

**DEPARTAMENTO DE CIÊNCIAS E TECNOLOGIAS
LICENCIATURA EM ENGENHARIA INFORMÁTICA**

Trabalho Prático

Relatório de Inteligência Artificial

Alunos: Diogo Paulos, nº 30001058
Luís Candelária, nº 30001572
Ricardo Melo, nº 30000486

Professor: Gonçalo Valadão

Junho de 2020

Lisboa

Índice

Introdução	3
Exercício I.....	4
Exercício II.....	6
Conclusão.....	8

Introdução

O projeto da disciplina de Inteligência Artificial consiste na realização de dois exercícios práticos, utilizando a linguagem de programação Python.

No primeiro exercício foi-nos pedido a resolução de um problema de sudoku, cujos valores de entrada foram-nos disponibilizados pelo professor. Este problema teria que ser resolvido através da consistência de arco, implementando o algoritmo AC-3.

No segundo exercício pretende-se a elaboração de um filtro de spam com base no algoritmo de Naive Bayes e com base no algoritmo do perceptrão. Neste problema utilizámos os documentos classificados de spam que nos foram providenciados pelo professor, de forma a que fosse possível a realização de um conjunto de treino e um conjunto de testes.

Exercício I

No segundo exercício criámos uma variável que designámos de `indexMatrix`. Esta variável contém todos os dados presentes na tabela sudoku do enunciado. Tal como podemos observar na figura 1, substituímos os espaços em branco por zeros, de forma a que facilitasse a execução do algoritmo.

```
indexMatrix = [
    [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],
]
```

Figura 1- `IndexMatrix`

Implementámos uma função `print_index_matrix` cujo objetivo é retornar o output do problema inicial, isto é, estruturar os valores da variável `indexMatrix` numa tabela de sudoku (9*9 constituída por quadrados 3*3). Para que isto fosse possível tivemos que recorrer a ciclos `for` e `if`. A figura 2 mostra a execução da função `print_index_matrix`.

```
=====
| Index Board |
=====
+-----+
| 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 |
+-----+
```

Figura 2- Output da tabela inicial

A função `find_zeros`, tal como o nome indica, tem a finalidade de encontrar todos os zeros da tabela. Esta função é posteriormente chamada na função `solve`, onde irá ter um papel bastante importante, uma vez que tem o objetivo de encontrar a posição de cada espaço a preencher pelo algoritmo de backtracking.

Elaborámos uma função `check` que verifica as posições de todos os elementos das linhas e das colunas da tabela, bem como a posição de cada quadrado (3*3) que podem ser também apelidados de regiões. Esta função impossibilita que qualquer elemento seja repetido mais do que uma vez em cada linha, coluna e região.

A função `solve` invoca a função `find_zeros` com o objetivo de verificar se ainda existem zeros na tabela. No caso já de não existirem, o programa dá-se por terminado. Por outro lado, se ainda existirem zeros na tabela, a função `solve` terá de invocar a função `check` que substituirá cada zero por um número qualquer, dentro do intervalo de números de um a nove. Por fim a função `solve` volta a verificar se ainda existem zeros na tabela e, se não houver, o programa dá por terminado.

Na figura 3 é possível visualizar o output da tabela de sudoku com a resolução correta.

```

=====
|  Result  |
=====

+-----+
| 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 |
+-----+

```

Figura 3- Output do resultado final

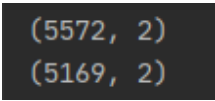
Exercício II

Em relação ao exercício de deteção de spam ou ham através do algoritmo de Naive Bayes e de perceção, começamos por ler o ficheiro CSV no python, onde tivemos que utilizar um encoding específico devido a alguns caracteres que não eram reconhecidos de forma autónoma pelo python, de seguida, decidimos remover as colunas vazias, e substituir o nome das colunas v1 e v2 por spam e text respetivamente.

Para facilitar uma leitura futura decidimos substituir a palavra ham pelo número 0 e spam pelo número 1 na coluna "spam", antiga v1.

Após estas alterações podemos observar que temos 5572 linhas e 2 colunas, agora vamos começar por "limpar" a coluna "text", antiga v2, começando por eliminar as linhas duplicadas.

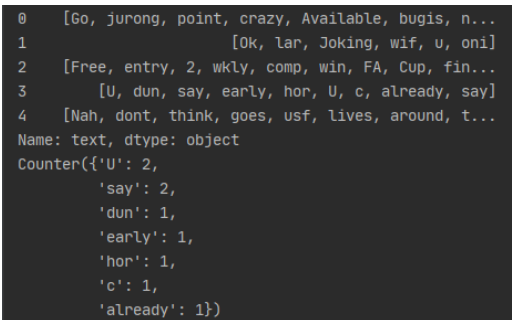
Podemos agora ver as alterações efetuadas, passando de 5572 linhas para 5169 linhas, ou seja, 403 linhas que eram duplicadas foram eliminadas, desta forma aumentamos a performance na análise aos dados, visto que, temos menos linhas para verificar.



```
(5572, 2)
(5169, 2)
```

Figura 4- Antes e depois da eliminação das linhas duplicadas

Agora vamos proceder a uma parte muito importante, vamos remover pontuações, palavras indesejadas e retornar uma lista limpa de palavras ou seja é criada uma espécie de array (matrix) onde cada palavra, (após ser tratada), é separada por vírgulas e assim é possível verificar a frequência de cada palavra, isto é fundamental para os nossos algoritmos de inteligência artificial, visto que estes recebem de entrada dados numéricos e não texto simples, resumindo, cada linha será transformada numa matrix em que cada palavra é uma coluna, onde é feita a contagem da repetição de cada uma das palavras nessa linha.



```
0 [Go, jurong, point, crazy, Available, bugis, n...
1 [Ok, lar, Joking, wif, u, oni]
2 [Free, entry, 2, wkly, comp, win, FA, Cup, fin...
3 [U, dun, say, early, hor, U, c, already, say]
4 [Nah, dont, think, goes, usf, lives, around, t...
Name: text, dtype: object
Counter({'U': 2,
        'say': 2,
        'dun': 1,
        'early': 1,
        'hor': 1,
        'c': 1,
        'already': 1})
```

Figura 5- Exemplo do pré-processamento da linha 3

Na imagem podemos observar esta "limpeza" e separação das palavras e ainda a contagem da frequência de palavras da linha com índice 3.

Através da biblioteca sklearn vamos separar os dados em 80% de treino e 20% de teste, para isso importamos o "train_test_split" da biblioteca "sklearn.model_selection", criamos 4 variáveis "x_train", "x_test", "y_train", "y_test", e definimos um parametro test_size=0.20, ou seja vamos testar de forma aleatoria 20% dos dados.

De seguida vamos usar os algoritmos de inteligencia artificial, treinando-os com os 80% de dados de treino, começamos pelo classificador do Naive Bayes, importamos da biblioteca sklearn.naive_bayes a implementação MultinomialNB, é o mais indicado para este tipo de tarefa que é a contagem de palavras para classificar um texto. Usando o método "fit" presente no MultinomialNB() demos como entrada as variáveis x_train e y_train para assim treinarmos este algoritmo.

De forma bastante similar fizemos o mesmo para o algoritmo de perceptrão, importamos o mesmo da biblioteca "sklearn.linear_model", usamos o método "fit" para treinar este algoritmo dando como entrada as variáveis x_train e y_train.

Depois de treinados os algoritmos, procedemos aos testes com os novos dados, podemos observar os resultados através de métricas de performance, tais como: precisão, recall, f1-score, support, confusion matrix e ainda accuracy, na imagem seguinte podemos verificar qual o algoritmo mais assertivo.

```

----- Naive performance: -----

              precision    recall  f1-score   support

     0       0.99         0.96         0.97         885
     1       0.80         0.93         0.86         149

 accuracy          0.89         0.94         0.92         1034
 macro avg          0.89         0.94         0.92         1034
weighted avg          0.96         0.96         0.96         1034

Confusion Matrix:
[[850  35]
 [ 11 138]]

Accuracy:  0.9555125725338491

----- Perceptron performance: -----

              precision    recall  f1-score   support

     0       0.99         0.98         0.98         885
     1       0.90         0.91         0.91         149

 accuracy          0.94         0.95         0.95         1034
 macro avg          0.94         0.95         0.95         1034
weighted avg          0.97         0.97         0.97         1034

Confusion Matrix:
[[870  15]
 [ 13 136]]

Accuracy:  0.9729206963249516

```

Figura 5-Taxa de sucesso de cada algoritmo

Conclusão

Infelizmente sentimos bastantes dificuldades em aplicar o algoritmo AC-3 no exercício I. Através de exaustivas pesquisas sobre este método potencializamos uma execução correta do exercício, todavia muitas das funções presentes nesse código foram incompreendidas por todos os elementos do grupo. Por isto, optamos por focarmo-nos no algoritmo de *backtracking* que, para além de funcionar corretamente, foi compreendido e elaborado por todos os constituintes.

Em relação ao exercício II- “filtro de spam com base no algoritmo de Naive Bayes e com base no algoritmo do perceptrão” concluímos que o algoritmo de perceptrão foi o mais acertivo em todos as métricas, observando por exemplo a Confusion Matrix, o algoritmo de Naive Bayes previu 46 observações de forma incorreta e 988 de forma correta enquanto o algoritmo de Perceptrão previu 28 observações de forma incorreta e 1006 de forma correta, tendo o Naive Bayes uma taxa de "Accuracy" de 95,6% enquanto o Perceptrão uma taxa de 97,2%.