**I. Pen-and-paper**

1. Assuming 1 is positive and 0 is negative

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 𝑑(i,j) |
|  |  |  |  |  |  |  |  | 1 |
| Uma imagem com texto, palavras cruzadas  Descrição gerada automaticamente |  |  |  |  |  |  |  | 2 |
|  |  |  |  |  |  |  |  | 3 |
|  |  |  |  |  |  |  |  | 4 |
|  |  |  |  |  |  |  |  | 5 |
|  |  |  |  |  |  |  |  | 6 |
|  |  |  |  |  |  |  |  | 7 |
|  |  |  |  |  |  |  |  | 8 |

1. Uma imagem com texto, mesa

   Descrição gerada automaticamenteAssuming 1 is positive and 0 is negative

Para class = 0:

Para class = 1:

1. Assuming 1 is positive and 0 is negative

The decision threshold 0.3 is the one that optimizes testing accuracy.

**II. Programming and critical analysis**

Confusion Matrix Naïve Bayes

Predicted class=0 Predicted class=1

Real class=0 67 125

Real class=1 69 495

Confusion Matrix kNN

Predicted class=0 Predicted class=1

Real class=0 50 142

Real class=1 67 497

1. p-value = 0.91 with H0: Naïve Bayes better or equal to kNN (H1: kNN better than Naïve Bayes). This means that this hypothesis H0 is accepted for levels of significance equal or under 91% and is rejected for higher levels. For the usual levels of significance (0.01, 0.05 and 0.1) H0 is accepted and H1 (the one we wanted to classify) is rejected. We can conclude that, in this situation, the hypothesis “kNN is statistically superior to Naïve Bayes regarding accuracy” cannot be classified as true.
3. kNN is sensitive to outliers. In this case, kNN only works with five elements (increasing the risk of overfit), while Naïve Bayes works with all of them.
4. Also, kNN did not considerate the weight and the data was not normalized, which may have decreased the accuracy.

**III. APPENDIX**

import pandas as pd

import math

from scipy.io.arff import loadarff

from sklearn.feature\_selection import SelectKBest

from sklearn.model\_selection import StratifiedKFold

from sklearn.naive\_bayes import GaussianNB

from sklearn.neighbors import KNeighborsClassifier

from sklearn import metrics

from scipy import stats

data = loadarff('pd\_speech.arff')

df = pd.DataFrame(data[0])

df['class'] = df['class'].str.decode('utf-8')

y = df['class']

X = df.drop('class', axis=1)

cv = StratifiedKFold(n\_splits=10, shuffle=True, random\_state=0)

naive\_bayes\_classifier = GaussianNB()

knn\_classifier = KNeighborsClassifier(n\_neighbors=5, weights='uniform', metric='euclidean')

cm\_kNN = [[0, 0], [0, 0]]

cm\_NB = [[0, 0], [0, 0]]

accuracy\_kNN = []

accuracy\_NB = []

for train\_index, test\_index in cv.split(X, y):

    X\_train, X\_test = X.iloc[train\_index], X.iloc[test\_index]

    y\_train, y\_test = y.iloc[train\_index], y.iloc[test\_index]

    naive\_bayes\_classifier.fit(X\_train, y\_train)

    y\_pred = naive\_bayes\_classifier.predict(X\_test)

    cm = metrics.confusion\_matrix(y\_test, y\_pred)

    cm\_NB = [ (a + b) for a, b in zip(cm\_NB, cm) ]

    accuracy\_NB += [metrics.accuracy\_score(y\_test, y\_pred)]

    knn\_classifier.fit(X\_train, y\_train)

    y\_pred = knn\_classifier.predict(X\_test)

    cm = metrics.confusion\_matrix(y\_test, y\_pred)

    cm\_kNN = [ (a + b) for a, b in zip(cm\_kNN, cm) ]

    accuracy\_kNN += [metrics.accuracy\_score(y\_test, y\_pred)]

confusion\_NB = pd.DataFrame(cm\_NB, index=['Real class=0', 'class=1'], columns=['Predicted class=0', 'Predicted class=1'])

confusion\_kNN = pd.DataFrame(cm\_kNN, index=['Real class=0', 'class=1'], columns=['Predicted class=0', 'Predicted class=1'])

print("Naïve Bayes Confusion Matrix\n ", confusion\_NB)

print("\n\nkNN Confusion Matrix\n", confusion\_kNN)

res = stats.ttest\_rel(accuracy\_kNN, accuracy\_NB, alternative='greater')

print(res.pvalue)

**END**