Lab3

November 23, 2021

1 Lab 3: ROC Analysis

1.1 Lena Feiler - i6246119

[1]:
```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
import matplotlib.pyplot as plt
```

[2]:
```python
class ROC:

    # parameterized constructor
    def __init__(self, probs, trueClass):
        #Probs contains the estimated probabilities of the given test instances␣
↪of the negative and positive classes
        self.Probs = probs
        #TrueClass contains the classification of the test instances
        self.TrueClass = trueClass
        self.ROC_coordinates = None


    # exercise B
    # sorting_strategy can be 'expected', 'pessimistic' or 'optimistic'
    # it defines the sorting strategy that is to be applied
    def compute_ROC_coordinates(self, sorting_strategy):
        val = pd.concat([self.Probs, self.TrueClass], axis=1)
        pessimistic = False
        if sorting_strategy == 'pessimistic' :
            pessimistic = True
        sorted_val = val.sort_values(by = ['positive', 'class'], ascending =␣
↪(False, pessimistic))

        # total number of negative and positive classifications in data
        occurence = sorted_val['class'].value_counts()
        N = occurence['tested_negative']
        P = occurence['tested_positive']

        FP = 0
```

```python
        TP = 0
        ROC_coord = {'TP_rate':[], 'FP_rate':[]}
        prev_prob = float('-inf')

        for index, instance in sorted_val.iterrows():
            if(instance[1] != prev_prob) or sorting_strategy != 'expected':
                ROC_coord['TP_rate'].append(TP/P)
                ROC_coord['FP_rate'].append(FP/N)
                prev_prob = instance[1]
            if(instance['class'] == 'tested_positive'):
                TP = TP+1
            else:
                FP = FP+1

        ROC_coord['TP_rate'].append(TP/P)
        ROC_coord['FP_rate'].append(FP/N)
        self.ROC_coordinates = pd.DataFrame(ROC_coord)
        return self.ROC_coordinates


    # exercise C
    # assumes that compute_ROC_coordinates was run
    def plot_ROC(self, ROC_coordinates):
        plt.plot(ROC_coordinates.loc[:,'FP_rate'], ROC_coordinates.loc[:,'TP_rate' ])
        plt.title('ROC curve')
        plt.xlabel('FP Rate')
        plt.ylabel('TP Rate')

    # exercise D
    # computes the area under the ROC curve
    # assumes that compute_ROC_coordinates was run
    # used the provided pdeudocode, but adjusted it to take the computed coordinates
    def compute_AUCROC(self, ROC_coordinates):
        area_ROC = 0
        FP_prev = TP_prev = 0
        FP = TP = 0
        for i in range(0, len(ROC_coordinates)):
            area_ROC = area_ROC + self.trapezoid_area(FP, FP_prev, TP, TP_prev)
            FP_prev = FP
            TP_prev = TP
            FP = ROC_coordinates['FP_rate'][i]
            TP = ROC_coordinates['TP_rate'][i]
        area_ROC = area_ROC + self.trapezoid_area(FP, FP_prev, TP, TP_prev)
        return area_ROC
```

```
    def trapezoid_area(self, FP, FP_prev, TP, TP_prev):
        base = abs(FP - FP_prev)
        height = (TP + TP_prev)/2
        return (base * height)
```

1.2 Data

We use the knn classifier from sklearn (KNeighborsClassifier()) to classify the data in the given diabetis dataset. By default, the number of neighbours is set to 3.

[3]:
```
data = pd.read_csv('diabetes.csv')
Y = data['class']
X = data.drop(['class'],axis=1)

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.34,␣
 ↪random_state=10)

# Create KNN classifier
knn = KNeighborsClassifier() # n assigned by default
# Fit the classifier to the data
knn.fit(X_train,Y_train)

# prediciton
probs = knn.predict_proba(X_test)
probs_df = pd.DataFrame(probs, columns = ['negative', 'positive'])
tv_df = Y_test.to_frame().reset_index(drop=True)
```

1.3 Answer Task B:

The given pseudocode is used to implement the "expected" strategy for generating the ROC curves. In this strategy, we only sort by the given probabilities. I have slightly modified the method, so that it takes and argument sorting_strategy. This argument can take the values 'expected' (as in the pseudocode), 'pessimistic' or 'optimistic'. For the last two strategies, I slightly adjusted the sorting method, such that the true_class is sorted, additionally to the probability which is sorted in descending order. In the case of the 'optimistic' strategy, we first out tge "tested_positive" values and in case of the 'pessimitic' strategy we first put the "tested_negative" values. For the 'expected' strategy, only the coordinates of the points where the probability differs to the previous point is added, while in the 'optimistic' and 'pessimistic' strategies, every point in the data is added to the coordinates.

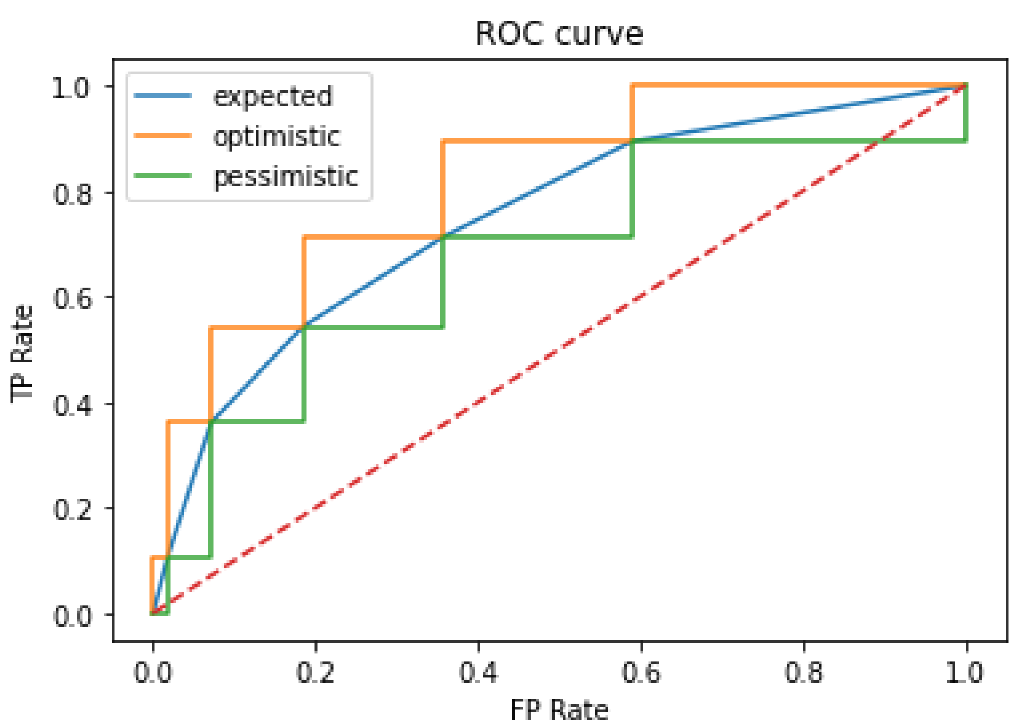
[4]:
```
roc = ROC(probs_df, tv_df)
coordinates = roc.compute_ROC_coordinates('expected')
coord_optimistic = roc.compute_ROC_coordinates('optimistic')
coord_pessimistic = roc.compute_ROC_coordinates('pessimistic')
```

1.4 Task C: plot the ROC curve

The method assumes that the compute_ROC_coordinates method has already been run. In the plot, we can see that the optimistic is higher than the average, while the pessimitic one is lower. Furthermore, the pessimistic strategy goes below the value of the red dashed line. This means that for some points, this means that the amount of correctly classified diabetis patients is smaller than the incorrectly classified amount.

[5]:
```
# plot
roc.plot_ROC(coordinates)
roc.plot_ROC(coord_optimistic)
roc.plot_ROC(coord_pessimistic)
plt.legend(['expected', 'optimistic', 'pessimistic'])
x= [0,1]
y= [0,1]
plt.plot(x,y, '--')
```

[5]:
```
[<matplotlib.lines.Line2D at 0x7fc848cca430>]
```



1.5 Task D: compute the area under the ROC curve

The method assumes that the compute_ROC_coordinates method has already been run. As expected, the 'optimistic' strategy has the largest area under the graph, while the 'pessimistic' one has the smallest and the 'expected area is in between the other two.

[7]:
```
area = roc.compute_AUCROC(coordinates)
area_pess = roc.compute_AUCROC(coord_pessimistic)
area_opt = roc.compute_AUCROC(coord_optimistic)
print('Area optimistic strategy:',area_opt)
print('Area expected strategy:', area)
print('Area pessimistic strategy:', area_pess)
```

```
Area optimistic strategy: 0.8238348530901722
Area expected strategy: 0.748290273556231
Area pessimistic strategy: 0.6727456940222909
```

[ ]:
```

```

[ ]:
```

```