Lab 3

1 Lab3 - ROC Curves

1.1 The ROC class

The assignment describes a problem that comes up when creating the ROC curve: What do we do if we find a number of instances for all of which the classifier predicted the same probability, but the true classes of the instances disagree. In the pseudocode given in the assignment, such instances are handled in a bulk. Since their probabilities are the same, only for the first instance a coordinate is created, the other instances are skipped and the next coordinate is only created once the probability changes. Of course FP and TP are still updated for those instances. This way of handling the problem aligns very well with the intuition behind the ROC curve, that we slide the probability threshold above which an instance will be classified positive. The classifier can not tell the instances apart, it only has the threshold. Since this solution already seems perfectly reasonable to me and I don't see any further problem, my solution to this problem is the one already given in the assignment sheet.

[2]:
```
import math
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_auc_score
from matplotlib import pyplot
```

[3]:
```
class ROC:
    def __init__(self, probs, true_class):
    # probabilities: A DataFrame of one column or a Series, giving the␣
↪predicted probabilities
    #     for an instance belonging to the positive class
    # true_class: DataFrame of one column or a Series of booleans giving␣
↪whether an instance truly belongs
    #    to the positive class (True) or not (False)
        # create new indices in case the passed data frames have incompatible␣
↪ones
        self.probs = probs.reset_index(drop=True)
        self.true_class = true_class.reset_index(drop=True)
        self.ROC_coordinates = None
```

```
        self.ROC_convex_hull = None

    def compute_ROC_coordinates(self):
        neg_count = self.true_class.value_counts()[False]
        pos_count = self.true_class.value_counts()[True]
        # put everything in one data frame to make handling it easier
        # Note that we created new indices in the constructor
        merged = pd.merge(self.probs, self.true_class, left_index=True,␣
↪right_index=True)
        # first column of merged is probs
        merged = merged.sort_values(merged.columns[0], ascending=False)
        false_positives = 0
        true_positives = 0
        ROC_coordinates = {'fpr': [], 'tpr': []}
        previous_probability = -math.inf
        for entry in merged.iloc:
            # entry[0] is predicted probability, entry[1] is the true class
            if entry[0] != previous_probability:
                ROC_coordinates['fpr'].append(false_positives/neg_count)
                ROC_coordinates['tpr'].append(true_positives/pos_count)
                previous_probability = entry[0]
            # update how many positive/negative instances we saw so far
            if entry[1]:
                true_positives += 1
            else:
                false_positives += 1
        ROC_coordinates['fpr'].append(false_positives/neg_count)
        ROC_coordinates['tpr'].append(true_positives/pos_count)
        ROC_coordinates = pd.DataFrame(ROC_coordinates)
        self.ROC_coordinates = ROC_coordinates
        return ROC_coordinates

    def compute_AUCROC(self):
        if self.ROC_coordinates is None:
            self.compute_ROC_coordinates()
        # to compute the area under the curve (AUC) we want to multiply the
        # y values (tpr) with the length of the x value range that has this␣
↪value
        # the value ranges on the x axis are: x value at i - x value at i - 1
        # for the first x value the previous x value is 0
        space = self.ROC_coordinates['fpr'] - np.concatenate([[0], self.
↪ROC_coordinates['fpr'].to_numpy()])[:-1]
        areas = self.ROC_coordinates['tpr'] * space
        return areas.sum()

    def compute_ROC_convex_hull_coordinates(self):
        if self.ROC_coordinates is None:
```

```
            self.compute_ROC_coordinates()
        convex_hull = {'fpr': [], 'tpr': []}
        coordinates = self.ROC_coordinates
        for c in self.ROC_coordinates.iloc:
            dominates = (coordinates['tpr'] >= c['tpr']) \
                        & (coordinates['fpr'] <= c['fpr']) \
                        & ~((coordinates['tpr'] == c['tpr']) &␣
↪(coordinates['fpr'] == c['fpr']))
            if not dominates.any():
                convex_hull['fpr'].append(c['fpr'])
                convex_hull['tpr'].append(c['tpr'])
        self.ROC_convex_hull = convex_hull
        return self.ROC_convex_hull

    def plot_ROC(self):
        if self.ROC_coordinates is None:
            self.compute_ROC_coordinates()
        # plot the comparison diagonal first
        pyplot.plot([0,1], [0,1], 'lightgray')
        pyplot.plot(self.ROC_coordinates['fpr'], self.ROC_coordinates['tpr'])
        if self.ROC_convex_hull is not None:
            pyplot.plot(self.ROC_convex_hull['fpr'], self.
↪ROC_convex_hull['tpr'])
        pyplot.legend(['Baseline', 'Classifiers ROC', 'ROC Convex Hull'])
        pyplot.xlim([0,1])
        pyplot.xlabel('False Positive rate')
        pyplot.ylim([0,1])
        pyplot.ylabel('True Positive rate')
        pyplot.title('ROC Curve')
```

1.2 Testing the ROC class

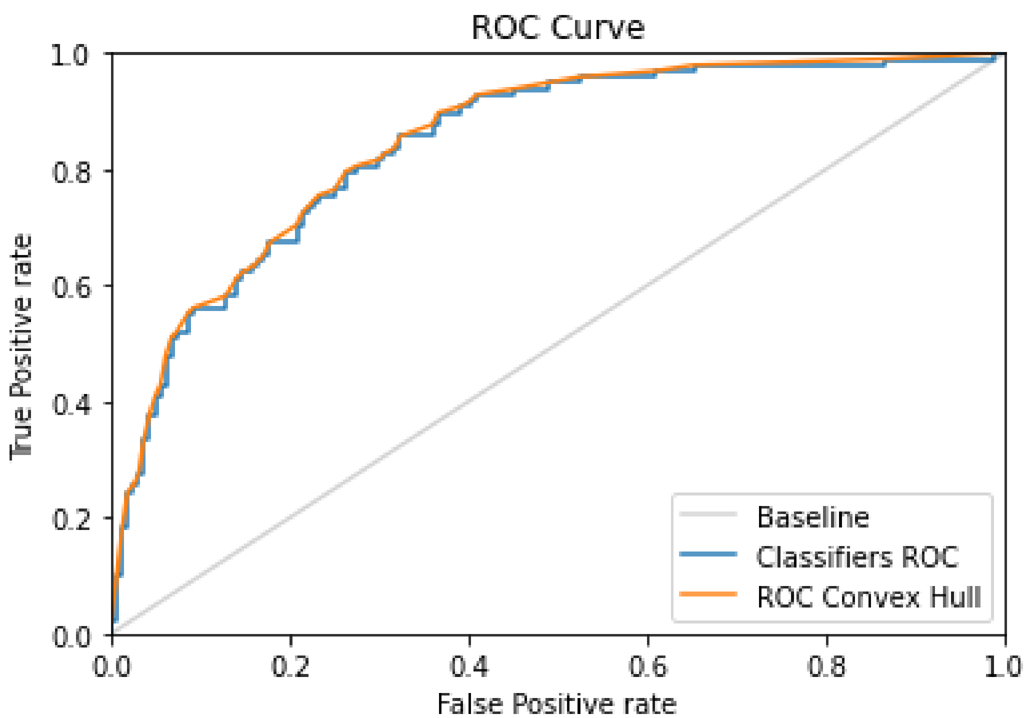
To test the ROC curves I compare two classifiers from sklearn: LogisticRegression and LinearSVC

[4]:
```
diabetes = pd.read_csv("diabetes.csv")
diabetes_Y = diabetes['class']
diabetes_X = diabetes.drop(['class'], axis=1)
X_train, X_test, Y_train, Y_test = train_test_split(diabetes_X, diabetes_Y,␣
↪test_size=0.34, random_state=97)

classifier1 = LogisticRegression(max_iter=250)
classifier1.fit(X_train, Y_train)
prediction1 = classifier1.predict_proba(X_test)
prediction1 = pd.DataFrame(prediction1, columns=classifier1.classes_)
prediction1 = prediction1['tested_positive']

classifier2 = SVC(probability=True)
```

```
classifier2.fit(X_train, Y_train)
prediction2 = classifier2.predict_proba(X_test)
prediction2 = pd.DataFrame(prediction2, columns=classifier2.classes_)
prediction2 = prediction2['tested_positive']
```

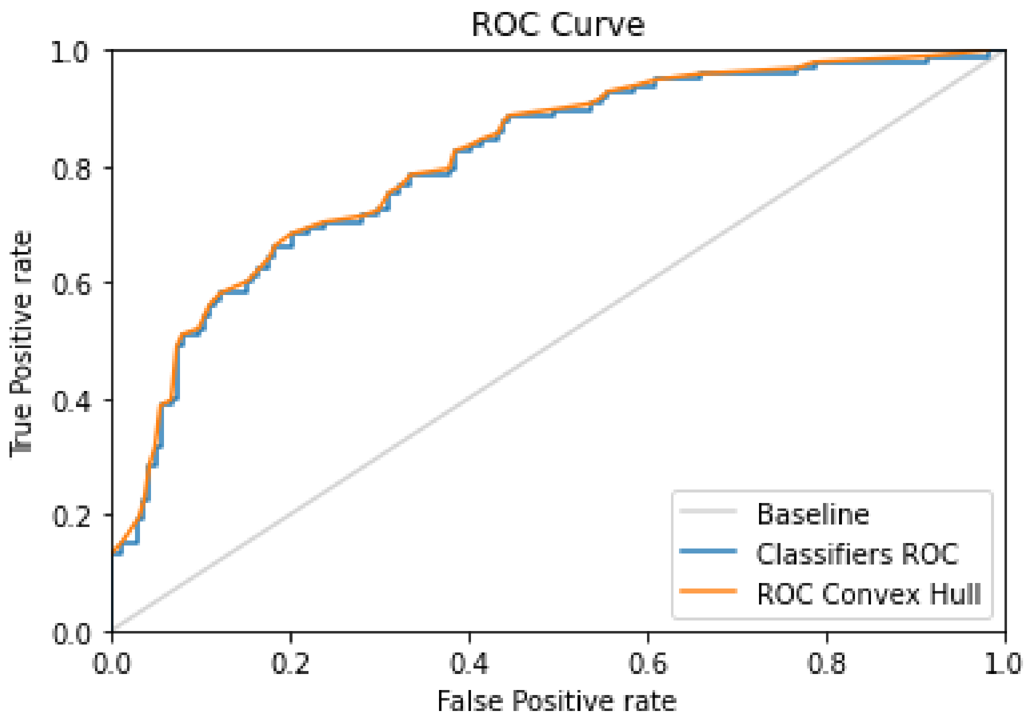
ROC Curve for LogisticRegression

[5]:
```
roc1 = ROC(prediction1, Y_test == 'tested_positive')
roc1.compute_ROC_coordinates()
roc1.compute_ROC_convex_hull_coordinates()
roc1.plot_ROC()
```



ROC Curve for SVC

[6]:
```
roc2 = ROC(prediction2, Y_test == 'tested_positive')
roc2.compute_ROC_coordinates()
roc2.compute_ROC_convex_hull_coordinates()
roc2.plot_ROC()
```



AUCROC Values

[7]:
```
auc_roc1 = roc1.compute_AUCROC()
auc_roc2 = roc2.compute_AUCROC()
print("AUCROC LogisticRegression: ", auc_roc1)
print("AUCROC SVC: ", auc_roc2)
```

```
AUCROC LogisticRegression:  0.8426455948232952
AUCROC SVC:  0.8066824290691886
```

We can see that the overall performance of both classifiers is similar, but based on the AUCROC scores LogisticRegression appears to perform a bit better.

Bot ROC curves show a moderate to poor separation of the classes.