INF264 - Homework 1

Instructions

Since it is the first homework, instructions will be as detailed as possible. Therefore, we provide you with a Jupyter notebook template that you can use to complete the exercises.

- **Format: Jupyter notebook**
- **Deadline: 30th August, 23:59**
- **Submission place:** `https://mitt.uib.no/courses/48632/assignments/89562`

Note: We will be using the machine learning library `scikit-learn` (also known as `sklearn`). If you have followed the installation instructions included in the Python crash course, you should already have it installed. If not, please follow the instructions given here: `https://github.com/odinhg/inf264-python-crash-course`.

k-NN for a classification problem on the Iris dataset

The Iris dataset consists of 150 samples of iris flowers. Each sample is described by a vector with four features: sepal length, sepal width, petal length, and petal width (all measurements are given in centimeters). The dataset is split into three classes, each representing a different species of iris flowers (see fig. 1).

These measurements tend to differ between the different species, thus it is possible to train and evaluate a classifier from this dataset whose task is to predict the species of an iris flower represented by aforementioned set of features. In this exercise we will use k-NN classifier.

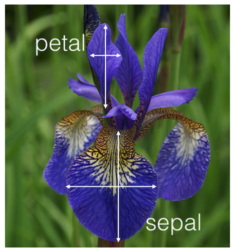

Figure 1: Left: The three species in the Iris dataset. Right: The four features in the Iris dataset (petal and sepal width and length)

The goal of this exercise is to train a k-NN classifier on the Iris dataset, using validation data to find the optimal value for k, and evaluate its performance on a test set. We will also visualize the decision boundaries of the classifier for different values of k, and as a bonus exercise, you can implement a k-NN classifier from scratch (without using `scikit-learn`).

Exercises

Part 1 – Training and Evaluating a k-NN Classifier

Loading and Splitting the Dataset

1. Load the Iris dataset using `load_iris()` from the module `sklearn.datasets`. Alternatively, you can manually download the dataset from `https://archive.ics.uci.edu/ml/datasets/iris`.

2. Store the first 2 columns/features (sepal length and sepal width) in a matrix (2-dimensional NumPy array) X, and labels in a vector (1-dimensional NumPy array) y. These arrays should have the following shapes: $X \in \mathbb{R}^{150\times 2}$ and $y \in \mathbb{R}^{150}$.

3. Split the dataset into 3 disjoint subsets: training data (70%), validation data (15%), and test data (15%). Store the training data in the variables `X_train` and `y_train`, validation data in `X_val` and `y_val`, and test data in `X_test` and `y_test`.

   You can use the function `train_test_split` from `sklearn.model_selection` to do this. Make sure to set the random seed to some fixed integer value for reproducibility via the argument `random_state`.

4. Plot the training data points using the first two features (sepal length and sepal width). Use a scatter plot and different colors for the three classes.

Training the Classifier and Finding the Optimal k

5. For each k in $\{1, 2, \dots, 9, 10, 15, 20, 25, 30\}$, do the following:

   (a) Create an instance of the `KNeighborsClassifier` class from `sklearn.neighbors` with the parameter `n_neighbors` set to k.

   (b) Train your instance of k-NN on your training data set.

   (c) Compute model accuracy on training dataset and validation dataset using the `score` method and store the results in lists (one for training accuracy and one for validation accuracy).

6. Plot a curve representing the training accuracy as a function of k and same for the validation accuracy, using the lists you created in the previous step. Plot the two curves in the same figure and make sure to label the axes and add a legend. What do you observe in terms of over-/under-fitting? Furthermore, what training accuracy do you get for $k = 1$ and is this expected?

7. Use `np.argmax` on the validation accuracy list to find the index (and value by indexing) of the k that maximized the validation accuracy. Print the value of k that maximized the validation accuracy and the corresponding validation accuracy. Store the value of k in a variable called `best_k`.

8. Concatenate the training and validation data to create a new training dataset. Train a k-NN classifier with the optimal k found in the previous step on this new training dataset. Compute the accuracy of the model on the test dataset. What is the accuracy of the model on the unseen test dataset?

Part 2 – Visualizing the Decision Boundaries

We are now going to visualize the decision boundaries of the k-NN classifier to see how the value of k affects the model's complexity. You are provided a almost complete function `plot_knn_decision_boundaries` that you can use to visualize the decision boundaries of the classifier. This function generates a grid of points in the feature space and colors them according to the class predicted by the classifier.

1. Complete the function `plot_knn_decision_boundaries`.

2. Let $k \in \{1, 5, 10, 25, 50, 75, 100\}$. For each value of k, do the following:
   (a) Train a k-NN classifier on the training data.
   (b) Use the function `plot_knn_decision_boundaries` to visualize the decision boundaries of the classifier.
3. What do you observe in terms of the decision boundaries as k increases? How does the complexity of the model change as k increases? Try with k equal to the number of samples in the training set (105). What do you observe in this case?

Part 3 (Bonus) – Implementing k-NN from Scratch

If you want to challenge yourself and test your understanding of the k-NN algorithm, you can implement the k-NN classifier from scratch. Some hints are provided in the Jupyter notebook template.