CITS5508 Machine Learning
Semester 1, 2022

# Lab Sheet 1

(Not assessed)

The lab task for this week is to learn and experiment with the Jupyter Notebook examples in Chapters 1 and 2 of the textbook in your own computer environment. All the Python code examples from the book *Hands-On Machine Learning with Scikit-Learn & TensorFlow* can be downloaded from https://github.com/ageron/handson-ml2.

To set up Jupyter Notebook, see the *software_installation.pdf* file provided on LMS. In our experience it seems best to use *anaconda* or *miniconda* to set everything up.

## Optional exercise

In this exercise, we will learn how to use the *k-nearest neighbours* (k-NN) algorithm to do regression. This is an *instance-based learning* method (see Figure 1-15 in the textbook for an example of k-NN classification). Suppose that the features $\mathbf{X}_i$, for $i = 1, \cdots, N$, of our training data are $n$-dimensional vectors and each feature vector has an associated $y_i$ scalar value. Our objective is to predict the scalar $y_{\text{test}}$ value for a given test instance $\mathbf{X}_{\text{test}} \in \mathbb{R}^n$. The way how the k-NN regression works can be summarized as follows:

1. finds the $k$ nearest neighbours of the test instance $\mathbf{X}_{\text{test}}$ in the feature space ($\mathbb{R}^n$) . This step gives a set of $k$ neighbours $\{\mathbf{X}_{(1)}, \mathbf{X}_{(2)}, \cdots, \mathbf{X}_{(k)}\}$, where $k$ is a positive integer supplied by the user. From the training data, the associated scalar values $\{y_{(1)}, y_{(2)}, \cdots, y_{(k)}\}$ of these neighbours can be extracted.
2. computes $y_{\text{test}}$ as the weighted sum of the $y$ values of these neighbours, i.e., $y_{\text{test}} = \sum_{i=1}^{k} w_i y_{(i)}$.

In Step 1, we need to use a distance function that defines the *nearness* of points in the feature space. Distance functions that are commonly used include: Manhattan distance ($\ell_1$ norm), Euclidean distance ($\ell_2$ norm), and Minkowski distance ($\ell_p$ norm, for some integer $p > 2$).

In Step 2, the value of each weight $w_i$ needs to be defined. The simplest formula is to set $w_i = 1/k, \forall i$, so $y_{\text{test}}$ is just the simple average of $\{y_{(1)}, \cdots, y_{(k)}\}$. A more complex formula is to set $w_i$ as the inverse of the distance or squared distance of each neighbour $\mathbf{X}_{(i)}$ to the test instance $\mathbf{X}_{\text{test}}$, i.e., let

$$w_i = \frac{1}{d(\mathbf{X}_{(i)}, \mathbf{X}_{\text{test}}) + \epsilon} \quad \text{or} \quad w_i = \frac{1}{d^2(\mathbf{X}_{(i)}, \mathbf{X}_{\text{test}}) + \epsilon}$$

where $d(\cdot, \cdot)$ can be the Manhattan distance, Euclidean distance, Minkowski distance, or any user-defined distance between the two input items. The term $\epsilon$ in the denominator is a small constant to avoid the division-by-zero problem when $\mathbf{X}_{(i)}$ and $\mathbf{X}_{\text{test}}$ happen to be the same point and $d(\mathbf{X}_{(i)}, \mathbf{X}_{\text{test}})$ becomes zero. These more complex formulae allow neighbouring points that are closer to the test instance $\mathbf{X}_{\text{test}}$ to have larger weights. If the test instance $\mathbf{X}_{\text{test}}$ happens to coincide with one of the neighbours, then the weight for that neighbour would be 1 and the weights for all other neighbours would be 0.

After appropriately defining $w_i$, it is important to normalize all the weight values as follows:

$$w_i \leftarrow \frac{w_i}{\sum_{j=1}^{k} w_j}$$

so that they sum to 1.

The Scikit-Learn library includes the class `KNeighborsRegressor` that performs $k$-nearest neighbours regression. See

https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html

The bottleneck of k-NN is in Step 1 – how to quickly find the $k$ nearest neighbours. This is a computationally expensive process especially when the dimension $n$ of the features is large and when the training set size $N$ is large also (having more training data is actually good for the regression task so we should not complain that $N$ is too large). The class `KNeighborsRegressor` includes the implementation of various algorithms (e.g., building a *ball-tree* or a *kd-tree* data structure) to speed up the neighbour search step.

Your task in this labsheet is to apply the k-NN algorithm to the California Housing Prices dataset described in Chapter 2 to predict the house prices. You can use the parameter `n_neighbors` to set your desired $k$ value.

You can randomly split the data into a training set (say 80%) and a test set (say 20%), apply the k-NN regressor, and evaluate how good the regressor performs by computing the *root mean square error* (RMSE) of the predicted house prices of the test set.

Try to experiment with a selection of columns (avoid those columns that contain text) from the dataset to form your feature vectors. The number of columns that you choose would become the dimension $n$ of your feature vectors. Note that you would need to do some normalization so that your data is not dominated by some columns that have large magnitudes.

Try also setting the parameter `weights` to '`uniform`' (this is equivalent to setting $w_i = 1/k, \forall i$) and to '`distance`' (this is equivalent to setting $w_i = 1/(d(\mathbf{X}_{(i)}, \mathbf{X}_{\text{test}}) + \epsilon)$) and then compare the *root mean squared errors* (RMSEs) of their predictions.

———-

Let $\{\mathbf{X}_1, \mathbf{X}_2, \cdots, \mathbf{X}_n\}$ and $\{y_1, y_2, \cdots, y_n\}$ be the groundtruth instances and their corresponding ground truth house price values in the test set. Let $\{\hat{y}_1, \hat{y}_2, \cdots, \hat{y}_n\}$ be the predicted house price values for these instances. Then the RMSE is given by:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}.$$