Assignment 3

April 15, 2021

---

*You are currently looking at **version 1.2** 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 Assignment 3 - Evaluation

In this assignment you will train several models and evaluate how effectively they predict instances of fraud using data based on this dataset from Kaggle. Each row in `fraud_data.csv` corresponds to a credit card transaction. Features include confidential variables `V1` through `V28` as well as `Amount` which is the amount of the transaction. The target is stored in the `class` column, where a value of 1 corresponds to an instance of fraud and 0 corresponds to an instance of not fraud.

```
In [2]: import numpy as np
        import pandas as pd
```

1.0.1 Question 1

Import the data from `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.

```
In [3]: #df = pd.read_csv('readonly/fraud_data.csv')
        #print(len(df))
        #df_1 = df[df['Class'] == 1]
        #print(len(df_1))
        def answer_one():

            # Your code here
            df = pd.read_csv('fraud_data.csv')
            df_1 = df[df['Class'] == 1]
            percentage = float(len(df_1)/len(df))


            return percentage # Return your answer
        #answer_one()
```

```
In [4]: # 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('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)
```

1.0.2 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)`*.*

```
In [5]: def answer_two():
            from sklearn.dummy import DummyClassifier
            from sklearn.metrics import recall_score, accuracy_score

            # Your code here
            # Negative class (0) is most frequent
            dummy_majority = DummyClassifier(strategy = 'most_frequent').fit(X_trai
            # Therefore the dummy 'most_frequent' classifier always predicts class
            y_dummy_predictions = dummy_majority.predict(X_test)

            #y_dummy_predictions
            #y_score = dummy_majority.decision_function(X_test)
            dummy_accuracy_score = accuracy_score(y_test, y_dummy_predictions)
            dummy_recall_score = recall_score(y_test, y_dummy_predictions)

            return dummy_accuracy_score,dummy_recall_score# Return your answer
        #answer_two()
```

1.0.3 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)`*.*

```
In [6]: def answer_three():
            from sklearn.metrics import recall_score, precision_score, accuracy_sco
            from sklearn.svm import SVC

            # Your code here
            svm = SVC().fit(X_train, y_train)
            y_svm_predictions = svm.predict(X_test)
            svm_accuracy_score = accuracy_score(y_test, y_svm_predictions)
            svm_recall_score = recall_score(y_test, y_svm_predictions)
            svm_precision_score = precision_score(y_test, y_svm_predictions)
            #svm.score(X_test, y_test)

            return svm_accuracy_score, svm_recall_score, svm_precision_score# Retur
        #answer_three()
```

1.0.4 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.

```
In [7]: def answer_four():
            from sklearn.metrics import confusion_matrix, precision_recall_curve
            from sklearn.svm import SVC

            # Your code here
            svm = SVC(C = 1000000000, gamma =  0.0000001).fit(X_train, y_train)
            #svm_predicted = svm.predict(X_test)
            threshold = -220
            #fpr, tpr, thresholds = roc_curve(y_true, y_pred)
            #res = pd.DataFrame({'FPR': fpr, 'TPR': tpr, 'Threshold': thresholds})
            #res = res[res['Threshold'] == -220]
            #svm_predicted = (svm.predict_proba(X_test)[:, 1] > threshold).astype(
            svm_predicted = (svm.decision_function(X_test) > threshold).astype('flo
            #average_precision = average_precision_score(y_test, y_score)
            #print(average_precision)
            #precision, recall, thresholds = precision_recall_curve(y_test, y_score
            confusion = confusion_matrix(y_test, svm_predicted)

            return confusion# Return your answer
        #answer_four()
```

1.0.5 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)`.

```
In [8]: def answer_five():

            # Your code here
            from sklearn.metrics import roc_curve
```

```
        from sklearn.linear_model import LogisticRegression

        lr = LogisticRegression().fit(X_train, y_train)
        lr_predicted = lr.predict_proba(X_test)[:,1]
        from sklearn.metrics import precision_recall_curve
        y_score_lr = lr.fit(X_train, y_train).decision_function(X_test)
        fpr_lr, tpr_lr, _ = roc_curve(y_test, y_score_lr)

        precision, recall, thresholds = precision_recall_curve(y_test, lr_predi
        a = None
        b = None
        #predict = max(precision, key=lambda x:abs(0.75))
        for prec, rec, in zip(precision, recall):
            #print(prec, rec)
            if prec == 0.75:
                a = rec
                #print(a)

        for fpr, tpr, in zip(fpr_lr, tpr_lr):
            #print(fpr, tpr)
            if fpr > 0.15 and fpr < 0.2:
                b = tpr
                #print(b)

        return a,b# Return your answer
    #answer_five()
```

1.0.6 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.

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

`'C':[0.01, 0.1, 1, 10, 100]`

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	?	?
100	?	?

This function should return a 5 by 2 numpy array with 10 floats.

Note: do not return a DataFrame, just the values denoted by '?' above in a numpy array. You might need to reshape your raw result to meet the format we are looking for.

```
In [60]: def answer_six():
        from sklearn.model_selection import GridSearchCV
        from sklearn.linear_model import LogisticRegression


        lr = LogisticRegression().fit(X_train, y_train)
        param_grid = {'penalty': ['l1', 'l2'],'C':[0.01, 0.1, 1, 10, 100]}
        #folds = 3

        # default metric to optimize over grid parameters: accuracy
        grid_lr_recall = GridSearchCV(lr, param_grid = param_grid, scoring = '
        grid_lr_recall.fit(X_train, y_train)
        y_decision_fn_scores_recall = grid_lr_recall.decision_function(X_test)
        results = grid_lr_recall.cv_results_
        df = pd.DataFrame(results)
        df = df[['mean_test_score','param_C','param_penalty']]
        df_L1 = df[df['param_penalty']=='l1']
        df_L1 = df_L1.set_index('param_C')
        #df_L1 = df_L1['mean_test_score']
        #df_L1 = df_L1.rename(columns={'mean_test_score':'l1'})

        df_L2 = df[df['param_penalty']=='l2']
        df_L2 = df_L2.set_index('param_C')
        #df_L2 = df_L2['mean_test_score']
        #df_L2 = df_L2['mean_test_score']
        #df_L2 = df_L2.rename({'mean_test_score':'l2'})
        #df = T.df
        df_L1['l2'] = df_L2['mean_test_score']
        df_L1 = df_L1[['mean_test_score','l2']]
        df_L1 = df_L1.rename(columns={'mean_test_score':'l1'})
        print(df_L1)
        my_list1 = df_L1['l1']
        my_list2 = df_L1['l2']
        #my_list = np.array([my_list1, my_list2])
        #my_list = my_list.reshape(5,2)
        my_list = np.dstack((my_list1,my_list2)).reshape(5,2)

        #print('Grid best parameter (max. accuracy): ', grid_lr_recall.best_pa
        #print('Grid best score (accuracy): ', grid_lr_recall.best_score_)


        return my_list# Return your answer
    #answer_six()
```

```
               l1        l2
param_C
0.01     0.666667  0.760870
0.10     0.800725  0.804348
1.00     0.811594  0.811594
```

```
10.00    0.807971  0.811594
100.00   0.807971  0.807971
```

```
Out[60]: array([[ 0.66666667,  0.76086957],
                [ 0.80072464,  0.80434783],
                [ 0.8115942 ,  0.8115942 ],
                [ 0.80797101,  0.8115942 ],
                [ 0.80797101,  0.80797101]])
```

```
In [61]: # Use the following function to help visualize results from the grid searc
         #scores=answer_six()
         def GridSearch_Heatmap(scores):
             %matplotlib notebook
             import seaborn as sns
             import matplotlib.pyplot as plt
             plt.figure()
             sns.heatmap(scores.reshape(5,2), xticklabels=['l1','l2'], yticklabels=
             plt.yticks(rotation=0);

         GridSearch_Heatmap(answer_six())
```

```
               l1        l2
param_C
0.01     0.666667  0.760870
0.10     0.800725  0.804348
1.00     0.811594  0.811594
10.00    0.807971  0.811594
100.00   0.807971  0.807971


<IPython.core.display.Javascript object>


<IPython.core.display.HTML object>
```

```
In [ ]:
```