Aprendizagem 2022/23

Homework II – Group 103

I. Pen-and-paper

1)

1	Dist.	Weight	Class
2	2.5	0.4	+
3	1.5	0.6667	+
4	0.5	2	+
5	1.5	0.6667	-
6	1.5	0.6667	-
7	1.5	0.6667	-
8	2.5	0.4	-
predicte	+	TRUE	+

2	Dist.	Weight	Class
1	2.5	0.4	+
3	1.5	0.6667	+
4	2.5	0.4	+
5	1.5	0.6667	-
6	1.5	0.6667	-
7	1.5	0.6667	-
8	0.5	2	-
predictec	-	TRUE	+

3	Dist.	Weight	Class
1	1.5	0.6667	+
2	1.5	0.6667	+
4	1.5	0.6667	+
5	2.5	0.4	-
6	2.5	0.4	-
7	0.5	2	-
8	1.5	0.6667	-
predicte	-	TRUE	+

4	Dist.	Weight	Class
1	0.5	2	+
2	2.5	0.4	+
3	1.5	0.6667	+
5	1.5	0.6667	-
6	1.5	0.6667	-
7	1.5	0.6667	-
8	2.5	0.4	-
predictec	+	TRUE	+

5	Dist.	Weight	Class
1	1.5	0.6667	+
2	1.5	0.6667	+
3	2.5	0.4	+
4	1.5	0.6667	+
6	0.5	2	-
7	2.5	0.4	-
8	1.5	0.6667	-
predicte	-	TRUE	-

6	Dist.	Weight	Class
1	1.5	0.6667	+
2	1.5	0.6667	+
3	2.5	0.4	+
4	1.5	0.6667	+
5	0.5	2	-
7	2.5	0.4	-
8	1.5	0.6667	-
predictec	-	TRUE	-

7	Dist.	Weight	Class
1	1.5	0.6667	+
2	1.5	0.6667	+
3	0.5	2	+
4	1.5	0.6667	+
5	2.5	0.4	-
6	2.5	0.4	-
8	1.5	0.6667	-
predicte	+	TRUE	-

8	Dist.	Weight	Class
1	2.5	0.4	+
2	0.5	2	+
3	1.5	0.6667	+
4	2.5	0.4	+
5	1.5	0.6667	-
6	1.5	0.6667	-
7	1.5	0.6667	-
predictec	+	TRUE	-

RECALL = TP/(TP+FN) = 2/(2+2) = 0.5

2) P(P|y1,y2,y3) = P(P)*P(y1,y2,y3|P) / P(y1,y2,y3)

P(N|y1,y2,y3) = P(N)*P(y1,y2,y3|N) / P(y1,y2,y3)

Because it's a classifier and the denominator of the two formulas is the same, we don't need to calculate that, and because y3 is independent from the others we can separate them:

P(y1,y2,y3|P) = P(y1,y2|P) * P(y3|P)

P(y1,y2,y3|N) = P(y1,y2|N) * P(y3|N)

Next, we calculate each member of the formula:

P(P) = 5/9 | P(N) = 4/9

P(y1,y2|P) = { A0-2/5 , A1-1/5 , B0-1/5 , B1-1/5} | P(y1,y2|N) = {A0-0 , A1-1/4 , B0-1/2 , B1-1/4}

P(y3|P) = N(y3|Pmean, Pstan.dev.) | P(y3|N) = N(y3|Nmean, Nstan.dev.)

Pmean = 0.84 | Nmean = 0.975

Pstan.dev. = 0.251 | Nsta.dev. = 0.17078

3) Totalmean = 0.9 | Totalstan.dev. = 0.21794

P(P|y1,y2,y3) = 5/9 * P(y1,y2|P) * N(y3|Pmean, Pstan.dev.) / P(y1,y2,y3)

P(y1,y2,y3) = P(y1,y2) * N(y3|Totalmean, Totalstan.dev.)

P(A,1,0.8) = 2/9 * N(0.8|0.9, 0.21794) = 0.3661367

P(P|A,1,0.8) = 5/9 * 1/5 * N(0.8|0.84, 0.251) / 0.3661367 = 0.47625084

P(B,1,1) = 2/9 * N(1|0.9, 0.21794) = 0.3661367

P(P|B,1,1) = 5/9 * 1/5 * N(1|0.84, 0.251) / 0.3661367 = 0.39365367

P(B,0,0.9) = 3/9 * N(0.9|0.9, 0.21794) = 0.61017142
P(P|B,0,0.9) = 5/9 * 1/5 * N(0.9|0.84, 0.251) / 0.61017142 = 0.2812767

4) P(P|X) > 0.5 => Positive

All the entries are predicted negative, which is an 33% accurate

P(P|X) > 0.3 => Positive

The two first entries are predicted positive and the last one predicted negative, which is true, so the accuracy is 100%

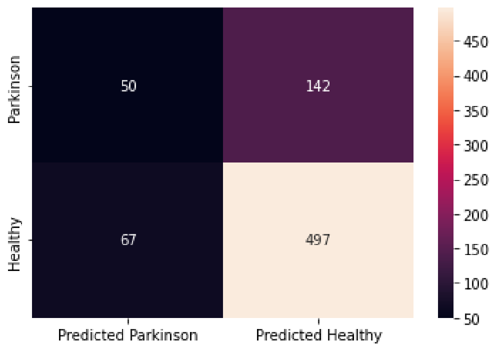
P(P|X) > 0.7 => Positive

All the entries are predicted negative, which is 33% accurate

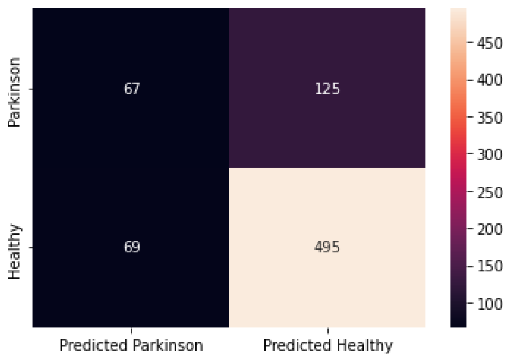
0.3 decision threshold optimizes testing accuracy

II. Programming and critical analysis

5)



kNN



NAÏVE BAYES

6) Pval = 0.9104476998751558

Given this value, we can't assume that kNN is statistically superior (in terms of accuracy) to Naïve Bayes.

7) Three reasons for the fact that we can't conclude that kNN is statistically superior to Naïve Bayes can be:

1. The number of neighbors (k=5) might not be adjusted to the context of the problem;
2. The data might be too extensive;
3. The fact that there are too few feature dependencies, Naïve Bayes might be favored and it's accuracy rises.

III. APPENDIX

5)

```
import pandas as pd
from sklearn.model_selection import StratifiedKFold, cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from scipy import stats

import numpy as np
import copy as cp
import matplotlib.pyplot as plt

import seaborn as sns
from typing import Tuple
from sklearn.metrics import confusion_matrix
from scipy.io.arff import loadarff

from sklearn.metrics import classification_report, confusion_matrix

# Reading the ARFF file
data = loadarff('pd_speech.arff')
df = pd.DataFrame(data[0])
df['class'] = df['class'].str.decode('utf-8')

X = df.drop(columns='class')
y = df["class"]

model_neig = KNeighborsClassifier(n_neighbors = 5)
model_gaus = GaussianNB()
kfold = StratifiedKFold(n_splits=10, random_state=0, shuffle=True)
```

```
def confusio(model):
  confusio_matrix = [[0,0],[0,0]]

  for train_ndx, test_ndx in kfold.split(X, y):

      train_X, test_X = X.iloc[train_ndx], X.iloc[test_ndx]
      train_y, test_y = y.iloc[train_ndx], y.iloc[test_ndx]

      model.fit(train_X, train_y)
      predicted_y = model.predict(test_X)

      matrix = confusion_matrix(test_y, predicted_y)

      tn, fp, fn, tp = matrix.ravel()
      confusio_matrix[0][0] += tn
      confusio_matrix[0][1] += fp
      confusio_matrix[1][0] += fn
      confusio_matrix[1][1] += tp

  cm = np.array(confusio_matrix)
  confusion = pd.DataFrame(cm, index=['Parkinson', 'Healthy'], columns=['Predicted Parkinson', 'Predicted Health
y'])

  sns.heatmap(confusion, annot=True, fmt='g')

#model_KNN
confusio(model_neig)

#model_gaussian
confusio(model_gaus)
```

6)

```
#p_value
acc_model = cross_val_score(model, X, y, cv=kfold, scoring='accuracy')
acc_model_1 = cross_val_score(model_1, X, y, cv=kfold, scoring='accuracy')
res = stats.ttest_rel(acc_model, acc_model_1, alternative='greater')
print("p1>p2? pval=",res.pvalue)
```

END