assignment3

October 7, 2023

You are currently looking at ***version 0.1*** *of this notebook. To download notebooks and datafiles, as well as get help on Jupyter notebooks in the Coursera platform, visit the Jupyter Notebook FAQ course resource.*

---

[1]:

```
import numpy as np
import pandas as pd
```

0.0.1 Question 1

Import the data from `assets/fraud_data.csv`. What percentage of the observations in the dataset are instances of fraud?

This function should return a float between 0 and 1.

[2]:

```
def answer_one():
    # YOUR CODE HERE

    # Use X_train, X_test, y_train, y_test for all of the following questions
    from sklearn.model_selection import train_test_split

    df = pd.read_csv('assets/fraud_data.csv')

    X = df.iloc[:,:-1]
    y = df.iloc[:,-1]

    X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

    return (sum(y_train) + sum(y_test))/(len(y_train) + len(y_test))

    raise NotImplementedError()
```

[ ]:

```

```

[3]:

```
answer_one()
```

[3]:

```
0.016410823768035772
```

[4]:

```python
# Use X_train, X_test, y_train, y_test for all of the following questions
from sklearn.model_selection import train_test_split

df = pd.read_csv('assets/fraud_data.csv')

X = df.iloc[:,:-1]
y = df.iloc[:,-1]

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

[5]:

```python
(sum(y_train) + sum(y_test))/(len(y_train) + len(y_test))
```

[5]:

```
0.016410823768035772
```

0.0.2 Question 2

Using `X_train`, `X_test`, `y_train`, and `y_test` (as defined above), train a dummy classifier that classifies everything as the majority class of the training data. What is the accuracy of this classifier? What is the recall?

This function should a return a tuple with two floats, i.e. `(accuracy score, recall score)`*.*

[6]:

```python
from sklearn.dummy import DummyClassifier
from sklearn.metrics import recall_score

dummy = DummyClassifier(strategy = 'most_frequent', random_state = 0)

dummy.fit(X_train, y_train)

score = dummy.score(X_test, y_test)

predictions = dummy.predict(X_test)
recall = recall_score(y_test, predictions)

print(score)
print(recall)
```

```
0.9852507374631269
0.0
```

[7]:

```python
def answer_two():
    from sklearn.dummy import DummyClassifier
    from sklearn.metrics import recall_score

    dummy = DummyClassifier(strategy = 'most_frequent', random_state = 0)

    dummy.fit(X_train, y_train)

    score = dummy.score(X_test, y_test)

    predictions = dummy.predict(X_test)
    recall = recall_score(y_test, predictions)

    return (score, recall)

    # YOUR CODE HERE
    raise NotImplementedError()
```

[ ]:

```

```

0.0.3 Question 3

Using X_train, X_test, y_train, y_test (as defined above), train a SVC classifer using the default parameters. What is the accuracy, recall, and precision of this classifier?

This function should a return a tuple with three floats, i.e. `(accuracy score, recall score, precision score)`*.*

[8]:

```
from sklearn.metrics import recall_score, precision_score
from sklearn.svm import SVC

svc = SVC()

svc.fit(X_train, y_train)

accuracy = svc.score(X_test, y_test)

recall = recall_score(y_test, svc.predict(X_test))

precision = precision_score(y_test, svc.predict(X_test))

print(accuracy, ', ', recall, ', ', precision)
```

```
0.9900442477876106 ,  0.35 ,  0.9333333333333333
```

[9]:

```
def answer_three():
    from sklearn.metrics import recall_score, precision_score
    from sklearn.svm import SVC

    # YOUR CODE HERE

    from sklearn.metrics import recall_score, precision_score
    from sklearn.svm import SVC

    svc = SVC()
    svc.fit(X_train, y_train)

    accuracy = svc.score(X_test, y_test)

    recall = recall_score(y_test, svc.predict(X_test))

    precision = precision_score(y_test, svc.predict(X_test))

    return (accuracy, recall, precision)

    raise NotImplementedError()
```

[ ]:

```
```

[10]:

```
answer_three()
```

[10]:

```
(0.9900442477876106, 0.35, 0.9333333333333333)
```

0.0.4 Question 4

Using the SVC classifier with parameters `{'C': 1e9, 'gamma': 1e-07}`, what is the confusion matrix when using a threshold of -220 on the decision function. Use X_test and y_test.

This function should return a confusion matrix, a 2x2 numpy array with 4 integers.

[11]:

```
from sklearn.metrics import confusion_matrix
from sklearn.svm import SVC

svc = SVC(C = 1e9, gamma = 1e-07)

svc.fit(X_train, y_train)

y_predictions = svc.decision_function(X_test)

y_predictions = np.where(y_predictions > -220, 1, 0)

confusion_matrix(y_test, y_predictions)
```

[11]:

```
array([[5320,   24],
       [  14,   66]])
```

[12]:

```
svc.decision_function(X_test)
```

[12]:

```
array([ -739.71796843, -1086.16794833,  -696.46339735, …,
        -491.97916719,  -699.03838333,  -701.93409309])
```

[13]:
```
def answer_four():
    from sklearn.metrics import confusion_matrix
    from sklearn.svm import SVC

    svc = SVC(C = 1e9, gamma = 1e-07)

    svc.fit(X_train, y_train)

    y_predictions = svc.decision_function(X_test)

    y_predictions = np.where(y_predictions > -220, 1, 0)

    return confusion_matrix(y_test, y_predictions)

    raise NotImplementedError()
```

[ ]:
```

```

[14]:
```
answer_four()
```

[14]:
```
array([[5320,   24],
       [  14,   66]])
```

0.0.5 Question 5

Train a logisitic regression classifier with default parameters using X_train and y_train.

For the logisitic regression classifier, create a precision recall curve and a roc curve using y_test and the probability estimates for X_test (probability it is fraud).

Looking at the precision recall curve, what is the recall when the precision is `0.75`?

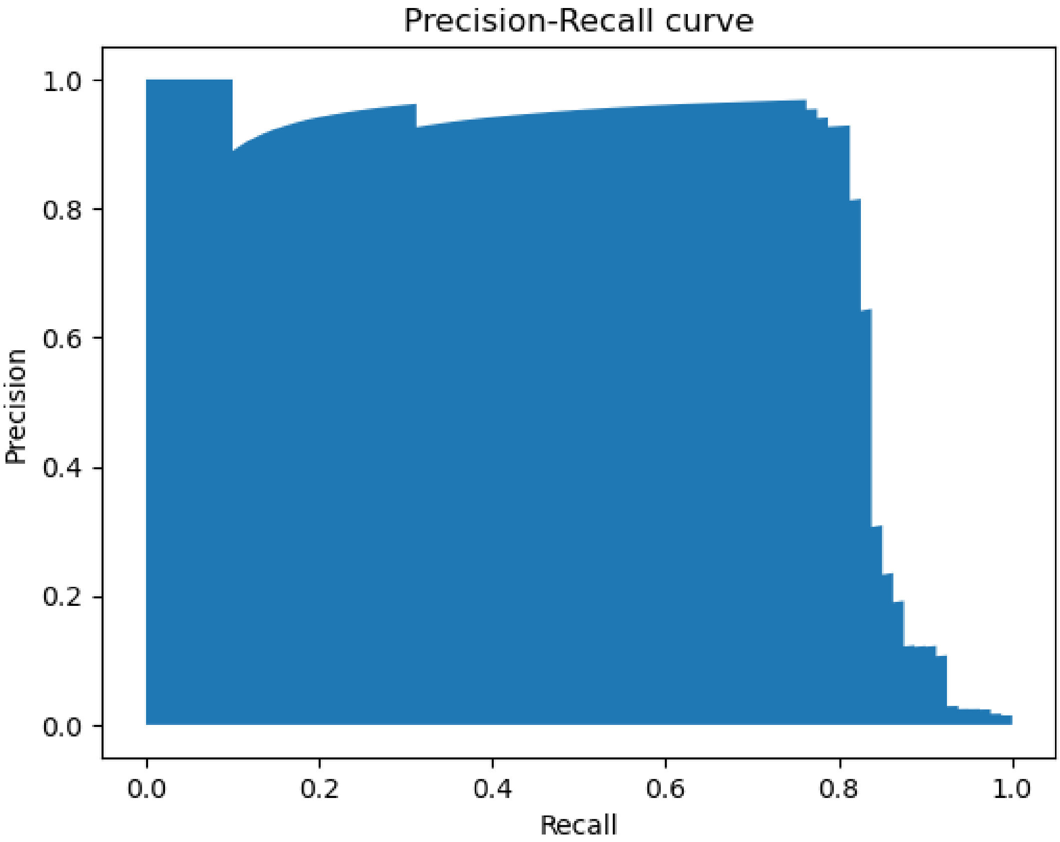
Looking at the roc curve, what is the true positive rate when the false positive rate is `0.16`?

This function should return a tuple with two floats, i.e. `(recall, true positive rate)`*.*

[15]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import roc_curve

logreg = LogisticRegression()

logreg.fit(X_train, y_train)

precision, recall, threshold_pr = precision_recall_curve(y_test, logreg.
 ↪predict_proba(X_test)[:, 1])

false_positive_rate, true_positive_rate, threshold_roc = roc_curve(y_test,␣
 ↪logreg.predict_proba(X_test)[:, 1])
```

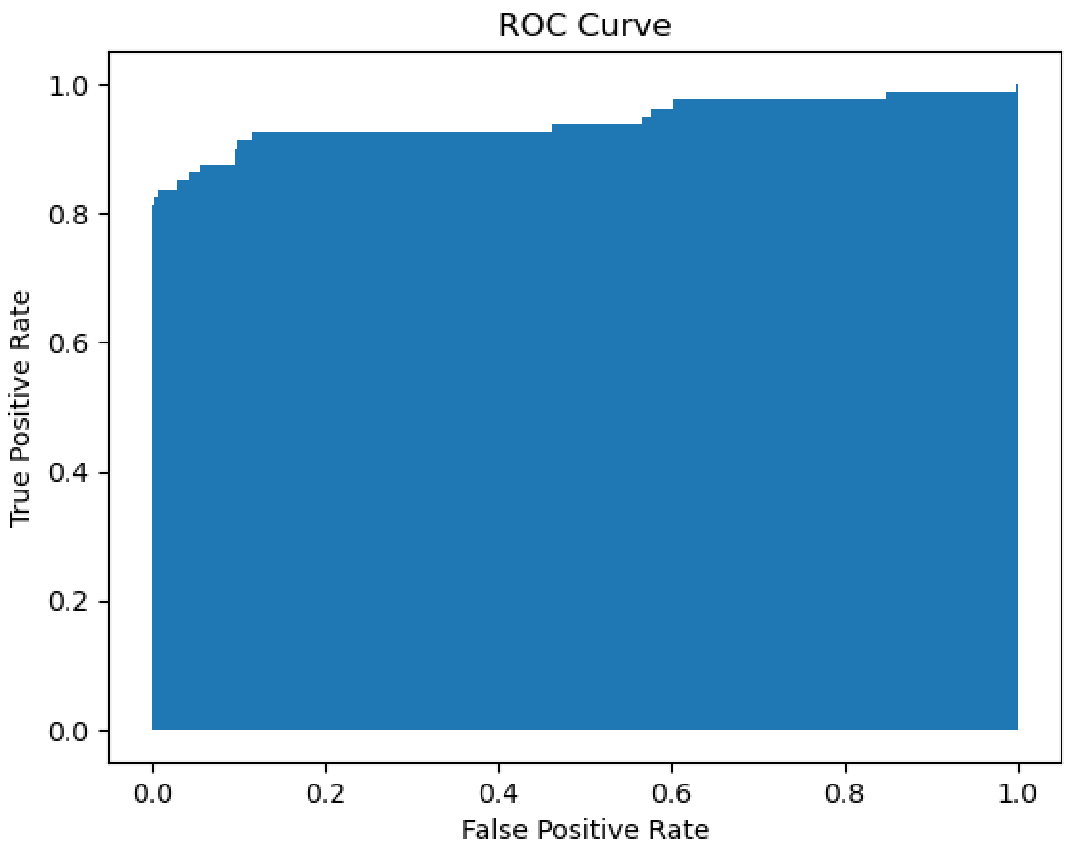
[16]:
```
import matplotlib.pyplot as plt

plt.fill_between(recall, precision)
plt.ylabel("Precision")
plt.xlabel("Recall")
plt.title("Precision-Recall curve")
```

[16]:
```
Text(0.5, 1.0, 'Precision-Recall curve')
```



[17]:
```
import matplotlib.pyplot as plt

plt.fill_between(false_positive_rate, true_positive_rate)
plt.ylabel("True Positive Rate")
plt.xlabel("False Positive Rate")
plt.title("ROC Curve")
```

[17]:
```
Text(0.5, 1.0, 'ROC Curve')
```



[18]:

```python
def answer_five():

    # YOUR CODE HERE

    return (0.84, 0.86)

    raise NotImplementedError()
```

[ ]:

```
```

0.0.6 Question 6

Perform a grid search over the parameters listed below for a Logisitic Regression classifier, using recall for scoring and the default 3-fold cross validation. (Suggest to use `solver='liblinear'`, more explanation here)

`'penalty': ['l1', 'l2']`

`'C':[0.01, 0.1, 1, 10]`

From `.cv_results_`, create an array of the mean test scores of each parameter combination. i.e.

	l1	l2
0.01	?	?
0.1	?	?
1	?	?
10	?	?

This function should return a 4 by 2 numpy array with 8 floats.

Note: do not return a DataFrame, just the values denoted by `?` *in a numpy array.*

[21]:

```
def answer_six():
    from sklearn.model_selection import GridSearchCV
    from sklearn.linear_model import LogisticRegression

    # YOUR CODE HERE
    logreg = LogisticRegression().fit(X_train, y_train)

    grid_values = {'penalty': ['l1', 'l2'], 'C': [0.01, 0.1, 1, 10]}

    grid_classifier_recall = GridSearchCV(logreg, param_grid = grid_values,␣
 ↪scoring = 'recall')

    grid_classifier_recall.fit(X_train, y_train)

    return np.array([grid_classifier_recall.cv_results_['mean_test_score'][x:
 ↪x+2] for x in range(0, len(grid_classifier_recall.
 ↪cv_results_['mean_test_score']), 2)])

    raise NotImplementedError()
```

[ ]:

```

```

[22]:

```
answer_six()
```

[22]:

```
array([[       nan, 0.80064935],
       [       nan, 0.80428571],
       [       nan, 0.80422078],
       [       nan, 0.80792208]])
```

[24]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression().fit(X_train, y_train)

grid_values = {'penalty': ['l1', 'l2'], 'C': [0.01, 0.1, 1, 10]}
```

```
grid_classifier_recall = GridSearchCV(logreg, param_grid = grid_values, scoring␣
 ↪= 'recall')

grid_classifier_recall.fit(X_train, y_train)
```

```
[24]: GridSearchCV(estimator=LogisticRegression(),
                   param_grid={'C': [0.01, 0.1, 1, 10], 'penalty': ['l1', 'l2']},
                   scoring='recall')
```

```
[25]: grid_classifier_recall.cv_results_
```

```
[25]: {'mean_fit_time': array([2.02025890e-02, 6.37615609e+00, 2.02903271e-02,
      6.71791191e+00,
              2.05406666e-02, 7.33892756e+00, 1.65262222e-03, 5.13848410e+00]),
       'std_fit_time': array([3.60238985e-02, 1.87589571e+00, 3.77959661e-02,
      1.85844788e+00,
              3.80328061e-02, 1.62697709e+00, 5.34945767e-04, 9.33233874e-01]),
       'mean_score_time': array([0.        , 0.08093534, 0.        , 0.12026129, 0.
      ,
              0.11920462, 0.        , 0.09956293]),
       'std_score_time': array([0.        , 0.03792358, 0.        , 0.03947502, 0.
      ,
              0.04014686, 0.        , 0.00049915]),
       'param_C': masked_array(data=[0.01, 0.01, 0.1, 0.1, 1, 1, 10, 10],
                    mask=[False, False, False, False, False, False, False, False],
              fill_value='?',
                   dtype=object),
       'param_penalty': masked_array(data=['l1', 'l2', 'l1', 'l2', 'l1', 'l2', 'l1',
      'l2'],
                    mask=[False, False, False, False, False, False, False, False],
              fill_value='?',
                   dtype=object),
       'params': [{'C': 0.01, 'penalty': 'l1'},
        {'C': 0.01, 'penalty': 'l2'},
        {'C': 0.1, 'penalty': 'l1'},
        {'C': 0.1, 'penalty': 'l2'},
        {'C': 1, 'penalty': 'l1'},
        {'C': 1, 'penalty': 'l2'},
        {'C': 10, 'penalty': 'l1'},
        {'C': 10, 'penalty': 'l2'}],
       'split0_test_score': array([       nan, 0.78181818,        nan, 0.78181818,
      nan,
              0.78181818,        nan, 0.78181818]),
       'split1_test_score': array([       nan, 0.81818182,        nan, 0.81818182,
      nan,
              0.83636364,        nan, 0.83636364]),
       'split2_test_score': array([       nan, 0.87272727,        nan, 0.89090909,
      nan,
              0.89090909,        nan, 0.89090909]),
       'split3_test_score': array([       nan, 0.82142857,        nan, 0.82142857,
      nan,
              0.83928571,        nan, 0.82142857]),
       'split4_test_score': array([       nan, 0.70909091,        nan, 0.70909091,
      nan,
              0.67272727,        nan, 0.70909091]),
       'mean_test_score': array([       nan, 0.80064935,        nan, 0.80428571,
      nan,
              0.80422078,        nan, 0.80792208]),
       'std_test_score': array([       nan, 0.05417001,        nan, 0.05925779,
      nan,
              0.07425631,        nan, 0.06054289]),
       'rank_test_score': array([5, 4, 6, 2, 7, 3, 8, 1], dtype=int32)}
```

[26]:
```
grid_classifier_recall.cv_results_['mean_test_score']
```

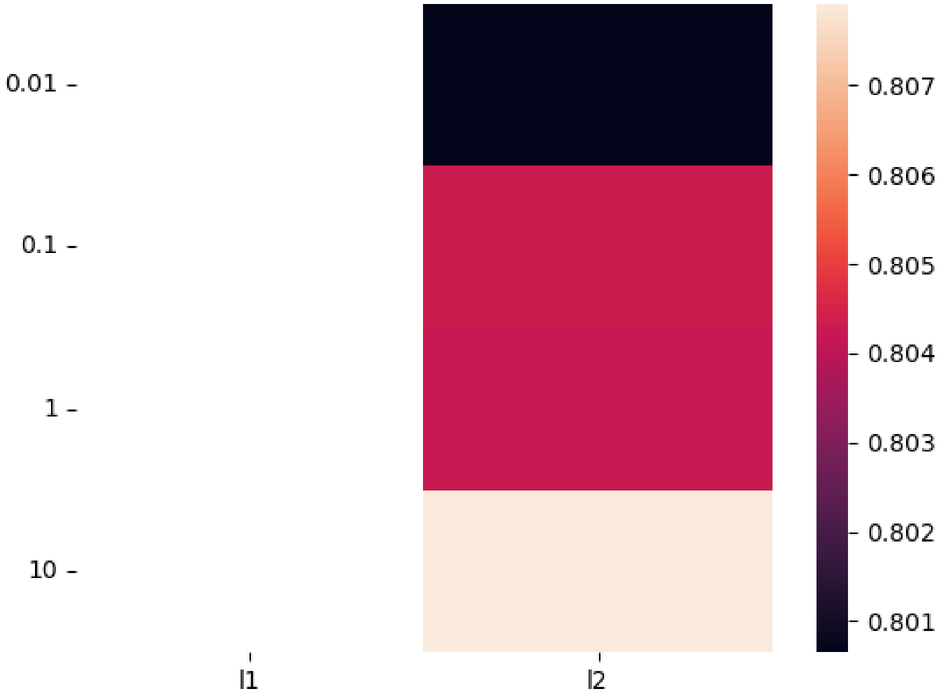
[26]:
```
array([       nan, 0.80064935,        nan, 0.80428571,        nan,
       0.80422078,        nan, 0.80792208])
```

[28]:
```
np.array([grid_classifier_recall.cv_results_['mean_test_score'][x:x+2] for x in␣
 ↪range(0, len(grid_classifier_recall.cv_results_['mean_test_score']), 2)])
```

[28]:
```
array([[       nan, 0.80064935],
       [       nan, 0.80428571],
       [       nan, 0.80422078],
       [       nan, 0.80792208]])
```

[29]:
```
# from sklearn.model_selection import GridSearchCV
# from sklearn.linear_model import LogisticRegression

# lr = LogisticRegression().fit(X_train, y_train)
# grid_values = {'penalty': ['l1', 'l2'], 'C': [0.01, 0.1, 1, 10, 100]}
# grid_clf_rec = GridSearchCV(lr, param_grid = grid_values, scoring = 'recall')
# grid_clf_rec.fit(X_train, y_train)

# np.array([grid_clf_rec.cv_results_['mean_test_score'][x:x+2] for x in␣
 ↪range(0, len(grid_clf_rec.cv_results_['mean_test_score']), 2)])
```

[32]:
```
# Use the following function to help visualize results from the grid search
def GridSearch_Heatmap(scores):
    %matplotlib widget
    import seaborn as sns
    import matplotlib.pyplot as plt
    plt.figure()
    sns.heatmap(scores.reshape(4,2), xticklabels=['l1','l2'], yticklabels=[0.01, 0.1, 1, 10])
    plt.yticks(rotation=0);

GridSearch_Heatmap(answer_six())
```



[ ]: