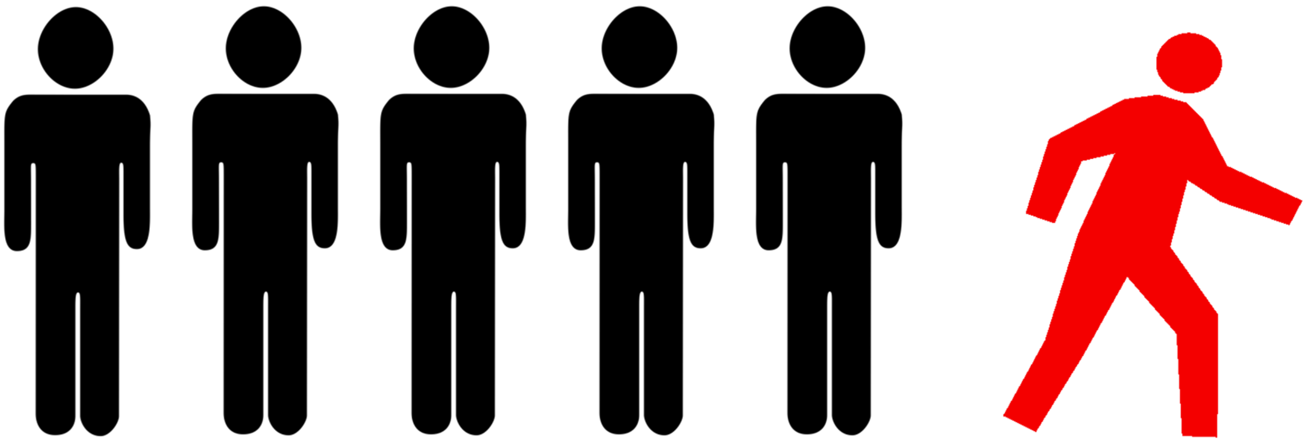
+ कोड + टेक्स्ट

BANK CUSTOMER CHURN MODEL



Learning Objective

1. Data Encoding
2. Feature Scaling
3. Handling Imblalance Data a. Random Under Sampling b. Random Over Sampling
4. Support Vector Machine Classifier
5. Grid Search for Hyperparameter Tunning

Import Library

```
import pandas as pd
```

```
import numpy as np
```

```
import matplotlib.pyplot as plt
```

```
import seaborn as sns
```

Import Data

```
df = pd.read_csv('http://github.com/YBI-Foundation/Dataset/raw/main/Bank%20Churn%20Modelling
```

```
df.head()
```

	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	Num O Product
0	15634602	Hargrave	619	France	Female	42	2	0.00	1
1	15647311	Hill	608	Spain	Female	41	1	83807.86	1
2	15619304	Onio	502	France	Female	42	8	159660.80	3
3	15701354	Boni	699	France	Female	39	1	0.00	2
4	15737888	Mitchell	850	Spain	Female	43	2	125510.82	1

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10000 entries, 15634602 to 15628319
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Surname           10000 non-null  object 
 1   CreditScore       10000 non-null  int64  
 2   Geography         10000 non-null  object 
 3   Gender            10000 non-null  object 
 4   Age               10000 non-null  int64  
 5   Tenure            10000 non-null  int64  
 6   Balance           10000 non-null  float64
 7   Num Of Products   10000 non-null  int64  
 8   Has Credit Card   10000 non-null  int64  
 9   Is Active Member  10000 non-null  int64  
 10  Estimated Salary  10000 non-null  float64
 11  Churn             10000 non-null  int64  
dtypes: float64(2), int64(7), object(3)
memory usage: 1015.6+ KB
```

```
df.duplicated('CustomerId').sum()
```

```
df = df.set_index('CustomerId')
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10000 entries, 15634602 to 15628319
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   Surname           10000 non-null  object 
 1   CreditScore       10000 non-null  int64  
 2   Geography         10000 non-null  object 
 3   Gender            10000 non-null  object 
 4   Age               10000 non-null  int64  
 5   Tenure            10000 non-null  int64  
 6   Balance           10000 non-null  float64
 7   Num Of Products   10000 non-null  int64  
 8   Has Credit Card   10000 non-null  int64  
 9   Is Active Member  10000 non-null  int64  
 10  Estimated Salary  10000 non-null  float64
 11  Churn             10000 non-null  int64  
dtypes: float64(2), int64(7), object(3)
memory usage: 1015.6+ KB
```

Encoding

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

```
France     5014
Germany    2509
Spain      2477
Name: Geography, dtype: int64
```

```
df.replace({'Geography': {'France': 2, 'Germany':1, 'Spain':0}}, inplace=True)
```

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

```
Male      5457
Female    4543
Name: Gender, dtype: int64
```

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

```
df['Num Of Products'].value_counts()
```

```
1    5084
2    4590
3     266
```

```
4      60
Name: Num Of Products, dtype: int64
```

```
df.replace({'Num Of Products': {1: 0, 2:1, 3:1, 4:1}}, inplace=True)
```

```
df['Has Credit Card'].value_counts()
```

```
1    7055
0    2945
Name: Has Credit Card, dtype: int64
```

```
df['Is Active Member'].value_counts()
```

```
1    5151
0    4849
Name: Is Active Member, dtype: int64
```

```
df.loc[(df['Balance']==0), 'Churn'].value_counts()
```

```
0    3117
1     500
Name: Churn, dtype: int64
```

```
df['Zero Balance'].hist()
```

```
df.groupby(['Churn', 'Geography']).count()
```

		Surname	CreditScore	Gender	Age	Tenure	Balance	Num Of Products	Has Credit Card
Churn	**Geography**								
0	**0**	2064	2064	2064	2064	2064	2064	2064	2064
	1	1695	1695	1695	1695	1695	1695	1695	1695
	2	4204	4204	4204	4204	4204	4204	4204	4204
1	**0**	413	413	413	413	413	413	413	413
	1	814	814	814	814	814	814	814	814
	2	810	810	810	810	810	810	810	810

Define Label and Features

```
df.columns
```

```
Index(['Surname', 'CreditScore', 'Geography', 'Gender', 'Age', 'Tenure',
       'Balance', 'Num Of Products', 'Has Credit Card', 'Is Active Member',
       'Estimated Salary', 'Churn'],
      dtype='object')
```

```
X = df.drop(['Surname','Churn'], axis = 1)
```

```
y = df['Churn']
```

```
X.shape, y.shape
```

```
((10000, 10), (10000,))
```

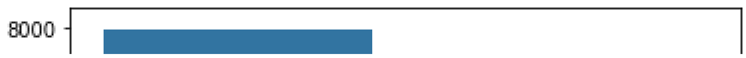
Handling Imbalance Data

Class imbalance is a common problem in machine learing, especially in classification problems as machine learning alogritms are designed to maximize accuracy and reduce errors. if the data set is imbalance then in such cases, just by predicting the majority class we get a pretty high accuracy, but fails to capture the minority class, which is most often the point of creating the modelin the first place.likein 1.fraud detection 2.spam filtering 3.disease screening 4.online sales churn 5.advertising click-throughs

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

```
0    7963
1    2037
Name: Churn, dtype: int64
```

```
sns.countplot(x ='Churn', data = df);
```




```
X.shape, y.shape
```

```
((10000, 10), (10000,))
```

Random Under Sampling

```
from imblearn.under_sampling import RandomUnderSampler
```

```
rus = RandomUnderSampler(random_state=2529)
```

```
X_rus, y_rus = rus.fit_resample(X, y)
```

```
X_rus.shape, y_rus.shape, X.shape, y.shape
```

```
((4074, 10), (4074,), (10000, 10), (10000,))
```

```
y.value_counts()
```

```
0    7963
1    2037
Name: Churn, dtype: int64
```

```
y_rus.value_counts()
```

```
0    2037
1    2037
Name: Churn, dtype: int64
```

Random Over Sampling

```
from imblearn.over_sampling import RandomOverSampler
```

```
ros = RandomOverSampler(random_state=2529)
```

```
X_ros, y_ros = ros.fit_resample(X, y)
```

```
X_ros.shape, y_ros.shape, X.shape, y.shape
```

```
((15926, 10), (15926,), (10000, 10), (10000,))
```

```
y.value_counts()
```

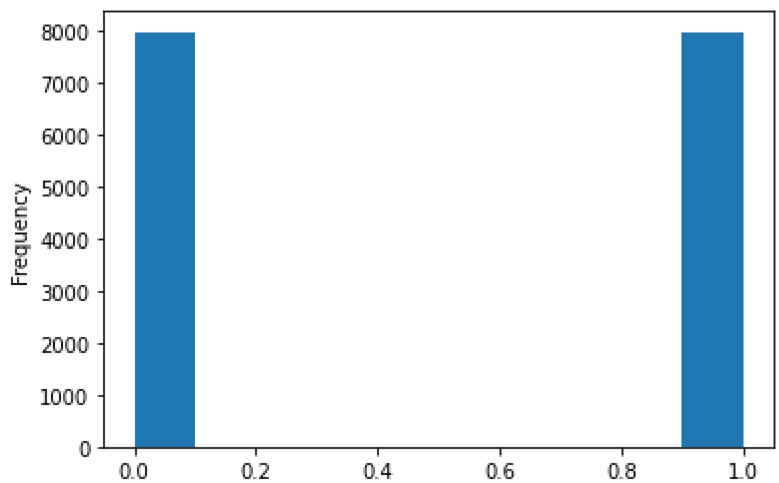
```
0    7963
1    2037
Name: Churn, dtype: int64
```

```
y_ros.value_counts()
```

```
1    7963
0    7963
Name: Churn, dtype: int64
```

```
y_ros.plot(kind = 'hist')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f90754bfad0>
```



Train Test Split

```
from sklearn.model_selection import train_test_split
```

Split Original Data

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=25)
```

Split Random Under Sampler Data

```
X_train_rus, X_test_rus, y_train_rus, y_test_rus = train_test_split(X_rus, y_rus, test_size_
```

Split Random Over Sample Data

```
X_train_ros, X_test_ros, y_train_ros, y_test_ros = train_test_split(X_ros, y_ros, test_size=
```

Standardize Feature

```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
```

Standardize Original Data

```
X_train[['CreditScore','Age','Tenure','Balance','Estimated Salary']] = sc.fit_transform(X_tr
```

```
X_test[['CreditScore','Age','Tenure','Balance','Estimated Salary']] = sc.fit_transform(X_tes
```

Standardize Random Over Sample Data

```
X_train_ros[['CreditScore','Age','Tenure','Balance','Estimated Salary']] = sc.fit_transform(
```

```
X_test_ros[['CreditScore','Age','Tenure','Balance','Estimated Salary']] = sc.fit_transform(X
```

Suport Vector Machine Classifier

```
from sklearn.svm import SVC
```

```
svc= SVC()
```

```
svc.fit(X_train, y_train)
```

```
SVC()
```

```
y_pred = svc.predict(X_test)
```

Model Accuracy

```
from sklearn.metrics import confusion_matrix, classification_report
```

```
confusion_matrix(y_test, y_pred)
```

```
    array([[2372,   47],
           [ 420,  161]])
```

```
print(classification_report(y_test, y_pred))
```

```
                  precision    recall  f1-score   support

               0       0.85      0.98      0.91      2419
               1       0.77      0.28      0.41       581

        accuracy                           0.84      3000
       macro avg       0.81      0.63      0.66      3000
    weighted avg       0.83      0.84      0.81      3000
```

```
from sklearn.model_selection import GridSearchCV
```

```
param_grid = {'C': [0.1,1, 10],
              'gamma': [1,0.1,0.01],
              'kernel': ['rbf'],
              'class_weight': ['balanced']}
```

```
grid = GridSearchCV(SVC(),param_grid,refit=True,verbose=2, cv = 2)
grid.fit(X_train,y_train)
```

```
    Fitting 2 folds for each of 9 candidates, totalling 18 fits
    [CV] END ..C=0.1, class_weight=balanced, gamma=1, kernel=rbf; total time=   2.6s
    [CV] END ..C=0.1, class_weight=balanced, gamma=1, kernel=rbf; total time=   1.6s
    [CV] END C=0.1, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   1.2s
    [CV] END C=0.1, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   1.1s
    [CV] END C=0.1, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   1.3s
    [CV] END C=0.1, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   1.3s
    [CV] END ....C=1, class_weight=balanced, gamma=1, kernel=rbf; total time=   1.4s
    [CV] END ....C=1, class_weight=balanced, gamma=1, kernel=rbf; total time=   1.4s
    [CV] END ..C=1, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   1.1s
    [CV] END ..C=1, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   1.0s
    [CV] END .C=1, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   1.1s
```

```
[CV] END .C=1, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   1.1s
[CV] END ...C=10, class_weight=balanced, gamma=1, kernel=rbf; total time=   1.3s
[CV] END ...C=10, class_weight=balanced, gamma=1, kernel=rbf; total time=   1.3s
[CV] END .C=10, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   1.1s
[CV] END .C=10, class_weight=balanced, gamma=0.1, kernel=rbf; total time=   1.1s
[CV] END C=10, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   1.1s
[CV] END C=10, class_weight=balanced, gamma=0.01, kernel=rbf; total time=   1.1s
GridSearchCV(cv=2, estimator=SVC(),
             param_grid={'C': [0.1, 1, 10], 'class_weight': ['balanced'],
                         'gamma': [1, 0.1, 0.01], 'kernel': ['rbf']},
             verbose=2)
```

```
print(grid.best_estimator_)
```

```
SVC(C=10, class_weight='balanced', gamma=1)
```

```
grid_predictions = grid.predict(X_test)
```

```
confusion_matrix(y_test,grid_predictions)
```

```
array([[2166,  253],
       [ 362,  219]])
```

```
print(classification_report(y_test,grid_predictions))
```

```
              precision    recall  f1-score   support

           0       0.86      0.90      0.88      2419
           1       0.46      0.38      0.42       581

    accuracy                           0.80      3000
   macro avg       0.66      0.64      0.65      3000
weighted avg       0.78      0.80      0.79      3000
```

Model with Random Under Sampling

```
svc_rus = SVC()
```

```
svc_rus.fit(X_train_rus, y-train_rus)
```

```
y_pred_rus = svc_rus.predict(X_test_rus)
```

Model Accuracy

```
confusion_matrix(y_test_rus, y_pred_rus)
```

```
print(classification_report(y_test_rus, y_pred_rus))
```

Hyperparameter Tunning

```
param_grid = {'C': [0.1,1, 10],
              'gamma': [1,0.1,0.01],
              'kernel': ['rbf'],
              'class_weight' : ['balance']}
```

```
grid_rus = GridSearchCV(SVC(),param_grid,refit=True,verbose=2, cv = 2)
grid_rus.fit(X_train_rus,y_train_rus)
```

```
print(grid_rus.best_estimator_)
```

```
grid_predictions_rus = grid_rus.predict(X_test_rus)
```

Model with Random Over Sampling

```
svc_ros = SVC()
```

```
svc_ros.fit(X_train_ros, y_train_ros)
```

```
y_pred_ros = svc__ros.predict(X_test_ros)
```

Model Accuracy

```
confusion_matrix(y_test_ros, y_pred_ros)
```

Hyperparameter Tunning

```
param_grid = {'C': [0.1,1, 10],
```

```
              'gamma': [1,0.1,0.01],
              'kernel': ['rbf'],
              'class_weight' : ['branched']}
```

```
grid_ros = GridSearchCV(SVC(),param_grid,refit=True,verbose=2, cv = 2)
grid_ros.fit(X_train_ros,y_train_ros)
```

```
print(grid_ros.best_estimator_)
```

```
grid_predictions_ros = grid_ros.predict(X_test_ros)
```

```
confusion_matrix(y_test_ros,grid_predictions_ros)
```

```
print(classification_report(y_test_ros,grid_predictions_ros))
```

```
print(classification_report(y_test_rus,grid_predictions_rus))
```

0 से° 11:43 pm पर पूरा किया गया