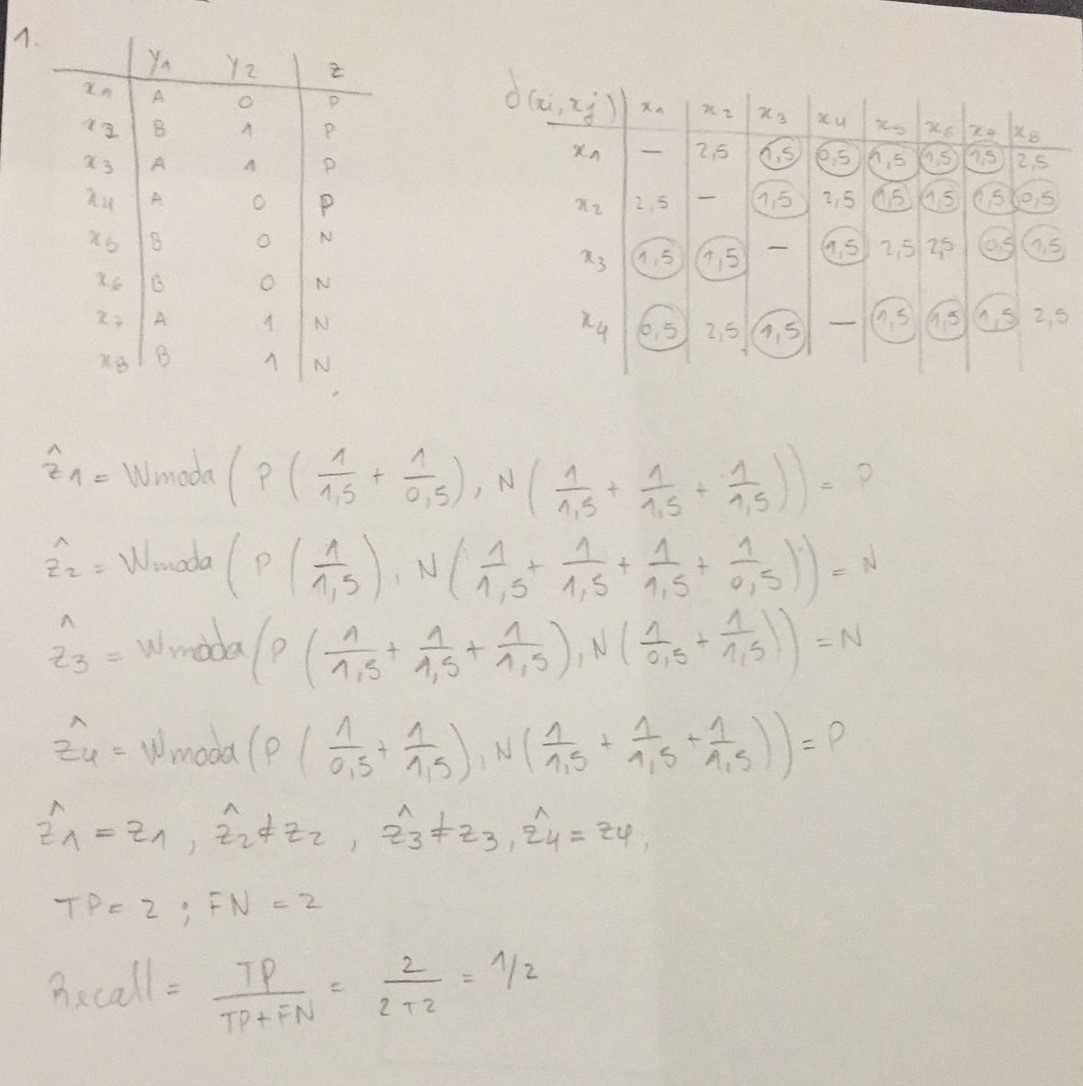
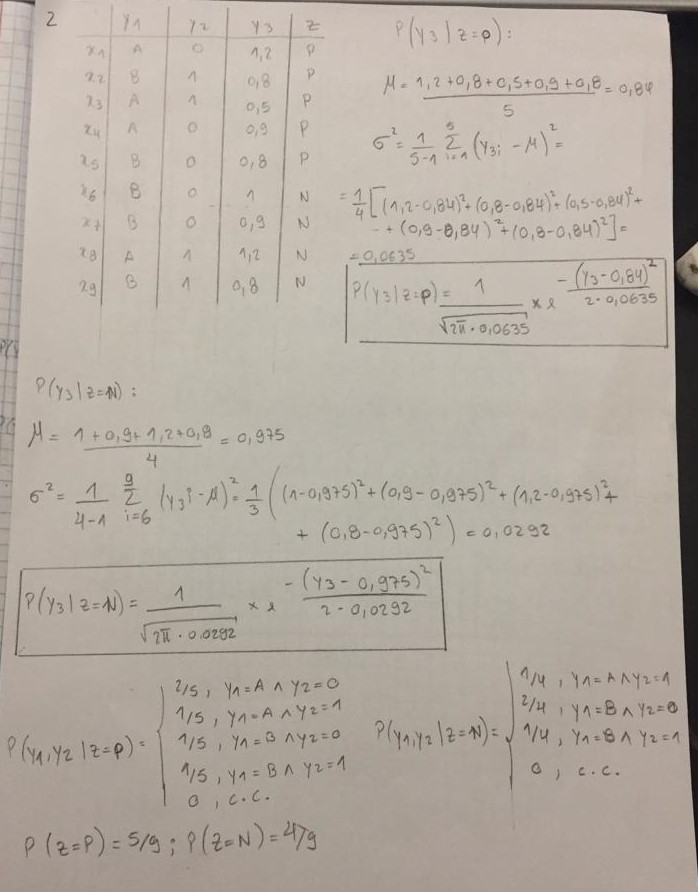
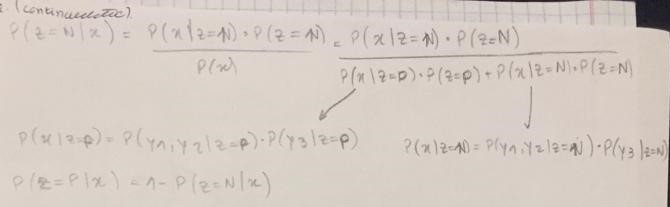
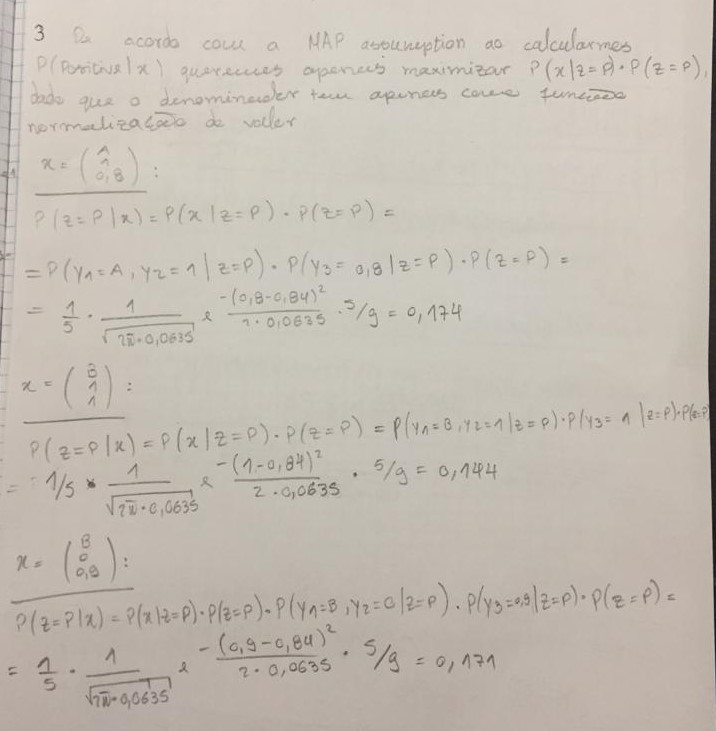
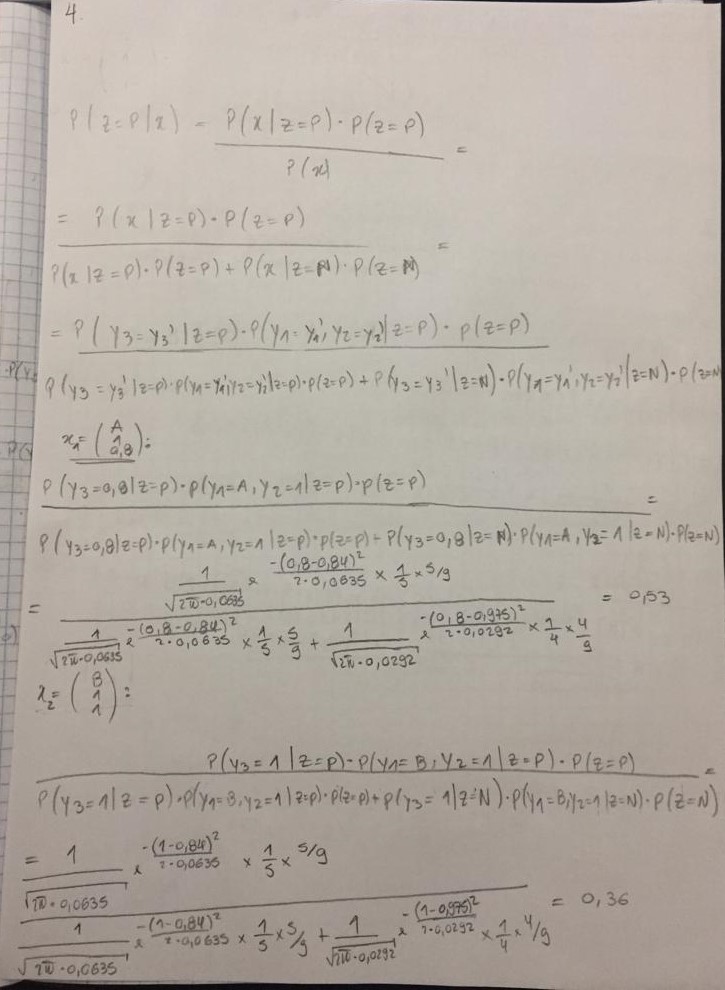
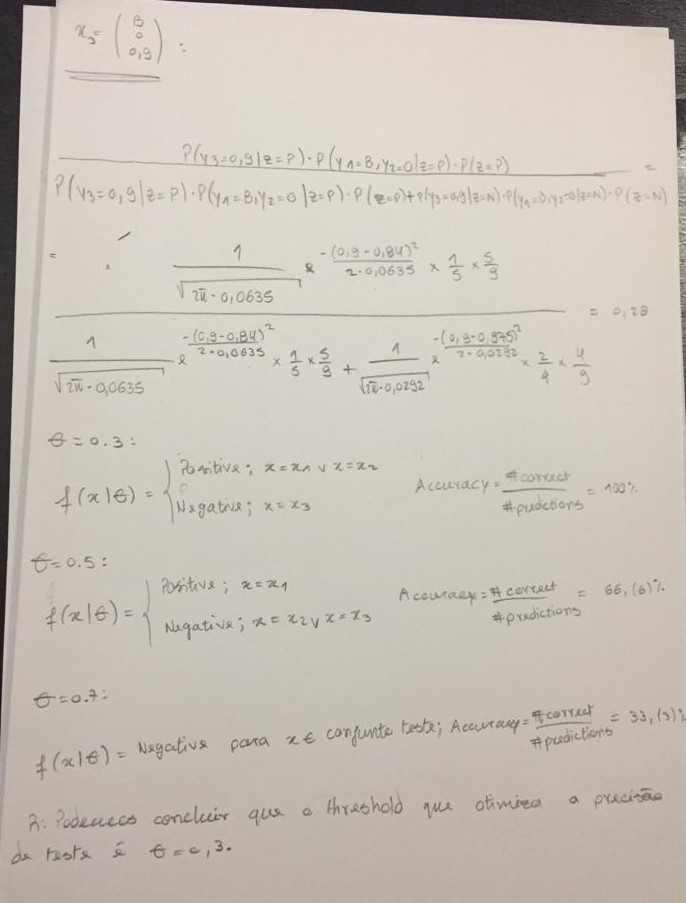
**I. Pen-and-paper**



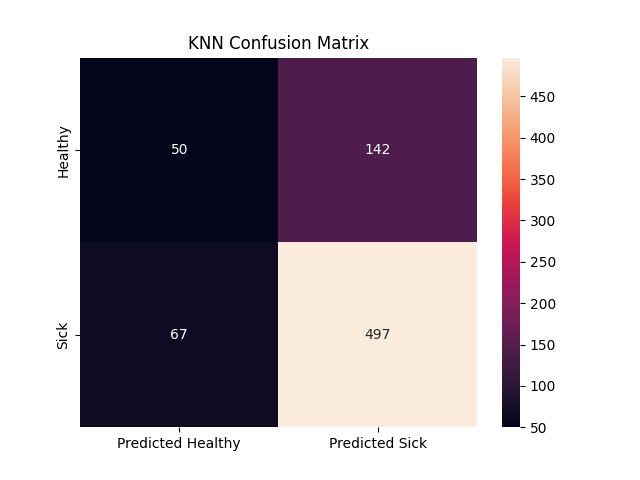


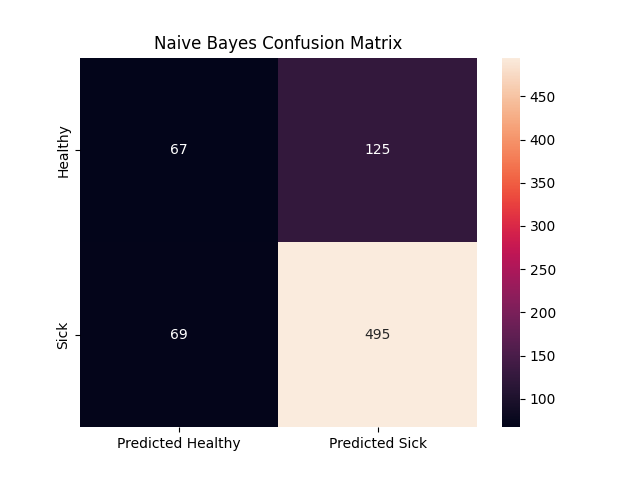


****

****

**II. Programming and critical analysis**

1. 



Tendo em conta que o p-value = 0.912 da hipótese “kNN é estatisticamente **superior** a Naive Bayes considerando precisão”, concluímos que, para um threshold de confiança de 0.1, podemos rejeitar esta hipótese (1 – p-value < 0.1), inferindo, desta forma, que é uma afirmação falsa.

O kNN apenas considera os 5 elementos mais próximos (poucos vizinhos), ao contrário do naive bayes, que tem em conta todo o conjunto para averiguar as probabilidades necessárias. Para além disso, nesta situação, o kNN não considerou os pesos dos elementos, diminuindo ainda mais a sua precisão, visto que atribuiu a mesma importância a valores a distâncias diferentes para o alvo.

**III. APPENDIX**

#######         Importing required libraries          #######

import pandas as pd

import seaborn as sns

from scipy import stats

import matplotlib.pyplot as plt

from sklearn import metrics

from sklearn.model\_selection import StratifiedKFold

from sklearn.neighbors import KNeighborsClassifier

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import confusion\_matrix

import numpy as np

from scipy.io.arff import loadarff

import warnings

def warn(\*args, \*\*kwargs): pass

warnings.warn = warn

#######            Reading the ARFF file              #######

# Load the data

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

df = pd.DataFrame(data[0])

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

#######            Folding and Classifiers            #######

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

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

# Creating the classifiers

predictor\_kNN = KNeighborsClassifier(weights='uniform', n\_neighbors=5, metric='euclidean')

predictor\_nb = GaussianNB()

#######   Running classifier and attesting results    #######

cm\_kNN, cm\_nb, kNN\_acc, nb\_acc = [], [], [], []

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

    # Getting the training and testing splits

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

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

    # Training the classifiers

    predictor\_kNN.fit(X\_train, y\_train)

    predictor\_nb.fit(X\_train, y\_train)

    # Predicting the classes

    y\_pred\_kNN = predictor\_kNN.predict(X\_test)

    y\_pred\_nb = predictor\_nb.predict(X\_test)

    # Computing the confusion matrices

    cm\_kNN.append(np.array(confusion\_matrix(y\_test, y\_pred\_kNN, labels=['0', '1'])))

    cm\_nb.append(np.array(confusion\_matrix(y\_test, y\_pred\_nb, labels=['0', '1'])))

    # Computing the accuracy

    kNN\_acc.append(round(metrics.accuracy\_score(y\_test, y\_pred\_kNN), 3))

    nb\_acc.append(round(metrics.accuracy\_score(y\_test, y\_pred\_nb), 3))

cm\_kNN = np.sum(cm\_kNN, axis=0)

cm\_nb = np.sum(cm\_nb, axis=0)

# Creating the confusion matrices' plot

confusion\_knn = pd.DataFrame(cm\_kNN, index=['Healthy', 'Sick', ], columns=['Predicted Healthy', 'Predicted Sick'])

confusion\_nb = pd.DataFrame(cm\_nb, index=['Healthy', 'Sick', ], columns=['Predicted Healthy', 'Predicted Sick'])

#######                     Ex 5                      #######

# KNN confusion matrix

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

plt.title('KNN Confusion Matrix')

plt.show()

# Naive Bayes confusion matrix

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

plt.title('Naive Bayes Confusion Matrix')

plt.show()

#######                     Ex 6                      #######

# Performing the t-test

res = stats.ttest\_rel(kNN\_acc, nb\_acc, alternative='greater')

print("knn accuracy: ", kNN\_acc)

print("nb accuracy: ", nb\_acc)

# Outputting the p-value

print("p1>p2? pval=",np.round(res.pvalue, 3))

**END**