MSDS 422-57 Assignment 3- Prerak, Mehta

Introduction / Summary

The data in works in this assignment is related with the marketing campaigns of a Portuguese banking institution. The data has various features that gives information regarding the people that were contacted. The target variable in this dataset is the 'yes/no' column that depicts whether the person contacted subscribed to the bank term deposit or not. A machine learning model using Logistic Regression and naive Bayes classification methods will be implemented to predict the responses of the people contacted based upon the feature information provided in the dataset.

In [38]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import pickle
import sys
import os
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

In [39]:
```
#open the pickle file that contains the data
with open('bankTrain.pickle','rb') as inFile:
    bankData=pickle.load(inFile)
```

In [41]:
```
#Should be true if the bankData is a pd DataFrame.
print(isinstance(bankData,pd.DataFrame))
#information on the shape of the dataset
print(bankData.shape)
#Columns in the dataset that provide feature information that will be us
eful in modeling
print(bankData.columns)
#data types of the columns
print(bankData.dtypes)
```

```
True
(40570, 22)
Index(['age', 'job', 'marital', 'education', 'default', 'housing', 'loa
n',
       'contact', 'month', 'day_of_week', 'duration', 'campaign', 'pday
s',
       'previous', 'poutcome', 'emp.var.rate', 'cons.price.idx',
       'cons.conf.idx', 'euribor3m', 'nr.employed', 'y', 'bankID'],
      dtype='object')
age                 int64
job                object
marital            object
education          object
default            object
housing            object
loan               object
contact            object
month              object
day_of_week        object
duration            int64
campaign            int64
pdays               int64
previous            int64
poutcome           object
emp.var.rate      float64
cons.price.idx    float64
cons.conf.idx     float64
euribor3m         float64
nr.employed       float64
y                  object
bankID              int64
dtype: object
```

In [43]:
```
print(bankData.describe())
```

```
                age      duration      campaign         pdays       prev
ious  \
count  40570.000000  40570.000000  40570.000000  40570.000000  40570.00
0000
mean      40.021863    258.428469      2.567833    962.457308       0.17
3305
std       10.421979    259.538128      2.772428    186.956234       0.49
5599
min       17.000000      0.000000      1.000000      0.000000       0.00
0000
25%       32.000000    102.000000      1.000000    999.000000       0.00
0000
50%       38.000000    180.000000      2.000000    999.000000       0.00
0000
75%       47.000000    320.000000      3.000000    999.000000       0.00
0000
max       98.000000   4918.000000     56.000000    999.000000       7.00
0000

       emp.var.rate  cons.price.idx  cons.conf.idx     euribor3m  \
count  40570.000000    40570.000000    40570.00000  40570.000000
mean       0.080658       93.575402      -40.50564      3.619832
std        1.571576        0.579181        4.63139      1.735075
min       -3.400000       92.201000      -50.80000      0.634000
25%       -1.800000       93.075000      -42.70000      1.344000
50%        1.100000       93.749000      -41.80000      4.857000
75%        1.400000       93.994000      -36.40000      4.961000
max        1.400000       94.767000      -26.90000      5.045000

        nr.employed        bankID
count  40570.000000  40570.000000
mean    5166.986495  20587.676091
std       72.269363  11899.158465
min     4963.600000      0.000000
25%     5099.100000  10268.250000
50%     5191.000000  20585.500000
75%     5228.100000  30894.750000
max     5228.100000  41187.000000
```

The describe function gives us quite a bit of information on the spread of the numeric values of some features. Right of the back we can see that the emp.var.rate, cons.price.idx, and nr.employed columns don't vary a lot and more likely are not good indicators for the model.

In [57]:
```
GroupBy = bankData.groupby(['y','month']).count()#[['day_of_week']]

GroupBy['bankID']
```

Out[57]:
```
y    month
no   apr       2062
     aug       5427
     dec         90
     jul       6428
     jun       4684
     mar        268
     may      12699
     nov       3624
     oct        399
     sep        309
yes  apr        531
     aug        648
     dec         88
     jul        647
     jun        551
     mar        272
     may        868
     nov        413
     oct        311
     sep        251
Name: bankID, dtype: int64
```

This shows a pretty even distribution for y grouped on day_of_week column. Thus showing that this feature has no real effect on the target value

Checks for missing or invalid data values, should go here. Look into suspicious. or unexpected, data types.

There are no missing values in the entire dataset and hence we do not have to worry about imputing or deleting such null values.

In [58]:
```
bankData.isnull().sum()
```

Out[58]:
```
age               0
job               0
marital           0
education         0
default           0
housing           0
loan              0
contact           0
month             0
day_of_week       0
duration          0
campaign          0
pdays             0
previous          0
poutcome          0
emp.var.rate      0
cons.price.idx    0
cons.conf.idx     0
euribor3m         0
nr.employed       0
y                 0
bankID            0
dtype: int64
```

Although there are many unkowns in a few columns, we will not be imputing them as unknowns, too, can act as predictors. Since there are multiple features at play here, imputing these unkowns or removing those entries might harm the eventual rare event (target value)

Feature Selection

We already decided to drop 'emp.var.rate', 'cons.price.idx', and 'nr.employed' from all the columns with numerical values due to their unvarying scale. However there are more features that logically will not contribute to the prediction of the target value.

- 'default' feature has almost 80% values as 'no' and hence does not contribute to the prediction.
- 'day_of_week' will be eliminiated as well due to the distribution shown above.
- 'duration' column will be eliminated as well since the duration of the call cannot be known before the call and the y is known after the call; thus becoming a useless feature for designing a predictive model.
- 'education' column has more than negligible amount of unknowns that return a 'yes' target value. However the 'job' column has lesser unkowns and in a way we assume relevance from job column should supercede information from education column. Thus we will not take the education column into consideration.

Rearrange the columns and put the data in a new dataframe without eliminated features. Also transform the 'pdays' column into 3 categories to fetch a better understanding of the data. Also transforming the target variable "yes" to 1 and "no" to 0.

Later each non-numeric categorical column with n distinct values will be converted into dummy variable n-1 columns. N-1 because the dummy variable trap will be avoided.

In [190]:
```
bankData2 = bankData[['y','bankID','previous','age','euribor3m','job','m
arital','housing','loan',
                      'contact','month','pdays','poutcome']]

f1 = list(range(0,11))
f2 = list(range(11,999))

bankData2["pdays"].replace(f1,"ten.days.or.less", inplace = True)
bankData2["pdays"].replace(f2,"ten.days.or.more", inplace = True)
bankData2["pdays"].replace([999],"Never.Contacted", inplace = True)
bankData2["y"].replace({"yes": 1, "no":0}, inplace = True)
```

```
//anaconda3/lib/python3.7/site-packages/pandas/core/generic.py:6746: Se
ttingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self._update_inplace(new_data)
```

In [191]:
```
bankData3=pd.get_dummies(bankData2, drop_first=True)
```

As mentioned the following model will be trained using all the features except the ones that were analyzed and logically uninformative towards predicting the target variable.

In [203]:
```
X = bankData3.iloc[:, 2:].values
y = bankData3.iloc[:, 0].values
```

In [114]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.
10, random_state = 0)
```

In [115]:
```
from sklearn.preprocessing import StandardScaler, MinMaxScaler
sc = MinMaxScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [67]:
```
X_train.shape
```

Out[67]:
```
(36513, 35)
```

In [68]:
```
from sklearn.linear_model import LogisticRegression
classifier_lg = LogisticRegression(intercept_scaling = 50,random_state =
0)
classifier_lg.fit(X_train, y_train)
```

Out[68]:
```
LogisticRegression(intercept_scaling=50, random_state=0)
```

In [120]:
```
y_pred = classifier_lg.predict(X_test)
ar =(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y
_test),1)),1))
print(ar)
```

```
[[0 0]
 [0 0]
 [0 0]
 ...
 [0 0]
 [0 0]
 [0 0]]
```

In [121]:
```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print('Confusion Matrix as follow: ')
print(cm)
print('Accuracy Score', accuracy_score(y_test, y_pred))
```

```
Confusion Matrix as follow:
[[3540   45]
 [ 387   85]]
Accuracy Score 0.8935173773724427
```

Using Gaussian NB Method

In [122]:
```
from sklearn.naive_bayes import GaussianNB
classifier_nb = GaussianNB()
classifier_nb.fit(X_train, y_train)
```

Out[122]:
```
GaussianNB()
```

In [123]:
```
y_pred_nb = classifier_nb.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred_nb),1), y_test.reshape(l
en(y_test),1)),1))
```

```
[[0 0]
 [0 0]
 [0 0]
 ...
 [0 0]
 [0 0]
 [0 0]]
```

In [124]:
```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred_nb)
print('Confusion Matrix as follow: ')
print(cm)
print('Accuracy Score', accuracy_score(y_test, y_pred_nb))
```

```
Confusion Matrix as follow:
[[3344  241]
 [ 269  203]]
Accuracy Score 0.8742913482869115
```

We noticed that the Logistic Regression model yields a better accuracy than the Guassian Naive Bayes model. Next we will use the K folds method with both these algorithms to see if it yields a better accuracy

In [207]:
```
#Using k folds
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import StratifiedKFold, StratifiedShuffleSp
lit
from sklearn.base import clone
skf = StratifiedKFold(n_splits=40, random_state=99,shuffle=True)
scaler=preprocessing.MinMaxScaler()


logit_kf=LogisticRegression(solver='lbfgs')
guass_nb=GaussianNB()
```

In [208]:
```
from sklearn.metrics import confusion_matrix, accuracy_score
hold_logit = []
hold_guass = []

for train_ndx, test_ndx in skf.split(X, y):
    clone_clf = clone(logit_clf)
    clone_guass = clone(guass_nb)
    X_trainS=scaler.fit_transform(X[train_ndx])
    y_train = y[train_ndx]
    X_testS=scaler.transform(X[test_ndx])
    y_test = y[test_ndx]

    foldfitlogit=clone_clf.fit(X_trainS, y_train)
    foldfitguass=clone_guass.fit(X_trainS, y_train)

    y_pred_test_logit=foldfitlogit.predict(X_testS)
    y_pred_train_logit=foldfitlogit.predict(X_trainS)
    y_pred_test_guass=foldfitguass.predict(X_testS)
    y_pred_train_guass=foldfitguass.predict(X_trainS)

    trainAcc_logit=accuracy_score(y_train,y_pred_train_logit)
    testAcc_logit=accuracy_score(y_test,y_pred_test_logit)
    trainAcc_guass=accuracy_score(y_train,y_pred_train_guass)
    testAcc_guass=accuracy_score(y_test,y_pred_test_guass)

    hold_logit.append({'train_accuracy':trainAcc_logit,'test_accuracy':t
estAcc_logit})
    hold_guass.append({'train_accuracy':trainAcc_guass,'test_accuracy':t
estAcc_guass})
```

In [209]:
```
print('K Folds method using Logistic Regression')
pd.DataFrame(hold_logit)[['train_accuracy','test_accuracy']].describe()
```

```
K Folds method using Logistic Regression
```

Out[209]:

	train_accuracy	test_accuracy
count	40.000000	40.000000
mean	0.899162	0.898496
std	0.000142	0.005226
min	0.898878	0.883629
25%	0.899072	0.895464
50%	0.899178	0.898422
75%	0.899237	0.902367
max	0.899459	0.908284

In [196]:
```
print('K Folds method using Naive bayes Classification')
pd.DataFrame(hold_guass)[['train_accuracy','test_accuracy']].describe()
```

K Folds method using Naive bayes Classification

Out[196]:

	train_accuracy	test_accuracy
count	40.000000	40.000000
mean	0.874987	0.874760
std	0.000257	0.008025
min	0.874381	0.857988
25%	0.874826	0.869919
50%	0.875011	0.872781
75%	0.875190	0.879714
max	0.875468	0.893491

In [197]:
```
from sklearn.metrics import average_precision_score

y_score = foldfitlogit.decision_function(X_testS)

average_precision = average_precision_score(y_pred_test_logit, y_score)

print('Average precision-recall score: {0:0.2f}'.format(
      average_precision))
```

Average precision-recall score: 1.00

In [202]:
```
from sklearn.metrics import roc_curve, precision_score, recall_score
from sklearn.metrics import accuracy_score, confusion_matrix, roc_auc_sc
ore
yTestPredProbs = foldfitlogit.predict_proba(X_testS)

testAUC=roc_auc_score(y_pred_test_logit, yTestPredProbs[:,1])
print('Test AUC: {0:4.3f}'.format(testAUC))
```

Test AUC: 1.000

From all the analysis it's best to say that the 'foldfitlogit' Logistic Regression model trained using the K folds method provides the best accuracy of almost 90%. We will use that model on the test dataframe provided.

In [148]:
```
with open('bankTest.pickle','rb') as inFile:
    bankDataTest=pickle.load(inFile)
```

Now the same preprocessing of the data will need to happen on the test dataset before we predict results using our trained model.

In [151]:
```
bankData4 = bankDataTest[['y','bankID','previous','age','euribor3m','jo
b','marital','housing','loan',
                         'contact','month','pdays','poutcome']]

f3 = list(range(0,11))
f4 = list(range(11,999))

bankData4["pdays"].replace(f3,"ten.days.or.less", inplace = True)
bankData4["pdays"].replace(f4,"ten.days.or.more", inplace = True)
bankData4["pdays"].replace([999],"Never.Contacted", inplace = True)

bankData5=pd.get_dummies(bankData4, drop_first=True)
```

In [154]:
```
X = bankData5.iloc[:, 2:].values
y = bankData5.iloc[:, 0].values
```

In [160]:
```
y_pred = foldfitlogit.predict(X)
```

In [164]:
```
bankID = bankData5.iloc[:, 1].values
```

In [177]:
```
y_pred_prob = foldfitlogit.predict_proba(X)

ar = np.concatenate((bankID.reshape(len(bankID),1), y_pred.reshape(len(y
_pred),1),
                     y_pred_prob[:,1].reshape(len(y_pred_prob),1)),1)

FinalDf = pd.DataFrame(data=ar, columns = ['bankID','pred_y','prob_y'])
FinalDf.rename(columns = {"0":"bankID","1":"pred_y","2":"prob_y"})
FinalDf.astype({'bankID': 'int32','pred_y':'int32'}).dtypes
```

Out[177]:

	bankID	pred_y	prob_y
0	13.0	0.0	0.004217
1	150.0	0.0	0.002259
2	228.0	0.0	0.005192
3	243.0	0.0	0.000365
4	280.0	1.0	0.618428
...	...	...	...
613	40996.0	0.0	0.005281
614	40998.0	0.0	0.000718
615	41112.0	0.0	0.004535
616	41118.0	0.0	0.439020
617	41132.0	0.0	0.000349

618 rows × 3 columns

Export to CSV

In [182]:
```
FinalDf.to_csv('preds_mehta_3.csv')
```

Assignment Discussion, Conclusion and recommendations

Receiving almost a 90% test accuracy is a great figure for most machine learning models. However this was a rare event dataset where the target value was a 'yes' only in about 11.3% (4580 'yes' out of 40570 total rows) of the cases. Thus we can safely say that if this model predicted 'no'as an asnwer for all test cases it would have been correct almost 88% of the times. Thus the accuracy of predicting of this model can be considered just average. In fact the average precision-recall and AUC score of the model come out as 1.00 which is extremely surprising. Out of the cut-down features that are currently considered in the model training, various features were taken out just to see how the accuracy of the model changes; but it didn't change considerably. One of the most prominent reasons could be improper feature scaling we utilized (MinMaxScaler). Another reason could be the abundance on the 'no' answers in the target variable column. Two recommendations I would like to give to the people that collect the data are 1) Do not have any unkowns in the dataset as those entries could have made a difference on how much the columns containing those unknown values might have had on the target prediction and 2) Retrieve more customer data regarding the possessions of the customer along with their debts as that will give a fair idea about their salaries. Salary can be one of the most deciding factors in deciding whether the customer would agree for a term deposit or not.