Chapter 9 - Exercise 3: Bank

- Sử dụng tập dữ liệu bank.csv chứa thông tin liên quan đến các chiến dịch tiếp thị trực tiếp - the direct marketing campaigns (dựa trên các cuộc gọi điện thoại) của một tổ chức ngân hàng Bồ Đào Nha. Thông thường, cần có nhiều contact cho cùng một khách hàng, để truy cập xem liệu có sản phẩm (tiền gửi ngân hàng có kỳ hạn - bank term deposit) sẽ được đăng ký (yes) hay không (no). Tập dữ liệu chứa một số thông tin khách hàng (như age, job...) và thông tin liên quan đến chiến dịch (chẳng hạn như contact hoặc communication type, day, month và duration của contact...).
- Đối với chiến dịch tiếp thị tiếp theo, công ty muốn sử dụng dữ liệu này và chỉ liên hệ với những khách hàng tiềm năng sẽ đăng ký tiền gửi có kỳ hạn, do đó giảm bớt nỗ lực cần thiết để liên hệ với những khách hàng không quan tâm. Để làm được điều này, cần tạo một mô hình có thể dự đoán liệu khách hàng có đăng ký tiền gửi có kỳ hạn hay không (y).

Yêu cầu: Làm lại bài Bank có:

- Áp dụng Cross Validation
- Áp dụng Grid Search và Random Search

Gợi ý:

In [1]:

```
import warnings
warnings.filterwarnings('ignore')
from sklearn.metrics import classification_report,confusion_matrix,accuracy_scor
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import StandardScaler, RobustScaler, MinMaxScaler
from collections import Counter
```

```
Using TensorFlow backend.
```

In [2]:

```python
# Đọc dữ liệu. Tìm hiểu sơ bộ về dữ liệu
bank = pd.read_csv('bank.csv', sep = ';')
bank.head()
```

Out[2]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	du
0	30	unemployed	married	primary	no	1787	no	no	cellular	19	oct	
1	33	services	married	secondary	no	4789	yes	yes	cellular	11	may	
2	35	management	single	tertiary	no	1350	yes	no	cellular	16	apr	
3	30	management	married	tertiary	no	1476	yes	yes	unknown	3	jun	
4	59	blue-collar	married	secondary	no	0	yes	no	unknown	5	may	

In [3]:

```python
bank['y']=bank['y'].replace({'no': 0, 'yes': 1})
```

In [4]:

```python
bank['month'].replace(['jan', 'feb', 'mar','apr',
                       'may','jun','jul','aug',
                       'sep','oct','nov','dec'],
                      [1,2,3,4,5,6,7,8,9,10,11,12],
                      inplace  = True)
```

In [5]:

```python
bank.shape
```

Out[5]:

```
(4334, 17)
```

In [6]:

```python
bank.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4334 entries, 0 to 4333
Data columns (total 17 columns):
age          4334 non-null int64
job          4334 non-null object
marital      4334 non-null object
education    4334 non-null object
default      4334 non-null object
balance      4334 non-null int64
housing      4334 non-null object
loan         4334 non-null object
contact      4334 non-null object
day          4334 non-null int64
month        4334 non-null int64
duration     4334 non-null int64
campaign     4334 non-null int64
pdays        4334 non-null int64
previous     4334 non-null int64
poutcome     4334 non-null object
y            4334 non-null int64
dtypes: int64(9), object(8)
memory usage: 575.7+ KB
```

In [7]:

```python
# Kiểm tra dữ liệu null
print(bank.isnull().sum())
# => Không có dữ liệu null
```

```
age          0
job          0
marital      0
education    0
default      0
balance      0
housing      0
loan         0
contact      0
day          0
month        0
duration     0
campaign     0
pdays        0
previous     0
poutcome     0
y            0
dtype: int64
```

In [8]:

```python
bank.describe()
```

Out[8]:

	age	balance	day	month	duration	campaign	pdays
count	4334.000000	4334.000000	4334.000000	4334.000000	4334.000000	4334.000000	4334.00000
mean	40.991924	1410.637517	15.913936	6.176050	264.544301	2.806876	39.67097
std	10.505378	3010.612091	8.216673	2.374798	260.642141	3.129682	99.93406
min	19.000000	-3313.000000	1.000000	1.000000	4.000000	1.000000	-1.00000
25%	33.000000	67.000000	9.000000	5.000000	104.000000	1.000000	-1.00000
50%	39.000000	440.000000	16.000000	6.000000	186.000000	2.000000	-1.00000
75%	48.000000	1464.000000	21.000000	8.000000	329.000000	3.000000	-1.00000
max	87.000000	71188.000000	31.000000	12.000000	3025.000000	50.000000	871.00000

In [9]:

```python
bank.describe(include=['O'])
```

Out[9]:

	job	marital	education	default	housing	loan	contact	poutcome
count	4334	4334	4334	4334	4334	4334	4334	4334
unique	12	3	3	2	2	2	3	4
top	management	married	secondary	no	yes	no	cellular	unknown
freq	942	2680	2306	4261	2476	3650	2801	3555

In [10]:
```
bank['y'].value_counts(0)
```

Out[10]:
```
0    3832
1     502
Name: y, dtype: int64
```

In [11]:
```
X = bank.drop(['y'], axis=1)
```

In [12]:
```
X.head()
```

Out[12]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	d
0	30	unemployed	married	primary	no	1787	no	no	cellular	19	10	
1	33	services	married	secondary	no	4789	yes	yes	cellular	11	5	
2	35	management	single	tertiary	no	1350	yes	no	cellular	16	4	
3	30	management	married	tertiary	no	1476	yes	yes	unknown	3	6	
4	59	blue-collar	married	secondary	no	0	yes	no	unknown	5	5	

In [13]:
```
y = bank['y']
```

In [14]:
```
# Dữ liệu có sự chênh lệch giữa 0 và 1
```

In [15]:
```
# Chuẩn hóa dữ liệu phân loại (kiểu chuỗi)
from sklearn.preprocessing import OneHotEncoder
```

In [16]:
```
ohe = OneHotEncoder()
ohe = ohe.fit(X[['job', 'marital', 'education','default',
                 'housing', 'loan','contact', 'poutcome']])
X_ohe = ohe.transform(X[['job', 'marital', 'education',
                         'default', 'housing', 'loan','contact', 'poutcome']])
```

In [17]:
```
X_ohe
```

Out[17]:
```
<4334x31 sparse matrix of type '<class 'numpy.float64'>'
        with 34672 stored elements in Compressed Sparse Row format>
```

In [18]:
```
X_ohe_new = X_ohe.toarray()
```

In [19]:
```
ohe.get_feature_names(['job', 'marital', 'education','default',
                       'housing', 'loan','contact', 'poutcome'])
```

Out[19]:
```
array(['job_admin.', 'job_blue-collar', 'job_entrepreneur',
       'job_housemaid', 'job_management', 'job_retired',
       'job_self-employed', 'job_services', 'job_student',
       'job_technician', 'job_unemployed', 'job_unknown',
       'marital_divorced', 'marital_married', 'marital_single',
       'education_primary', 'education_secondary', 'education_tertiary',
       'default_no', 'default_yes', 'housing_no', 'housing_yes',
       'loan_no', 'loan_yes', 'contact_cellular', 'contact_telephone',
       'contact_unknown', 'poutcome_failure', 'poutcome_other',
       'poutcome_success', 'poutcome_unknown'], dtype=object)
```

In [20]:
```
X_ohe_new[:5]
```

Out[20]:
```
array([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 1., 0., 1.,
        0., 0., 1., 0., 1., 0., 1., 0., 1., 0., 0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 1., 0., 0.,
        1., 0., 1., 0., 0., 1., 0., 1., 1., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.,
        0., 1., 1., 0., 0., 1., 1., 0., 1., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.,
        0., 1., 1., 0., 0., 1., 0., 1., 0., 0., 1., 0., 0., 0., 1.],
       [0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0.,
        1., 0., 1., 0., 0., 1., 1., 0., 0., 0., 1., 0., 0., 0., 1.]])
```

In [21]:
```
X_ohe_df = pd.DataFrame(X_ohe_new,
                        columns=ohe.get_feature_names(['job', 'marital',
                                                       'education','default',
                                                       'housing', 'loan',
                                                       'contact', 'poutcome']))
```

In [56]:
```
X_ohe_df.head(2)
```

Out[56]:

	job_admin.	job_blue-collar	job_entrepreneur	job_housemaid	job_management	job_retired	job_self-employec
0	0.0	0.0	0.0	0.0	0.0	0.0	0.(
1	0.0	0.0	0.0	0.0	0.0	0.0	0.(

2 rows × 31 columns

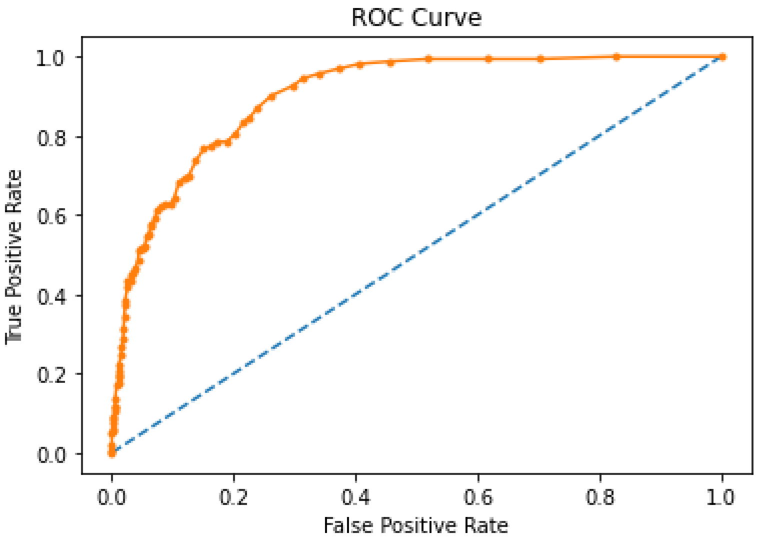
In [23]:
```
X_new = pd.concat([X[['age', 'balance', 'day','month', 'duration',
                      'campaign', 'pdays', 'previous']], X_ohe_df],
                  axis=1)
# X_new.info()
```

Cross validation

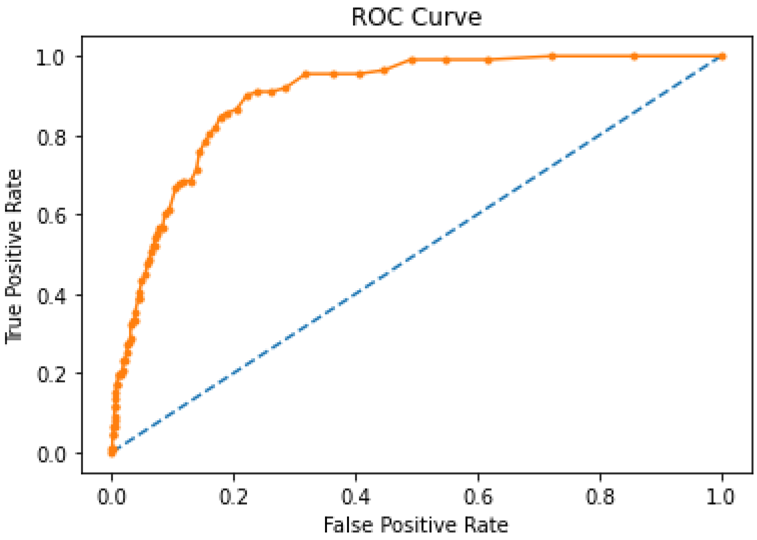
In [26]:

```
from sklearn.metrics import roc_curve,auc
# 70%, 75%, 80% training and 30%, 25%, 25% test
test_size_lst = [0.3, 0.25, 0.2]
for i in test_size_lst:
    print("***** With [", 1-i, ":", i, "] *****")
    X_train_1, X_test_1, y_train_1, y_test_1 = train_test_split(X_new, y,
                                                                test_size=i)
    model= RandomForestClassifier(n_estimators=100)
    model.fit(X_train_1,y_train_1)

    score_train = model.score(X_train_1, y_train_1)
    score_test = model.score(X_test_1, y_test_1)

    print("Score train is ", round(score_train,2),
          ", score test is", round(score_test,2),
          "diff is", round(abs(score_train-score_test),2))

    # Đánh giá model
    y_pred_1 = model.predict(X_test_1)
    print(confusion_matrix(y_test_1, y_pred_1))
    print(classification_report(y_test_1, y_pred_1))

    probs = model.predict_proba(X_test_1)
    scores = probs[:,1]
    fpr, tpr, thresholds = roc_curve(y_test_1, scores)
    print("Auc is:", auc(fpr, tpr))

    plt.plot([0, 1], [0, 1], linestyle='--')
    plt.plot(fpr, tpr, marker='.')
    plt.title("ROC Curve")
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.show()
```

```
***** With [ 0.7 : 0.3 ] *****
Score train is  1.0 , score test is 0.89 diff is 0.11
[[1121   17]
 [ 123   40]]
              precision    recall  f1-score   support

           0       0.90      0.99      0.94      1138
           1       0.70      0.25      0.36       163

    accuracy                           0.89      1301
   macro avg       0.80      0.62      0.65      1301
weighted avg       0.88      0.89      0.87      1301

Auc is: 0.9019321379667268
```

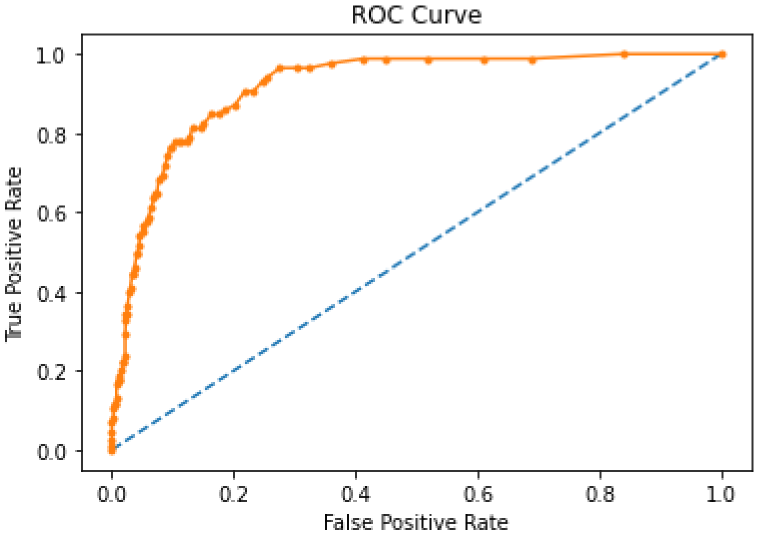


```
***** With [ 0.75 : 0.25 ] *****
Score train is  1.0 , score test is 0.9 diff is 0.1
[[949  24]
 [ 83  28]]
              precision    recall  f1-score   support

           0       0.92      0.98      0.95       973
           1       0.54      0.25      0.34       111

    accuracy                           0.90      1084
   macro avg       0.73      0.61      0.65      1084
weighted avg       0.88      0.90      0.88      1084

Auc is: 0.8979750562484375
```



```
***** With [ 0.8 : 0.2 ] *****
Score train is  1.0 , score test is 0.91 diff is 0.09
[[763  19]
 [ 56  29]]
              precision    recall  f1-score   support

           0       0.93      0.98      0.95       782
           1       0.60      0.34      0.44        85
```

```
    accuracy                           0.91       867
   macro avg       0.77      0.66      0.69       867
weighted avg       0.90      0.91      0.90       867

Auc is: 0.917459004061983
```



In [27]:

```
# Compare: 70%-30%, 75%-25% and 80%-20%
# Choose the best one
# (Can run many times to make sure your choice)
```

K-folds

In [28]:

```
from sklearn import model_selection
from sklearn.model_selection import KFold
```

In [29]:

```
model2 = RandomForestClassifier(n_estimators=100)
kfold = KFold(n_splits=10, random_state=42)
results = model_selection.cross_val_score(model2, X_new, y, cv=kfold)
print("Accuracy: %.3f%% (%.3f%%)" % (results.mean()*100.0,
                                     results.std()*100.0))
```

```
Accuracy: 90.240% (1.101%)
```

In [30]:

```
results
# Nhận xét: Model có tính ổn định khá tốt.
```

Out[30]:

```
array([0.89400922, 0.91474654, 0.88940092, 0.9078341 , 0.88452656,
       0.90993072, 0.8960739 , 0.92147806, 0.90531178, 0.90069284])
```

GridSearchCV

In [31]:
```
X_train, X_test, y_train, y_test = train_test_split(X_new, y,
                                                            test_size=i)
```

In [32]:
```
## Split 70-30
from sklearn.model_selection import GridSearchCV
```

In [33]:
```
param_grid = {
    'n_estimators': [20, 50, 100, 150, 200],
    'max_features': ['auto', 'sqrt', 'log2'],
    'min_samples_split': [2, 3, 4, 5, 6, 7, 8, 9, 10],
    'random_state': [0, 1, 42]
}
```

In [34]:
```
import datetime
x1 = datetime.datetime.now()
print(x1)
```

```
2020-10-13 09:55:47.691893
```

In [35]:
```
CV_model = GridSearchCV(estimator=RandomForestClassifier(),
                      param_grid=param_grid,
                      cv= 5)
```

In [36]:
```
CV_model.fit(X_train, y_train)
```

Out[36]:
```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=RandomForestClassifier(bootstrap=True, class_weight=Non
e,
                                              criterion='gini', max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators='warn', n_jobs=None,
                                              oob_score=False,
                                              random_state=None, verbose=0,
                                              warm_start=False),
             iid='warn', n_jobs=None,
             param_grid={'max_features': ['auto', 'sqrt', 'log2'],
                         'min_samples_split': [2, 3, 4, 5, 6, 7, 8, 9, 10],
                         'n_estimators': [20, 50, 100, 150, 200],
                         'random_state': [0, 1, 42]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

In [37]:
```
print(CV_model.best_params_)
```

```
{'max_features': 'log2', 'min_samples_split': 2, 'n_estimators': 20, 'random_st
ate': 0}
```

In [ ]:

In [38]:
```
x2 = datetime.datetime.now()
print(x2)
```

```
2020-10-13 10:04:59.457826
```

In [39]:
```
d = x2 - x1
print(d)
```

```
0:09:11.765933
```

In [40]:
```
y_pred3=CV_model.predict(X_test)
```

In [41]:
```
print("Accuracy:", accuracy_score(y_test, y_pred3))
```

```
Accuracy: 0.9042675893886967
```

In [42]:
```
# Kiểm tra độ chính xác
print("The Training R^2 score is: ",
      CV_model.score(X_train,y_train)*100,"%")
print("The Testing R^2 score is: ",
      CV_model.score(X_test,y_test)*100,"%")
```

```
The Training R^2 score is:  99.71156619555812 %
The Testing R^2 score is:  90.42675893886967 %
```

In [43]:
```
print(confusion_matrix(y_test, y_pred3))
print(classification_report(y_test, y_pred3))
```

```
[[767   8]
 [ 75  17]]
              precision    recall  f1-score   support

           0       0.91      0.99      0.95       775
           1       0.68      0.18      0.29        92

    accuracy                           0.90       867
   macro avg       0.80      0.59      0.62       867
weighted avg       0.89      0.90      0.88       867
```

Random Search

In [44]:
```
from sklearn.model_selection import RandomizedSearchCV
```

In [45]:
```
param_dist = {'n_estimators': [20, 50, 100, 150, 200],
    'max_features': ['auto', 'sqrt', 'log2'],
    'min_samples_split': [2, 3, 4, 5, 6, 7, 8, 9, 10],
    'random_state': [0, 1, 42]
}
```

In [46]:

```
x1 = datetime.datetime.now()
print(x1)
```

```
2020-10-13 10:04:59.561781
```

In [47]:

```
forest_random = RandomizedSearchCV(estimator=RandomForestClassifier(),
                                   param_distributions=param_dist,
                                   cv=5)
```

In [48]:

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

Out[48]:

```
RandomizedSearchCV(cv=5, error_score='raise-deprecating',
                   estimator=RandomForestClassifier(bootstrap=True,
                                                    class_weight=None,
                                                    criterion='gini',
                                                    max_depth=None,
                                                    max_features='auto',
                                                    max_leaf_nodes=None,
                                                    min_impurity_decrease=0.0,
                                                    min_impurity_split=None,
                                                    min_samples_leaf=1,
                                                    min_samples_split=2,
                                                    min_weight_fraction_leaf=0.
0,
                                                    n_estimators='warn',
                                                    n_jobs=None,
                                                    oob_score=False,
                                                    random_state=None,
                                                    verbose=0,
                                                    warm_start=False),
                   iid='warn', n_iter=10, n_jobs=None,
                   param_distributions={'max_features': ['auto', 'sqrt',
                                                         'log2'],
                                        'min_samples_split': [2, 3, 4, 5, 6, 7,
                                                              8, 9, 10],
                                        'n_estimators': [20, 50, 100, 150, 20
0],
                                        'random_state': [0, 1, 42]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring=None, verbose=0)
```

In [49]:

```
forest_random_best = forest_random.best_estimator_
print("Best Model Parameter: ",forest_random.best_params_)
```

```
Best Model Parameter:  {'random_state': 1, 'n_estimators': 150, 'min_samples_sp
lit': 6, 'max_features': 'log2'}
```

In [50]:

```
x2 = datetime.datetime.now()
print(x2)
```

```
2020-10-13 10:05:14.014603
```

In [51]:

```
d = x2-x1
print(d)
```

```
0:00:14.452822
```

In [52]:

```
y_pred4 = forest_random.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred4))
```

```
Accuracy: 0.9008073817762399
```

In [53]:

```
# Kiểm tra độ chính xác
print("The Training R^2 score is: ",
      forest_random.score(X_train,y_train)*100,"%")
print("The Testing R^2 score is: ",
      forest_random.score(X_test,y_test)*100,"%")
```

```
The Training R^2 score is:  97.05797519469282 %
The Testing R^2 score is:  90.08073817762399 %
```

In [54]:

```
print(confusion_matrix(y_test, y_pred4))
print(classification_report(y_test, y_pred4))
```

```
[[761  14]
 [ 72  20]]
              precision    recall  f1-score   support

           0       0.91      0.98      0.95       775
           1       0.59      0.22      0.32        92

    accuracy                           0.90       867
   macro avg       0.75      0.60      0.63       867
weighted avg       0.88      0.90      0.88       867
```

In [55]:

```
# Model mất cân bằng dữ liệu dẫn đến kết quả không được tốt.
# Tìm giải pháp để cải thiện kết quả.
```