(train_data['y']==0)/sum(train_data['y']==1)

#split data by type
#selct only numeric columns
only_float=train_data.select_dtypes(include ='float64')
only_cat=train_data.select_dtypes(include ='object')
response_data=train_data['y']

#create data frame for modeling
model_data = pd.concat(
    (
        only_float,
        get_one_hot_encodings(only_cat,only_cat.columns)
    ), axis = 1)

print("Total Features after One-Hot Encoding: ",
model_data.shape )

#create data frame for modeling
model_datatosave = pd.concat(
    (
        only_float,
        get_one_hot_encodings(only_cat,only_cat.columns),
        response_data
    ), axis = 1)
# Save csv
model_datatosave.to_csv("model_datatosave.csv",index=False)

#fit model and calculate CV score using default parameters
for baselone score
#using different estimaters

X = model_data.values
y = response_data.values

# Creat eth Cross Validation Objected used for all tests
num_cv_iterations = 5
random_st = 42
```

```
kfold_cv = KFold(
    n_splits=num_cv_iterations,
    random_state = random_st
)

clfs = {
    "Logistic Regression" :
LogisticRegression(random_state=42),
    "SVM (SGD)" :
SGDClassifier(loss="hinge",random_state=42),
    "Bernouli Naive Bayes" : BernoulliNB(),
    "Gaussian Naive Bayes" : GaussianNB(),
    "Decision Tree" : DecisionTreeClassifier(),
    "AdaBoost" : AdaBoostClassifier(),
    "Random Forest" :
RandomForestClassifier(random_state=42),
    "XGB" : XGBClassifier(random_state=42)
}
print(kfold_cv,"- fold cross validation:\n")

#Store values for future reference
stats = []
print('Using the default values from scikit-learn version is
{}.'.format(sklearn.__version__))
print('We achieved the following area under curve scores.')
print ("\n")
for label, clf in clfs.items():
    start = time()
    scores = cross_val_score(clf, X, y, cv=kfold_cv,
scoring='accuracy')
    r = {
        "Classifier":label,
        "AreaUnderCurve":scores.mean(),
        "AUC_StandardDeviation":scores.std()
    }
    stats.append(r)
    print("%s - Accuracy: %0.2f (+/- %0.5f)"
          % (label, scores.mean(), scores.std()))
    tt=time() - start
    print("Process Time : ",tt)
    print ("\n")

%%time
#RF has better score, next let's try to remove features with
Feature ranking with
#recursive feature elimination and cross-validated selection
of the best number of features
#we will optimize the methid using precsion as scoring

from sklearn.feature_selection import RFECV
from sklearn.model_selection import train_test_split

X = model_data.values
y = response_data.values

x_train, x_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

clf_rf = RandomForestClassifier(random_state=42)
```

```
rfecv = RFECV(estimator=clf_rf, step=5, cv=5,
scoring='precision')
rfecv = rfecv.fit(x_train, y_train)

print('Optimal number of features :', rfecv.n_features_)
print('Best features :', model_data.columns[rfecv.support_])

# Plot of number of features vs CV score
plt.figure()
plt.xlabel("Number of features selected")
plt.ylabel("Cross validation score of number of selected
features")
plt.plot(range(1, len(rfecv.grid_scores_) + 1),
rfecv.grid_scores_)
plt.show()

# convert data after reducing features

x_train_rfecv = rfecv.transform(x_train)
x_test_rfecv = rfecv.transform(x_test)
rf_rfecv_model = clf_rf.fit(x_train_rfecv, y_train)

print ("Train Accuracy  :: ", accuracy_score(y_train,
rf_rfecv_model.predict(x_train_rfecv)))
print ("Test Accuracy  :: ", accuracy_score(y_test,
rf_rfecv_model.predict(x_test_rfecv)))
print ("AUC" , roc_auc_score(y_test,
rf_rfecv_model.predict(x_test_rfecv)))

print("=== Confusion Matrix ===")
print(confusion_matrix(y_test,
rf_rfecv_model.predict(x_test_rfecv)))
print('\n')
print("=== Classification Report ===")
print(classification_report(y_test,
rf_rfecv_model.predict(x_test_rfecv)))
print('\n')

# After removing features we down to 15 only
model_data_cut=model_data[model_data.columns[rfecv.support_]
]
model_data_cut.shape

# Save csv
model_data_cut.to_csv("VB_Clean_cut.csv",index=False)

%%time
#Grid Random search using RF

from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import KFold
from sklearn.model_selection import RandomizedSearchCV

# Creat eth Cross Validation Objected used for all tests
num_cv_iterations = 5
random_st = 42
```

```
kfold_cv = KFold(
    n_splits=num_cv_iterations,
    random_state = random_st
)

clf = RandomForestClassifier(random_state=42)

grid_params = {
    "max_features" : [0.20, 0.30],
    "n_estimators" : [100,200],
    "min_samples_leaf" : [25, 50],
    "min_samples_split": [8, 10, 12]
}

scoring = {
    'acc':'accuracy',
    'precision':'precision',
    'recall':'recall',
     'auc':'roc_auc',
     'mse':'neg_mean_squared_error',
     'r2':'r2'
}

grid_clf_random = RandomizedSearchCV(
    estimator = clf,
    param_distributions=grid_params,
    cv=kfold_cv,
    scoring=scoring,
    refit='precision',
    n_jobs=-2,verbose=1,return_train_score=False, n_iter=10)

grid_clf_random.fit(X, y)

# Print scores and best parameters
report(grid_clf_random, clf_name="Random Forest")

%%time
# Grid Random search using  XGB classifier

X = model_data_cut.values
y = response_data.values

num_cv_iterations = 5
random_st = 42
kfold_cv = KFold(
    n_splits=num_cv_iterations
)

clf_xgb = XGBClassifier(random_st)

grid_params = {
        'min_child_weight': [1, 5, 10],
        'gamma': [0.5, 1, 1.5, 2, 5],
        'subsample': [0.6, 0.8, 1.0],
        'colsample_bytree': [0.6, 0.8, 1.0],
        'max_depth': [3, 4, 5]
        }

scoring = {
```

```
    'acc':'accuracy',
    'precision':'precision',
    'recall':'recall',
     'auc':'roc_auc',
     'mse':'neg_mean_squared_error',
     'r2':'r2'
}

grid_clf_random_xgb = RandomizedSearchCV(
    estimator = clf_xgb,
    param_distributions=grid_params,
    cv=kfold_cv,
    scoring=scoring,
    refit='precision',
    n_jobs=-2,verbose=1,return_train_score=False, n_iter=10)

grid_clf_random_xgb.fit(X, y)

# Print scores and best parameters
report(grid_clf_random_xgb, clf_name="XGB_Booster")

#Create a RF model using best parameters
rf_Kfold=KFold(n_splits=5)

rf=RandomForestClassifier(n_estimators= 200,
min_samples_split= 10, min_samples_leaf= 25, max_features=
0.3)

# using Kfold calculate the ROC score for each fold
rf_oof_predict_proba=np.zeros(X.shape[0])
rf_oof_predict=np.zeros(X.shape[0])
ave_score=[]
for train_idx, test_idx in rf_Kfold.split(X):
    rf.fit(X[train_idx],y[train_idx])

rf_oof_predict_proba[test_idx]=rf.predict_proba(X[test_idx])
[:,1]
    rf_oof_predict[test_idx]=rf.predict(X[test_idx])

ave_score.append(roc_auc_score(y[test_idx],rf_oof_predict[te
st_idx]))
#     print("Fold ROC score is %f" %
roc_auc_score(y[test_idx],rf_oof_predict_proba[test_idx]))
#     print("Fold ROC score is %f" %
roc_auc_score(y[test_idx],rf_oof_predict[test_idx]))
print("Averaged Fold ROC score using predict function is %f"
% np.mean(ave_score))

#Create a XGB model using best parameters
xgb_Kfold=KFold(n_splits=5)

xgb=XGBClassifier(subsample= 0.6, min_child_weight= 10,
max_depth= 5, gamma= 1.5, colsample_bytree= 1.0)

# using Kfold calculate the ROC score for each fold
xgb_oof_predict_proba=np.zeros(X.shape[0])
xgb_oof_predict=np.zeros(X.shape[0])
ave_score_xgb=[]
for train_idx, test_idx in xgb_Kfold.split(X):
```

```
    xgb.fit(X[train_idx],y[train_idx])

xgb_oof_predict_proba[test_idx]=xgb.predict_proba(X[test_idx
])[:,1]
    xgb_oof_predict[test_idx]=xgb.predict(X[test_idx])

ave_score_xgb.append(roc_auc_score(y[test_idx],xgb_oof_predi
ct[test_idx]))
#     print("Fold ROC score is %f" %
roc_auc_score(y[test_idx],xgb_oof_predict_proba[test_idx]))
#     print("Fold ROC score is %f" %
roc_auc_score(y[test_idx],xgb_oof_predict[test_idx]))
print("Averaged Fold ROC score using predict function is %f"
% np.mean(ave_score_xgb))


#Prepare data for logisitc regression that contains
probability from RF and SGB Stage 1
stage2_data=pd.DataFrame({'RF_data':rf_oof_predict_proba,'XG
B_Data':xgb_oof_predict_proba})
stage2_data.head()

# create logisitic  Regresssion model
stage2=LogisticRegression(random_state=42)
stage2.fit(stage2_data,y)
stage2.coef_

#Fit the whole train data set using stage 1  RF and xgb
models
rf.fit(X,y)
xgb.fit(X,y)

# Next step prepare test data set to perfrom stage 2
modeling
model_data_test=test_data[['x2', 'x7', 'x12', 'x20', 'x23',
'x27', 'x28', 'x32', 'x37', 'x38','x40', 'x42', 'x46',
'x48', 'x49']]
response_data_test=test_data['y']

#Perfrom prediction for Test data using RF and XGB models
from stage 1
test_xgb_proba=xgb.predict_proba(model_data_test.values)[:,1
]
test_xgb=xgb.predict(model_data_test.values)
print("XGB auc score using probability is %f" %
roc_auc_score(response_data_test,test_xgb_proba))
print("XGB auc score using predict  is %f" %
roc_auc_score(response_data_test,test_xgb))

print('\n')
print("=== Confusion Matrix XGB Model ===")
print(confusion_matrix(response_data_test, test_xgb))
print('\n')
print("=== Classification Report  XGB Model ===")
print(classification_report(response_data_test, test_xgb))
print
("=====================================================")
print('\n')

test_rf_proba=rf.predict_proba(model_data_test.values)[:,1]
```

```
test_rf=rf.predict(model_data_test.values)
print("RF auc score using probability is %f" %
roc_auc_score(response_data_test,test_rf_proba))
print("RF auc score using predict  is %f" %
roc_auc_score(response_data_test,test_rf))

print('\n')
print("=== Confusion Matrix RF Model ===")
print(confusion_matrix(response_data_test, test_rf))
print('\n')
print("=== Classification Report  RF Model ===")
print(classification_report(response_data_test, test_rf))
print
("=======================================================")
print('\n')

#Perfrom prediction for Test data using ensemble LR model
test_s2=pd.DataFrame({'RF data':test_xgb_proba,'XGB
Data':test_rf_proba})
test_stage2=stage2.predict(test_s2)
test_stage2_proba=stage2.predict_proba(test_s2)[:,1]

print("Ensemble Stage2 auc score using probability is % f" %
roc_auc_score(response_data_test,test_stage2_proba))
print("Ensemble Stage2 auc score using predict is % f" %
roc_auc_score(response_data_test,test_stage2))

print('\n')
print("=== Confusion Matrix Ensemble Model ===")
print(confusion_matrix(response_data_test, test_stage2))
print('\n')
print("=== Classification Report Ensemble Model ===")
print(classification_report(response_data_test,
test_stage2))
print
("=======================================================")
print('\n')

#Utility finction to plot ROC curve
def plot_roc_curve(fpr, tpr, label=None):
    """
    The ROC curve, modified from
    Hands-On Machine learning with Scikit-Learn and
TensorFlow; p.91
    """
    plt.figure(figsize=(8,8))
    plt.title('ROC Curve')
    plt.plot(fpr, tpr, linewidth=2, label=label)
    plt.plot([0, 1], [0, 1], 'k--')
    plt.axis([-0.005, 1, 0, 1.005])
    plt.xticks(np.arange(0,1, 0.05), rotation=90)
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate (Recall)")
    plt.legend(loc='best')

#Utility function to calculate confusion matrix for each
threhold
def adjusted_classes(y_scores, t):
```

```
    return [1 if y >= t else 0 for y in y_scores]

def precision_recall_threshold(p, r,
thresholds,y_scores,y_test,plotyes, t=0.5):

    if plotyes==1:
        y_pred_adj = adjusted_classes(y_scores, t)
        print(pd.DataFrame(confusion_matrix(y_test,
y_pred_adj),
                           columns=['Pred_1', 'Pred_0'],
                           index=['Actual_1', 'Actual_0']))

    else:
        y_pred_adj = adjusted_classes(y_scores, t)
    return y_pred_adj

#Utility function to calculate confusion matrix for each
threhold
def threshold_calc(y_test,classifier):

    p, r, thresholds = precision_recall_curve(y_test,
classifier)

    plot_precision_recall_vs_threshold(p, r, thresholds)
    print("BaseLine confusion Matrix")
    a=precision_recall_threshold(p, r, thresholds,
classifier, y_test, 1, 0.5)
    s=confusion_matrix(y_test, a).ravel()[1]*100
    d=confusion_matrix(y_test, a).ravel()[2]*10
    f=confusion_matrix(y_test, a).ravel()[0]
    g=confusion_matrix(y_test, a).ravel()[3]
    print("------------------------")
    print("Baseline loss in dollars : ", s+d)
    print("------------------------")
    print ("The correct BaseLine classification  TP+TN is:",
f+g)
    print("------------------------")
    print("------------------------")

    basel=[]
    baseg=[]
    #threshold_list =
[0.05,0.1,0.15,0.2,0.25,0.3,0.35,0.4,0.45,0.5,0.55,0.6,0.65,
.7,.75,.8,.85,.9,.95,.99,1]
    threshold_list =np.arange(0.1, 1.05, 0.1).tolist()

    for i in threshold_list:
        a=precision_recall_threshold(p, r, thresholds,
classifier, y_test,0, i)
        s=confusion_matrix(y_test, a).ravel()[1]*100
        d=confusion_matrix(y_test, a).ravel()[2]*10
        f=confusion_matrix(y_test, a).ravel()[0]
        g=confusion_matrix(y_test, a).ravel()[3]
        sum_loss=s+d
        sum_good=f+g
        basel.append(sum_loss)
        baseg.append(sum_good)
    print("Optimized confusion Matrix")
    precision_recall_threshold(p, r,
```

```
thresholds,classifier,y_test,
1,threshold_list[basel.index(min(basel))])
    plt.figure(figsize=(6,6))
    plt.title("Precision and Recall curve ^ = current
threshold")
    plt.step(r, p, color='b', alpha=0.2,
                         where='post')
    plt.fill_between(r, p, step='post', alpha=0.2,
                                color='b')
    plt.xlim([0, 1.01])
    plt.xlabel('Threshold')
    plt.ylabel('Dollar Loss')
    plt.plot(threshold_list, basel, '^',
c='k',markersize=10)
    print("------------------------")
    print("Best Thershold to minimize the loss : ",
threshold_list[basel.index(min(basel))])
    print ("The estimated loss with new threshold :",
min(basel))
    print ("The correct at best threshold classification
TP+TN is :",baseg[basel.index(min(basel))])
    print("------------------------")
########################################3
    plt.figure(figsize=(6,6))
    plt.title("Precision and Recall curve ^ = current
threshold")
    plt.step(r, p, color='b', alpha=0.2,
                         where='post')
    plt.fill_between(r, p, step='post', alpha=0.2,
                                color='b')
    plt.xlim([0, 1.01])
    plt.xlabel('Threshold')
    plt.ylabel('Correct Classification Amount TP+TN')
    plt.plot(threshold_list, baseg, '^',
c='k',markersize=10)
    print("------------------------")
    print("Best Thershold for correct classification TP+TN
is : ", threshold_list[baseg.index(max(baseg))])
    print("------------------------")

#Utility function to plot example Precison and recall
dependency
def plot_precision_recall_vs_threshold(precisions, recalls,
thresholds):
    plt.figure(figsize=(8, 8))
    plt.title("Example Precision and Recall Scores as a
function of the decision threshold")
    plt.plot(thresholds, precisions[:-1], "b--",
label="Precision")
    plt.plot(thresholds, recalls[:-1], "g-", label="Recall")
    plt.ylabel("Score")
    plt.xlabel("Decision Threshold")
    plt.legend(loc='best')

# Print the confusiion matrix with new threshold
test_stage2_adj=adjusted_classes(test_stage2_proba, 0.9)

print("=== Confusion Matrix ===")
print(confusion_matrix(response_data_test, test_stage2_adj))
```

```
print('\n')
print("=== Classification Report ===")
print(classification_report(response_data_test,
test_stage2_adj))
print('\n')
print("Ensemble model auc Score with default threshold using
predict is %f" %
roc_auc_score(response_data_test,test_stage2))
print("Ensemble model auc Score with optimized threshold
using predict is %f" %
roc_auc_score(response_data_test,test_stage2_adj))
print('\n')
```