Practical Activity Classification using SVM

1 Practical Activity

1.1 Classi ication U sing S upport Vector Machines

This notebook is an exercise for developing a SVM classifier for predicting customer attrition. We apply the concepts discussed in Week 8. We walk through SVM Classifier in this practical and will use SVM regression model in the next practical activity.

Note: this activity is unmarked. It develops your skills for predictive model development using SVM.

2 ATH LEAPS Bank Data

In this task, we will predict which customers in the future is more likely to churn or to stop availing the banks' services. Once an adequate prediction model is developed, the bank will be better informed on how to develop strategies to retain customers or at least lose less customers in the future.

This practical will build Suport Vector Machine in predicting customer attrition using the bank data set.

Data source: https://www.kaggle.com/giananncheta/predictive-analysis-of-bank-churners/data

According to the dataset description - "PLEASE IGNORE THE LAST 2 COLUMNS (NAIVE BAYES CLAS…). I SUGGEST TO RATHER DELETE IT BEFORE DOING ANYTHING"

```python
# Loading the dataset
import pandas as pd
```

[1]:
```python
df = pd.read_csv('BankChurners.csv')

# check the columns.
df.columns
```

[2]:
```
Index(['CLIENTNUM', 'Attrition_Flag', 'Customer_Age', 'Gender',
```

[2]:
```
       'Dependent_count', 'Education_Level', 'Marital_Status',
       'Income_Category', 'Card_Category', 'Months_on_book',
       'Total_Relationship_Count', 'Months_Inactive_12_mon',
       'Contacts_Count_12_mon', 'Credit_Limit', 'Total_Revolving_Bal',
       'Avg_Open_To_Buy', 'Total_Amt_Chng_Q4_Q1', 'Total_Trans_Amt',
```

```
       'Total_Trans_Ct', 'Total_Ct_Chng_Q4_Q1', 'Avg_Utilization_Ratio',
       'Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mo
n_Dependent_count_Education_Level_Months_Inactive_12_mon_1',
       'Naive_Bayes_Classifier_Attrition_Flag_Card_Category_Contacts_Count_12_mo
n_Dependent_count_Education_Level_Months_Inactive_12_mon_2'],
      dtype='object')
```

After loading the dataset, we remove the unwanted columns. In this case, we will remove the last two columns and the client number (CLIENTNUM) columns.

[3]:
```
df = df.iloc[:, 1:-2]

#remove the client num attribute
#df = df.iloc[:, 1:]
```

[4]:
```
df.head()
```

[4]:
```
      Attrition_Flag  Customer_Age Gender  Dependent_count Education_Level  \
0  Existing Customer            45      M                3     High School
1  Existing Customer            49      F                5        Graduate
2  Existing Customer            51      M                3        Graduate
3  Existing Customer            40      F                4     High School
4  Existing Customer            40      M                3      Uneducated

  Marital_Status Income_Category Card_Category  Months_on_book  \
0        Married     $60K - $80K          Blue              39
1         Single  Less than $40K          Blue              44
2        Married    $80K - $120K          Blue              36
3        Unknown  Less than $40K          Blue              34
4        Married     $60K - $80K          Blue              21

   Total_Relationship_Count  Months_Inactive_12_mon  Contacts_Count_12_mon  \
0                         5                       1                      3
1                         6                       1                      2
2                         4                       1                      0
3                         3                       4                      1
4                         5                       1                      0

   Credit_Limit  Total_Revolving_Bal  Avg_Open_To_Buy  Total_Amt_Chng_Q4_Q1  \
0       12691.0                  777          11914.0                 1.335
1        8256.0                  864           7392.0                 1.541
2        3418.0                    0           3418.0                 2.594
3        3313.0                 2517            796.0                 1.405
4        4716.0                    0           4716.0                 2.175

   Total_Trans_Amt  Total_Trans_Ct  Total_Ct_Chng_Q4_Q1  Avg_Utilization_Ratio
0             1144              42                1.625                  0.061
```

```
1            1291            33              3.714                    0.105
2            1887            20              2.333                    0.000
3            1171            20              2.333                    0.760
4             816            28              2.500                    0.000
```

Now we check the variable types.

[5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10127 entries, 0 to 10126
Data columns (total 20 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Attrition_Flag            10127 non-null  object
 1   Customer_Age              10127 non-null  int64
 2   Gender                    10127 non-null  object
 3   Dependent_count           10127 non-null  int64
 4   Education_Level           10127 non-null  object
 5   Marital_Status            10127 non-null  object
 6   Income_Category           10127 non-null  object
 7   Card_Category             10127 non-null  object
 8   Months_on_book            10127 non-null  int64
 9   Total_Relationship_Count  10127 non-null  int64
 10  Months_Inactive_12_mon    10127 non-null  int64
 11  Contacts_Count_12_mon     10127 non-null  int64
 12  Credit_Limit              10127 non-null  float64
 13  Total_Revolving_Bal       10127 non-null  int64
 14  Avg_Open_To_Buy           10127 non-null  float64
 15  Total_Amt_Chng_Q4_Q1      10127 non-null  float64
 16  Total_Trans_Amt           10127 non-null  int64
 17  Total_Trans_Ct            10127 non-null  int64
 18  Total_Ct_Chng_Q4_Q1       10127 non-null  float64
 19  Avg_Utilization_Ratio     10127 non-null  float64
dtypes: float64(5), int64(9), object(6)
memory usage: 1.5+ MB
```

[6]:
```
df.shape
```

[6]:
```
(10127, 20)
```

This dataset is larger than the datasets we used in the previous weeks. Let us now check the class distributions.

[7]:
```
df.Attrition_Flag.value_counts()
```

[7]:
```
Existing Customer    8500
Attrited Customer    1627
```

```
Name: Attrition_Flag, dtype: int64
```

We observe that, we have more samples of the "Attrited Customer" than "Existing Customer". This means the model we will train using this dataset is expected to be baised towards the "Existing Customer" which means the model will predict most of the test instances as "Existing Customer". We will test this in the last part of the excercise.

2.1 Note

We have 19 features in this dataset. Using all the feautures will take a long time to train the model. Therefore, we will select a subset of the features to train our model. Ideally, we will measure correlations and chi^2 or other statistics to find the best set of variables to use. However, for this practical we select the following variables: - Customer_Age (int64) - Income_Category (object) - Credit_Limit (float64) - Total_Revolving_Bal (int64) - Total_Trans_Amt (int64)

Among our selected variables, Income category is a categorical variable, we need to encode this.

[8]:
```python
#Encoding the categorical variables
from sklearn import preprocessing

le = preprocessing.LabelEncoder()


df['en_Income_Category'] = le.fit_transform(df['Income_Category'])
df['en_Attrition_Flag'] = le.fit_transform(df['Attrition_Flag'])
```

[9]:
```python
features = ['Customer_Age', 'en_Income_Category', 'Credit_Limit',
            'Total_Revolving_Bal', 'Total_Trans_Amt' ]
```

3 Feature scaling

[10]:
```python
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

X_scaled = scaler.fit_transform(df[features])

#reverting back to df
X = pd.DataFrame(X_scaled)

X['target'] = df['en_Attrition_Flag']
```

4 Building a SVM classifier

sklearn provides different SVM implementations. https://scikit-learn.org/stable/modules/svm.html#svm-classification

We will explore three types of kernels in this practical, linear, polynomial and rbf.

[11]:
```
from sklearn.model_selection import train_test_split

train, test = train_test_split(X, test_size = 0.3, stratify = X['target'])

X_train = train.drop('target', axis=1)
y_train = train['target']

X_test = test.drop('target', axis = 1)
y_test = test['target']
```

[12]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm
```

[13]:
```
C = 1.0  # SVM regularization parameter
svc = svm.SVC(kernel='linear', C=C).fit(X_train, y_train)
```

4.1 Evaluate

[14]:
```
from sklearn.metrics import accuracy_score

predictions = svc.predict(X_test)
acc = accuracy_score(y_test, predictions)

print(f'Accuracy of the model is {acc}')
```

```
Accuracy of the model is 0.8394208621256992
```

[15]:
```
X_test.shape
```

[15]:
```
(3039, 5)
```

The model is almost 84% accurate which seems to be a good model at the first attmept without tuning any params. However, as we mentioned earlier, we have an imbalanced dataset and the model should be biased towards the majority class, in this case, “Existing Customer” which is encoded as 1.

We now inspect the predicitons of the model. We can print the predictions. We have 3039 test samples i.e., the predictions array is too big. We can also count the number of 0s and 1s in the predicitons array. To count the elements in predictions, we need to use python collections library.

[16]:
```
#print(predictions.tolist()) uncomment and check the array.
import collections

counter = collections.Counter(predictions.tolist())
print(counter)
```

```
Counter({1: 3039})
```

Interestingly, we observe that the model has predicted all the test samples as instances of class 1 i.e., it does not predict class 0 at all. Still it has an accuracy of 84%.

The above shows the problem with imbalanced dataset and the problem of accuracy measure.

4.2 classification_report

sklearn provides a detailed classification performance results as https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html

[17]:
```python
from sklearn.metrics import classification_report

target_names = ['Attrited Customer', 'Existing Customer']
print(classification_report(y_test, predictions, target_names=target_names))
```

```
                   precision    recall  f1-score   support

Attrited Customer       0.00      0.00      0.00       488
Existing Customer       0.84      1.00      0.91      2551

         accuracy                           0.84      3039
        macro avg       0.42      0.50      0.46      3039
     weighted avg       0.70      0.84      0.77      3039


C:\Users\islmy008\Anaconda3\lib\site-
packages\sklearn\metrics\_classification.py:1221: UndefinedMetricWarning:
Precision and F-score are ill-defined and being set to 0.0 in labels with no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

The classificaiton report shows that we have 488 instances of attrited customer in our test set and the model failed to predict any of them.

4.3 Dealing with imbalanced dataset

The simplest thing we can do is to balance the dataset by sampling the majority class to make the distribution equal.

[18]:
```python
df_minor = X.loc[X['target']==0]
```

[19]:
```python
df_minor.shape
```

[19]:
```
(1627, 6)
```

[20]:
```python
#sampling the majority class
df_major = X.loc[X['target']==1].sample(n=df_minor.shape[0])
```

[21]:
```python
df_major.shape
```

[21]: (1627, 6)

df_major and df_minor has same number of samples. We need to merge them together and use for training.

[22]:
```
df_sub = df_major.append(df_minor)

df_sub.head()
```

[22]:

	0	1	2	3	4	target
4692	-0.165406	1.419670	-0.651930	0.776966	-0.096288	1
9043	1.830498	-1.238799	-0.710247	0.203922	0.775079	1
668	0.832546	0.090436	2.848054	1.661686	-0.862267	1
166	0.084082	0.755053	-0.772524	-0.511461	-0.880518	1
7288	0.333570	1.419670	-0.670085	-0.383846	0.167183	1

[23]:
```
df_sub.shape
```

[23]: (3254, 6)

[24]:
```
# shuffle the dataset. In the current version first 1627 are of class 1 and␣
↪last 1627 are of class 0

df_sub = df_sub.sample(frac=1)

df_sub.head()
```

[24]:

	0	1	2	3	4	target
7249	-0.539638	0.755053	-0.649949	1.572109	0.165122	1
9307	0.084082	-1.238799	1.688875	-1.426858	0.038833	0
318	0.333570	0.755053	-0.638836	1.113184	-0.994738	1
2691	1.082034	-1.238799	-0.612539	-1.426858	-0.259080	1
9687	0.707802	-1.238799	-0.418774	-1.426858	1.230780	0

[25]:
```
#training model

train, test = train_test_split(df_sub, test_size = 0.3, stratify =␣
↪df_sub['target'])

X_train = train.drop('target', axis=1)
y_train = train['target']

X_test = test.drop('target', axis = 1)
y_test = test['target']

svc = svm.SVC(kernel='linear', C=1).fit(X_train, y_train)
```

[26]:
```
#evaluate model
predictions = svc.predict(X_test)
acc = accuracy_score(y_test, predictions)

print(f'Accuracy of the new model is {acc}')
```

```
Accuracy of the new model is 0.72978505629478
```

[27]:
```
counter = collections.Counter(predictions.tolist())
print(counter)
```

```
Counter({1: 497, 0: 480})
```

Predicted both 0 and 1 classes.

[28]:
```
print(classification_report(y_test, predictions, target_names=target_names))
```

```
                   precision    recall  f1-score   support

Attrited Customer       0.73      0.72      0.73       488
Existing Customer       0.73      0.74      0.73       489

         accuracy                           0.73       977
        macro avg       0.73      0.73      0.73       977
     weighted avg       0.73      0.73      0.73       977
```

Though our new model predicts both classes, the performance is not probimising. Let us use other types of kernels and compare their performances.

[29]:
```
rbf_svc = svm.SVC(kernel='rbf', gamma=0.7, C=C).fit(X_train, y_train)
poly_svc = svm.SVC(kernel='poly', degree=4, C=C).fit(X_train, y_train)
```

[30]:
```
predictions = rbf_svc.predict(X_test)
print(f'Accuracy of the model with rbf kernel is {accuracy_score(y_test,␣
↪predictions)}')

predictions = poly_svc.predict(X_test)
print(f'Accuracy of the model with poly kernel is {accuracy_score(y_test,␣
↪predictions)}')
```

```
Accuracy of the model with rbf kernel is 0.8065506653019447
Accuracy of the model with poly kernel is 0.7553735926305015
```

We observe that rbf kernel gives the most promising model for our dataset. We can also find the support vectors.

[31]:
```
rbf_svc.n_support_
```

[31]:
```
array([599, 633])
```

[32]:
```
rbf_svc.support_vectors_
```

[32]:
```
array([[ 1.83049827, -0.5741815 ,  0.35507591,  0.59045052,  1.33234176],
       [-0.78912553, -1.23879873, -0.53221602,  1.6616857 , -0.8157544 ],
       [ 0.0840824 , -0.5741815 ,  0.15723974, -1.42685834, -0.47162333],
       …,
       [ 0.45831437,  0.09043572, -0.44353084,  1.6616857 ,  0.19191083],
       [ 0.95729034, -0.5741815 , -0.69781304,  0.33276508, -0.96706609],
       [ 0.33357038,  1.41967017, -0.38873528,  0.01863426, -0.17223813]])
```

In the above, rbf_svc.n_support_ shows that the model has learned 588 support vectors for class 0 and 633 support vectors for class 1.

rbf_svc.support_vectors_ shows the support vectors for each class.