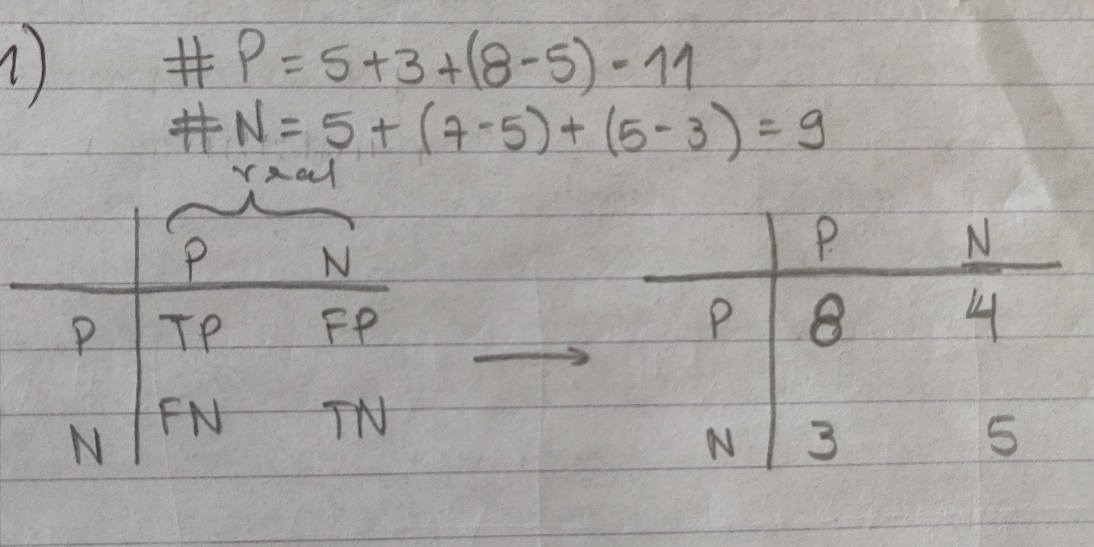
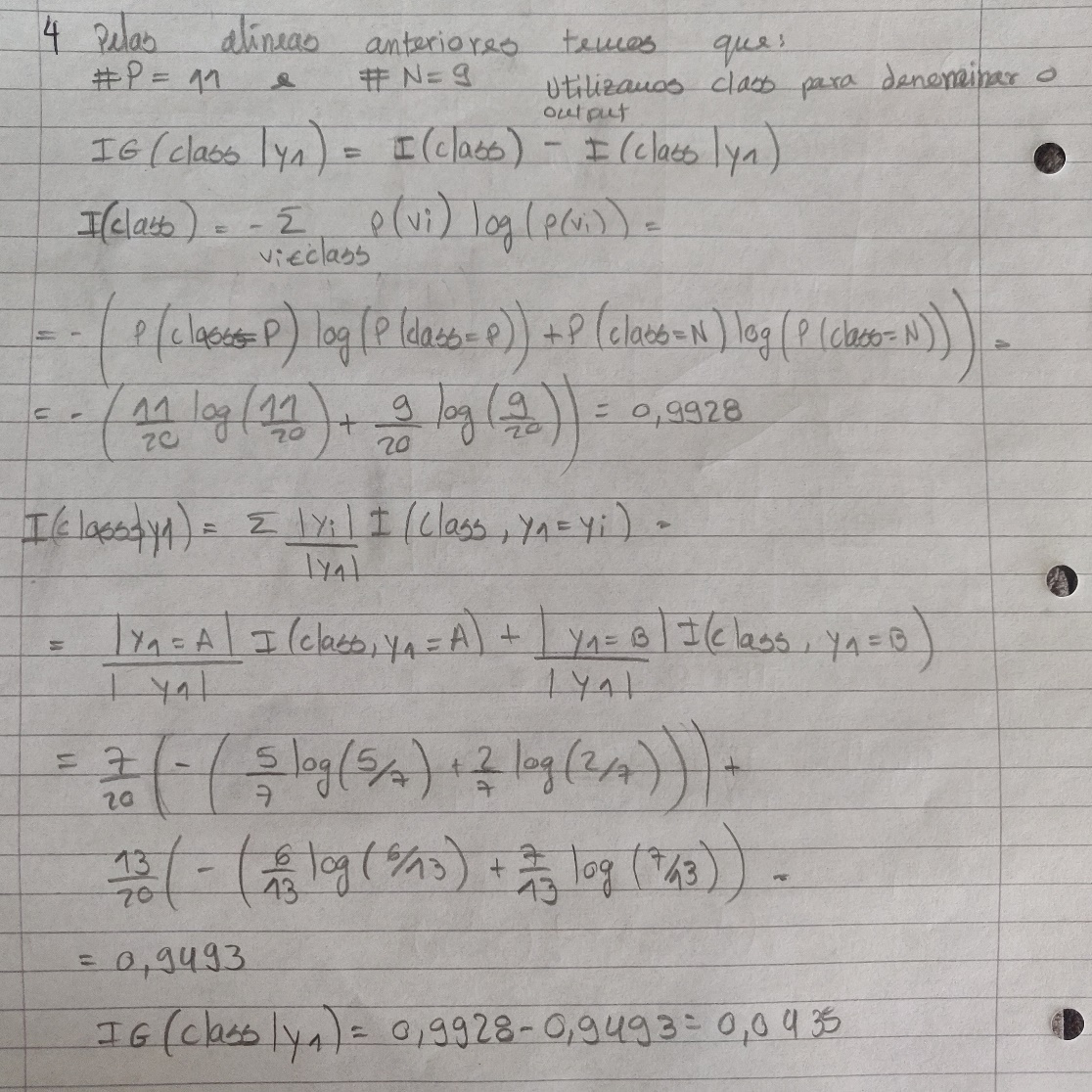
**I. Pen-and-paper**

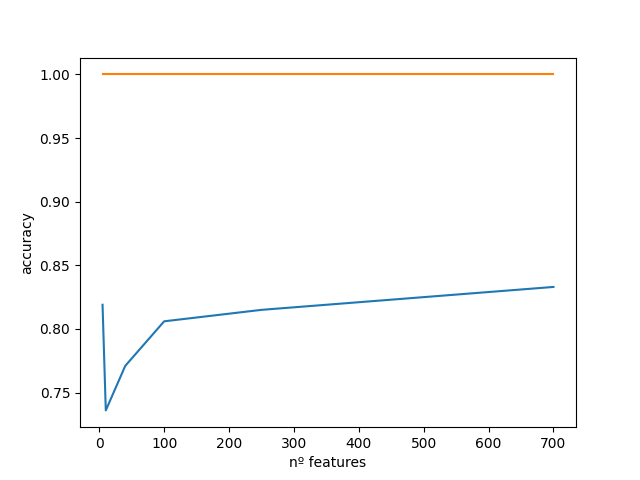


1. Uma imagem com texto

   Descrição gerada automaticamente
2. Podemos ter que, para qualquer outra variável, o seu information gain quando y1=A seja nulo, pelo que não se justifica realizar a decomposição.
3. Ao olharmos para a árvore conseguimos verificar que quando y1=A temos uma precisão de 5/7, podendo, desta forma, pressupor que a expansão deste nó levaria a um cenário de overfitting, pelo que o seu crescimento foi interrompido.

****

**II. Programming and critical analysis**



Como não estabelecemos nenhum limite para a profundidade da árvore de decisão, esta irá ramificar-se indefinidamente, adaptando-se perfeitamente ao training set, fazendo com que a training accuracy seja 1. Ao observarmos o gráfico verificamos que a precisão no test set é significativamente inferior à do training set (<1, como seria de esperar), pelo que podemos concluir que estamos perante um cenário de overfitting.

**III. APPENDIX**

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

import pandas as pd

from scipy.io.arff import loadarff

from sklearn.model\_selection import train\_test\_split

from sklearn.feature\_selection import mutual\_info\_classif

from sklearn import tree, metrics

import matplotlib.pyplot as plt

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

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

df = pd.DataFrame(data[0])

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

#######      Creating the training-testing split      #######

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, train\_size=0.7, stratify=y, random\_state=1)

####### Feature ranking based on discriminative power #######

dp = mutual\_info\_classif(X, y, random\_state=1)

dict1, count = {}, 0

for feature in X\_train.columns.values:

    dict1[feature] = dp[count]

    count += 1

dict2 = dict(sorted(dict1.items(), key=lambda item: item[1], reverse=True))

features = list(dict2.keys())

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

train, test = [], []

predictor = tree.DecisionTreeClassifier(random\_state=1)

for n in [5, 10, 40, 100, 250, 700]:

    predictor.fit(X\_train[features[0:n]], y\_train)

    y\_pred1 = predictor.predict(X\_test[features[0:n]])

    y\_pred2 = predictor.predict(X\_train[features[0:n]])

    test.append(round(metrics.accuracy\_score(y\_test, y\_pred1), 3))

    train.append(round(metrics.accuracy\_score(y\_train, y\_pred2), 3))

#######                Creating plot                  #######

plt.plot([5, 10, 40, 100, 250, 700], test)

plt.plot([5, 10, 40, 100, 250, 700], train)

plt.xlabel('nº features')

plt.ylabel('accuracy')

plt.show()

**END**