UNIVERSIDADE AUTÓNOMA DE LISBOA

Inteligência Artificial

Professor: Gonçalo Valadão

Trabalho realizado por:

Daniel Ventura – 30002840

Gonçalo Dinis – 30002640

Rodrigo Gomes - 30002639

1. Introdução

No âmbito da disciplina de Inteligência Artificial foi nos proposto criar 3 programas em *python*:

- Um que resolva um sudoku usando um algoritmo baseado no AC-3
- Um que resolva um filtro de spam/ham usando um algoritmo baseado no Nayve Bayes
- Um que resolva um filtro de spam/ham usando um algoritmo baseado no Perceptrão

Iremos fazer uma breve introdução aos conceitos que achámos importantes em cada um dos exercícios.

De seguida vamos mostrar o código feito para cada um dos programas, deixando algumas notas para acompanhar o código, que está devidamente comentado.

2. Sudoku com base no AC-3

Sudoku, por vezes escrito Su Doku (数独 'sūdoku') é um jogo baseado na colocação lógica de números. O objetivo do jogo é a colocação de números de 1 a 9 em cada uma das células vazias numa grade de 9x9, constituída por 3x3 subgrades chamadas regiões.

O quebra-cabeça contém algumas pistas iniciais, que são números inseridos em algumas células, de maneira a permitir uma indução ou dedução dos números em células que estejam vazias. Cada coluna, linha e região só pode ter um número de cada um dos 1 a 9.

Resolver o problema requer apenas raciocínio lógico e algum tempo. Os problemas são normalmente classificados em relação à sua realização.

O algoritmo AC-3 (abreviação de Arc Consistency Algorithm # 3) é um de uma série de algoritmos usados para a solução de problemas de satisfação de restrições (ou CSP's). Foi desenvolvido por Alan Mackworth em 1977. Os algoritmos AC anteriores são frequentemente considerados muito ineficientes, e muitos dos posteriores são difíceis de implementar, portanto AC-3 é o mais frequentemente ensinado e usado em solucionadores de restrições muito simples.

Código:
Importar a biblioteca queue para podermos criar filas:

```
1   # -*- coding: utf-8 -*-
2   import queue
3
```

Criação das diferentes linhas do sudoku:

```
116
117 #coluna / linha
118 linha1 = [0] * 9
119 linha2 = [0] * 9
120 linha3 = [0] * 9
121 linha4 = [0] * 9
122 linha5 = [0] * 9
123 linha6 = [0] * 9
124 linha7 = [0] * 9
125 linha8 = [0] * 9
126 linha9 = [0] * 9
127
```

Preenchimento das linhas do sudoku com os valores default que vêm com o exercício:

```
127
128 # primeira linha
129 linha1[2] = 3
130 linha1[4] = 2
131 linha1[6] = 6
132
133 # segunda linha
134 linha2[0] = 9
135 linha2[3] = 3
136 linha2[5] = 5
137 linha2[8] = 1
138
139 # terceira linha
140 linha3[2] = 1
141 linha3[3] = 8
142 linha3[5] = 6
143 linha3[6] = 4
144
145 # quarta linha
146 linha4[2] = 8
147 linha4[3] = 1
148 linha4[5] = 2
149 linha4[6] = 9
150
151 # quinta linha
152 linha5[0] = 7
153 linha5[8] = 8
154
155 # sexta linha
156 linha6[2] = 6
157 linha6[3] = 7
158 linha6[5] = 8
159 linha6[6] = 2
160
161 # setima linha
162 linha7[2] = 2
163 linha7[3] = 6
164 linha7[5] = 9
165 linha7[6] = 5
166
167 # oitava linha
168 linha8[0] = 8
169 linha8[3] = 2
170 linha8[5] = 3
171 linha8[8] = 9
172
173 # nona linha
174 linha9[2] = 5
175 linha9[4] = 1
176 linha9[6] = 3
177
178
179
180 sudoku = [linha1, linha2, linha3, linha4, linha5, linha6, linha7, linha8, linha9]
181
```

Mostrar como o sudoku está antes da sua resolução, inicia o processo de resolução do sudoku e quando acabar mostra o seu aspeto final:

```
182
183
184     #mostrar como está o sudoku antes da sua resolução
185     for item in range(9):
186         print(sudoku[item])
187
188
189
190
191     #iniciar o processo de resolução do sudoku
192     sudoku = ac3(sudoku)
193
194
195     #mostrar o sudoku resolvido
196     print("\nAspecto final do Sudoku: \n")
197     for item in range(9):
198         print(sudoku[item])
199
```

Cria uma fila para ser preenchida com todos os arcos do csp, cria uma lista “csp” que contém 2 listas: na posição 0 tem a lista das coordenadas de cada um dos i-ésimos quadrados do sudoku e na posição 1 tem a lista do domínio de valores possíveis para o i-ésimo quadrado. De seguida preenche a variável csp.

```
36
37  #função que recebe como parâmetro o sudoku original e retorna o csp resolvido
38  def ac3(sudoku):
39      q = queue.Queue()
40      csp = [[], []]
41      #[0] = Linha & Coluna
42      #[1] = ValoresPossiveis
43
44      #adicionar os valores possiveis ao dominio dos diferentes quadrados
45      for linha in range(9):
46          for coluna in range(9):
47              if(sudoku[linha][coluna] == 0):
48                  csp[0].append([linha, coluna])
49                  csp[1].append([1,2,3,4,5,6,7,8,9])
50              else:
51                  csp[0].append([linha, coluna])
52                  csp[1].append([sudoku[linha][coluna]])
53
```

Percorre o sudoku com 2 quadrados ao mesmo tempo e caso possuam uma restrição entre sí, adiciona esse arco à fila q.

```
    #CRIAR A PRIMEIRA FILA DE ARCOS
    #andar pelo sudoku com o primeiro quadrado
    for linhaA in range(9):
        for colunaA in range(9):
            #andar pelo sudoku com o segundo quadrado
            for linhaB in range(9):
                for colunaB in range(9):
                    #se estiverem na mesma posição passa a frente (um quadrado não pode ser arco dele próprio)
                    if((linhaA == linhaB) and (colunaA == colunaB)):
                        pass
                    else:
                        #verificar se têm ou a mesma linha, ou a mesma coluna ou se pretencem à mesma região
                        if(linhaA == linhaB) or (colunaA == colunaB) or (verificarSeEstaoNoMesmoQuadrado(linhaA, colunaA, linhaB, colunaB)):
                            q.put([[linhaA, colunaA], [linhaB, colunaB]])
```

Função usada para verificar se dois quadrados pertencem à mesma região:

```
#função usada para verificar se dois quadrados estão na mesma região ou não
def verificarSeEstaoNoMesmoQuadrado(linhaA, colunaA, linhaB, colunaB):
    if(linhaA < 3) and (colunaA < 3) and (linhaB < 3) and (colunaB < 3):
        return True
    elif((linhaA < 6) and (linhaA >= 3)) and (colunaA < 3) and ((linhaB < 6) and (linhaB >= 3)) and (colunaB < 3):
        return True
    elif((linhaA < 9) and (linhaA >= 6)) and (colunaA < 3) and ((linhaB < 9) and (linhaB >= 6)) and (colunaB < 3):
        return True
    elif(linhaA < 3) and ((colunaA < 6) and (colunaA >= 3)) and (linhaB < 3) and ((colunaB < 6) and (colunaB >= 3)):
        return True
    elif((linhaA < 6) and (linhaA >= 3)) and ((colunaA < 6) and (colunaA >= 3)) and ((linhaB < 6) and (linhaB >= 3)) and ((colunaB < 6) and (colunaB >= 3)):
        return True
    elif((linhaA < 9) and (linhaA >= 6)) and((colunaA < 6) and (colunaA >= 3)) and ((linhaB < 9) and (linhaB >= 6)) and ((colunaB < 6) and (colunaB >= 3)):
        return True
    elif(linhaA < 3) and ((colunaA < 9) and (colunaA >= 6)) and (linhaB < 3) and ((colunaB < 9) and (colunaB >= 6)):
        return True
    elif((linhaA < 6) and (linhaA >= 3)) and ((colunaA < 9) and (colunaA >= 6)) and ((linhaB < 6) and (linhaB >= 3)) and ((colunaB < 9) and (colunaB >= 6)):
        return True
    elif((linhaA < 9) and (linhaA >= 6)) and ((colunaA < 9) and (colunaA >= 6)) and ((linhaB < 9) and (linhaB >= 6)) and ((colunaB < 9) and (colunaB >= 6)):
        return True
    else:
        return False
```

Após o primeiro preenchimento da fila q vamos começar a tratar dos arcos. Chama a função revise para cada arco que encontra na fila para verificar se o segundo quadrado tem apenas um valor no seu domínio para ser preenchido, se sim retira-se esse valor do domínio do primeiro quadrado. Se tiver sido retirado pelo menos um valor do domínio do primeiro quadrado verifica-se se o domínio do primeiro quadrado ficou vazio (se ficou a resolução termina pois não foi encontrada uma resolução para o problema). Se não chama-se a função neighbors que dado um quadrado retorna uma lista de quadrados que fazem vizinhança com o primeiro e adiciona-se os arcos de cada um desses quadrados com o quadrado inicial à fila q, pois como alterou o seu domínio é possível que ajude a simplificar o problema com os outros quadrados:

```
    #percorrer a fila de arcos
    while not q.empty():
        arc = q.get()
        quadrado1 = arc[0]
        linha = quadrado1[0]
        coluna = quadrado1[1]
        quadrado2 = arc[1]
        #chama a função revise para verificar se o segundo quadrado tem apenas uma opção para ser preenchido, se sim retira-se esse valor do domínio do primeiro quadrado
        if(revise(csp, quadrado1, quadrado2)):
            #verificar se o domínio do primeiro quadrado ficou vazio, se sim retorna falso pois é impossível de resolver com o ac-3
            if(len(csp[1][csp[0].index(quadrado1)]) == 0):
                return False
            else:
            #se não vai verificar os vizinhos do quadrado 1 pois como alterou o seu domínio já pode ter disponíbilizado novas oportunidades
                neighborsAux = neighbors(linha, coluna, q)
                for i in neighborsAux:
                    q.put([i, quadrado1])
                    q.put([quadrado1, i])
```

Função usada para a partir do csp e de dois quadrados, verificar se o segundo quadrado tem apenas uma opção para ser preenchido, se sim retira-se esse valor do domínio de valores possíveis do primeiro quadrado, e quando acabar vai retornar true, para o programa perceber que o domínio do primeiro quadrado foi alterado e tirar partido disso.

```
17
18  #função usada para a partir do csp e de dois quadrados, verificar se o segundo quadrado tem apenas uma opção para ser preenchido
19  #se sim retira-se esse valor do domínio de valores possíveis do primeiro quadrado
20  def revise(csp, quadrado1, quadrado2):
21      linha1 = quadrado1[0]
22      coluna1 = quadrado1[1]
23      linha2 = quadrado2[0]
24      coluna2 = quadrado2[1]
25      revised = False
26      aux = [linha1, coluna1]
27      for i in csp[1][csp[0].index(aux)]:
28          i = [i]
29          if(len(csp[1][csp[0].index([linha2, coluna2])]) == 1) and (i == csp[1][csp[0].index([linha2, coluna2])]):
30              csp[1][csp[0].index([linha1, coluna1])].pop(csp[1][csp[0].index([linha1, coluna1])].index(i[0]))
31              revised = True
32      return revised
33
34
```

Função usada para dadas as coordenadas de um quadrado, guardar numa lista quais são os quadrados vizinhos e retorna-la:

```
5   #função usada para dadas as coordenadas de um quadrado, guardar numa lista quais são os quadrados vizinhos e retorna-la
6   def neighbors(linha, coluna, q):
7       neighborsAux = []
8       for linhaAux in range(9):
9           for colunaAux in range(9):
10              if((linhaAux == linha) and (colunaAux == coluna)):
11                  pass
12              else:
13                  if(linhaAux == linha) or (colunaAux == coluna) or (verificarSeEstaoNoMesmoQuadrado(linha, coluna, linhaAux, colunaAux)):
14                          neighborsAux.append([linhaAux, colunaAux])
15      return neighborsAux
16
```

Depois de a fila q terminar, ou seja de já não existirem arcos para serem analisados vamos colocar o csp na variável sudoku para que se consiga visualizar em linha por coluna e não os valores todos seguidos (para o programa ser mais user friendly), e retorna-se a variável sudoku.

```
87
88          #formata de forma a que seja mais visivel a trasnformação e retorna o sudoku resolvido
89          k = 0
90          for i in range (9):
91              for j in range (9):
92                  sudoku[i][j] = csp[1][k]
93                  k = k + 1
94          return sudoku
```

3. Filtro Spam com base no algoritmo de Naive Bayes

Em estatística, os classificadores Naive Bayes são uma família de "classificadores probabilísticos" simples baseados na aplicação do teorema de Bayes com suposições de independência fortes (ingênuas) entre os recursos. Eles estão entre os modelos de rede bayesiana mais simples, mas, juntamente com a estimativa de densidade do Kernel, podem atingir níveis de precisão mais altos.

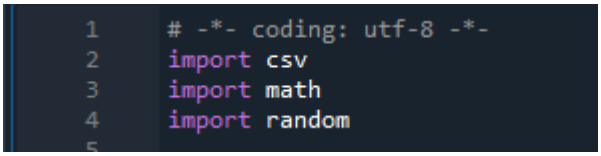
Os classificadores Naïve Bayes são altamente escaláveis, exigindo uma série de parâmetros lineares no número de variáveis (recursos / preditores) em um problema de aprendizagem. O treinamento de máxima verossimilhança pode ser feito avaliando uma expressão de forma fechada, que leva tempo linear, ao invés de uma cara aproximação iterativa como usada para muitos outros tipos de classificadores.

Várias técnicas anti-spam são usadas para evitar spam de e-mail (e-mail em massa não solicitado).

Nenhuma técnica é uma solução completa para o problema de spam, e cada uma tem compensações entre rejeitar incorretamente e-mail legítimo (falsos positivos) em oposição a não rejeitar todo o spam (falsos negativos) - e os custos associados em tempo, esforço e custo de obstruindo injustamente o correio válido.

As técnicas anti-spam podem ser divididas em quatro categorias amplas: aquelas que requerem ações individuais, aquelas que podem ser automatizadas por administradores de email, aquelas que podem ser automatizadas por remetentes de email e aquelas empregadas por pesquisadores e policiais.

Código:
Importar as bibliotecas: csv para podermos utilizar fichieros csv, math para podermos realizar operações matemáticas como logaritmos e random para podermos baralhar os dados.



```
# -*- coding: utf-8 -*-
import csv
import math
import random
```

Inicialização das listas: coluna1, coluna2 e linha. Vai guardar os dados necessários do ficheiro csv. Depois baralha as linhas termos uma experiencia diferente sempre que corremos o programa e assim podermos fazer as nossas análises e tirar as nossas conclusões de forma mais eficiente. Faz uma “lavagem” para retirar os dados que não são interessantes e que podiam criar interferência se não fossem tirados como é o caso da pontuação:

```
#--------------------------inicio do programa-------------------------------
#criar as listas coluna1, coluna2 e linha
coluna1 = []
coluna2 = []
linha = []

#ir buscar os dados necessários ao ficheiro spam.csv
with open('spam.csv') as csv_file:
    csv_reader = csv.reader(csv_file, delimiter='#')
    line_count = 0
    #string que contém os simbolos que vão ser apagados pois não são relevantes
    retirar = '''!()-[]{};:'"\, <>./?@#$%^&*_~0123456789'''
    i = 0

    for row in csv_reader:
        #passar a frente o cabeçalho (v1, v2,,,)
        if(i == 0):
            i = i + 1
        else:
            #dividir apenas uma vez para podermos ter a palavra "ham" ou "spam" no split[0] e termos o resto da mensagem no split[1]
            split = row[0].split(',', 1)
            linha.append(split[0] + "," + split[1].lower())
            i = i + 1
    #baralhar as linhas
    random.shuffle(linha)
    #depois de baralhadas vamos adicionar cada linha ao par de colunas
    for i in linha:
        split = i.split(',', 1)
        coluna1.append(split[0])
        coluna2.append(split[1])
    print('\nAcabou de ler dados do ficheiro.\n')

    #retirar pontuação de cada linha da coluna2 e colocar um espaço em vez disso
    for ele in range(len(coluna2)):
        for ele2 in range(len(coluna2[ele])):
            if coluna2[ele][ele2] in retirar:
                coluna2[ele] = coluna2[ele].replace(coluna2[ele][ele2], " ")
```

Colocar na variável treino 70 porcento dos dados, na variável validação 15 porcento dos dados e na variável teste os restantes 15 porcento dos dados.
Chamar a função funcaoTreino que leva como parâmetro a variável treino que são o dicionário de dados do conjunto de treino e que retorna para a variável valores um conjunto de dados importantes para os restantes conjuntos.
Chamar a função funcaoTeste que leva como parâmetros a variável teste e os valores retornados da função anterior e retorna para a variável valores o melhorC para ser utilizado no conjunto de teste.
Chamar a função funcaoValidacao que leva como parâmetros a variável validacao e os valores retornados das funções anteriores:

```
267  #colocar na variavel treino 70 por cento dos dados
268  coluna1Aux = coluna1[:int(len(coluna1)*0.7)]
269  coluna2Aux = coluna2[:int(len(coluna2)*0.7)]
270  treino = coluna1Aux, coluna2Aux
271
272  #colocar na variavel validacao 15 por cento dos dados
273  coluna1Aux = coluna1[int(len(coluna1)*0.7):int(len(coluna1)*0.85)]
274  coluna2Aux = coluna2[int(len(coluna2)*0.7):int(len(coluna2)*0.85)]
275  teste = coluna1Aux, coluna2Aux
276
277  #colocar na variavel teste os restantes 15 por cento dos dados
278  coluna1Aux = coluna1[int(len(coluna1)*0.85):]
279  coluna2Aux = coluna2[int(len(coluna2)*0.85):]
280  validacao = coluna1Aux, coluna2Aux
281
282  #chamar a funcao funcaoTreino que leva como parametro a variavel treino que são o dicionario de dados do conjunto de treino
283  #e que retorna para a variavel valores um conjunto de dados importantes para os restantes conjuntos
284  print("\n\n\tCONJUNTO DE TREINO\n")
285  valores = funcaoTreino(treino)
286
287  #chamar a funcao funcaoTeste que leva como parametros a variavel teste e os valores retornados da funcao anterior
288  #e retorna para a variavel valores o melhorC para ser utilizado no conjunto de teste
289  print("\n\n\tCONJUNTO DE TESTE\n")
290  valores.append(funcaoTeste(teste, valores))
291
292  #chamar a funcao funcaoValidacao que leva como parametros a variavel validacao e os valores retornados das funcoes anteriores
293  print("\n\n\tCONJUNTO DE VALIDAÇÃO\n")
294  funcaoValidacao(validacao, valores)
```

A função funcaoTreino é a função que trata do conjunto de treino, conta quantos emails, emails ham e emails spam existem. Depois chama a função calcularPalavras que retorna o p (bag of words). Depois retorna as variáveis mHam, mSpam e o p.

```
204  #funcao que trata do conjunto de treino
205  def funcaoTreino(treino):
206      coluna1 = treino[0]
207      m = 0
208      mHam = 0
209      mSpam = 0
210      #precorrer a coluna1
211      for i in range(len(coluna1)):
212          #contar emails ham/spam
213          if(coluna1[i] == "ham"):
214              mHam = mHam + 1
215          elif(coluna1[i] == "spam"):
216              mSpam = mSpam + 1
217          m = m + 1
218
219      print('\nAcabou de calcular as mensagens.\n')
220
221      #Funcao que recebe o m, mHam, mSpam e p e que calcula o p (bag of words)
222      p = calcularPalavras(m, mHam, mSpam)
223
224      #agrupa variaveis para retornar e retorna-as
225      retorno = [mHam, mSpam, p]
226      return retorno
227
```

A função calcularPalavras é a função que calcula o p (bag of words). Percorre cada uma das palavras das mensagens, adiciona-as ao p[0] e incrementa 1 sempre que ela volta a aparecer no p[1] (se for ham) ou no p[2] (se for spam). Depois chama a função normalizar para tornar o conteúdo das variáveis p[1] e p[2] em frequências relativas.

```
157  #funcao que calcula o p (bag of words)
158  def calcularPalavras(m, mHam, mSpam):
159
160      #bag of words
161      p = [[], [], []]
162      #[0] -> palavra
163      #[1] -> numero de vezes que aparece no ham
164      #[2] -> numero de vezes que aparece no spam
165
166      wHamList = []
167      wSpamList = []
168      wHam = 0
169      wSpam = 0
170      #precorre cada uma das mensagens
171      for i in range(m):
172          #dividir as frases em palavras
173          split = coluna2[i].split()
174          #precorrer as palavras das frases e adiciona-las ao P
175          #se já existem no p apenas incrementa-se o numero de vezes que a palavra
176          #aparece no ham ou no spam dependendo do suposto
177          for j in range (len(split)):
178              if(coluna1[i] == "ham"):
179                  if(split[j] in wHamList):
180                      p[1][p[0].index(split[j])] = p[1][p[0].index(split[j])] + 1
181                  else:
182                      p[0].append(split[j])
183                      p[1].append(2)
184                      p[2].append(1)
185                      wHamList.append(split[j])
186                      wHam = wHam + 1
187
188              elif(coluna1[i] == "spam"):
189                  if(split[j] in wSpamList):
190                      p[2][p[0].index(split[j])] = p[2][p[0].index(split[j])] + 1
191                  else:
192                      p[0].append(split[j])
193                      p[1].append(1)
194                      p[2].append(2)
195                      wSpamList.append(split[j])
196                      wSpam = wSpam + 1
197
198      print('\nAcabou de calcular as palavras das mensagens.\n')
199
200      #função que normaliza os dados do p (bag of words)
201      normalizar(p, wHam, wSpam)
202      return p
```

A função normalizar é a função que normaliza o p (bag of words) tornando o conteúdo das variáveis p[1] e de p[2] em frequências relativas.

```
149
150     #funcao que normaliza o p(bag of words)
151     def normalizar(p, wHam, wSpam):
152         for i in range(len(p[0])):
153             p[1][i] = p[1][i]/ wHam
154             p[2][i] = p[2][i]/ wSpam
155         print("\nAcabou de normalizar o P\n")
156
```

A função funcaoTeste é a função que trata da escolha do melhor c dependendo de determinadas métricas de classificação. Para isso são testados 11 valores diferentes para serem usados como c. Para cada um dos valores é chamada a função calcularClassificações que calcula os valores tp, fp, tn e fn e retorna os resultados. Depois mostra os resultados das métricas e adiciona às listas supostas os resultados.

```
78  #funcao que trata da escolha do melhor c dependendo de determinados metricas de classificacao
79  def funcaoTeste(teste, valores):
80
81      coluna1 = teste[0]
82      coluna2 = teste[1]
83      mHam = valores[0]
84      mSpam = valores[1]
85      p = valores[2]
86
87      #vamos testar o c com os seguintes valores e vamos verificar qual dos 11 valores é o melhor para ser usado
88      c = [0.05, 0.1, 0.5, 1, 5, 10, 20, 30, 50, 75, 100]
89
90      #criar as listas que vao verificar qual é o melhor c
91      calcularMinimosErradosC = []
92      calcularAccuracyC = []
93      calcularErrorRateC = []
94      calcularPrecisao = []
95
96      #vamos classificar os dados de validacao com os diferentes valores de c para escolhermos o melhor
97      for a in c:
98          #chama uma funcao que calcula os valores: tp, fp, tn e fn e retorna os resultados
99          valores = calcularClassificacoes(coluna1, coluna2, mHam, mSpam, p, a)
100         tp = valores[0]
101         fp = valores[1]
102         tn = valores[2]
103         fn = valores[3]
104
105         #mostra o resultado das métricas e adiciona às listas supostas os resultados
106         calcularMinimosErradosC.append(fp + fn)
107         print("Erradas: ", (fp + fn))
108
109         calcularAccuracyC.append((tp + tn)/(tp + fp + tn + fn))
110         print("Accuracy: ", (tp + tn)/(tp + fp + tn + fn))
111
112         calcularErrorRateC.append((fp + fn)/(tp + fp + tn + fn))
113         print("Error Rate: ", (fp + fn)/(tp + fp + tn + fn))
114
115         calcularPrecisao.append(tp/(tp + fp))
116         print("Precisão: ", (fp + fn)/(tp + fp + tn + fn))
```

A função calcularClassificacoes calcula os valores tp, fp, tn e fn e retorna os resultados. Chama a função classify que retorna a classificação de uma determinada mensagem levando como parâmetros a mensagem, o b (treshold) e o p (bag of words).

Depois calcula quais das classificações acertou e quais não e retorna os valores:

```
#chama uma funcao que calcula os valores: tp, fp, tn e fn e retorna os resultados
def calcularClassificacoes(coluna1, coluna2 ,mHam, mSpam, p, c):
    b = math.log(c) + math.log(mHam) - math.log(mSpam)

    classificacoes = []
    for i in coluna2:
        #chama a funcao classify que retorna a classificacao de uma determinada mensagem
        #levando como parametros a mensagem, o b (treshold) e o p (bag of words)
        classificacao = classify(i, b, p)
        classificacoes.append(classificacao)


    print("\nAcabou de classificar as mensagens\n")

    #faz as contas de quais foram os hams e os spams que erraram e acertaram para o determinado c
    hC = 0
    hE = 0
    sC = 0
    sE = 0
    for i in range(len(classificacoes)):
        if(coluna1[i] == "ham"):
            if("ham" == classificacoes[i]):
                hC = hC + 1
            else:
                hE = hE + 1
        else:
            if("spam" == classificacoes[i]):
                sC = sC + 1
            else:
                sE = sE + 1
    print("\nAcabou de verificar as classificações com c igual a ", c, ":\n")

    print("Ham corretos: ", hC)
    print("Ham errados: ", hE)
    print("Spam corretos: ", sC)
    print("Spam errados: ", sE)
    #hC = ham correto = tp = true positive
    #hE = ham errado = fp = false positive
    #sC = spam correto = tn = true negative
    #sE = spam errado = fn = false negative
    valores = [hC, hE, sC, sE]
    return valores
```

A função classify é uma função que retorna a classificação de uma determinada mensagem levando como parâmetros a mensagem, o b (treshold) e o p (bag of words). Vai colocando palava a palavra o logaritmo das vezes que aparecem no spam – (menos) o logaritmo das vezes que aparecem no ham (dados estão em frequência relativa) e vai somando o resultado ao t para no final se t for superior a 0, o email é classificado como spam, senão é classificado como ham.

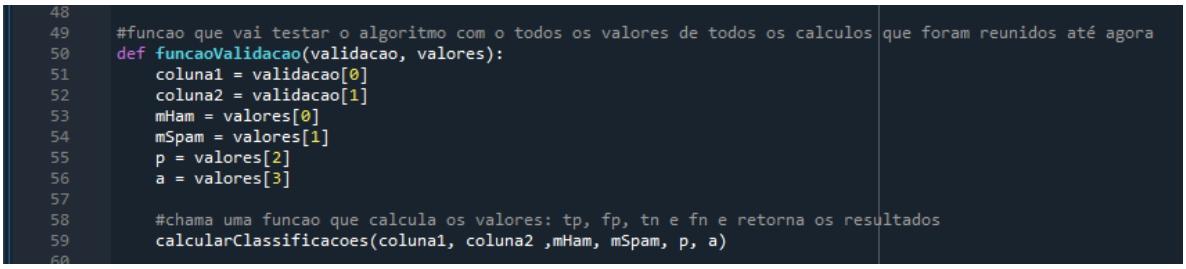
```
#chama a funcao classify que retorna a classificacao de uma determinada mensagem
#levando como parametros a mensagem, o b (treshold) e o p (bag of words)
def classify(mensagem, b, p):
    t = -b
    split = mensagem.split()
    #vai calcular palavra a palavra o logaritmo das vezes que aparecem no spam - (menos) o logaritmo das vezes que aparecem no ham
    #(dados em frequencia relativa) e vai somando o resultado ao t para no final se t for superior a 0, o email é classificado como
    #spam, senão o email é classificado como ham.
    for k in split:
        if(k in p[0]):
            t = t + (math.log(p[2][p[0].index(k)]) - math.log(p[1][p[0].index(k)]))

    if(t>0):
        return "spam"
    else:
        return "ham"
```

Continuando com a função funcaoTeste, depois de mostrar os resultados das métricas e adicioná-los às listas supostas os resultados, vão ser calculados os melhores valores de c para as determinadas métricas. Cada uma das 11 posições representa um valor de c, dependendo de qual receber mais votos é escolhido para ser o c no conjunto dos testes. O melhor valor de c é retornado.

```
118         #calcula os melhores valores de c para as determinadas métricas
119         MinimosErradosC = calcularMinimosErradosC.index(min(calcularMinimosErradosC))
120         MaxAccC = calcularAccuracyC.index(max(calcularAccuracyC))
121         MinErrC = calcularErrorRateC.index(min(calcularErrorRateC))
122         MaxPrec = calcularPrecisao.index(max(calcularPrecisao))
123 
124         #Mostra quais foram os melhores valores de C para as determinadas Métricas
125         print("Errou menos: ", c[MinimosErradosC])
126         print("Maior Acc: ", c[MaxAccC])
127         print("Menor taxa de erros: ", c[MinErrC])
128         print("Maior precisão: ", c[MaxPrec])
129 
130         #cada uma das 11 posições representa um valor de c, dependendo de qual receber mais votos ganha
131         escolha = [0] * 11
132         votos = []
133         votos.append(MinimosErradosC)
134         votos.append(MaxAccC)
135         votos.append(MinErrC)
136         votos.append(MaxPrec)
137 
138         #faz os votos
139         for i in range(len(votos)):
140             escolha[votos[i]] = escolha[votos[i]] + 1
141         print("\n\nVotos: ", escolha)
142         #o melhor c é o valor que tiver mais votos
143         melhorC = c[escolha.index(max(escolha))]
144 
145         print("\n\nO melhor C foi: ", melhorC)
146         #retorna o melhor valor de C
147         return melhorC
```

A função funcaoValidacao é a função que vai testar o algoritmo com o p, mHam e mSpam que foram definidos no conjunto de treino em conjunto com o a que é o melhor valor de c que foi definido no conjunto de teste. É chamada a função calcularClassificacoes que já foi falada anteriormente.



```
48 
49  #funcao que vai testar o algoritmo com o todos os valores de todos os calculos que foram reunidos até agora
50  def funcaoValidacao(validacao, valores):
51      coluna1 = validacao[0]
52      coluna2 = validacao[1]
53      mHam = valores[0]
54      mSpam = valores[1]
55      p = valores[2]
56      a = valores[3]
57 
58      #chama uma funcao que calcula os valores: tp, fp, tn e fn e retorna os resultados
59      calcularClassificacoes(coluna1, coluna2 ,mHam, mSpam, p, a)
60 
```

4. Filtro Spam com base no algoritmo de Perceptrão

Na aprendizagem de uma máquina, o perceptrão é um algoritmo de aprendizagem supervisionada de classificadores binários. Um classificador binário é uma função que pode decidir se uma entrada, representada por um vetor de números, pertence ou não a alguma classe específica. É um tipo de classificador linear, ou seja, um algoritmo de classificação que faz suas previsões com base em uma função de preditor linear combinando um conjunto de pesos com o vetor de características.

Código:

Importar as bibliotecas: csv para podermos utilizar fichieros csv e random para podermos baralhar os dados.

```
1   # -*- coding: utf-8 -*-
2
3   import csv
4   import random
```

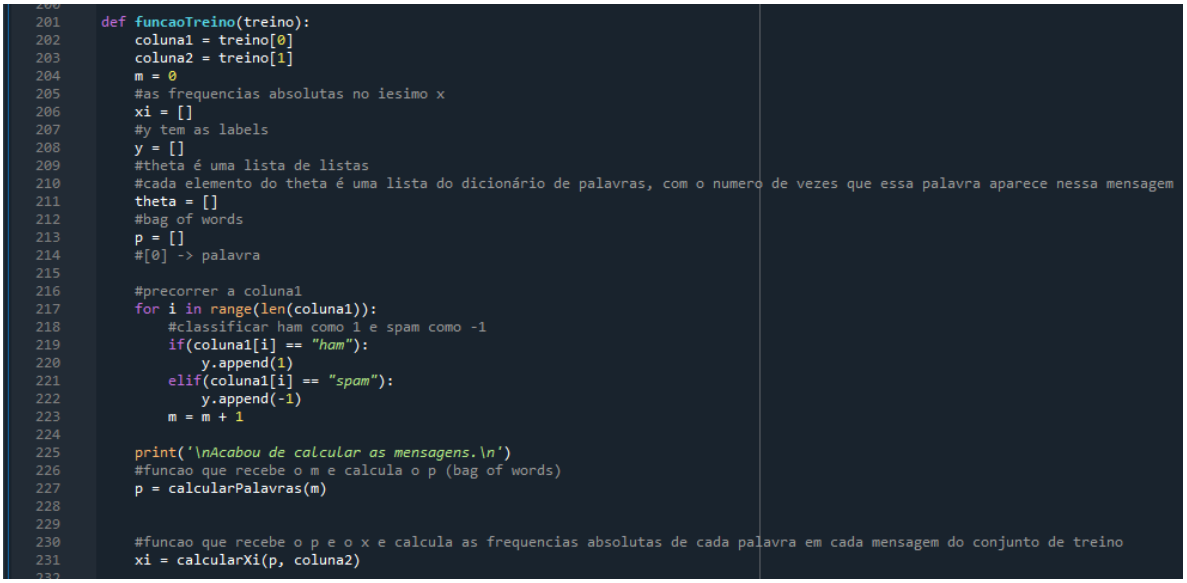
Inicialização das listas: coluna1, coluna2 e linha. Vai guardar os dados necessários do ficheiro csv. Depois baralha as linhas termos uma experiencia diferente sempre que corremos o programa e assim podermos fazer as nossas análises e tirar as nossas conclusões de forma mais eficiente. Faz uma “lavagem” para retirar os dados que não são interessantes e que podiam criar interferência se não fossem tirados como é o caso da pontuação:

```
274 #--------------------------inicio do programa-------------------------------
275 #criar as listas coluna1, coluna2 e linha
276 coluna1 = []
277 coluna2 = []
278 linha = []
279
280 #ir buscar os dados necessários ao ficheiro spam.csv
281 with open('spam.csv') as csv_file:
282     csv_reader = csv.reader(csv_file, delimiter='#')
283     line_count = 0
284     #string que contém os simbolos que vão ser apagados pois não são relevantes
285     retirar = '''!()-[]{};:'"\, <>./?@#$%^&*_~0123456789'''
286     i = 0
287
288     for row in csv_reader:
289         #passar a frente o cabeçalho (v1, v2,,,)
290         if(i == 0):
291             i = i + 1
292         else:
293             #dividir apenas uma vez para podermos ter a palavra "ham" ou "spam" no split[0] e termos o resto da mensagem no split[1]
294             split = row[0].split(',', 1)
295             if(len(split[1]) != 0):
296                 linha.append(split[0] + "-----^^-----" + split[1].lower())
297             i = i + 1
298     #baralhar as linhas
299     random.shuffle(linha)
300     #depois de baralhadas vamos adicionar cada linha ao par de colunas
301     for i in linha:
302         split = i.split("-----^^-----")
303         coluna1.append(split[0])
304         coluna2.append(split[1])
305     print('\nAcabou de ler dados do ficheiro.\n')
306
307     #retirar pontuação de cada linha da coluna2 e colocar um espaço em vez disso
308     for ele in range(len(coluna2)):
309         for ele2 in range(len(coluna2[ele])):
310             if coluna2[ele][ele2] in retirar:
311                 coluna2[ele] = coluna2[ele].replace(coluna2[ele][ele2], " ")
312
```

Colocar na variável treino 70 porcento dos dados, na variável validação 15 porcento dos dados e na variável teste os restantes 15 porcento dos dados.
Chamar a função funcaoTreino que leva como parâmetro a variável treino que são o dicionário de dados do conjunto de treino e que retorna para a variável valores um conjunto de dados importantes para os restantes conjuntos.
Chamar a função funcaoTeste que leva como parâmetros a variável teste e os valores retornados da função anterior e retorna para a variável valores o melhorT para ser utilizado no conjunto de teste.
Chamar a função funcaoValidacao que leva como parâmetros a variável validacao e os valores retornados das funções anteriores:

```
313
314     #colocar na variavel treino 70 por cento dos dados
315     coluna1Aux = coluna1[:int(len(coluna1)*0.7)]
316     coluna2Aux = coluna2[:int(len(coluna2)*0.7)]
317     treino = coluna1Aux, coluna2Aux
318
319     #colocar na variavel validacao 15 por cento dos dados
320     coluna1Aux = coluna1[int(len(coluna1)*0.7):int(len(coluna1)*0.85)]
321     coluna2Aux = coluna2[int(len(coluna2)*0.7):int(len(coluna2)*0.85)]
322     teste = coluna1Aux, coluna2Aux
323
324     #colocar na variavel teste os restantes 15 por cento dos dados
325     coluna1Aux = coluna1[int(len(coluna1)*0.85):]
326     coluna2Aux = coluna2[int(len(coluna2)*0.85):]
327     validacao = coluna1Aux, coluna2Aux
328
329     #chamar a funcao funcaoTreino que leva como parametro a variavel treino que é o dicionario de dados do conjunto de treino
330     #e que retorna para a variavel valores um conjunto de dados importantes para os restantes conjuntos
331     print("\n\n\tCONJUNTO DE TREINO\n")
332     valores = funcaoTreino(treino)
333
334     #chamar a funcao funcaoTeste que leva como parametros a variavel teste e os valores retornados da funcao anterior
335     #e retorna para a variavel valores o melhorT para ser utilizado no conjunto de teste
336     print("\n\n\tCONJUNTO DE TESTE\n")
337     valores = funcaoTeste(teste, valores)
338
339     #chamar a funcao funcaoValidacao que leva como parametros a variavel validacao e os valores retornados das funcoes anteriores
340     print("\n\n\tCONJUNTO DE VALIDAÇÃO\n")
341     funcaoValidacao(validacao, valores)
```

A função funcaoTreino é a função que trata do conjunto de treino, para cada label da coluna1 (lista com apenas as palavras “ham” ou “spam”) vai adicionar à lista y o valor 1 se a label for “ham” e adiciona o valor -1 se a label for “spam”. Adiciona ainda mais 1 ao m para cada elemento da lista. Chama a função calcularPalavras que leva como parâmetro o m e retorna o p (bag of words). Chama a função calcularXi que leva como parâmetro o p e a coluna2 e retorna as frequências absolutas de cada palavra em cada mensagem do conjunto de treino.



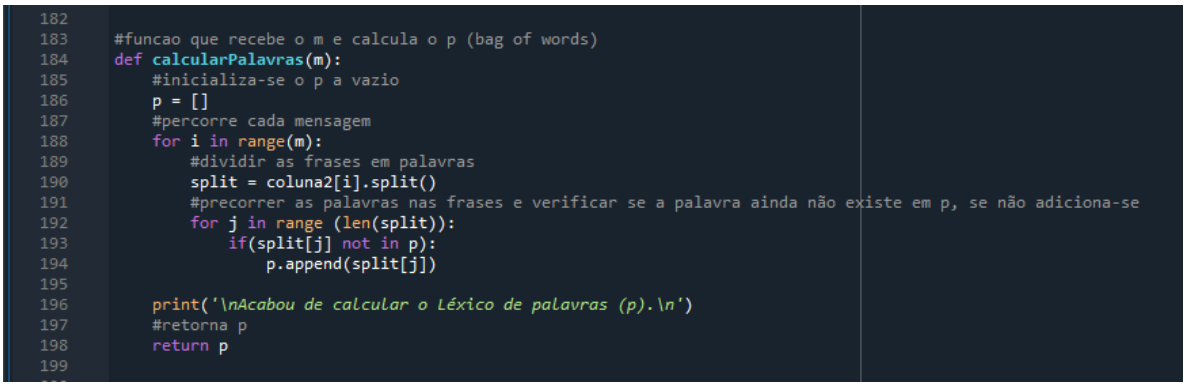
```
def funcaoTreino(treino):
    coluna1 = treino[0]
    coluna2 = treino[1]
    m = 0
    #as frequencias absolutas no iesimo x
    xi = []
    #y tem as labels
    y = []
    #theta é uma lista de listas
    #cada elemento do theta é uma lista do dicionário de palavras, com o numero de vezes que essa palavra aparece nessa mensagem
    theta = []
    #bag of words
    p = []
    #[0] -> palavra

    #precorrer a coluna1
    for i in range(len(coluna1)):
        #classificar ham como 1 e spam como -1
        if(coluna1[i] == "ham"):
            y.append(1)
        elif(coluna1[i] == "spam"):
            y.append(-1)
        m = m + 1

    print('\nAcabou de calcular as mensagens.\n')
    #funcao que recebe o m e calcula o p (bag of words)
    p = calcularPalavras(m)


    #funcao que recebe o p e o x e calcula as frequencias absolutas de cada palavra em cada mensagem do conjunto de treino
    xi = calcularXi(p, coluna2)
```

A função calcularPalavras recebe m e calcula p (bag of words). Percorre cada palavra da mensagem e se ainda não existir em p, adiciona-a.



```

#funcao que recebe o m e calcula o p (bag of words)
def calcularPalavras(m):
    #inicializa-se o p a vazio
    p = []
    #percorre cada mensagem
    for i in range(m):
        #dividir as frases em palavras
        split = coluna2[i].split()
        #precorrer as palavras nas frases e verificar se a palavra ainda não existe em p, se não adiciona-se
        for j in range (len(split)):
            if(split[j] not in p):
                p.append(split[j])

    print('\nAcabou de calcular o Léxico de palavras (p).\n')
    #retorna p
    return p

```

A função calcularXi recebe p e x como parâmetros e calcula xi, lista que contém as frequências absolutas de cada palavra em cada mensagem do x.

```
154
155 #funcao que recebe o p e o x e calcula as frequencias absolutas de cada palavra em cada mensagem do conjunto de x
156 def calcularXi(p, x):
157     #inicializa-se o xi a vazio
158     xi = []
159
160     #percorrer cada mensagem
161     for mensagem in x:
162         split = mensagem.split()
163         #percorrer palavra do lexico
164         numeroDePalavrasDoLexico = []
165         for lexico in p:
166             numero = 0
167             #percorrer cada palavra de cada mensagem
168             for palavraDaMensagem in split:
169                 #se a palavra da mensagem for igual a palavra do lexico adiciona-se a variavel numero o valor dela mais 1
170                 if(palavraDaMensagem == lexico):
171                     numero = numero + 1
172             #aux vai ter o numero de vezes que uma palavra do lexico aparece numa determinada mensagem
173             aux = numero
174             #adiciona-se aux à lista numeroDePalavrasDoLexico
175             numeroDePalavrasDoLexico.append(aux)
176         #no final de correr o lexico de palavras todo, adiciona-se a lista numeroDePalavrasDoLexico à lista xi e passa-se para a prox mensagem
177         xi.append(numeroDePalavrasDoLexico)
178     print("\nAcabou de calcular o Xi.\n")
179     #no final de percorrer todas as mensagens retorna-se xi
180     return xi
181
```

Voltando para a função funcaoTreino, depois de calcular o p (bag of words), vamos percorrer a lista t que tem os valores que vão ser avaliados para se perceber qual é o melhor T, e no final de cada um deles vamos guardar o theta e o thetaZero calculados.

Em cada uma das iterações vamos percorrer o algoritmo do perceptrão as vezes que forem o numeroDeEpocas.

Em cada uma das iterações vamos percorrer cada mensagem, fazer o produto interno de theta com xi e adicionar thetaZero a esse resultado. Se a multiplicação do resultado com a label da mensagem for negativa (não acertou a predição) vai adicionar a cada valor de theta o seu valor somado com a multiplicação de cada vez que cada valor de theta aparece em cada mensagem com 1 ou -1, dependendo da label da mensagem (vai adicionar a theta o número de vezes que cada palavra aparece na mensagem, esse valor vai ser positivo se for uma mensagem ham ou vai ser negativo se for uma mensagem spam).

No final retorna o p, o conjuntoDeThetas e t:

```
        #a lista t tem as opções de numeros de epocas que vão ser avaliadas para se perceber qual é o melhor T
        t = [1, 3, 5, 10]
        #variavel que vai ter os diferentes conjuntos de thetas, [0] = theta, [1] = thetaZero
        conjuntoDeThetas = []

        #precorrer o algoritmo do perceptrao para cada opção de T
        for numeroDeEpocas in t:

            #preencher o theta a 0
            theta = [0] * len(p)

            #inicializar o tetaZero com 0
            thetaZero = 0

            #algoritmo do perceptrão:
            #precorrer o algoritmo para cada t igual a numeroDeEpocas
            for epoca in range(numeroDeEpocas):
                #precorrer cada mensagem
                for i in range(len(coluna2)):
                    #declarar aux = 0
                    aux = 0
                    #fazer o produto interno de theta com xi
                    for j in range(len(theta)):
                        aux = aux + theta[j] * xi[i][j]
                    #adicionar thetaZero ao numero produto interno de theta e xi
                    aux = aux + thetaZero
                    #verificar se acertou na predição
                    if(y[i] * aux <= 0):
                        #se não acertou vai adicionar a cada valor de theta o seu valor
                        #somado com a multiplicacao de cada vez que cada valor de theta aparece em cada mensagem
                        #com 1 ou -1, dependendo da label da mensagem
                        for l in range(len(theta)):
                            theta[l] = theta[l] + y[i] * xi[i][l]
                        thetaZero = thetaZero + y[i]
            #adicionar o theta e o thetaZero ao conjunto de thetas, para no final de percorrer t, verificar-se qual o melhor t
            conjuntoDeThetas.append([theta, thetaZero])
            print("Acabou agora de calcular o theta e o thetaZero para ", numeroDeEpocas, " número(s) de epocas!\n")
        print('\nAcabou de calcular os diferentes thetas.\n')
        #no final retorna-se o p (bag of words) e o conjuntoDeThetas
        retorno = [p, conjuntoDeThetas, t]
        return retorno
```

A função funcaoTeste é a função que trata da escolha do melhor valor para T dependendo de determinadas métricas de classificação. Chama a função calcularXi que já foi falada, mas desta vez é para calcular as frequências absolutas de cada palavra em cada mensagem do conjunto de teste. E inicializamos as listas que vão verificar qual é o melhor valor para T:

```
74
75      #funcao que trata da escolha do melhor T dependendo de determinados metricas de classificacao
76      def funcaoTeste(teste, valores):
77          coluna1 = teste[0]
78          coluna2 = teste[1]
79          p = valores[0]
80          conjuntoDeThetas = valores[1]
81          t = valores[2]
82
83          #chama a funcao calcularXi que recebe o p e o coluna2 e calcula as frequencias absolutas
84          #de cada palavra em cada mensagem do conjunto de teste
85          v = calcularXi(p, coluna2)
86
87
88
89          #criar as listas que vao verificar qual é o melhor T
90          calcularMinimosErradosT = []
91          calcularAccuracyT = []
92          calcularErrorRateT = []
93          calcularPrecisaoT = []
94
```

Para cada valor de t vamos classificar os dados de validação para depois escolhermos o melhor.

Chama a função calcularClassificacoes que leva como parâmetros: theta, thetaZero, numeroDeEpocas, v e coluna1. Depois mostra o resultado das métricas e adiciona às listas supostas os resultados.

Cada lista calcula qual o melhor valor de T com a sua métrica e depois cada uma vota num valor de T, o que for mais votado ganha. No final retorna p, conjuntoDeThetas, o melhorT e t.

```
95        #vamos classificar os dados de validacao com os diferentes valores de T para escolhermos o melhor
96        for cT in range(len(t)):
97            numeroDeEpocas = t[cT]
98            theta = conjuntoDeThetas[cT][0]
99            thetaZero = conjuntoDeThetas[cT][1]
100           #chama a funcao que calcula os valores: tp, fp, tn e fn e retorna os resultados
101           valores = calcularClassificacoes(theta, thetaZero, numeroDeEpocas, v, coluna1)
102           tp = valores[0]
103           fp = valores[1]
104           tn = valores[2]
105           fn = valores[3]
106
107           #mostra o resultado das métricas e adiciona às listas supostas os resultados
108           calcularMinimosErradosT.append(fp + fn)
109           print("Erradas: ", (fp + fn))
110
111           calcularAccuracyT.append((tp + tn)/(tp + fp + tn + fn))
112           print("Accuracy: ", (tp + tn)/(tp + fp + tn + fn))
113
114           calcularErrorRateT.append((fp + fn)/(tp + fp + tn + fn))
115           print("Error Rate: ", (fp + fn)/(tp + fp + tn + fn))
116
117           calcularPrecisaoT.append(tp/(tp + fp))
118           print("Precisão: ", (fp + fn)/(tp + fp + tn + fn))
119
120       #calcula os melhores valores de T para as determinadas métricas
121       MinimosErradosT = calcularMinimosErradosT.index(min(calcularMinimosErradosT))
122       MaxAccT = calcularAccuracyT.index(max(calcularAccuracyT))
123       MinErrT = calcularErrorRateT.index(min(calcularErrorRateT))
124       MaxPreT = calcularPrecisaoT.index(max(calcularPrecisaoT))
125
126       #Mostra quais foram os melhores valores de T para as determinadas Métricas
127       print("\n\nErrou menos: ", t[MinimosErradosT])
128       print("Maior Acc: ", t[MaxAccT])
129       print("Menor taxa de erros: ", t[MinErrT])
130       print("Maior precisão: ", t[MaxPreT])
131
132       #cada uma das 4 posições representa um valor de T, dependendo de qual receber mais votos ganha
133       escolha = [0] * len(t)
134       votos = []
135       votos.append(MinimosErradosT)
136       votos.append(MaxAccT)
137       votos.append(MinErrT)
138       votos.append(MaxPreT)
139
140       #faz os votos
141       for i in range(len(votos)):
142           escolha[votos[i]] = escolha[votos[i]] + 1
143       print("\n\nVotos: ", escolha)
144       #o melhor T é o valor que tiver mais votos
145       melhorT = t[escolha.index(max(escolha))]
146
147       print("\n\nO melhor T foi: ", melhorT)
148
149       retorno = [p, conjuntoDeThetas, melhorT, t]
150       #retorna o melhor valor de T
151       return retorno
```

A função calcularClassificações é a função que calcula os valores de tp, fp, tn, fn e retorna os resultados. Basicamente faz o mesmo que o algoritmo do perceptrão faz no conjunto de treino, só que não verifica se acertou na predição, esse cálculo é feito depois para calcular o tp, fp, tn e fn e retorná-los.

```
6   #funcao que calcula os valores: tp, fp, tn e fn e retorna os resultados
7   def calcularClassificacoes(theta, thetaZero, numeroDeEpocas, v, coluna1):
8       #inicializa a lista classificacoes tudo a 0
9       classificacoes = [0] * len(coluna1)
10      #percorrer cada mensagem do conjunto de teste
11      for i in range(len(v)):
12          aux = 0
13          #fazer o produto interno de theta com v
14          for j in range(len(v[i])):
15              aux = aux + theta[j] * v[i][j]
16          aux = aux + thetaZero
17          #se for positivo é classificado como ham, se der negativo é classificado como spam
18          if(aux >= 0):
19              classificacoes[i] = 1
20          else:
21              classificacoes[i] = -1
22
23      hC = 0
24      hE = 0
25      sC = 0
26      sE = 0
27      #classificar as classifições: se for 1 é ham, se for -1 é spam
28      for i in range(len(coluna1)):
29          if(coluna1[i] == "ham"):
30              if(classificacoes[i] == 1):
31                  hC = hC + 1
32              else:
33                  hE = hE + 1
34          elif(coluna1[i] == "spam"):
35              if(classificacoes[i] == -1):
36                  sC = sC + 1
37              else:
38                  sE = sE + 1
39
40      print("\nAcabou de verificar as classificações para ", numeroDeEpocas, " numeros de epocas:\n")
41      #mostrar quantos valores de ham e de spam acertou e errou
42      print("Ham corretos: ", hC)
43      print("Ham errados: ", hE)
44      print("Spam corretos: ", sC)
45      print("Spam errados: ", sE)
46      #hC = ham correto = tp = true positive
47      #hE = ham errado = fp = false positive
48      #sC = spam correto = tn = true negative
49      #sE = spam errado = fn = false negative
50      valores = [hC, hE, sC, sE]
51      #retorna os valores calculados
52      return valores
```

A função funcaoValidacao é a função que vai testar o algoritmo com o p, com o theta e com o thetaZero que foram definidos no conjunto de treino em conjunto com o a que é o melhor valor de t que foi definido no conjunto de teste. É chamada a função calcularXi para calcular as frequências absolutas de cada palavra em cada mensagem do conjunto de validação. Depois é chamada a função calcularClassificacoes que já foi falada anteriormente.

```
54  #funcao que vai testar o algoritmo com o todos os valores de todos os calculos que foram reunidos até agora
55  def funcaoValidacao(validacao, valores):
56      coluna1 = validacao[0]
57      coluna2 = validacao[1]
58      p = valores[0]
59      conjuntoDeThetas = valores[1]
60
61      #chama a funcao calcularXi que recebe o p e o coluna2 e calcula as frequencias absolutas
62      #de cada palavra em cada mensagem do conjunto de validacao
63      v = calcularXi(p, coluna2)
64
65      numeroDeEpocas = valores[2]
66      t = valores[3]
67
68      #a partir dos valores retornados da funcaoTeste já se sabe quais são os melhores thetas e thetaZeros
69      theta = conjuntoDeThetas[t.index(numeroDeEpocas)][0]
70      thetaZero = conjuntoDeThetas[t.index(numeroDeEpocas)][1]
71
72      #chama uma funcao que calcula os valores: tp, fp, tn e fn e retorna os resultados
73      calcularClassificacoes(theta, thetaZero, numeroDeEpocas, v, coluna1)
74
```

5. Outputs
    a. Sudoku

```
In [1]: runfile('C:/Users/Daniel/Desktop/Trabalho de IA/sudoku_com_ac3.py', wdir='C:/Users/Daniel/Desktop/Trabalho de IA')
[0, 0, 3, 0, 2, 0, 6, 0, 0]
[9, 0, 0, 3, 0, 5, 0, 0, 1]
[0, 0, 1, 8, 0, 6, 4, 0, 0]
[0, 0, 8, 1, 0, 2, 9, 0, 0]
[7, 0, 0, 0, 0, 0, 0, 0, 8]
[0, 0, 6, 7, 0, 8, 2, 0, 0]
[0, 0, 2, 6, 0, 9, 5, 0, 0]
[8, 0, 0, 2, 0, 3, 0, 0, 9]
[0, 0, 5, 0, 1, 0, 3, 0, 0]

Aspecto final do Sudoku:

[[4], [8], [3], [9], [2], [1], [6], [5], [7]]
[[9], [6], [7], [3], [4], [5], [8], [2], [1]]
[[2], [5], [1], [8], [7], [6], [4], [9], [3]]
[[5], [4], [8], [1], [3], [2], [9], [7], [6]]
[[7], [2], [9], [5], [6], [4], [1], [3], [8]]
[[1], [3], [6], [7], [9], [8], [2], [4], [5]]
[[3], [7], [2], [6], [8], [9], [5], [1], [4]]
[[8], [1], [4], [2], [5], [3], [7], [6], [9]]
[[6], [9], [5], [4], [1], [7], [3], [8], [2]]

In [2]:
```

b. Naive Bayes

```
In [1]: runfile('C:/Users/Daniel/Desktop/Trabalho de IA/naive_bayes.py', wdir='C:/Users/Daniel/Desktop/Trabalho de IA')

Acabou de ler dados do ficheiro.


    CONJUNTO DE TREINO


Acabou de calcular as mensagens.


Acabou de calcular as palavras das mensagens.


Acabou de normalizar o P


    CONJUNTO DE TESTE


Acabou de classificar as mensagens


Acabou de verificar as classificações com c igual a  0.05 :

Ham corretos:  708
Ham errados:  20
Spam corretos:  104
Spam errados:  3
Erradas:  23
Accuracy:  0.9724550898203593
Error Rate:  0.027544910179640718
Precisão:  0.027544910179640718

Acabou de classificar as mensagens


Acabou de verificar as classificações com c igual a  0.1 :

Ham corretos:  712
Ham errados:  16
Spam corretos:  104
Spam errados:  3
Erradas:  19
Accuracy:  0.9772455089820359
Error Rate:  0.022754491017964073
Precisão:  0.022754491017964073

Acabou de classificar as mensagens
```

```
Acabou de verificar as classificações com c igual a  0.5 :

Ham corretos:  722
Ham errados:  6
Spam corretos:  104
Spam errados:  3
Erradas:  9
Accuracy:  0.9892215568862276
Error Rate:  0.010778443113772455
Precisão:  0.010778443113772455

Acabou de classificar as mensagens


Acabou de verificar as classificações com c igual a  1 :

Ham corretos:  724
Ham errados:  4
Spam corretos:  104
Spam errados:  3
Erradas:  7
Accuracy:  0.9916167664670659
Error Rate:  0.008383233532934131
Precisão:  0.008383233532934131

Acabou de classificar as mensagens


Acabou de verificar as classificações com c igual a  5 :

Ham corretos:  727
Ham errados:  1
Spam corretos:  99
Spam errados:  8
Erradas:  9
Accuracy:  0.9892215568862276
Error Rate:  0.010778443113772455
Precisão:  0.010778443113772455

Acabou de classificar as mensagens

Acabou de verificar as classificações com c igual a  10 :

Ham corretos:  728
Ham errados:  0
Spam corretos:  97
Spam errados:  10
Erradas:  10
Accuracy:  0.9880239520958084
Error Rate:  0.011976047904191617
Precisão:  0.011976047904191617

Acabou de classificar as mensagens


Acabou de verificar as classificações com c igual a  20 :

Ham corretos:  728
Ham errados:  0
Spam corretos:  97
Spam errados:  10
Erradas:  10
Accuracy:  0.9880239520958084
Error Rate:  0.011976047904191617
Precisão:  0.011976047904191617

Acabou de classificar as mensagens


Acabou de verificar as classificações com c igual a  30 :

Ham corretos:  728
Ham errados:  0
Spam corretos:  97
Spam errados:  10
Erradas:  10
Accuracy:  0.9880239520958084
Error Rate:  0.011976047904191617
Precisão:  0.011976047904191617

Acabou de classificar as mensagens
```

```
Acabou de verificar as classificações com c igual a  50 :

Ham corretos:  728
Ham errados:  0
Spam corretos:  97
Spam errados:  10
Erradas:  10
Accuracy:  0.9880239520958084
Error Rate:  0.011976047904191617
Precisão:  0.011976047904191617

Acabou de classificar as mensagens


Acabou de verificar as classificações com c igual a  75 :

Ham corretos:  728
Ham errados:  0
Spam corretos:  97
Spam errados:  10
Erradas:  10
Accuracy:  0.9880239520958084
Error Rate:  0.011976047904191617
Precisão:  0.011976047904191617

Acabou de classificar as mensagens


Acabou de verificar as classificações com c igual a  100 :

Ham corretos:  728
Ham errados:  0
Spam corretos:  96
Spam errados:  11
Erradas:  11
Accuracy:  0.9868263473053892
Error Rate:  0.013173652694610778
Precisão:  0.013173652694610778


Errou menos:  1
Maior Acc:  1
Menor taxa de erros:  1
Maior precisão:  10


Votos:  [0, 0, 0, 3, 0, 1, 0, 0, 0, 0, 0]


O melhor C foi:  1


    CONJUNTO DE VALIDAÇÃO


Acabou de classificar as mensagens


Acabou de verificar as classificações com c igual a  1 :

Ham corretos:  711
Ham errados:  9
Spam corretos:  106
Spam errados:  9

In [2]:
```

c. Perceptrão

```
In [1]: runfile('C:/Users/Daniel/Desktop/Trabalho de IA/perceptrao.py', wdir='C:/Users/Daniel/Desktop/Trabalho de IA')

Acabou de ler dados do ficheiro.


    CONJUNTO DE TREINO


Acabou de calcular as mensagens.


Acabou de calcular o Léxico de palavras (p).


Acabou de calcular o Xi.

Acabou agora de calcular o theta e o thetaZero para  1  número(s) de epocas!

Acabou agora de calcular o theta e o thetaZero para  3  número(s) de epocas!

Acabou agora de calcular o theta e o thetaZero para  5  número(s) de epocas!

Acabou agora de calcular o theta e o thetaZero para  10  número(s) de epocas!


Acabou de calcular os diferentes thetas.


    CONJUNTO DE TESTE


Acabou de calcular o Xi.


Acabou de verificar as classificações para  1  numeros de épocas:

Ham corretos:  732
Ham errados:  4
Spam corretos:  85
Spam errados:  13
Erradas:  17
Accuracy:  0.9796163069544365
Error Rate:  0.02038369304556355
Precisão:  0.02038369304556355

Acabou de verificar as classificações para  3  numeros de épocas:

Ham corretos:  735
Ham errados:  1
Spam corretos:  87
Spam errados:  11
Erradas:  12
Accuracy:  0.9856115107913669
Error Rate:  0.014388489208633094
Precisão:  0.014388489208633094

Acabou de verificar as classificações para  5  numeros de épocas:

Ham corretos:  734
Ham errados:  2
Spam corretos:  86
Spam errados:  12
Erradas:  14
Accuracy:  0.9832134292565947
Error Rate:  0.016786570743405275
Precisão:  0.016786570743405275
```

```
Acabou de verificar as classificações para  10  numeros de épocas:

Ham corretos:  735
Ham errados:  1
Spam corretos:  88
Spam errados:  10
Erradas:  11
Accuracy:  0.986810551558753
Error Rate:  0.013189448441247002
Precisão:  0.013189448441247002


Errou menos:  10
Maior Acc:  10
Menor taxa de erros:  10
Maior precisão:  3


Votos:  [0, 1, 0, 3]


O melhor T foi:  10


    CONJUNTO DE VALIDAÇÃO


Acabou de calcular o Xi.


Acabou de verificar as classificações para  10  numeros de épocas:

Ham corretos:  729
Ham errados:  2
Spam corretos:  98
Spam errados:  6

In [2]:
```

6. Manual do utilizador

Para a correta utilização e funcionamento dos programas é apenas necessário que quando forem postos a correr que estejam no mesmo local do ficheiro “spam.csv”, para assim os programas conseguirem o acessar.

7. Conclusão

Com a conclusão deste Trabalho entendemos melhor a importancia da inteligência artificial e do *machine learning* na tecnologia e compreendemos que poderá vir a ter uma importancia bastante elevada no future.
Foi um projeto longo e difícil, fomos por várias vezes trocando emails com o professor para tirarmos dúvidas do que era o suposto para o trabalho e achamos que cumprimos o objetivo.

8. Webiografia

https://en.wikipedia.org/wiki/Sudoku
https://en.wikipedia.org/wiki/AC-3_algorithm
https://en.wikipedia.org/wiki/Anti-spam_techniques
https://en.wikipedia.org/wiki/Naive_Bayes_classifier