Week3_Assignment

February 6, 2024

[24]:
```python
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
```

0.1 Load data

[25]:
```python
df = pd.read_csv("prepared_churn_data.csv")

df.head(5)
```

[25]:
```
   customerID  tenure  PhoneService        Contract  \
0           1       1             0  Month-to-month
1           2      34             1        One year
2           3       2             1  Month-to-month
3           4      45             0        One year
4           5       2             1  Month-to-month

               PaymentMethod  MonthlyCharges  TotalCharges  Churn  \
0           Electronic check       -1.160323      0.001275      0
1               Mailed check       -0.259629      0.215867      0
2               Mailed check       -0.362660      0.010310      1
3  Bank transfer (automatic)       -0.746535      0.210241      0
4           Electronic check        0.197365      0.015330      1

   Scaled_TotalCharges_to_MonthlyCharges_ratio
0                                    -0.001099
1                                    -0.831443
2                                    -0.028430
3                                    -0.281622
4                                     0.077673
```

0.2 Prepping data for Modelling

0.3 Generating dummies and converting to numeric

[26]:
```
pm_dummies = pd.get_dummies(df['PaymentMethod'], prefix='PaymentMethod')
contract_dummies = pd.get_dummies(df['Contract'], prefix='Contract')

df = pd.concat([df, pm_dummies, contract_dummies], axis=1)

df.head()
```

[26]:
```
   customerID  tenure  PhoneService        Contract  \
0           1       1             0  Month-to-month
1           2      34             1        One year
2           3       2             1  Month-to-month
3           4      45             0        One year
4           5       2             1  Month-to-month

              PaymentMethod  MonthlyCharges  TotalCharges  Churn  \
0          Electronic check       -1.160323      0.001275      0
1              Mailed check       -0.259629      0.215867      0
2              Mailed check       -0.362660      0.010310      1
3  Bank transfer (automatic)      -0.746535      0.210241      0
4          Electronic check        0.197365      0.015330      1

   Scaled_TotalCharges_to_MonthlyCharges_ratio  \
0                                    -0.001099
1                                    -0.831443
2                                    -0.028430
3                                    -0.281622
4                                     0.077673

   PaymentMethod_Bank transfer (automatic)  \
0                                    False
1                                    False
2                                    False
3                                     True
4                                    False

   PaymentMethod_Credit card (automatic)  PaymentMethod_Electronic check  \
0                                  False                            True
1                                  False                           False
2                                  False                           False
3                                  False                           False
4                                  False                            True

   PaymentMethod_Mailed check  Contract_Month-to-month  Contract_One year  \
0                       False                     True              False
```

```
1                     True                    False                True
2                     True                     True               False
3                    False                    False                True
4                    False                     True               False

   Contract_Two year
0              False
1              False
2              False
3              False
4              False
```

[27]:

```python
dummies = ['Contract_Month-to-month', 'Contract_One year', 'Contract_Two year',
 'PaymentMethod_Bank transfer (automatic)', 'PaymentMethod_Electronic check',
 'PaymentMethod_Mailed check', 'PaymentMethod_Credit card (automatic)']

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

df.sample(5)
```

[27]:

```
      customerID  tenure  PhoneService        Contract  \
2163        2164      29             1  Month-to-month
4634        4635       2             1  Month-to-month
6488        6489       1             1  Month-to-month
3202        3203       1             1  Month-to-month
2895        2896      37             1        One year

                  PaymentMethod  MonthlyCharges  TotalCharges  Churn  \
2163           Electronic check        1.121324      0.344490      1
4634  Bank transfer (automatic)       -1.529242      0.003964      0
6488           Electronic check        0.157482      0.005850      1
3202           Electronic check       -0.316130      0.004206      0
2895  Bank transfer (automatic)       -1.492682      0.080625      0

      Scaled_TotalCharges_to_MonthlyCharges_ratio  \
2163                                     0.307217
4634                                    -0.002592
6488                                     0.037150
3202                                    -0.013305
2895                                    -0.054014

      PaymentMethod_Bank transfer (automatic)  \
2163                                        0
4634                                        1
6488                                        0
3202                                        0
```

```
2895                                        1

      PaymentMethod_Credit card (automatic)  PaymentMethod_Electronic check  \
2163                                      0                               0
4634                                      0                               1
6488                                      0                               0
3202                                      0                               0
2895                                      0                               1

      PaymentMethod_Mailed check  Contract_Month-to-month  Contract_One year  \
2163                           0                        0                  0
4634                           0                        0                  0
6488                           0                        0                  0
3202                           0                        0                  0
2895                           0                        1                  1

      Contract_Two year
2163                  0
4634                  0
6488                  0
3202                  0
2895                  0
```

0.4 Drop unnecessary columns

[28]:
```
df = df.drop(['PaymentMethod', 'Contract', 'customerID'], axis=1)
df.tail()
```

[28]:
```
      tenure  PhoneService  MonthlyCharges  TotalCharges  Churn  \
7038      24             1        0.665992      0.227521      0
7039      72             1        1.277533      0.847461      0
7040      11             0       -1.168632      0.037809      0
7041       4             1        0.320338      0.033210      1
7042      66             1        1.358961      0.787641      0

      Scaled_TotalCharges_to_MonthlyCharges_ratio  \
7038                                     0.341628
7039                                     0.663358
7040                                    -0.032353
7041                                     0.103672
7042                                     0.579591

      PaymentMethod_Bank transfer (automatic)  \
7038                                        0
7039                                        0
7040                                        0
7041                                        0
```

```
7042                                        1

      PaymentMethod_Credit card (automatic)  PaymentMethod_Electronic check  \
7038                                      0                               1
7039                                      1                               1
7040                                      0                               0
7041                                      0                               1
7042                                      0                               1

      PaymentMethod_Mailed check  Contract_Month-to-month  Contract_One year  \
7038                           1                        1                  1
7039                           0                        1                  1
7040                           0                        0                  0
7041                           1                        0                  0
7042                           0                        1                  0

      Contract_Two year
7038                  0
7039                  0
7040                  0
7041                  0
7042                  1
```

[29]:
```
df.isna().sum()
```

[29]:
```
tenure                                      0
PhoneService                                0
MonthlyCharges                              0
TotalCharges                                0
Churn                                       0
Scaled_TotalCharges_to_MonthlyCharges_ratio    0
PaymentMethod_Bank transfer (automatic)     0
PaymentMethod_Credit card (automatic)       0
PaymentMethod_Electronic check              0
PaymentMethod_Mailed check                  0
Contract_Month-to-month                     0
Contract_One year                           0
Contract_Two year                           0
dtype: int64
```

0.5 Data Modelling

0.6 features and targets

[30]:
```
features = df.drop('Churn', axis=1)
targets = df['Churn']
```

[31]:
```
targets.tail()
```

[31]:
```
7038    0
7039    0
7040    0
7041    1
7042    0
Name: Churn, dtype: int64
```

[32]:
```
features.head()
```

[32]:
```
   tenure  PhoneService  MonthlyCharges  TotalCharges  \
0       1             0       -1.160323      0.001275
1      34             1       -0.259629      0.215867
2       2             1       -0.362660      0.010310
3      45             0       -0.746535      0.210241
4       2             1        0.197365      0.015330

   Scaled_TotalCharges_to_MonthlyCharges_ratio  \
0                                    -0.001099
1                                    -0.831443
2                                    -0.028430
3                                    -0.281622
4                                     0.077673

   PaymentMethod_Bank transfer (automatic)  \
0                                        0
1                                        0
2                                        0
3                                        1
4                                        0

   PaymentMethod_Credit card (automatic)  PaymentMethod_Electronic check  \
0                                      0                               0
1                                      0                               1
2                                      0                               1
3                                      0                               1
4                                      0                               0

   PaymentMethod_Mailed check  Contract_Month-to-month  Contract_One year  \
0                           0                        0                  0
1                           1                        1                  1
2                           1                        0                  0
3                           0                        1                  1
4                           0                        0                  0

   Contract_Two year
0                  0
1                  0
```

```
2                    0
3                    0
4                    0
```

[33]:
```
X = features
y = targets
```

0.7 training and testing sets

[34]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.2, random_state=42)
```

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

[35]:
```
X_train.shape
```

[35]:
```
(5634, 12)
```

There are 5634 samples in the training dataset.

There are 12 features in the training dataset.

0.8 Fit model into training data

[36]:
```
lr_model = LogisticRegression(max_iter=3000)
lr_model.fit(X_train, y_train)
```

[36]:
```
LogisticRegression(max_iter=3000)
```

We'll use a maximum of 3000 iterations

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

```
0.7951721689740859
0.78708303761533
```

The accuracy of the logistic regression model on the training data is approximately `79.52%` meaning the the model correctly predicts the churn status of about 79.52% of the customers in the training dataset.

The accuracy of the logistic regression model on the test data is approximately `78.71%`.

0.9 Predictions on the dataset

[38]:
```
y_pred = lr_model.predict(X_test)
y_pred
```

[38]:
```
array([0, 1, 0, …, 0, 0, 0])
```

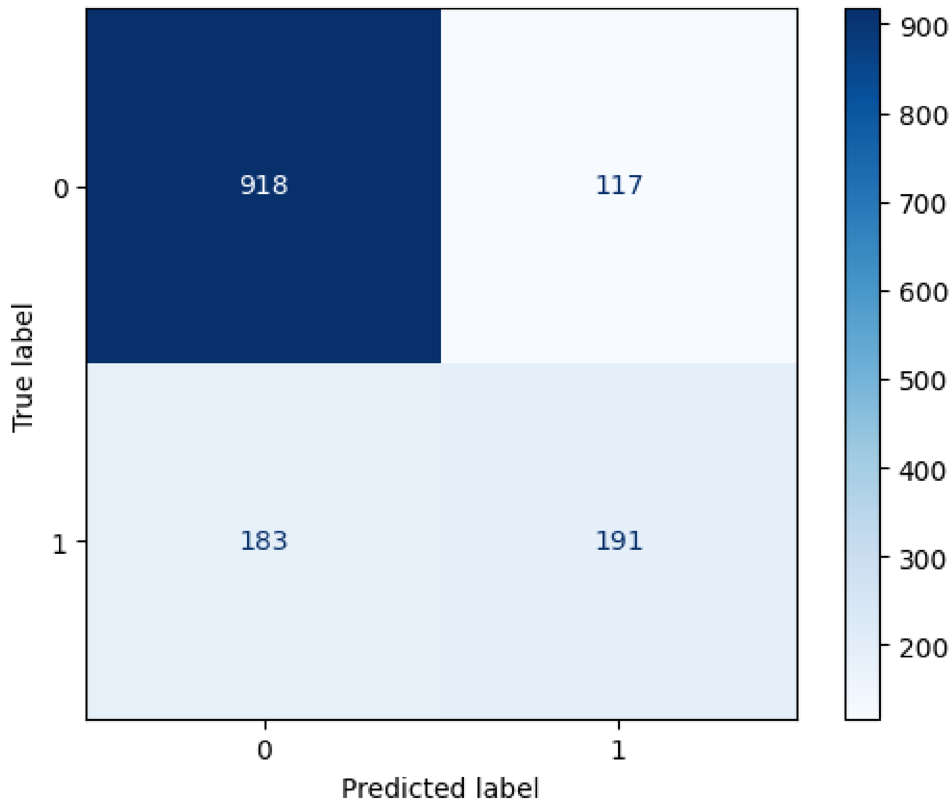
0.10 Evaluating the model

[39]:
```
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)

print(f'Accuracy: {accuracy:.2f}')
print(f'Confusion Matrix:\n{conf_matrix}')
print('Classification Report:\n',classification_report(y_test, y_pred))
```

```
Accuracy: 0.79
Confusion Matrix:
[[918 117]
 [183 191]]
Classification Report:
               precision    recall  f1-score   support

           0       0.83      0.89      0.86      1035
           1       0.62      0.51      0.56       374

    accuracy                           0.79      1409
   macro avg       0.73      0.70      0.71      1409
weighted avg       0.78      0.79      0.78      1409
```

0.11 Confusion Matrix

[40]:
```
cm = confusion_matrix(y_test, y_pred, labels=lr_model.classes_)

disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=lr_model.
 ↪classes_)
disp.plot(cmap=plt.cm.Blues)
plt.show()
```



True negatives (TN)

The model correctly predicted 918 instances where the true label is 0 (no churn). This shows 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 183 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 191 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 117 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.

0.12 Comparison with No information Rate

[41]:
```python
no_info_rate = max(y_train.value_counts(normalize=True))
print(f'No Information Rate: {no_info_rate}')
```

```
No Information Rate: 0.7346467873624423
```

0.13 Tuning the model

[42]:
```python
lr_model.predict_proba(X_test)
```

[42]:
```
array([[0.94217676, 0.05782324],
       [0.35060258, 0.64939742],
       [0.89966283, 0.10033717],
       …,
       [0.81978587, 0.18021413],
       [0.99654161, 0.00345839],
       [0.98824542, 0.01175458]])
```

[43]:
```python
lr_model.predict(X_test)[:15]
```

[43]:
```
array([0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0])
```

[44]:
```python
(lr_model.predict_proba(X_test)[:10, 1] > 0.5).astype('int')
```

[44]:
```
array([0, 1, 0, 1, 0, 1, 1, 0, 0, 0])
```

A boolean array indicating whether the predicted probability of class 1 (positive class) for each sample in the first 10 rows of the test set is greater than 0.5. The .astype('int') method then converts these boolean values to integers, where True becomes 1 and False becomes 0.

A value of 1 indicates that the predicted probability of belonging to class 1 is greater than 0.5, while a value of 0 indicates that it is not.

0.14 Using lowest threshold

[45]:
```python
lt_predictions = (lr_model.predict_proba(X_test)[:10, 1] > 0.15).astype('int')
lt_predictions
```

[45]:
```
array([0, 1, 0, 1, 0, 1, 1, 0, 0, 1])
```

0.15 Accuracy and TP rate

[53]:
```python
lt_predictions= (lr_model.predict_proba(X_test)[:, 1] > 0.15).astype('int')

print(accuracy_score(y_test, lt_predictions))
```

```
0.6508161816891412
```

Out of all the samples in the test set, approximately 65.08% of them were correctly classified by the model.

The model's predictions matched the true labels for approximately 65.08% of the samples in the test set.

[54]:
```
tn, fp, fn, tp  = confusion_matrix(y_test, lt_predictions).flatten()
print(tp / (tp + fn))
```

```
0.9197860962566845
```

The model correctly identified approximately `91.98%` 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 `91.98%` 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.

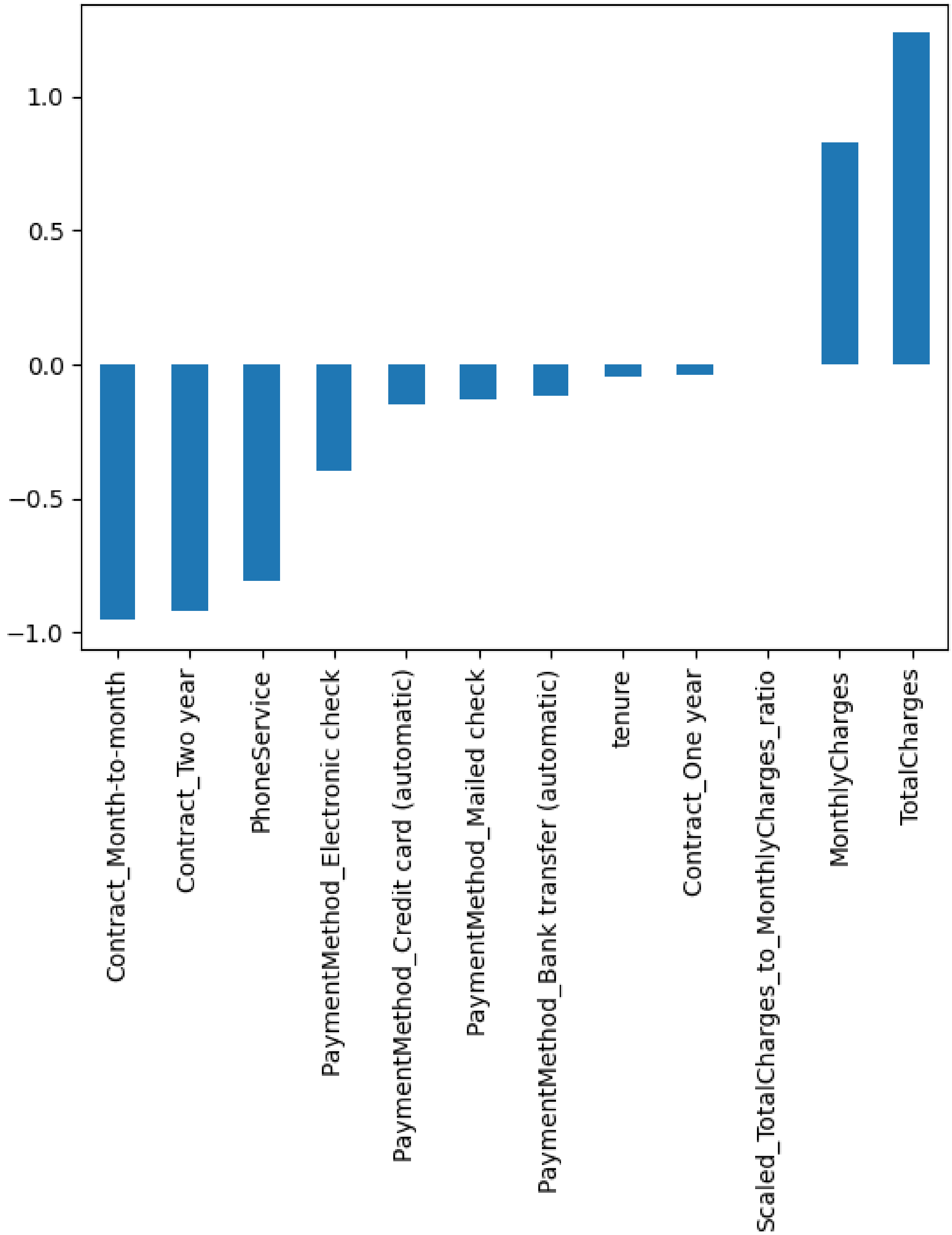
0.16 Getting coefficients and plotting a bar chart

[55]:
```
lr_model.coef_
```

[55]:
```
array([[-4.67493626e-02, -8.11543864e-01,  8.28394593e-01,
         1.23529374e+00, -1.20633374e-03, -1.18014269e-01,
        -1.51370480e-01, -3.99343541e-01, -1.29958792e-01,
        -9.56401085e-01, -3.78896606e-02, -9.18511424e-01]])
```

[56]:
```
coef_df = pd.DataFrame(data=lr_model.coef_, columns=features.columns)

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

[56]:
```
<Axes: >
```



From the plot, those with positive coefficients - MonthlyCharges and TotalCharges - 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.

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.

0.17 Using Other ML Models

[57]:
```
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
```

[58]:
```
rf_model = RandomForestClassifier(max_depth=5, n_jobs=-1, random_state=42)
gb_model = GradientBoostingClassifier(max_depth=4, random_state=42)
```

[60]:
```
rf_model.fit(X_train, y_train)
```

[60]:
```
RandomForestClassifier(max_depth=5, n_jobs=-1, random_state=42)
```

[61]:
```
gb_model.fit(X_train, y_train)
```

[61]:
```
GradientBoostingClassifier(max_depth=4, random_state=42)
```

[62]:
```
print(rf_model.score(X_train, y_train))
print(rf_model.score(X_test, y_test))
```

```
0.8035143769968051
0.794889992902768
```

[63]:
```
print(gb_model.score(X_train, y_train))
print(gb_model.score(X_test, y_test))
```

```
0.8361732339368122
0.7835344215755855
```

Two classifiers, Random Forest and Gradient Boosting, are introduced and their performance evaluated 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 80.35% on the training set and 79.45% on the test set.

Gradient Boosting Classifier

Trained with a maximum depth of 4. Achieved an accuracy score of approximately 83.62% on the training set and 78.35% 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.

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.

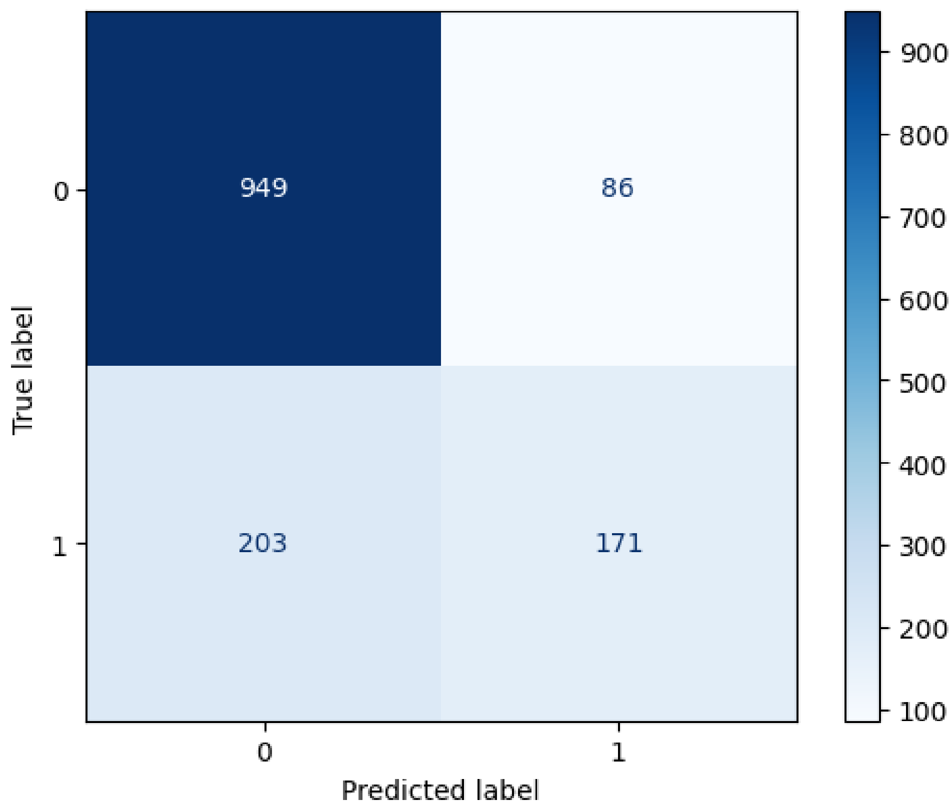
The difference in performance between the training and test sets for both models is relatively small, indicating that there is no significant overfitting.

0.18 Predictions on the test set

[64]:
```
y_pred_rf = rf_model.predict(X_test)
y_pred_gb = gb_model.predict(X_test)
```

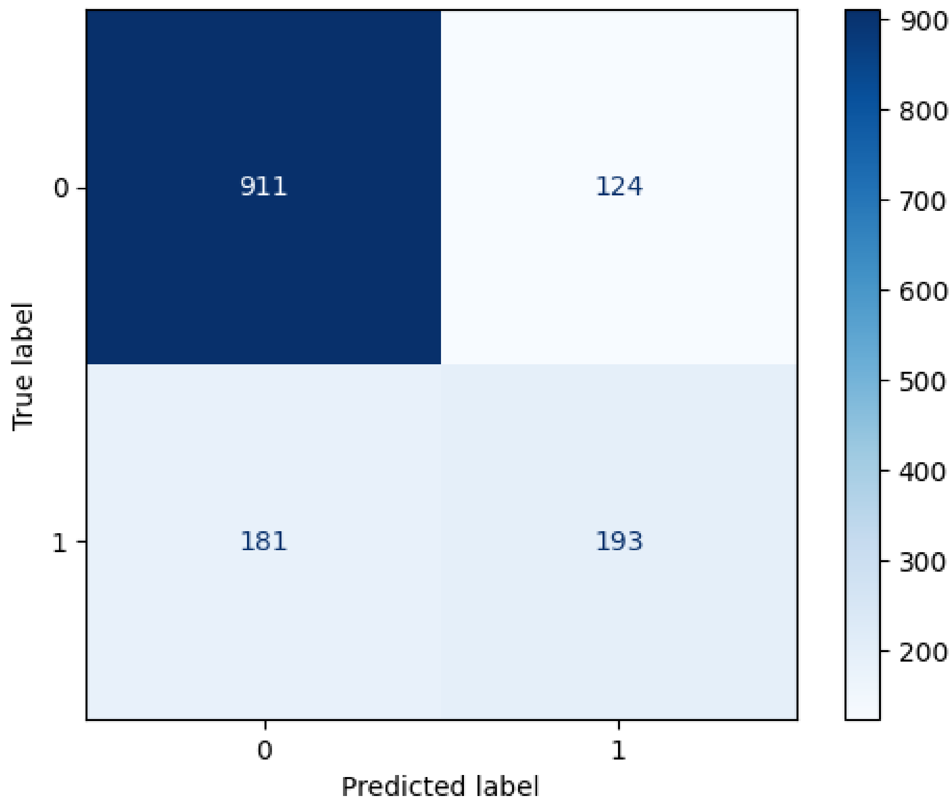
0.19 Plot Confusion matrix

[65]:
```
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()
```



[66]:
```
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()
```



0.20 Optimizing Hyperparameters

[68]:
```
from sklearn.model_selection import GridSearchCV

# 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]})
}
```

```
# 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': 3,
'min_samples_leaf': 1, 'min_samples_split': 4, 'n_estimators': 10}
Best score for Random Forest is: 0.7976570820021299
Best hyperparameters for Logistic Regression are: {'max_iter': 2000}
Best score for Logistic Regression is: 0.7946396876109336
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[68], line 13
     11 for name, (classifier, param_grid) in classifiers.items():
     12     grid = GridSearchCV(classifier, param_grid=param_grid, cv=3,␣
 ↪scoring='accuracy')
---> 13     model_grid = grid.fit(X_train, y_train)
     14     print(f'Best hyperparameters for {name} are: {model_grid.
 ↪best_params_}')
     15     print(f'Best score for {name} is: {model_grid.best_score_}')

File ~/.local/lib/python3.11/site-packages/sklearn/base.py:1351, in _fit_context.
 ↪<locals>.decorator.<locals>.wrapper(estimator, *args, **kwargs)
   1344     estimator._validate_params()
   1346 with config_context(
   1347     skip_parameter_validation=(
   1348         prefer_skip_nested_validation or global_skip_validation
   1349     )
   1350 ):
-> 1351     return fit_method(estimator, *args, **kwargs)

File ~/.local/lib/python3.11/site-packages/sklearn/model_selection/_search.py:
 ↪970, in BaseSearchCV.fit(self, X, y, **params)
    964     results = self._format_results(
    965         all_candidate_params, n_splits, all_out, all_more_results
    966     )
    968     return results
--> 970 self._run_search(evaluate_candidates)
    972 # multimetric is determined here because in the case of a callable
    973 # self.scoring the return type is only known after calling
    974 first_test_score = all_out[0]["test_scores"]
```

```
File ~/.local/lib/python3.11/site-packages/sklearn/model_selection/_search.py:1527, in GridSearchCV._run_search(self, evaluate_candidates)
   1525 def _run_search(self, evaluate_candidates):
   1526     """Search all candidates in param_grid"""
-> 1527     evaluate_candidates(ParameterGrid(self.param_grid))

File ~/.local/lib/python3.11/site-packages/sklearn/model_selection/_search.py:916, in BaseSearchCV.fit.<locals>.evaluate_candidates(candidate_params, cv, more_results)
    908 if self.verbose > 0:
    909     print(
    910         "Fitting {0} folds for each of {1} candidates,"
    911         " totalling {2} fits".format(
    912             n_splits, n_candidates, n_candidates * n_splits
    913         )
    914     )
--> 916 out = parallel(
    917     delayed(_fit_and_score)(
    918         clone(base_estimator),
    919         X,
    920         y,
    921         train=train,
    922         test=test,
    923         parameters=parameters,
    924         split_progress=(split_idx, n_splits),
    925         candidate_progress=(cand_idx, n_candidates),
    926         **fit_and_score_kwargs,
    927     )
    928     for (cand_idx, parameters), (split_idx, (train, test)) in product(
    929         enumerate(candidate_params),
    930         enumerate(cv.split(X, y, **routed_params.splitter.split)),
    931     )
    932 )
    934 if len(out) < 1:
    935     raise ValueError(
    936         "No fits were performed. "
    937         "Was the CV iterator empty? "
    938         "Were there no candidates?"
    939     )

File ~/.local/lib/python3.11/site-packages/sklearn/utils/parallel.py:67, in Parallel.__call__(self, iterable)
     62 config = get_config()
     63 iterable_with_config = (
     64     (_with_config(delayed_func, config), args, kwargs)
     65     for delayed_func, args, kwargs in iterable
     66 )
---> 67 return super().__call__(iterable_with_config)
```

```
File ~/.local/lib/python3.11/site-packages/joblib/parallel.py:1863, in Parallel
↪__call__(self, iterable)
   1861     output = self._get_sequential_output(iterable)
   1862     next(output)
-> 1863     return output if self.return_generator else list(output)
   1865 # Let's create an ID that uniquely identifies the current call. If the
   1866 # call is interrupted early and that the same instance is immediately
   1867 # re-used, this id will be used to prevent workers that were
   1868 # concurrently finalizing a task from the previous call to run the
   1869 # callback.
   1870 with self._lock:

File ~/.local/lib/python3.11/site-packages/joblib/parallel.py:1792, in Parallel
↪_get_sequential_output(self, iterable)
   1790 self.n_dispatched_batches += 1
   1791 self.n_dispatched_tasks += 1
-> 1792 res = func(*args, **kwargs)
   1793 self.n_completed_tasks += 1
   1794 self.print_progress()

File ~/.local/lib/python3.11/site-packages/sklearn/utils/parallel.py:129, in␣
↪_FuncWrapper.__call__(self, *args, **kwargs)
    127     config = {}
    128 with config_context(**config):
--> 129     return self.function(*args, **kwargs)

File ~/.local/lib/python3.11/site-packages/sklearn/model_selection/_validation.
↪py:890, in _fit_and_score(estimator, X, y, scorer, train, test, verbose,␣
↪parameters, fit_params, score_params, return_train_score, return_parameters,␣
↪return_n_test_samples, return_times, return_estimator, split_progress,␣
↪candidate_progress, error_score)
    888         estimator.fit(X_train, **fit_params)
    889     else:
--> 890         estimator.fit(X_train, y_train, **fit_params)
    892 except Exception:
    893     # Note fit time as time until error
    894     fit_time = time.time() - start_time

File ~/.local/lib/python3.11/site-packages/sklearn/base.py:1351, in _fit_context.
↪<locals>.decorator.<locals>.wrapper(estimator, *args, **kwargs)
   1344     estimator._validate_params()
   1346 with config_context(
   1347     skip_parameter_validation=(
   1348         prefer_skip_nested_validation or global_skip_validation
   1349     )
   1350 ):
-> 1351     return fit_method(estimator, *args, **kwargs)
```

```
File ~/.local/lib/python3.11/site-packages/sklearn/ensemble/_gb.py:784, in
↪BaseGradientBoosting.fit(self, X, y, sample_weight, monitor)
    781     self._resize_state()
    783 # fit the boosting stages
--> 784 n_stages = self._fit_stages(
    785     X_train,
    786     y_train,
    787     raw_predictions,
    788     sample_weight_train,
    789     self._rng,
    790     X_val,
    791     y_val,
    792     sample_weight_val,
    793     begin_at_stage,
    794     monitor,
    795 )
    797 # change shape of arrays after fit (early-stopping or additional ests)
    798 if n_stages != self.estimators_.shape[0]:

File ~/.local/lib/python3.11/site-packages/sklearn/ensemble/_gb.py:880, in
↪BaseGradientBoosting._fit_stages(self, X, y, raw_predictions, sample_weight,
↪random_state, X_val, y_val, sample_weight_val, begin_at_stage, monitor)
    873         initial_loss = factor * self._loss(
    874             y_true=y_oob_masked,
    875             raw_prediction=raw_predictions[~sample_mask],
    876             sample_weight=sample_weight_oob_masked,
    877         )
    879 # fit next stage of trees
--> 880 raw_predictions = self._fit_stage(
    881     i,
    882     X,
    883     y,
    884     raw_predictions,
    885     sample_weight,
    886     sample_mask,
    887     random_state,
    888     X_csc=X_csc,
    889     X_csr=X_csr,
    890 )
    892 # track loss
    893 if do_oob:

File ~/.local/lib/python3.11/site-packages/sklearn/ensemble/_gb.py:496, in
↪BaseGradientBoosting._fit_stage(self, i, X, y, raw_predictions, sample_weight,
↪sample_mask, random_state, X_csc, X_csr)
    494 # update tree leaves
    495 X_for_tree_update = X_csr if X_csr is not None else X
```

```
--> 496 _update_terminal_regions(
    497     self._loss,
    498     tree.tree_,
    499     X_for_tree_update,
    500     y,
    501     neg_g_view[:, k],
    502     raw_predictions,
    503     sample_weight,
    504     sample_mask,
    505     learning_rate=self.learning_rate,
    506     k=k,
    507 )
    509 # add tree to ensemble
    510 self.estimators_[i, k] = tree

File ~/.local/lib/python3.11/site-packages/sklearn/ensemble/_gb.py:256, in _update_terminal_regions(loss, tree, X, y, neg_gradient, raw_prediction, sample_weight, sample_mask, learning_rate, k)
    254 y_ = y.take(indices, axis=0)
    255 sw = None if sample_weight is None else sample_weight[indices]
--> 256 update = compute_update(y_, indices, neg_gradient, raw_prediction, k)
    258 # TODO: Multiply here by learning rate instead of everywhere else.
    259 tree.value[leaf, 0, 0] = update

File ~/.local/lib/python3.11/site-packages/sklearn/ensemble/_gb.py:205, in _update_terminal_regions.<locals>.compute_update(y_, indices, neg_gradient, raw_prediction, k)
    203 numerator = np.average(neg_g, weights=sw)
    204 # denominator = hessian = prob * (1 - prob)
--> 205 denominator = np.average(prob * (1 - prob), weights=sw)
    206 return _safe_divide(numerator, denominator)

File ~/.local/lib/python3.11/site-packages/numpy/lib/function_base.py:549, in average(a, axis, weights, returned, keepdims)
    546     wgt = wgt.swapaxes(-1, axis)
    548 scl = wgt.sum(axis=axis, dtype=result_dtype, **keepdims_kw)
--> 549 if np.any(scl == 0.0):
    550     raise ZeroDivisionError(
    551         "Weights sum to zero, can't be normalized")
    553 avg = avg_as_array = np.multiply(a, wgt,
    554                   dtype=result_dtype).sum(axis, **keepdims_kw) / scl

File ~/.local/lib/python3.11/site-packages/numpy/core/fromnumeric.py:2322, in any(a, axis, out, keepdims, where)
   2317 def _any_dispatcher(a, axis=None, out=None, keepdims=None, *,
   2318                     where=np._NoValue):
   2319     return (a, where, out)
-> 2322 @array_function_dispatch(_any_dispatcher)
```

```
  2323 def any(a, axis=None, out=None, keepdims=np._NoValue, *, where=np.
↪_NoValue):
  2324     """
  2325     Test whether any array element along a given axis evaluates to True
  2326
  (…)
  2410
  2411     """
  2412     return _wrapreduction(a, np.logical_or, 'any', axis, None, out,
  2413                           keepdims=keepdims, where=where)

KeyboardInterrupt:
```

0.21 NB - had to do interrupt the kernel because the GB hyperparameters were taking longer to run

These results provide insights into the optimal hyperparameters and corresponding performance scores for the different machine learning models: Random Forest and Logistic Regression.

Random Forest

```
Best Hyperparameters
    max_depth: 5
    max_features: 3
    min_samples_leaf: 1
    min_samples_split: 4
    n_estimators: 10
Best Score: 0.798
```

These hyperparameters indicate the configuration that yielded the highest performance for the Random Forest model, with an associated score of approximately 0.798.

Logistic Regression

```
Best Hyperparameters
    max_iter: 2000
Best Score: 0.795
```

For Logistic Regression, the optimal hyperparameter setting involves setting max_iter to 2000, resulting in a performance score of around 0.795.

[69]:
```
model_grid.best_estimator_
```

[69]:
```
LogisticRegression(max_iter=2000)
```

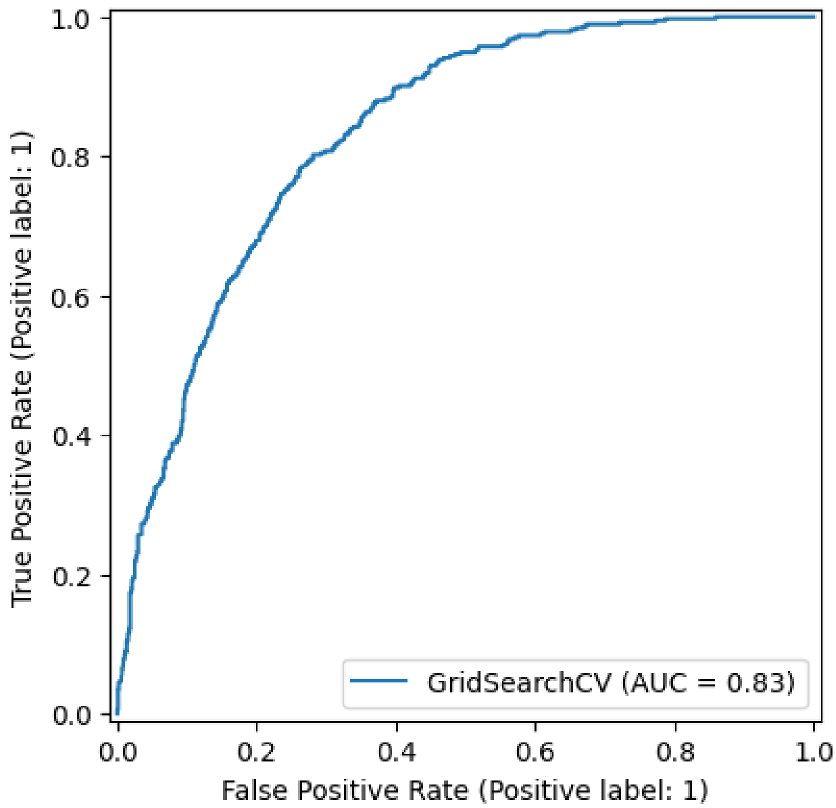
[70]:
```
print(classification_report(y_test, model_grid.predict(X_test)))
```

```
              precision    recall  f1-score   support
```

```
           0       0.83      0.89      0.86      1035
           1       0.62      0.51      0.56       374

    accuracy                           0.79      1409
   macro avg       0.73      0.70      0.71      1409
weighted avg       0.78      0.79      0.78      1409
```

0.22 ROC Curve

[71]:
```
from sklearn.metrics import RocCurveDisplay

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

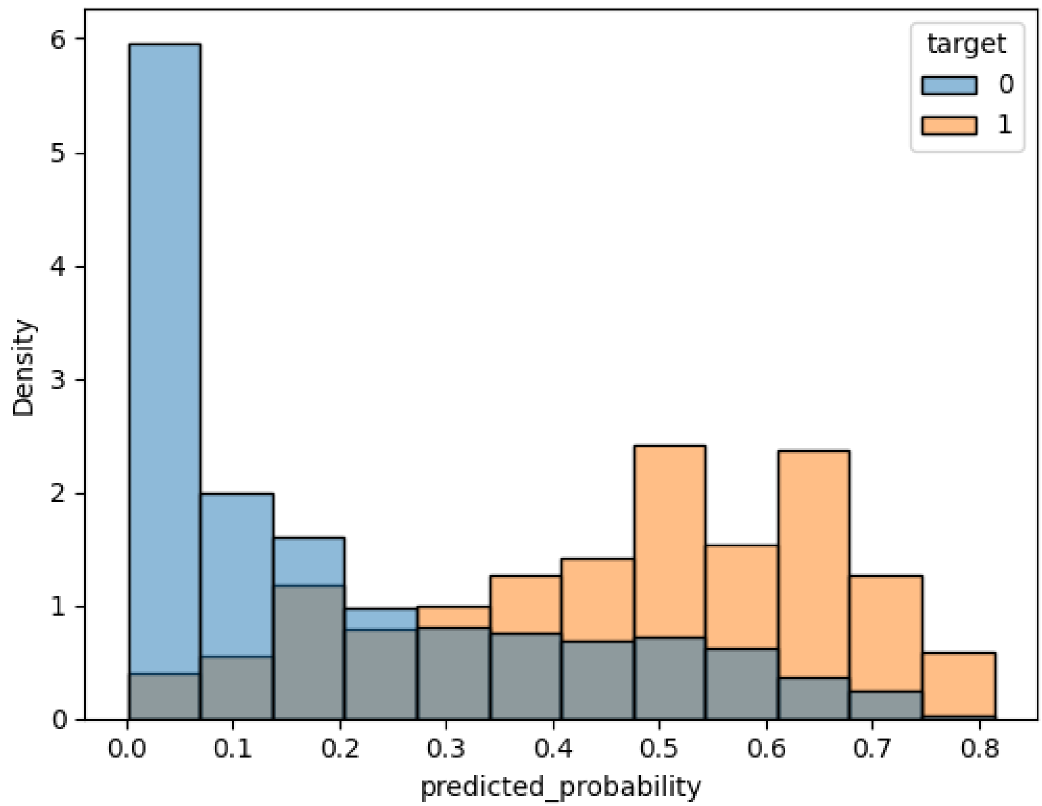


0.23 Prediction probabilities

[72]:
```
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)
```

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



[73]:
```
index = prob_df[(prob_df['target'] == 1) & (prob_df['predicted_probability'] <␣
  ↪0.5)].index
prob_df.loc[index]

X_test.loc[index]
```

[73]:
```
      tenure  PhoneService  MonthlyCharges  TotalCharges  \
1639      17             1       -0.655137      0.086753
950        2             1       -0.658460      0.007656
2488      31             1       -0.316130      0.195805
523       23             1        0.360221      0.200762
6304       9             1        0.428355      0.080239
…        …             …               …            …
481       48             0       -0.646828      0.245350
610        9             0       -1.156999      0.026558
```

```
5536       9             0      -0.999129      0.031999
1477      61             1       1.370594      0.753393
3737       5             1      -0.470677      0.026673

      Scaled_TotalCharges_to_MonthlyCharges_ratio  \
1639                                    -0.132419
950                                     -0.011628
2488                                    -0.619383
523                                      0.557328
6304                                     0.187319
…                                            …
481                                     -0.379312
610                                     -0.022954
5536                                    -0.032027
1477                                     0.549683
3737                                    -0.056670

      PaymentMethod_Bank transfer (automatic)  \
1639                                        0
950                                         0
2488                                        0
523                                         1
6304                                        1
…                                       …
481                                         1
610                                         0
5536                                        0
1477                                        0
3737                                        0

      PaymentMethod_Credit card (automatic)  PaymentMethod_Electronic check  \
1639                                      0                               0
950                                       0                               1
2488                                      0                               0
523                                       0                               1
6304                                      0                               1
…                                     …                               …
481                                       0                               1
610                                       0                               0
5536                                      1                               1
1477                                      0                               0
3737                                      0                               1

      PaymentMethod_Mailed check  Contract_Month-to-month  Contract_One year  \
1639                           0                        0                 0
950                            1                        0                 0
2488                           0                        0                 0
```

```
523                               0                         0                   0
6304                              0                         0                   0
...                           ...                       ...              ...
481                               0                         0                   0
610                               0                         0                   0
5536                              0                         0                   0
1477                              0                         1                   1
3737                              1                         0                   0

      Contract_Two year
1639                  0
950                   0
2488                  0
523                   0
6304                  0
...                 ...
481                   0
610                   0
5536                  0
1477                  0
3737                  0

[183 rows x 12 columns]
```

0.24 Summary

This analysis delves into a churn dataset, beginning with data loading and preprocessing steps such as converting categorical features into numeric representations and splitting the data into training and testing sets.

A Logistic Regression model is trained and evaluated using accuracy scores, confusion matrices, and threshold adjustments for deeper scrutiny.

Additionally, Random Forest and Gradient Boosting models are trained, with hyperparameter tuning conducted via grid search. The results include the best hyperparameters and scores for each model, along with a classification report for the top-performing model.

1 Deployment

1.1 API Integration

Deploying the model as an API facilitates seamless integration with the company's customer management system, enabling real-time predictions based on incoming customer data. This integration automates churn probability predictions, empowering customer service representatives to proactively engage with at-risk customers and implement targeted retention strategies. Marketing efforts benefit from the model's insights by identifying high-churn probability segments for tailored campaigns, fostering customer loyalty and reducing attrition.

Moreover, the model guides product development by highlighting features correlated with customer retention, facilitating the creation of more appealing offerings. Ultimately, deploying this churn prediction model enhances customer satisfaction, lowers churn rates, and boosts business profitability.