Classification

November 6, 2022

1 Import software libraries

[1]:
```
# Import required libraries.
import sys                                                   # Read system␣
 ↪parameters.
import numpy as np                                           # Work with␣
 ↪multi-dimensional arrays.
import pandas as pd                                          # Manipulate and␣
 ↪analyze data.
import matplotlib                                            # Create and format␣
 ↪charts.
import matplotlib.pyplot as plt
import seaborn as sns                                        # Make charting␣
 ↪easier.
import sklearn                                               # Train and␣
 ↪evaluate machine learning models.
from sklearn.model_selection import train_test_split, \
                                    learning_curve, \
                                    cross_val_score
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, \
                            confusion_matrix, \
                            classification_report, \
                            scorer, \
                            f1_score, \
                            recall_score, \
                            precision_score, \
                            roc_auc_score, \
                            plot_roc_curve, \
                            plot_precision_recall_curve, \
                            plot_confusion_matrix
from sklearn.dummy import DummyClassifier
import xgboost                                               # Build gradient␣
 ↪boosting models.
```

```
from xgboost import XGBClassifier
import pickle                                                    # Save Python␣
 ↪objects as binary files.
from collections import Counter
import warnings                                                  # Suppress warnings.
warnings.filterwarnings('ignore')

# Ensure results are reproducible.
np.random.seed(1)

# Summarize software libraries used.
print('Libraries used in this project:')
print('- Python {}'.format(sys.version))
print('- NumPy {}'.format(np.__version__))
print('- pandas {}'.format(pd.__version__))
print('- Matplotlib {}'.format(matplotlib.__version__))
print('- Seaborn {}'.format(sns.__version__))
print('- scikit-learn {}'.format(sklearn.__version__))
print('- XGBoost {}'.format(xgboost.__version__))
```

```
Libraries used in this project:
- Python 3.7.6 | packaged by conda-forge | (default, Mar 23 2020, 23:03:20)
[GCC 7.3.0]
- NumPy 1.19.2
- pandas 1.1.3
- Matplotlib 3.3.2
- Seaborn 0.11.0
- scikit-learn 0.23.2
- XGBoost 1.3.3
```

2 Read and examine the data

[2]:
```
# Read the data.
df = pd.read_pickle('data/customer_data.pickle')


# Preview the first five rows of the data.
df.head()
```

[2]:
```
        frequency  recency  tenure  monetary_value  number_unique_items  \
u12747        6.0    367.0   369.0           39.19                    3
u12748       41.0    365.0   369.0           12.01                    9
u12749        2.0    127.0   130.0           22.28                    2
u1282         0.0      0.0   326.0            0.00                    1
u12822        0.0      0.0    87.0            0.00                    1
```

```
        churned
u12747     True
u12748    False
u12749     True
u1282     False
u12822     True
```

[3]:
```
# Check the structure of the data.
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 2130 entries, u12747 to u18283
Data columns (total 6 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   frequency           2130 non-null   float64
 1   recency             2130 non-null   float64
 2   tenure              2130 non-null   float64
 3   monetary_value      2130 non-null   float64
 4   number_unique_items 2130 non-null   int64
 5   churned             2130 non-null   bool
dtypes: bool(1), float64(4), int64(1)
memory usage: 101.9+ KB
```

3 Prepare the data

[4]:
```
# Define the target variable and get the count of each value in the variable.
df.churned.value_counts()
```

[4]:
```
False    1380
True      750
Name: churned, dtype: int64
```

[5]:
```
# Split the data into target and features.
target_data = df.churned
features = df.drop(['churned'], axis=1)
```

[6]:
```
# Split the dataset into separate training and testing sets.
X_train, X_test, y_train, y_test = train_test_split(features, target_data,␣
↪test_size = 0.3)
```

```
# Get the shape of both the training dataset and the test dataset.
print('Training data features: ', X_train.shape)
print('Test data features: ', y_train.shape)
```

```
Training data features:  (1491, 5)
Test data features:  (1491,)
```

[7]:
```
# Use the Counter library to get the count of each value in the target variable␣
↪(test data).
print(Counter(y_test))
```

```
Counter({False: 414, True: 225})
```

4 Train a logistic regression model

[8]:
```
# Normalize the training data.
norm = MinMaxScaler().fit(X_train)
X_train_norm = norm.transform(X_train)
X_train_norm
```

[8]:
```
array([[0.02439024, 0.30727763, 0.71774194, 0.14221084, 0.125     ],
       [0.26829268, 0.77628032, 0.9811828 , 0.15911225, 0.375     ],
       [0.02439024, 0.69272237, 0.78763441, 0.05633803, 0.        ],
       …,
       [0.02439024, 0.37466307, 0.60215054, 0.07042254, 0.25      ],
       [0.07317073, 0.2884097 , 0.7983871 , 0.13401622, 0.25      ],
       [0.04878049, 0.18867925, 0.23655914, 0.06338028, 0.125     ]])
```

[9]:
```
# Create a LogisticRegression() model and fit it on the scaled training data.
logreg = LogisticRegression()
logreg.fit(X_train_norm, y_train)
```

[9]:
```
LogisticRegression()
```

[11]:
```
# Make predictions on the test data.
y_logreg_pred = logreg.predict(X_test)


# Get a count of each prediction value.
print(Counter(y_logreg_pred))
```

```
Counter({True: 636, False: 3})
```

5 Perform a quick evaluation of the logistic regression model

[12]:
```
# Obtain the accuracy of the model's predictions.
accuracy_score(y_test, y_logreg_pred)
```

[12]: 0.3536776212832551

[13]:
```
# Use the classification_report() function to get a table of additional metric␣
↪scores.
print(classification_report(y_test, y_logreg_pred))
```

```
              precision    recall  f1-score   support

       False       0.67      0.00      0.01       414
        True       0.35      1.00      0.52       225

    accuracy                           0.35       639
   macro avg       0.51      0.50      0.26       639
weighted avg       0.56      0.35      0.19       639
```

6 Train a random forest model

[14]:
```
# Create a RandomForestClassifier() model and fit it on the scaled training␣
↪data.
rf = RandomForestClassifier()
rf.fit(X_train_norm, y_train)
```

[14]: RandomForestClassifier()

[17]:
```
# Make predictions on the test data.
y_rf_pred = rf.predict(X_test)


# Get a count of each prediction value.
print(Counter(y_rf_pred))
```

```
Counter({False: 639})
```

7 Perform a quick evaluation of the logistic regression model

[18]:
```python
# Obtain the accuracy of the model's predictions.
accuracy_score(y_test, y_rf_pred)
```

[18]: 0.647887323943662

[19]:
```python
# Use the classification_report() function to get a table of additional metric␣
↪scores.
print(classification_report(y_test, y_rf_pred))
```

```
              precision    recall  f1-score   support

       False       0.65      1.00      0.79       414
        True       0.00      0.00      0.00       225

    accuracy                           0.65       639
   macro avg       0.32      0.50      0.39       639
weighted avg       0.42      0.65      0.51       639
```

8 Compare evaluation metrics for each model

[20]:
```python
# List will hold model objects.

models = []

# DummyClassifier() used as a baseline algorithm.

models.append(('Dummy Classifier', DummyClassifier(strategy = 'stratified')))

# Logistic Regression model.

models.append(('Logistic Regression', LogisticRegression()))

# Random Forest model.

models.append(('Random Forest', RandomForestClassifier()))

# XGBoost model.

models.append(('XGBoost', XGBClassifier(eval_metric = 'logloss', n_jobs = 1)))
```

[30]:
```python
# List will hold dictionaries of model scores.
```

```
scoring_df = []

# Train each model in the list and output multiple scores for each model.

for name, model in models:
    if name in ['Logistic Regression']:
        X_train_1 = X_train_norm
    else:
        X_train_1 = X_train

    model.fit(X_train_1, y_train)

    y_pred = model.predict(X_test)

    # Calcualte the evaluation metrics for the model.

    accuracy = accuracy_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    auc = roc_auc_score(y_test, y_pred)

    scoring_dict = {'Model': name,
                    'Accuracy': round(accuracy, 4),
                    'F1 Score': round(f1, 4),
                    'Precision' : round(precision, 4),
                    'Recall' : round(recall, 4),
                    'AUC' : round(auc ,4),
                   }

    scoring_df.append(scoring_dict)
```

[33]:
```
# Create a DataFrame from scoring_df.
scoring_df = pd.DataFrame(scoring_df)
scoring_df

# Sort the DataFrame by accuracy score (descending), then print it.
scoring_df.sort_values(by='Accuracy', ascending=False)
```

[33]:

	Model	Accuracy	F1 Score	Precision	Recall	AUC
3	XGBoost	0.5712	0.2789	0.3419	0.2356	0.4946
2	Random Forest	0.5649	0.2760	0.3333	0.2356	0.4898
0	Dummy Classifier	0.5180	0.3246	0.3203	0.3289	0.4748
1	Logistic Regression	0.3537	0.5203	0.3522	0.9956	0.5002

9 Begin evaluating the best model

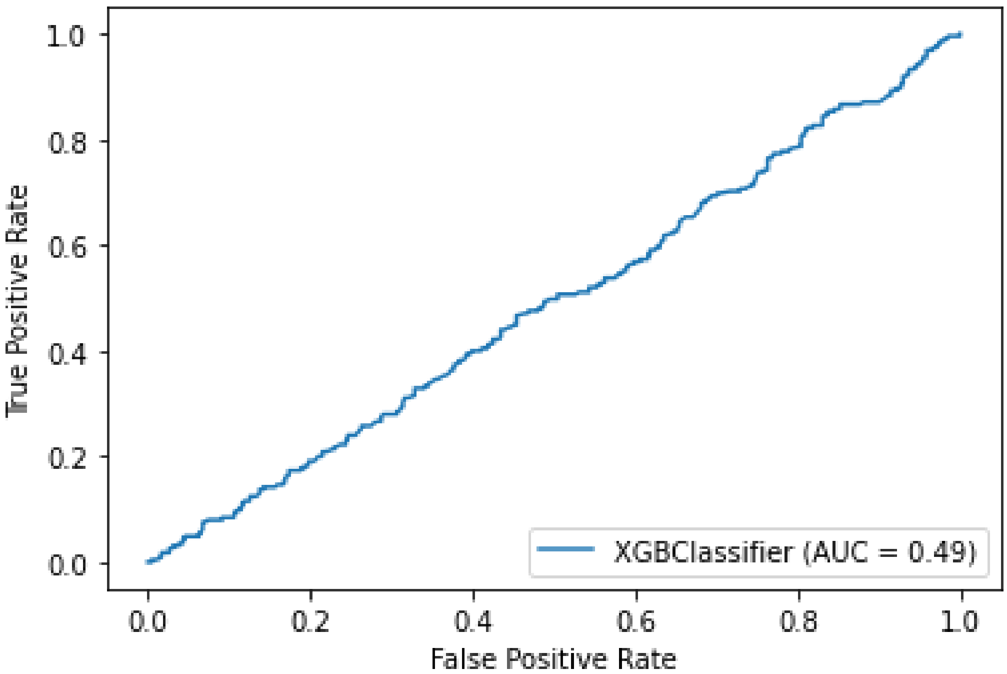
[40]:
```python
# Retrain the model with the highest accuracy score.
xgb = XGBClassifier(eval_metric='logloss', n_jobs =1)
xgb.fit(X_train, y_train)
```

[40]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, eval_metric='logloss',
              gamma=0, gpu_id=-1, importance_type='gain',
              interaction_constraints='', learning_rate=0.300000012,
              max_delta_step=0, max_depth=6, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimators=100, n_jobs=1,
              num_parallel_tree=1, random_state=0, reg_alpha=0, reg_lambda=1,
              scale_pos_weight=1, subsample=1, tree_method='exact',
              validate_parameters=1, verbosity=None)
```

[42]:
```python
# Make predictions on the test data.
y_xgb_pred = xgb.predict(X_test)


# Get a count of each prediction value.
print(Counter(y_xgb_pred))
```

```
Counter({False: 484, True: 155})
```

[43]:
```python
# Plot a ROC curve.
plot_roc_curve(xgb, X_test, y_test)
plt.show();
```

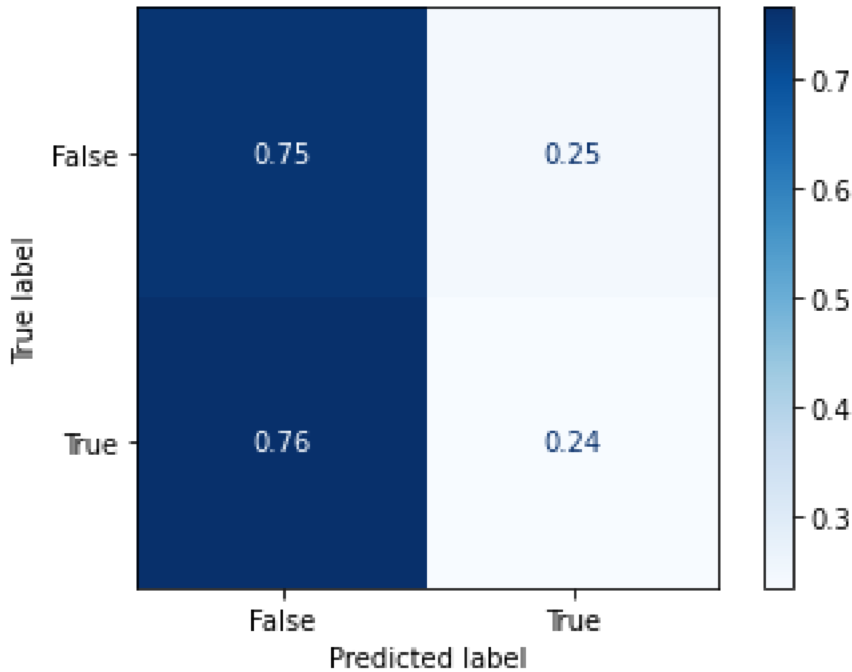


10 Generate a confusion matrix of the best model

[45]:
```
# Generate a confusion matrix.
confusion_matrix(y_test, y_xgb_pred)
```

[45]:
```
array([[312, 102],
       [172,  53]])
```

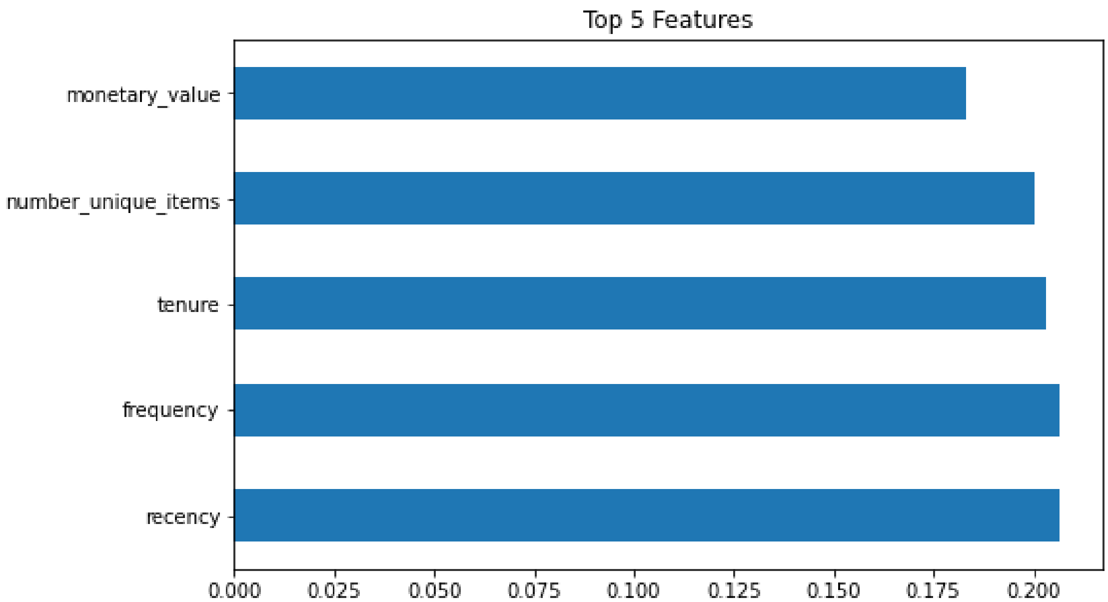
[46]:
```
# Plot the confusion matrix.
plot_confusion_matrix(xgb, X_test, y_test, cmap = plt.cm.Blues,␣
 ↪normalize='true')
plt.show();
```



11 Generate a feature importance plot for the best model

[47]:
```
# This function generates a feature importance plot on a bar chart.

def feature_importance_plot(model, X_train, n):
    """Plots feature importance. This only works for random forest and XGBoost␣
 ↪models."""

    plt.figure(figsize=(8, 5))  # Set figure size.
    feat_importances = pd.Series(model.feature_importances_,
                                 index = X_train.columns)
    feat_importances.nlargest(n).plot(kind = 'barh')
    plt.title(f'Top {n} Features')
    plt.show()
```

[48]:
```
# Plot the feature importances.
feature_importance_plot(xgb, X_train, 5)
```

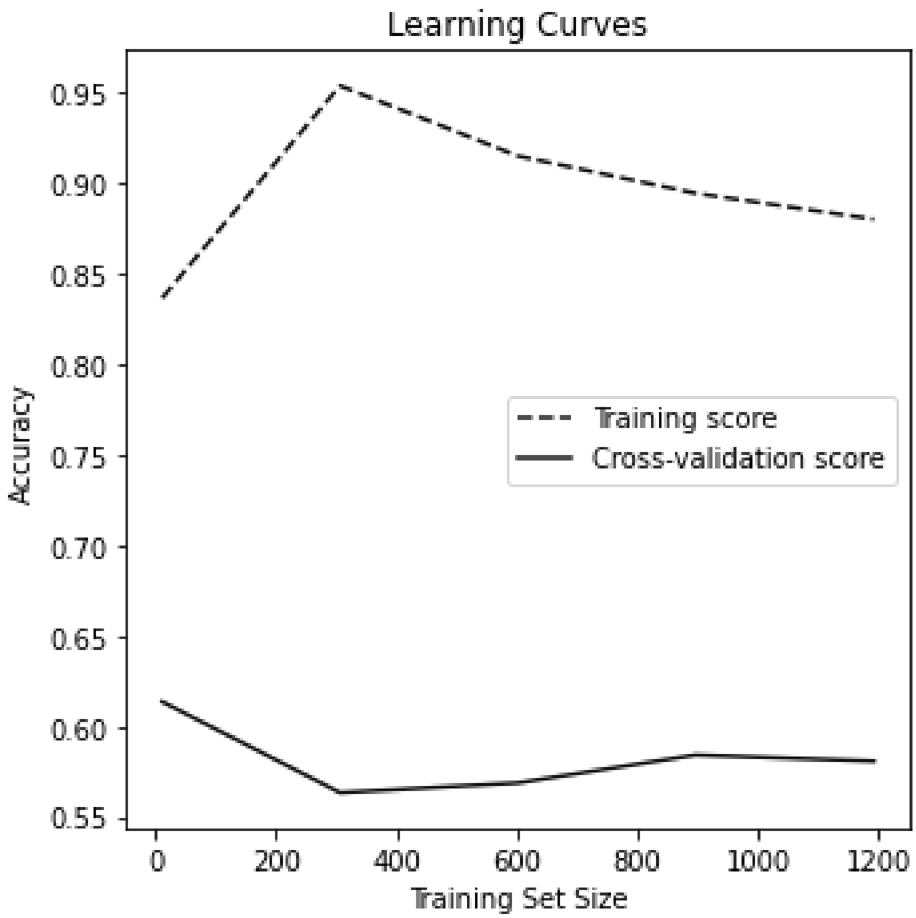


12 Plot a learning curve for the best model

[49]:
```
# This function generates and plots a learning curve.

def plot_learning_curves(model, X_train, y_train):
    """Plots learning curves for model validation."""

    plt.figure(figsize=(5, 5))  # Set figure size.
    train_sizes, train_scores, test_scores = learning_curve(model,
                                                            X_train,
                                                            y_train,
                                                            cv = 5,  # Number␣
 ↪of folds in cross-validation.
                                                            scoring =␣
 ↪'accuracy',  # Evaluation metric.
                                                            n_jobs = 1,
                                                            shuffle = True,
                                                            train_sizes = np.
 ↪linspace(0.01, 1.0, 5))  # 5 different sizes of the training set.

    # Create means and standard deviations of training set scores.

    train_mean = np.mean(train_scores, axis = 1)
    train_std = np.std(train_scores, axis = 1)
```

```
    # Create means and standard deviations of test set scores.

    test_mean = np.mean(test_scores, axis = 1)
    test_std = np.std(test_scores, axis = 1)

    # Draw lines.

    plt.plot(train_sizes, train_mean, '--', color = '#111111', label =␣
↪'Training score')
    plt.plot(train_sizes, test_mean, color = '#111111', label =␣
↪'Cross-validation score')

    # Create plot.

    plt.title('Learning Curves')
    plt.xlabel('Training Set Size'), plt.ylabel('Accuracy'), plt.legend(loc =␣
↪'best')
    plt.tight_layout()

    plt.show()
```

[50]:
```
# Call the function to plot learning curves for the best model.
plot_learning_curves(xgb, X_train, y_train)
```



13 Save the best model

[52]:
```
# Save the best model as a pickle file named best_classification_model.pickle.
pickle.dump(xgb, open('best_classification_model.pickle', 'wb'))
```

[ ]: