Lukas Ruppert
Raphael Braun
Timo Sachsenberg
Wolfgang Fuhl
Dimitrios Koutsogiannis
Frieder Wallner
Martin Röhm

Eberhard Karls
Universität
Tübingen

Programming in C++

Sheet 3

Submission date: 06.09.2023 12:00

K Nearest Neighbour classification (KNN)
Today, we will implement a simple Machine learning / Classification algorithm. We will classify an unlabeled object x (described by a feature vector - in our case a dynamic point) based on the class labels of its closest neighbours. The k nearest (already classified) objects participate in a majority decision by voting. The majority class, the class represented by the largest number of objects of these k neighbours, is then assigned to the unlabeled object. The distance measure that defines what is considered a nearest point can be one of the commonly used ones (Euclidean distance and Manhattan distance). In this task we will implement the KNN algorithm and a feature vector (Dynamic Point) visiting some of the core concepts we saw in the lecture.
This simple classification method is called k Nearest Neighbours (KNN).
For the implementation of the KNN algorithm we will use structs and pass the distance function as function pointers.
Please also take a look at main.cpp how KNN objects are instantiated, training data is set and how the final classification results are evaluated and printed.

3.1 KNN and dynamic feature vector (100 Points [equal distrib.])

C++

Please implement the missing parts in **KNNStruct.h** and submit the zipped submission folder to InfoMark.

a) First we will implement the dynamic feature vector **DYNPoint**. Implement the function **createRandomPoint** which receives the parameters **size, minimum, maximum**. The function should create a point of dimension `size` (to hold `size` values) in `data` of type `std::vector`. The values for the individual dimensions should be filled randomly in the range `minimum` to `maximum`. (Random points can be generated with the help of `std::rand` which creates a random integer between zero and the maximum integer the system supports.) This means for a minimum of -5000, a maximum of 5000, the values are in the range of `[-5000, 5000]`.

b) Now we implement two distance functions which are used to calculate the distance between points in the KNN.

- **i)** Implement the **DistanceManhatten** function according to $\frac{\sum_{i=1}^{size} |A.data[i] - B.data[i]|}{size}$. The absolute value can be calculated with `std::abs`.
- **ii)** Implement the **DistanceEuclid** function according to $\sqrt{\frac{\sum_{i=1}^{size} (A.data[i] - B.data[i])^2}{size}}$.

c) Now implement the **classify** function of the **KNN struct**. Determine the **k** nearest neigbours for a given point in the training set and select the class that is most often represented among the k neighbours. If multiple classes have the same amount of neighbours, it doesn't matter which one you select.

d) Implement the **createDataset** function. It receives the parameters **data_set, amount, class_label, point_size, minimum, maximum**. The `amount` is the number of points which should be added to the dataset. `class_label` is an unsigned integer and represents the class of the created points.

e) Take a look at the **evaluateKNN** function. Read and try to understand it. (Not graded)

It computes the accuracy of the KNN on the dataset according to $\frac{\sum_{i=1}^{datasetsize} Prediction==Label}{datasetsize}$ and prints it out. (Not graded)

f) Take a look at the **main.cpp**. Read and try to understand the code. It creates the data, the KNN, trains the KNN and evaluates the KNN for different **k** and distance functions. (Not graded)