Week6_PythonCode_Demo

September 20, 2021

0.1 Importing Libraries

[1]:
```
import numpy as np
import pandas as pd
import matplotlib as plt

from sklearn.impute import SimpleImputer
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.feature_selection import RFE
from sklearn.metrics import roc_curve
from sklearn.metrics import accuracy_score

import statsmodels.api as sm


# Set a random seed
np.random.seed(12345)

# For supressing warnings
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
```

0.2 Data loading

[2]:
```
data = pd.read_csv("pmad_pva.csv")
```

0.3 Data Description

[3]:
```
# Top 5 rows of the dataframe

data.head()
```

[3]:
```
   TargetB      ID  TargetD  GiftCnt36  GiftCntAll  GiftCntCard36  \
0        0   14974      NaN          2           4              1
1        0    6294      NaN          1           8              0
2        1   46110      4.0          6          41              3
3        1  185937     10.0          3          12              3
4        0   29637      NaN          1           1              1

   GiftCntCardAll  GiftAvgLast  GiftAvg36  GiftAvgAll  …  PromCntCardAll  \
0               3         17.0      13.50        9.25  …              13
1               3         20.0      20.00       15.88  …              24
2              20          6.0       5.17        3.73  …              22
3               8         10.0       8.67        8.50  …              16
4               1         20.0      20.00       20.00  …               6

   StatusCat96NK  StatusCatStarAll  DemCluster  DemAge  DemGender  \
0              A                 0           0     NaN          F
1              A                 0          23    67.0          F
2              S                 1           0     NaN          M
3              E                 1           0     NaN          M
4              F                 0          35    53.0          M

   DemHomeOwner  DemMedHomeValue  DemPctVeterans DemMedIncome
0             U                0               0          NaN
1             U           186800              85          NaN
2             U            87600              36      38750.0
3             U           139200              27      38942.0
4             U           168100              37      71509.0

[5 rows x 28 columns]
```

[4]:
```
# All the variables in dataset
data.columns
```

[4]:
```
Index(['TargetB', 'ID', 'TargetD', 'GiftCnt36', 'GiftCntAll', 'GiftCntCard36',
       'GiftCntCardAll', 'GiftAvgLast', 'GiftAvg36', 'GiftAvgAll',
       'GiftAvgCard36', 'GiftTimeLast', 'GiftTimeFirst', 'PromCnt12',
       'PromCnt36', 'PromCntAll', 'PromCntCard12', 'PromCntCard36',
       'PromCntCardAll', 'StatusCat96NK', 'StatusCatStarAll', 'DemCluster',
       'DemAge', 'DemGender', 'DemHomeOwner', 'DemMedHomeValue',
       'DemPctVeterans', 'DemMedIncome'],
      dtype='object')
```

[5]:
```
# Summary Statistics

data.describe().T
```

[5]:

	count	mean	std	min	25%	\
TargetB	9686.0	0.500000	0.500026	0.00	0.00	
ID	9686.0	97975.474086	56550.171120	12.00	48835.50	
TargetD	4843.0	15.624344	12.445137	1.00	10.00	
GiftCnt36	9686.0	3.205451	2.133421	0.00	2.00	
GiftCntAll	9686.0	10.507640	8.993401	1.00	4.00	
GiftCntCard36	9686.0	1.856597	1.595419	0.00	1.00	
GiftCntCardAll	9686.0	5.582490	4.736894	0.00	2.00	
GiftAvgLast	9686.0	16.017739	12.041805	0.00	10.00	
GiftAvg36	9686.0	14.876203	10.057007	0.00	9.60	
GiftAvgAll	9686.0	12.489325	9.209297	1.50	7.75	
GiftAvgCard36	7906.0	14.224431	10.022710	1.33	8.67	
GiftTimeLast	9686.0	18.002168	4.073549	4.00	16.00	
GiftTimeFirst	9686.0	71.100351	37.691984	15.00	36.00	
PromCnt12	9686.0	12.988850	4.823458	2.00	11.00	
PromCnt36	9686.0	29.348235	7.809743	4.00	25.00	
PromCntAll	9686.0	48.483481	23.061483	5.00	29.00	
PromCntCard12	9686.0	5.392009	1.323648	0.00	5.00	
PromCntCard36	9686.0	11.954677	4.571568	2.00	7.00	
PromCntCardAll	9686.0	19.007124	8.562193	2.00	12.00	
StatusCatStarAll	9686.0	0.540574	0.498377	0.00	0.00	
DemCluster	9686.0	27.150320	14.832665	0.00	14.00	
DemAge	7279.0	59.150845	16.516400	0.00	47.00	
DemMedHomeValue	9686.0	110986.299814	98670.855450	0.00	52300.00	
DemPctVeterans	9686.0	30.604274	11.394988	0.00	25.00	
DemMedIncome	7329.0	53513.457361	19805.168339	2499.00	40389.00	

	50%	75%	max
TargetB	0.50	1.00	1.0
ID	99106.00	148538.75	191779.0
TargetD	13.00	20.00	200.0
GiftCnt36	3.00	4.00	16.0
GiftCntAll	8.00	15.00	91.0
GiftCntCard36	1.00	3.00	9.0
GiftCntCardAll	4.00	8.00	41.0
GiftAvgLast	15.00	20.00	450.0
GiftAvg36	13.50	18.50	260.0
GiftAvgAll	10.71	15.00	450.0
GiftAvgCard36	12.50	18.00	260.0
GiftTimeLast	18.00	20.00	27.0
GiftTimeFirst	68.00	105.00	260.0
PromCnt12	12.00	13.00	59.0
PromCnt36	31.00	33.00	78.0
PromCntAll	48.00	65.00	174.0
PromCntCard12	6.00	6.00	17.0
PromCntCard36	13.00	16.00	28.0
PromCntCardAll	19.00	26.00	56.0

```
StatusCatStarAll      1.00       1.00       1.0
DemCluster           27.00      40.00      53.0
DemAge               60.00      73.00      87.0
DemMedHomeValue   76900.00  128175.00  600000.0
DemPctVeterans       31.00      37.00      85.0
DemMedIncome      48699.00   62385.00  200001.0
```

[6]:
```python
# Dimensions of the data

data.shape
```

[6]:
```
(9686, 28)
```

[7]:
```python
# Structure of the data

data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9686 entries, 0 to 9685
Data columns (total 28 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   TargetB            9686 non-null   int64
 1   ID                 9686 non-null   int64
 2   TargetD            4843 non-null   float64
 3   GiftCnt36          9686 non-null   int64
 4   GiftCntAll         9686 non-null   int64
 5   GiftCntCard36      9686 non-null   int64
 6   GiftCntCardAll     9686 non-null   int64
 7   GiftAvgLast        9686 non-null   float64
 8   GiftAvg36          9686 non-null   float64
 9   GiftAvgAll         9686 non-null   float64
 10  GiftAvgCard36      7906 non-null   float64
 11  GiftTimeLast       9686 non-null   int64
 12  GiftTimeFirst      9686 non-null   int64
 13  PromCnt12          9686 non-null   int64
 14  PromCnt36          9686 non-null   int64
 15  PromCntAll         9686 non-null   int64
 16  PromCntCard12      9686 non-null   int64
 17  PromCntCard36      9686 non-null   int64
 18  PromCntCardAll     9686 non-null   int64
 19  StatusCat96NK      9686 non-null   object
 20  StatusCatStarAll   9686 non-null   int64
 21  DemCluster         9686 non-null   int64
 22  DemAge             7279 non-null   float64
 23  DemGender          9686 non-null   object
 24  DemHomeOwner       9686 non-null   object
 25  DemMedHomeValue    9686 non-null   int64
```

```
 26  DemPctVeterans    9686 non-null   int64
 27  DemMedIncome      7329 non-null   float64
dtypes: float64(7), int64(18), object(3)
memory usage: 2.1+ MB
```

0.4 Data preprocessing

1) Change the `object` columns into numeric `float64` columns.

2) Impute the missing values. Use `mean` for interval variables and `mode` for categorical variables
3) Dropping `ID` and few other variables from analysis.
4) Creating dummies for categorical variables.
5) Splitting the data into training and testing set

[8]:
```
# Turn object columns into numeric float64 columns

for i in data.columns:
    if data[i].dtype == 'object':
        data[i] = data[i].str.decode('utf-8')
```

[9]:
```
# Number of missing values

data.isnull().sum()
```

[9]:
```
TargetB              0
ID                   0
TargetD           4843
GiftCnt36            0
GiftCntAll           0
GiftCntCard36        0
GiftCntCardAll       0
GiftAvgLast          0
GiftAvg36            0
GiftAvgAll           0
GiftAvgCard36     1780
GiftTimeLast         0
GiftTimeFirst        0
PromCnt12            0
PromCnt36            0
PromCntAll           0
PromCntCard12        0
PromCntCard36        0
PromCntCardAll       0
StatusCat96NK     9686
StatusCatStarAll     0
DemCluster           0
DemAge            2407
DemGender         9686
```

```
DemHomeOwner        9686
DemMedHomeValue        0
DemPctVeterans         0
DemMedIncome        2357
dtype: int64
```

[10]:
```python
# Impute missing values --> mean for interval, mode for categorical

for i in data.columns:
    if data[i].dtype == "float64":
        data[i].fillna(data[i].mean(), inplace = True)
    else:
        data[i].fillna(data[i].mode(), inplace = True)
```

[11]:
```python
data.isnull().sum()
```

[11]:
```
TargetB                0
ID                     0
TargetD                0
GiftCnt36              0
GiftCntAll             0
GiftCntCard36          0
GiftCntCardAll         0
GiftAvgLast            0
GiftAvg36              0
GiftAvgAll             0
GiftAvgCard36          0
GiftTimeLast           0
GiftTimeFirst          0
PromCnt12              0
PromCnt36              0
PromCntAll             0
PromCntCard12          0
PromCntCard36          0
PromCntCardAll         0
StatusCat96NK       9686
StatusCatStarAll       0
DemCluster             0
DemAge                 0
DemGender           9686
DemHomeOwner        9686
DemMedHomeValue        0
DemPctVeterans         0
DemMedIncome           0
dtype: int64
```

[12]:
```python
# How many number of variables in the categorical Variables

data[[ "DemCluster", "DemGender", "DemHomeOwner", "StatusCat96NK"]].nunique()
```

[12]:
```
DemCluster       54
DemGender         0
DemHomeOwner      0
StatusCat96NK     0
dtype: int64
```

[13]:
```python
# Dropping Id and few other variables

data = data.drop(['ID', 'StatusCat96NK', 'DemGender', 'DemHomeOwner'], axis = 1)
```

[14]:
```python
# Create dummies for the categorical variables

data = pd.get_dummies(data, drop_first=True)
data.head()
```

[14]:
```
   TargetB    TargetD  GiftCnt36  GiftCntAll  GiftCntCard36  GiftCntCardAll  \
0        0  15.624344          2           4              1               3
1        0  15.624344          1           8              0               3
2        1   4.000000          6          41              3              20
3        1  10.000000          3          12              3               8
4        0  15.624344          1           1              1               1

   GiftAvgLast  GiftAvg36  GiftAvgAll  GiftAvgCard36  …  PromCntAll  \
0         17.0      13.50        9.25      17.000000  …          26
1         20.0      20.00       15.88      14.224431  …          79
2          6.0       5.17        3.73       5.000000  …          51
3         10.0       8.67        8.50       8.670000  …          44
4         20.0      20.00       20.00      20.000000  …          13

   PromCntCard12  PromCntCard36  PromCntCardAll  StatusCatStarAll  DemCluster  \
0              3              8              13                 0           0
1              5              5              24                 0          23
2              5             11              22                 1           0
3              2              6              16                 1           0
4              4              7               6                 0          35

      DemAge  DemMedHomeValue  DemPctVeterans  DemMedIncome
0  59.150845                0               0  53513.457361
1  67.000000           186800              85  53513.457361
2  59.150845            87600              36  38750.000000
3  59.150845           139200              27  38942.000000
4  53.000000           168100              37  71509.000000
```

```
[5 rows x 24 columns]
```

[15]:
```
# Split data into X and y

X = data.drop(['TargetB', 'TargetD'], axis = 1)
y = data['TargetB'].astype('category')
```

[16]:
```
# Split the data into train:test 70:30 split

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

0.5 Logistic Regression

[ ]:
```
log_model = LogisticRegression(penalty='l1', solver='liblinear', C = 1e9).
↪fit(X_train, y_train)
```

[ ]:
```
accuracy_score(log_model.predict(X_test), y_test)
```

0.6 Step-wise Regression

[ ]:
```
#execute this function once, as there are no pre-existing libraries in python␣
↪to run step wise regression model

def stepwise_selection(data, target, SL_in = 0.05, SL_out = 0.05):
    initial_features = data.columns.tolist()
    best_features = []

    while (len(initial_features)>0):
        remaining_features = list(set(initial_features) - set(best_features))
        new_pval = pd.Series(index = remaining_features, dtype='float64')

        for new_column in remaining_features:
            model = sm.OLS(target, sm.
↪add_constant(data[best_features+[new_column]])).fit()
            new_pval[new_column] = model.pvalues[new_column]
        min_p_value = new_pval.min()

        if (min_p_value < SL_in):
            best_features.append(new_pval.idxmin())

            while(len(best_features)>0):
                best_features_with_constant = sm.
↪add_constant(data[best_features])
                p_values = sm.OLS(target, best_features_with_constant).fit().
↪pvalues[1:]

                max_p_value = p_values.max()
```

```python
                if(max_p_value >= SL_out):
                    excluded_feature = p_values.idxmax()
                    best_features.remove(excluded_feature)
                else:
                    break
        else:
            break
    return best_features

##Source: https://www.analyticsvidhya.com/blog/2020/10/
 ↪a-comprehensive-guide-to-feature-selection-using-wrapper-methods-in-python/
```

[ ]:
```python
X1 = stepwise_selection(X,y)
```

[ ]:
```python
# Variables that are selected

print(X1)
```

[ ]:
```python
X_stepwise = data[X1]
```

[ ]:
```python
# Split the data in 70:30

X_train, X_test, y_train, y_test = train_test_split(X_stepwise, y, test_size=0.
 ↪3)
```

[24]:
```python
# logistic Modelling

log_model = LogisticRegression(penalty='l1', solver='liblinear', C = 1e9).
 ↪fit(X_train, y_train)

accuracy_score(log_model.predict(X_test), y_test)
```

[24]: 0.5784583620096352

0.7 Decision Tree Modelling

[25]:
```python
# Split the data in 70:30

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

[26]:
```python
# Decision Tree modelling

DT_model = DecisionTreeClassifier().fit(X_train, y_train)

# Accuracy
```

```
accuracy_score(DT_model.predict(X_test), y_test)
```

[26]: 0.5313145216792843

0.8 Gradient Boosting

[27]:
```
# Gradient boosting

gb_model = GradientBoostingClassifier()
gb_model.fit(X_train, y_train)

# Accuracy

accuracy_score(gb_model.predict(X_test), y_test)
```

[27]: 0.5719201651754989

0.9 Random Forest Modeling

[28]:
```
rf_model = RandomForestClassifier()
rf_model.fit(X_train, y_train)

# Accuracy

accuracy_score(rf_model.predict(X_test), y_test)
```

[28]: 0.5646937370956642

0.10 SVM Model

0.10.1 a. Linear Kernel

[29]:
```
svm_model = SVC(kernel= 'linear').fit(X_train, y_train)

# Accuracy

accuracy_score(svm_model.predict(X_test), y_test)
```

[29]: 0.5588437715072264

0.10.2 b. Polynomial Kernel

[30]:
```
svm_model = SVC(kernel= 'poly').fit(X_train, y_train)

# Accuracy

accuracy_score(svm_model.predict(X_test), y_test)
```

[30]: 0.4865794907088782

0.10.3 c. Gaussian Kernel

[31]:
```
svm_model = SVC(kernel= 'rbf').fit(X_train, y_train)

# Accuracy

accuracy_score(svm_model.predict(X_test), y_test)
```

[31]: 0.526152787336545