Linnaeus University
Introduction to Machine learning, 2DV516
Jonas Lundberg
`jonas.lundberg@lnu.se`

Assignment 1: k-Nearest Neighbors

Introduction

In this assignment you will handle four exercises related to the *k-Nearest Neighbors* algorithm. The main purpose is to get you up and running in Python and to give a first introduction the scikit-learn machine learning library.

Submission instructions

All exercises are individual. We expect you to submit at least one .py-file (or Jupyter Notebook) for each exercise and your submission should include all the datasets and files we need to run your programs. When grading your assignments we will in addition to functionality also take into account code quality. We expect well structured and efficient solutions. Finally, keep all your files in a single folder named as `username_A1` and submit a zipped version of this folder.

Exercise 1: k-NN Classification

In Lecture 1 we used as an example the Iris2D dataset for classifying three subspecies of the flower Iris based on petal and sepal leaf areas. We also presented a figure showing the original dataset and the decision boundary with corresponding training error for $k = 1, 3, 5$.

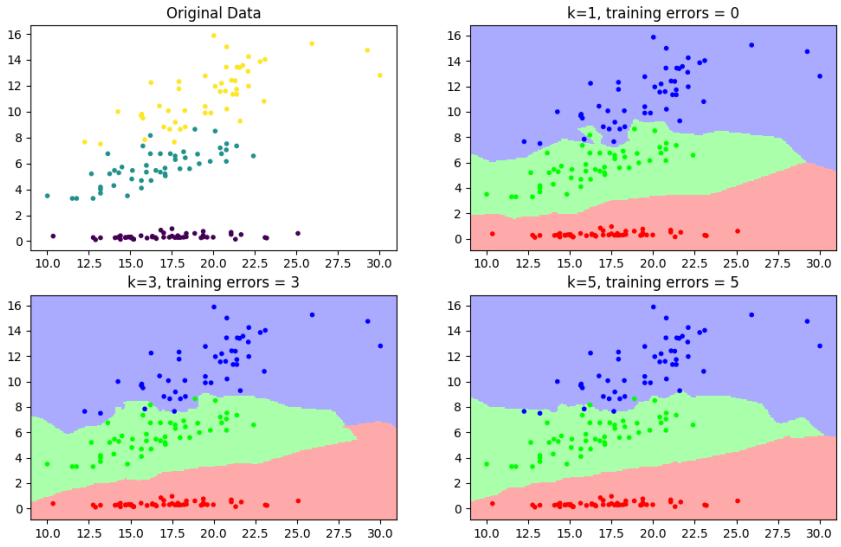


Figure 1: Figure from Lecture 1

In this exercise you should create a similar figure for the dataset in `microchips.csv` containing data related to an attempt to identify flaws in a microchip based on two numerical chip properties. Each row in the file represents one microchip and contains three comma separated values, the two properties and an integer (1 or 0) indicating if the microchip turned out to be OK (1) or Failed (0).

Create a program `Microchips` that:

1. Plot the original microchip data using different markers for the two classes OK and Fail.
2. Implement and use k-NN to predict whether three unknown microchips are likely to be OK or Fail. The properties associated with the three unknown microchips are $(-0.3, 1.0), (-0.5, -0, 1)$, and $(0.6, 0.0)$, and you should repeat the experiments for $k = 1, 3, 5, 7$. Hence, a total of 12 predictions. For example, for $k = 5$ the print out might look like:

```
k = 5
    chip1: [-0.3, 1.0] ==> Fail
    chip2: [-0.5, -0.1] ==> OK
    chip3: [0.6, 0.0] ==> OK
```

3. Display a 2×2 plot similar to Figure 1 showing the decision boundary and the training error for $k = 1, 3, 5, 7$.

Notice: Implement k-NN from scratch. Do not use any of the available k-NN solvers that are available in Python. We will start to use external machine learning libraries (scikit-learn) in Exercise 4.

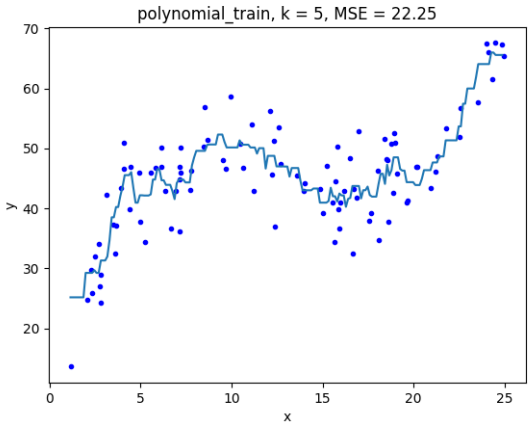
Exercise 2: k-NN Regression

The datasets `polynomial200.csv` contains 200 x, y samples generated from the function

$$y = f(x) = 5 + 12x - x^2 + 0.025x^3 + \underbrace{\text{normrnd}(0, 5)}_{= \varepsilon \text{ (Noise)}}.$$

Create a program `Polynomial` that:

1. Divide the dataset into a training set of size 100, and test set of size 100.
2. Plot the training and test set side-by-side in a 1×2 pattern.
3. Display a 2×2 plot showing the k-NN regression result and the MSE *training error* for $k = 1, 3, 5, 7$. For example, the plot for $k = 5$ might look something like this



4. Compute and present the MSE test error for $k = 1, 3, 5, 7$.
5. Which k gives the best regression? Motivate your answer!

Exercise 3: MNIST k-NN classification (VG Exercise)

This exercise is optional for passing the assignment, but required to obtain grades A or B.

In this final exercise, we will use k-NN for classifying handwritten digits using the very famous MNIST dataset. Input to the algorithm is an image (28x28 pixel) with a handwritten digit (0-9) and the output should be a classification 0-9. The dataset and a description of it is available at http://yann.lecun.com/exdb/mnist/. Google `MNIST Python` to learn how to access it.

The objective is to use your k-NN classifier to perform as good as possible on recognizing handwritten images. Describe your effort and what you found out to be the best k to lower the test error. The complete dataset has 60,000 digits for training and 10,000 digits for testing. Hence the computations might be heavy, so start of by a smaller subset rather than using the entire dataset. The final testing should (if possible) be done for the full test set but we will accept solutions that use "only" 10,000 digits for training and 1,000 digits for testing.

The description of this exercise is deliberately vague as you are supposed to, on your own, find a suitable way to solve this problem in detail. This is why it is important that you document your effort and progress in your report.

Finally, feel free to Google for help and ideas but make sure to mention what resources you have used in your report. Also, you are not allowed to use any of the predefined k-NN solvers available in Python. Use your own k-NN implementation. You might need to rewrite (i.e. optimize for performance) your Exercise 1 code in order to handle the heavy computations in a reasonable time.

Exercise 4: k-NN Classification using scikit-learn

Repeat Exercise 1 but this time using the `KNeighborsClassifier` from scikit-learn.