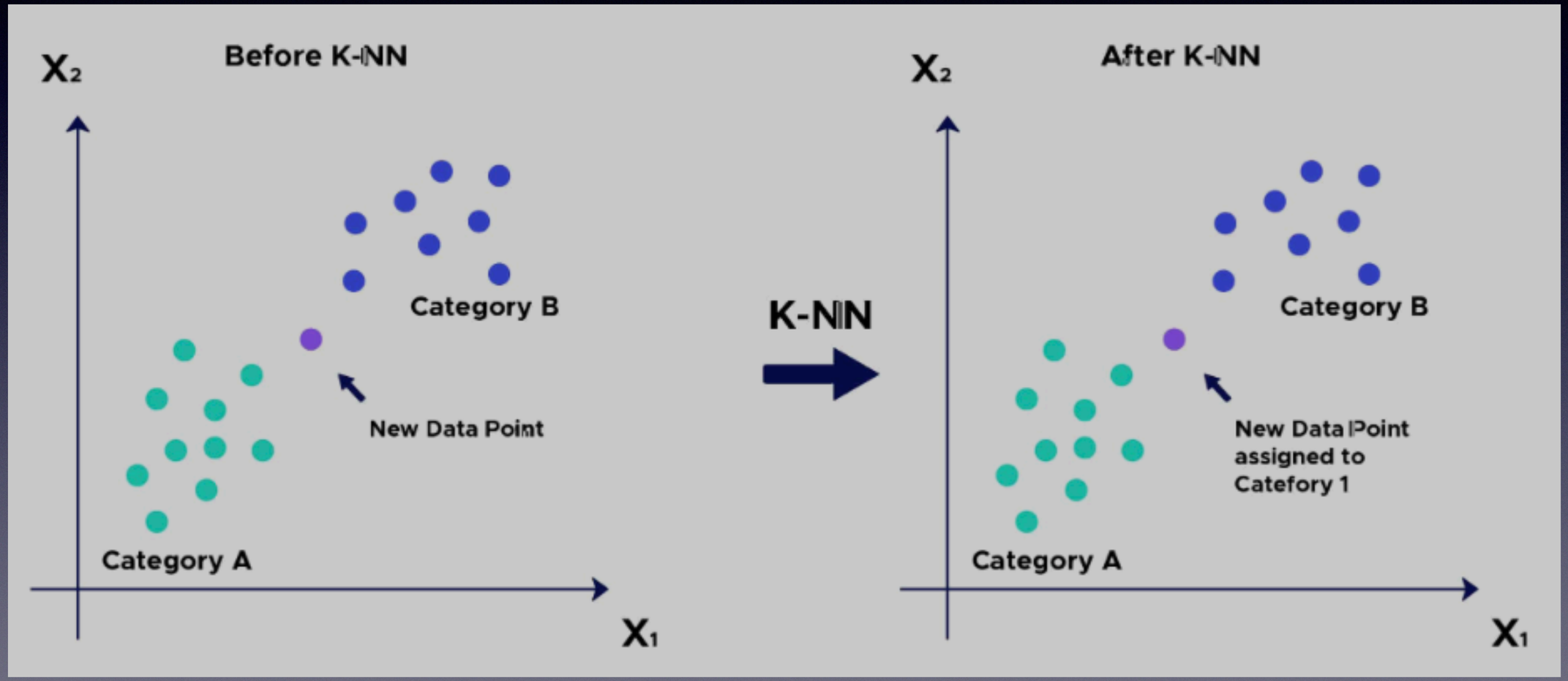
Algoritmo K-nearest Neighbor

Mafalda Aires - 202106550
Gonçalo Monteiro - 202105821

Algoritmo K-NN



Algoritmo K-NN

- No slide anterior, vemos que surge um novo data point no gráfico, o qual precisa de ser associado a uma classe (*Category A* ou *Category B*).
- Chamemos P ao novo data point. Para descobrirmos a classe de P, selecionamos k data points mais próximos de P.
- A classe, tendo em conta os k data points selecionados, que estiver em maioria será a classe escolhida para P.
- Nota: k é um número à nossa escolha, podendo variar entre 1 e o número total de data points no gráfico. Tipicamente, k é um número ímpar.

Algoritmo K-NN

The pseudocode of the k-NN algorithm is presented next. It is possible to see that k-NN does not have an explicit learning phase.

Algorithm K-NN test algorithm.

```
1: INPUT D_train, the training set
2: INPUT D_test, the test set
3: INPUT d, the distance measure
4: INPUT x_i objects in the test set
5: INPUT K, the number of neighbors
6: INPUT n, the number of objects in the test set
7: for all object x_i in D_test do
8:     for all object x_j in D_test do
9:         Find the k objects from D_train closest to x_i according to the chosen
   distance measure d
10:    Assign x_i the class label most frequent in the k closest objects
```

- O K-NN é um algoritmo de classificação bastante simples.
- Este algorItmo prevê a classe de um objeto com base na classe dos k números de objetos mais próximos.
- Assim, o algoritmo possui como único hiper-parâmetro o valor de k.
- O K-NN é baseado em ”lazy learning”. Isto significa que não existe um verdadeiro processo de treinamento de dados. Pelo contrário, o algoritmo guarda apenas em memória os objetos do training-set comparando-os posteriormente ao objeto que queremos prever.

O que pretendemos analisar

- O objetivo do trabalho é avaliar como se comporta o K-NN em datasets com diferentes características.

- Tentar compreender o modelo empiricamente e, com base nas observações, fazer uma alteração que torne o algoritmo mais robusto.

- Iremos, então comparar K-NN original ao K-NN com a alteração.

- Para isso, temos que dividir o trabalho em 2 partes. Como teremos que fazer alterações ao K-NN, então este algoritmo tem que ser implementado de raiz para permitir a sua manipulação. Na primeira parte, iremos implementar K-NN de raiz e compará-lo ao K-NN do *scikit-learn*. Na segunda parte, vamos analisar o K-NN para os 3 datasets e tentar melhorar o algoritmo.

- Por fim, iremos concluir que o algoritmo alterado tem uma maior “accuracy” em relação ao algoritmo original.

Características do K-NN

1. A performance do K-NN depende muito do valor de k. Se k for muito grande, estamos a incluir objetos muito diferentes do objeto a ser previsto. Por outro lado, se k for muito pequeno, apenas objetos muito semelhantes serão incluídos.

2. Como o K-NN é um algoritmo que se baseia em distâncias, então um pouca informação pode impedir uma boa classificação, já que esta pode ser facilmente definida pelos outliers. Por outro lado, datasets muito grandes podem promover uma execução demorada do K-NN, já que terão que ser calculadas muitas distâncias para uma previsão.

Características do K-NN

3. Ainda assim o K-NN apresenta muitas outras vantagem: É um algoritmo muito simples; Com bons resultados de previsão em diferentes tarefas; Se alterações forem feitas ao dataset, estas serão imediatamente incorporadas no fitting model.

Parte 1

Implementação do K-NN

In [2]:
```
def euclidean_distance(x,y):
    return np.sqrt(np.sum((x-y)**2))
```

In [3]:
```
class KNN(BaseEstimator, ClassifierMixin):
    def __init__(self,k):
        self.k = k


    def fit(self,X,y):
        self.X_train = X
        self.y_train = y


    def predict(self,X):
        predicted_labels=[self._predict(x) for x in X]
        return np.array(predicted_labels)


    def _predict(self,x):
        distances = [euclidean_distance(x,x_train) for x_train in self.X_train]

        k_idx = np.argsort(distances)[: self.k]

        k_neighbor_labels = [self.y_train[i] for i in k_idx]

        most_common = Counter(k_neighbor_labels).most_common(1)

        return most_common[0][0]


    def accuracy(self,y_true, y_pred):
        return np.sum(y_true == y_pred) / len(y_true)
```

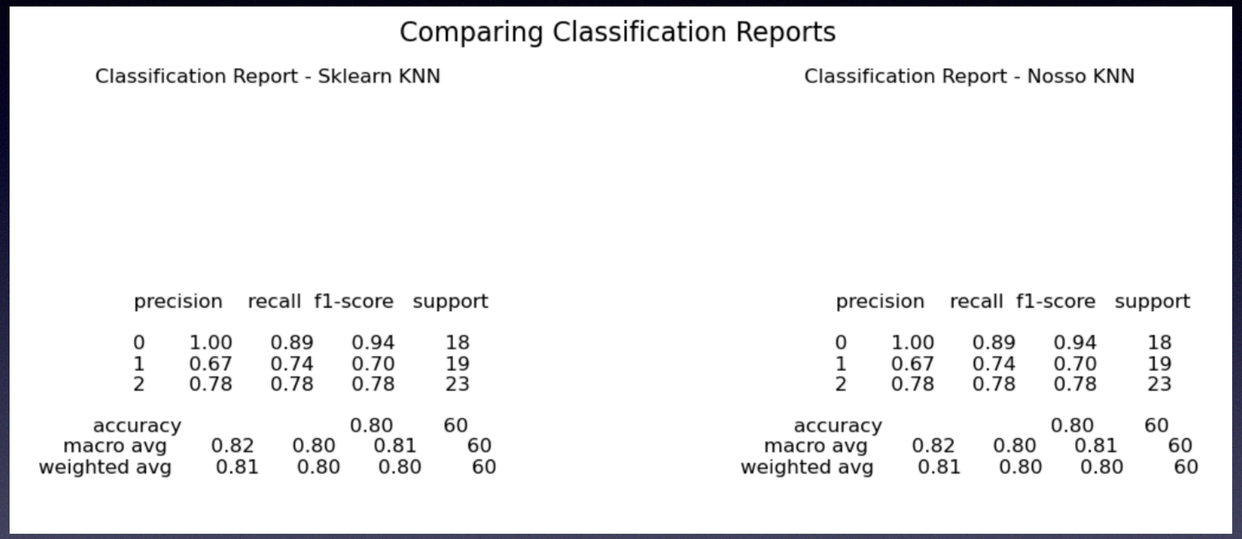
Comparar o K-NN implementado com o K-NN do *scikit-learn*

- A comparação feita entre o K-NN implementado e o K-NN do s*cikit-learn* tem como objetivo verificar a credibilidade do algoritmo (K-NN) implementado.

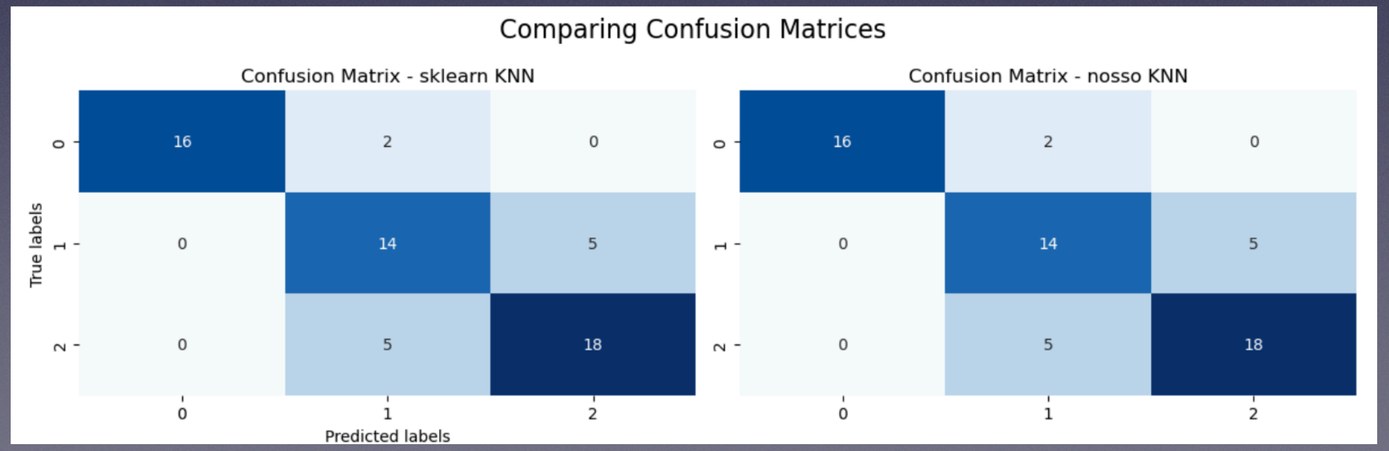
- De facto ambos os algoritmos têm uma performance muito semelhante ou até igual, o que nos leva a assumir que podemos usar o algoritmo implementado para as previsões e respetivas avaliações que serão feitas sobre o algoritmo K-NN.

- Para além disso, como o K-NN terá que sofrer alterações, podemos também confiar na “accuracy” do novo K-NN alterado.

Comparar o K-NN implementado com o K-NN do *scikit-learn*




```
Precisão do meu KNN:  0.8
Precisão do KNN do sklearn:  0.8
```



Parte 2

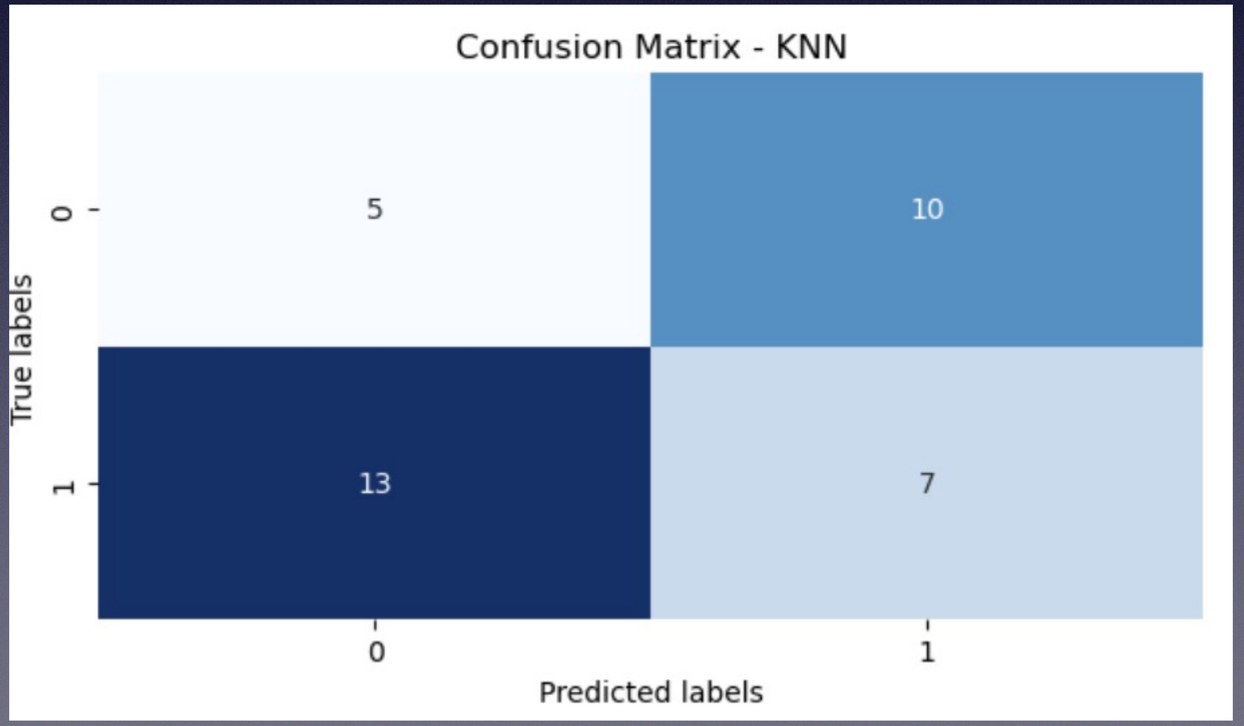
Dataset 1

O nosso primeiro dataset é *breast_cancer_coimbra.csv* , com 117 registos. Com este dataset, o modelo de machine learning terá que classificar com base em 9 atributos se uma pessoa é doente (2) ou saudável (1).

Age	BMI	Glucose	Insulin	HOMA	Leptin	Adiponectin	Resistin	MCP.1	Classification
48	23.5	70	2.707	0.467408667	8.8071	9.7024	7.99585	417.114	1
83	20.69049454	92	3.115	0.706897333	8.8438	5.429285	4.06405	468.786	1
82	23.12467037	91	4.498	1.009651067	17.9393	22.43204	9.27715	554.697	1
68	21.36752137	77	3.226	0.612724933	9.8827	7.16956	12.766	928.22	1
86	21.11111111	92	3.549	0.8053864	6.6994	4.81924	10.57635	773.92	1

Dataset 1

Por análise da Confusion Matrix, podemos facilmente reconhecer que a “accuracy” do modelo não é muito satisfatória, mas porque?



Dataset 1

- Outra coisa que sabemos sobre este dataset é que é equilibrado, ou seja, as classes possuem o mesmo número de registos. Portanto, não há muita coisa que possa explicar a má performance do algoritmo a não ser o facto de o dataset ser relativamente pequeno, com apenas 117 registos.

- O K-NN, sendo um algoritmo simples de classificação baseado em distâncias é relativamente sensível a outliers, por isso sendo este dataset pequeno, a performance do K-NN é severamente afetada negativamente. Seria, portanto, de esperar que atribuir pesos às distâncias mitigasse o efeito dos outliers e melhorasse a performance da classificação.

Weighted K-NN

In [11]:

```python
class WeightedKNN(BaseEstimator, ClassifierMixin):
    def __init__(self,k):
        self.k = k


    def fit(self,X,y):
        self.X_train = X
        self.y_train = y
        self.std_devs = np.std(X, axis=0)
        self.weights = self.weights = (1 / self.std_devs) / np.sum(1 / self.std_devs)


    def predict(self, X):
        y_pred = []
        for sample in X:
            distances = np.sqrt(np.sum(((self.X_train - sample) * self.weights) ** 2, axis=1))
            sorted_indices = np.argsort(distances)
            k_indices = sorted_indices[:self.k]
            k_nearest_classes = self.y_train[k_indices]
            unique_classes, counts = np.unique(k_nearest_classes, return_counts=True)
            most_frequent_class = unique_classes[np.argmax(counts)]
            y_pred.append(most_frequent_class)
        return y_pred


    def accuracy(self,y_true, y_pred):
        return np.sum(y_true == y_pred) / len(y_true)
```

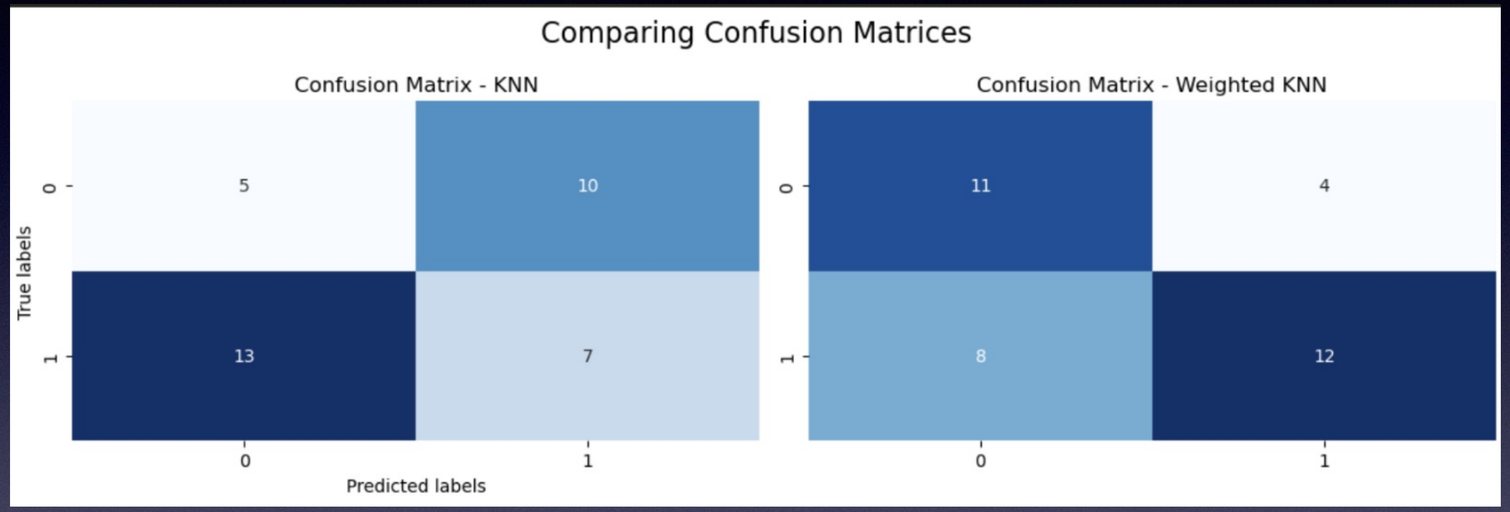
Weighted K-NN

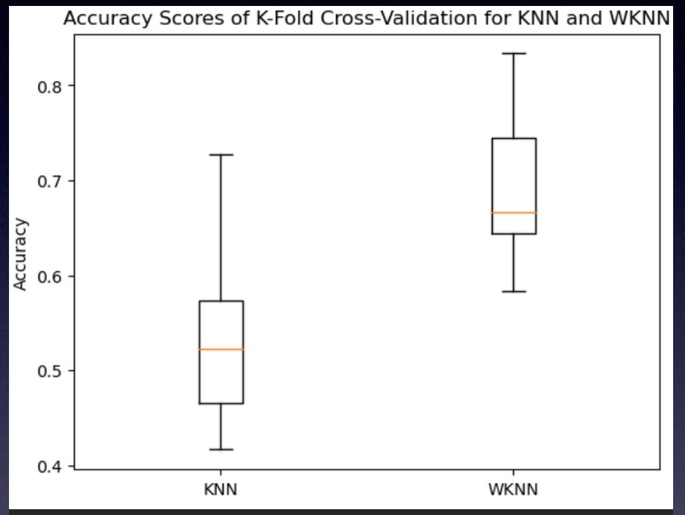
- O Weighted K-NN distingue-se se do K-NN na utilização de pesos para calcular a distância euclidiana.

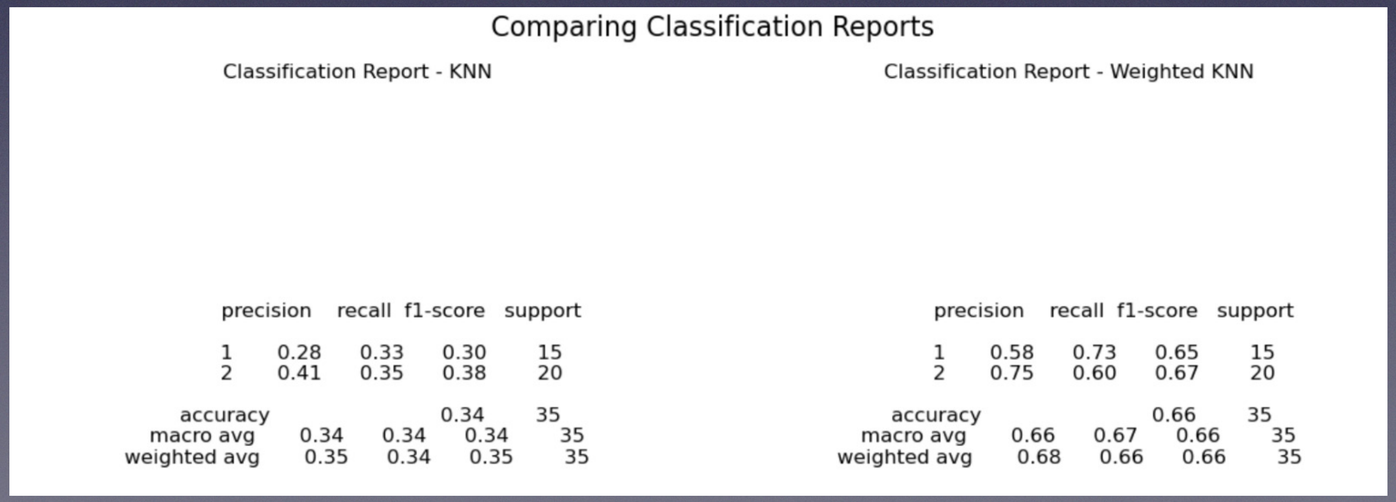
- Neste caso, o peso de uma distância para um registo baseia-se no próprio desvio padrão. Isto é, distâncias muito afastadas da média terão menor peso no cálculo da distância ao passo que distâncias associadas a um menor desvio padrão terão mais peso.

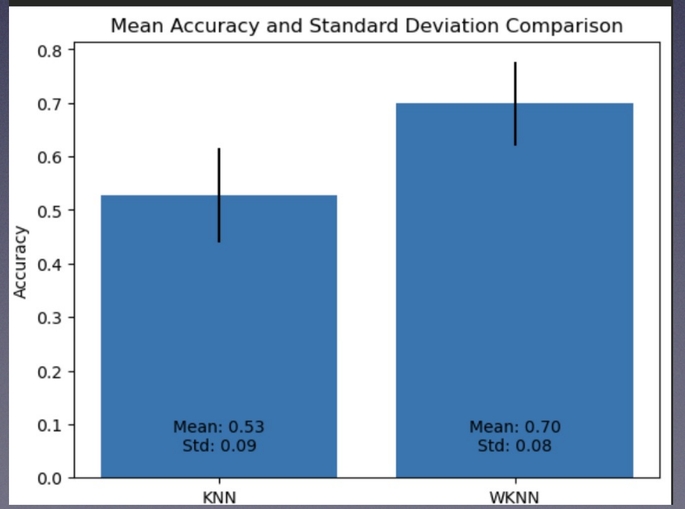
- No algoritmo, *self.std_devs* é responsável por calcular o desvio padrão e *self.weights* é responsável por determinar o peso associado às distâncias, calculando o inverso do desvio padrão para a respetiva distância.

De facto todas as análises corroboram isso:









Dataset 2

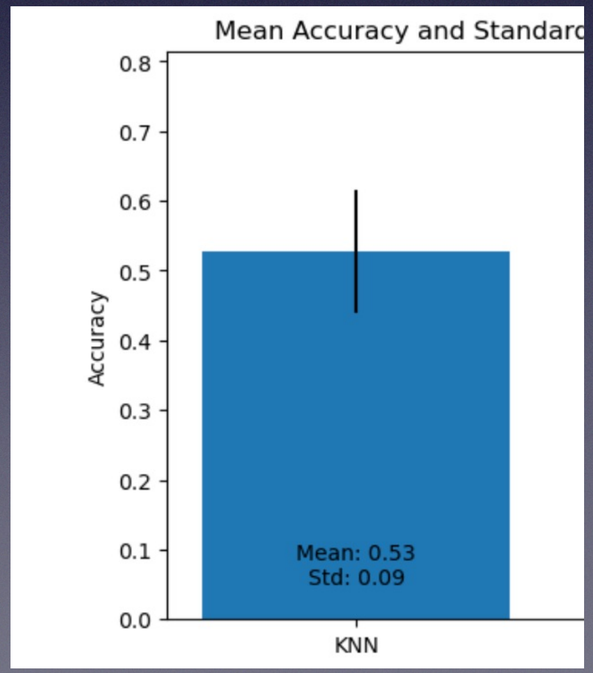
Out[52]:

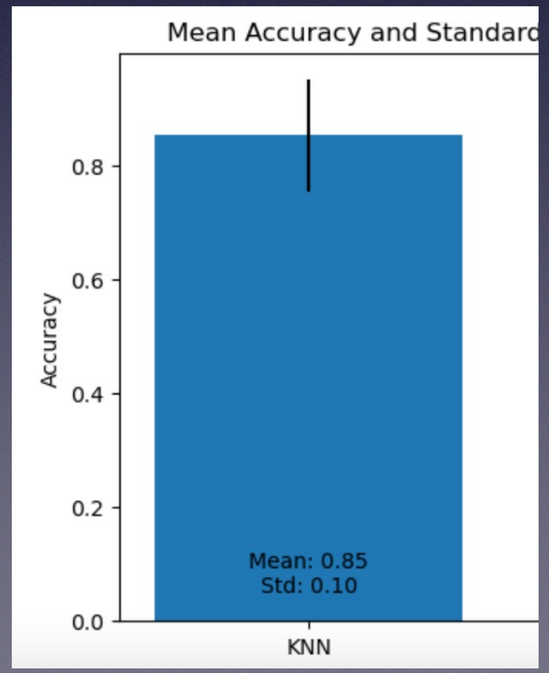
	Bit-0	Bit-1	Bit-2	Bit-3	Bit-4	Bit-5	Bit-6	Bit-7	Bit-8	Bit-9	class
0	0	1	1	1	0	1	0	0	0	1	1
1	1	0	1	1	1	0	1	0	1	1	1
2	1	0	0	1	0	1	0	1	0	1	1
3	0	0	1	0	0	1	0	0	1	0	1
4	1	0	1	0	0	1	0	0	1	0	1

- O dataset 2 apresenta a “M-of-N rule”. O alvo é 1 se M bits são 1.
- É um dataset com muita informação (1324 registos).

Dataset 2 vs Dataset 1

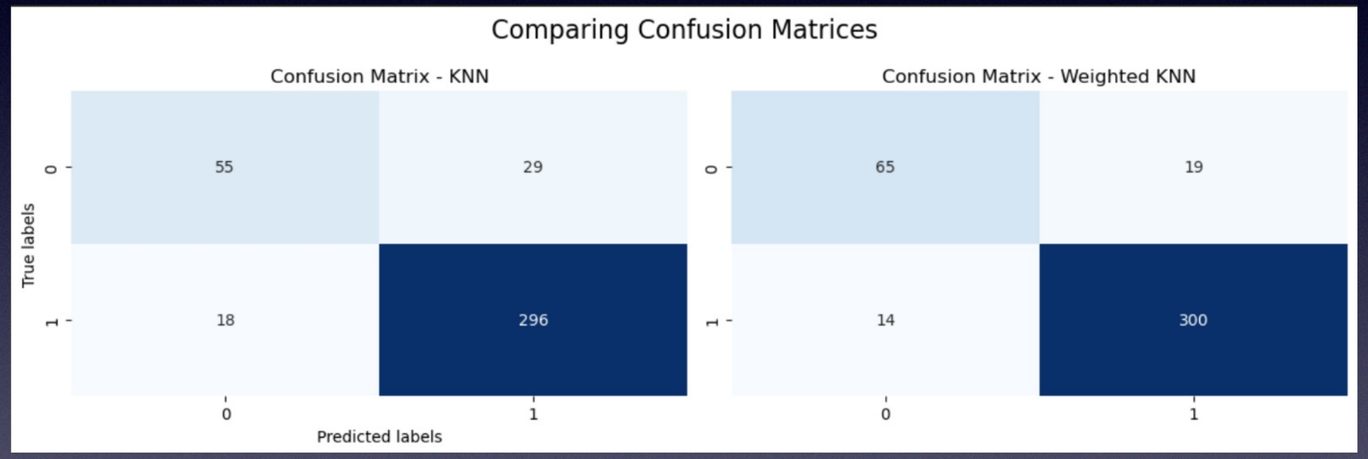
O dataset 2 descreve regras lógicas onde os atributos apenas são 1 ou 0, portanto é uma dataset que não apresenta muita variação atributos para dificultar a classificação: assim, se nos focarmos apenas no K-NN sem pesos notamos uma maior “accuracy” no dataset 2 em relação ao dataset 1 pelo simples facto de ter mais informação.

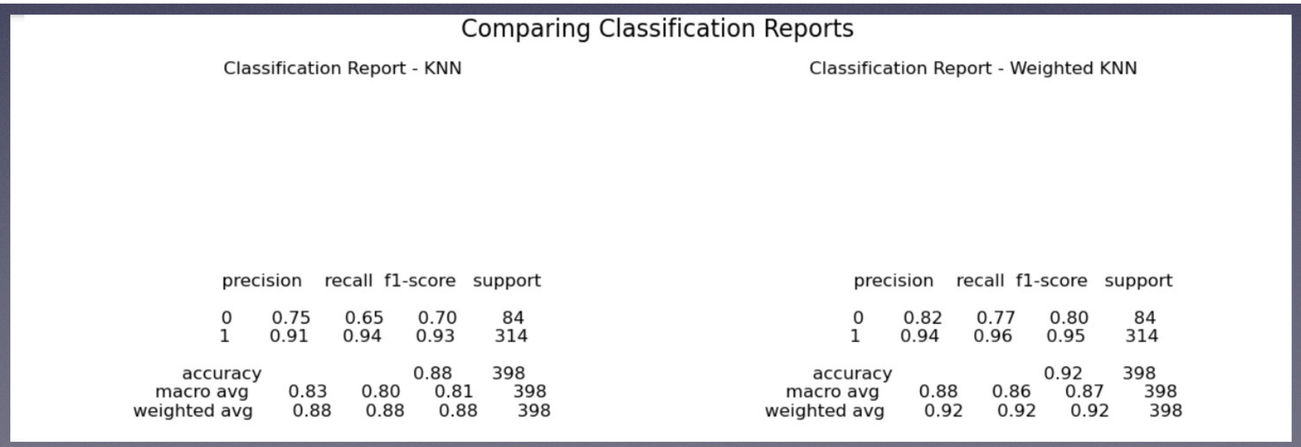


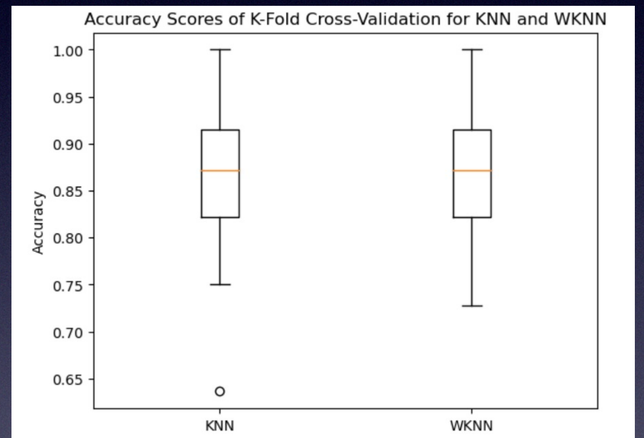


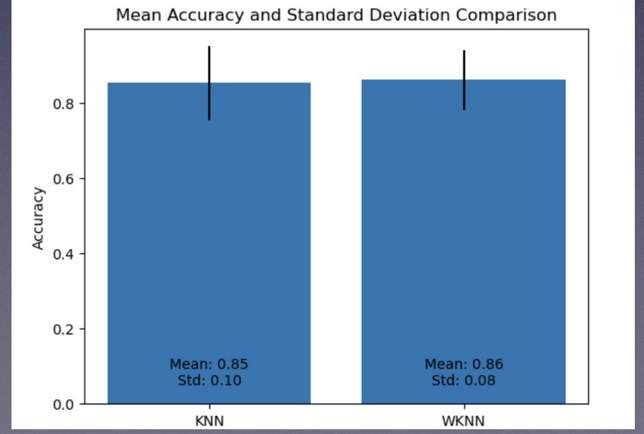
Dataset 2

Ainda assim, verificamos que o K-NN com pesos tem uma melhor performance, já que, aqui, pesar as distâncias é uma mais valia.









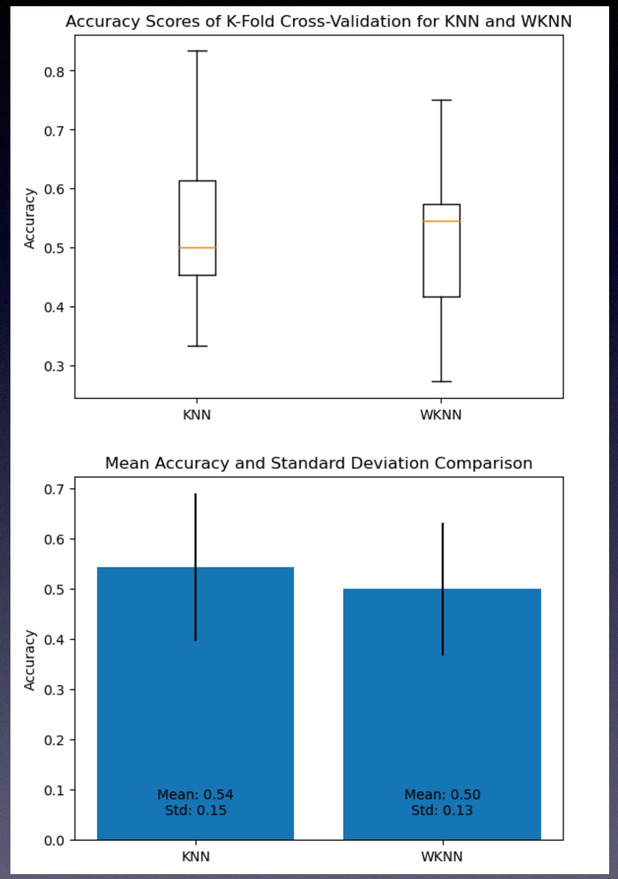
Dataset 3

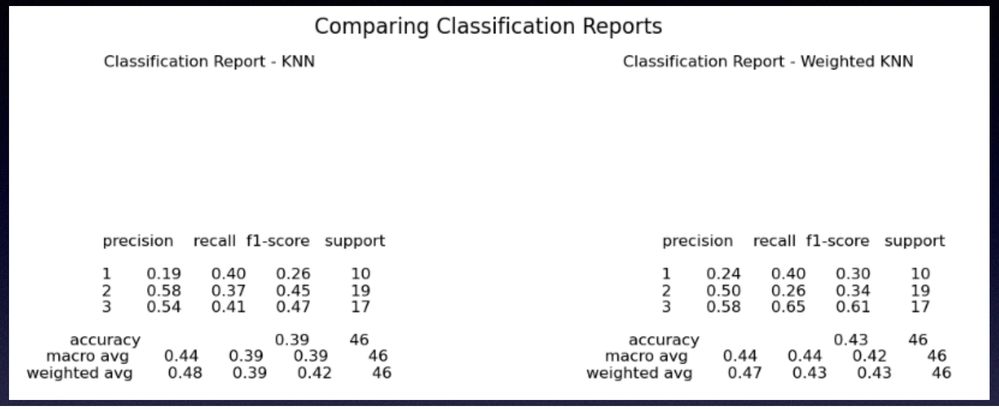
Out[111]:

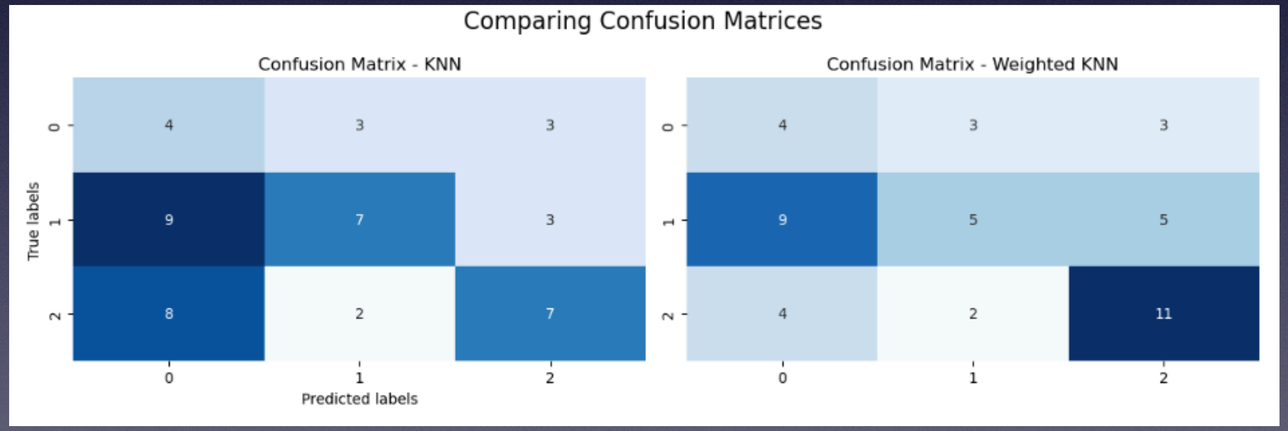
	Whether_of_not_the_TA_is_a_native_English_speaker	Course_instructor	Course	Summer_or_regular_semester	Class_size	Class_attribute
0	1	23.0	3.0	1	19.0	3
1	2	15.0	3.0	1	17.0	3
2	1	23.0	3.0	2	49.0	3
3	1	5.0	2.0	2	33.0	3
4	2	7.0	11.0	2	55.0	3

- De acordo com 5 atributos, avalia o nível de inglês de uma pessoa: Baixo(1) , Médio(2) ou Alto (3).
- A grande diferença deste dataset para os anteriores é o facto de este possuir 3 target classes. Escolhemos este dataset para ver o comportamento do K-NN quando a classificação da tarefa não é binária.

Dataset 3







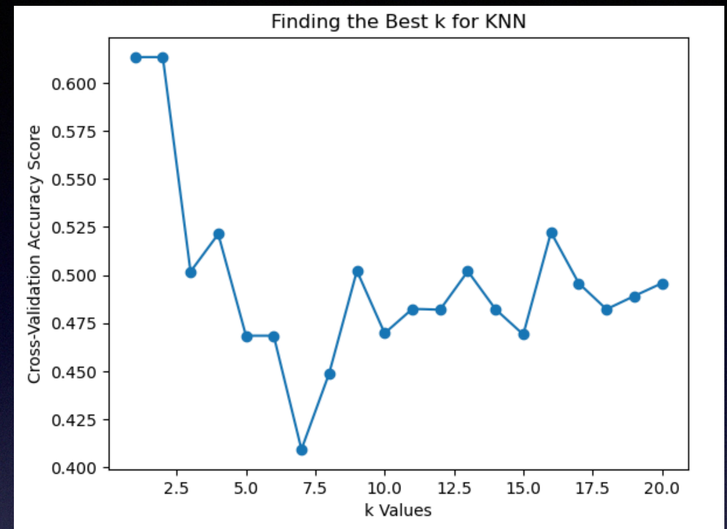
- Com estes gráficos, podemos verificar que a accuracy do K-NN com pesos é ligeiramente maior que o K-NN sem pesos. Porém, ao contrário dos outros datasets, quando fazemos uma “Accuracy Scores of K-Fold Cross-Validation” para 10 folds o K-NN simples sai-se ligeiramente melhor.

- É importante informar que a comparação feita foi com k=3 para ambos os algoritmos.

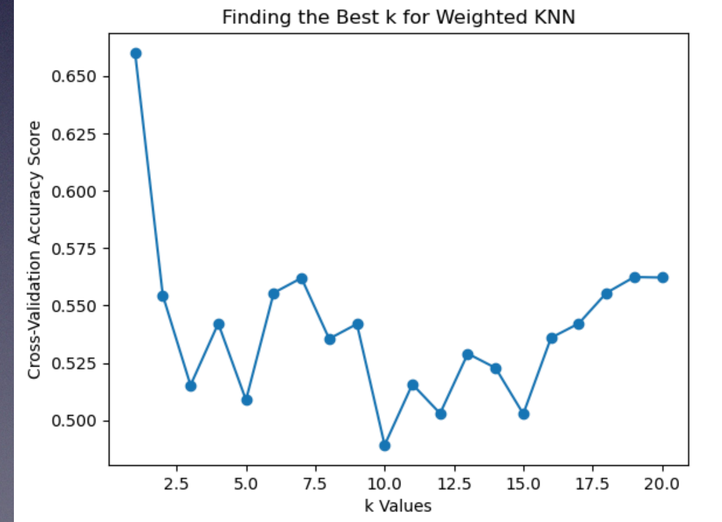
Dataset 3

- Sabemos que o algoritmo K-NN depende muito do valor de k e, se analisarmos o gráfico ao lado, para k=3 de facto o K-NN simples é melhor. O problema é que 3 é um mau valor de k para ambos, uma vez que ambos tem uma “accuracy” baixa. Por isso, se analisarmos para o melhor k (k=1), onde a “accuracy” é maior tanto no K-NN sem pesos como no K-NN com pesos, o K-NN com pesos continua a ter melhor performance.

- No fundo, o que queremos concluir é que o K-NN com pesos continua a ser melhor do que o K-NN sem pesos e que o K-NN continua a ser um modelo de classificação válido mesmo para problemas não binários.

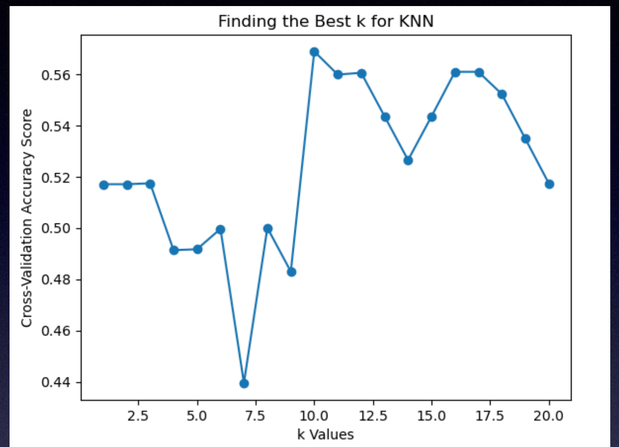



```
Best k for knn:  1
```

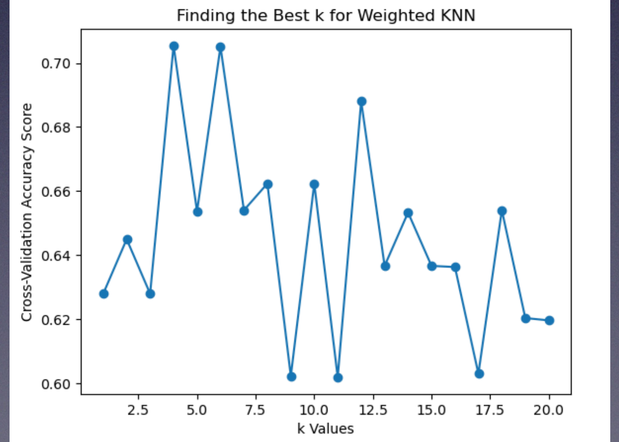



```
Best k for wknn:  1
```

Importância do valor de k



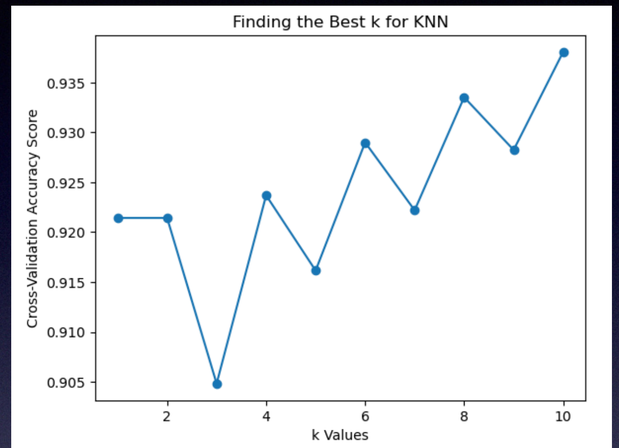

```
Best k for knn:  10
```



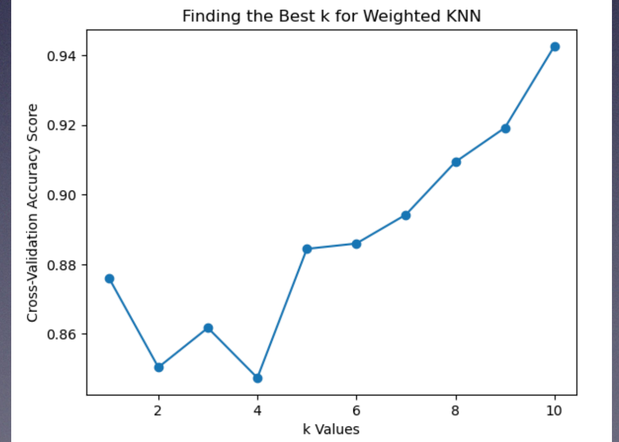

```
Best k for wknn:  4
```

Dataset 1

Para diferentes valores de k temos diferentes valores de “accuracy”.




```
Best k for knn:  10
```




```
Best k for wknn:  10
```

Dataset 2