ModelSelectionForLogisticRegression

August 8, 2023

1 Lab 5: Model Selection for Logistic Regression

[27]:
```
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪precision_recall_curve
```

In this lab assignment, you will:

1. Load the Airbnb "listings" data set.
2. Train and test a logistic regression (LR) model using the scikit-learn default hyperparameter values.
3. Perform a grid search to identify the LR hyperparameter value that results in the best cross-validation score.
4. Fit the optimal model to the training data and make predictions on the test data.
5. Create a confusion matrix for both models.
6. Plot a precision-recall curve for both models.
7. Plot the ROC and compute the AUC for both models.
8. Perform feature selection.

Note: Some of the code cells in this notebook may take a while to run.

1.1 Part 1: Load the Data Set

We will work with a preprocessed version of the Airbnb NYC "listings" data set.

Task: In the code cell below, use the same method you have been using to load the data using `pd.read_csv()` and save it to DataFrame `df`.

You will be working with the file named "airbnb_readytofit.csv.gz" that is located in a folder named "data".

[28]:
```
# YOUR CODE HERE
df=pd.read_csv('data/airbnb_readytofit.csv.gz')
```

```
df.head()
```

[28]:
```
   host_is_superhost  host_has_profile_pic  host_identity_verified  \
0              False                  True                    True
1              False                  True                    True
2              False                  True                    True
3              False                  True                   False
4              False                  True                    True

   has_availability  instant_bookable  host_response_rate  \
0              True             False           -0.578829
1              True             False           -4.685756
2              True             False            0.578052
3              True             False            0.578052
4              True             False           -0.054002

   host_acceptance_rate  host_listings_count  host_total_listings_count  \
0             -2.845589            -0.054298                  -0.054298
1             -0.430024            -0.112284                  -0.112284
2             -2.473964            -0.112284                  -0.112284
3              1.010024            -0.112284                  -0.112284
4             -0.066308            -0.112284                  -0.112284

   accommodates  ...  n_host_verifications  \
0     -1.007673  ...              1.888373
1      0.067470  ...              0.409419
2      0.605041  ...             -1.069535
3     -0.470102  ...             -0.576550
4     -1.007673  ...              0.902404

   neighbourhood_group_cleansed_Bronx  neighbourhood_group_cleansed_Brooklyn  \
0                                 0.0                                    0.0
1                                 0.0                                    1.0
2                                 0.0                                    1.0
3                                 0.0                                    0.0
4                                 0.0                                    0.0

   neighbourhood_group_cleansed_Manhattan  \
0                                     1.0
1                                     0.0
2                                     0.0
3                                     1.0
4                                     1.0

   neighbourhood_group_cleansed_Queens  \
0                                  0.0
1                                  0.0
```

```
2                                0.0
3                                0.0
4                                0.0

   neighbourhood_group_cleansed_Staten Island  room_type_Entire home/apt  \
0                                         0.0                        1.0
1                                         0.0                        1.0
2                                         0.0                        1.0
3                                         0.0                        0.0
4                                         0.0                        0.0

   room_type_Hotel room  room_type_Private room  room_type_Shared room
0                   0.0                     0.0                    0.0
1                   0.0                     0.0                    0.0
2                   0.0                     0.0                    0.0
3                   0.0                     1.0                    0.0
4                   0.0                     1.0                    0.0

[5 rows x 50 columns]
```

1.2 Part 2: Create Training and Test Data Sets

1.2.1 Create Labeled Examples

Task: Create labeled examples from DataFrame `df`. In the code cell below, carry out the following steps:

- Get the `host_is_superhost` column from DataFrame `df` and assign it to the variable `y`. This will be our label.
- Get all other columns from DataFrame `df` and assign them to the variable `X`. These will be our features.

First, we will store the label column as a separate object, called `y`, and consequently remove that column from the `X` feature set:

[29]:
```
# YOUR CODE HERE
y=df['host_is_superhost']
X=df.drop('host_is_superhost',axis=1)
```

1.2.2 Split Labeled Examples Into Training and Test Sets

Task: In the code cell below, create training and test sets out of the labeled examples.

1. Use scikit-learn's `train_test_split()` function to create the data sets.
2. Specify:
   - A test set that is 10 percent of the size of the data set.
   - A seed value of '1234'.

[30]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=.
↪1,random_state=1234)
```

[31]:
```
X_train.head()
```

[31]:
```
       host_has_profile_pic  host_identity_verified  has_availability  \
326                    True                    True              True
26890                  True                   False              True
16767                  True                    True              True
27743                  True                    True              True
9783                   True                   False              True

       instant_bookable  host_response_rate  host_acceptance_rate  \
326               False           -0.868049             -2.706229
26890             False            0.578052              1.010024
16767             False            0.578052              0.638399
27743              True            0.288832              1.010024
9783              False            0.578052              1.010024

       host_listings_count  host_total_listings_count  accommodates  \
326              -0.120567                  -0.120567     -0.470102
26890            -0.120567                  -0.120567      0.605041
16767            -0.112284                  -0.112284     -0.470102
27743            -0.120567                  -0.120567      2.755328
9783             -0.104000                  -0.104000      0.605041

       bathrooms  ...  n_host_verifications  \
326    -0.337606  ...             -0.083566
26890  -0.337606  ...             -1.562519
16767  -0.337606  ...              1.395388
27743   2.036990  ...             -1.562519
9783   -0.337606  ...             -1.069535

       neighbourhood_group_cleansed_Bronx  \
326                                   0.0
26890                                 0.0
16767                                 0.0
27743                                 0.0
9783                                  0.0

       neighbourhood_group_cleansed_Brooklyn  \
326                                      0.0
26890                                    0.0
16767                                    1.0
27743                                    0.0
9783                                     0.0

       neighbourhood_group_cleansed_Manhattan  \
```

```
326                                       1.0
26890                                     0.0
16767                                     0.0
27743                                     1.0
9783                                      1.0

       neighbourhood_group_cleansed_Queens  \
326                                    0.0
26890                                  1.0
16767                                  0.0
27743                                  0.0
9783                                   0.0

       neighbourhood_group_cleansed_Staten Island  room_type_Entire home/apt  \
326                                           0.0                        1.0
26890                                         0.0                        1.0
16767                                         0.0                        1.0
27743                                         0.0                        1.0
9783                                          0.0                        1.0

       room_type_Hotel room  room_type_Private room  room_type_Shared room
326                     0.0                     0.0                    0.0
26890                   0.0                     0.0                    0.0
16767                   0.0                     0.0                    0.0
27743                   0.0                     0.0                    0.0
9783                    0.0                     0.0                    0.0

[5 rows x 49 columns]
```

1.3 Part 3: Fit and Evaluate a Logistic Regression Model With Default Hyperparameter Values

Task: In the code cell below:

1. Using the scikit-learn `LogisticRegression` class, create a logistic regression model object with the following arguments: `max_iter=1000`. You will use the scikit-learn default value for hyperparameter C, which is 1.0. Assign the model object to the variable `model_default`.

2. Fit the model to the training data.

[32]:
```
# 1. Create the  Scikit-learn LogisticRegression model object below and assign␣
 ↪to variable 'model_default'
model_default=LogisticRegression(max_iter=1000)

# 2. Fit the model to the training data below
model_default.fit(X_train,y_train)
```

[32]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=1000,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

Task: Test your model on the test set (`X_test`).

1. Use the `predict_proba()` method to use the fitted model to predict class probabilities for the test set. Note that the `predict_proba()` method returns two columns, one column per class label. The first column contains the probability that an unlabeled example belongs to class `False` (`host_is_superhost` is "False") and the second column contains the probability that an unlabeled example belongs to class `True` (`host_is_superhost` is "True"). Save the values of the *second* column to a list called `proba_predictions_default`.

2. Use the `predict()` method to use the fitted model `model_default` to predict the class labels for the test set. Store the outcome in the variable `class_label_predictions_default`. Note that the `predict()` method returns the class label (True or False) per unlabeled example.

[33]:
```
# 1. Make predictions on the test data using the predict_proba() method
proba_predictions_default=model_default.predict_proba(X_test)[:,1]

# 2. Make predictions on the test data using the predict() method
class_label_predictions_default=model_default.predict(X_test)
```

Task: Evaluate the accuracy of the model using a confusion matrix. In the cell below, create a confusion matrix out of `y_test` and `class_label_predictions_default`.

First, create the confusion matrix, then create a Pandas DataFrame out of the confusion matrix for display purposes. Recall that we are predicting whether the host is a 'superhost' or not. Label the confusion matrix accordingly.

[34]:
```
cm=confusion_matrix(y_test,class_label_predictions_default)
cm_df=pd.DataFrame(cm,index=['Actual False','Actual True'],columns=['Predicted␣
 ↪False','Predicted True'])
print(cm_df)
```

```
              Predicted False  Predicted True
Actual False             1997              91
Actual True               450             265
```

1.4 Part 4: Perform Logistic Regression Model Selection Using `GridSearchSV`

Our goal is to find the optimal choice of hyperparameter C.

1.4.1 Set Up a Parameter Grid

The code cell below creates a dictionary called `param_grid` with: * a key called 'C' * a value which is a list consisting of 10 values for the hyperparameter C

It uses a scikit-learn function `l1_min_c()` to assist in the creation of possible values for C. For more information, consult the online documentation.

[35]:
```
from sklearn.svm import l1_min_c

cs = l1_min_c(X_train, y_train, loss="log") * np.logspace(0, 7, 16)
param_grid = dict(C = list(cs))
param_grid
```

[35]:
```
{'C': [0.0001537633581917429,
  0.0004503182232067712,
  0.0013188220167462046,
  0.0038623609310518637,
  0.011311482347345912,
  0.03312731129440893,
  0.09701812016301883,
  0.28413159028558327,
  0.8321204375281983,
  2.436984996480532,
  7.137062864015964,
  20.901920364088983,
  61.214295464518635,
  179.2749136895258,
  525.0325015504883,
  1537.633581917429]}
```

1.4.2 Perform Grid Search Cross-Validation

Task: Use `GridSearchCV` to search over the different values of hyperparameter C to find the one that results in the best cross-validation (CV) score.

Complete the code in the cell below.

[36]:
```
print('Running Grid Search...')

# 1. Create a LogisticRegression model object with the argument max_iter=1000.
#    Save the model object to the variable 'model'
model=LogisticRegression(max_iter=1000)

# 2. Run a grid search with 5-fold cross-validation and assign the output to␣
↪the
# object 'grid'.
param_grid={'C':[0.1,1,10,100]}
grid=GridSearchCV(estimator=model,param_grid=param_grid,cv=5)

# 3. Fit the model on the training data and assign the fitted model to the
#    variable 'grid_search'
grid_search=grid.fit(X_train,y_train)

print('Done')
```

```
Running Grid Search...
```

```
Done
```

Task: Retrieve the value of the hyperparameter C for which the best score was attained. Save the result to the variable `best_c`.

[37]:
```python
# YOUR CODE HERE
best_c=grid_search.best_params_['C']
```

1.5 Part 5: Fit and Evaluate the Optimal Logistic Regression Model

Task: Initialize a `LogisticRegression` model object with the best value of hyperparameter `C` model and fit the model to the training data. The model object should be named `model_best`. Note: Supply `max_iter=1000` as an argument when creating the model object.

[38]:
```python
# 1. Create the  model object below and assign to variable 'model_best'
model_best=LogisticRegression(C=best_c,max_iter=1000)

# 2. Fit the model to the training data below
model_best.fit(X_train,y_train)
```

[38]:
```
LogisticRegression(C=100, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=1000,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

Task: Test your model on the test set (`X_test`).

1. Use the `predict_proba()` method to use the fitted model `model_best` to predict class probabilities for the test set. Save the values of the *second* column to a list called `proba_predictions_best`.
2. Use the `predict()` method to use the fitted model `model_best` to predict the class labels for the test set. Store the outcome in the variable `class_label_predictions_best`.

[39]:
```python
# 1. Make predictions on the test data using the predict_proba() method
proba_predictions_best=model_best.predict_proba(X_test)[:,1]

# 2. Make predictions on the test data using the predict() method
class_label_predictions_best=model_best.predict(X_test)
```

Task: Evaluate the accuracy of the model using a confusion matrix. In the cell below, create a confusion matrix out of `y_test` and `class_label_predictions_best`.

[40]:
```python
# YOUR CODE HERE
cm_best=confusion_matrix(y_test,class_label_predictions_best)
cm_df_best=pd.DataFrame(cm_best,index=['Actual False','Actual␣
 ↪True'],columns=['Predicted False','Predicted True'])
print(cm_df_best)
```

```
              Predicted False  Predicted True
Actual False             1999              89
Actual True               445             270
```

1.6 Part 6: Plot Precision-Recall Curves for Both Models

Task: In the code cell below, use `precision_recall_curve()` to compute precision-recall pairs for both models.

For `model_default`: * call `precision_recall_curve()` with `y_test` and `proba_predictions_default` * save the output to the variables `precision_default`, `recall_default` and `thresholds_default`, respectively

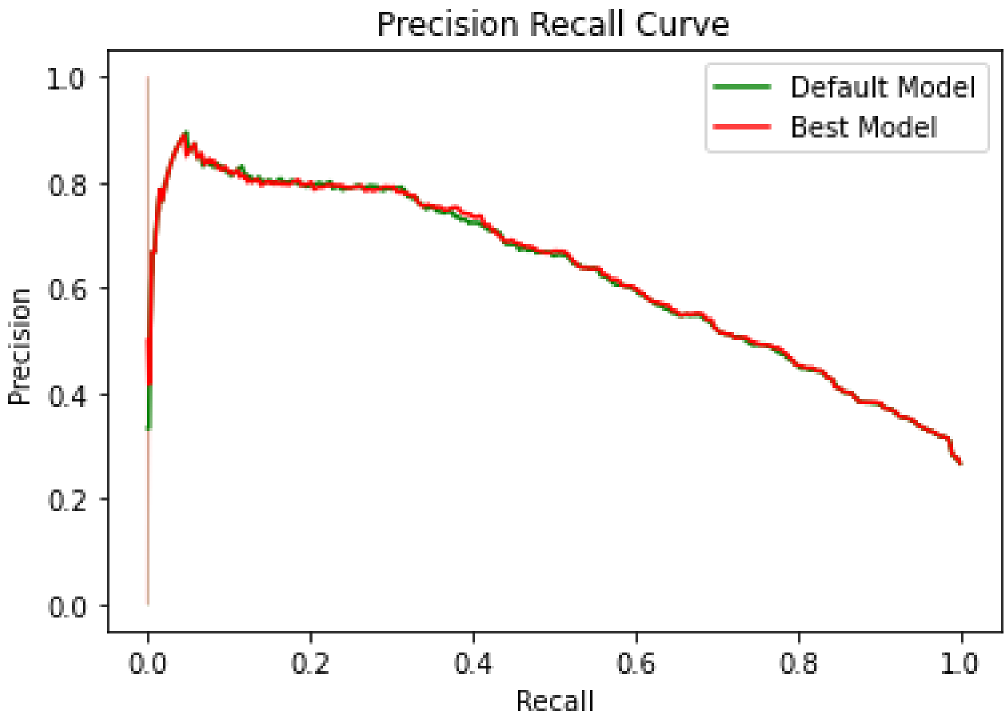
For `model_best`: * call `precision_recall_curve()` with `y_test` and `proba_predictions_best` * save the output to the variables `precision_best`, `recall_best` and `thresholds_best`, respectively

[41]:
```
precision_default, recall_default, thresholds_default =␣
 ↪precision_recall_curve(y_test,proba_predictions_default)
precision_best, recall_best, thresholds_best =␣
 ↪precision_recall_curve(y_test,proba_predictions_best)
```

In the code cell below, create two `seaborn` lineplots to visualize the precision-recall curve for both models. "Recall" will be on the x-axis and "Precision" will be on the y-axis.

The plot for "default" should be green. The plot for the "best" should be red.

[51]:
```
# YOUR CODE HERE
sns.lineplot(x=recall_default,y=precision_default,color='green',label='Default␣
 ↪Model')
sns.lineplot(x=recall_best,y=precision_best,color='red',label='Best Model')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision Recall Curve')
plt.legend()
plt.show()
```



1.7 Part 7: Plot ROC Curves and Compute the AUC for Both Models

You will next use scikit-learn's `roc_curve()` function to plot the receiver operating characteristic (ROC) curve and the `auc()` function to compute the area under the curve (AUC) for both models.

- An ROC curve plots the performance of a binary classifier for varying classification thresholds. It plots the fraction of true positives out of the positives vs. the fraction of false positives out of the negatives. For more information on how to use the `roc_curve()` function, consult the scikit-learn documentation.
- The AUC measures the trade-off between the true positive rate and false positive rate. It provides a broad view of the performance of a classifier since it evaluates the performance for all the possible threshold values; it essentially provides a value that summarizes the the ROC curve. For more information on how to use the `auc()` function, consult the scikit-learn documentation.

Let's first import the functions.

[46]:
```
from sklearn.metrics import roc_curve
from sklearn.metrics import auc
```

Task: Using the `roc_curve()` function, record the true positive and false positive rates for both models.

1. Call `roc_curve()` with arguments `y_test` and `proba_predictions_default`. The `roc_curve` function produces three outputs. Save the three items to the following variables, respectively: `fpr_default` (standing for 'false positive rate'), `tpr_default` (standing for 'true positive rate'), and `thresholds_default`.

2. Call `roc_curve()` with arguments `y_test` and `proba_predictions_best`. The `roc_curve` function produces three outputs. Save the three items to the following variables, respectively: `fpr_best` (standing for 'false positive rate'), `tpr_best` (standing for 'true positive rate'), and `thresholds_best`.

[47]:
```
fpr_default, tpr_default, thresholds_default =␣
↪fpr_default,tpr_default,thresholds_default=roc_curve(y_test,proba_predictions_default)
fpr_best, tpr_best, thresholds_best =␣
↪fpr_best,tpr_best,thresholds_best=roc_curve(y_test,proba_predictions_best)
```

Task: Create two `seaborn` lineplots to visualize the ROC curve for both models.

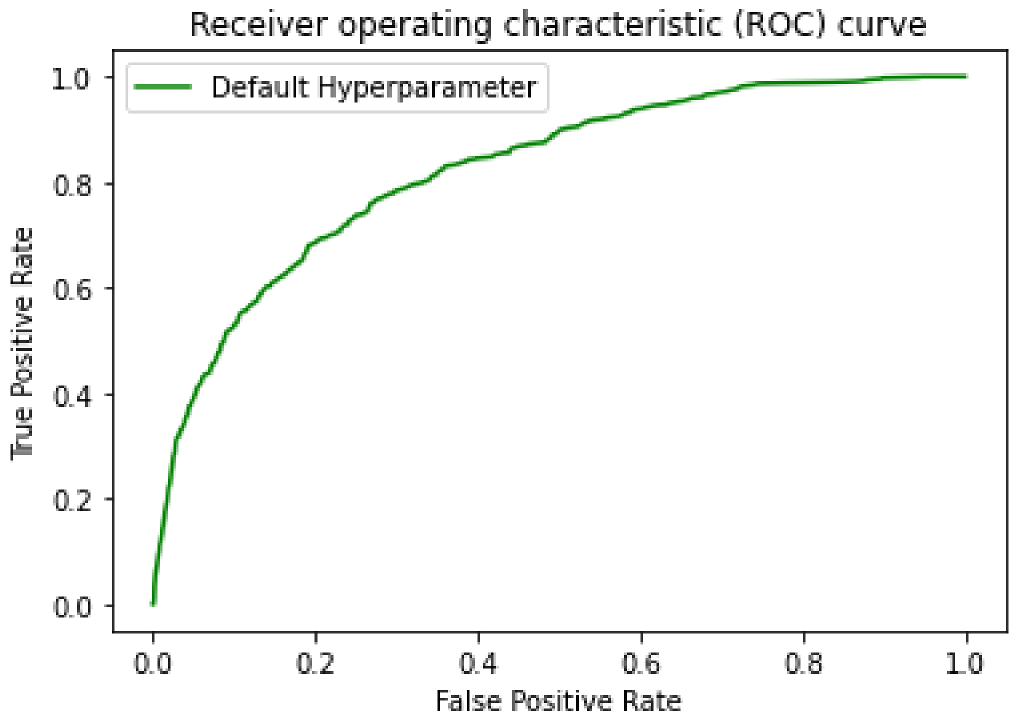
The plot for the default hyperparameter should be green. The plot for the best hyperparameter should be red.

- In each plot, the `fpr` values should be on the x-axis.
- In each plot, the`tpr` values should be on the y-axis.
- In each plot, label the x-axis "False positive rate".
- In each plot, label the y-axis "True positive rate".
- Give each plot the title "Receiver operating characteristic (ROC) curve".
- Create a legend on each plot indicating that the plot represents either the default hyperparameter value or the best hyperparameter value.

Note: It may take a few minutes to produce each plot.

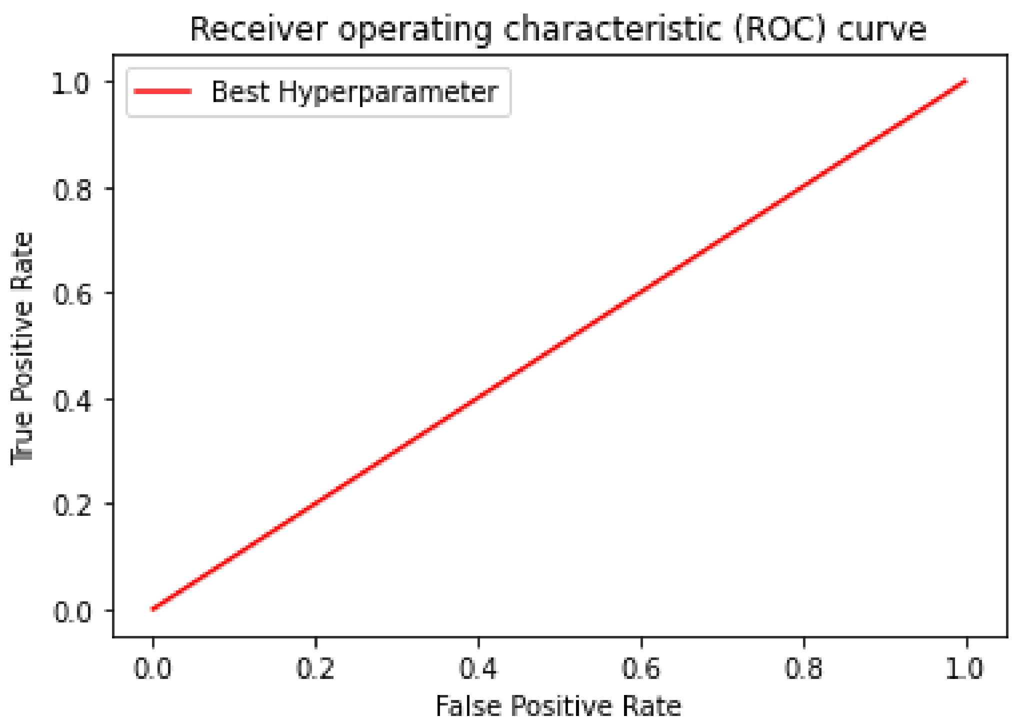
Plot ROC Curve for Default Hyperparameter:

[48]:
```
# YOUR CODE HERE
plt.figure()
sns.lineplot(x=fpr_default,y=tpr_default,color='green',label='Default␣
↪Hyperparameter')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic (ROC) curve')
plt.legend()
plt.show()
```



Plot ROC Curve for Best Hyperparameter:

[49]:
```
# YOUR CODE HERE
plt.figure()
sns.lineplot(x=tpr_best,y=tpr_best,color='red',label='Best Hyperparameter')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic (ROC) curve')
plt.legend()
plt.show()
```



Task: Use the `auc()` function to compute the area under the receiver operating characteristic (ROC) curve for both models.

For each model, call the function with the `fpr` argument first and the `tpr` argument second.

Save the result of the `auc()` function for `model_default` to the variable `auc_default`. Save the result of the `auc()` function for `model_best` to the variable `auc_best`. Compare the results.

[50]:
```
# YOUR CODE HERE
auc_default=auc(fpr_default,tpr_default)
auc_best=auc(fpr_best,tpr_best)
print(auc_default)
print(auc_best)
```

```
0.8227761701899632
0.8239470299815128
```

1.8 Deep Dive: Feature Selection Using SelectKBest

In the code cell below, you will see how to use scikit-learn's `SelectKBest` class to obtain the best features in a given data set using a specified scoring function. For more information on how to use `SelectKBest`, consult the online documentation.

We will extract the best 5 features from the Airbnb "listings" data set to create new training data, then fit our model with the optimal hyperparameter C to the data and compute the AUC. Walk through the code to see how it works and complete the steps where prompted. Analyze the results.

[62]:
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif

# Note that k=5 is specifying that we want the top 5 features
selector = SelectKBest(f_classif, k=5)
selector.fit(X, y)
filter = selector.get_support()
top_5_features = X.columns[filter]

print("Best 5 features:")
print(top_5_features)

# Create new training and test data for features
new_X_train = X_train[top_5_features]
new_X_test = X_test[top_5_features]


# Initialize a LogisticRegression model object with the best value of␣
 ↪hyperparameter C
# The model object should be named 'model'
# Note: Supply max_iter=1000 as an argument when creating the model object
# YOUR CODE HERE
model=LogisticRegression(max_iter=1000)

# Fit the model to the new training data
# YOUR CODE HERE
model.fit(new_X_train,y_train)

# Use the predict_proba() method to use your model to make predictions on the␣
 ↪new test data
# Save the values of the second column to a list called 'proba_predictions'
# YOUR CODE HERE
proba_predictions=model.predict_proba(new_X_test)[:, 1]

# Compute the auc-roc
fpr, tpr, thresholds = roc_curve(y_test, proba_predictions)
auc_result = auc(fpr, tpr)
print(auc_result)
```

```
Best 5 features:
Index(['host_response_rate', 'number_of_reviews', 'number_of_reviews_ltm',
       'number_of_reviews_l30d', 'review_scores_cleanliness'],
      dtype='object')
0.7972192079950702
```

Task: Consider the results. Change the specified number of features and re-run your code. Does this change the AUC value? What number of features results in the best AUC value? Record your findings in the cell below.

I changed the specified number of features from 5 to 7 to 3. I did this to compare increasing the number to decreasing the number. When doing so I found that increasing the number of features creates a better value. When the number was 5, the AUC score was 0.7972, at 7 it was 0.8112, and at 3 it was 0.7608. Since higher the AUC value, the better the accuracy of your model is I continued to increase the number of features to the maximum number of columns, which is 49. At 49, the AUC score is 0.8227.

[ ]:

```

```