Introduction to Data Science

Naive Bayes Dataset 2

Bank Term Deposit

By: Pujesh Pradhan

Table of Content

Objective and Data Description

Overview

Banks provide term deposit to their customers and to attract customers to the feature, there are various marketing campaigns conducted. One way is a direct marketing through phone calls. Based on the history of the customer and the type of customers, we can predict on the likeliness of a customer to subscribe the term deposit. This prediction helps us to focus on those clients first and give high priority during the marketing campaign.

Objective

Predict the likeliness of a customer to subscribe a term deposit.

About the Dataset

This dataset is being used from the Kaggle website and the URL to the dataset is: Bank Term Deposit

It has 16 fields and the description of each field of the dataset are:

1. **AGE** - Age of the customer.
2. **JOB** - Type of job of the customer.
3. **MARITAL** - Marital status of the customer.
4. **EDUCATION** - Education level of the customer.
5. **DEFAULT** - If the customer has credit in default.
6. **BALANCE** - Average yearly balance of the customer.
7. **HOUSING** - If the customer has housing loan.
8. **LOAN** - If the customer has personal loan.
9. **DAY** - Last contact day of the month.
10. **CONTACT** - Contact communication type.
11. **MONTH** - Last contact month of the year.
12. **DURATION** - Last contact duration (in seconds).
13. **CAMPAIGN** - Number of contacts performed during this campaign for the customer.
14. **PDAYS** - Number of days that passed by after the client was last contacted from a previous campaign.
15. **PREVIOUS** - Number of contacts performed in the previous campaign.
16. **POUTCOME** - Outcome of the previous marketing campaign.
17. **TARGET** - Whether the client subscribed a term deposit.

In [1]:
```
# All the required libraries are imported here.
import numpy as np
import pandas as pd
from pandas.plotting import scatter_matrix

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB, BernoulliNB
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.ensemble import ExtraTreesClassifier

import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
#Describing the size, shape and dimension of the dataset.
bank_data = pd.read_csv('bank_full.csv')
print("The total size of the dataset is {} bytes.".format(bank_data.size))
print("There are {} number of records and {} number of fields with a {} shape.".format(bank_data.shape[0], bank_data.shape[1], bank_data.shape))
print("The dataset has a {} dimensional structure.".format(bank_data.ndim))
```

```
The total size of the dataset is 768587 bytes.
There are 45211 number of records and 17 number of fields with a (45211, 17) shape.
The dataset has a 2 dimensional structure.
```

In [3]:
```
print("The top 5 records of the dataset are: \n{}".format(bank_data.head()))
```

```
The top 5 records of the dataset are:
   age           job  marital  education default  balance housing loan  \
0   58    management  married   tertiary      no     2143     yes   no
1   44    technician   single  secondary      no       29     yes   no
2   33  entrepreneur  married  secondary      no        2     yes  yes
3   47   blue-collar  married    unknown      no     1506     yes   no
4   33       unknown   single    unknown      no        1      no   no

   contact  day month  duration  campaign  pdays  previous poutcome Target
0  unknown    5   may       261         1     -1         0  unknown     no
1  unknown    5   may       151         1     -1         0  unknown     no
2  unknown    5   may        76         1     -1         0  unknown     no
3  unknown    5   may        92         1     -1         0  unknown     no
4  unknown    5   may       198         1     -1         0  unknown     no
```

In [4]:
```
#Listing the unique values for all the variables of object type.
for data_keys in bank_data.keys():
    if bank_data[data_keys].dtype == 'object':
        print("The unique value of {} variables are : {}".format(data_keys, list(bank_data[data_keys].unique())))
```

```
The unique value of job variables are : ['management', 'technician', 'entrepreneur', 'blue-collar', 'unknown', 'retired', 'admin.', 'services', 'self-employed', 'unempl
oyed', 'housemaid', 'student']
The unique value of marital variables are : ['married', 'single', 'divorced']
The unique value of education variables are : ['tertiary', 'secondary', 'unknown', 'primary']
The unique value of default variables are : ['no', 'yes']
The unique value of housing variables are : ['yes', 'no']
The unique value of loan variables are : ['no', 'yes']
The unique value of contact variables are : ['unknown', 'cellular', 'telephone']
The unique value of month variables are : ['may', 'jun', 'jul', 'aug', 'oct', 'nov', 'dec', 'jan', 'feb', 'mar', 'apr', 'sep']
The unique value of poutcome variables are : ['unknown', 'failure', 'other', 'success']
The unique value of Target variables are : ['no', 'yes']
```

In [5]:
```
#Each categorical variables of object types are replaced with numberical numberes
for data_keys in bank_data.keys():
    if bank_data[data_keys].dtype == 'object':
        print("{}:".format(data_keys))
        print("Before values: {}".format(bank_data[data_keys].unique()))
        uniq_value = list(bank_data[data_keys].unique())
        bank_data[data_keys] = bank_data[data_keys].replace(uniq_value, list(np.arange(len(uniq_value))))
        print("After values: {}\n".format(list(bank_data[data_keys].unique())))
```

```
job:
Before values: ['management' 'technician' 'entrepreneur' 'blue-collar' 'unknown'
 'retired' 'admin.' 'services' 'self-employed' 'unemployed' 'housemaid'
 'student']
After values: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

marital:
Before values: ['married' 'single' 'divorced']
After values: [0, 1, 2]

education:
Before values: ['tertiary' 'secondary' 'unknown' 'primary']
After values: [0, 1, 2, 3]

default:
Before values: ['no' 'yes']
After values: [0, 1]

housing:
Before values: ['yes' 'no']
After values: [0, 1]

loan:
Before values: ['no' 'yes']
After values: [0, 1]

contact:
Before values: ['unknown' 'cellular' 'telephone']
After values: [0, 1, 2]

month:
Before values: ['may' 'jun' 'jul' 'aug' 'oct' 'nov' 'dec' 'jan' 'feb' 'mar' 'apr' 'sep']
After values: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

poutcome:
Before values: ['unknown' 'failure' 'other' 'success']
After values: [0, 1, 2, 3]

Target:
Before values: ['no' 'yes']
After values: [0, 1]
```

In [6]:
```
#The updated dataset looks like this:
bank_data.head()
```

Out[6]:

	age	job	marital	education	default	balance	housing	loan	contact	day	month	duration	campaign	pdays	previous	poutcome	Target
0	58	0	0	0	0	2143	0	0	0	5	0	261	1	-1	0	0	0
1	44	1	1	1	0	29	0	0	0	5	0	151	1	-1	0	0	0
2	33	2	0	1	0	2	0	1	0	5	0	76	1	-1	0	0	0
3	47	3	0	2	0	1506	0	0	0	5	0	92	1	-1	0	0	0
4	33	4	1	2	0	1	1	0	0	5	0	198	1	-1	0	0	0

In [7]:
```
#Finding the number of null values in each field.
bank_data.isnull().sum()
```

Out[7]:
```
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
Target       0
dtype: int64
```

In [8]:
```
#Printing the data types of the variables
print("The datatype of the dataset are: \n{}".format(bank_data.dtypes))
```

```
The datatype of the dataset are:
age          int64
job          int64
marital      int64
education    int64
default      int64
balance      int64
housing      int64
loan         int64
contact      int64
day          int64
month        int64
duration     int64
campaign     int64
pdays        int64
previous     int64
poutcome     int64
Target       int64
dtype: object
```

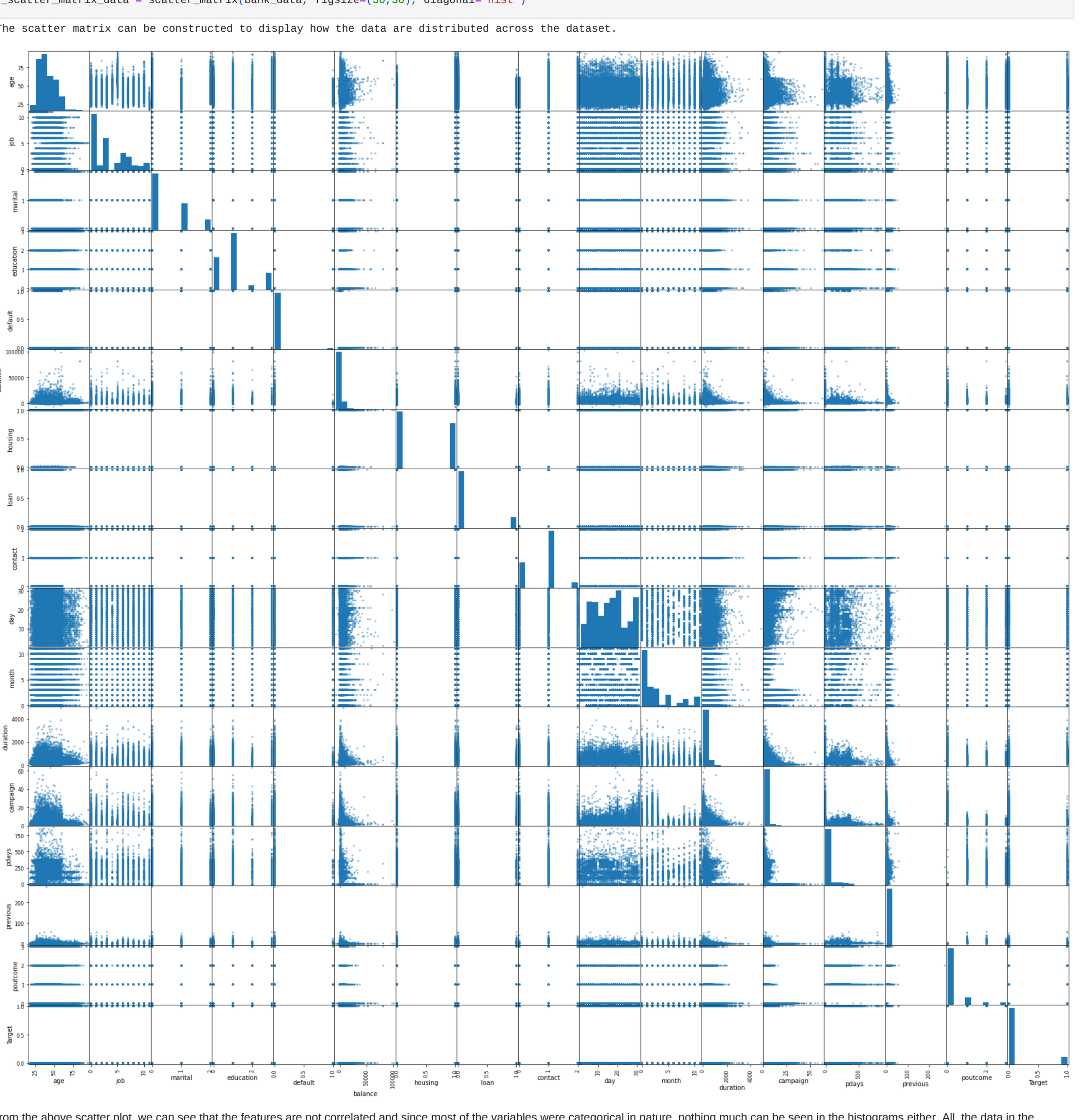
In [9]:
```
#Statistical description of the data
print(bank_data.describe().T)
```

```
             count         mean          std     min    25%    50%     75%  \
age        45211.0    40.936210    10.618762    18.0   33.0   39.0    48.0
job        45211.0     3.526133     3.098343     0.0    1.0    3.0     6.0
marital    45211.0     0.513238     0.692948     0.0    0.0    0.0     1.0
education  45211.0     1.049944     0.968986     0.0    0.0    1.0     1.0
default    45211.0     0.018027     0.133049     0.0    0.0    0.0     0.0
balance    45211.0  1362.272058  3044.765829 -8019.0   72.0  448.0  1428.0
housing    45211.0     0.444162     0.496878     0.0    0.0    0.0     1.0
loan       45211.0     0.160226     0.366820     0.0    0.0    0.0     0.0
contact    45211.0     0.776293     0.549747     0.0    0.0    1.0     1.0
day        45211.0    15.806419     8.322476     1.0    8.0   16.0    21.0
month      45211.0     2.940722     3.179741     0.0    0.0    2.0     5.0
duration   45211.0   258.163080   257.527812     0.0  103.0  180.0   319.0
campaign   45211.0     2.763841     3.098021     1.0    1.0    2.0     3.0
pdays      45211.0    40.197828   100.128746    -1.0   -1.0   -1.0    -1.0
previous   45211.0     0.580323     2.303441     0.0    0.0    0.0     0.0
poutcome   45211.0     0.290062     0.698469     0.0    0.0    0.0     0.0
Target     45211.0     0.116985     0.321406     0.0    0.0    0.0     0.0

                max
age            95.0
job            11.0
marital         2.0
education       3.0
default         1.0
balance    102127.0
housing         1.0
loan            1.0
contact         2.0
day            31.0
month          11.0
duration     4918.0
campaign       63.0
pdays         871.0
previous      275.0
poutcome        3.0
Target          1.0
```

Statistical Numerical and Graphical Summaries

In [10]:
```
print("The scatter matrix can be constructed to display how the data are distributed across the dataset.\n")

_scatter_matrix_data = scatter_matrix(bank_data, figsize=(30,30), diagonal='hist')
```

```
The scatter matrix can be constructed to display how the data are distributed across the dataset.
```



From the above scatter plot, we can see that the features are not correlated and since most of the variables were categorical in nature, nothing much can be seen in the histograms either. All, the data in the scatter plot also seems to be separated in categorical format thus not much can be seen from the diagram above.

Algorithm Implementation and Statistical Tests

In [11]:
```
#Considering all the features for classification.
pred_var = bank_data.iloc[:, 0:15]
targ_var = bank_data['Target']

X_train, X_test, y_train, y_test = train_test_split(pred_var, targ_var, test_size=0.25)

# Using the Standard Scaler for standardization of our dataset.
scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

nb = GaussianNB()
nb.fit(X_train, y_train)

y_pred = nb.predict(X_test)

print("The accuracy score of the Gaussian Naive Bayes is {}".format(nb.score(X_test, y_test) * 100))
```

```
The accuracy score of the Gaussian Naive Bayes is 84.85357869592144
```

In [12]:
```
#Creating confusion matrix and classification report on the prediction.
y_pred = nb.predict(X_test)
matrix_data = pd.DataFrame(confusion_matrix(y_test, y_pred), index=['F', 'T'], columns=['F', 'T'])
print("Confusion Matrix:\n",matrix_data)

print(classification_report(y_test,np.ravel(y_pred)))
```

```
Confusion Matrix:
       F     T
F  8955  1021
T   691   636
              precision    recall  f1-score   support

           0       0.93      0.90      0.91      9976
           1       0.38      0.48      0.43      1327

    accuracy                           0.85     11303
   macro avg       0.66      0.69      0.67     11303
weighted avg       0.86      0.85      0.86     11303
```

We are able to gain a pretty high accuracy using the Gaussian Naive Bayes algorithm. Since the target variable predicts a boolean type, where it determines if a customer has subscribed the term deposit that was provided by the bank, Bernoulli Naive Bayes might not be able to predict with good accuracy score. This is because, Bernoulli Naive Bayes works well when the predictor features are in boolean type.

Let us see how Bernoulli Naive Bayes can predict for the given dataset.

In [13]:
```
#Considering all the features for classification.
pred_var = bank_data.iloc[:, 0:15]
targ_var = bank_data['Target']

X_train, X_test, y_train, y_test = train_test_split(pred_var, targ_var, test_size=0.25)

# Using the Standard Scaler for standardization of our dataset.
scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

nb = BernoulliNB()
nb.fit(X_train, y_train)

y_pred = nb.predict(X_test)

print("The accuracy score of the Bernoulli Naive Bayes is {}".format(nb.score(X_test, y_test) * 100))
```

```
The accuracy score of the Bernoulli Naive Bayes is 86.90613111563302
```

In [14]:
```
#Creating confusion matrix and classification report on the prediction.
y_pred = nb.predict(X_test)
matrix_data = pd.DataFrame(confusion_matrix(y_test, y_pred), index=['F', 'T'], columns=['F', 'T'])
print("Confusion Matrix:\n",matrix_data)

print(classification_report(y_test,np.ravel(y_pred)))
```

```
Confusion Matrix:
       F    T
F  9478  519
T   961  345
              precision    recall  f1-score   support

           0       0.91      0.95      0.93      9997
           1       0.40      0.26      0.32      1306

    accuracy                           0.87     11303
   macro avg       0.65      0.61      0.62     11303
weighted avg       0.85      0.87      0.86     11303
```

Here, the Bernoulli was able to predict better than the Gaussian Naive Bayes classifier slightly. The accuracy is similar and any of the algorithm can be used.

We have so far tested by considering all the features provided to us. Let us see if we can list the important feature and get higher accuracy score. We will use the SelectKBest with chi2 function and ExtraTreeClassifiers algorithm to find the top list of important features.

Performance Improvement

In [15]:
```
#Finding the features that has negative value. Features having negative values cannot be used for finding important featuers. We will be removing these features from ou
print("List of predictor features that contains negative values:")
for data_keys in bank_data.keys():
    if any(bank_data[data_keys] < 0):
        print(data_keys)
```

```
List of predictor features that contains negative values:
balance
pdays
```

In [16]:
```
#Considering all of the features to find the top features and removing pdays and balance since it contains negative values.
X = bank_data[['age', 'job', 'marital', 'education', 'default', 'housing', 'loan',\
               'contact', 'day', 'month', 'duration', 'campaign', 'previous', 'poutcome']]
y = bank_data.iloc[:, -1]

select_k_best = SelectKBest(score_func=chi2, k=10)

best_result = select_k_best.fit(X, y)

scores = pd.DataFrame(best_result.scores_)
columns = pd.DataFrame(X.columns)

feature_scores = pd.concat([columns, scores], axis=1)
feature_scores.columns = ['Field', 'Score']

print("Listing the top 5 features : \n{}".format(feature_scores.nlargest(10, 'Score')))
```

```
Listing the top 5 features :
        Field         Score
10   duration  1.807711e+06
13   poutcome  5.113205e+03
9       month  4.317618e+03
12   previous  3.593220e+03
11   campaign  8.405821e+02
5     housing  4.867440e+02
7     contact  3.001611e+02
6        loan  1.765161e+02
8         day  1.592004e+02
3   education  1.391912e+02
```

The top five important features in finding the likeliness of customers subscribing to term deposits are duration, poutcome, month, previous and campaign.

We will use these top 5 features to see how well the accuracy score is obtained.

In [17]:
```
#Considering top five important features
pred_var = bank_data[['duration', 'poutcome', 'month', 'previous', 'campaign']]
targ_var = bank_data['Target']

X_train, X_test, y_train, y_test = train_test_split(pred_var, targ_var, test_size=0.25)

# Using the Standard Scaler for standardization of our dataset.
scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

nb = GaussianNB()
nb.fit(X_train, y_train)

y_pred = nb.predict(X_test)

print("The accuracy score of the Gaussian Naive Bayes is {}".format(nb.score(X_test, y_test) * 100))
```

```
The accuracy score of the Gaussian Naive Bayes is 86.39299301070513
```

In [18]:
```
#Creating confusion matrix and classification report on the prediction.
y_pred = nb.predict(X_test)
matrix_data = pd.DataFrame(confusion_matrix(y_test, y_pred), index=['F', 'T'], columns=['F', 'T'])
print("Confusion Matrix:\n",matrix_data)

print(classification_report(y_test,np.ravel(y_pred)))
```

```
Confusion Matrix:
       F    T
F  9153  830
T   708  612
              precision    recall  f1-score   support

           0       0.93      0.92      0.92      9983
           1       0.42      0.46      0.44      1320

    accuracy                           0.86     11303
   macro avg       0.68      0.69      0.68     11303
weighted avg       0.87      0.86      0.87     11303
```

In [19]:
```
#Considering top five important features
pred_var = bank_data[['duration', 'poutcome', 'month', 'previous', 'campaign']]
targ_var = bank_data['Target']

X_train, X_test, y_train, y_test = train_test_split(pred_var, targ_var, test_size=0.25)

# Using the Standard Scaler for standardization of our dataset.
scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

nb = BernoulliNB()
nb.fit(X_train, y_train)

y_pred = nb.predict(X_test)

print("The accuracy score of the Bernoulli Naive Bayes is {}".format(nb.score(X_test, y_test) * 100))
```

```
The accuracy score of the Bernoulli Naive Bayes is 87.23347783774219
```

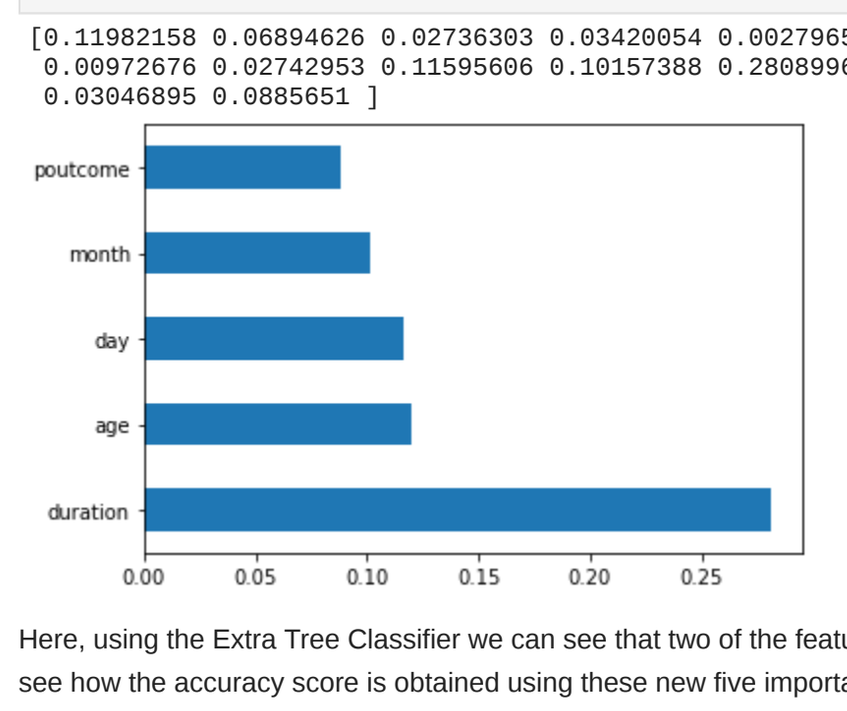
In [20]:
```
#Creating confusion matrix and classification report on the prediction.
y_pred = nb.predict(X_test)
matrix_data = pd.DataFrame(confusion_matrix(y_test, y_pred), index=['F', 'T'], columns=['F', 'T'])
print("Confusion Matrix:\n",matrix_data)

print(classification_report(y_test,np.ravel(y_pred)))
```

```
Confusion Matrix:
       F    T
F  9567  390
T  1053  293
              precision    recall  f1-score   support

           0       0.90      0.96      0.93      9957
           1       0.43      0.22      0.29      1346

    accuracy                           0.87     11303
   macro avg       0.66      0.59      0.61     11303
weighted avg       0.84      0.87      0.85     11303
```

Here, both the Gaussian and Bernoulli Naive Bayes' provided with similar accuracy score. This shows that both of the algorithm provides similar result and any of them could be used.

In [21]:
```
#Using Extra Trees Classifier to find the top 5 important features.
etc = ExtraTreesClassifier()

etc.fit(X,y)
print(etc.feature_importances_) #use inbuilt class feature_importances of tree based classifiers

#plot graph of feature importances for better visualization
feat_importances = pd.Series(etc.feature_importances_, index=X.columns)
feat_importances.nlargest(5).plot(kind='barh')
plt.show()
```

```
[0.11982158 0.06894626 0.02736303 0.03420054 0.00279653 0.0240335
 0.00972676 0.02742953 0.11595606 0.10157388 0.28089968 0.0682186
 0.03046895 0.0885651 ]
```



Here, using the Extra Tree Classifier we can see that two of the features out of five that was obtained previously are replaced. The campaign and previous features are replaced by day and age features. We will see how the accuracy score is obtained using these new five important features with both Gaussian and Bernoulli algorithm.

In [22]:
```
#Considering top five features only that were obtained using ExtraTreeClassifier
pred_var = bank_data[['duration', 'age', 'day', 'month', 'poutcome']]
targ_var = bank_data['Target']

X_train, X_test, y_train, y_test = train_test_split(pred_var, targ_var, test_size=0.25)

# Using the Standard Scaler for standardization of our dataset.
scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

nb = GaussianNB()
nb.fit(X_train, y_train)

y_pred = nb.predict(X_test)

print("The accuracy score of the Gaussian Naive Bayes is {}".format(nb.score(X_test, y_test) * 100))
```

```
The accuracy score of the Gaussian Naive Bayes is 87.33079713350438
```

In [23]:
```
#Creating confusion matrix and classification report on the prediction.
y_pred = nb.predict(X_test)
matrix_data = pd.DataFrame(confusion_matrix(y_test, y_pred), index=['F', 'T'], columns=['F', 'T'])
print("Confusion Matrix:\n",matrix_data)

print(classification_report(y_test,np.ravel(y_pred)))
```

```
Confusion Matrix:
       F    T
F  9275  670
T   762  596
              precision    recall  f1-score   support

           0       0.92      0.93      0.93      9945
           1       0.47      0.44      0.45      1358

    accuracy                           0.87     11303
   macro avg       0.70      0.69      0.69     11303
weighted avg       0.87      0.87      0.87     11303
```

In [24]:
```
#Considering top five features only that were obtained using ExtraTreeClassifier
pred_var = bank_data[['duration', 'age', 'day', 'month', 'poutcome']]
targ_var = bank_data['Target']

X_train, X_test, y_train, y_test = train_test_split(pred_var, targ_var, test_size=0.25)

# Using the Standard Scaler for standardization of our dataset.
scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

nb = BernoulliNB()
nb.fit(X_train, y_train)

y_pred = nb.predict(X_test)

print("The accuracy score of the Bernoulli Naive Bayes is {}".format(nb.score(X_test, y_test) * 100))
```

```
The accuracy score of the Bernoulli Naive Bayes is 87.86162965584357
```

In [25]:
```
#Creating confusion matrix and classification report on the prediction.
y_pred = nb.predict(X_test)
matrix_data = pd.DataFrame(confusion_matrix(y_test, y_pred), index=['F', 'T'], columns=['F', 'T'])
print("Confusion Matrix:\n",matrix_data)

print(classification_report(y_test,np.ravel(y_pred)))
```

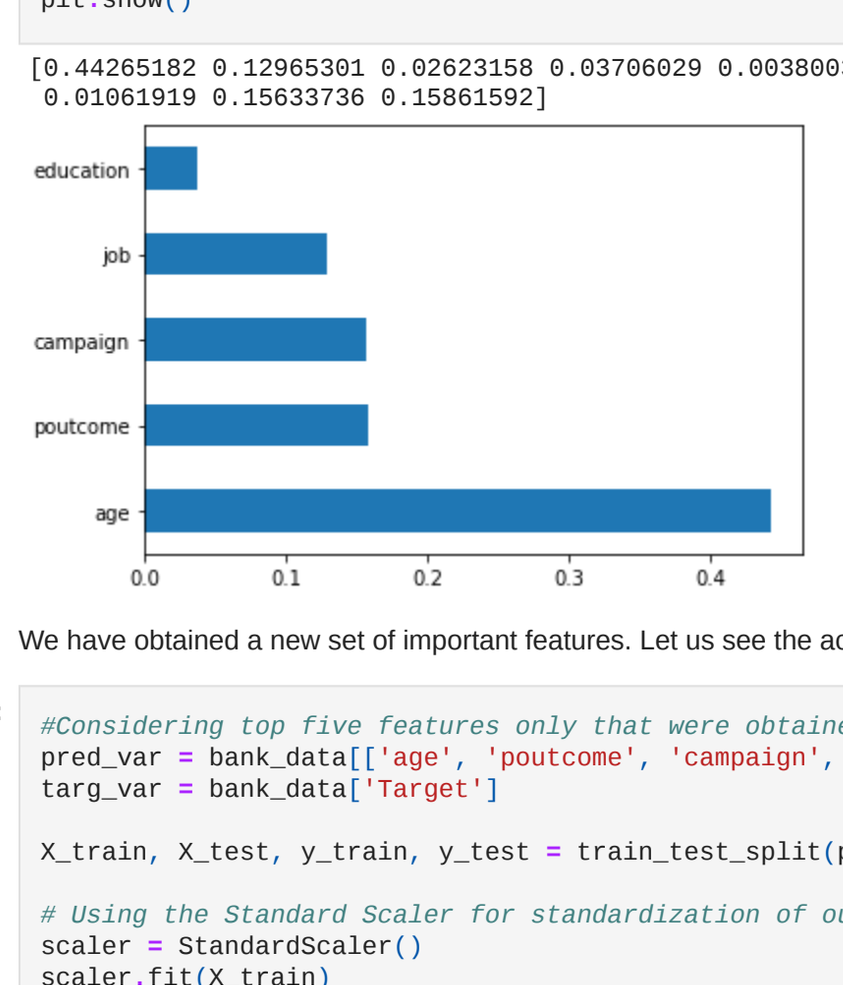
```
Confusion Matrix:
       F    T
F  9794  185
T  1187  137
              precision    recall  f1-score   support

           0       0.89      0.98      0.93      9979
           1       0.43      0.10      0.17      1324

    accuracy                           0.88     11303
   macro avg       0.66      0.54      0.55     11303
weighted avg       0.84      0.88      0.84     11303
```

With these new five important features we are able to obtain the highest accuracy score so far. These five feature - duration, age, day, month and poutcome looks to provide better accuracy score as compared to other combination of features.

However, the features such as duration and day, month might not be helpful or give better insights on predicting the likelihood of subscribing to the term deposit. We will remove few of the feature that might not be well suited in this prediction to see how well the accuracy score is obtained and what other features are important.

In [26]:
```
#Considering all of the features to find the top features and removing pdays and balance since it contains negative values.
X = bank_data[['age', 'job', 'marital', 'education', 'default', 'housing', 'loan',\
               'campaign', 'poutcome']]
y = bank_data.iloc[:, -1]

#Using Extra Trees Classifier to find the top 5 important features.
etc = ExtraTreesClassifier()

etc.fit(X,y)
print(etc.feature_importances_) #use inbuilt class feature_importances of tree based classifiers

#plot graph of feature importances for better visualization
feat_importances = pd.Series(etc.feature_importances_, index=X.columns)
feat_importances.nlargest(5).plot(kind='barh')
plt.show()
```

```
[0.44265182 0.12965301 0.02623158 0.03706029 0.00380033 0.03503049
 0.01061919 0.15633736 0.15861592]
```



We have obtained a new set of important features. Let us see the accuracy score provided with these features.

In [27]:
```
#Considering top five features only that were obtained using ExtraTreeClassifier
pred_var = bank_data[['age', 'poutcome', 'campaign', 'job', 'education']]
targ_var = bank_data['Target']

X_train, X_test, y_train, y_test = train_test_split(pred_var, targ_var, test_size=0.25)

# Using the Standard Scaler for standardization of our dataset.
scaler = StandardScaler()
scaler.fit(X_train)

X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

nb = BernoulliNB()
nb.fit(X_train, y_train)

y_pred = nb.predict(X_test)

print("The accuracy score of the Bernoulli Naive Bayes is {}".format(nb.score(X_test, y_test) * 100))
```

```
The accuracy score of the Bernoulli Naive Bayes is 88.29514288242059
```

In [28]:
```
#Creating confusion matrix and classification report on the prediction.
y_pred = nb.predict(X_test)
matrix_data = pd.DataFrame(confusion_matrix(y_test, y_pred), index=['F', 'T'], columns=['F', 'T'])
print("Confusion Matrix:\n",matrix_data)

print(classification_report(y_test,np.ravel(y_pred)))
```

```
Confusion Matrix:
       F  T
F  9980  0
T  1323  0
              precision    recall  f1-score   support

           0       0.88      1.00      0.94      9980
           1       0.00      0.00      0.00      1323

    accuracy                           0.88     11303
   macro avg       0.44      0.50      0.47     11303
weighted avg       0.78      0.88      0.83     11303
```

Using these above features, the likelihood of a customer subscribing to the term deposit is not predicted properly.

Due to this, we were not able to calculate the Precision and Recall for the Target of value 1.

Conclusion

Thus, we can conclude that we are able to find accuracy of 86% using the Naive Bayes' algorithm to predict the classification of customers that are willing to subscribe the term deposit provided by a bank.

Using this method we can use to find and prioritize the list of customers that are likely to subscribe to the term deposit. This will help the bank to increase their revenue and have a successful campaign of attracting clients to the bank.

However, the important feature that are predicted by the algorithm might not be correct since the number of calls and duration of the call would show that the people subscribing the term deposit are likely to be on call for longer period than the ones that do not subscribe the term deposit.

By removing few featuers and predicting the accuracy score, we were able to gain slight increase on the score but one of the target variables were not predicted at all. To determine which features are to be used and excluded needs to be obtained from the domain expertise to find better accuracy score.

In [ ]: