Week 10 – Session 3

DW 10.009 – Introduction to Python Programming

Week 10 Breakdown

- Session 1: Introduction to Data Science
  - Introduction to Numpy
  - Core ideas about data science
  - Data Manipulation and Visualization
- Session 2: Introduction to regression
  - Key parameters for regression
  - Linear regression
  - Multiple linear regression
- Session 3: About classification
  - Key parameters for classification
  - K-NN Classification

Week 10 Breakdown

- Session 1: Introduction to Data Science
  - Introduction to Numpy
  - Core ideas about data science
  - Data Manipulation and Visualization
- Session 2: Introduction to regression
  - Key parameters for regression
  - Linear regression
  - Multiple linear regression
- Session 3: About classification
  - Key parameters for classification
  - K-NN Classification

Let us practice a bit

Problem set 10 – Q2 & Q3 (5-number summary and normalization)

Q2: Five-number summary

- The **five-number summary**, is an informative function about data, listing
  - The **minimal** value in a given array
  - The **maximal** value in a given array
  - The **median** value in a given array
  - The **first quarter percentile** value in a given array
  - The **third quarter percentile** value in a given array

For Q2: Min, Max, Mean, Median, Percentile

- Numpy has functions for finding the
  - **Minimal** value,
  - **Maximal** value,
  - **Mean** value,
  - **Median** values,
  - Etc.
- For any given array, containing data.

```
# Minimal value
print(np.min(matrix))
```

```
5.052808826566668e-06
```

```
# Maximal value
print(np.max(matrix))
```

```
0.999945892478096
```

```
# Mean value
print(np.mean(matrix))
```

```
0.5024344386819765
```

```
# Median value
print(np.median(matrix))
```

```
0.5042077048148175
```

```
# n%-percentile value: value of the element,
# which is greater than n% of the samples in matrix
n = 25
print(np.percentile(matrix, n))
```

```
0.2509906081803283
```

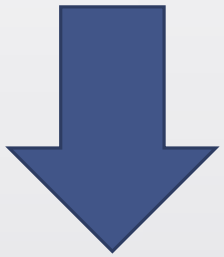
Let us practice a bit

Problem set 10 – Q2 & Q3 (5-number summary and normalization)

Q3: data normalization

- Data normalization is a typical operation in Machine Learning.
  - It re-scales the data, so that the minimal value in the data will become 0.
  - And the maximal value will become 1.

Input x1	Input x2
1	10
2	6
3	2
4	4
5	0

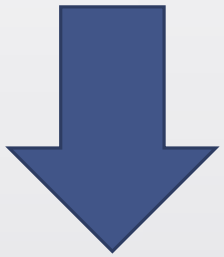


Input x1 (normalized)	Input x2 (normalized)
0	1
0.25	0.6
0.5	0.2
0.75	0.4
1	0

Q3: data normalization

- Data normalization is a typical operation in Machine Learning.
  - It re-scales the data, so that the minimal value in the data will become 0.
  - And the maximal value will become 1.
- Q3: write a function that receives a data array, and normalize the columns of the array one-by-one.

Input x1	Input x2
1	10
2	6
3	2
4	4
5	0



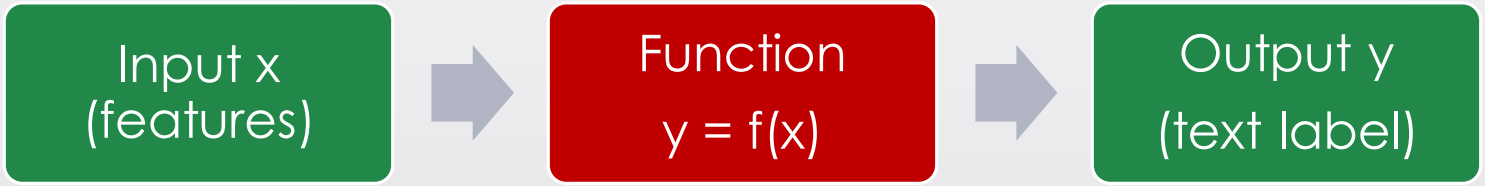
Input x1 (normalized)	Input x2 (normalized)
0	1
0.25	0.6
0.5	0.2
0.75	0.4
1	0

Introduction to classification with KNN

Key concepts about linear regression

Classification: core idea

- Today, we investigate a second type of data science problems, which is called a classification problem



- To demonstrate the core idea behind the KNN classification algorithm, we will use a dataset consisting of weights, heights and gender of humans.

Dataset: weight, height and gender

- To demonstrate, we use a dataset containing:
  - A feature **x1** containing **heights** in centimeters
  - A feature **x2** containing **weights** in kilograms
  - A label **y** containing a **gender** (**0** for '**Female**' or **1** for '**Male**')

```
1  print(data.shape)
2  print("Number of women entries:", nw)
3  print("Number of men entries:", nm)
4  print(data_features)
5  print(data)
6
```

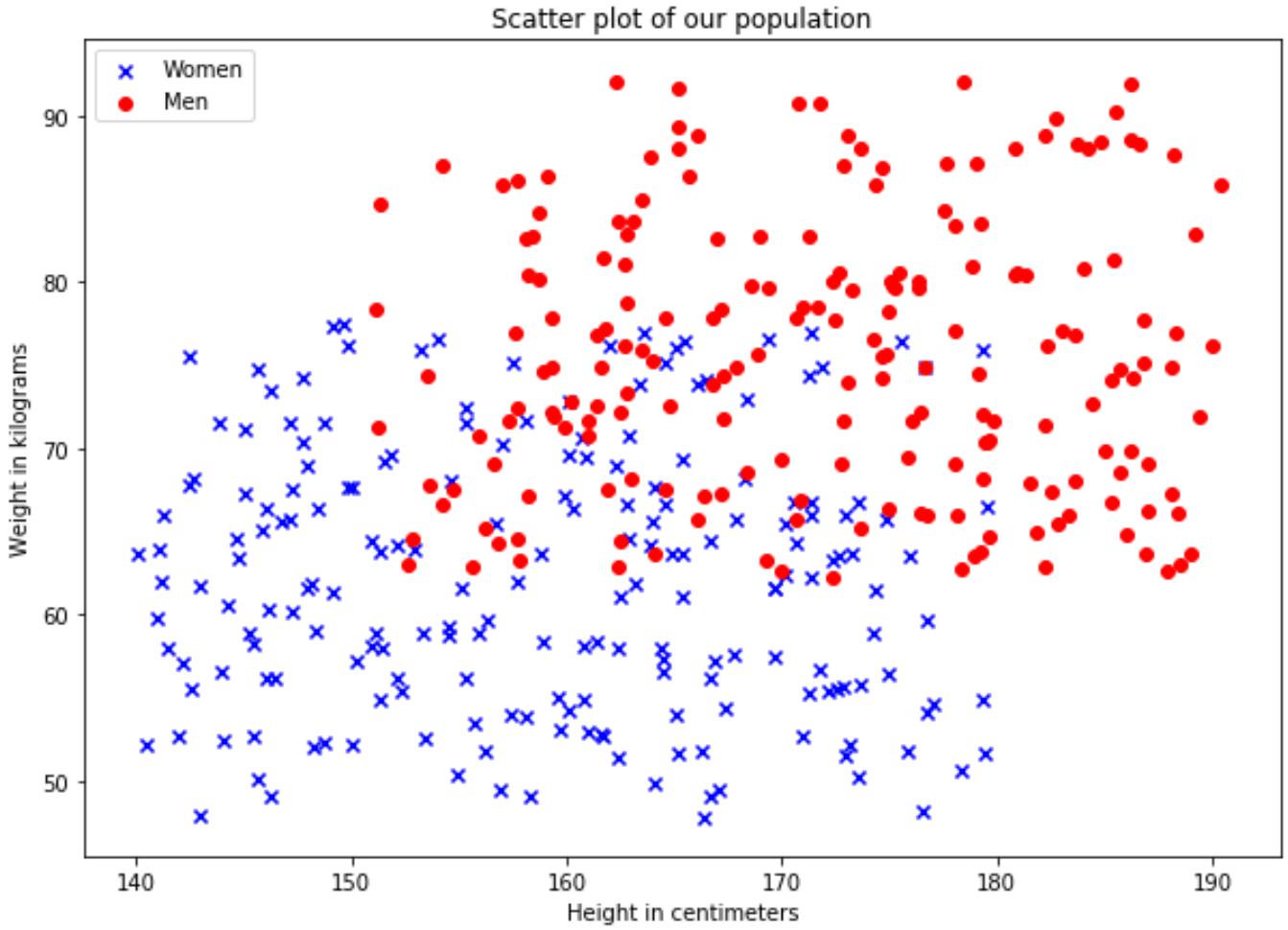
```
(400, 3)
Number of women entries: 200
Number of men entries: 200
['Height', 'Weight', 'Gender']
[[ 0.66799205 55.7          0.        ]
 [ 0.67793241 58.8          0.        ]
 [ 0.5526839  65.7          0.        ]
 ...
 [ 0.5944334  62.6          1.        ]
 [ 0.3359841  85.8          1.        ]
 [ 0.48707753 67.5          1.        ]]
```

Scatter plot

- As usual, the first step should be to make a scatter plot of our data.

- Here we can see two clusters of points in our scatter plot.

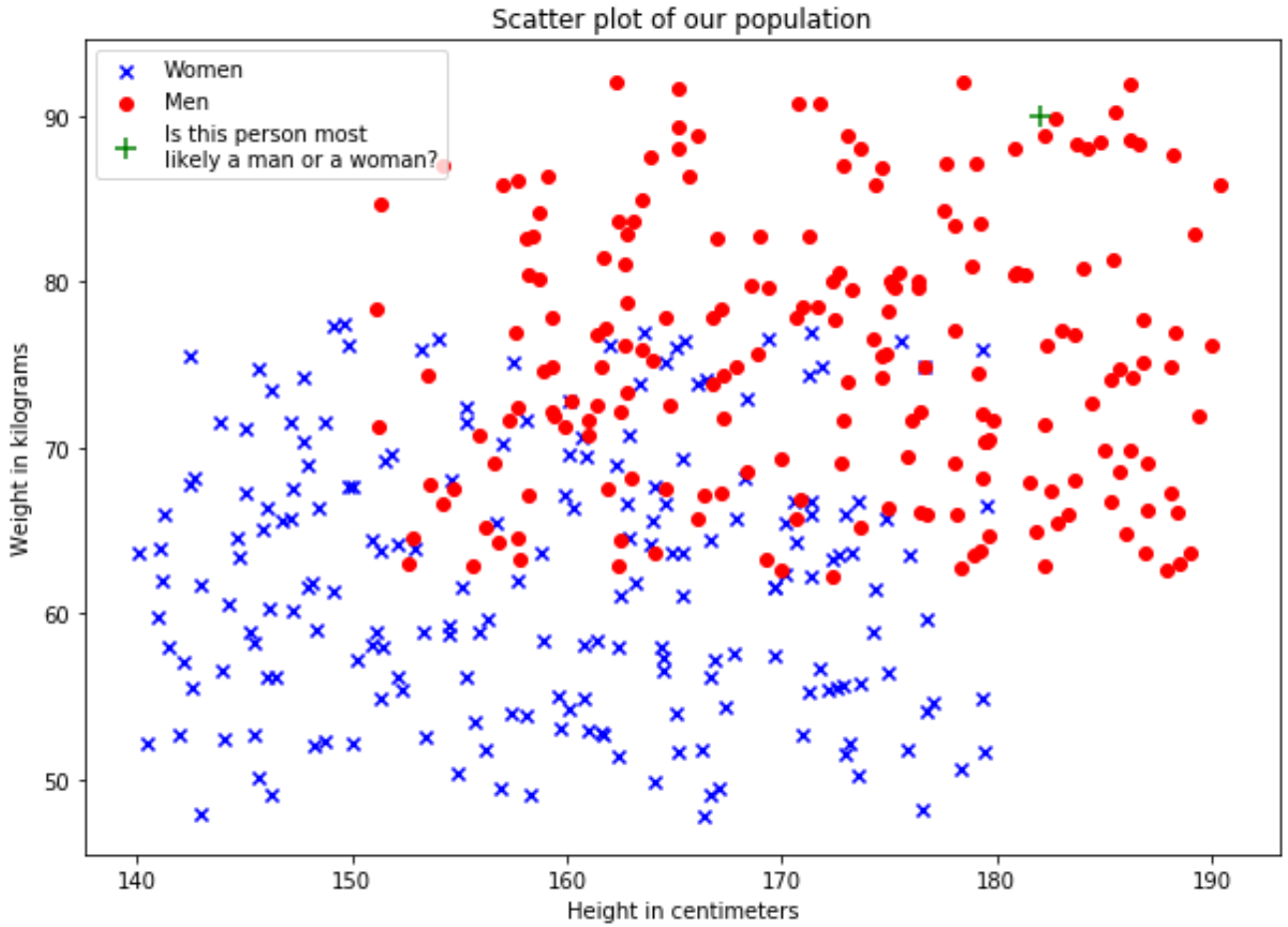
```
:
1  fig = plt.figure(figsize = (10,7))
2  plt.scatter(data[:nw, 0], data[:nw, 1], color = 'blue', marker = 'x', label = 'Women')
3  plt.scatter(data[nw:, 0], data[nw:, 1], 36, color = 'red', marker = 'o', label = 'Men')
4  plt.legend(loc = 'best')
5  plt.xlabel('Height in centimeters')
6  plt.ylabel('Weight in kilograms')
7  plt.title('Scatter plot of our population')
8  plt.show()
```



Intuition

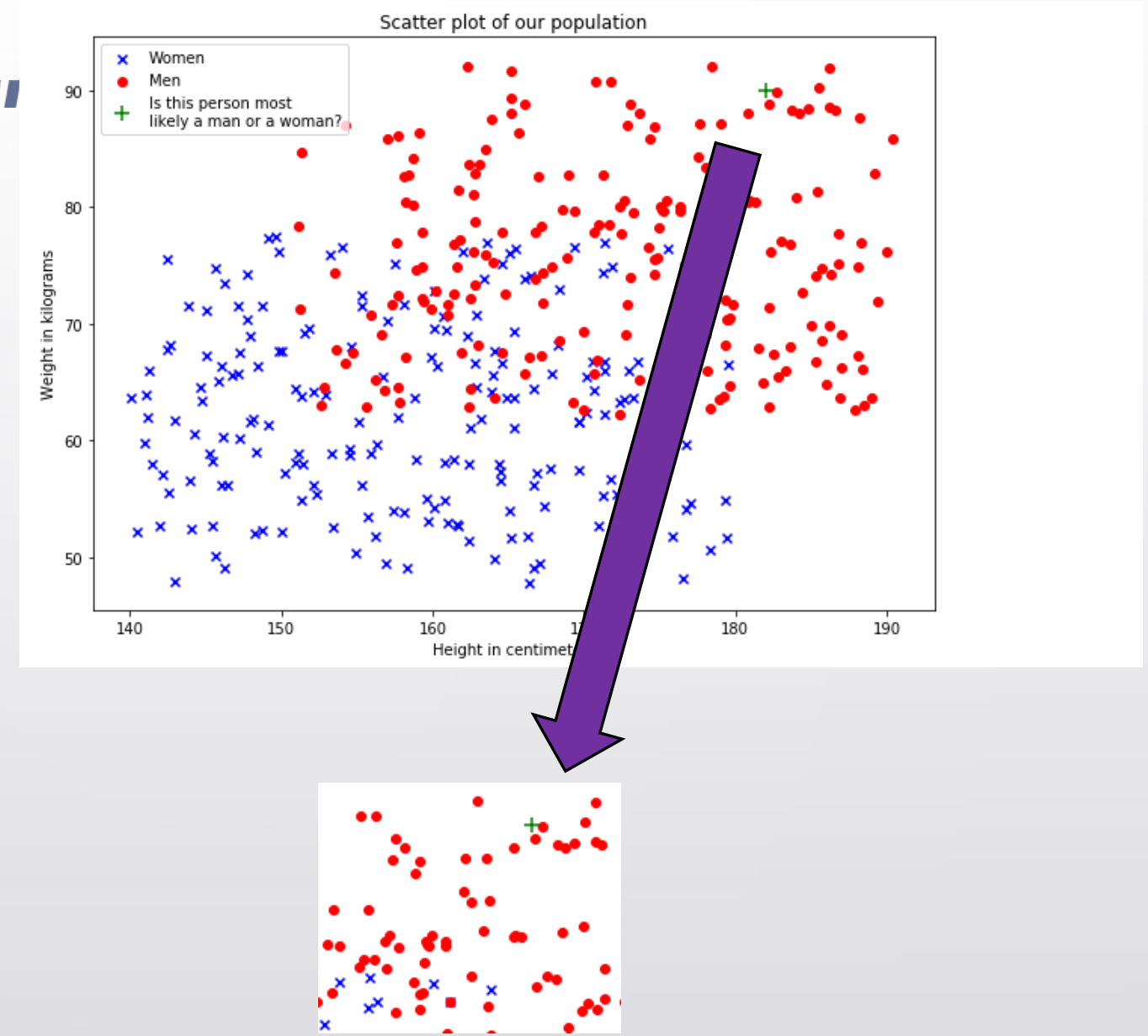
- What if we were to try and guess the gender of a person based on its height and weight?
- What is most likely the gender of a person whose
  - Height is 182cm
  - And weight is 90kg?
  - (Green point on scatter plot here)

```
fig = plt.figure(figsize = (10,7))
plt.scatter(data[0:nw, 0], data[0:nw, 1], color = 'blue', marker = 'x', label = 'Women')
plt.scatter(data[nw:, 0], data[nw:, 1], 36, color = 'red', marker = 'o', label = 'Men')
plt.scatter(182, 90, color = 'green', marker = '+', s = 100, \
            label = 'Is this person most \nlikely a man or a woman?')
plt.legend(loc = 'best')
plt.xlabel('Height in centimeters')
plt.ylabel('Weight in kilograms')
plt.title('Scatter plot of our population')
plt.show()
```



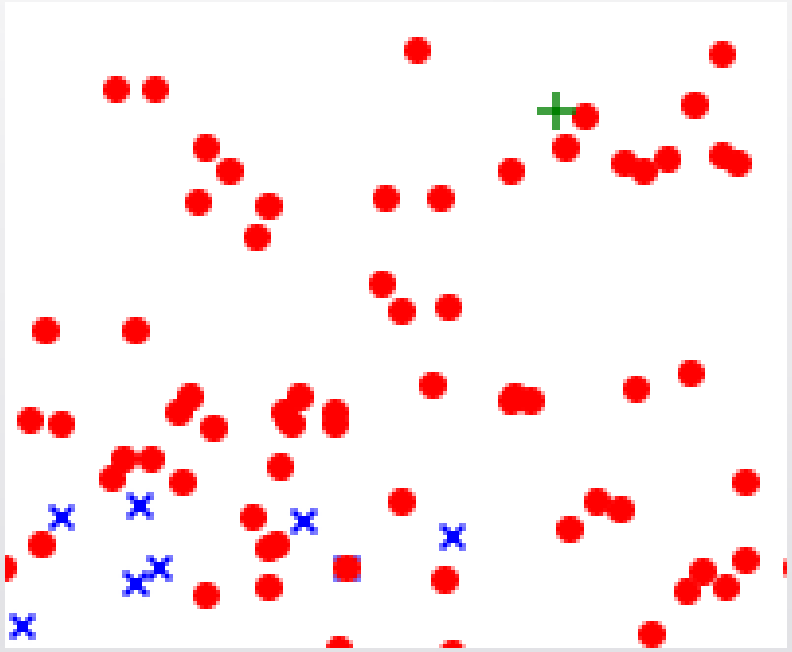
Intuition

- Most neighbouring points around the green one were men.
- Our guess is then that the new point (in green) is most likely a man as well.



Algorithm: the K Nearest Neighbours (K-NN) classification algorithm

- Decide on $\boldsymbol{K}$, a number (often an odd number, $\boldsymbol{K} \geq \mathbf{1}$).
- **Step 1:** for a new point (in green), find the K closest points to the green point (i.e. the K nearest neighbours)
- **Step 2:** our guess on the label of the green point is the most frequent label among the K nearest neighbours.



Algorithm: the K Nearest Neighbours (K-NN) classification algorithm

- As before with the Linear Regression…
- We will need to perform a training and testing samples split on our record.
- Then, we have a function in the sklearn library, that handles the KNN classifier.

```
1 from sklearn import neighbors
```

```
# A minimal example of a KNN
def knn_classifier(x, y, size, seed , k):

    # Normalize the data (helps, but optionnal)
    x = normalize_minmax(x)

    # Train and test split
    x_train, x_test, y_train, y_test = train_test_split(x, y, \
                                                        test_size = size, \
                                                        random_state = seed )


    # KNN classifier object and fit
    clf = neighbors.KNeighborsClassifier(k)
    clf.fit(x_train, y_train)

    # Use predict on our x_test
    y_pred = clf.predict(x_test)

    # Get performance results
    results = get_metrics(y_test, y_pred, [0,1])

    return results
```

Let us practice a bit

Problem set 10 – Q4 (first attempt at KNN)

Q4: our first attempt at KNN

Step 1: Load your dataset and select the data used for classification.

Step 1bis: plot a histogram, showing the repartition of values in our dataset.

Step 1ter (optional): normalize your data.

Step 2: Use the train_test_split, to split your record/experience into training (x_train, y_train) and testing (x_test, y_test) samples.

Step 3: Use the KNN model from sklearn, and fit your KNN to your training data.

Step 4: Predict your test samples using this trained KNN on your x_test samples and store the result in y_pred.

Step 5: Compute the accuracy of your classifier with the get_metrics function, using y_pred and y_test.

Step 6: Display your final results!

Improving our KNN

- We designed a basic KNN for our breast cancer dataset.
- However, in Q4, we have decided on a value for K, arbitrarily.

- **Question: How can we decide which value of K should be used?**

Algorithm: choosing the optimal K value (1/3)

- We will mostly reuse our classifier from Q4.

- First, we will split the record in three sets
  - Training samples (60%)
  - Validation samples (20%)
  - Testing samples (20%)
  - (You can do so by using the train_test_split function twice)

Algorithm: choosing the optimal K value (2/3)

- We run the our KNN classifier function on the same training data, for multiple values of K.
- And store the accuracy results in a list (one element for each K), by using the validation data for performance evaluation.

- Once done, find the K value with the maximal accuracy, and call it K*.

Algorithm: choosing the optimal K value (3/3)

- Then, recreate the KNN classifier with the optimal K* value.
- And test its accuracy on the testing samples.

- Finally, return a dictionnary, with
  - The best K value, K*
  - The accuracy results on the validation samples
  - The accuracy results on the testing samples

Let us practice a bit

Problem set 10 – Q4 (first attempt at KNN)

Q7: our final KNN with optimal K* value.

1. First, we will split the record in three sets
   - Training samples (60%)
   - Validation samples (20%)
   - Testing samples (20%)
   - (You can do so by using the train_test_split function twice)
2. We run the our KNN classifier function on the same training data, for multiple values of K.
   - And store the accuracy results in a list (one element for each K), by using the validation data for performance evaluation.
3. Once done, find the K value with the maximal accuracy, and call it K*.
4. Then, recreate the KNN classifier with the optimal K* value, and test its accuracy on the testing samples.
5. Finally, return a dictionnary, with
   - The best K value, K*
   - The accuracy results on the validation samples
   - The accuracy results on the testing samples