Machine Learning Model Deployment with IBM Cloud Watson Studio

Team member

732721104015 : Dhivya S

Phase 4 Submission Document

DEVELOPMENT PART 2

INTRODUCTION:

- *In the second part of our model development journey, we are advancing our customer churn prediction model to new heights.*
- *Building upon the foundations laid in part 1, this phase focuses on implementing advanced machine learning techniques, hyperparameter tuning, and cross-validation to enhance model performance.*
- *Feature engineering and addressing class imbalance remain pivotal, and we'll revisit dimensionality reduction techniques like PCA for further optimization.*
- *Model interpretability takes center stage, allowing us to gain deeper insights into customer behavior.*
- *Beyond model refinement, we are concentrating on deployment strategies, creating user-friendly interfaces, and prioritizing data privacy and security.*
- *Our goal is to deliver a robust, interpretable, and accessible solution that maximizes its impact on reducing customer churn and business success, ensuring a seamless transition from development to deployment.*

MODEL BUILDING:

- *In the realm of customer churn prediction, this comprehensive model building process stands out as a robust solution to address the challenges posed by imbalanced datasets. The dataset at hand contains critical customer information, and the goal is to predict whether a customer will churn. With the key focus on enhancing the performance of the predictive model, several pivotal steps were taken.*
- *First, libraries such as pandas, scikit-learn, and imbalanced-learn were imported to empower the model building process. These tools paved the way for data manipulation, resampling, and model evaluation.*
- *The initial Decision Tree Classifier was constructed, but it became evident that, given the dataset's imbalance, accuracy alone wasn't an ideal metric. The hunt for a more accurate model led to the implementation of SMOTEENN resampling, an ingenious approach that combines over-sampling and under-sampling techniques. The result was a Decision Tree Classifier with remarkable performance. It boasted a striking 93% accuracy rate, thanks to the meticulous fine-tuning of parameters.*

Moreover, the recall and precision for both churned and non-churned customers soared, indicating the model's capability to effectively identify potential churners.

- The model building journey didn't stop there. A Random Forest Classifier was enlisted for comparison and, once again, fine-tuned to unleash its full predictive power. This Random Forest Classifier pre-SMOTEENN, and post-SMOTEENN, delivered promising results. The post-SMOTEENN model achieved an impressive accuracy rate of 94% while maintaining balanced recall and precision figures.

- In the pursuit of dimensionality reduction, Principal Component Analysis (PCA) was put to the test, but it didn't yield significant improvements in model performance. Hence, it was the Random Forest Classifier after SMOTEENN resampling that emerged as the model of choice.

Code:

In [1]:

```
import pandas as pd


from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.metrics import recall_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.tree import DecisionTreeClassifier
from imblearn.combine import SMOTEENN
```

In [2]:

```
df=pd.read_csv("ML Model Dataset.csv")
df.head()
```

Out [2]:

	Unnamed: 0	SeniorCitizen	MonthlyCharges	TotalCharges	Churn	gender_Female	gender_Male	Partner_No	Partner_Yes	Dependents_
0	0	0	29.85	29.85	0	1	0	0	1	
1	1	0	56.95	1889.50	0	0	1	1	0	
2	2	0	53.85	108.15	1	0	1	1	0	
3	3	0	42.30	1840.75	0	0	1	1	0	
4	4	0	70.70	151.65	1	1	0	1	0	

5 rows × 52 columns

In [3]:

```
df=df.drop('Unnamed: 0',axis=1)
```

In [4]:

```
x=df.drop('Churn',axis=1)
```

```
x
```

Out [4]:

	SeniorCitizen	MonthlyCharges	TotalCharges	gender_Female	gender_Male	Partner_No	Partner_Yes	Dependents_No	Dependents
0	0	29.85	29.85	1	0	0	1	1	
1	0	56.95	1889.50	0	1	1	0	1	
2	0	53.85	108.15	0	1	1	0	1	
3	0	42.30	1840.75	0	1	1	0	1	
4	0	70.70	151.65	1	0	1	0	1	
5	0	99.65	820.50	1	0	1	0	1	
6	0	89.10	1949.40	0	1	1	0	0	
7	0	29.75	301.90	1	0	1	0	1	
8	0	104.80	3046.05	1	0	0	1	1	
9	0	56.15	3487.95	0	1	1	0	0	
10	0	49.95	587.45	0	1	0	1	0	
11	0	18.95	326.80	0	1	1	0	1	
12	0	100.35	5681.10	0	1	0	1	1	
13	0	103.70	5036.30	0	1	1	0	1	
14	0	105.50	2686.05	0	1	1	0	1	
15	0	113.25	7895.15	1	0	0	1	0	
16	0	20.65	1022.95	1	0	1	0	1	
17	0	106.70	7382.25	0	1	1	0	0	
7014	0	93.05	1679.40	1	0	1	0	1	
7015	0	44.20	403.35	1	0	1	0	1	
7016	0	73.35	931.55	0	1	1	0	1	
7017	0	64.10	4326.25	1	0	0	1	1	
7018	1	44.40	263.05	1	0	1	0	1	
7019	0	20.05	39.25	1	0	1	0	1	
7020	1	60.00	3316.10	0	1	0	1	1	
7021	1	75.75	75.75	0	1	1	0	1	
7022	0	69.50	2625.25	0	1	1	0	1	
7023	0	102.95	6886.25	1	0	1	0	1	
7024	0	78.70	1495.10	0	1	1	0	1	
7025	0	60.65	743.30	1	0	1	0	1	
7026	0	21.15	1419.40	1	0	1	0	1	
7027	0	84.80	1990.50	0	1	0	1	0	
7028	0	103.20	7362.90	1	0	0	1	0	
7029	0	29.60	346.45	1	0	0	1	0	
7030	1	74.40	306.60	0	1	0	1	1	
7031	0	105.65	6844.50	0	1	1	0	1	

7032 rows × 50 columns

In [5]:

```
y=df['Churn']
```

```
y
```

Out [5]:

```
0     0
1     0
2     1
3     0
4     1
5     1
6     0
7     0
8     1
9     0
10    0
11    0
12    0
13    1
14    0
15    0
16    0
17    0
18    1
19    0
20    1
21    0
22    1
23    0
24    0
25    0
26    1
27    1
28    0
29    1
     ..
7002  0
7003  0
7004  0
7005  0
7006  0
7007  1
7008  0
7009  0
7010  1
7011  0
7012  0
7013  0
7014  0
7015  1
7016  0
7017  0
7018  0
7019  0
7020  0
7021  1
7022  0
7023  1
7024  0
7025  0
7026  0
7027  0
7028  0
7029  0
7030  1
7031  0
Name: Churn, Length: 7032, dtype: int64
```

Train Test Split

In [6]:

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2)
```

Decision Tree Classifier

In [7]:

```
model_dt=DecisionTreeClassifier(criterion = "gini",random_state = 100,max_depth=6,min_samples_leaf=8)
```

In [8]:

```
model_dt.fit(x_train,y_train)
```

Out[8]:

```
DecisionTreeClassifier(max_depth=6,min_samples_leaf=8,random_state=100)
```

In [9]:

```
y_pred=model_dt.predict(x_test)
y_pred
```

Out[9]:

```
array([0, 0, 1, ..., 0, 0, 0], dtype=int64)
```

In [10]:

```
model_dt.score(x_test,y_test)
```

Out[10]:

```
0.7818052594171997
```

In [11]:

```
print(classification_report(y_test,y_pred,labels=[0,1]))
```

```
              precision    recall  f1-score   support

           0       0.82      0.89      0.86      1023
           1       0.63      0.49      0.55       384

    accuracy                           0.78      1407
   macro avg       0.73      0.69      0.70      1407
weighted avg       0.77      0.78      0.77      1407
```

In [12]:

```
sm = SMOTEENN()
X_resampled, y_resampled = sm.fit_sample(x,y)
```

In [13]:

```
xr_train,xr_test,yr_train,yr_test=train_test_split(X_resampled, y_resampled,test_size=0.2)
```

In [14]:

```
model_dt_smote=DecisionTreeClassifier(criterion = "gini",random_state = 100,max_depth=6,min_samples_leaf=8)
```

In [15]:

```
model_dt_smote.fit(xr_train,yr_train)
yr_predict = model_dt_smote.predict(xr_test)
model_score_r = model_dt_smote.score(xr_test, yr_test)
print(model_score_r)
print(metrics.classification_report(yr_test, yr_predict))
```

```
0.9344122657580920
              precision    recall  f1-score   support

           0       0.97      0.88      0.93       540
           1       0.91      0.98      0.94       634

    accuracy                           0.93      1174
```

```
 macro avg    0.94    0.93    0.93    1174
weighted avg    0.94    0.93    0.93    1174
```

In [16]:

```
print(metrics.confusion_matrix(yr_test, yr_predict))
[[477  63]
 [ 14 620]]
```

Random Forest Classifier

In [17]:

```
from sklearn.ensemble import RandomForestClassifier
```

In [18]:

```
model_rf=RandomForestClassifier(n_estimators=100, criterion='gini', random_state = 100,max_depth=6,
min_samples_leaf=8)
```

In [19]:

```
model_rf.fit(x_train,y_train)
```

Out[19]:

```
RandomForestClassifier(max_depth=6, min_samples_leaf=8, random_state=100)
```

In [20]:

```
y_pred=model_rf.predict(x_test)
```

In [21]:

```
model_rf.score(x_test,y_test)
```

Out[21]:

```
0.7953091684434968
```

In [22]:

```
print(classification_report(y_test,y_pred,labels=[0,1]))
```

```
              precision    recall  f1-score   support

           0       0.82      0.92      0.87      1023
           1       0.69      0.45      0.55       384

    accuracy                           0.80      1407
   macro avg       0.75      0.69      0.71      1407
weighted avg       0.78      0.80      0.78      1407
```

In [23]:

```
sm = SMOTEENN()
X_resampled1, y_resampled1 = sm.fit_sample(x,y)
```

In [24]:

```
xr_train1,xr_test1,yr_train1,yr_test1=train_test_split(X_resampled1, y_resampled1,test_size=0.2)
```

In [25]:

```
model_rf_smote=RandomForestClassifier(n_estimators=100, criterion='gini', random_state = 100,max_depth=6,
min_samples_leaf=8)
```

In [26]:

```
model_rf_smote.fit(xr_train1,yr_train1)
```

Out[26]:

```
RandomForestClassifier(max_depth=6, min_samples_leaf=8, random_state=100)
```

In [27]:

```
yr_predict1 = model_rf_smote.predict(xr_test1)
```

In [28]:

```
model_score_r1 = model_rf_smote.score(xr_test1, yr_test1)
```

In [29]:

```
print(model_score_rl)
print(metrics.classification_report(yr_testl,yr_predictl))
```

```
0.9427350427350427
              precision    recall  f1-score   support

           0       0.95      0.92      0.93       518
           1       0.94      0.96      0.95       652

    accuracy                           0.94      1170
   macro avg       0.94      0.94      0.94      1170
weighted avg       0.94      0.94      0.94      1170
```

In [30]:

```
print(metrics.confusion_matrix(yr_testl,yr_predictl))
```

```
[[478  40]
 [ 27 625]]
```

Performing PCA

In [31]:

```
from sklearn.decomposition import PCA
pca = PCA(0.9)
xr_train_pca = pca.fit_transform(xr_trainl)
xr_test_pca = pca.transform(xr_testl)
explained_variance = pca.explained_variance_ratio_
```

In [32]:

```
model=RandomForestClassifier(n_estimators=100, criterion='gini', random_state = 100,max_depth=6,
min_samples_leaf=8)
```

In [33]:

```
model.fit(xr_train_pca,yr_trainl)
```

Out[33]:

```
RandomForestClassifier(max_depth=6, min_samples_leaf=8, random_state=100)
```

In [34]:

```
yr_predict_pca = model.predict(xr_test_pca)
```

In [35]:

```
model_score_r_pca = model.score(xr_test_pca, yr_testl)
```

In [36]:

```
print(model_score_r_pca)
print(metrics.classification_report(yr_testl, yr_predict_pca))
```

```
0.7239316239316239
              precision    recall  f1-score   support

           0       0.72      0.61      0.66       518
           1       0.72      0.81      0.77       652

    accuracy                           0.72      1170
   macro avg       0.72      0.71      0.71      1170
weighted avg       0.72      0.72      0.72      1170
```

Pickling the model

In [37]:

```
import pickle
In [38]:

filename = 'model.sav'
In [39]:

pickle.dump(model_rf_smote, open(filename, 'wb'))
In [40]:

load_model = pickle.load(open(filename, 'rb'))
In [41]:

model_score_r1 = load_model.score(xr_test1, yr_test1)
In [42]:

model_score_r1
Out[42]:

0.9427350427350427
```

Model Deployment:

Below code showcases the development of a customer churn prediction web application using Flask. It begins by importing essential libraries, including Pandas and Scikit-Learn for data processing and model deployment. The web application allows users to input customer information, which is then utilized to predict the likelihood of a customer churning. A pre-trained Random Forest Classifier model is employed for this task, and the prediction outcome, along with a confidence score, is displayed. This tool is valuable for businesses seeking to identify and retain at-risk customers. To implement it fully, an HTML template ("home.html") for user interaction and Flask routes for handling requests are essential.

Code:

```
import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier

from sklearn import metrics

from flask import Flask, request, render_template

import pickle


app = Flask("__name__")


df_1=pd.read_csv("first_telc.csv")


q = ""


@app.route("/")
```

```
def loadPage():
        return render_template('home.html', query="")



@app.route("/", methods=['POST'])
def predict():

  '''
  SeniorCitizen
  MonthlyCharges
  TotalCharges
  gender
  Partner
  Dependents
  PhoneService
  MultipleLines
  InternetService
  OnlineSecurity
  OnlineBackup
  DeviceProtection
  TechSupport
  StreamingTV
  StreamingMovies
  Contract
  PaperlessBilling
  PaymentMethod
  tenure
  '''



  inputQuery1 = request.form['query1']
  inputQuery2 = request.form['query2']
  inputQuery3 = request.form['query3']
  inputQuery4 = request.form['query4']
  inputQuery5 = request.form['query5']
```

```
inputQuery6 = request.form['query6']

inputQuery7 = request.form['query7']

inputQuery8 = request.form['query8']

inputQuery9 = request.form['query9']

inputQuery10 = request.form['query10']

inputQuery11 = request.form['query11']

inputQuery12 = request.form['query12']

inputQuery13 = request.form['query13']

inputQuery14 = request.form['query14']

inputQuery15 = request.form['query15']

inputQuery16 = request.form['query16']

inputQuery17 = request.form['query17']

inputQuery18 = request.form['query18']

inputQuery19 = request.form['query19']


model = pickle.load(open("model.sav","rb"))


data = [[inputQuery1, inputQuery2, inputQuery3, inputQuery4, inputQuery5, inputQuery6, inputQuery7,
    inputQuery8, inputQuery9, inputQuery10, inputQuery11, inputQuery12, inputQuery13, inputQuery14,
    inputQuery15, inputQuery16, inputQuery17, inputQuery18, inputQuery19]]

new_df = pd.DataFrame(data, columns = ['SeniorCitizen', 'MonthlyCharges', 'TotalCharges', 'gender',
                    'Partner', 'Dependents', 'PhoneService', 'MultipleLines', 'InternetService',
                    'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport',
                    'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling',
                    'PaymentMethod', 'tenure'])


df_2 = pd.concat([df_1, new_df], ignore_index = True)

# Group the tenure in bins of 12 months

labels = ["{0} - {1}".format(i, i + 11) for i in range(1, 72, 12)]


df_2['tenure_group'] = pd.cut(df_2.tenure.astype(int), range(1, 80, 12), right=False, labels=labels)

#drop column customerID and tenure

df_2.drop(columns= ['tenure'], axis=1, inplace=True)
```

```
new_df__dummies = pd.get_dummies(df_2[['gender','SeniorCitizen','Partner','Dependents','PhoneService',
    'MultipleLines','InternetService','OnlineSecurity','OnlineBackup',
    'DeviceProtection','TechSupport','StreamingTV','StreamingMovies',
    'Contract','PaperlessBilling','PaymentMethod','tenure_group']])


#final_df=pd.concat([new_df__dummies, new_dummy], axis=1)


single = model.predict(new_df__dummies.tail(1))
probablity = model.predict_proba(new_df__dummies.tail(1))[:,1]

if single==1:
   o1 = "This customer is likely to be churned!!"
   o2 = "Confidence: {}".format(probablity*100)
else:
   o1 = "This customer is likely to continue!!"
   o2 = "Confidence: {}".format(probablity*100)

return render_template('home.html',output1=o1,output2=o2,
                query1 = request.form['query1'],
                query2 = request.form['query2'],
                query3 = request.form['query3'],
                query4 = request.form['query4'],
                query5 = request.form['query5'],
                query6 = request.form['query6'],
                query7 = request.form['query7'],
                query8 = request.form['query8'],
                query9 = request.form['query9'],
                query10 = request.form['query10'],
                query11 = request.form['query11'],
                query12 = request.form['query12'],
                query13 = request.form['query13'],
                query14 = request.form['query14'],
```

```
            query15 = request.form['query15'],
            query16 = request.form['query16'],
            query17 = request.form['query17'],
            query18 = request.form['query18'],
            query19 = request.form['query19'])

app.run()
```

`

Conclusion:

In conclusion, the process of model building and deployment is a crucial step in applying machine learning solutions to real-world problems. Through this journey, we began by importing necessary libraries and data, conducted data preprocessing, and employed various machine learning algorithms to build predictive models. We evaluated these models to select the one with the best performance, considering metrics like accuracy, precision, recall, and F1-score, especially when dealing with imbalanced datasets. Following the model building phase, we integrated deployment into the pipeline, enabling end-users to interact with the model via a web-based interface using Flask. By deploying the chosen model, we make its predictions accessible for practical decision-making, providing valuable insights into the data at hand. This entire process is a fundamental aspect of harnessing the potential of machine learning in real-world applications.