Week3_Assignment

February 4, 2024

0.1 Import Libraries

[4]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import ConfusionMatrixDisplay
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix
```

[5]:
```
df = pd.read_csv("prepped_churn_data.csv")

df.index = range(1, len(df) + 1)

df.insert(0, "customerID", df.index)

df.head(5)
```

[5]:
```
   customerID  tenure  PhoneService        Contract  \
1           1       1             0  Month-to-month
2           2      34             1        One year
3           3       2             1  Month-to-month
4           4      45             0        One year
5           5       2             1  Month-to-month

               PaymentMethod  MonthlyCharges  TotalCharges  Churn  \
1           Electronic check           29.85         29.85      0
2               Mailed check           56.95       1889.50      0
3               Mailed check           53.85        108.15      1
4  Bank transfer (automatic)           42.30       1840.75      0
5           Electronic check           70.70        151.65      1

   MonthlyCharges_to_TotalCharges_Ratio
1                              1.000000
2                              0.030140
3                              0.497920
4                              0.022980
```

```
5                             0.466205
```

0.2 Prepping the data further

[6]:
```
payment_method_dummies = pd.get_dummies(df['PaymentMethod'])
contract_dummies = pd.get_dummies(df['Contract'])

# Combine the dummy variables with the original DataFrame
df = pd.concat([df, payment_method_dummies, contract_dummies], axis=1)
df.head()
```

[6]:
```
   customerID  tenure  PhoneService        Contract  \
1           1       1             0  Month-to-month
2           2      34             1        One year
3           3       2             1  Month-to-month
4           4      45             0        One year
5           5       2             1  Month-to-month

              PaymentMethod  MonthlyCharges  TotalCharges  Churn  \
1          Electronic check           29.85         29.85      0
2              Mailed check           56.95       1889.50      0
3              Mailed check           53.85        108.15      1
4 Bank transfer (automatic)           42.30       1840.75      0
5          Electronic check           70.70        151.65      1

   MonthlyCharges_to_TotalCharges_Ratio  Bank transfer (automatic)  \
1                              1.000000                      False
2                              0.030140                      False
3                              0.497920                      False
4                              0.022980                       True
5                              0.466205                      False

   Credit card (automatic)  Electronic check  Mailed check  Month-to-month  \
1                    False              True         False            True
2                    False             False          True           False
3                    False             False          True            True
4                    False             False         False           False
5                    False              True         False            True

   One year  Two year
1     False     False
2      True     False
3     False     False
4      True     False
5     False     False
```

0.3 Convert dummy variables to numeric

[7]:
```
categorical_columns = ['Electronic check', 'Mailed check', 'Bank transfer␣
↪(automatic)', 'Credit card (automatic)', 'Month-to-month', 'One year', 'Two␣
↪year']

for column in categorical_columns:
    df[column] = pd.factorize(df[column])[0]

df.sample(5)
```

[7]:
```
      customerID  tenure  PhoneService        Contract  \
862          862       3             1  Month-to-month
5220        5220       9             1  Month-to-month
5193        5193      70             1        One year
2318        2318      49             1        Two year
2767        2767      13             1  Month-to-month

                 PaymentMethod  MonthlyCharges  TotalCharges  Churn  \
862           Electronic check           95.10        307.40      1
5220              Mailed check           44.40        348.15      0
5193          Electronic check          106.50       7397.00      0
2318   Credit card (automatic)           20.45       1024.65      0
2767              Mailed check           63.15        816.80      0

      MonthlyCharges_to_TotalCharges_Ratio  Bank transfer (automatic)  \
862                               0.309369                          0
5220                              0.127531                          0
5193                              0.014398                          0
2318                              0.019958                          0
2767                              0.077314                          0

      Credit card (automatic)  Electronic check  Mailed check  Month-to-month  \
862                         0                 0             0               0
5220                        0                 1             1               0
5193                        0                 0             0               1
2318                        1                 1             0               1
2767                        0                 1             1               0

      One year  Two year
862          0         0
5220         0         0
5193         1         0
2318         0         1
2767         0         0
```

[8]:
```
# Drop the original categorical columns
df.drop(['PaymentMethod', 'Contract'], axis=1, inplace=True)
df.head(5)
```

[8]:
```
   customerID  tenure  PhoneService  MonthlyCharges  TotalCharges  Churn  \
1           1       1             0           29.85         29.85      0
2           2      34             1           56.95       1889.50      0
3           3       2             1           53.85        108.15      1
4           4      45             0           42.30       1840.75      0
5           5       2             1           70.70        151.65      1

   MonthlyCharges_to_TotalCharges_Ratio  Bank transfer (automatic)  \
1                              1.000000                          0
2                              0.030140                          0
3                              0.497920                          0
4                              0.022980                          1
5                              0.466205                          0

   Credit card (automatic)  Electronic check  Mailed check  Month-to-month  \
1                        0                 0             0               0
2                        0                 1             1               1
3                        0                 1             1               0
4                        0                 1             0               1
5                        0                 0             0               0

   One year  Two year
1         0         0
2         1         0
3         0         0
4         1         0
5         0         0
```

[9]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7032 entries, 1 to 7032
Data columns (total 14 columns):
 #   Column                                Non-Null Count  Dtype
---  ------                                --------------  -----
 0   customerID                            7032 non-null   int64
 1   tenure                                7032 non-null   int64
 2   PhoneService                          7032 non-null   int64
 3   MonthlyCharges                        7032 non-null   float64
 4   TotalCharges                          7032 non-null   float64
 5   Churn                                 7032 non-null   int64
 6   MonthlyCharges_to_TotalCharges_Ratio  7032 non-null   float64
 7   Bank transfer (automatic)             7032 non-null   int64
 8   Credit card (automatic)               7032 non-null   int64
```

```
 9   Electronic check                           7032 non-null   int64
 10  Mailed check                               7032 non-null   int64
 11  Month-to-month                             7032 non-null   int64
 12  One year                                   7032 non-null   int64
 13  Two year                                   7032 non-null   int64
dtypes: float64(3), int64(11)
memory usage: 769.3 KB
```

[10]:
```
df.isna().sum()
```

[10]:
```
customerID                               0
tenure                                   0
PhoneService                             0
MonthlyCharges                           0
TotalCharges                             0
Churn                                    0
MonthlyCharges_to_TotalCharges_Ratio     0
Bank transfer (automatic)                0
Credit card (automatic)                  0
Electronic check                         0
Mailed check                             0
Month-to-month                           0
One year                                 0
Two year                                 0
dtype: int64
```

1 Modelling

1.1 break data into features and targets

[11]:
```
features = df.drop('Churn', axis=1)
targets = df['Churn']
```

[12]:
```
features.sample()
```

[12]:
```
      customerID  tenure  PhoneService  MonthlyCharges  TotalCharges  \
2789        2789       4             1            80.1        336.15

      MonthlyCharges_to_TotalCharges_Ratio  Bank transfer (automatic)  \
2789                              0.238286                          0

      Credit card (automatic)  Electronic check  Mailed check  Month-to-month  \
2789                        0                 0             0               0

      One year  Two year
2789         0         0
```

[13]:
```
targets.head()
```

[13]:
```
1    0
2    0
3    1
4    0
5    1
Name: Churn, dtype: int64
```

[14]:
```
X = features
y = targets
```

1.2 split data into training and test sets

[15]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,␣
 ↪test_size=0.2, random_state=42)
```

Here we perform a train-test split on the dataset.

X represents the features (input variables) of the dataset.

y represents the target variable (churn).

stratify=y ensures that the class distribution of the target variable y is preserved in the train-test split. This means that the proportion of different classes in y will be the same in both the training and testing datasets. This ensures that each class is represented in both training and testing sets.

test_size=0.2 specifies the proportion of the dataset to include in the testing set. We set it to 20%, meaning that 20% of the dataset will be reserved for testing, and the remaining 80% will be used for training.

random_state=42 sets the seed for random number generation. This ensures that the split is reproducible, meaning if you run the code multiple times with the same random_state, you'll get the same split each time. It's useful for reproducibility and debugging.

After execution

```
X_train contains the features of the training dataset.

X_test contains the features of the testing dataset.

y_train contains the target variable values corresponding to the training dataset - churn

y_test contains the target variable values corresponding to the testing dataset - churn
```

[16]:
```
X_train.shape
```

[16]:
```
(5625, 13)
```

The output (5625, 13) from X_train.shape means that,

```
X_train is a NumPy array or DataFrame representing the features (input variables) of the train
```

```
The first number, 5625, represents the number of samples (rows) in the training dataset while
```

Therefore,

```
There are 5625 samples in the training dataset.

There are 13 features in the training dataset.
```

This information is crucial for understanding the dimensions of the dataset, which is essential for various operations, including training machine learning models.

[17]:
```
X_test.shape
```

[17]:
```
(1407, 13)
```

[18]:
```
y_train.shape
```

[18]:
```
(5625,)
```

[19]:
```
y_test.shape
```

[19]:
```
(1407,)
```

1.3 Fit model into training data

[20]:
```
lr_model = LogisticRegression(max_iter=5000)
lr_model.fit(X_train, y_train)

lr_model.fit(X_train, y_train)
```

[20]:
```
LogisticRegression(max_iter=5000)
```

This means that the model will use a maximum of 5000 iterations during training.

[21]:
```
df['Churn'].value_counts(normalize=True)
```

[21]:
```
Churn
0    0.734215
1    0.265785
Name: proportion, dtype: float64
```

Here we calculate the frequency (count) of each unique value in the Churn column of the DataFrame df. Setting normalize=True returns the relative frequencies (proportions) instead of raw counts. The output shows the proportion of the two unique values as,

```
Yes: 0.265785
No : 0.734215
```

This information is essential for understanding the class distribution and assessing the balance of the dataset.

[22]:
```
print(lr_model.score(X_train, y_train))
print(lr_model.score(X_test, y_test))
```

```
0.7996444444444445
0.7860696517412935
```

Here we calculate the accuracy score of the logistic regression model on the training data (X_train, y_train) and test data (X_test, y_test). The score method in scikit-learn returns the mean accuracy on the given test data and labels. It computes the accuracy of the model by comparing the predicted labels to the actual labels and then calculates the proportion of correct predictions.

The accuracy of the logistic regression model on the training data is approximately `79.96%`. This means that the model correctly predicts the churn status of about 79.96% of the customers in the training dataset.

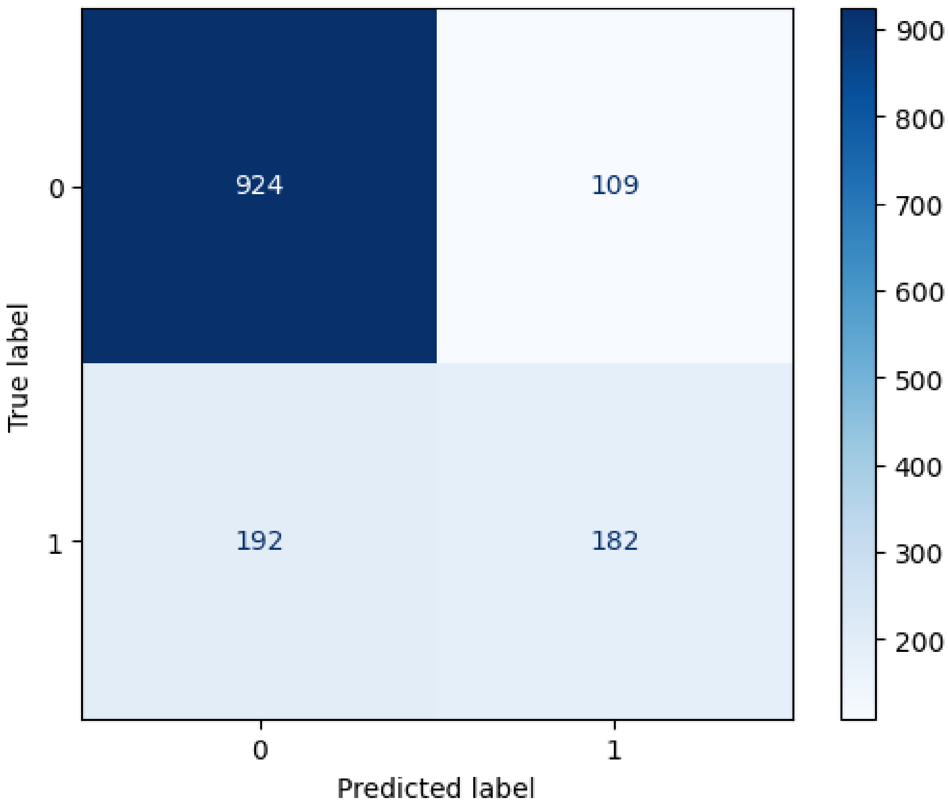
The accuracy of the logistic regression model on the test data is approximately `78.61%`. This means that the model correctly predicts the churn status of about 78.61% of the customers in the test dataset.

1.4 Plotting confusion matrix

[23]:
```
# Predict test dataset

predictions = lr_model.predict(X_test)
predictions
```

[23]:
```
array([0, 0, 0, …, 0, 0, 0])
```

[24]:
```
# construct the confusion matix
cm = confusion_matrix(y_test, predictions, labels=lr_model.classes_)

# format and display the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=lr_model.
 ↪classes_)
disp.plot(cmap=plt.cm.Blues)
plt.show()
```



The confusion matrix shows true negatives (TN, or a prediction of 924 when the true label is 0), false negatives (FN, prediction=0 true=1) of 192, true positives (TP, prediction=true=1) of 182 and false positives (FP, prediction=1 true=0) of 109. From this, we can get an idea of how the algorithm is performing and compare multiple models.

1.5 Interpretation based on Business context

True negatives (TN): The model correctly predicted 924 instances where the true label is 0 (no churn). This signifies the number of customers who were correctly identified as not churning. These are satisfied customers who were retained by the company.

False negatives (FN): The model incorrectly predicted 0 (no churn) when the true label is 1 (churn) in 192 instances. This signifies the number of customers who were incorrectly identified as not churning, leading to missed opportunities for intervention. These are customers who churned despite the model predicting otherwise.

True positives (TP): The model correctly predicted 182 instances where the true label is 1 (churn). This signifies the number of customers who were correctly identified as churning. These are customers who actually churned, and the model successfully flagged them for attention or intervention.

False positives (FP): The model incorrectly predicted 1 (churn) when the true label is 0 (no churn) in 109 instances. This signifies the number of customers who were incorrectly identified as churning, leading to unnecessary intervention or resources being allocated to customers who were not at risk of churning.

1.6 Tuning the model

[25]:
```
lr_model.predict_proba(X_test)
```

[25]:
```
array([[0.97621656, 0.02378344],
       [0.56233557, 0.43766443],
       [0.99573486, 0.00426514],
       …,
       [0.95368072, 0.04631928],
       [0.96761752, 0.03238248],
       [0.99504415, 0.00495585]])
```

[26]:
```
lr_model.predict(X_test)[:15]
```

[26]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0])
```

[27]:
```
(lr_model.predict_proba(X_test)[:10, 1] > 0.5).astype('int')
```

[27]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 1, 0])
```

The result is an array of integers indicating the binary predictions (0 or 1) for the first 10 samples in the test set, based on whether the predicted probability of the positive class is greater than 0.5.

1.7 Lowering threshold

[28]:
```
predictions_lower_thresh = (lr_model.predict_proba(X_test)[:, 1] > 0.2).
 ↪astype('int')
predictions_lower_thresh
```

[28]:
```
array([0, 1, 0, …, 0, 0, 0])
```

The result is an array of integers indicating the binary predictions (0 or 1) for all samples in the test set, based on whether the predicted probability of the positive class is greater than the lower threshold of 0.2. This allows for flexibility in adjusting the sensitivity-specificity trade-off of the model by setting different probability thresholds for making binary predictions.

1.8 Checking accuracy and TP rate

[29]:
```
print(accuracy_score(y_test, predictions_lower_thresh))
```

```
0.6901208244491827
```

The accuracy score of approximately 69.01% signifies the overall proportion of correct predictions made by the model on the test set. Specifically, it indicates that:

Out of all the samples in the test set, approximately 69.01% of them were correctly classified by the model.

The model's predictions matched the true labels for approximately 69.01% of the samples in the test set.

[30]:
```
tn, fp, fn, tp  = confusion_matrix(y_test, predictions_lower_thresh).flatten()
print(tp / (tp + fn))
```

```
0.8743315508021391
```

This means that the model correctly identified approximately 87.43% of the actual churn cases from the total number of churn cases in the dataset. Out of all the customers who actually churned, the model correctly identified 87.43% of them as churned.

A high True Positive Rate (TPR) signifies that the model is effective in identifying positive cases (churned customers), which is vital for making informed decisions and taking appropriate actions to address churn and retain valuable customers.

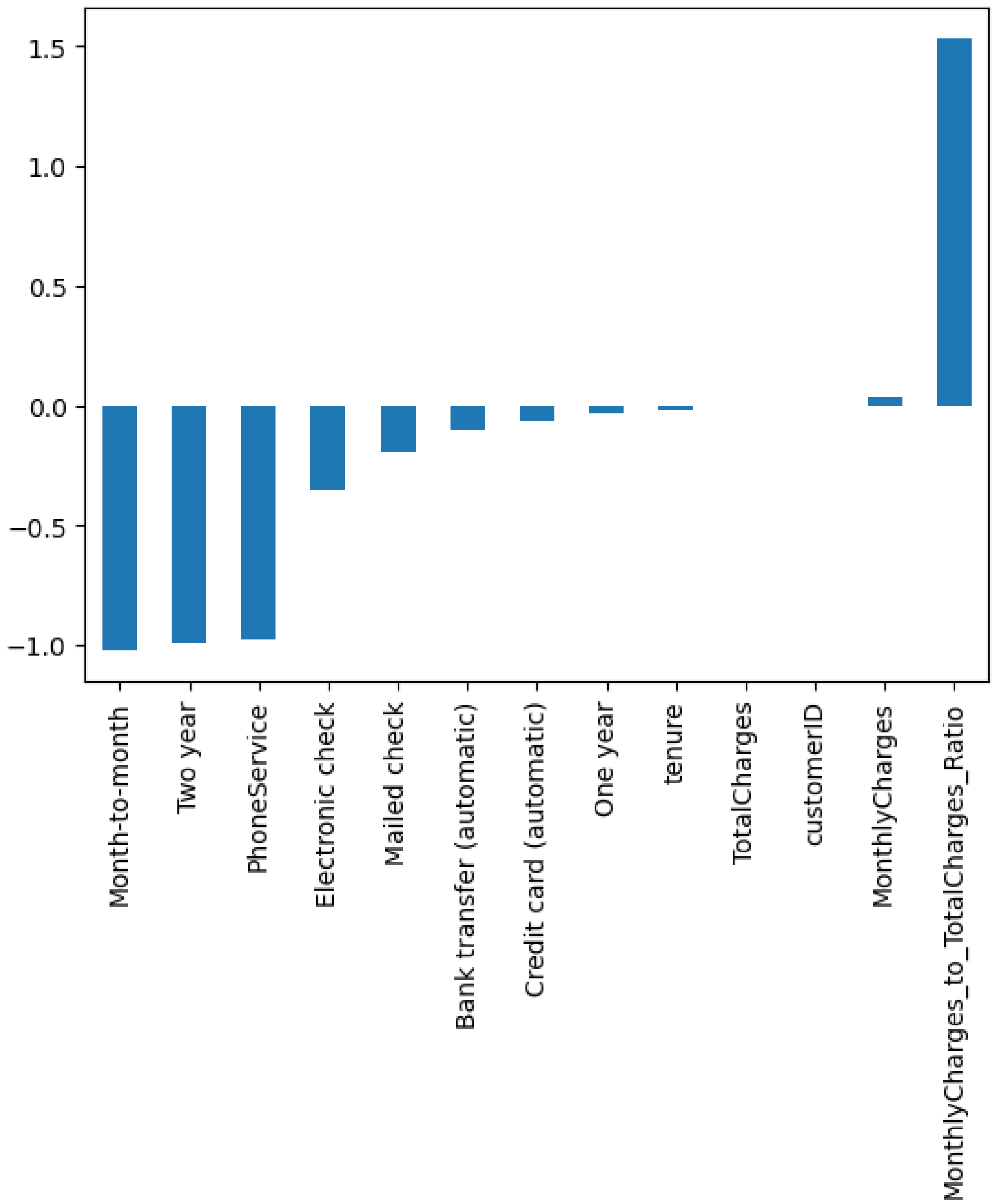
1.9 coefficients from the model

[31]:
```
lr_model.coef_
```

[31]:
```
array([[ 3.55337362e-05, -2.20531172e-02, -9.73417931e-01,
         3.37869664e-02, -1.49824975e-05,  1.53230198e+00,
        -9.98800565e-02, -6.08130784e-02, -3.55526975e-01,
        -1.94833841e-01, -1.02315128e+00, -3.00118082e-02,
        -9.93139470e-01]])
```

[32]:
```
coef_df = pd.DataFrame(data=lr_model.coef_, columns=features.columns)

coef_df.T.sort_values(by=0).plot.bar(legend=False)
```

[32]:
```
<Axes: >
```



The output lr_model.coef_ provides the coefficients (weights) assigned to each feature by the logistic regression model. These coefficients indicate the strength and direction of the relationship between each feature and the target variable (churn).

Coefficients Interpretation

Positive coefficients indicate a positive relationship with the target variable (churn), meaning that as the feature value increases, the likelihood of churn also increases.

Negative coefficients indicate a negative relationship with the target variable (churn), meaning that as the feature value increases, the likelihood of churn decreases.

Larger coefficient magnitudes (absolute values) indicate stronger associations with the target variable.

Plot Interpretation

The plot visualizes the coefficients for each feature in a bar chart format.

Features with positive coefficients are represented by bars pointing upwards, while features with negative coefficients are represented by bars pointing downwards.

The length of each bar indicates the magnitude of the coefficient, representing the strength of the association with the target variable.

Features with longer bars (either positive or negative) such as the contracts of Month-to-month abd Two year, PhoneService and the MonthlyCharges_to_TotalCharges_Ratio have a stronger influence on the model's predictions.

Those with positive coefficients - MonthlyCharges_toTotalCharges_Ratio - have a positive impact on the likelihood of churn, while features with negative coefficients (downward bars) have a negative impact.

The plot provides insights into which features are most influential in predicting churn, allowing stakeholders to prioritize and focus on key factors affecting customer churn.

The coefficients and the plot help in understanding the relative importance of different features in predicting churn, guiding decision-making processes aimed at reducing churn rates and improving customer retention strategies.

2 Advanced section

2.1 Other ML Models

[33]:
```
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
```

[34]:
```
rf_model = RandomForestClassifier(max_depth=5, n_jobs=-1, random_state=42)
gb_model = GradientBoostingClassifier(max_depth=4, random_state=42)
```

[35]:
```
rf_model.fit(X_train, y_train)
gb_model.fit(X_train, y_train)
```

[35]:
```
GradientBoostingClassifier(max_depth=4, random_state=42)
```

[36]:
```
print(rf_model.score(X_train, y_train))
print(rf_model.score(X_test, y_test))
```

```
0.8055111111111111
0.7889125799573561
```

[37]:
```
print(gb_model.score(X_train, y_train))
print(gb_model.score(X_test, y_test))
```

```
0.8387555555555556
0.7860696517412935
```

We introduce two additional classifiers, Random Forest and Gradient Boosting, and evaluate their performance on the training and test sets.

Random Forest Classifier

Trained with a maximum depth of 5 and using all available CPU cores for parallel processing. Achieved an accuracy score of approximately 80.55% on the training set and 78.89% on the test set.

Gradient Boosting Classifier

Trained with a maximum depth of 4. Achieved an accuracy score of approximately 83.88% on the training set and 78.61% on the test set.

Interpretation

Both models demonstrate decent performance, with the Gradient Boosting Classifier slightly outperforming the Random Forest Classifier on the training set.

However, the Random Forest Classifier performs slightly better on the test set compared to the Gradient Boosting Classifier, suggesting that it may generalize slightly better to unseen data.

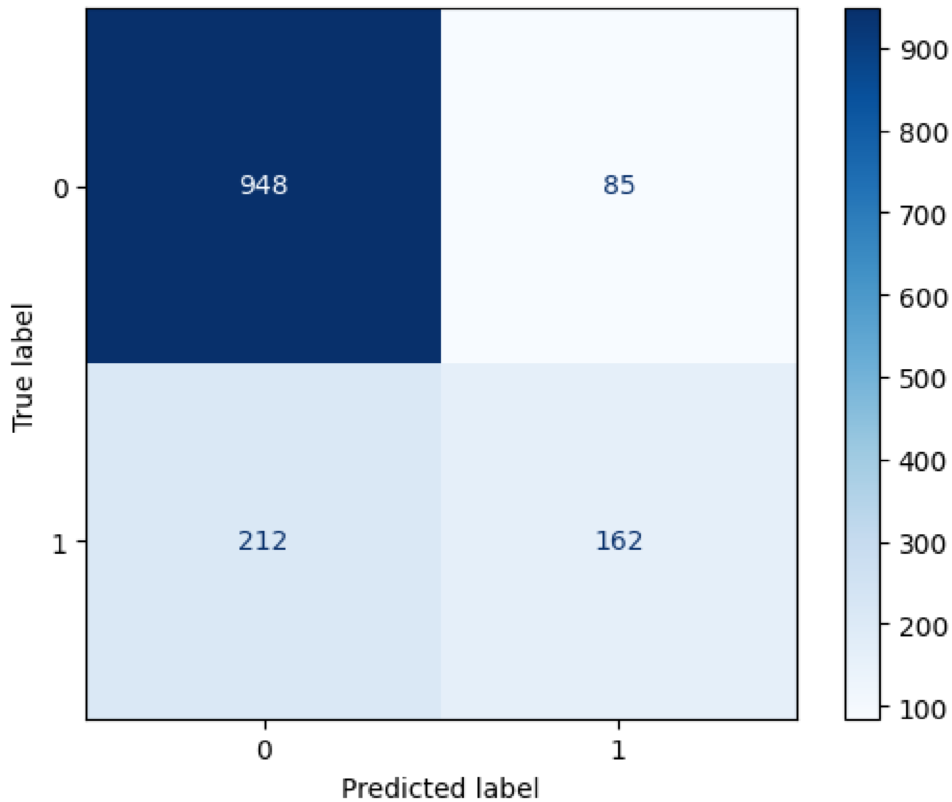
The difference in performance between the training and test sets for both models is relatively small, indicating that there is no significant overfitting.

Both models seem to provide reasonable accuracy in predicting churn, with the Gradient Boosting Classifier showing a slight advantage in terms of training set accuracy, while the Random Forest Classifier performs slightly better on the test set.

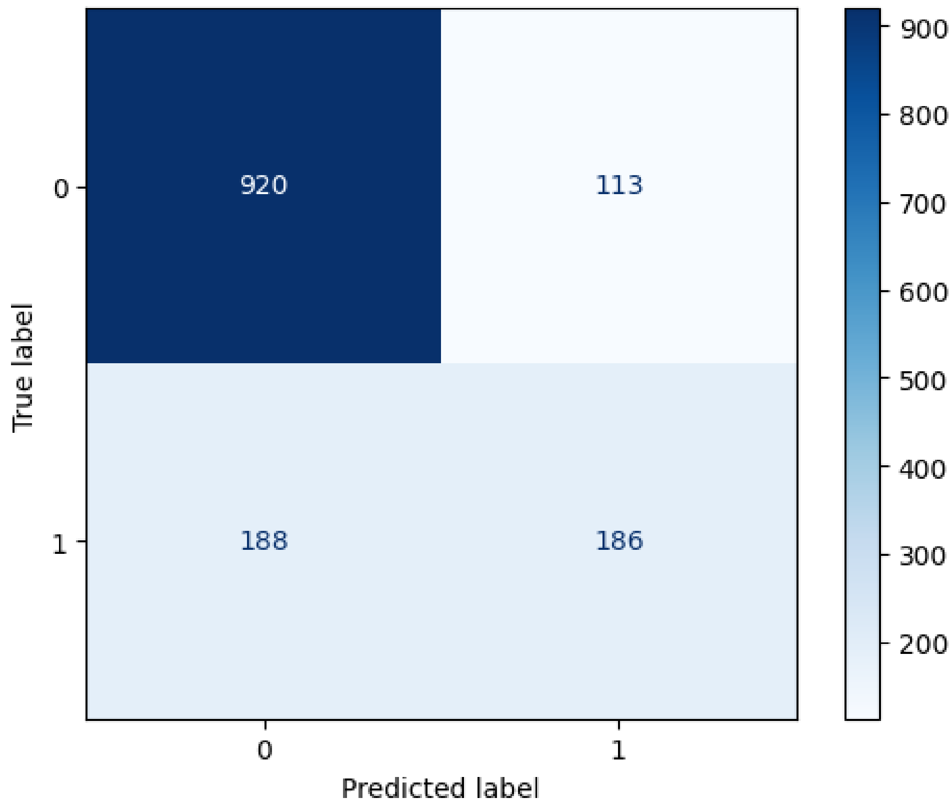
[38]:
```
# Make predictions on the test set
y_pred_rf = rf_model.predict(X_test)
y_pred_gb = gb_model.predict(X_test)
```

2.2 Confusion matix

[39]:
```
cm = confusion_matrix(y_test, y_pred_rf, labels=rf_model.classes_)

# format and display the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=rf_model.
 ↪classes_)
disp.plot(cmap=plt.cm.Blues)
plt.show()
```



[40]:
```
cm = confusion_matrix(y_test, y_pred_gb, labels=gb_model.classes_)

# format and display the confusion matrix
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=gb_model.
 ↪classes_)
disp.plot(cmap=plt.cm.Blues)
plt.show()
```



2.3 Optimizing Hyperparameters

[48]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier

# Define classifiers and hyperparameters
classifiers = {
    'Random Forest': (RandomForestClassifier(), {'max_depth': [3, 5, 10, None],␣
 ↪'n_estimators': [10, 100, 200], 'max_features': [1, 3, 5, 7],␣
 ↪'min_samples_leaf': [1, 2, 3], 'min_samples_split': [2, 3, 4]}),
    'Logistic Regression': (LogisticRegression(), {'max_iter': [2000, 4000,␣
 ↪6000]}),
    'Gradient Boosting': (GradientBoostingClassifier(), {'max_depth': [3, 5,␣
 ↪10, None], 'n_estimators': [10, 100, 200], 'max_features': [1, 3, 5, 7],␣
 ↪'min_samples_leaf': [1, 2, 3], 'min_samples_split': [2, 3, 4]})
}

# Perform grid search for each classifier
for name, (classifier, param_grid) in classifiers.items():
```

```
    grid = GridSearchCV(classifier, param_grid=param_grid, cv=3,␣
↪scoring='accuracy')
    model_grid = grid.fit(X_train, y_train)
    print(f'Best hyperparameters for {name} are: {model_grid.best_params_}')
    print(f'Best score for {name} is: {model_grid.best_score_}')
```

```
Best hyperparameters for Random Forest are: {'max_depth': 5, 'max_features': 5,
'min_samples_leaf': 3, 'min_samples_split': 3, 'n_estimators': 100}
Best score for Random Forest is: 0.7973333333333333
```

```
/home/sensei/.local/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:469: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/home/sensei/.local/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:469: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
/home/sensei/.local/lib/python3.11/site-
packages/sklearn/linear_model/_logistic.py:469: ConvergenceWarning: lbfgs failed
to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

```
Best hyperparameters for Logistic Regression are: {'max_iter': 2000}
Best score for Logistic Regression is: 0.8001777777777778
Best hyperparameters for Gradient Boosting are: {'max_depth': 3, 'max_features':
3, 'min_samples_leaf': 3, 'min_samples_split': 4, 'n_estimators': 100}
Best score for Gradient Boosting is: 0.7964444444444444
```

The output provides information about the best hyperparameters found during the grid search process for each of the three models (Random Forest, Logistic Regression, and Gradient Boosting), as well as the corresponding best scores achieved.

Random Forest

```
Best Hyperparameters
    max_depth: 5
    max_features: 5
    min_samples_leaf: 3
    min_samples_split: 3
    n_estimators: 100
Best Score: 0.7973
```

Logistic Regression

```
Best Hyperparameters
    max_iter: 2000
Best Score: 0.8002
```

Gradient Boosting

```
Best Hyperparameters
    max_depth: 3
    max_features: 3
    min_samples_leaf: 3
    min_samples_split: 4
    n_estimators: 100
Best Score: 0.7964
```

These results indicate the combination of hyperparameters that yielded the highest cross-validated accuracy score during the grid search. The scores provide an estimate of how well each model is expected to perform on unseen data.

Logistic regression achieved the highest score among the three models.

[42]:
```
model_grid.best_estimator_
```

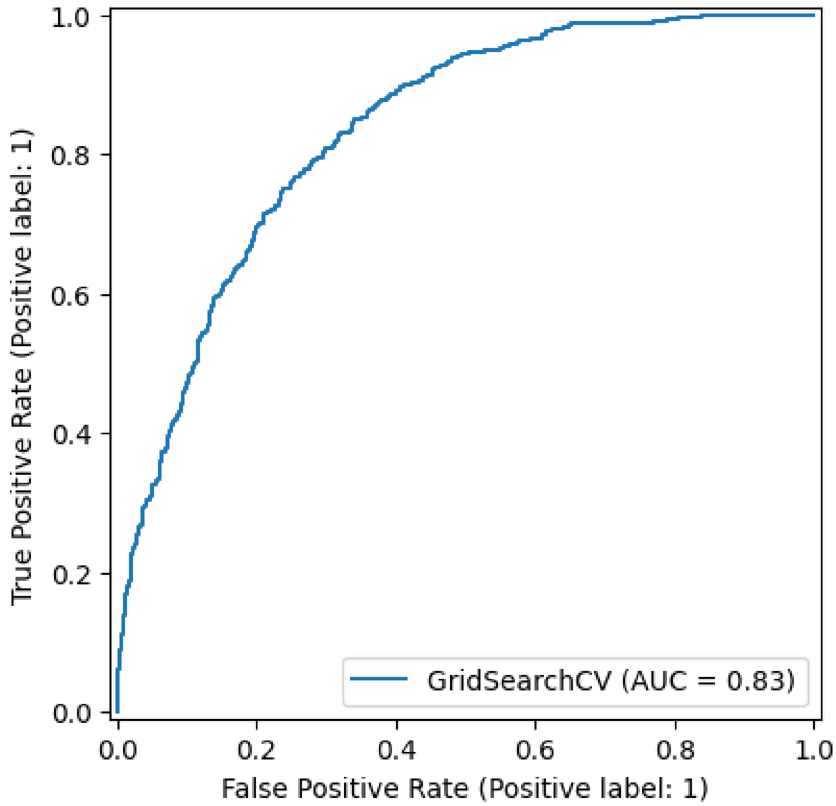
[42]:
```
LogisticRegression(max_iter=2000)
```

[43]:
```
print(classification_report(y_test, model_grid.predict(X_test)))
```

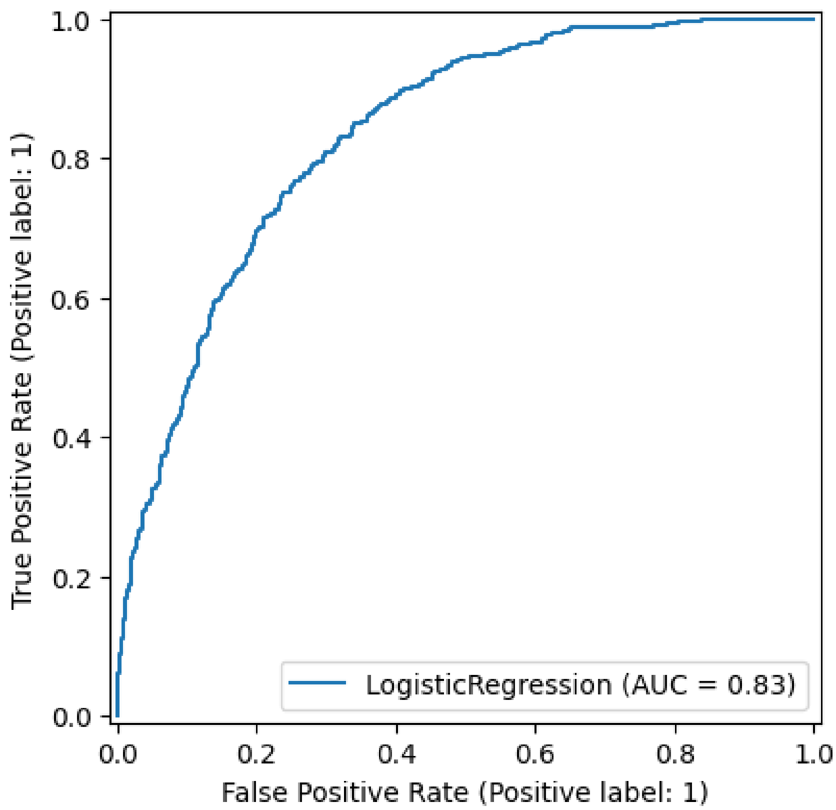
```
              precision    recall  f1-score   support

           0       0.83      0.89      0.86      1033
           1       0.63      0.49      0.55       374

    accuracy                           0.79      1407
   macro avg       0.73      0.69      0.70      1407
weighted avg       0.77      0.79      0.78      1407
```

3 ROC Curve

[44]:
```
from sklearn.metrics import RocCurveDisplay

RocCurveDisplay.from_estimator(model_grid, X_test, y_test)
plt.show()
```



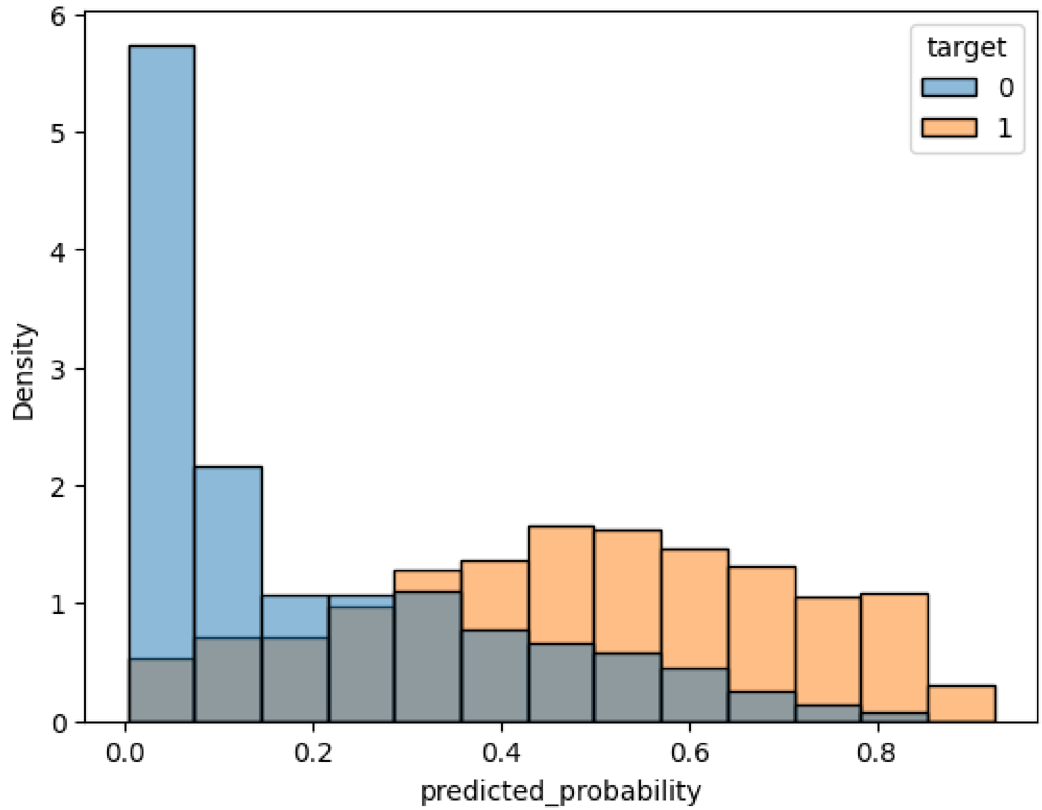
[45]:
```
RocCurveDisplay.from_estimator(lr_model, X_test, y_test)
plt.show()
```



3.1 Using Prediction probabilities

[46]:
```
probabilities = lr_model.predict_proba(X_test)[:, 1]

prob_df = pd.DataFrame(data={'predicted_probability': probabilities, 'target':␣
↪y_test})

sns.histplot(data=prob_df, x='predicted_probability', hue='target',␣
↪stat='density', common_norm=False)
```

[46]:
```
<Axes: xlabel='predicted_probability', ylabel='Density'>
```



[47]:
```
index = prob_df[(prob_df['target'] == 1) & (prob_df['predicted_probability'] <
    0.5)].index
prob_df.loc[index]

X_test.loc[index]
```

[47]:

	customerID	tenure	PhoneService	MonthlyCharges	TotalCharges	\
3716	3716	2	1	20.65	38.70	
446	446	60	1	105.90	6396.45	
4646	4646	30	0	51.20	1561.50	
2035	2035	22	1	79.35	1730.35	
2296	2296	48	1	103.25	5037.55	
…	…	…	…	…	…	
3511	3511	3	1	43.30	123.65	
4457	4457	12	1	81.70	858.60	
6737	6737	4	1	56.50	235.10	
948	948	2	1	44.95	85.15	
2950	2950	7	1	75.45	480.75	

	MonthlyCharges_to_TotalCharges_Ratio	Bank transfer (automatic)	\
3716	0.533592	0	

```
446                                  0.016556                          0
4646                                 0.032789                          0
2035                                 0.045858                          0
2296                                 0.020496                          0
…                                       …                         …
3511                                 0.350182                          0
4457                                 0.095155                          1
6737                                 0.240323                          0
948                                  0.527892                          0
2950                                 0.156942                          0

      Credit card (automatic)  Electronic check  Mailed check  Month-to-month  \
3716                        0                 1             1               0
446                         0                 0             0               0
4646                        1                 1             0               0
2035                        0                 0             0               0
2296                        1                 1             0               1
…                         …                …             …              …
3511                        0                 0             0               0
4457                        0                 1             0               0
6737                        0                 1             1               0
948                         0                 1             1               0
2950                        0                 1             1               0

      One year  Two year
3716         0         0
446          0         0
4646         0         0
2035         0         0
2296         1         0
…          …        …
3511         0         0
4457         0         0
6737         0         0
948          0         0
2950         0         0

[192 rows x 13 columns]
```

This involves evaluating the logistic regression model and visualizing its performance using various techniques.

Best Estimator: The best estimator obtained from the grid search for the logistic regression model has a max_iter parameter set to 2000.

Classification Report

Precision: Precision measures the proportion of true positive predictions among all positive predictions. A precision of 0.63 for class 1 suggests that among all instances predicted as churn (positive cases), approximately 63% are actually churned customers.

Recall: Recall, also known as sensitivity, measures the proportion of actual positive cases that were correctly identified by the model. A recall of 0.49 for class 1 indicates that the model correctly identified approximately 49% of all churned customers.

F1-score: The F1-score is the harmonic mean of precision and recall, providing a balanced measure of the model's performance. A higher F1-score indicates better balance between precision and recall.

Accuracy: Accuracy represents the overall correctness of the model's predictions, regardless of class. An accuracy of 0.79 indicates that approximately 79% of all predictions made by the model on the test set are correct.

ROC Curve Display

The ROC (Receiver Operating Characteristic) curve display is used to visualize the trade-off between the true positive rate (sensitivity) and false positive rate (1 - specificity) for different thresholds of the logistic regression model's predictions.

Histogram of Predicted Probabilities

The histogram plots the distribution of predicted probabilities output by the logistic regression model. The distribution is shown separately for each target class (0 and 1).

Analysis of Low Probability Predictions

Identification of instances in the test set where the logistic regression model predicted a low probability of churn (predicted probability < 0.5) despite the actual churn label being positive (target $= 1$). It retrieves the corresponding rows from the test set for further analysis. This analysis helps understand cases where the model might be less confident or where it may need further improvement.

3.2 Summary

A comprehensive analysis of a churn dataset is performed. We start by loading and preprocessing the data, converting categorical features into numeric representations, and splitting it into training and testing sets. Then, a Logistic Regression model is trained and its performance evaluated using accuracy score, confusion matrix, and adjustment of the classification threshold for further evaluation. Additionally, we train Random Forest and Gradient Boosting models and perform hyperparameter tuning using grid search.

The best hyperparameters and scores for each model are printed out, a classification report for the best-performing model is generated, and ROC curves plotted for comparison. The distribution of predicted probabilities for each class are visualized and we identify specific records where the model's predictions differ from the actual target values, providing a thorough understanding of the models' performance and insights into potential areas for improvement.

4 Deployment

4.1 API Integration

The model can be deployed as an API (Application Programming Interface), allowing seamless communication between the customer management system and the predictive model. APIs enable real-time predictions by sending customer data to the model and receiving churn probability scores back from the model.

It can be integrated into the company's existing customer management system to automatically predict the likelihood of churn for each customer based on their historical data and current behavior. This information can then be used by customer service representatives to proactively reach out to at-risk customers, offering personalized retention offers or resolving any issues they might be experiencing.

Moreover, the model's predictions can inform marketing strategies by identifying customer segments with a high churn probability, enabling targeted campaigns to incentivize loyalty and reduce attrition.

Additionally, the model's insights can guide product development initiatives by highlighting features or services that are correlated with customer retention, aiding in the creation of more appealing offerings.

The deployment of this churn prediction model can lead to improved customer satisfaction, reduced churn rates, and increased profitability for the business.