Week5_Nagarjuna_Devaray

February 18, 2024

[ ]:
```
import pandas as pd
from pycaret.classification import setup, compare_models, predict_model,␣
↪save_model, load_model
import pickle
from IPython.display import Code
```

0.1 Load dataset

[3]:
```
df = pd.read_csv("new_churn_data.csv")
df
```

[3]:
```
      tenure  MonthlyCharges  TotalCharges  Churn  MonthlyCharges_log  \
0          1           29.85         29.85      0            3.396185
1         34           56.95       1889.50      0            4.042174
2          2           53.85        108.15      1            3.986202
3         45           42.30       1840.75      0            3.744787
4          2           70.70        151.65      1            4.258446
…        …              …            …      …                 …
7027      24           84.80       1990.50      0            4.440296
7028      72          103.20       7362.90      0            4.636669
7029      11           29.60        346.45      0            3.387774
7030       4           74.40        306.60      1            4.309456
7031      66          105.65       6844.50      0            4.660132

      TotalCharges_Tenure_Ratio  MonthlyCharges_to_TotalCharges_Ratio  \
0                     29.850000                              1.000000
1                     55.573529                              0.030140
2                     54.075000                              0.497920
3                     40.905556                              0.022980
4                     75.825000                              0.466205
…                           …                                    …
7027                  82.937500                              0.042602
7028                 102.262500                              0.014016
7029                  31.495455                              0.085438
7030                  76.650000                              0.242661
7031                 103.704545                              0.015436

      Bank transfer (automatic)  Credit card (automatic)  Electronic check  \
```

```
0                                  0                             0                    0
1                                  0                             0                    1
2                                  0                             0                    1
3                                  1                             0                    1
4                                  0                             0                    0
…                               …                            …                  …
7027                               0                             0                    1
7028                               0                             1                    1
7029                               0                             0                    0
7030                               0                             0                    1
7031                               1                             0                    1

      Mailed check  Month-to-month  One year  Two year
0                0               0         0         0
1                1               1         1         0
2                1               0         0         0
3                0               1         1         0
4                0               0         0         0
…             …              …       …        …
7027             1               1         1         0
7028             0               1         1         0
7029             0               0         0         0
7030             1               0         0         0
7031             0               1         0         1

[7032 rows x 14 columns]
```

0.2 Initialize the auto ML environment

[4]:
```
automl_setup = setup(df, target='Churn')
```

```
<pandas.io.formats.style.Styler at 0x7f23bc4af2d0>
```

The output summarizes the setup information for the PyCaret auto ML environment.

Session id: 8322 - A unique identifier for the PyCaret session.

Target: Churn - The target variable for the classification task is Churn.

Target type: Binary - The target variable is binary, indicating a binary classification task (Churn or no Churn).

Original data shape: (7032, 14) - The original dataset has 7032 rows and 14 columns.

Transformed data shape: (7032, 14) - The transformed dataset after preprocessing remains the same size as the original dataset.

Transformed train set shape: (4922, 14) - The training set after preprocessing contains 4922 samples.

Transformed test set shape: (2110, 14) - The test set after preprocessing contains 2110 samples.

Numeric features: 13 - There are 13 numeric features in the dataset.

Preprocess: True - The data has been preprocessed.

Imputation type: simple - Simple imputation method has been used for handling missing values.

Numeric imputation: mean - Mean imputation has been applied to numeric features.

Categorical imputation: mode - Mode imputation has been applied to categorical features.

Fold Generator: StratifiedKFold - Stratified K-Fold cross-validation is used during model training.

Number: 10 - 10 folds are used in cross-validation.

CPU Jobs: -1 - The number of CPU jobs is set to -1, allowing PyCaret to utilize all available CPUs.

Use GPU: False - GPU acceleration is not utilized for model training.

Log Experiment: False - Logging of the experiment is turned off.

Experiment Name: clf-default-name - The default name for the classification experiment is 'clf-default-name'.

USI: 627f - A unique identifier for the experiment setup.

[5]:
```
automl_type = type(automl_setup)
automl_type
```

[5]:
```
pycaret.classification.oop.ClassificationExperiment
```

0.3 Compare models and select the best one

[7]:
```
best_model = compare_models()
```

```
Initiated  . . . . . . . . . . . . . . . . . .              15:30:09
Status     . . . . . . . . . . . . . . . . . .   Selecting Estimator
Estimator  . . . . . . . . . . . . . . . . . .   Logistic Regression
```

```
<pandas.io.formats.style.Styler at 0x7f23b7ce2990>
```

```
<IPython.core.display.HTML object>
```

This output summarizes the performance metrics of various machine learning models trained on the prepared churn dataset, including accuracy, area under the curve (AUC), recall, precision, F1 score, Kappa, Matthews correlation coefficient (MCC), and training time in seconds.

The best performing model based on accuracy:

The Linear Discriminant Analysis(LDA) achieved the highest accuracy of 80.07% followed closely by Ridge Classifier accuracy of 79.97%. Logistic Regression (LR) is another high performer with an accuracy of 79.87%.

Interpreting the results

Accuracy: Indicates the proportion of correctly classified instances out of the total instances.

AUC: Represents the area under the receiver operating characteristic (ROC) curve, which measures the model's ability to distinguish between classes.

Recall: Denotes the proportion of actual positive cases that were correctly identified by the model.

Precision: Indicates the proportion of positive identifications that were actually correct.

F1 Score: Harmonic mean of precision and recall, providing a balance between the two metrics.

Kappa: Measures the agreement between predicted and actual classifications, considering the possibility of the agreement occurring by chance.

MCC (Matthews Correlation Coefficient): Another measure of the quality of binary classifications, considering both false positives and false negatives.

Training Time (TT): Indicates the time taken by each model to train on the dataset.

[8]:
```
best_model_info = best_model
best_model_info
```

[8]:
```
LinearDiscriminantAnalysis(covariance_estimator=None, n_components=None,
                           priors=None, shrinkage=None, solver='svd',
                           store_covariance=False, tol=0.0001)
```

0.4 select specific rows

[9]:
```
selected_rows = df.iloc[:15]
selected_rows
```

[9]:
```
    tenure  MonthlyCharges  TotalCharges  Churn  MonthlyCharges_log  \
0        1           29.85         29.85      0            3.396185
1       34           56.95       1889.50      0            4.042174
2        2           53.85        108.15      1            3.986202
3       45           42.30       1840.75      0            3.744787
4        2           70.70        151.65      1            4.258446
5        8           99.65        820.50      1            4.601664
6       22           89.10       1949.40      0            4.489759
7       10           29.75        301.90      0            3.392829
8       28          104.80       3046.05      1            4.652054
9       62           56.15       3487.95      0            4.028027
10      13           49.95        587.45      0            3.911023
11      16           18.95        326.80      0            2.941804
12      58          100.35       5681.10      0            4.608664
13      49          103.70       5036.30      1            4.641502
14      25          105.50       2686.05      0            4.658711

    TotalCharges_Tenure_Ratio  MonthlyCharges_to_TotalCharges_Ratio  \
0                   29.850000                              1.000000
1                   55.573529                              0.030140
2                   54.075000                              0.497920
```

3	40.905556	0.022980
4	75.825000	0.466205
5	102.562500	0.121450
6	88.609091	0.045706
7	30.190000	0.098543
8	108.787500	0.034405
9	56.257258	0.016098
10	45.188462	0.085029
11	20.425000	0.057987
12	97.950000	0.017664
13	102.781633	0.020591
14	107.442000	0.039277

	Bank transfer (automatic)	Credit card (automatic)	Electronic check	\
0	0	0	0	
1	0	0	1	
2	0	0	1	
3	1	0	1	
4	0	0	0	
5	0	0	0	
6	0	1	1	
7	0	0	1	
8	0	0	0	
9	1	0	1	
10	0	0	1	
11	0	1	1	
12	0	1	1	
13	1	0	1	
14	0	0	0	

	Mailed check	Month-to-month	One year	Two year
0	0	0	0	0
1	1	1	1	0
2	1	0	0	0
3	0	1	1	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0
7	1	0	0	0
8	0	0	0	0
9	0	1	1	0
10	1	0	0	0
11	0	1	0	1
12	0	1	1	0
13	0	0	0	0
14	0	0	0	0

0.5 use best model to predict target variable

[10]:
```
predict_model(best_model, selected_rows)
```

```
<pandas.io.formats.style.Styler at 0x7f23bec9e050>
```

[10]:
```
    tenure  MonthlyCharges  TotalCharges  MonthlyCharges_log  \
0        1       29.850000     29.850000            3.396185
1       34       56.950001   1889.500000            4.042174
2        2       53.849998    108.150002            3.986202
3       45       42.299999   1840.750000            3.744787
4        2       70.699997    151.649994            4.258446
5        8       99.650002    820.500000            4.601664
6       22       89.099998   1949.400024            4.489759
7       10       29.750000    301.899994            3.392829
8       28      104.800003   3046.050049            4.652054
9       62       56.150002   3487.949951            4.028027
10      13       49.950001    587.450012            3.911022
11      16       18.950001    326.799988            2.941804
12      58      100.349998   5681.100098            4.608664
13      49      103.699997   5036.299805            4.641502
14      25      105.500000   2686.050049            4.658711

    TotalCharges_Tenure_Ratio  MonthlyCharges_to_TotalCharges_Ratio  \
0                   29.850000                              1.000000
1                   55.573528                              0.030140
2                   54.075001                              0.497920
3                   40.905556                              0.022980
4                   75.824997                              0.466205
5                  102.562500                              0.121450
6                   88.609093                              0.045706
7                   30.190001                              0.098543
8                  108.787498                              0.034405
9                   56.257259                              0.016098
10                  45.188461                              0.085029
11                  20.424999                              0.057987
12                  97.949997                              0.017664
13                 102.781631                              0.020591
14                 107.442001                              0.039277

    Bank transfer (automatic)  Credit card (automatic)  Electronic check  \
0                           0                        0                 0
1                           0                        0                 1
2                           0                        0                 1
3                           1                        0                 1
4                           0                        0                 0
5                           0                        0                 0
6                           0                        1                 1
```

```
7                            0                        0                  1
8                            0                        0                  0
9                            1                        0                  1
10                           0                        0                  1
11                           0                        1                  1
12                           0                        1                  1
13                           1                        0                  1
14                           0                        0                  0

    Mailed check  Month-to-month  One year  Two year  Churn  prediction_label  \
0              0               0         0         0      0                 1
1              1               1         1         0      0                 0
2              1               0         0         0      1                 0
3              0               1         1         0      0                 0
4              0               0         0         0      1                 1
5              0               0         0         0      1                 1
6              0               0         0         0      0                 0
7              1               0         0         0      0                 0
8              0               0         0         0      1                 1
9              0               1         1         0      0                 0
10             1               0         0         0      0                 0
11             0               1         0         1      0                 0
12             0               1         1         0      0                 0
13             0               0         0         0      1                 0
14             0               0         0         0      0                 1

    prediction_score
0             0.6476
1             0.9583
2             0.6443
3             0.9650
4             0.7753
5             0.8431
6             0.5757
7             0.9234
8             0.6990
9             0.9649
10            0.8820
11            0.9748
12            0.9185
13            0.7105
14            0.7297
```

Model Performance Metrics

```
Model - LDA
Accuracy: 73.33%
AUC: 86.00%
```

```
Recall: 60.00%
Precision: 60.00%
F1 Score: 60.00%
Kappa: 40.00%
MCC: 40.00%
```

These metrics evaluate the performance of the LDA model on the selected data. An accuracy of 73.33% suggests that 73.33% of the predictions made by the model are correct. A high AUC of 86.00% indicates that the model has a good ability to distinguish between the positive and negative classes. The recall, precision, and F1 score of 60.00% indicate that the model correctly identifies 60.00% of the positive cases, and when it predicts positive, it is correct 60.00% of the time. The Kappa and MCC scores are 40.00%, indicating moderate agreement and correlation between predicted and actual classifications, respectively.

0.6 Determining wrong predictions

[11]:
```
predicted_rows = predict_model(best_model, selected_rows)
wrong_predictions = (predicted_rows['Churn'] !=␣
↪predicted_rows['prediction_label']).sum()

print("Number of times the model was wrong:", wrong_predictions)
```

```
<pandas.io.formats.style.Styler at 0x7f2400481f10>
```

```
Number of times the model was wrong: 4
```

Out of the total predictions made, the model was incorrect in predicting the churn status of 4 customers.

0.7 Save to disk

[12]:
```
save_model(best_model, 'LDA')
```

```
Transformation Pipeline and Model Successfully Saved
```

[12]:
```
(Pipeline(memory=Memory(location=None),
          steps=[('numerical_imputer',
                  TransformerWrapper(exclude=None,
                                     include=['tenure', 'MonthlyCharges',
                                              'TotalCharges',
                                              'MonthlyCharges_log',
                                              'TotalCharges_Tenure_Ratio',
'MonthlyCharges_to_TotalCharges_Ratio',
                                              'Bank transfer (automatic)',
                                              'Credit card (automatic)',
                                              'Electronic check', 'Mailed
check',
                                              'Month-to-month', 'One year',
                                              'Two y…
strategy='most_frequent',
```

```
verbose='deprecated'))),
                 ('clean_column_names',
                  TransformerWrapper(exclude=None, include=None,
transformer=CleanColumnNames(match='[\\]\\[\\,\\{\\}\\"\\:]+'))),
                 ('trained_model',
                  LinearDiscriminantAnalysis(covariance_estimator=None,
                                             n_components=None, priors=None,
                                             shrinkage=None, solver='svd',
                                             store_covariance=False,
                                             tol=0.0001))],
          verbose=False),
 'LDA.pkl')
```

0.8 Use pickle serialization to save the best_model

[13]:
```
with open('LDA_model.pk', 'wb') as f:
    pickle.dump(best_model, f)
```

0.9 Load the saved model using pickle deserialization

[14]:
```
with open('LDA_model.pk', 'rb') as f:
    loaded_model = pickle.load(f)
```

0.10 Create new_data

[15]:
```
new_data = selected_rows.copy()
new_data.drop('Churn', axis=1, inplace=True)
new_data.to_csv('newest_churn_data.csv', index=False)
```

0.11 predict churn for the loaded data

[16]:
```
loaded_model.predict(new_data)
```

[16]:
```
array([1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1], dtype=int8)
```

1: Predicted churn (the model predicts that the customer will churn). 0: Predicted no churn (the model predicts that the customer will not churn).

So, interpreting the predictions

```
The first customer is predicted to churn.
The second customer is predicted not to churn.
The third customer is predicted not to churn.
And so on, for each customer in the new data.
```

[17]:
```
loaded_ridge = load_model('LDA')
predict_model(loaded_ridge, new_data)
```

```
Transformation Pipeline and Model Successfully Loaded
```

<IPython.core.display.HTML object>

[17]:

```
    tenure  MonthlyCharges  TotalCharges  MonthlyCharges_log  \
0        1       29.850000     29.850000            3.396185
1       34       56.950001   1889.500000            4.042174
2        2       53.849998    108.150002            3.986202
3       45       42.299999   1840.750000            3.744787
4        2       70.699997    151.649994            4.258446
5        8       99.650002    820.500000            4.601664
6       22       89.099998   1949.400024            4.489759
7       10       29.750000    301.899994            3.392829
8       28      104.800003   3046.050049            4.652054
9       62       56.150002   3487.949951            4.028027
10      13       49.950001    587.450012            3.911022
11      16       18.950001    326.799988            2.941804
12      58      100.349998   5681.100098            4.608664
13      49      103.699997   5036.299805            4.641502
14      25      105.500000   2686.050049            4.658711

    TotalCharges_Tenure_Ratio  MonthlyCharges_to_TotalCharges_Ratio  \
0                   29.850000                              1.000000
1                   55.573528                              0.030140
2                   54.075001                              0.497920
3                   40.905556                              0.022980
4                   75.824997                              0.466205
5                  102.562500                              0.121450
6                   88.609093                              0.045706
7                   30.190001                              0.098543
8                  108.787498                              0.034405
9                   56.257259                              0.016098
10                  45.188461                              0.085029
11                  20.424999                              0.057987
12                  97.949997                              0.017664
13                 102.781631                              0.020591
14                 107.442001                              0.039277

    Bank transfer (automatic)  Credit card (automatic)  Electronic check  \
0                           0                        0                 0
1                           0                        0                 1
2                           0                        0                 1
3                           1                        0                 1
4                           0                        0                 0
5                           0                        0                 0
6                           0                        1                 1
7                           0                        0                 1
8                           0                        0                 0
9                           1                        0                 1
```

```
10                            0                        0                 1
11                            0                        1                 1
12                            0                        1                 1
13                            1                        0                 1
14                            0                        0                 0

    Mailed check  Month-to-month  One year  Two year  prediction_label  \
0              0               0         0         0                 1
1              1               1         1         0                 0
2              1               0         0         0                 0
3              0               1         1         0                 0
4              0               0         0         0                 1
5              0               0         0         0                 1
6              0               0         0         0                 0
7              1               0         0         0                 0
8              0               0         0         0                 1
9              0               1         1         0                 0
10             1               0         0         0                 0
11             0               1         0         1                 0
12             0               1         1         0                 0
13             0               0         0         0                 0
14             0               0         0         0                 1

    prediction_score
0             0.6476
1             0.9583
2             0.6443
3             0.9650
4             0.7753
5             0.8431
6             0.5757
7             0.9234
8             0.6990
9             0.9649
10            0.8820
11            0.9748
12            0.9185
13            0.7105
14            0.7297
```

0.12 Python module to predict churn

[18]:

```
Code('predict_churn.py')
```

[18]:

```
import pandas as pd
from pycaret.classification import predict_model, load_model

def predict_churn():
```

```
    df = pd.read_csv('newest_churn_data.csv')
    model = load_model('LDA')
    predictions = predict_model(model, df)
    predictions.rename({'prediction_label': 'Churn_prediction'}, axis=1,␣
 ↪inplace=True)
    predictions['Churn_prediction'].replace({1: 'Churn', 0: 'No Churn'},␣
 ↪inplace=True)
    return predictions['Churn_prediction']

# Call the function and print the predictions
print(predict_churn())
```

[19]:
```
%run predict_churn.py
```

```
Transformation Pipeline and Model Successfully Loaded

<IPython.core.display.HTML object>

0         Churn
1      No Churn
2      No Churn
3      No Churn
4         Churn
5         Churn
6      No Churn
7      No Churn
8         Churn
9      No Churn
10     No Churn
11     No Churn
12     No Churn
13     No Churn
14        Churn
Name: Churn_prediction, dtype: object
```

The output indicates the churn predictions for each customer in the dataset. Each entry in the output corresponds to a customer, and it shows whether the model predicts that the customer will churn or not churn.

0.13 Comparison with actual churn status

Prediction for index 0: Churn Actual churn status - Churn

Prediction for index 1: No Churn Actual churn status - No Churn

Prediction for index 2: No Churn Actual churn status - No Churn

Prediction for index 3: No Churn Actual churn status - No Churn

Prediction for index 4: Churn Actual churn status - Churn

Prediction for index 5: Churn Actual churn status - Churn

Prediction for index 6: No Churn Actual churn status - No Churn

Prediction for index 7: No Churn Actual churn status - No Churn

Prediction for index 8: Churn Actual churn status - Churn

Prediction for index 9: No Churn Actual churn status - No Churn

Prediction for index 10: No Churn Actual churn status - No Churn

Prediction for index 11: No Churn Actual churn status - No Churn

Prediction for index 12: No Churn Actual churn status - No Churn

Prediction for index 13: No Churn Actual churn status - No Churn

Prediction for index 14: Churn Actual churn status - Churn

It can be seen that the predictions align with the actual churn status of the dataset

0.14 Summary

Necessary libraries and functions were imported for our task, which involves building a churn prediction model using PyCaret, a Python library for automating machine learning workflows.

We successfully achieved the following,

Loaded and prepared the churn data.

Set up an auto ML environment and compared classification models.

Selected the best-performing model which was LDA

Predicted the churn status for 15 specific rows of data using the selected model.

Saved the best-performing model to a file using PyCaret's save_model function.

Serialized and deserialized the model using pickle.

Predicted the churn status for new data using both the loaded model and PyCaret's load_model function