## Machine Learning for Biomedical Application

Imbalanced Classification for Biomedical Application

About Dataset The imbalanced classification dataset is the mammography dataset that involves detecting breast cancer from radiological scans, specifically the presence of clusters of microcalcification that appear bright on a mammogram. This dataset was constructed by scanning the images, segmenting them into candidate objects, and using computer vision techniques to describe each candidate object.

Features (description from the woods' paper)

1. Area of object (in pixels).

* Number of pixels in object.

1. Average gray level of the object.

* Absolute values of calcification pixels are averaged.

1. Gradient strength of the object's perimeter pixels.

* Mean value of Robert's gradient for each perimeter pixel. To perform edge detection with the Roberts Cross operator, the original image is convolved with two 2x2 kernels
* and
* After the convolution, the gradient magnitude can be obtained. The gradient points in the direction of the most rapid change in intensity, and the gradient direction is perpendicular to the edge.

1. Root mean square noise fluctuation in the object.

* In the context of images, noise can be caused by various factors, including sensor imperfections, environmental conditions, and quantization errors during image processing. RMS noise is calculated as the square root of the average of the squares of the individual noise values. (artifact,random dark and bright pixels)

1. Contrast

* average gray level of the object minus the average of a two-pixel wide border surrounding the object.

1. A low order moment based on shape descriptor.

* Image moments are weighted averages of the image pixels' intensities, and they can be used to describe various properties of an object, such as area, centroid, and orientation. Shape descriptor: Calcification aspect ratio defined as maximum radius divided by minimum radius.

There are two classes and the goal is to distinguish between Microcalcification and non-Microcalcification using the features for a given segmented object.

Non-Microcalcification: negative case, or majority class. Microcalcification: positive case, or minority class.

<https://www.kaggle.com/datasets/sudhanshu2198/microcalcification-classification>

import pandas as pd  
import numpy as np

## Data exploration

df = pd.read\_csv('microcalcification.csv')  
df.head()

Area Grey Level Gradient Strength Noise Fluctuation Contrast \  
0 0.230020 5.072578 -0.276061 0.832444 -0.377866   
1 0.155491 -0.169390 0.670652 -0.859553 -0.377866   
2 -0.784415 -0.443654 5.674705 -0.859553 -0.377866   
3 0.546088 0.131415 -0.456387 -0.859553 -0.377866   
4 -0.102987 -0.394994 -0.140816 0.979703 -0.377866   
  
 Shape Descriptor Microcalcification   
0 0.480322 '-1'   
1 -0.945723 '-1'   
2 -0.945723 '-1'   
3 -0.945723 '-1'   
4 1.013566 '-1'

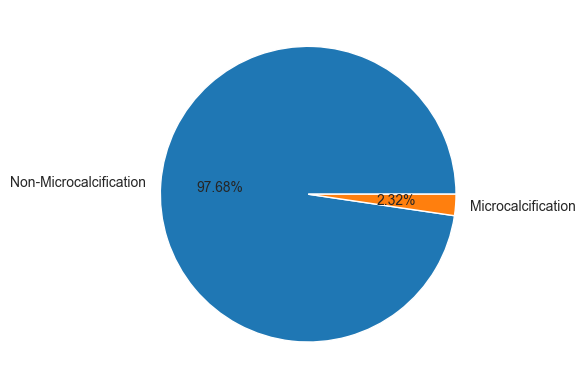
df.info()

<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 11183 entries, 0 to 11182  
Data columns (total 7 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 Area 11183 non-null float64  
 1 Grey Level 11183 non-null float64  
 2 Gradient Strength 11183 non-null float64  
 3 Noise Fluctuation 11183 non-null float64  
 4 Contrast 11183 non-null float64  
 5 Shape Descriptor 11183 non-null float64  
 6 Microcalcification 11183 non-null object   
dtypes: float64(6), object(1)  
memory usage: 611.7+ KB

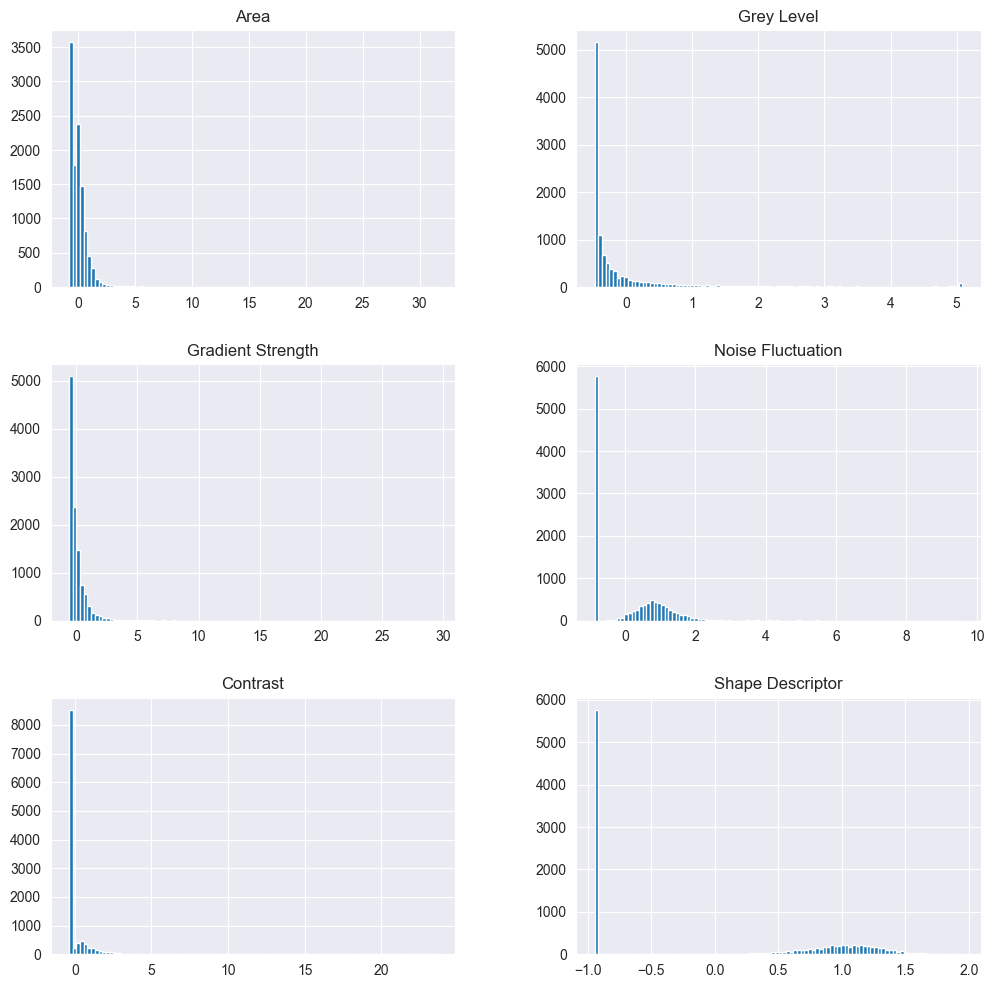
from sklearn.preprocessing import LabelEncoder  
# Use label encoder function to re-label  
# use 0 to represent the majority class and 1 to represent the minority class  
df = pd.read\_csv('microcalcification.csv')  
label\_encoder = LabelEncoder()  
df['Microcalcification'] = label\_encoder.fit\_transform(df['Microcalcification'])

from matplotlib import pyplot as plt  
%matplotlib inline  
from collections import Counter  
  
print(dict(Counter(df['Microcalcification'])))  
print('Class Ratio')  
print(df['Microcalcification'].value\_counts()/len(df))  
plt.pie(df['Microcalcification'].value\_counts(),labels=['Non-Microcalcification','Microcalcification'],autopct='%1.2f%%'); # use semicolon to hide the output text

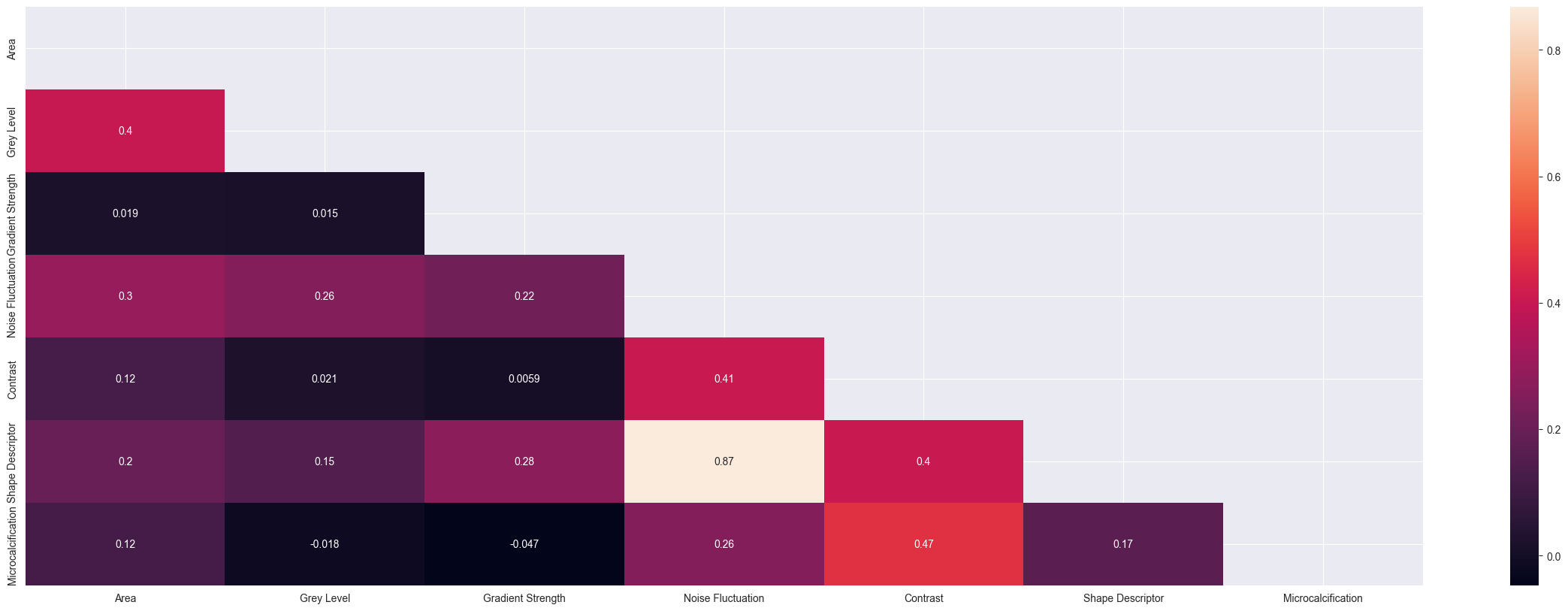
{0: 10923, 1: 260}  
Class Ratio  
0 0.97675  
1 0.02325  
Name: Microcalcification, dtype: float64



# use histogram to show distribution  
# two classes are visually well seperated  
df.drop(columns='Microcalcification').hist(figsize=(12,12),bins=100);



# Correlation matrix  
import seaborn as sns  
# correlation matrix  
cor = df.corr()  
  
# plot heatmap  
plt.figure(figsize=(30,10))  
mask=np.triu(np.ones\_like(df.corr()))  
# .triu() returns the upper triangle of an array  
# ones\_like() returns an array of ones with the same shape and type as a given array  
sns.heatmap(cor, annot=True, mask=mask);



## Preprocessing

Normalize dataset to range[0,1], because the negative values may affect classifications in some cases Power transformation can make the distribution gaussian like

X = df.drop(columns='Microcalcification')  
y = df['Microcalcification']

from sklearn.model\_selection import train\_test\_split  
# 80:20  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, stratify=y,random\_state=42)

## Model Evaluation

shuffling need to be considered

from sklearn.model\_selection import StratifiedKFold, cross\_val\_score  
  
  
def evaluate(X, y, model):  
 cv = StratifiedKFold(n\_splits=5, shuffle=True, random\_state=1)  
 auc = cross\_val\_score(model, X, y, cv=cv, scoring='roc\_auc', n\_jobs=-1)  
 recall = cross\_val\_score(model, X, y, cv=cv, scoring='recall', n\_jobs=-1)  
 precision = cross\_val\_score(model, X, y, cv=cv, scoring='precision', n\_jobs=-1)  
 return auc, recall, precision  
# 5 fold cross validation

## Baseline Performance

from sklearn.dummy import DummyClassifier  
  
model=DummyClassifier(strategy='stratified')  
# DummyClassifier makes predictions that ignore the input features.  
# This classifier serves as a simple baseline to compare against other more complex classifiers.  
auc, recall, precision= evaluate(X\_train,y\_train,model)  
   
print(f'\nAuc score for Dummy Classifier: {auc.mean()}({auc.std()})')  
print(f'\nRecall score for Dummy Classifier: {recall.mean()}({recall.std()})')  
print(f'\nPrecision score for Dummy Classifier: {precision.mean()}({precision.std()})')

Auc score for Dummy Classifier: 0.49790897773856085(0.009818773975862012)  
  
Recall score for Dummy Classifier: 0.02868757259001161(0.023305911509673368)  
  
Precision score for Dummy Classifier: 0.038128632125093195(0.015947307497153194)

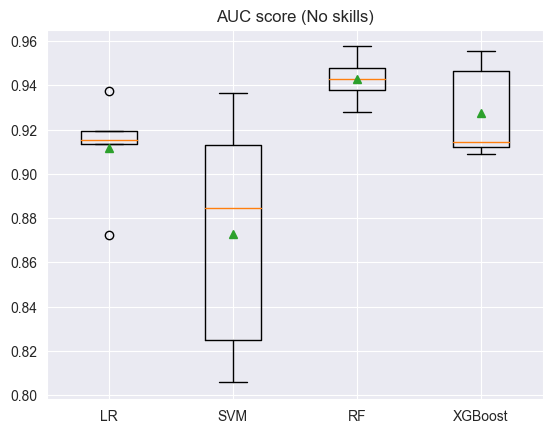
## Algorithm Spot Checking

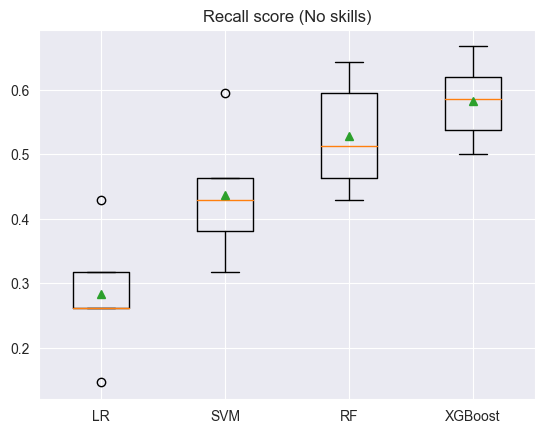
from sklearn.ensemble import RandomForestClassifier  
from sklearn.svm import SVC  
from sklearn.linear\_model import LogisticRegression  
import xgboost as xgb  
  
def get\_models():  
 models,names=list(),list()  
   
 models.append(LogisticRegression())  
 names.append('LR')  
   
 models.append(SVC(probability=True))  
 names.append('SVM')  
   
 models.append(RandomForestClassifier(n\_estimators=1000))  
 names.append('RF')  
 # It is easy to overfit with too few trees, adding more trees can reduce the overfitting, but there is no way to eliminate it completely  
 # https://escholarship.org/content/qt35x3v9t4/qt35x3v9t4.pdf  
   
 models.append(xgb.XGBClassifier())  
 names.append('XGBoost')  
   
 return models,names

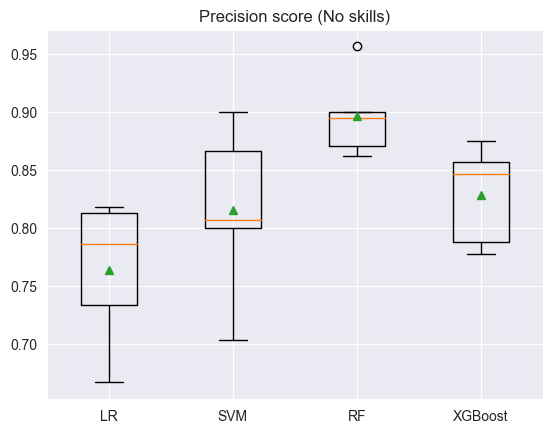
from imblearn.pipeline import Pipeline  
from sklearn.preprocessing import MinMaxScaler, PowerTransformer  
  
models,names=get\_models()  
results1=list()  
results2=list()  
results3=list()  
z1=list()  
z2=list()  
z3=list()  
  
for i in range(len(models)):  
   
 steps=[('t1',MinMaxScaler()),('t2',PowerTransformer()),('m',models[i])]  
 pipeline=Pipeline(steps=steps)  
 # use pipeline to avoid information leaking  
 # Need to include preprocessing in cross-validation  
 # https://amueller.github.io/COMS4995-s19/slides/aml-05-preprocessing/#15  
 # https://machinelearningmastery.com/data-preparation-without-data-leakage/  
 auc, recall, precision = evaluate(X\_train,y\_train,pipeline) # like this  
 results1.append(auc)  
 results2.append(recall)  
 results3.append(precision)  
 z1.append(auc.mean())  
 z2.append(recall.mean())  
 z3.append(precision.mean())  
   
 print(f'Auc score for {names[i]}: {auc.mean()}({auc.std()})')  
 print(f'Recall score for {names[i]}: {recall.mean()}({recall.std()})')  
 print(f'Precision score for {names[i]}: {precision.mean()}({precision.std()})')  
  
# store to the final csv   
df1\_auc = pd.DataFrame(z1)  
df1\_auc = df1\_auc.T  
df1\_recall = pd.DataFrame(z2)  
df1\_recall = df1\_recall.T  
df1\_precision = pd.DataFrame(z3)  
df1\_precision = df1\_precision.T

Auc score for LR: 0.9116334922842922(0.02147728418251929)  
Recall score for LR: 0.28315911730545873(0.09156142086129666)  
Precision score for LR: 0.7632792207792207(0.05687012262052182)  
Auc score for SVM: 0.8729056657531388(0.0501463536082928)  
Recall score for SVM: 0.43704994192799074(0.09309332522395415)  
Precision score for SVM: 0.8153643966547193(0.06719928721506482)  
Auc score for RF: 0.9427579165056589(0.00991077010752529)  
Recall score for RF: 0.5284552845528455(0.08002398736047495)  
Precision score for RF: 0.8968590577376846(0.03302942814865178)  
Auc score for XGBoost: 0.9274952387834864(0.01953084688089563)  
Recall score for XGBoost: 0.5815331010452962(0.05888581919931004)  
Precision score for XGBoost: 0.8287906537906539(0.03877299537708349)

plt.figure()  
plt.boxplot(results1,labels=names,showmeans=True);  
plt.title("AUC score (No skills)");  
plt.figure()  
plt.boxplot(results2,labels=names,showmeans=True);  
plt.title("Recall score (No skills)");  
plt.figure()  
plt.boxplot(results3,labels=names,showmeans=True);  
plt.title("Precision score (No skills)");





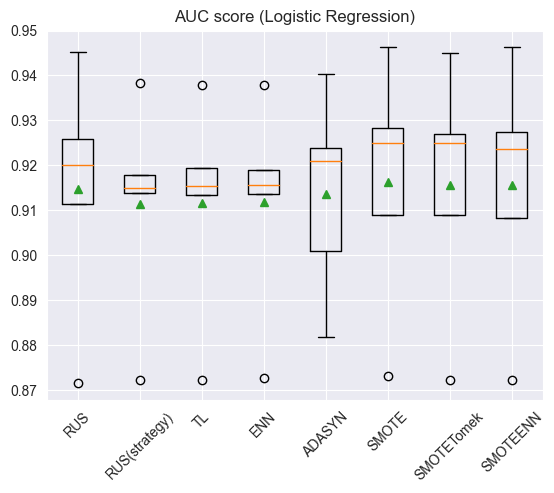


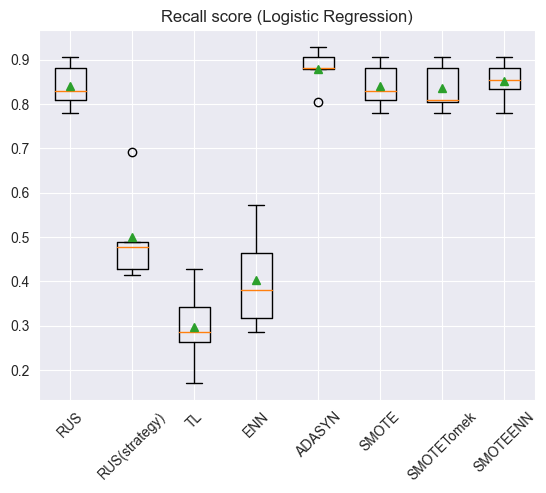
## Data Sampling

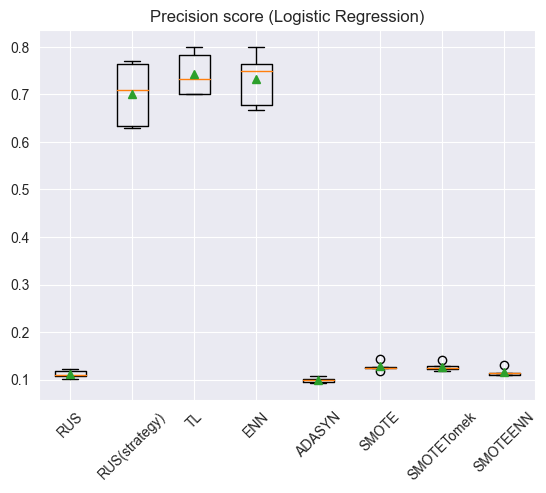
from imblearn.over\_sampling import SMOTE, ADASYN  
from imblearn.combine import SMOTETomek, SMOTEENN  
from imblearn.under\_sampling import TomekLinks, EditedNearestNeighbours, RandomUnderSampler  
  
# creat a list that contain these resample functions  
def get\_sampling():  
   
 sampling,seq=list(),list()  
   
 sampling.append(RandomUnderSampler()) # undersample  
 seq.append('RUS')  
   
 sampling.append(RandomUnderSampler(sampling\_strategy={0:3000})) # undersample  
 seq.append('RUS(strategy)')  
   
 sampling.append(TomekLinks()) # undersample  
 seq.append('TL')  
   
 sampling.append(EditedNearestNeighbours()) # undersample   
 # Edited Nearest Neighbours (ENN) is an undersampling technique used to handle imbalanced datasets by removing samples close to the decision boundary  
 seq.append('ENN')  
   
 sampling.append(ADASYN()) # oversample  
 seq.append('ADASYN')  
   
 sampling.append(SMOTE()) # oversample  
 seq.append('SMOTE')  
   
 sampling.append(SMOTETomek(tomek=TomekLinks(sampling\_strategy='majority'))) # ensemble method  
 seq.append('SMOTETomek')  
   
 sampling.append(SMOTEENN(enn=EditedNearestNeighbours(sampling\_strategy='majority'))) # ensemble method  
 seq.append('SMOTEENN')  
   
 # sampling.append(sv.MSMOTE()) # ensemble method  
 # seq.append('MSMOTE')  
   
 return sampling,seq

# Logistic Regression  
sampling,seq=get\_sampling()  
results1=list()  
results2=list()  
results3=list()  
nums=list()  
z1=list()  
z2=list()  
z3=list()  
  
for i in range(len(sampling)):  
 steps=[('t1',MinMaxScaler()),('t2',PowerTransformer()),('s',sampling[i]),('m',LogisticRegression())]  
 pipeline=Pipeline(steps=steps)  
   
 auc, recall, precision =evaluate(X\_train,y\_train,pipeline)  
 results1.append(auc)  
 results2.append(recall)  
 results3.append(precision)  
 z1.append(auc.mean())  
 z2.append(recall.mean())  
 z3.append(precision.mean())  
  
 # output the number of samples after resampling  
 res\_X, y\_res = sampling[i].fit\_resample(X\_train,y\_train)  
 nums.append(dict(Counter(y\_res)))  
  
 print(f'Auc score for {seq[i]}: {auc.mean()}({auc.std()}), resampled y\_train: {nums[i]}')  
 print(f'Recall score for {seq[i]}: {recall.mean()}({recall.std()}), resampled y\_train: {nums[i]}')  
 print(f'Precision score for {seq[i]}: {precision.mean()}({precision.std()}), resampled y\_train: {nums[i]}')  
  
  
df2\_auc1 = z1  
df2\_recall1 = z2  
df2\_precision1 = z3  
   
print('\n')  
  
plt.figure()  
plt.boxplot(results1,labels=seq,showmeans=True);  
plt.title("AUC score (Logistic Regression)");  
plt.xticks(rotation=45);  
plt.figure()  
plt.boxplot(results2,labels=seq,showmeans=True);  
plt.title("Recall score (Logistic Regression)");  
plt.xticks(rotation=45);  
plt.figure()  
plt.boxplot(results3,labels=seq,showmeans=True);  
plt.title("Precision score (Logistic Regression)");  
plt.xticks(rotation=45);

Auc score for RUS: 0.9147900767632429(0.024287215182301574), resampled y\_train: {0: 208, 1: 208}  
Recall score for RUS: 0.8409988385598142(0.04572063542390822), resampled y\_train: {0: 208, 1: 208}  
Precision score for RUS: 0.11192335726854863(0.007536122774819808), resampled y\_train: {0: 208, 1: 208}  
Auc score for RUS(strategy): 0.911458427203949(0.021542982686334262), resampled y\_train: {0: 3000, 1: 208}  
Recall score for RUS(strategy): 0.49953542392566785(0.09938318187127679), resampled y\_train: {0: 3000, 1: 208}  
Precision score for RUS(strategy): 0.7005253518411413(0.06065701741481283), resampled y\_train: {0: 3000, 1: 208}  
Auc score for TL: 0.9116663633255191(0.02149614427252159), resampled y\_train: {0: 8708, 1: 208}  
Recall score for TL: 0.29767711962833915(0.08554351924402061), resampled y\_train: {0: 8708, 1: 208}  
Precision score for TL: 0.7431884057971014(0.04149485686681927), resampled y\_train: {0: 8708, 1: 208}  
Auc score for ENN: 0.9117640820992265(0.021409297006735998), resampled y\_train: {0: 8564, 1: 208}  
Recall score for ENN: 0.40371660859465736(0.10357702479063342), resampled y\_train: {0: 8564, 1: 208}  
Precision score for ENN: 0.7319887955182073(0.051265448017402085), resampled y\_train: {0: 8564, 1: 208}  
Auc score for ADASYN: 0.9135771958033002(0.02024188133408557), resampled y\_train: {0: 8738, 1: 8748}  
Recall score for ADASYN: 0.8794425087108013(0.04151819711959892), resampled y\_train: {0: 8738, 1: 8748}  
Precision score for ADASYN: 0.0992570361530081(0.005014603729195131), resampled y\_train: {0: 8738, 1: 8748}  
Auc score for SMOTE: 0.9163005247889707(0.02469733703944951), resampled y\_train: {0: 8738, 1: 8738}  
Recall score for SMOTE: 0.8409988385598142(0.04572063542390822), resampled y\_train: {0: 8738, 1: 8738}  
Precision score for SMOTE: 0.12770098982607736(0.008246778675497182), resampled y\_train: {0: 8738, 1: 8738}  
Auc score for SMOTETomek: 0.9155803032912175(0.02455700935758777), resampled y\_train: {0: 8713, 1: 8738}  
Recall score for SMOTETomek: 0.8361207897793264(0.04795833914142167), resampled y\_train: {0: 8713, 1: 8738}  
Precision score for SMOTETomek: 0.12726681689596459(0.007831752761269574), resampled y\_train: {0: 8713, 1: 8738}  
Auc score for SMOTEENN: 0.9155324552823986(0.024825708027060373), resampled y\_train: {0: 8251, 1: 8738}  
Recall score for SMOTEENN: 0.8506387921022067(0.042609819904384685), resampled y\_train: {0: 8251, 1: 8738}  
Precision score for SMOTEENN: 0.11526116054750599(0.007654954407019954), resampled y\_train: {0: 8251, 1: 8738}

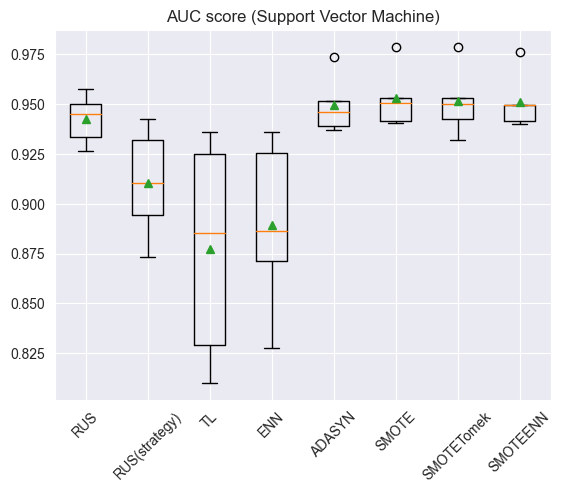




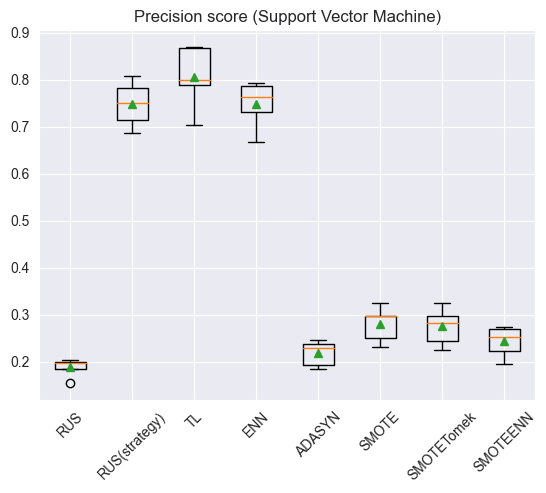


# Support Vector Machine  
sampling,seq=get\_sampling()  
results1=list()  
results2=list()  
results3=list()  
nums=list()  
z1=list()  
z2=list()  
z3=list()  
  
for i in range(len(sampling)):  
 steps=[('t1',MinMaxScaler()),('t2',PowerTransformer()),('s',sampling[i]),('m',SVC(probability=True))]  
 pipeline=Pipeline(steps=steps)  
   
 auc, recall, precision=evaluate(X\_train,y\_train,pipeline)  
 results1.append(auc)  
 results2.append(recall)  
 results3.append(precision)  
 z1.append(auc.mean())  
 z2.append(recall.mean())  
 z3.append(precision.mean())  
  
 # output the number of samples after resampling  
 res\_X, y\_res = sampling[i].fit\_resample(X\_train,y\_train)  
 nums.append(dict(Counter(y\_res)))  
  
 print(f'Auc score for {seq[i]}: {auc.mean()}({auc.std()}), resampled y\_train: {nums[i]}')  
 print(f'Recall score for {seq[i]}: {recall.mean()}({recall.std()}), resampled y\_train: {nums[i]}')  
 print(f'Precision score for {seq[i]}: {precision.mean()}({precision.std()}), resampled y\_train: {nums[i]}')  
  
df2\_auc2 = z1  
df2\_recall2 = z2  
df2\_precision2 = z3  
   
print('\n')  
  
plt.figure()  
plt.boxplot(results1,labels=seq,showmeans=True);  
plt.title("AUC score (Support Vector Machine)");  
plt.xticks(rotation=45);  
plt.figure()  
plt.boxplot(results2,labels=seq,showmeans=True);  
plt.title("Recall score (Support Vector Machine)");  
plt.xticks(rotation=45);  
plt.figure()  
plt.boxplot(results3,labels=seq,showmeans=True);  
plt.title("Precision score (Support Vector Machine)");  
plt.xticks(rotation=45);

Auc score for RUS: 0.9425026011721132(0.011133004442881084), resampled y\_train: {0: 208, 1: 208}  
Recall score for RUS: 0.8221835075493613(0.02395327838647695), resampled y\_train: {0: 208, 1: 208}  
Precision score for RUS: 0.18757387891448446(0.01739308989921437), resampled y\_train: {0: 208, 1: 208}  
Auc score for RUS(strategy): 0.9104138851583802(0.025063457755070673), resampled y\_train: {0: 3000, 1: 208}  
Recall score for RUS(strategy): 0.5815331010452962(0.07866783420272852), resampled y\_train: {0: 3000, 1: 208}  
Precision score for RUS(strategy): 0.7477884615384616(0.04401970835963298), resampled y\_train: {0: 3000, 1: 208}  
Auc score for TL: 0.8771015109157195(0.05018345877766628), resampled y\_train: {0: 8708, 1: 208}  
Recall score for TL: 0.45133565621370497(0.10185400012360235), resampled y\_train: {0: 8708, 1: 208}  
Precision score for TL: 0.8055628751280925(0.060908023248766874), resampled y\_train: {0: 8708, 1: 208}  
Auc score for ENN: 0.8892412604139629(0.03895898015413327), resampled y\_train: {0: 8564, 1: 208}  
Recall score for ENN: 0.5430894308943089(0.0877017319669606), resampled y\_train: {0: 8564, 1: 208}  
Precision score for ENN: 0.7475949489107384(0.04576045691432794), resampled y\_train: {0: 8564, 1: 208}  
Auc score for ADASYN: 0.9493817804763843(0.013268496949032628), resampled y\_train: {0: 8738, 1: 8748}  
Recall score for ADASYN: 0.8849012775842043(0.03432129443780982), resampled y\_train: {0: 8738, 1: 8748}  
Precision score for ADASYN: 0.21797610488035088(0.024451468361174705), resampled y\_train: {0: 8738, 1: 8748}  
Auc score for SMOTE: 0.9528367549067683(0.013666185054798658), resampled y\_train: {0: 8738, 1: 8738}  
Recall score for SMOTE: 0.8562137049941928(0.05132146471138589), resampled y\_train: {0: 8738, 1: 8738}  
Precision score for SMOTE: 0.27957412412412413(0.03394663650047322), resampled y\_train: {0: 8738, 1: 8738}  
Auc score for SMOTETomek: 0.9512476062893163(0.015516079694264226), resampled y\_train: {0: 8719, 1: 8738}  
Recall score for SMOTETomek: 0.8562137049941928(0.05132146471138589), resampled y\_train: {0: 8719, 1: 8738}  
Precision score for SMOTETomek: 0.2747006159597446(0.03531418554247301), resampled y\_train: {0: 8719, 1: 8738}  
Auc score for SMOTEENN: 0.9512091396561072(0.013021831948516437), resampled y\_train: {0: 8249, 1: 8738}  
Recall score for SMOTEENN: 0.8753774680603948(0.04019925524150236), resampled y\_train: {0: 8249, 1: 8738}  
Precision score for SMOTEENN: 0.24339391093208113(0.029729405469229014), resampled y\_train: {0: 8249, 1: 8738}

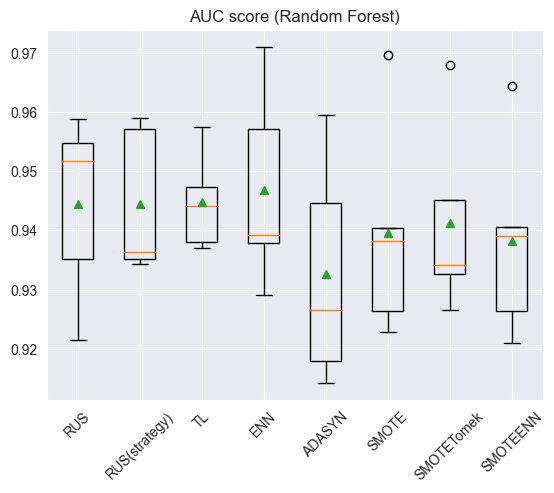


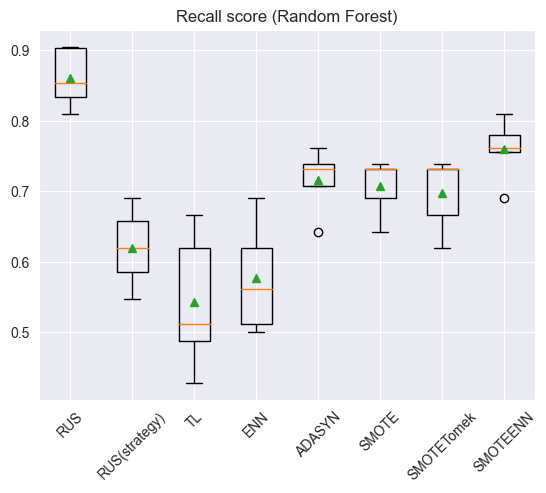


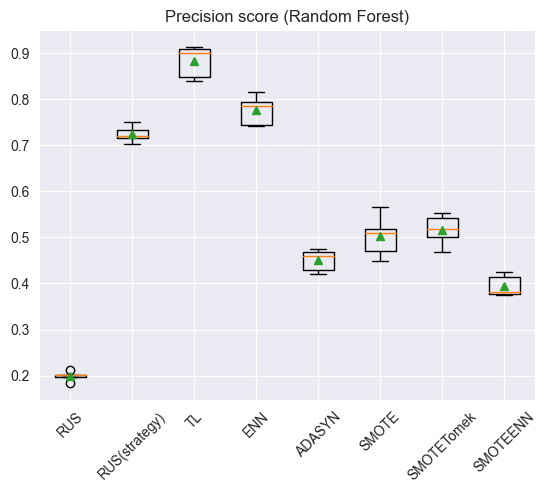


# Random Forest  
sampling,seq=get\_sampling()  
results1=list()  
results2=list()  
results3=list()  
nums=list()  
z1=list()  
z2=list()  
z3=list()  
  
for i in range(len(sampling)):  
 steps=[('t1',MinMaxScaler()),('t2',PowerTransformer()),('s',sampling[i]),('m',RandomForestClassifier(n\_estimators=1000))]  
 pipeline=Pipeline(steps=steps)  
   
 auc, recall, precision =evaluate(X\_train,y\_train,pipeline)  
 results1.append(auc)  
 results2.append(recall)  
 results3.append(precision)  
 z1.append(auc.mean())  
 z2.append(recall.mean())  
 z3.append(precision.mean())  
  
 # output the number of samples after resampling  
 res\_X, y\_res = sampling[i].fit\_resample(X\_train,y\_train)  
 nums.append(dict(Counter(y\_res)))  
  
 print(f'Auc score for {seq[i]}: {auc.mean()}({auc.std()}), resampled y\_train: {nums[i]}')  
 print(f'Recall score for {seq[i]}: {recall.mean()}({recall.std()}), resampled y\_train: {nums[i]}')  
 print(f'Precision score for {seq[i]}: {precision.mean()}({precision.std()}), resampled y\_train: {nums[i]}')  
  
df2\_auc3 = z1  
df2\_recall3 = z2  
df2\_precision3 = z3  
   
print('\n')  
  
plt.figure()  
plt.boxplot(results1,labels=seq,showmeans=True);  
plt.title("AUC score (Random Forest)");  
plt.xticks(rotation=45);  
plt.figure()  
plt.boxplot(results2,labels=seq,showmeans=True);  
plt.title("Recall score (Random Forest)");  
plt.xticks(rotation=45);  
plt.figure()  
plt.boxplot(results3,labels=seq,showmeans=True);  
plt.title("Precision score (Random Forest)");  
plt.xticks(rotation=45);

Auc score for RUS: 0.9443853405255455(0.013990571359592632), resampled y\_train: {0: 208, 1: 208}  
Recall score for RUS: 0.8607433217189314(0.037685825843569595), resampled y\_train: {0: 208, 1: 208}  
Precision score for RUS: 0.1990493855474527(0.008962399900387634), resampled y\_train: {0: 208, 1: 208}  
Auc score for RUS(strategy): 0.9443788992470298(0.011185325233319526), resampled y\_train: {0: 3000, 1: 208}  
Recall score for RUS(strategy): 0.6202090592334495(0.05077632656412751), resampled y\_train: {0: 3000, 1: 208}  
Precision score for RUS(strategy): 0.7234891468123175(0.016183958728801816), resampled y\_train: {0: 3000, 1: 208}  
Auc score for TL: 0.9448005917060499(0.00739656512510913), resampled y\_train: {0: 8708, 1: 208}  
Recall score for TL: 0.5428571428571428(0.08736809798139572), resampled y\_train: {0: 8708, 1: 208}  
Precision score for TL: 0.8818657826511963(0.03168230495605159), resampled y\_train: {0: 8708, 1: 208}  
Auc score for ENN: 0.9468191196092952(0.015128526418301034), resampled y\_train: {0: 8564, 1: 208}  
Recall score for ENN: 0.5765389082462253(0.07075201530153265), resampled y\_train: {0: 8564, 1: 208}  
Precision score for ENN: 0.7752989347205143(0.028691768359906226), resampled y\_train: {0: 8564, 1: 208}  
Auc score for ADASYN: 0.9325892626994641(0.01707156152335106), resampled y\_train: {0: 8738, 1: 8748}  
Recall score for ADASYN: 0.716376306620209(0.04066135575121917), resampled y\_train: {0: 8738, 1: 8748}  
Precision score for ADASYN: 0.4502735332956046(0.02134913487720557), resampled y\_train: {0: 8738, 1: 8748}  
Auc score for SMOTE: 0.9395117452580195(0.01647758652668622), resampled y\_train: {0: 8738, 1: 8738}  
Recall score for SMOTE: 0.7069686411149825(0.03626335437556129), resampled y\_train: {0: 8738, 1: 8738}  
Precision score for SMOTE: 0.5018565552419695(0.04049469717756246), resampled y\_train: {0: 8738, 1: 8738}  
Auc score for SMOTETomek: 0.9412493004806407(0.014603475773933257), resampled y\_train: {0: 8718, 1: 8738}  
Recall score for SMOTETomek: 0.697444831591173(0.047103523737557544), resampled y\_train: {0: 8718, 1: 8738}  
Precision score for SMOTETomek: 0.516245894909688(0.030199970939505345), resampled y\_train: {0: 8718, 1: 8738}  
Auc score for SMOTEENN: 0.938275947887402(0.014985353689644668), resampled y\_train: {0: 8254, 1: 8738}  
Recall score for SMOTEENN: 0.7596980255516842(0.03930475583079039), resampled y\_train: {0: 8254, 1: 8738}  
Precision score for SMOTEENN: 0.39458888199449504(0.020665635566719916), resampled y\_train: {0: 8254, 1: 8738}

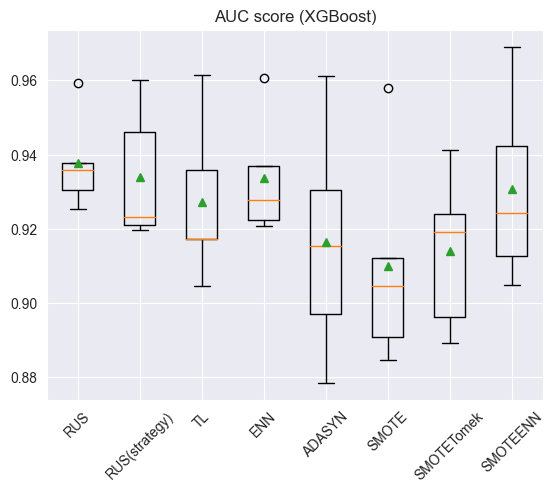


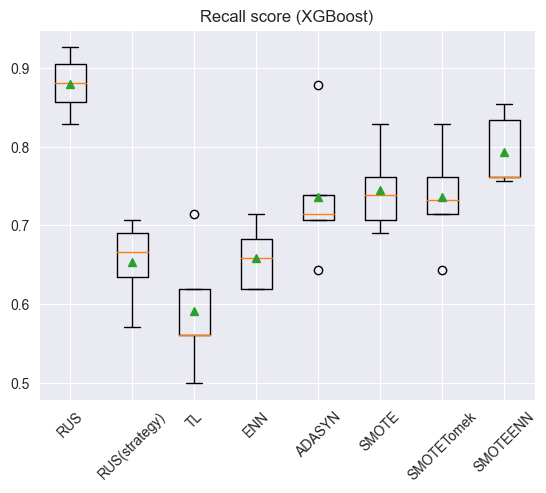


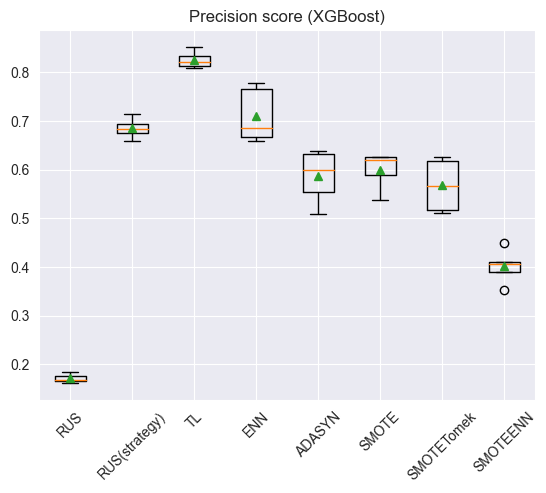


# XGBoost  
sampling,seq=get\_sampling()  
results1=list()  
results2=list()  
results3=list()  
nums=list()  
z1=list()  
z2=list()  
z3=list()  
  
for i in range(len(sampling)):  
 steps=[('t1',MinMaxScaler()),('t2',PowerTransformer()),('s',sampling[i]),('m',xgb.XGBClassifier())]  
 pipeline=Pipeline(steps=steps)  
   
 auc, recall, precision=evaluate(X\_train,y\_train,pipeline)  
 results1.append(auc)  
 results2.append(recall)  
 results3.append(precision)  
 z1.append(auc.mean())  
 z2.append(recall.mean())  
 z3.append(precision.mean())  
  
 # output the number of samples after resampling  
 res\_X, y\_res = sampling[i].fit\_resample(X\_train,y\_train)  
 nums.append(dict(Counter(y\_res)))  
  
 print(f'Auc score for {seq[i]}: {auc.mean()}({auc.std()}), resampled y\_train: {nums[i]}')  
 print(f'Recall score for {seq[i]}: {recall.mean()}({recall.std()}), resampled y\_train: {nums[i]}')  
 print(f'Precision score for {seq[i]}: {precision.mean()}({precision.std()}), resampled y\_train: {nums[i]}')  
  
df2\_auc4 = z1  
df2\_recall4 = z2  
df2\_precision4 = z3  
# combine to one form of second part  
df2\_auc = pd.DataFrame(list(zip(df2\_auc1, df2\_auc2, df2\_auc3, df2\_auc4)))  
df2\_recall = pd.DataFrame(list(zip(df2\_recall1, df2\_recall2, df2\_recall3, df2\_recall4)))  
df2\_precision = pd.DataFrame(list(zip(df2\_precision1, df2\_precision2, df2\_precision3, df2\_precision4)))  
  
print('\n')  
  
plt.figure()  
plt.boxplot(results1,labels=seq,showmeans=True);  
plt.title("AUC score (XGBoost)");  
plt.xticks(rotation=45);  
plt.figure()  
plt.boxplot(results2,labels=seq,showmeans=True);  
plt.title("Recall score (XGBoost)");  
plt.xticks(rotation=45);  
plt.figure()  
plt.boxplot(results3,labels=seq,showmeans=True);  
plt.title("Precision score (XGBoost)");  
plt.xticks(rotation=45);

Auc score for RUS: 0.9377051404174033(0.011638445327647254), resampled y\_train: {0: 208, 1: 208}  
Recall score for RUS: 0.8797909407665505(0.03435979017347385), resampled y\_train: {0: 208, 1: 208}  
Precision score for RUS: 0.1708655922141279(0.008121658697995561), resampled y\_train: {0: 208, 1: 208}  
Auc score for RUS(strategy): 0.9340763790113262(0.01626795585979991), resampled y\_train: {0: 3000, 1: 208}  
Recall score for RUS(strategy): 0.654006968641115(0.04805387713472659), resampled y\_train: {0: 3000, 1: 208}  
Precision score for RUS(strategy): 0.6850454801032464(0.018829639410973587), resampled y\_train: {0: 3000, 1: 208}  
Auc score for TL: 0.9272735901830998(0.019771426321938422), resampled y\_train: {0: 8708, 1: 208}  
Recall score for TL: 0.5910569105691057(0.07220796092696573), resampled y\_train: {0: 8708, 1: 208}  
Precision score for TL: 0.8253612128612128(0.015864270129579142), resampled y\_train: {0: 8708, 1: 208}  
Auc score for ENN: 0.9337220239360411(0.01458929380953022), resampled y\_train: {0: 8564, 1: 208}  
Recall score for ENN: 0.6587688734030197(0.036935989240977386), resampled y\_train: {0: 8564, 1: 208}  
Precision score for ENN: 0.7103794876958057(0.050551356948045925), resampled y\_train: {0: 8564, 1: 208}  
Auc score for ADASYN: 0.9165036017928465(0.028341681430232146), resampled y\_train: {0: 8738, 1: 8748}  
Recall score for ADASYN: 0.7361207897793263(0.07765563140198377), resampled y\_train: {0: 8738, 1: 8748}  
Precision score for ADASYN: 0.5863845649102922(0.04913160289045667), resampled y\_train: {0: 8738, 1: 8748}  
Auc score for SMOTE: 0.9100114302186301(0.02582697656776607), resampled y\_train: {0: 8738, 1: 8738}  
Recall score for SMOTE: 0.7454123112659697(0.04863453637122899), resampled y\_train: {0: 8738, 1: 8738}  
Precision score for SMOTE: 0.5990544662309368(0.033906556413270623), resampled y\_train: {0: 8738, 1: 8738}  
Auc score for SMOTETomek: 0.913999455051624(0.018943232150135027), resampled y\_train: {0: 8720, 1: 8738}  
Recall score for SMOTETomek: 0.7360046457607432(0.06088605779477209), resampled y\_train: {0: 8720, 1: 8738}  
Precision score for SMOTETomek: 0.5672529709819694(0.04843861313862937), resampled y\_train: {0: 8720, 1: 8738}  
Auc score for SMOTEENN: 0.9306976449400681(0.022942384550858064), resampled y\_train: {0: 8253, 1: 8738}  
Recall score for SMOTEENN: 0.7933797909407666(0.041475612841790396), resampled y\_train: {0: 8253, 1: 8738}  
Precision score for SMOTEENN: 0.4016289513825537(0.03114683248362869), resampled y\_train: {0: 8253, 1: 8738}







## Cost Sensitive Learning

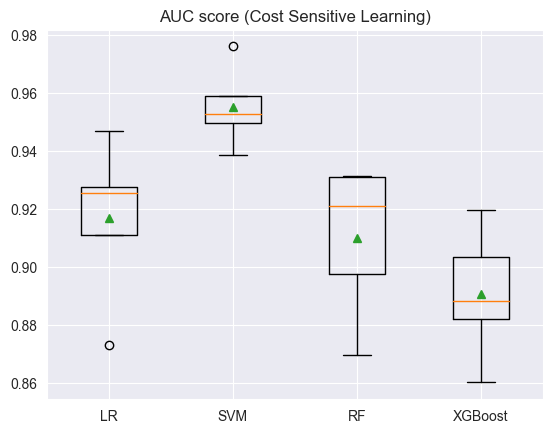
Instead of trying to optimize the accuracy, it is to minimize the total misclassification cost

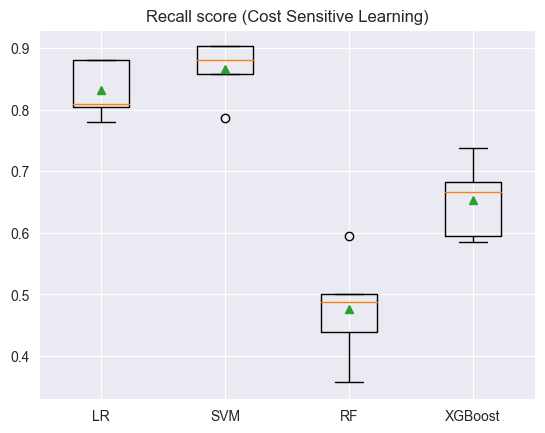
# use the hyperparameter class\_weight to implement  
def get\_csl\_models():  
 models,names=list(),list()  
   
 models.append(LogisticRegression(class\_weight='balanced'))  
 names.append('LR')  
   
 models.append(SVC(probability=True,class\_weight='balanced'))  
 names.append('SVM')  
   
 models.append(RandomForestClassifier(n\_estimators=1000,class\_weight='balanced'))  
 names.append('RF')  
   
 models.append(xgb.XGBClassifier(scale\_pos\_weight=float(np.sum(y\_train == 0)) / np.sum(y\_train == 1), seed=42))  
 names.append('XGBoost')  
   
 return models,names

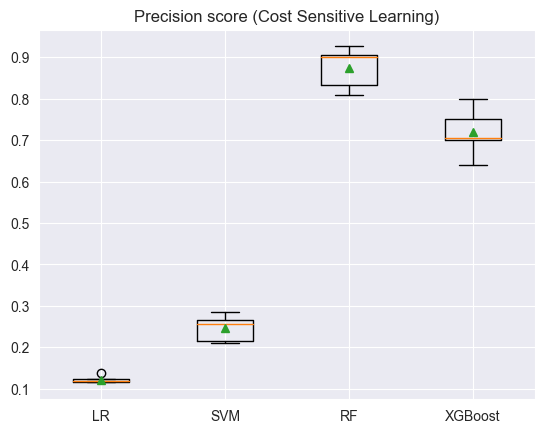
models,names=get\_csl\_models()  
results1=list()  
results2=list()  
results3=list()  
z1=list()  
z2=list()  
z3=list()  
for i in range(len(models)):  
   
 steps=[('t1',MinMaxScaler()),('t2',PowerTransformer()),('m',models[i])]  
 pipeline=Pipeline(steps=steps)  
   
 auc, recall, precision=evaluate(X\_train,y\_train,pipeline)  
 results1.append(auc)  
 results2.append(recall)  
 results3.append(precision)  
 z1.append(auc.mean())  
 z2.append(recall.mean())  
 z3.append(precision.mean())  
   
   
 print(f'Auc score for {names[i]}: {auc.mean()}({auc.std()})')  
 print(f'Recall score for {names[i]}: {recall.mean()}({recall.std()})')  
 print(f'Precision score for {names[i]}: {precision.mean()}({precision.std()})')  
   
# store to the final table   
df3\_auc = pd.DataFrame(z1)  
df3\_auc = df3\_auc.T  
df3\_recall = pd.DataFrame(z2)  
df3\_recall = df3\_recall.T  
df3\_precision = pd.DataFrame(z3)  
df3\_precision = df3\_precision.T

Auc score for LR: 0.9167190810328603(0.02470598944875057)  
Recall score for LR: 0.8313588850174216(0.041677096065506264)  
Precision score for LR: 0.12206465507604032(0.008169879875877011)  
Auc score for SVM: 0.9551156944170802(0.012371886264589427)  
Recall score for SVM: 0.8657375145180023(0.04336606760587959)  
Precision score for SVM: 0.24608596353590792(0.02920926575482835)  
Auc score for RF: 0.9100303535458891(0.023605275260754368)  
Recall score for RF: 0.4758420441347271(0.07800919160775513)  
Precision score for RF: 0.8743426943426943(0.045537890174517405)  
Auc score for XGBoost: 0.8905625214914978(0.02004696706804715)  
Recall score for XGBoost: 0.6536585365853659(0.05697760871641967)  
Precision score for XGBoost: 0.7191142191142191(0.053238837913180355)

plt.figure()  
plt.boxplot(results1,labels=names,showmeans=True);  
plt.title("AUC score (Cost Sensitive Learning)");  
plt.figure()  
plt.boxplot(results2,labels=names,showmeans=True);  
plt.title("Recall score (Cost Sensitive Learning)");  
plt.figure()  
plt.boxplot(results3,labels=names,showmeans=True);  
plt.title("Precision score (Cost Sensitive Learning)");







## Model Selection

choose the model with best performance

ind = ['No skill'] + seq +['Cost Sensitive Learning']  
df\_auc = pd.concat([df1\_auc, df2\_auc, df3\_auc], ignore\_index=True)  
df\_auc.index = ind  
df\_auc.columns = names  
  
df\_recall = pd.concat([df1\_recall, df2\_recall, df3\_recall], ignore\_index=True)  
df\_recall.index = ind  
df\_recall.columns = names  
  
df\_precision = pd.concat([df1\_precision, df2\_precision, df3\_precision], ignore\_index=True)  
df\_precision.index = ind  
df\_precision.columns = names  
  
df\_auc = df\_auc.T  
df\_recall = df\_recall.T  
df\_precision = df\_precision.T  
df\_auc.to\_csv("AUC scores.csv", index=True)  
df\_recall.to\_csv("Recall scores.csv", index=True)  
df\_precision.to\_csv("Precision scores.csv", index=True)

# find the biggest value of each metric  
biggest\_value = df\_auc.max().max()  
column\_name = df\_auc.max().idxmax()  
row\_name = df\_auc[column\_name].idxmax()  
print("Biggest AUC score:", biggest\_value, row\_name, column\_name)  
df\_auc

Biggest AUC score: 0.9551156944170802 SVM Cost Sensitive Learning

No skill RUS RUS(strategy) TL ENN ADASYN \  
LR 0.911633 0.914790 0.911458 0.911666 0.911764 0.913577   
SVM 0.872906 0.942503 0.910414 0.877102 0.889241 0.949382   
RF 0.942758 0.944385 0.944379 0.944801 0.946819 0.932589   
XGBoost 0.927495 0.937705 0.934076 0.927274 0.933722 0.916504   
  
 SMOTE SMOTETomek SMOTEENN Cost Sensitive Learning   
LR 0.916301 0.915580 0.915532 0.916719   
SVM 0.952837 0.951248 0.951209 0.955116   
RF 0.939512 0.941249 0.938276 0.910030   
XGBoost 0.910011 0.913999 0.930698 0.890563

biggest\_value = df\_recall.max().max()  
column\_name = df\_recall.max().idxmax()  
row\_name = df\_recall[column\_name].idxmax()  
print("Biggest Recall score:", biggest\_value, row\_name, column\_name)  
df\_recall

Biggest Recall score: 0.8849012775842043 SVM ADASYN

No skill RUS RUS(strategy) TL ENN ADASYN \  
LR 0.283159 0.840999 0.499535 0.297677 0.403717 0.879443   
SVM 0.437050 0.822184 0.581533 0.451336 0.543089 0.884901   
RF 0.528455 0.860743 0.620209 0.542857 0.576539 0.716376   
XGBoost 0.581533 0.879791 0.654007 0.591057 0.658769 0.736121   
  
 SMOTE SMOTETomek SMOTEENN Cost Sensitive Learning   
LR 0.840999 0.836121 0.850639 0.831359   
SVM 0.856214 0.856214 0.875377 0.865738   
RF 0.706969 0.697445 0.759698 0.475842   
XGBoost 0.745412 0.736005 0.793380 0.653659

biggest\_value = df\_precision.max().max()  
column\_name = df\_precision.max().idxmax()  
row\_name = df\_precision[column\_name].idxmax()  
print("Biggest Precision score:", biggest\_value, row\_name, column\_name)  
df\_precision

Biggest Precision score: 0.8968590577376846 RF No skill

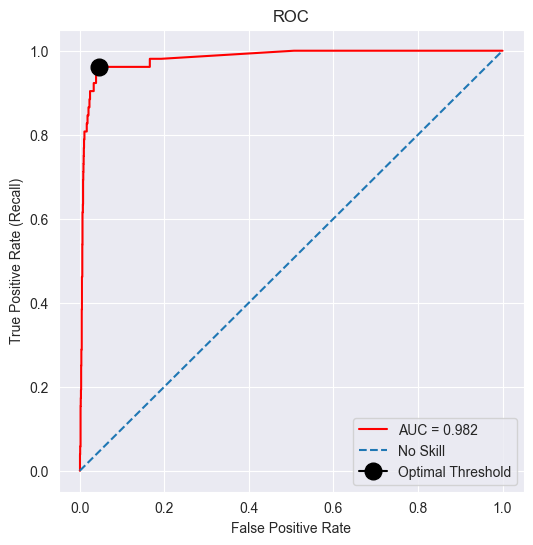
No skill RUS RUS(strategy) TL ENN ADASYN \  
LR 0.763279 0.111923 0.700525 0.743188 0.731989 0.099257   
SVM 0.815364 0.187574 0.747788 0.805563 0.747595 0.217976   
RF 0.896859 0.199049 0.723489 0.881866 0.775299 0.450274   
XGBoost 0.828791 0.170866 0.685045 0.825361 0.710379 0.586385   
  
 SMOTE SMOTETomek SMOTEENN Cost Sensitive Learning   
LR 0.127701 0.127267 0.115261 0.122065   
SVM 0.279574 0.274701 0.243394 0.246086   
RF 0.501857 0.516246 0.394589 0.874343   
XGBoost 0.599054 0.567253 0.401629 0.719114

## Choose model according to the higest AUC

model=SVC(probability=True,class\_weight='balanced')  
  
steps=[('t1',MinMaxScaler()),('t2',PowerTransformer()),('m',model)]  
pipeline=Pipeline(steps=steps)  
  
pipeline.fit(X\_train,y\_train)  
y\_probs=pipeline.predict\_proba(X\_test)[:,1]

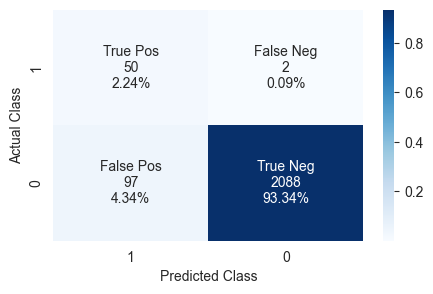
# roc curve  
from sklearn.metrics import auc  
from sklearn.metrics import roc\_curve, roc\_auc\_score  
fpr, tpr, thresholds = roc\_curve(y\_test, y\_probs)  
   
roc\_auc = auc(fpr, tpr) # roc\_auc\_score(y\_test, y\_probs)  
  
g\_means = np.sqrt(tpr\*(1-fpr))  
ix = np.argmax(g\_means)  
thresh=thresholds[ix]  
  
print('Best Threshold=%f, g\_means=%.3f' % (thresholds[ix], g\_means[ix]))  
  
plt.figure(figsize=(6,6))  
plt.title('ROC')  
plt.plot(fpr, tpr, 'r', label = 'AUC = %0.3f' % roc\_auc)  
plt.plot([0, 1], [0, 1],'--',label='No Skill')  
plt.plot(fpr[ix],tpr[ix],marker='o', markersize=12,color='black',label='Optimal Threshold');  
plt.legend(loc = 'lower right')  
plt.ylabel('True Positive Rate (Recall)')  
plt.xlabel('False Positive Rate')  
plt.show()

Best Threshold=0.054155, g\_means=0.959



# Threshold Moving  
from numpy import flip  
from sklearn.metrics import classification\_report, confusion\_matrix  
  
y\_pred=(y\_probs>=thresh) #   
  
print(f"Classification report:\n{classification\_report(y\_test, y\_pred)}")  
  
# Define a function to plot the confusion matrix as a heatmap  
def plot\_confusion\_matrix(y\_true, y\_prediction):  
 cm = flip(confusion\_matrix(y\_true, y\_prediction))  
 group\_names = ["True Pos", "False Neg", "False Pos", "True Neg"]  
 group\_counts = ["{0:0.0f}".format(value) for value in cm.flatten()]  
 group\_percentages = ["{0:.2%}".format(value) for value in cm.flatten()/np.sum(cm)] # flatten() is used to return a copy of a given array  
 labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in zip(group\_names, group\_counts, group\_percentages)]  
 labels = np.asarray(labels).reshape(2,2)  
  
 plt.subplots(figsize = (5,3))  
 ax = sns.heatmap(cm/np.sum(cm), annot=labels, fmt="", cmap='Blues')  
 ax.set\_xlabel('Predicted Class')  
 ax.set\_ylabel('Actual Class')  
 ax.xaxis.set\_ticklabels(['1', '0'])  
 ax.yaxis.set\_ticklabels(['1', '0'])  
 plt.show()  
  
# plot the confusion matrix as a heatmap  
plot\_confusion\_matrix(y\_test, y\_pred)  
  
from sklearn.metrics import balanced\_accuracy\_score  
# balanced-accuracy = (tpr+tnr)/2  
print('balanced-accuracy:%.3f'%balanced\_accuracy\_score(y\_test, y\_pred))

Classification report:  
 precision recall f1-score support  
  
 0 1.00 0.96 0.98 2185  
 1 0.34 0.96 0.50 52  
  
 accuracy 0.96 2237  
 macro avg 0.67 0.96 0.74 2237  
weighted avg 0.98 0.96 0.97 2237



balanced-accuracy:0.959

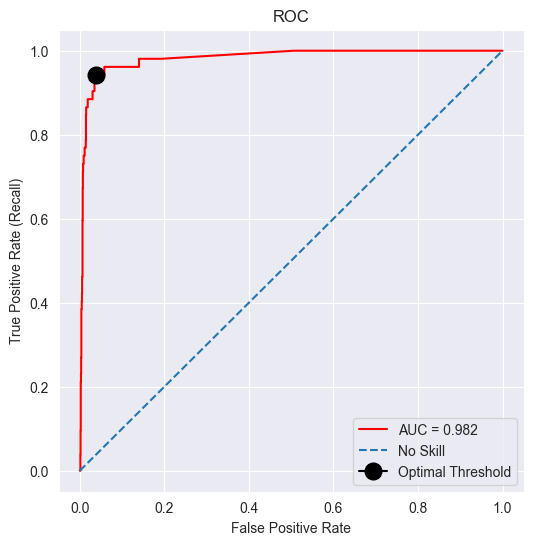
Although the sample size of FP is higher, the sample size of FN is at a lower level, which is more reasonable in practical applications

## Choose model according to the highest recall

model=SVC(probability=True)  
  
steps=[('t1',MinMaxScaler()),('t2',PowerTransformer()),('s',ADASYN()),('m',model)]  
pipeline=Pipeline(steps=steps)  
  
pipeline.fit(X\_train,y\_train)  
y\_probs=pipeline.predict\_proba(X\_test)[:,1]

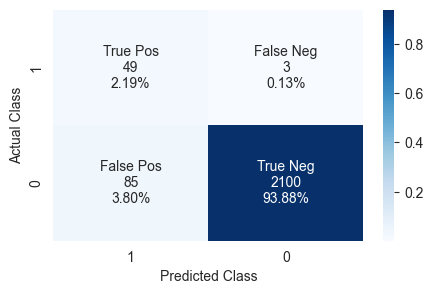
# roc curve  
from sklearn.metrics import auc  
from sklearn.metrics import roc\_curve, roc\_auc\_score  
fpr, tpr, thresholds = roc\_curve(y\_test, y\_probs)  
   
roc\_auc = auc(fpr, tpr) # roc\_auc\_score(y\_test, y\_probs)  
  
g\_means = np.sqrt(tpr\*(1-fpr))  
ix = np.argmax(g\_means)  
thresh=thresholds[ix]  
  
print('Best Threshold=%f, g\_means=%.3f' % (thresholds[ix], g\_means[ix]))  
  
plt.figure(figsize=(6,6))  
plt.title('ROC')  
plt.plot(fpr, tpr, 'r', label = 'AUC = %0.3f' % roc\_auc)  
plt.plot([0, 1], [0, 1],'--',label='No Skill')  
plt.plot(fpr[ix],tpr[ix],marker='o', markersize=12,color='black',label='Optimal Threshold');  
plt.legend(loc = 'lower right')  
plt.ylabel('True Positive Rate (Recall)')  
plt.xlabel('False Positive Rate')  
plt.show()

Best Threshold=0.770155, g\_means=0.952



# threshold moving  
from numpy import flip  
from sklearn.metrics import classification\_report, confusion\_matrix  
  
y\_pred=(y\_probs>=thresh) #   
  
print(f"Classification report:\n{classification\_report(y\_test, y\_pred)}")  
  
# Define a function to plot the confusion matrix as a heatmap  
def plot\_confusion\_matrix(y\_true, y\_prediction):  
 cm = flip(confusion\_matrix(y\_true, y\_prediction))  
 group\_names = ["True Pos", "False Neg", "False Pos", "True Neg"]  
 group\_counts = ["{0:0.0f}".format(value) for value in cm.flatten()]  
 group\_percentages = ["{0:.2%}".format(value) for value in cm.flatten()/np.sum(cm)] # flatten() is used to return a copy of a given array  
 labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in zip(group\_names, group\_counts, group\_percentages)]  
 labels = np.asarray(labels).reshape(2,2)  
  
 plt.subplots(figsize = (5,3))  
 ax = sns.heatmap(cm/np.sum(cm), annot=labels, fmt="", cmap='Blues')  
 ax.set\_xlabel('Predicted Class')  
 ax.set\_ylabel('Actual Class')  
 ax.xaxis.set\_ticklabels(['1', '0'])  
 ax.yaxis.set\_ticklabels(['1', '0'])  
 plt.show()  
  
# plot the confusion matrix as a heatmap  
plot\_confusion\_matrix(y\_test, y\_pred)  
  
from sklearn.metrics import balanced\_accuracy\_score  
# balanced-accuracy = (tpr+tnr)/2  
print('balanced-accuracy:%.3f'%balanced\_accuracy\_score(y\_test, y\_pred))

Classification report:  
 precision recall f1-score support  
  
 0 1.00 0.96 0.98 2185  
 1 0.37 0.94 0.53 52  
  
 accuracy 0.96 2237  
 macro avg 0.68 0.95 0.75 2237  
weighted avg 0.98 0.96 0.97 2237



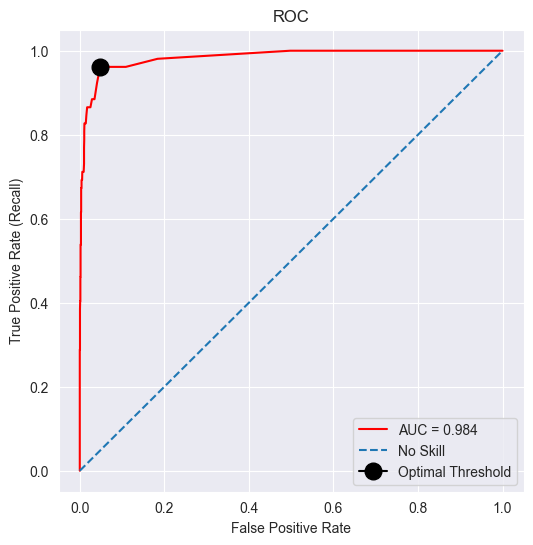
balanced-accuracy:0.952

## Choose model according to the highest precision

model=RandomForestClassifier()  
  
steps=[('t1',MinMaxScaler()),('t2',PowerTransformer()),('m',model)]  
pipeline=Pipeline(steps=steps)  
  
pipeline.fit(X\_train,y\_train)  
y\_probs=pipeline.predict\_proba(X\_test)[:,1]

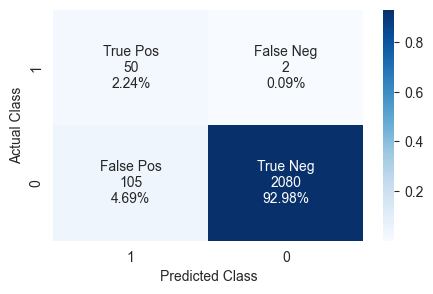
# roc curve  
from sklearn.metrics import auc  
from sklearn.metrics import roc\_curve, roc\_auc\_score  
fpr, tpr, thresholds = roc\_curve(y\_test, y\_probs)  
   
roc\_auc = auc(fpr, tpr) # roc\_auc\_score(y\_test, y\_probs)  
  
g\_means = np.sqrt(tpr\*(1-fpr))  
ix = np.argmax(g\_means)  
thresh=thresholds[ix]  
  
print('Best Threshold=%f, g\_means=%.3f' % (thresholds[ix], g\_means[ix]))  
  
plt.figure(figsize=(6,6))  
plt.title('ROC')  
plt.plot(fpr, tpr, 'r', label = 'AUC = %0.3f' % roc\_auc)  
plt.plot([0, 1], [0, 1],'--',label='No Skill')  
plt.plot(fpr[ix],tpr[ix],marker='o', markersize=12,color='black',label='Optimal Threshold');  
plt.legend(loc = 'lower right')  
plt.ylabel('True Positive Rate (Recall)')  
plt.xlabel('False Positive Rate')  
plt.show()

Best Threshold=0.050000, g\_means=0.957



# threshold moving  
from numpy import flip  
from sklearn.metrics import classification\_report, confusion\_matrix  
  
y\_pred=(y\_probs>=thresh) #   
  
print(f"Classification report:\n{classification\_report(y\_test, y\_pred)}")  
  
# Define a function to plot the confusion matrix as a heatmap  
def plot\_confusion\_matrix(y\_true, y\_prediction):  
 cm = flip(confusion\_matrix(y\_true, y\_prediction))  
 group\_names = ["True Pos", "False Neg", "False Pos", "True Neg"]  
 group\_counts = ["{0:0.0f}".format(value) for value in cm.flatten()]  
 group\_percentages = ["{0:.2%}".format(value) for value in cm.flatten()/np.sum(cm)] # flatten() is used to return a copy of a given array  
 labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in zip(group\_names, group\_counts, group\_percentages)]  
 labels = np.asarray(labels).reshape(2,2)  
  
 plt.subplots(figsize = (5,3))  
 ax = sns.heatmap(cm/np.sum(cm), annot=labels, fmt="", cmap='Blues')  
 ax.set\_xlabel('Predicted Class')  
 ax.set\_ylabel('Actual Class')  
 ax.xaxis.set\_ticklabels(['1', '0'])  
 ax.yaxis.set\_ticklabels(['1', '0'])  
 plt.show()  
  
# plot the confusion matrix as a heatmap  
plot\_confusion\_matrix(y\_test, y\_pred)  
  
from sklearn.metrics import balanced\_accuracy\_score  
# balanced-accuracy = (tpr+tnr)/2  
print('balanced-accuracy:%.3f'%balanced\_accuracy\_score(y\_test, y\_pred))

Classification report:  
 precision recall f1-score support  
  
 0 1.00 0.95 0.97 2185  
 1 0.32 0.96 0.48 52  
  
 accuracy 0.95 2237  
 macro avg 0.66 0.96 0.73 2237  
weighted avg 0.98 0.95 0.96 2237



balanced-accuracy:0.957