Module : Applied Statistics and Machine Learning (B9BA102_2122_TMD1S)

Under the Guidance : Prof. Kunwar Madan

Submitted by :- Rohan Sapkal

Student ID: 10592020

INDEX

1. Problem Statement:

For classification, the data that is used is Portuguese bank marketing data. It took several phone calls to determine whether a product (bank term deposit) would be subscribed by the same client. It has one target feature that is “Subscription”. The target feature contains two instances ‘1’ and ‘2’. ‘1’ means yes and ‘2’ means no.

To solve the classification problem, need to use the random forest and support vector machine classification models in Python.

2. Data Information:

Dataset is downloaded from Kaggle and below is a link for the dataset: https://www.kaggle.com/aakashverma8900/portuguese-bank-marketing

There are 16 independent features in the dataset and one target feature in total there are 17 features and 45212 instances.

16 independent variables:

1. Age:
   Datatype: int64
   Numeric

2. Job:
   Datatype: object
   Categorical: 'management', 'technician', 'entrepreneur', 'blue-collar', 'unknown', 'retired', 'admin.', 'services', 'self-employed', 'unemployed', 'housemaid', 'student'

3. Marital Status:
   Datatype: object
   Categorical: 'married', 'single', 'divorced'

4. Education:
   Datatype: object
   Categorical: 'tertiary', 'secondary', 'unknown', 'primary'

5. Credit:
   Datatype: object
   Binary: 'no', 'yes'

6. Balance (euros):
   Datatype: int64
   Numeric

7. Housing Loan:
   Datatype: object
   Binary: 'no', 'yes'

8. Personal Loan:
   Datatype: object
   Binary: 'no', 'yes'

9. Contact:
   Datatype: object
   Categorical: 'unknown', 'cellular', 'telephone'

10. Last Contact Day:
    Datatype: int64
    Numeric

11. Last Contact Month:
    Datatype: object
    Categorical: 'may', 'jun', 'jul', 'aug', 'oct', 'nov', 'dec', 'jan', 'feb', 'mar', 'apr', 'sep'

12. Last Contact Duration:
    Datatype: int64
    Numeric

13. Campaign:
    Datatype: int64
    Numeric

14. Pdays:
    Datatype: int64
    Numeric

15. Previous:
    Datatype: int64
    Numeric

16. Poutcome:
    Datatype: object
    Categorical: 'unknown', 'failure', 'other', 'success'

One target variable:

17. Subscription:
    Datatype: int64
    Numeric (1, 2)

3. Data cleaning and pre-processing:

1. Imported the dataset and checked the dataset if there are any outliers and error values and null values.

2. There are outliers in the dataset and some error values such as 'unknown'.

	Age	Balance (euros)	Last Contact Day	Last Contact Duration	\
count	45211.000000	45211.000000	45211.000000	45211.000000	
mean	40.936210	1362.272058	15.806419	258.163080	
std	10.618762	3044.765829	8.322476	257.527812	
min	18.000000	-8019.000000	1.000000	0.000000	
25%	33.000000	72.000000	8.000000	103.000000	
50%	39.000000	448.000000	16.000000	180.000000	
75%	48.000000	1428.000000	21.000000	319.000000	
max	95.000000	102127.000000	31.000000	4918.000000	

	Campaign	Pdays	Previous	Subscription
count	45211.000000	45211.000000	45211.000000	45211.000000
mean	2.763841	40.197828	0.580323	1.116985
std	3.098021	100.128746	2.303441	0.321406
min	1.000000	-1.000000	0.000000	1.000000
25%	1.000000	-1.000000	0.000000	1.000000
50%	2.000000	-1.000000	0.000000	1.000000
75%	3.000000	-1.000000	0.000000	1.000000
max	63.000000	871.000000	275.000000	2.000000

3. Need to clean the unknown values. To clean error values replace them with null values.

4. Drop the null. After dropping the null values, we have 7842 instances.

5. After dropping null values, checked for the duplicate values and didn't find any duplicate values.

6. As after cleaning the dataset now, it's time for data pre-processing.

7. Observing the clean dataset, found that there are some object datatypes for classification problems that need to covert it in integer datatype.

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7842 entries, 24060 to 45210
Data columns (total 17 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   Age                    7842 non-null   int64
 1   Job                    7842 non-null   object
 2   Marital Status         7842 non-null   object
 3   Education              7842 non-null   object
 4   Credit                 7842 non-null   object
 5   Balance (euros)        7842 non-null   int64
 6   Housing Loan           7842 non-null   object
 7   Personal Loan          7842 non-null   object
 8   Contact                7842 non-null   object
 9   Last Contact Day       7842 non-null   int64
 10  Last Contact Month     7842 non-null   object
 11  Last Contact Duration  7842 non-null   int64
 12  Campaign               7842 non-null   int64
 13  Pdays                  7842 non-null   int64
 14  Previous               7842 non-null   int64
 15  Poutcome               7842 non-null   object
 16  Subscription           7842 non-null   int64
dtypes: int64(8), object(9)
memory usage: 1.1+ MB
```

8. To convert an object to integer datatype, the use of map function and get_dummies function is made.

9. Map() is used on the below columns as mentioned columns had two values that can map to ‘0’ & ’1’.
   Credit, Housing Loan, Contact.

10. Used map() on Last Contact Month columns as it ordinal. It contains on months so we can order them from ‘0’ to ‘11’

11. For the remaining columns, need to use the get_dummies() as they are not having two values or are ordinal. They are called nominal columns. Below are the columns which are converted using the get_dummies().

12. Dataset after converting the datatype.

```
(7842, 33)
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7842 entries, 24060 to 45210
Data columns (total 33 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Age                      7842 non-null   int64
 1   Credit                   7842 non-null   int64
 2   Balance (euros)          7842 non-null   int64
 3   Housing Loan             7842 non-null   int64
 4   Personal Loan            7842 non-null   int64
 5   Contact                  7842 non-null   int64
 6   Last Contact Day         7842 non-null   int64
 7   Last Contact Month       7842 non-null   int64
 8   Last Contact Duration    7842 non-null   int64
 9   Campaign                 7842 non-null   int64
 10  Pdays                    7842 non-null   int64
 11  Previous                 7842 non-null   int64
 12  Subscription             7842 non-null   int64
 13  Job_admin.               7842 non-null   uint8
 14  Job_blue-collar          7842 non-null   uint8
 15  Job_entrepreneur         7842 non-null   uint8
 16  Job_housemaid            7842 non-null   uint8
 17  Job_management           7842 non-null   uint8
 18  Job_retired              7842 non-null   uint8
 19  Job_self-employed        7842 non-null   uint8
 20  Job_services             7842 non-null   uint8
 21  Job_student              7842 non-null   uint8
 22  Job_technician           7842 non-null   uint8
 23  Job_unemployed           7842 non-null   uint8
 24  Marital Status_divorced  7842 non-null   uint8
 25  Marital Status_married   7842 non-null   uint8
 26  Marital Status_single    7842 non-null   uint8
 27  Education_primary        7842 non-null   uint8
 28  Education_secondary      7842 non-null   uint8
 29  Education_tertiary       7842 non-null   uint8
 30  Poutcome_failure         7842 non-null   uint8
 31  Poutcome_other           7842 non-null   uint8
 32  Poutcome_success         7842 non-null   uint8
dtypes: int64(13), uint8(20)
```

13. Determine X and Y:

    X: It contains all independent features and the target feature [Subscription] is dropped.

    Y: It only contains the target feature [Subscription].

14. StandardScaler(): As there is an outlier in independent features to reduce it and normalize the numerical features so that each feature has to mean 0 and variance 1.

```
print(X_scaled)
```

```
[[-0.6899209  -0.084808   -0.21733462 ...  0.82219167 -0.53596827
  -0.4688127 ]
 [ 0.10779272 -0.084808   -0.58337213 ... -1.21626141  1.8657821
  -0.4688127 ]
 [-0.6899209  -0.084808    0.61330144 ...  0.82219167 -0.53596827
  -0.4688127 ]
 ...
 [ 2.85547297 -0.084808    0.42071837 ...  0.82219167 -0.53596827
  -0.4688127 ]
 [ 2.76683812 -0.084808    1.34959124 ... -1.21626141 -0.53596827
   2.13304802]
 [-0.33538151 -0.084808    0.45994825 ... -1.21626141  1.8657821
  -0.4688127 ]]
```

4. Random Forest Classification model:

Imported Depedencies

- Imported dependecies are:
- from sklearn.ensemble import RandomForestClassifier
- from sklearn.svm import SVC
- from sklearn.model_selection import train_test_split

Split data

- Splited the data into train and test using dependeny train_test_split()

Buliding and Trainning model

- Build the random forest model
- Train the model by using train data.

Testing model

- Tested Random Forest model by using test data.

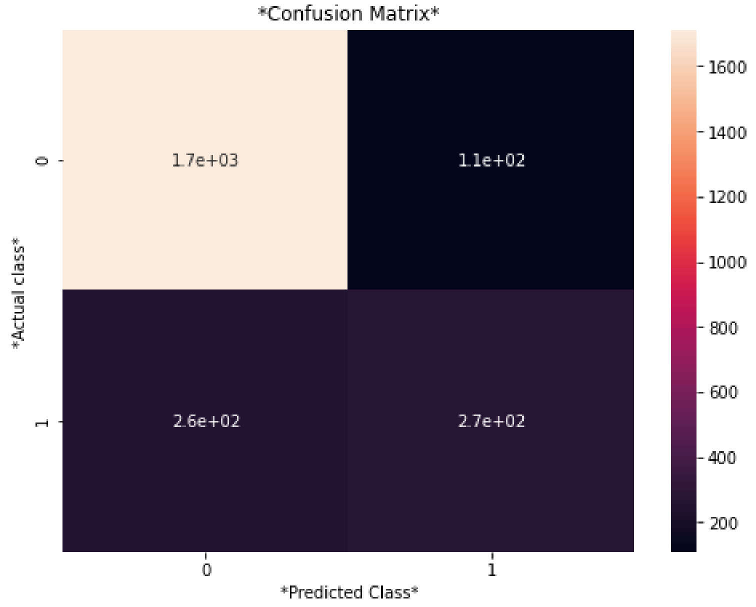
Constructed Confusion matrix

- Constructed matrix and plotted it in graph

4.1. Random Forest classification with default parameter:

After constructing the confusion matrix, find that for the better performance of the model need to minimize the false negative. Below is the confusion matrix of the random forest:

```
Accuracy Score of Random forest: 0.844028899277518
```




```
Confusion matrix for Random Forest:
 [[1712  109]
 [ 258  274]]
True Positive:  274
True Negative:  1712
False Positive:  109
False Negative:  258
```

4.2. Random Forest Classification using hyperparameter tunning:

Hyperparameter is used to improve the performance of the model and can increase the speed to train the model.

```
# Implementing Random Forest Classifier
# Tuning the random forest parameter 'n_estimators' and implementing cross-validation using Grid Search
model = Pipeline([
        ('balancing', SMOTE(random_state = 101)),
        ('classification', RandomForestClassifier(criterion='entropy', max_features='auto', random_state=1) )
    ])
grid_param = {'classification__n_estimators': [10,20,30,40,50,100]}

#Calling 'recall' score to minimize false negative as in confusion matrix FN needs to reduce.
gd_sr = GridSearchCV(estimator=model, param_grid=grid_param, scoring='recall', cv=5)

gd_sr.fit(X_scaled, Y)

best_parameters = gd_sr.best_params_
print(best_parameters)

best_result = gd_sr.best_score_ # Mean cross-validated score of the best_estimator
print(best_result)
```

```
{'classification__n_estimators': 10}
0.7448697718657085
```

Pipeline(): It is used to tie data transformations together and execute them sequentially.

SMOTE(): Synthetic Minority Oversampling Technique, or SMOTE, increases the number of cases in your dataset in a balanced way.

Parameters for Random Forest Classifier:
Criterion = entropy
Max_features = auto
Random_state = 1

GridSearchCV(): Using Grid Search, all the specified hyperparameters can be combined with different values, and then the performance of each combination is calculated, and the best hyperparameter value is selected.

Parameters for Grid search:
Estimator = Contains random forest model with mentioned parameter.
Param_grid = Contains Classification_n_estimators list that contains small values and also tried to use large value above 150 but the score was low compare to these small values.
Scoring = 'recall' is used to minimize false negative.
cv = 5. It means it will use 5 cross folds.

Output:

Classification_n_estimators:10
Recall Score : 0.74

4.3. Highest significant variables:

```
# Building random forest using the tuned parameter
rfc = RandomForestClassifier(n_estimators=10, criterion='entropy', max_features='auto', random_state=1)
rfc.fit(X_scaled,Y)
featimp = pd.Series(rfc.feature_importances_, index=list(X)).sort_values(ascending=False)
print(featimp.head())
```

```
Last Contact Duration    0.215988
Poutcome_success         0.109228
Pdays                    0.108932
Balance (euros)          0.077081
Age                      0.073151
dtype: float64
```

By using feature_importances we find the most significant variables.

As the best n_estimator is 10 found in random forest hyperparameter tunning.

Output:

Last Contact Duration 0.215988
Poutcome_success 0.109228
Pdays 0.108932
Balance (euros) 0.077081
Age 0.073151

4.4. Random Forest using significant variables:

```
# Selecting features with higher significance and redefining feature set
X_ = final_data[['Last Contact Duration','Poutcome_success','Pdays','Balance (euros)']]

feature_scaler = StandardScaler()
X_scaled_ = feature_scaler.fit_transform(X_)

#Tuning the random forest parameter 'n_estimators' and implementing cross-validation using Grid Search
model = Pipeline([
        ('balancing', SMOTE(random_state = 1)),
        ('classification', RandomForestClassifier(criterion='entropy', max_features='auto', random_state=1) )
    ])
grid_param = {'classification__n_estimators': [10,20,30,40,50,100,150]}

#Calling 'recall' score to minimize false negative as in confusion matrix FN needs to reduce.
gd_sr = GridSearchCV(estimator=model, param_grid=grid_param, scoring='recall', cv=5)

gd_sr.fit(X_scaled_, Y)

best_parameters = gd_sr.best_params_
print(best_parameters)

best_result = gd_sr.best_score_ # Mean cross-validated score of the best_estimator
print(best_result)
```

```
{'classification__n_estimators': 10}
0.8001958123145098
```

Selected top 4 significant variables in the random forest hyperparameter tunning.

Pipeline(): It is used to tie data transformations together and execute them sequentially.

SMOTE(): Synthetic Minority Oversampling Technique, or SMOTE, increases the number of cases in your dataset in a balanced way.

Parameters for Random Forest Classifier:
Criterion = entropy
Max_features = auto
Random_state = 1

GridSearchCV(): Using Grid Search, all the specified hyperparameters can be combined with different values, and then the performance of each combination is calculated, and the best hyperparameter value is selected.

Parameters for Grid search:

Estimator = Contains random forest model with mentioned parameter.

Param_grid = Contains Classification_n_estimators list that contains small values and also tried to use large value above 150 but the score was low compare to these small values.

Scoring = 'recall' is used to minimize the false negative.

cv = 5. It means it will use 5 cross folds.

Output:

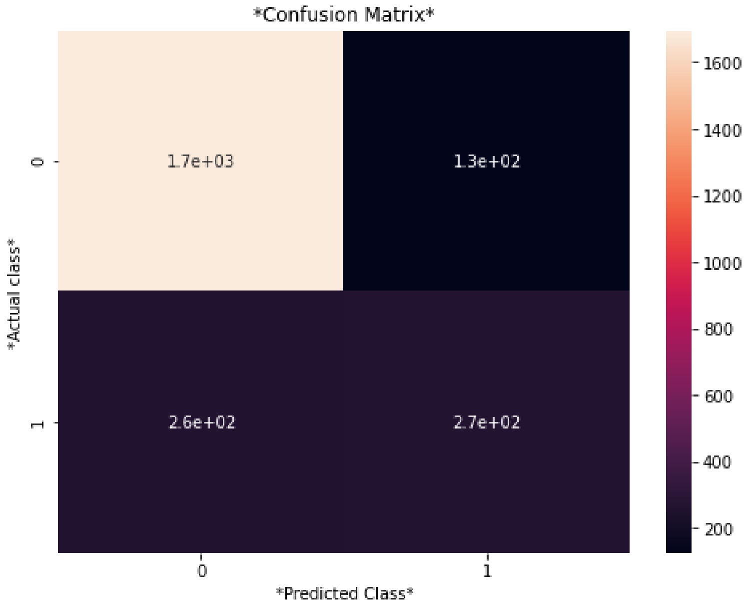
Classification_n_estimators:10

Recall Score : 0.80

5. Support Vector Machine:

5.1. Support Vector Machine with default parameter:

```
Accuracy Score of SVM: 0.8355291117722057
```




```
Confusion matrix for SVM:
 [[1695  126]
 [ 261  271]]
True Positive:  271
True Negative:  1695
False Positive:  126
False Negative:  261
```

After executing the SVM with default parameters we get an accuracy score of 0.83 but it is observed that we need to reduce the false negative in confusion matrix.

5.2. Support Vector Machine using hyperparameter tunning:

```python
# Implementing Support Vector Classifier
# Tuning the kernel parameter and implementing cross-validation using Grid Search
model = Pipeline([
        ('balancing', SMOTE(random_state = 101)),
        ('classification', SVC(random_state=1) )
    ])
grid_param = {'classification__kernel': ['linear','poly','rbf','sigmoid'], 'classification__C': [.001,.01,.1,1,10,100]}

#Calling 'recall' score to minimize false negative as in confusion matrix FN needs to reduce.
gd_sr = GridSearchCV(estimator=model, param_grid=grid_param, scoring='recall', cv=5)

gd_sr.fit(X_scaled, Y)

best_parameters = gd_sr.best_params_
print(best_parameters)

best_result = gd_sr.best_score_ # Mean cross-validated score of the best_estimator
print(best_result)
```

```
{'classification__C': 0.01, 'classification__kernel': 'poly'}
0.9065295299141806
```

Pipeline(): It is used to tie data transformations together and execute them sequentially.

SMOTE(): Synthetic Minority Oversampling Technique, or SMOTE, increases the number of cases in your dataset in a balanced way.

Parameters for SVM:
Random_state = 1

GridSearchCV(): Using Grid Search, all the specified hyperparameters can be combined with different values, and then the performance of each combination is calculated, and the best hyperparameter value is selected.

Parameters for Grid search:
Estimator = Contains random forest model with mentioned parameter.
Param_grid = Contains Classification_kernel and classification_C
Classification_kernel: liner, poly, rbf, sigmoid is used to find best.
Classification_C: Regularization parameter
Scoring = ‘recall’ is used to minimize the false negative.
cv = 5. It means it will use 5 cross folds.

Output:

classification_C : 0.01
classification_kernel: poly
Recall score: 0.90

6. Conclusion:

As per the outcome from both the model we need the model which contains of highest recall score as we want to minimize the false negative to increase the customer’s subscription in the bank new product.

Recall score of models:

1. Random Forest:
   Random Forest with hyper parameter: 0.74
   Random Forest with significant variables: 0.80
2. Support Vector Machine:
   Support Vector Machine with kernel = poly: 0.90

By observing both the models we can say that support vector machine classification is best for the business as its recall score is 0.90 that is best then the random forest classification.