Customer Churn Prediction

"A Bank wants to take care of it’s customer retention for its product." The bank wants to identify customers likely to churn balances below the minimum balance.

Data :

The dataset which can be cleanly divided in 3 categories:

Demographic information about customers: customer_id

vintage

age

gender

dependents

occupation

city

Customer Bank Relationship: customer_nw_category

branch_code

days_since_last_transaction

Transactional Information : current_balance

previous_month_end_balance

average_monthly_balance_prevQ

average_monthly_balance_prevQ2

current_month_credit

previous_month_credit

current_month_debit

previous_month_debit

current_month_balance

previous_month_balance

churn - Average balance of customer falls below minimum balance in the next quarter (1/0)

Load Packages

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
#import random as rd
#from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression as LG
from sklearn.metrics import f1_score
from sklearn.model_selection import KFold, StratifiedKFold, train_test_split
from sklearn.metrics import roc_auc_score, accuracy_score, confusion_matrix, roc_curve, preci
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
warnings.simplefilter(action='ignore', category=UserWarning)


#Load Data
df=pd.read_csv('churn_prediction.csv')


df.shape
```

```
(28382, 21)
```

```
df.isnull().sum()
```

```
customer_id                       0
vintage                           0
age                               0
gender                          525
dependents                     2463
occupation                       80
city                            803
customer_nw_category              0
branch_code                       0
days_since_last_transaction    3223
current_balance                   0
previous_month_end_balance        0
average_monthly_balance_prevQ     0
average_monthly_balance_prevQ2    0
current_month_credit              0
previous_month_credit             0
current_month_debit               0
previous_month_debit              0
current_month_balance             0
previous_month_balance            0
churn                             0
dtype: int64
```

Missing Vaue Imputation

```
df['gender'].fillna(value=df['gender'].mode()[0],inplace=True)
```

```
df=df.astype({"customer_nw_category" : 'object',"churn" : 'object'})
```

```
df['occupation'].fillna(value=df['occupation'].mode()[0],inplace=True)
```

```
df['dependents'].fillna(value=df['dependents'].mode()[0],inplace=True)
```

```
df['city'].fillna(value=df['city'].mean(),inplace=True)
```

```
df['days_since_last_transaction'].fillna(value=df['days_since_last_transaction'].mean(),inpla
```

```
df.shape
```

```
(28382, 21)
```

```
df['gender'].value_counts()
```

```
Male      17073
Female    11309
Name: gender, dtype: int64
```

```
df['occupation'].value_counts()
```

```
self_employed    17556
salaried          6704
student           2058
retired           2024
company             40
Name: occupation, dtype: int64
```

Preprocessing

```
#Convert Gender
dict_gender = {'Male': 1, 'Female':0}
df.replace({'gender': dict_gender}, inplace = True)

df['gender'] = df['gender'].fillna(-1)
```

```
df.shape
```

```
(28382, 21)
```

```
df.isnull().sum()
```

```
customer_id                      0
vintage                          0
age                              0
gender                           0
dependents                       0
occupation                       0
city                             0
customer_nw_category             0
branch_code                      0
days_since_last_transaction      0
current_balance                  0
previous_month_end_balance       0
average_monthly_balance_prevQ    0
average_monthly_balance_prevQ2   0
current_month_credit             0
previous_month_credit            0
current_month_debit              0
previous_month_debit             0
current_month_balance            0
previous_month_balance           0
churn                            0
dtype: int64
```

```
data=pd.get_dummies(df['occupation'])
```

```
df=pd.concat([data,df],axis=1)
```

```
df.drop(['occupation'],axis=1,inplace=True)
```

```
df.head(30)
```

	company	retired	salaried	self_employed	student	customer_id	vintage	age	gende
0	0	0	0	1	0	1	3135	66	
1	0	0	0	1	0	2	310	35	
2	0	0	1	0	0	4	2356	31	
3	0	0	0	1	0	5	478	90	
4	0	0	0	1	0	6	2531	42	
5	0	0	0	1	0	7	263	42	
6	0	1	0	0	0	8	5922	72	
7	0	0	0	1	0	9	1145	46	
8	0	0	1	0	0	10	2132	31	
9	0	0	0	1	0	11	3379	40	
10	0	1	0	0	0	12	661	68	
11	0	0	1	0	0	13	7108	32	
12	0	1	0	0	0	14	2438	73	
13	0	0	1	0	0	15	5703	50	
14	0	0	0	1	0	16	2314	48	
15	0	0	0	1	0	17	1934	51	
16	0	0	0	1	0	19	2723	49	
17	0	0	0	1	0	20	6111	52	
18	0	0	0	1	0	21	5821	47	

Normalize

```
num_cols = ['customer_nw_category', 'current_balance',
            'previous_month_end_balance', 'average_monthly_balance_prevQ2', 'average_monthly_
            'current_month_credit','previous_month_credit', 'current_month_debit',
            'previous_month_debit','current_month_balance', 'previous_month_balance']

#for i in num_cols:
 #   df[i] = np.log(df[i] + 17000)

std = StandardScaler()
scaled = std.fit_transform(df[num_cols])
scaled = pd.DataFrame(scaled,columns=num_cols)
```

```
df_df_og = df.copy()
df = df.drop(['customer_nw_category', 'current_balance',
            'previous_month_end_balance', 'average_monthly_balance_prevQ2', 'average_monthly_
            'current_month_credit','previous_month_credit', 'current_month_debit',
            'previous_month_debit','current_month_balance', 'previous_month_balance'],axis =
df = df.merge(scaled,left_index=True,right_index=True,how = "left")


X=df.drop(['churn', 'customer_id'],axis=1)
y=df['churn']
```

Baseline Column

```
baseline_cols = ['current_month_debit', 'previous_month_debit','current_balance','previous_mo
                ,'retired', 'salaried','self_employed', 'student']


df_baseline = df[baseline_cols]
```

Train_Test_Split

```
train_X,test_X,train_y,test_y= train_test_split(df_baseline,y,random_state=56,test_size=0.2,s


train_X.shape
```

```
(22705, 9)
```

```
test_X.shape
```

```
(5677, 9)
```

```
train_y.shape
```

```
(22705,)
```

```
test_y.shape
```

```
(5677,)
```

Logistic Regression

```
lg=LG()


train_y.shape
```

```
(22705,)
```

```
train_y.head(20)
```

```
5102     0
1428     0
12337    0
9826     0
20876    0
10564    0
3659     0
27226    0
23398    0
558      0
3260     0
14400    1
16939    0
13932    0
12210    0
27610    1
5995     0
6        0
8519     0
26727    0
Name: churn, dtype: object
```

```
type(train_X)
```

```
pandas.core.frame.DataFrame
```

```
type(train_y)
```

```
pandas.core.series.Series
```

```
train_y=list(train_y)
```

```
lg.fit(train_X,train_y)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

```
pred = lg.predict_proba(test_X)[:,1]
```

f1 score calculation

```
train_predict=lg.predict(train_X)
```

```
train_predict
```

```
array([0, 0, 0, ..., 0, 0, 0])
```

```
k=f1_score(train_predict,train_y)
```

```
print('training f1_score',k)
```

```
training f1_score 0.012192262602579131
```

```
test_predict=lg.predict(test_X)
```

```
test_predict
```

```
array([0, 0, 0, ..., 0, 0, 0])
```

```
type(test_predict)
```

```
numpy.ndarray
```

```
type(test_y)
```

```
pandas.core.series.Series
```

```
test_y=list(test_y)
```

```
k=f1_score(test_predict,test_y)
```

```
print('test_score',k)   #  checking model success
```

```
test_score 0.018691588785046728
```

```
df.head()
```

	company	retired	salaried	self_employed	student	customer_id	vintage	age	gender
1	0	0	0	1	0	2	310	35	1

cross validation

```
def cv_score(ml_model, rstate = 12, thres = 0.5, cols = df.columns):
    i = 1
    cv_scores = []
    df1 = df.copy()
    # print(df1[['current_month_debit']].head())
    df1 = df[cols]
    print(df1.head())
    y1 = list(y)

     # 5 Fold cross validation stratified on the basis of target
    kf= StratifiedKFold(n_splits=3,random_state=rstate,shuffle=True)
    for df_index,test_index in kf.split(df1,y1):
        print('\n{} of kfold {}'.format(i,kf.n_splits))
        print("{} {}".format(df_index, test_index))
        xtr,xvl = df1.loc[df_index],df1.loc[test_index]
        # ytr,yvl = y1[df_index],y1[test_index]
        ytr = [y1[idx] for idx in df_index]
        yvl = [y1[idx] for idx in test_index]

        # Define model for fitting on the training set for each fold
        model = ml_model
        model.fit(xtr, ytr)
        pred_probs = model.predict_proba(xvl)
        pp = []

        # Use threshold to define the classes based on probability values
        for j in pred_probs[:,1]:
            if j>thres:
                pp.append(1)
            else:
                pp.append(0)

        # Calculate scores for each fold and print
        pred_val = pp
        roc_score = roc_auc_score(yvl,pred_probs[:,1])
        recall = recall_score(yvl,pred_val)
        precision = precision_score(yvl,pred_val)
        sufix = ""
        msg = ""
        msg += "ROC AUC Score: {}, Recall Score: {:.4f}, Precision Score: {:.4f} ".format(roc
        print("{}".format(msg))

         # Save scores
        cv_scores.append(roc_score)
        i+=1
    return cv_scores
```

```
baseline_scores = cv_score(lg, cols=baseline_cols)
```

```
   current_month_debit  previous_month_debit  ...  self_employed  student
0                 0.20                  0.20  ...              1        0
1              5486.27                100.56  ...              1        0
2              6046.73                259.23  ...              0        0
3                 0.47               2143.33  ...              1        0
4               588.62               1538.06  ...              1        0

[5 rows x 9 columns]

1 of kfold 3
[    1     2     3 ... 28374 28377 28380] [    0     4    11 ... 28378 28379 28381]
ROC AUC Score: 0.655550600334929, Recall Score: 0.0548, Precision Score: 0.7619

2 of kfold 3
[    0     3     4 ... 28378 28379 28381] [    1     2     5 ... 28371 28373 28380]
ROC AUC Score: 0.6490887609910225, Recall Score: 0.0849, Precision Score: 0.6804

3 of kfold 3
[    0     1     2 ... 28379 28380 28381] [    3     6    10 ... 28372 28374 28377]
ROC AUC Score: 0.6482922637727713, Recall Score: 0.0736, Precision Score: 0.7167
```

```
all_feat_scores = cv_score(LG())
```

```
   company  retired  ...  previous_month_balance  churn
0        0        0  ...                 1458.71      0
1        0        0  ...                 8787.61      0
2        0        0  ...                 5070.14      0
3        0        0  ...                 1669.79      1
4        0        0  ...                 1677.16      1

[5 rows x 25 columns]

1 of kfold 3
[    1     2     3 ... 28374 28377 28380] [    0     4    11 ... 28378 28379 28381]
ROC AUC Score: 0.7259839385724998, Recall Score: 0.0998, Precision Score: 0.7883

2 of kfold 3
[    0     3     4 ... 28378 28379 28381] [    1     2     5 ... 28371 28373 28380]
ROC AUC Score: 0.7143875039040313, Recall Score: 0.0878, Precision Score: 0.7000

3 of kfold 3
[    0     1     2 ... 28379 28380 28381] [    3     6    10 ... 28372 28374 28377]
ROC AUC Score: 0.7111984563562318, Recall Score: 0.0890, Precision Score: 0.7156
```

Random Forest

```
from sklearn.ensemble import RandomForestClassifier
```

```
rf_all_features = cv_score(RandomForestClassifier(n_estimators=10, max_depth=3))
```

```
rf_all_features = cv_score(RandomForestClassifier(n_estimators=10, max_depth=5))
```

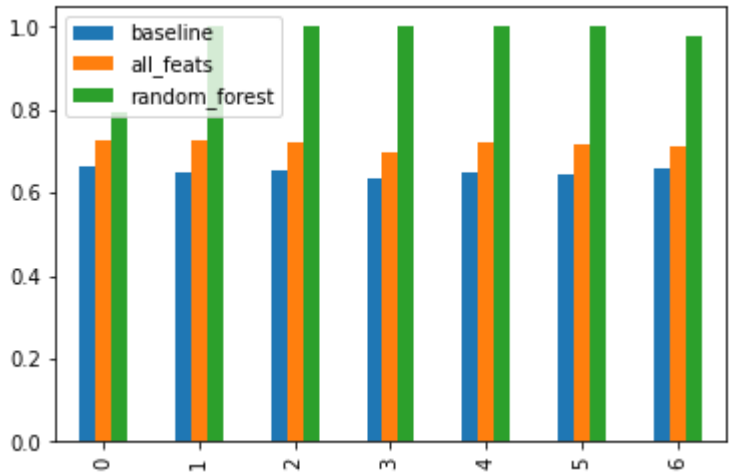
```
   company  retired  ...  previous_month_balance  churn
0        0        0  ...                 1458.71      0
1        0        0  ...                 8787.61      0
2        0        0  ...                 5070.14      0
3        0        0  ...                 1669.79      1
4        0        0  ...                 1677.16      1

[5 rows x 25 columns]

1 of kfold 3
[    1     2     3 ... 28374 28377 28380] [    0     4    11 ... 28378 28379 28381]
ROC AUC Score: 0.9844572178289662, Recall Score: 0.4238, Precision Score: 0.9893

2 of kfold 3
[    0     3     4 ... 28378 28379 28381] [    1     2     5 ... 28371 28373 28380]
ROC AUC Score: 0.9985634052414849, Recall Score: 0.6613, Precision Score: 1.0000

3 of kfold 3
[    0     1     2 ... 28379 28380 28381] [    3     6    10 ... 28372 28374 28377]
ROC AUC Score: 0.9942558202139674, Recall Score: 0.5191, Precision Score: 0.9956
```

Comparison of Different model fold wise

```
results_df = pd.DataFrame({'baseline':baseline_scores, 'all_feats': all_feat_scores, 'random_
```

```
results_df.plot(y=["baseline", "all_feats", "random_forest"], kind="bar")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f598e8f2978>
```



Life Cycle of Project

Loading Packages and Data

Missing Value Imputation using mode value

Preprocessing of data (dummy with mutiple categories to keep data stricty numeric)

Normalize data using standardscalar to avoid outilers

Model building (logistic Regression and Random Forest)

Evauation metrics (f1_score - weighted average of precision and recall and roc_auc_score for logistic Regression)

Metrics roc_auc_score for Random Forest

Conclusion

Here, we can see that the random forest model is giving the best result for each fold

This model is 99% in agreement with the demand of the actual business model.