University of East Anglia

Laboratory Sheet – Nearest Neighbour Classifiers

CMP-6002A - Machine Learning

Dr Daniel Paredes-Soto

Autumn 2023

Aims and Objectives

In this lab you will become familiar with the implementation of the Nearest Neighbours classifier, and different distance measures to assess the similarity between two feature vectors. In the first part of this lab, you will create functions to calculate the similarity between two feature vectors using distance metrics. In the second part, you will code a 1-NN Classifier using the insect classification toy problem seen in the lecture. In the last part of this lab, you will optimise a NN classifier with *n*-fold cross validation to find the best *k* for classification of the given dataset. The last section is very important to summarise the *classifier parameter refinement (tuning)* process discussed in Lecture 4, slide 48 "Finding the values of the important parameters of an algorithm".

Part A. Similarity measure between feature vectors

In this part of the lab you will create a set of functions to compute a similarity measure between a pair of feature vectors. Create a new Python script file (Lab5_NN.py). Create the functions `get_euclidean`, `get_minkowski` and `get_manhattan`. The functions will receive two variables, `x` and `y` representing two feature vectors to be compared. For instance, in a 2D space, you can compute the Euclidean distance between the point `[1, 1]` and the point `[2, 2]`. Therefore, your function will receive the features vectors `x=[1, 1]` and `y=[2, 2]` as arguments.

Comment your code to describe the formula that is applied when calling your functions. In PyCharm, you can start commenting a function by adding three double-quote symbols under the declaration and pressing Enter. This will generate a *comment template* where you can enter the description, arguments and output. Also, you can bring the *comment template* by selecting "Show Context Actions" / "Insert a documentation string stub" from the context menu over a function definition.

```python
def get_euclidean(x, y):
    """

    :param x:
    :param y:
    :return:
    """
```

Apply the formulas accordingly (with Minkowski parameter `p=3`) to compute the distance between two feature vectors. Make sure that your functions work for *n*-dimensional feature vectors. Test the following simple cases:

- `x=[1, 1], y=[2, 2]`
- `x=[0, 0, 0], y=[255, 255, 255]`

Hints: You can verify your output with other tools or with the scipy.spatial.distance class. Euclidean and Manhattan distances can be obtained by varying the parameter p in the Minkowski distance.

```
[output]:
# get_euclidean([1, 1], [2, 2]) = 1.41421...
# get_manhattan([1, 1], [2, 2]) = 2.0
# get_minkowski([1, 1], [2, 2], p=3) = 1.25992...
```

Part B. 1-Nearest Neighbour Classifier with toy example dataset

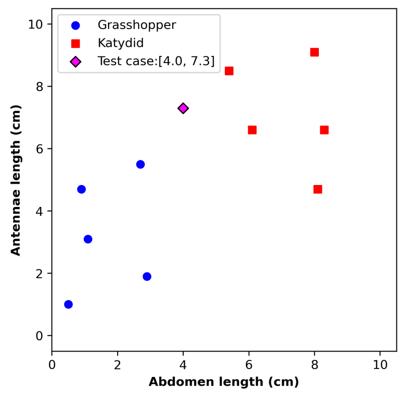
Use the insect dataset as the corpus of a NN classifier. Use the testing case `Abdomen length=4.00, Antennae length=7.30` to predict the insect type when k=1; this means, that you will label the testing case as the same as the most similar case in the corpus. Use `loops` (e.g., `for`) to scan all the cases in the corpus and compute the similarity between each case and the testing case. Use the functions created in Part A.

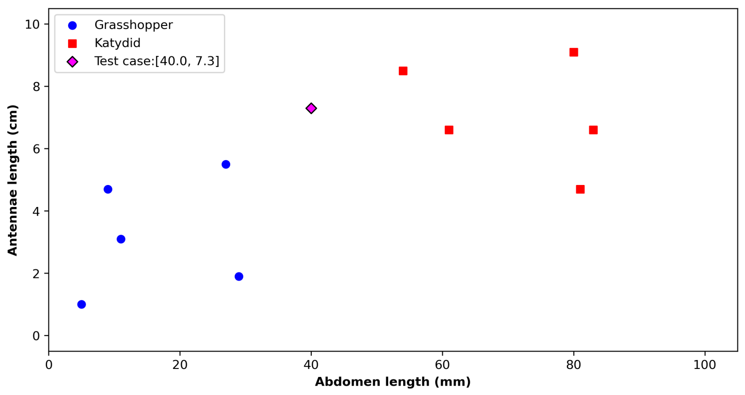
- Which of the cases in the corpus is in the 1-Nearest Neighbourhood of the testing case?
- What is the shortest/nearest distance (Euclidean)?
- What is the prediction made on the testing case using Euclidean?
- Is it the same prediction on the other distance metrics?

```
[output]:
```

Now, scale values in column `Abdomen length` to be measured in **mm**. Repeat the exercise and check whether the most similar case in the corpus is the same case or changed. You can create a scatter plot to visualise the 2D data points in the corpus and the testing case at the different scale values.

```
[output]:
```

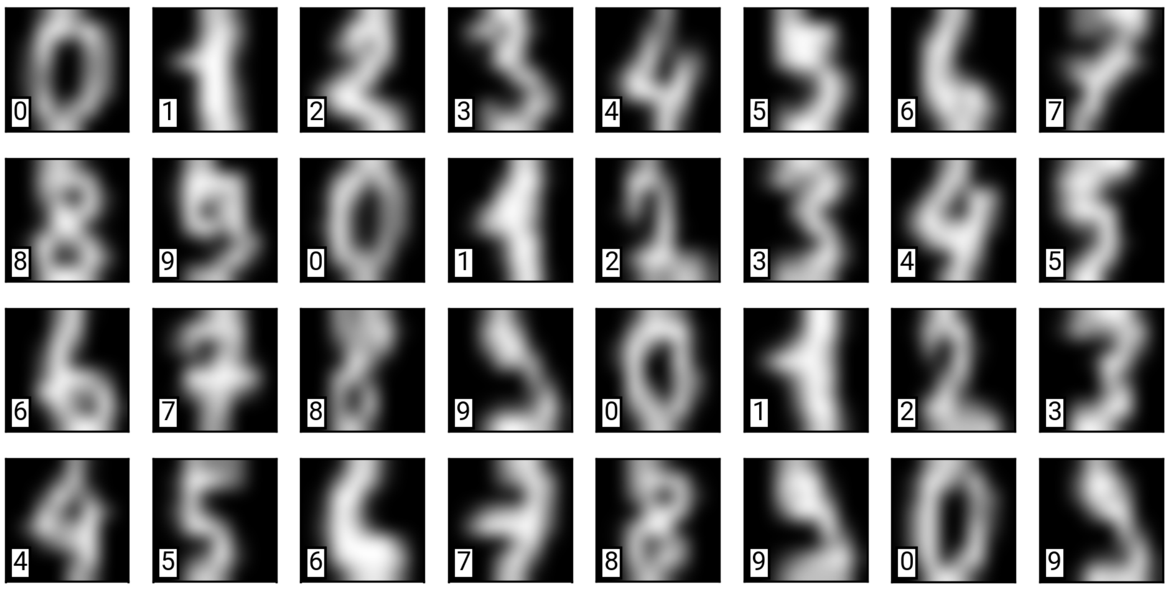




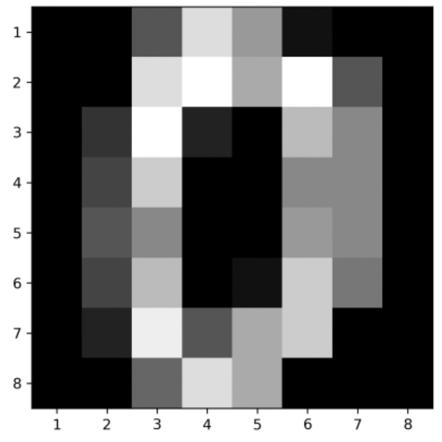
Part C. Optimisation of NN classifier with *n*-fold cross validation

In this exercise, you will implement the framework discussed in Week 4 to assess the performance of a classifier using cross-validation. You will find the value of the *k* that allow the highest performance on a given dataset. Then assess the performance on unseen data.

Create a new Python script (Lab5_optimisation.py). Load the MNIST (Modified National Institute of Standards and Technology) dataset contained in the `sklearn` library. The MNIST contains a collection of images of hand-written numbers (0, 1, . . . , 9). All images are uniformly resized to 8x8 pixels. Therefore, the dataset is formed by 64 attributes—each column in the dataset correspond to a pixel intensity value—; and, there are 10 different class labels named 0, 1, 2,. . . , 9. The numerical values in the dataset correspond to a greyscale value ranging between 0 and 15, where 0 is black and 15 is white. Some examples are shown below.



Example at index=0



```
X[0] = [ 0,  0,  5, 13,  9,  1,  0,  0,
         0,  0, 13, 15, 10, 15,  5,  0,
         0,  3, 15,  2,  0, 11,  8,  0,
         0,  4, 12,  0,  0,  8,  8,  0,
         0,  5,  8,  0,  0,  9,  8,  0,
         0,  4, 11,  0,  1, 12,  7,  0,
         0,  2, 14,  5, 10, 12,  0,  0,
         0,  0,  6, 13, 10,  0,  0,  0]

y[0] = 0 - Class Label "0"
```

> *Hint*. Use load_digits from the sklearn.datasets class[1]. You can use load_digits().data to extract the data (flatten 8x8 images into array), and load_digits().target to extract the class labels of the cases in ".data". You can follow the X, y notation for *data* and *labels* used in previous labs.

Print the number of rows and columns in your data and target variables to check that your data is loaded correctly. Check your values with the *total samples* and *dimensionality* reported in the MNIST sklearn documentation[2].

```
[output]:
```

[1]https://scikit-learn.org/0.15/modules/generated/sklearn.datasets.load_digits.html
[2]https://scikit-learn.org/0.15/modules/generated/sklearn.datasets.load_digits.html

Data split

It has been discussed that algorithms must be assessed on data that has not been used in the training or optimisation (parameter refinement). Therefore, you will perform an initial split to generate two sets, named **training_validation** and **testing**. The first set will be used for the experimentation to find the optimum *k*; and the latter, will be used to assess the performance of the NN classifier with the optimum *k*.

Use the `train_test_split` method used so far to split your data, with the parameter values as follows:

- `random_state = 1 # seed set to 1`
- `test_size = 0.3 # 30% of the MNIST examples for testing`

Print the number of examples in your **training_validation** and **testing** sets and check that the figures correspond to the 70/30 split.

```
[output]:
```

n-fold preparation

In cross-validation, instead of splitting the *training_validation* data into two parts, the data is split a number of times corresponding to the number of folds defined. In lab 4, you used `sklearn.model_selection.KFold`[3] to perform the splits and generate models.

In this exercise, use a 5-fold cross validation. Therefore, you will create five NN classifier models using—at somehow—different data from the training_validation set. Set the `KFold` parameter configuration as follows:

- `random_state = 2 # seed set to 2`
- `kfold_splits = 5 # 5-fold cross validation`

Cross validation for *k*-values

In this step, you will perform the cross validation of a NN classifier model. To recap, a NN model with some configuration will be trained and tested on the training_validation set.

The configuration of your NN classifier will vary by changing the value for *k*. This will allow you to analyse the performance of your NN classifier with cross-validation when increasing the number of cases in the *neighbourhood* of a testing case to predict the class it belongs to.

Use the number of *k* values (odd) between 1 and 50. For each of the *k* values, run a cross validation with the `cross_val_score`[4] class in sklearn (you can reuse your work in Lab 4). Use the `KNeighborsClassifier`[5] in `sklearn` as “estimator” in your cross-validation object.

Hint: You can follow the next code structure for cross-validation on a range of *k* values

```
kk = np.arange(1, 50, 2) # [1 3 5 ... 49]
for k in kk:
  # instance of KNN classifier with 'k'-neighbours

  # perform cross-validation with KNN, 5-fold-cv, on training_validation, report
  ↪   'accuracy'

  # get the mean accuracy across the 5-folds

  # store the mean accuracy for 'k', plot/visualisation purposes
```

[3]https://scikit-learn.org /stable/modules/generated/sklearn.model_selection.KFold.html
[4]https://scikit-learn.org /stable/modules/generated/sklearn.model_selection.cross_val_score.html
[5]https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

```
# select the 'k' with the highest mean accuracy
```

Print the highest accuracy and the optimum value for *k* (where the highest accuracy was achieved) in your experiment.

```
[output]:
```

Imagine you have a classifier that allow you refining more than one parameter, e.g. Support Vector Machines. You can "optimise" the algorithm by generating various models combining parameter values.

Assess the performance of a NN Classifier

To assess the performance of a classifier, it has to be built—trained—on the full *training_validation* set using the best parameter values found with the cross validation. In this exercise, you will train a `KNeighborsClassifier` (with *k* = the optimum *k*) on the *training_validation* set. Finally, the performance of an algorithm is given by testing unseen data; it is the **testing** set you separated in the first split.

Follow the standard pipeline—sequence of steps—to train and test a classifier. Then, compute the testing accuracy score.

Note. The NN classifier doesn't perform training, it only loads the data to form the 'corpus' of the classifier. `sklearn KNeighborsClassifier` requires to "fit" the classifier, but it will only load the training_validation data.

What is the configuration of your optimised KNN classifier?

```
[output]:
n_neighbors:
```

What is the testing accuracy of your optimised KNN classifier?

```
[output]:
accuracy_score:
```

Visualisation of your optimisation process

- You can set `verbose=True` in `cross_val_score` to report the accuracy scores at each of the folds in the cross validation, so you can observe how the mean accuracy in the cross validation is calculated.
- Bar plots (or line graphs if using continuous variable steps) can be produced to visualise the performance of a classifier through the different configurations (parameter value combinations) in the optimisation. In this exercise, you refine only one parameter (*k*), other ML classifiers allow refinement of more parameters.

You have now completed the fifth lab session.