## Customer Churn Prediction based on Customer Profile

*Report Written by Yuxuan Liu, Peicong He*

November 13, 2021

Table of Content

1. Introduction

2. EDA

2.1 Overview

2.2 Continuous Features

2.3 Categorical Features

3. Feature Engineering

4. Model Building

4.1 Principal Components Analysis (PCA)

4.2 Logistic Regression

4.3 Random Forest

4.4 LightGBM

4.5 Support Vector Machine (SVM)

4.6 Model Comparison

5. Conclusion

**Introduction**

This report summarizes analysis results and the statistical modeling associated with credit card customers churn study. The aim is to predict who will leave the bank’s credit card services and to decide what aspects to improve to prevent churning based on customers’ demographic and account related information. The customer churn rate reflects the quality of products and the ability to retain users. Customer churn rate is the number of customers leave divided by the total number of customers.

**Exploratory Data Analysis (EDA)**

**2.1 Overview**

The dataset consists of 10,127 customers, 20 features. The target is ‘Attrition\_Flag’ which is binomial with 0 and 1. 1 indicates churned customers. There are 13 categorical features and 6 numerical features. There are no missing values besides for some “Unknowns” and there are no duplicates. In general, this data set is clean.

**2.1 Continuous Features**  
 Findings from heatmap and histograms:

1. Attrition\_Flag is positively correlated with Month\_Inactive\_12\_mon and Contacts\_Count\_12\_mon.
2. Attrition\_Flag is negatively correlated with Total\_Trans\_Ct and Total\_Revolving\_Bal, etc. They are all potentially significant features.
3. Avg\_Open\_To\_Buy is highly correlated with Credit\_Limit. Maybe drop one of them later.
4. Credit\_Limit, Avg\_Open\_To\_Buy are highly right-skewed. Maybe apply transformation later.
5. No outliers. Data looks normal for other features.

**2.2 Categorical Features**

Findings from the above distribution histograms:

1. The target Attrition\_Flag is imbalanced.

2. There are no missing values but there are “unknown values” in Education\_Level, Marital\_Status and Income\_Category. Maybe use KNN to fill the variables.

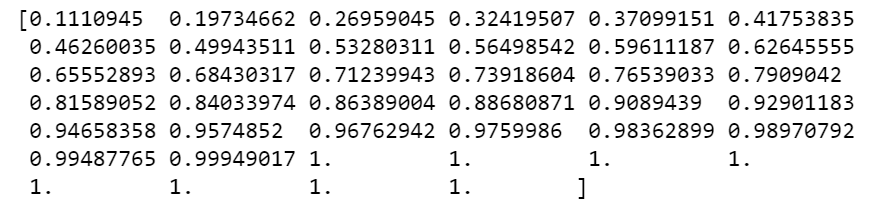
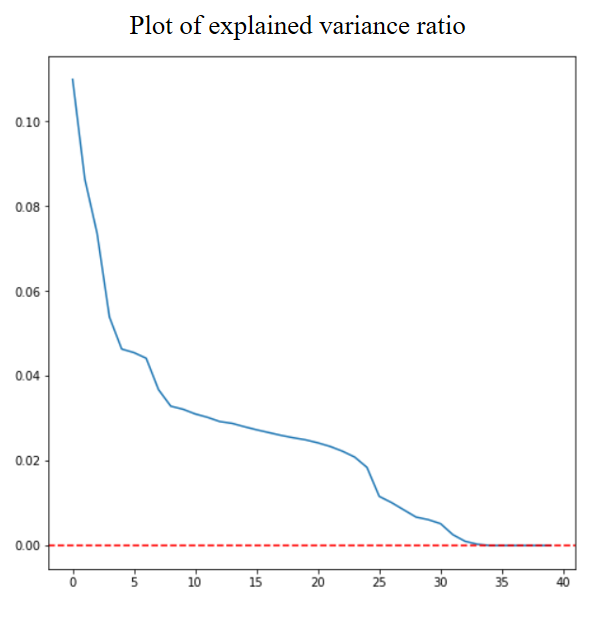
3. Churn rates are not significantly different among subcategories of Education\_Level, Marital\_Status and Income\_Category. So a KNN not necessarily improves the prediction.

**Feature engineering**

Although we spent a lot of time on feature engineering, we do not introduce in detail in this report since it is not an important topic in this course. In feature engineering, we use KNN algorithm with n\_neighbors = 5 to fill the unknowns in Education\_Level, Marital\_Status and Income\_Category. We added 6 new features based on existing features. We use Box-cox transformation to improve the shape of Total\_Trans\_Amt, Credit\_Limit, Avg\_Open\_To\_Buy. We use dummy variables to process categorical features. And finally standardize all the variables. There are 40 features after feature engineering.

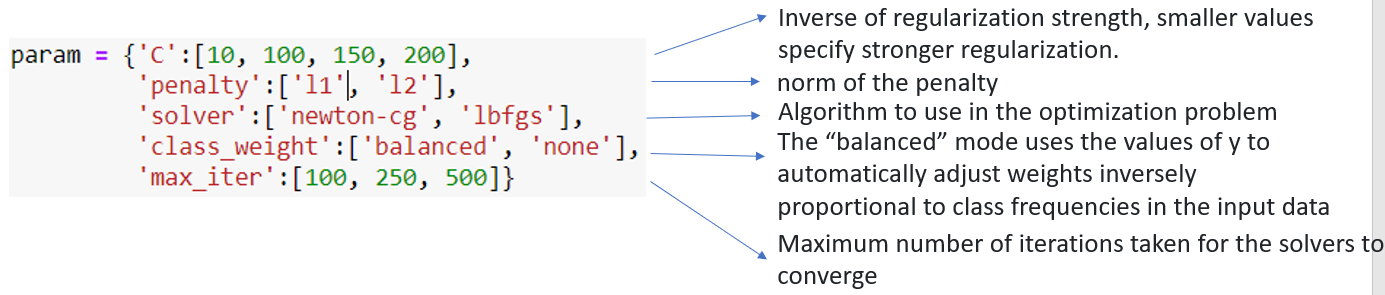
**Model Building**

**4.1 PCA**

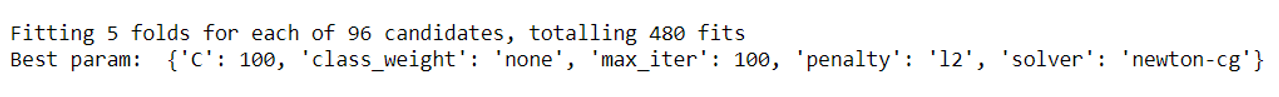


From the cumulative sum of explained variance ratio, we see that the first 32 principal components can explain 99.9% of the total variance. So we choose to use first 32 principal components as the features. However, applying PCA reduces prediction AUC slightly. So we use the original features.

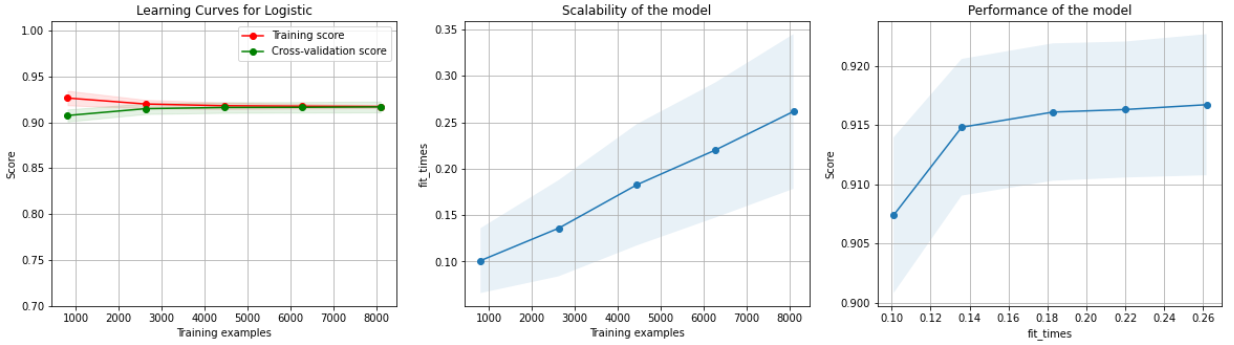
**4.2 Logistic Regression**



To choose the best parameters, we use grid search cross validation. The above parameters in the grid are the most commonly used parameters. Below are the best parameters given by the grid search.



The parameter C = 100 means the regularization strength is small. L2 norm for the penalty is better. Newton-cg algorithm is better. Class\_weight = ‘none’ means no automatic adjust in y.



We also plot the learning curve, which is used to decide how the data is trained. As training examples increases, the testing score increases and training score decreases. They almost converge at one point, which indicates a good fit.

文本

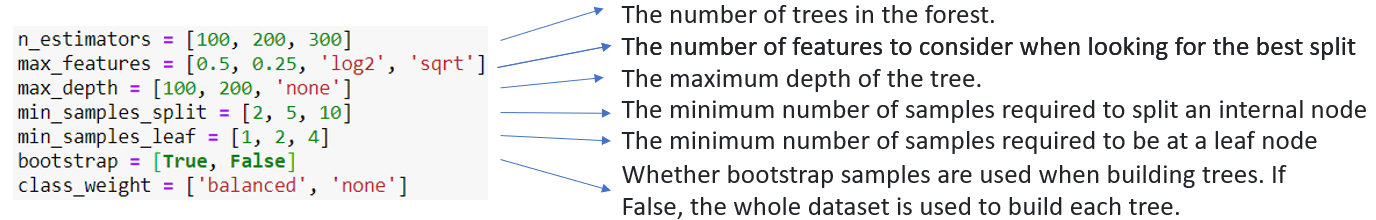
描述已自动生成

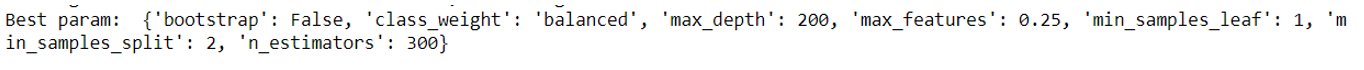
图片包含 文本

描述已自动生成

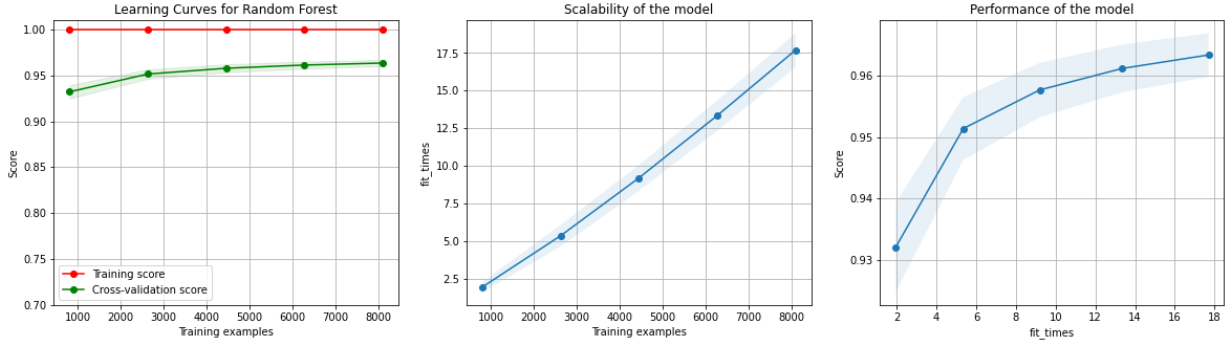
Above are the confusion matrix and the scores for prediction. In the test set, there are total 2532 customers and we correctly predict 2008+298=2306 customers. Our precision = In this customer churn prediction, we want to predict as much churned customers as possible so that we can provide better services to change their mind. It is fine if we incorrectly predict un-churned customers as churned customers. Therefore, we focus more on recall, which is the proportion of churned customers correctly predicted by us, and less on precision. Here, a 0.711 recall is not very ideal. It means about 29% of the churned customers are not predicted by us. F1 score is a relatively general score which is defined by . AUC is the area under the roc curve.

**4.3 Random forest**



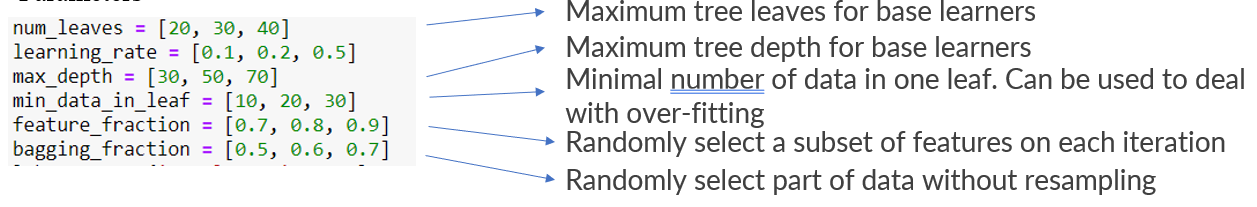


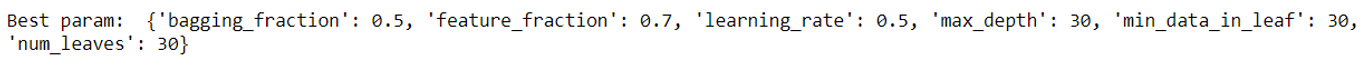
Random forest is an ensemble method consists of random decision trees. N\_estimators = 300 means there are 300 decision trees. The max depth of trees is 200. The max features is 0.25, meaning that we use 10 random features for each tree. The min\_samples\_leaf is 1, meaning that 1 is the minimum nunber of samples required for a leaf node. Min\_samples\_split is 2, meaning that 2 is the minimum number of samples required to split an internal node.



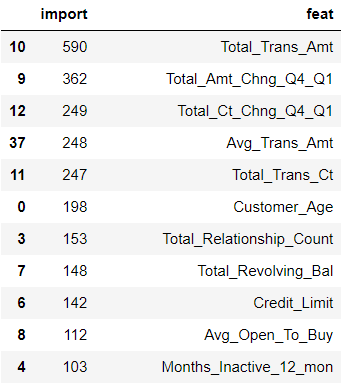
By the above learning curve, we can see that the training score is perfect and the testing score increases as training examples increases. Since training score is higher than the testing score, I think an overfitting might exist. More training examples would be helpful to improve prediction.

**4.4 LightGBM**



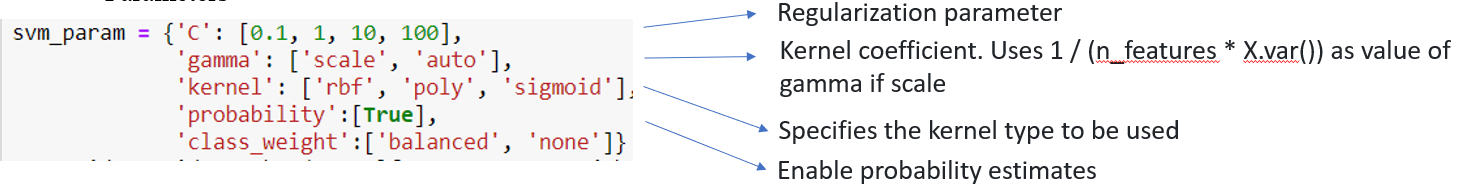


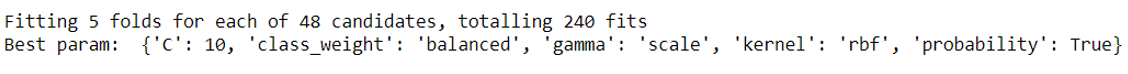
LightGBM is a gradient boosting framework that uses tree-based learning algorithms. Boosting starts with a weak classifier like decision tree and iterate until converge. The num\_leaves=30 means the base classifier has at most 30 leaves. The bagging\_fraction is 0.5, meaning that 50% of the samples are randomly selected for each iteration. The feature\_fraction is 0.7, meaning that 28 features are randomly selected for each iteration. The max\_depth is 30. The min\_data\_in\_leaf is 30, meaning that there are at least 30 data in one leaf.



Above is the feature importance from LGB. Feature importance is calculated from the features’ split and gain after split. Higher importance means more significance. We can pay more attention to the important features in the evaluation to improve model.

**4.5 Support vector machine**

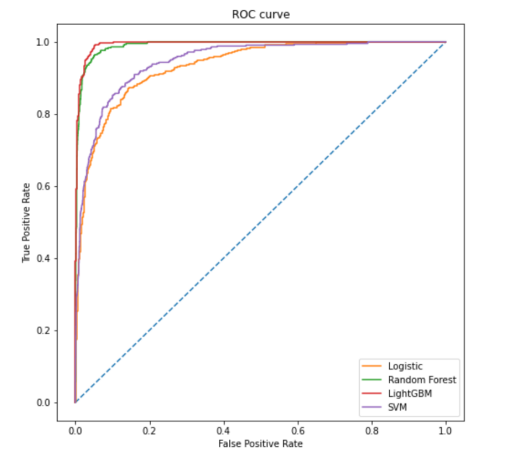




Similar to Logistic, SVM has C as regularization strength. Here C=10 indicates normal regularization. Gamma is the coefficient of kernel. Here gamma=’scale’ means to use as gamma. Kernel=’rbf’ means to use as the kernel.

**4.6 Model comparison**

表格

描述已自动生成 

Logistic model runs fast but not very accurate. LGBM and random forest perform better than Logistic since they are more complex algorithms. LGBM is faster and also more accurate than Random Forest, which is why LGBM is widely used nowadays. SVM is slightly more accurate than Logistic but too slow. To conclude, LightGBM will be the best method for this dataset.

**Conclusion**

In this project, we did EDA and preprocess of the a credit card customer churn data. Then applied Logistic regression, random forest, lightgbm and svm algorithms. The highest recall reaches 0.885, which means 88.5% of the churned customers are correctly predicted by us. By feature importance, customers churn is closely related to user transactions, including transaction amount and transaction counts. Therefore, in order to preventing our customers from leaving, we can make return offers, give gifts, grant points, etc. for them.

Liu Yuxuan is mainly responsible for feature engineering and model building.

Peicong He is mainly responsible for EDA.

Python Code:

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

from sklearn.linear\_model import LogisticRegression

from sklearn.model\_selection import train\_test\_split, GridSearchCV, RandomizedSearchCV, learning\_curve, ShuffleSplit

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import f1\_score, roc\_auc\_score, roc\_curve

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.pipeline import make\_pipeline

from pandas\_profiling import ProfileReport

from sklearn.impute import KNNImputer

from sklearn.ensemble import RandomForestClassifier

import warnings

warnings.filterwarnings("ignore")

import lightgbm as lgb

from sklearn import svm

from scipy.special import boxcox1p

from scipy.stats import boxcox\_normmax

pd.set\_option('display.max\_columns', 50)

pd.set\_option('display.max\_rows', 100)

bank\_df = pd.read\_csv("BankChurners.csv")

bank\_df

# Drop useless columns

drop\_cols = list(bank\_df.iloc[:,[0,-1,-2]].columns)

bank\_df.drop(columns=drop\_cols, inplace=True)

# Chnage response from categorical to int

bank\_df['Attrition\_Flag'] = (bank\_df['Attrition\_Flag']=='Attrited Customer').astype(int)

bank\_df.columns

# EDA

report = ProfileReport(bank\_df)

report

bank\_df.describe()

# Contacts\_Count\_12\_mon

by\_contact\_df = bank\_df.groupby('Contacts\_Count\_12\_mon')['Attrition\_Flag'].mean()

by\_contact\_df.plot(kind='bar', ylabel='Attrited Ratio')

# Months\_Inactive\_12\_mon

by\_inactive\_df = bank\_df.groupby('Months\_Inactive\_12\_mon')['Attrition\_Flag'].mean()

by\_inactive\_df.plot(kind='bar', ylabel='Attrited Ratio')

# Education level

by\_edu\_df = bank\_df.groupby('Education\_Level')['Attrition\_Flag'].mean()

by\_edu\_df.plot(kind='bar', ylabel='Attrited Ratio')

# Marital\_Status

by\_marry\_df = bank\_df.groupby('Marital\_Status')['Attrition\_Flag'].mean()

by\_marry\_df.plot(kind='bar', ylabel='Attrited Ratio')

# Income\_Category

by\_income\_df = bank\_df.groupby('Income\_Category')['Attrition\_Flag'].mean()

by\_income\_df.plot(kind='bar', ylabel='Attrited Ratio')

# Gender

by\_gender\_df = bank\_df.groupby('Gender')['Attrition\_Flag'].mean()

by\_gender\_df.plot(kind='bar', ylabel='Attrited Ratio')

# Card\_Category

by\_card\_df = bank\_df.groupby('Card\_Category')['Attrition\_Flag'].mean()

by\_card\_df.plot(kind='bar', ylabel='Attrited Ratio')

# Credit Limit

print("Avg attrition of customers with minimun credit limit: ",

bank\_df[bank\_df['Credit\_Limit']==bank\_df['Credit\_Limit'].min()]['Attrition\_Flag'].mean())

print("Avg attrition of customers with more credit limit: ",

bank\_df[bank\_df['Credit\_Limit']!=bank\_df['Credit\_Limit'].min()]['Attrition\_Flag'].mean())

# Total\_Revolving\_Bal

print("Avg attrition of customers with 0 revolving balance: ",

bank\_df[bank\_df['Total\_Revolving\_Bal']==0]['Attrition\_Flag'].mean())

print("Avg attrition of customers with more revolving balance: ",

bank\_df[bank\_df['Total\_Revolving\_Bal']!=0]['Attrition\_Flag'].mean())

# Utilization\_Ratio

print("Avg attrition of customers with 0 utilization ratio: ",

bank\_df[bank\_df['Avg\_Utilization\_Ratio']==bank\_df['Avg\_Utilization\_Ratio'].min()]['Attrition\_Flag'].mean())

print("Avg attrition of customers with more utilization ratio: ",

bank\_df[bank\_df['Avg\_Utilization\_Ratio']!=bank\_df['Avg\_Utilization\_Ratio'].min()]['Attrition\_Flag'].mean())

# Feature Engineering

# KNN for Unknown

tmp\_bank\_df = bank\_df.copy()

le\_ls = []

for col in ['Education\_Level', 'Marital\_Status', 'Income\_Category']:

le = LabelEncoder()

tmp\_bank\_df[col] = le.fit\_transform(tmp\_bank\_df[col])

keys = le.classes\_

values = le.transform(le.classes\_)

dictionary = dict(zip(keys, values))

le\_ls.append(le)

print(dictionary)

tmp\_bank\_df.loc[tmp\_bank\_df[col]==dictionary['Unknown'], col] = np.nan

tmp\_bank\_df = pd.get\_dummies(tmp\_bank\_df)

# Use KNN to fill for each column

for col in ['Marital\_Status', 'Income\_Category', 'Education\_Level']:

imputer = KNNImputer(n\_neighbors = 5)

fill\_tmp\_bank\_df = pd.DataFrame(imputer.fit\_transform(tmp\_bank\_df.iloc[:,1:]),

index=tmp\_bank\_df.index, columns=tmp\_bank\_df.columns[1:])

tmp\_bank\_df[col] = fill\_tmp\_bank\_df[col]

i = 0

for col in ['Education\_Level', 'Marital\_Status', 'Income\_Category']:

tmp\_bank\_df[col] = le\_ls[i].inverse\_transform(tmp\_bank\_df[col].astype(int))

i += 1

bank\_df = tmp\_bank\_df

# One-hot encoding

bank\_df = pd.get\_dummies(bank\_df)

# Add new features

# Avg trans amt

bank\_df['Avg\_Trans\_Amt'] = bank\_df['Total\_Trans\_Amt']/bank\_df['Total\_Trans\_Ct']

# Creat new categorical feature based on whether credit limit equals min credit limit

bank\_df.loc[bank\_df['Credit\_Limit']==bank\_df['Credit\_Limit'].min(), 'Min\_Credit\_Limit'] = 1

bank\_df['Min\_Credit\_Limit'].fillna(0, inplace=True)

# Creat new categorical feature based on whether total balance equals 0

bank\_df.loc[bank\_df['Total\_Revolving\_Bal']==0, '0\_Total\_Revolving\_Bal'] = 1

bank\_df['0\_Total\_Revolving\_Bal'].fillna(0, inplace=True)

# Creat new categorical feature based on whether total balance equals 0

bank\_df.loc[bank\_df['Avg\_Utilization\_Ratio']==0, '0\_Avg\_Utilization\_Ratio'] = 1

bank\_df['0\_Avg\_Utilization\_Ratio'].fillna(0, inplace=True)

# Determine amount decrease or increase based on Amt\_Chng\_Q4\_Q1

bank\_df.loc[bank\_df['Total\_Amt\_Chng\_Q4\_Q1']<bank\_df['Total\_Amt\_Chng\_Q4\_Q1'].median(), 'Amt\_Q4\_Q1\_Dec'] = 1

bank\_df['Amt\_Q4\_Q1\_Dec'].fillna(0, inplace=True)

# Determine count decrease or increase based on Ct\_Chng\_Q4\_Q1

bank\_df.loc[bank\_df['Total\_Ct\_Chng\_Q4\_Q1']<bank\_df['Total\_Ct\_Chng\_Q4\_Q1'].median(), 'Ct\_Q4\_Q1\_Dec'] = 1

bank\_df['Ct\_Q4\_Q1\_Dec'].fillna(0, inplace=True)

# Transformation

skewed\_col = ['Total\_Trans\_Amt', 'Credit\_Limit', 'Avg\_Open\_To\_Buy']

trans\_bank\_df = bank\_df.copy()

for col in skewed\_col:

plt.figure(figsize=(8, 4))

plt.subplot(1, 2, 1)

plt.hist(bank\_df[col])

plt.title(f"{col}: Before")

trans\_bank\_df[col] = boxcox1p(bank\_df[col], boxcox\_normmax(bank\_df[col] + 1))

plt.subplot(1, 2, 2)

plt.hist(trans\_bank\_df[col])

plt.title(f"{col}: After")

# Standardize

std = StandardScaler()

std\_df = pd.DataFrame(std.fit\_transform(trans\_bank\_df.iloc[:,1:]),

index = trans\_bank\_df.index,

columns = trans\_bank\_df.columns[1:])

std\_bank\_df = pd.concat([trans\_bank\_df.iloc[:,0], std\_df], axis=1)

# Logistic Regression

# train test split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(std\_bank\_df.iloc[:,1:], std\_bank\_df.iloc[:,0], test\_size=0.25, random\_state=42)

# PCA

pca = PCA(n\_components=40) # all 40 features

pca.fit(X\_train)

prop = pca.explained\_variance\_ratio\_

print(prop.cumsum())

plt.figure(figsize=(8,8))

plt.plot(prop)

plt.axhline(y=0, color='red', linestyle='--')

pca = PCA(n\_components=32) # Use the first 32 pcs

pca.fit(X\_train)

X\_train\_pc = pca.transform(X\_train)

X\_test\_pc = pca.transform(X\_test)

param = {'C':[10, 100, 150, 200],

'penalty':['l1', 'l2'],

'solver':['newton-cg', 'lbfgs'],

'class\_weight':['balanced', 'none'],

'max\_iter':[100, 250, 500]}

lr\_grid = GridSearchCV(LogisticRegression(random\_state=42), param, cv=5, verbose=1, n\_jobs=-1)

lr\_grid.fit(X\_train, y\_train)

print("Best param: ", lr\_grid.best\_params\_)

lr\_pred = lr\_grid.predict(X\_test)

lr\_pred\_prob = lr\_grid.predict\_proba(X\_test)[:,1]

lr\_pred\_df = pd.DataFrame({"pred":lr\_pred, "prob":lr\_pred\_prob, "actual":y\_test})

# mannually adjust pred based on prob

thresh = np.quantile(lr\_pred\_prob, (1-y\_train.mean()))

lr\_pred\_df['pred'] = (lr\_pred\_df['prob']>thresh).astype(int)

lr\_pred\_df

# Reference: https://scikit-learn.org/stable/auto\_examples/model\_selection/plot\_learning\_curve.html

def plot\_learning\_curve(

estimator,

title,

X,

y,

axes=None,

ylim=None,

cv=None,

n\_jobs=None,

train\_sizes=np.linspace(0.1, 1.0, 5),

):

"""

Generate 3 plots: the test and training learning curve, the training

samples vs fit times curve, the fit times vs score curve.

Parameters

----------

estimator : estimator instance

An estimator instance implementing `fit` and `predict` methods which

will be cloned for each validation.

title : str

Title for the chart.

X : array-like of shape (n\_samples, n\_features)

Training vector, where ``n\_samples`` is the number of samples and

``n\_features`` is the number of features.

y : array-like of shape (n\_samples) or (n\_samples, n\_features)

Target relative to ``X`` for classification or regression;

None for unsupervised learning.

axes : array-like of shape (3,), default=None

Axes to use for plotting the curves.

ylim : tuple of shape (2,), default=None

Defines minimum and maximum y-values plotted, e.g. (ymin, ymax).

cv : int, cross-validation generator or an iterable, default=None

Determines the cross-validation splitting strategy.

Possible inputs for cv are:

- None, to use the default 5-fold cross-validation,

- integer, to specify the number of folds.

- :term:`CV splitter`,

- An iterable yielding (train, test) splits as arrays of indices.

For integer/None inputs, if ``y`` is binary or multiclass,

:class:`StratifiedKFold` used. If the estimator is not a classifier

or if ``y`` is neither binary nor multiclass, :class:`KFold` is used.

Refer :ref:`User Guide <cross\_validation>` for the various

cross-validators that can be used here.

n\_jobs : int or None, default=None

Number of jobs to run in parallel.

``None`` means 1 unless in a :obj:`joblib.parallel\_backend` context.

``-1`` means using all processors. See :term:`Glossary <n\_jobs>`

for more details.

train\_sizes : array-like of shape (n\_ticks,)

Relative or absolute numbers of training examples that will be used to

generate the learning curve. If the ``dtype`` is float, it is regarded

as a fraction of the maximum size of the training set (that is

determined by the selected validation method), i.e. it has to be within

(0, 1]. Otherwise it is interpreted as absolute sizes of the training

sets. Note that for classification the number of samples usually have

to be big enough to contain at least one sample from each class.

(default: np.linspace(0.1, 1.0, 5))

"""

if axes is None:

\_, axes = plt.subplots(1, 3, figsize=(20, 5))

axes[0].set\_title(title)

if ylim is not None:

axes[0].set\_ylim(\*ylim)

axes[0].set\_xlabel("Training examples")

axes[0].set\_ylabel("Score")

train\_sizes, train\_scores, test\_scores, fit\_times, \_ = learning\_curve(

estimator,

X,

y,

cv=cv,

n\_jobs=n\_jobs,

train\_sizes=train\_sizes,

return\_times=True,

)

train\_scores\_mean = np.mean(train\_scores, axis=1)

train\_scores\_std = np.std(train\_scores, axis=1)

test\_scores\_mean = np.mean(test\_scores, axis=1)

test\_scores\_std = np.std(test\_scores, axis=1)

fit\_times\_mean = np.mean(fit\_times, axis=1)

fit\_times\_std = np.std(fit\_times, axis=1)

# Plot learning curve

axes[0].grid()

axes[0].fill\_between(

train\_sizes,

train\_scores\_mean - train\_scores\_std,

train\_scores\_mean + train\_scores\_std,

alpha=0.1,

color="r",

)

axes[0].fill\_between(

train\_sizes,

test\_scores\_mean - test\_scores\_std,

test\_scores\_mean + test\_scores\_std,

alpha=0.1,

color="g",

)

axes[0].plot(

train\_sizes, train\_scores\_mean, "o-", color="r", label="Training score"

)

axes[0].plot(

train\_sizes, test\_scores\_mean, "o-", color="g", label="Cross-validation score"

)

axes[0].legend(loc="best")

# Plot n\_samples vs fit\_times

axes[1].grid()

axes[1].plot(train\_sizes, fit\_times\_mean, "o-")

axes[1].fill\_between(

train\_sizes,

fit\_times\_mean - fit\_times\_std,

fit\_times\_mean + fit\_times\_std,

alpha=0.1,

)

axes[1].set\_xlabel("Training examples")

axes[1].set\_ylabel("fit\_times")

axes[1].set\_title("Scalability of the model")

# Plot fit\_time vs score

axes[2].grid()

axes[2].plot(fit\_times\_mean, test\_scores\_mean, "o-")

axes[2].fill\_between(

fit\_times\_mean,

test\_scores\_mean - test\_scores\_std,

test\_scores\_mean + test\_scores\_std,

alpha=0.1,

)

axes[2].set\_xlabel("fit\_times")

axes[2].set\_ylabel("Score")

axes[2].set\_title("Performance of the model")

return plt

X, y = std\_bank\_df.iloc[:,1:], std\_bank\_df.iloc[:,0]

title = "Learning Curves for Logistic"

# Cross validation with 100 iterations to get smoother mean test and train

# score curves, each time with 20% data randomly selected as a validation set.

cv = ShuffleSplit(n\_splits=100, test\_size=0.2, random\_state=0)

estimator = LogisticRegression(C=100,

class\_weight='none',

max\_iter=100,

penalty='l2',

solver='newton-cg')

plot\_learning\_curve(

estimator, title, X, y, ylim=(0.7, 1.01), cv=cv, n\_jobs=4

)

plt.show()

# report

def get\_eval(pred\_df\_ls, model\_ls, grid\_ls):

eval\_df = pd.DataFrame()

for pred\_df, model, grid in zip(pred\_df\_ls, model\_ls, grid\_ls):

confusion = confusion\_matrix(pred\_df['pred'], pred\_df['actual'])

print(f'Confusion matrix for {model}: \n',

pd.DataFrame(confusion, index=['Pred Neg', 'Pred Pos'], columns=['Actual Neg', 'Actual Pos']))

recall = pred\_df[pred\_df['actual']==1]['pred'].mean()

accuracy = (confusion[0,0]+confusion[1,1])/confusion.sum()

precision = pred\_df[pred\_df['pred']==1]['actual'].mean()

error\_rate = 1 - accuracy

f1 = f1\_score(pred\_df['pred'], pred\_df['actual'])

auc = roc\_auc\_score(pred\_df['actual'], pred\_df['prob'])

run\_time = grid.cv\_results\_['mean\_fit\_time'].mean()

eval\_df = pd.concat([eval\_df, pd.DataFrame({"Model": [model], "Recall":[round(recall, 3)], "Accuracy":[round(accuracy, 3)], "Precision": [round(precision, 3)],

"Error Rate":[round(error\_rate, 3)], "F1 Score":[round(f1, 3)], "AUC":[round(auc, 3)], "Run time":[run\_time]})])

return eval\_df

pred\_df\_ls = []

model\_ls = []

grid\_ls = []

pred\_df\_ls.append(lr\_pred\_df)

model\_ls.append('Logistic')

grid\_ls.append(lr\_grid)

get\_eval(pred\_df\_ls, model\_ls, grid\_ls)

# roc curve

def plot\_roc(pred\_df\_ls, model\_ls):

plt.figure(figsize=(8,8))

plt.plot([0, 1], [0, 1], linestyle='--')

plt.xlabel("False Positive Rate")

plt.ylabel("True Positive Rate")

plt.title("ROC curve")

for pred\_df, model in zip(pred\_df\_ls, model\_ls):

fpr, tpr, thresholds = roc\_curve(pred\_df['actual'], pred\_df['prob'])

# plot

plt.plot(fpr, tpr, label=model)

plt.legend()

plot\_roc(pred\_df\_ls, model\_ls)

# Random Forest

rf = RandomForestClassifier()

n\_estimators = [100, 200, 300]

max\_features = [0.5, 0.25, 'log2', 'sqrt']

max\_depth = [100, 200, 'none']

min\_samples\_split = [2, 5, 10]

min\_samples\_leaf = [1, 2, 4]

bootstrap = [True, False]

class\_weight = ['balanced', 'none']

rf\_param = {'n\_estimators': n\_estimators,

'max\_features': max\_features,

'max\_depth': max\_depth,

'min\_samples\_split': min\_samples\_split,

'min\_samples\_leaf': min\_samples\_leaf,

'bootstrap': bootstrap,

'class\_weight': class\_weight}

rf\_grid = GridSearchCV(rf, rf\_param, n\_jobs=-1, cv=5, verbose=1)

rf\_grid.fit(X\_train, y\_train)

print("Best param: ", rf\_grid.best\_params\_)

rf\_pred = rf\_grid.predict(X\_test)

rf\_pred\_prob = rf\_grid.predict\_proba(X\_test)[:,1]

rf\_pred\_df = pd.DataFrame({"pred":rf\_pred, "prob":rf\_pred\_prob, "actual":y\_test})

# mannually adjust pred based on prob

thresh = np.quantile(rf\_pred\_prob, (1-y\_train.mean()))

rf\_pred\_df['pred'] = (rf\_pred\_df['prob']>thresh).astype(int)

rf\_pred\_df

X, y = std\_bank\_df.iloc[:,1:], std\_bank\_df.iloc[:,0]

title = "Learning Curves for Random Forest"

cv = ShuffleSplit(n\_splits=100, test\_size=0.2, random\_state=0)

estimator = RandomForestClassifier(bootstrap=False,

class\_weight='balanced',

max\_depth=200,

max\_features=0.25,

min\_samples\_leaf=1,

min\_samples\_split=2,

n\_estimators=300)

plot\_learning\_curve(

estimator, title, X, y, ylim=(0.7, 1.01), cv=cv, n\_jobs=4

)

plt.show()

pred\_df\_ls.append(rf\_pred\_df)

model\_ls.append('Random Forest')

grid\_ls.append(rf\_grid)

get\_eval(pred\_df\_ls, model\_ls, grid\_ls)

# roc curve

plot\_roc(pred\_df\_ls, model\_ls)

# LightGBM

lgb\_clf = lgb.LGBMClassifier()

num\_leaves = [20, 30, 40]

learning\_rate = [0.1, 0.2, 0.5]

max\_depth = [30, 50, 70]

min\_data\_in\_leaf = [10, 20, 30]

feature\_fraction = [0.7, 0.8, 0.9]

bagging\_fraction = [0.5, 0.6, 0.7]

lgb\_param = {'num\_leaves': num\_leaves,

'learning\_rate': learning\_rate,

'max\_depth': max\_depth,

'min\_data\_in\_leaf': min\_data\_in\_leaf,

'feature\_fraction': feature\_fraction,

'bagging\_fraction': bagging\_fraction}

lgb\_grid = GridSearchCV(lgb\_clf, lgb\_param, n\_jobs=-1, cv=5, verbose=1)

lgb\_grid.fit(X\_train, y\_train)

print("Best param: ", lgb\_grid.best\_params\_)

lgb\_pred = lgb\_grid.predict(X\_test)

lgb\_pred\_prob = lgb\_grid.predict\_proba(X\_test)[:,1]

lgb\_pred\_df = pd.DataFrame({"pred":lgb\_pred, "prob":lgb\_pred\_prob, "actual":y\_test})

# mannually adjust pred based on prob

thresh = np.quantile(lgb\_pred\_prob, (1-y\_train.mean()))

lgb\_pred\_df['pred'] = (lgb\_pred\_df['prob']>thresh).astype(int)

lgb\_pred\_df

X, y = std\_bank\_df.iloc[:,1:], std\_bank\_df.iloc[:,0]

title = "Learning Curves for LGBM"

cv = ShuffleSplit(n\_splits=100, test\_size=0.2, random\_state=0)

estimator = lgb.LGBMClassifier(bagging\_fraction=0.5,

class\_weight='balanced',

feature\_fraction=0.7,

learning\_rate=0.5,

max\_depth=30,

min\_data\_in\_leaf=30,

num\_leaves=30)

plot\_learning\_curve(

estimator, title, X, y, ylim=(0.7, 1.01), cv=cv, n\_jobs=4

)

plt.show()

# Evaluation

pred\_df\_ls.append(lgb\_pred\_df)

model\_ls.append('LightGBM')

grid\_ls.append(lgb\_grid)

get\_eval(pred\_df\_ls, model\_ls, grid\_ls)

# roc curve

plot\_roc(pred\_df\_ls, model\_ls)

# SVM

svm\_clf = svm.SVC()

svm\_param = {'C': [0.1, 1, 10, 100],

'gamma': ['scale', 'auto'],

'kernel': ['rbf', 'poly', 'sigmoid'],

'probability':[True],

'class\_weight':['balanced', 'none']}

svm\_grid = GridSearchCV(svm\_clf, svm\_param, n\_jobs=-1, cv=5, verbose=1)

svm\_grid.fit(X\_train, y\_train)

print("Best param: ", svm\_grid.best\_params\_)

svm\_pred = svm\_grid.predict(X\_test)

svm\_pred\_prob = svm\_grid.predict\_proba(X\_test)[:,1]

svm\_pred\_df = pd.DataFrame({"pred":svm\_pred, "prob":svm\_pred\_prob, "actual":y\_test})

# mannually adjust pred based on prob

thresh = np.quantile(svm\_pred\_prob, (1-y\_train.mean()))

svm\_pred\_df['pred'] = (svm\_pred\_df['prob']>thresh).astype(int)

svm\_pred\_df

X, y = std\_bank\_df.iloc[:,1:], std\_bank\_df.iloc[:,0]

title = "Learning Curves for SVM"

cv = ShuffleSplit(n\_splits=100, test\_size=0.2, random\_state=0)

estimator = svm.SVC(C=10,

class\_weight='balanced',

gamma='scale',

kernel='rbf',

probability=True)

plot\_learning\_curve(

estimator, title, X, y, ylim=(0.7, 1.01), cv=cv, n\_jobs=4

)

plt.show()

# Evaluation

pred\_df\_ls.append(svm\_pred\_df)

model\_ls.append('SVM')

grid\_ls.append(svm\_grid)

get\_eval(pred\_df\_ls, model\_ls, grid\_ls)

# roc curve

plot\_roc(pred\_df\_ls, model\_ls)

# Error Analysis

lgb\_clf.fit(X\_train, y\_train)

feat\_imp\_df = pd.DataFrame({'import':lgb\_clf.feature\_importances\_, 'feat':X\_train.columns})

feat\_imp\_df.sort\_values('import', ascending=False)

lgb\_pred\_df[lgb\_pred\_df['pred']!=lgb\_pred\_df['actual']]

bank\_df['Total\_Ct\_Chng\_Q4\_Q1'].median()

bank\_df.iloc[[3668,321,8254, 8209, 8900]]

bank\_df.iloc[[4889,6125,1558,6811,3130]]