PS
Perceção Sensorial

2ª Sessão prática

Vamos nesta aula familiarizarmo-nos com os classificadores *K-nearest neighbor* através de um exemplo de *Machine Learning* clássico aproveitando esta oportunidade para experimentarmos algumas funções disponíveis no *Python* e que podem ser muito úteis para a análise de dados.
Vamos então utilizar a base de dados *Abalone* que está disponível no repositório do *Center for Machine Learning and Intelligent* Systems da Universidade da California, USA, no endereço https://archive.ics.uci.edu/ml/datasets/Abalone.
Este exemplo é composto por 4177 exemplos de medições de conchas *Abalone* para as quais foram avaliadas sete medidas contínuas (comprimento, diâmetro, altura, peso total, peso descascado, peso das vísceras, peso da casca), e dois valores discretos (sexo e número de anéis). O número de anéis, somado de 1.5, corresponde ao número de anos da concha. O objetivo é prever o número de anéis com base nos restantes indicadores.

1 – Faça o download do conjunto de dados a partir do site da UCI ou da página Moodle de PS

2 – Utilizando o Python, leia o ficheiro que descarregou. Dado que o ficheiro contém os dados na forma *CSV* sugere-se a utilização da função *read_csv* da biblioteca *Pandas*.

3 – Visualize a *dataframe* resultante. Deverá notar que os nomes das colunas não estão corretos. Poderá (e deverá) corrigir este problema inserindo no ficheiro dos dados uma primeira linha com os nomes das colunas separados por vírgulas. Sugere-se a utilização dos seguintes nomes:

- Sex
- Length
- Diameter
- Height
- Whole_weight
- Shucked_weight
- Viscera_weight
- Shell_weight
- Rings

Deverá assim obter uma *dataframe* com estes nomes nas respetivas colunas.

4 – Depois de obter a *dataframe* com os dados observe a distribuição dos dados pelas 29 classes que são o número de aneis das conchas.
Sugestões:
df['Rings'].value_counts()
df['Rings'].value_counts().sort_index() - para ver ordenado pelo número de rings

5 – Como os modelos que vamos desenvolver nesta aula utilizam exclusivamente dados numéricos, é necessário transformar os valores não numéricos em valores numéricos. Como poderá verificar, todas as colunas são numéricas exceto a coluna *Sex* que é uma string. Assim, transforme os valores desta coluna em 1 se 'M' e 0 de 'F'.
Sugestão: apply(lambda x: 1 if x=='M' else 0)

6 – Como terá constatado, a distribuição dos dados por classes não é nada uniforme, existindo classes com pouquíssimos exemplos. Como é evidente, não é viável trabalhar com classes que são representadas por tão poucos exemplos como é o caso das classes 1, 2, 3, 21, 22, 23, etc. Assim, vamos juntar as classes 1, 2, 3, 4 e 5 numa classe única de identificador 5 e as classes 20 a 29 numa classe de identificador 20.
Para que as classes fiquem com uma numeração "simpática" vamos ainda renumerar todas elas entre 1 e 16 subtraindo 4 ao valor da classe original.
Deveremos então ficar com um conjunto de dados com 16 classes e com a seguinte distribuição de dados:

Classe	Número de exemplos
1	189
2	259
3	391
4	568
5	689
6	634
7	487
8	267
9	203
10	126
11	103
12	67
13	58
14	42
15	32
16	62

Sugestão: usar apply de uma função que faça a transformação do valor da classe.

7 – Depois de criado este conjunto de dados é necessário dividir os dados em *Predictors* (dados que servem para predizer a classe) e *Class* (coluna que indica a classe de cada exemplo).
Os dados de treino podem ser os dados originais fazendo *drop* à coluna da classe (*Rings*) e os dados da classe são simplesmente a coluna *Rings*.

8 – Depois de separados em dois conjuntos há que dividir os dados em conjunto de treino e conjunto de teste. O código abaixo permite fazer essa divisão criando um conjunto de treino com 75% dos dados e um conjunto de teste com os restantes 25% onde X são os dados e Y a classe correspondente.

```
X_train,X_test,y_train,y_test=train_test_split(X,Y,random_state=0)
```

9 – Está então na altura de se testar o primeiro classificador K-NN (K nearest neighbor). Podemos obter este classificador com o código:

```
knn=KNeighborsClassifier(n_neighbors=5)   # KNN com K=5
knn.fit(X_train,y_train)                  # train the classifier
knn.score(X_test,y_test)                  # test the result
```

Nota: se quiser visualizar a configuração do classificar KNN basta visualizar a variável *knn*

10 – Como poderá constatar, a precisão do classificado é muito baixa (na ordem dos 24%). Esta baixa precisão deve-se ao elevado número de classes que foram consideradas (16). Como se pode constatar ma literatura deste exemplo, os resultados obtidos por diversos estudos são na ordem dos 65% mas utilizando apenas três classes: 1..8, 9..10, 11..29. Vamos então refazer o reordenamento das classes para estas três classes que numeraremos de 1 a 3.

11 – Com os dados reorganizados em três classes refaça o classificador e veja qual a precisão obtida com o conjunto de teste e com o conjunto de treino (deverá ser na ordem dos 63%).

12 – Construa a matriz de confusão para o classificador anterior.
Sugestão: utilize a função *confusion_matrix* da biblioteca *sklearn.metrics*

13 – Repita o classificador anterior com K=1. Justifique a precisão obtida com o conjunto de treino e com o conjunto de teste.

14 – Analise a precisão obtida com o conjunto de treino e com o conjunto de teste variando o K entre 1 e 15 (só valores impares, claro). Se possível faça um gráfico com os valores de precisão para ambas as situações em função de K.

Sugestão: vai necessitar da biblioteca pyplot (import matplotlib.pyplot as plt) da qual poderá utilizar a função *plot* para criar os gráficos e a função *show* para mostrar os gráficos produzidos.