RUHR-UNIVERSITÄT BOCHUM

FAKULTÄT FÜR INFORMATIK

k-Nearest Neighbors Classifier Project Brief

Introduction to Python

tns Group

August 22, 2023

1 Introduction

Python has become very popular in the fields of data analysis and machine learning. This is because there are a number of useful machine learning libraries for Python, and Python is also suitable for rapid prototyping of new ideas. One common task in machine learning is the classification of new data based on a set of already classified (a.k.a. labeled) data points. This task is called supervised learning. In this project you implement the popular k-Nearest neighbors (kNN) classification algorithm. This algorithm classifies new points according to their closest *neighboring* points that are already classified. Unlike most classifier models kNN does not learn a hidden projection or representation of the possible inputs but directly uses the labeled reference points to classify new data.

The kNN algorithm classifies new data-points based on the class of their k closest neighbors among the reference data-points. The definition of "closest" depends on the selection of a distance function d to measure and compare the distances between points.

Attributes of the kNN algorithm. kNN does not require training as such, but it is possible and often necessary to experiment with its parameters in order to calibrate the algorithm to a dataset. When implemented in a naive manner the algorithm is not fast because it has to compute the distance of each new point to all reference points. It tends to perform well on large, low-dimensional data sets.

The two parameters that have to be set to configure the algorithm are

- the number of neighbors k to take into account when classifying a new data-point (see Figure 1.1) and
- the distance function d which specifies how to measure the distance between points (see Figure 1.2).

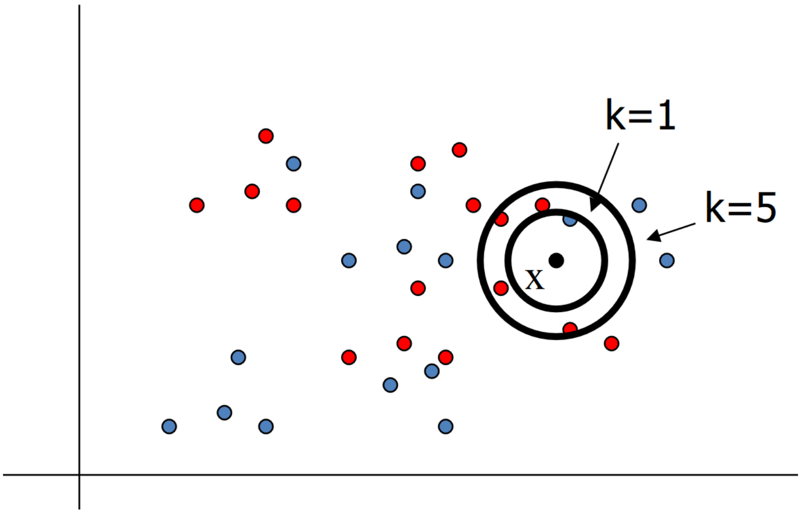

Figure 1.1: Alternative values for the number of neighbors to consider (k).

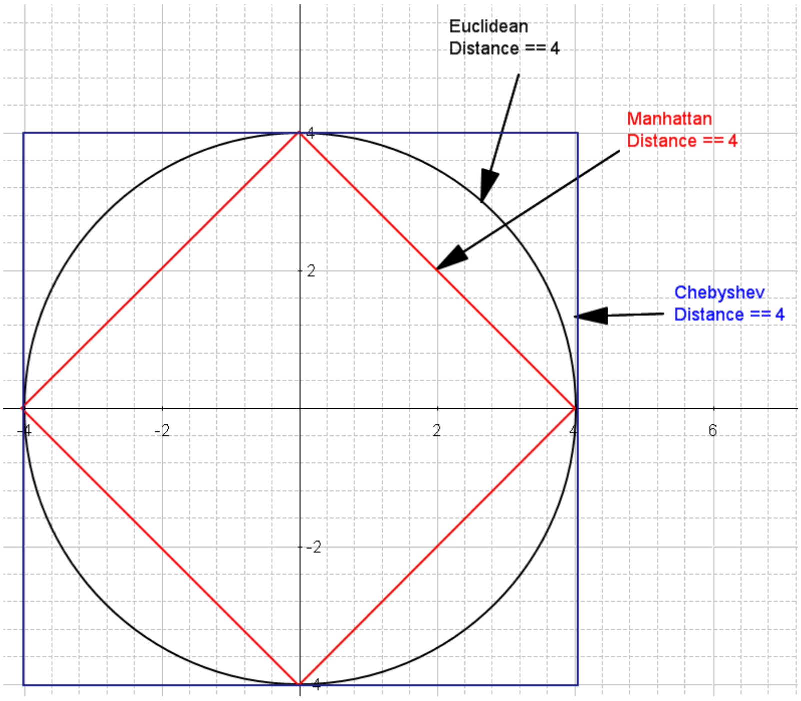

Figure 1.2: Different distance functions on R^n.

2 Stages of the core algorithm

These are the stages of the main algorithm that you should implement. When presenting the algorithm during the exam please explain its stages and show the corresponding code section / functions.

- Receive a training dataset $(x_0, y_0), (x_1, y_1), \ldots, (x_N, y_N)$ where $x \in \mathbb{R}^n$ are the input variables, i.e. input dimensions, and $y \in \mathbb{R}$ is the target variable, a value for $k \in \mathbb{N}$ and a distance function $d : \mathbb{R}^n \times \mathbb{R}^n \mapsto \mathbb{R}_{\geq 0}$. In our case, y is the `diagnosis` column/variable and should be treated as a categorical variable.
- Compute the mean and standard deviation (std) of the training dataset along each of the n input dimensions.
- Normalize the training dataset by subtracting the mean and dividing by the std that correspond to each variable.
- Observe a new data point $x \in \mathbb{R}^n$ and normalize it in the same way you normalized the training dataset.
- Find k data points (x_i, y_i), $1 \leq i \leq k$ in the training dataset such that $\sum_i d(x_i, x)$ is minimal. I.e., find the closest reference points in the training data according to distance function d.
- **Classification:** Assign x to the class y that occurs most frequently among its k nearest neighbors. If there are multiple most frequent classes return any one of those.
- **Regression:** Assign to x the mean y value of its k-nearest-neighbors.

2.1 Capacity, Overfitting, and Underfitting

We calibrate the kNN model to the problem we try to solve by properly choosing the values for its parameters k and d. Our goal is to obtain a model that is sensitive to the actual regularities (i.e. that avoids under-fitting) of the dataset and that ignores the noise as much as possible (avoids over-fitting). See Figure 2.1 for examples of over- and under-fitting and the Wikipedia article on the bias-variance trade-off for a discussion.

This is the procedure you should follow to obtain a model that fits the data well:

- Split the dataset into 3 sets such that they all follow the same distribution: Form training, validation, and test sets by randomly partitioning the rows in the data.
- Try to fit the validation set using the training set and observe the model's performance, e.g. its accuracy, on the validation set.
- Optimize the parameters of kNN such as to maximize the algorithm's performance on the validation set.

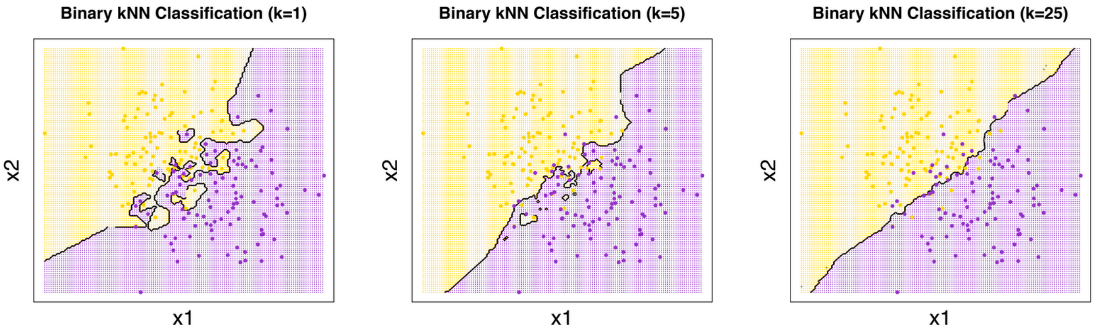

Figure 2.1: The left figure depicts an over-fitted model: The model represents spurious, non-generalizing structure from the training data. The model on the right in under-fitted: It fails to account for many data points. The figure in the middle depicts a model that has been calibrated appropriately.

- Finally, measure the performance on the test set as an approximation to the performance of the model on new data.
- If a model fits the training dataset and also fits the test dataset well, minimal overfitting has taken place.

2.2 Implementation: kNN class

Some ideas how you could structure a python class to represent a kNN predictor:

- Attributes:
  - *trained* tells us whether the model is trained.
  - k and d.
  - X: input data/variables for all data points.
  - Y: classes or target values corresponding to each point in X.
  - ...
- Methods:
  - *fit(·)* to fit/train the model to some data.
  - *predict(·)* to predict classification or regression result for new data.
  - *confusionMatrix(·)* to compute the confusion matrix.

3 Expected Functionality

You need to implement all features listed in this section in order to receive full grades.

1. Load the data from `KNNAlgorithmDataset.csv`. You are allowed to use the `pd.read_csv` function from the `pandas` package for this. The `diagnosis` variable should be the target variable, i.e. y. The dataset represents malign ($\texttt{diagnosis} = 1$) and benign ($\texttt{diagnosis} = 0$) tumors. Shuffle the samples in the dataset before the next step.

2. Split the data into training, validation and test sets. For example, use 70%, 20% and 10% of the samples in the dataset for the train, validation, and test set, respectively.

3. Implement the kNN algorithm in a reasonable object-oriented manner realizing all functionality described in Section 2.

   - Normalize the data.
   - Allow for different valid parameter configurations k and d and check that they are valid.
   - Implement all distance functions in Figure 1.2: Manhattan, Euclidean and Chebyshev.
   - Provide a simple API so that your implementation can be used by others. That means: structure your code using classes, functions and/or methods that make it simple and intuitive to use the functionality you implemented. Use doc-strings and type annotations where appropriate.
   - Provide a function, that computes the confusion matrix of your model on user-specified data.

4. Experiment with different values for k and d and report which parameter combinations perform better regarding model accuracy on the validation set. Then, report the accuracy and the confusion matrix of the best configuration on the test data.

5. Write a visualizer that displays subsets of the training and validation data sets (e.g. 60 and 40 points, respectively) by drawing the points in different shapes for evaluation and training points. The visualizer should also color in the areas in the background according to how points there would be classified, using different colors for different classes (like in Figure 3.1). Please reduce the input *X* to the kNN predictor to the two variables that you are going to plot. I.e. remove all other columns from the data in order to unambiguously color every point. Label the axes in the plot based on the name, i.e. the meaning, of the two dimensions you are plotting.

   Hint: You may have to iterate over all positions in the canvas behind the scatter plot for which you will need a prediction in order to color the background.

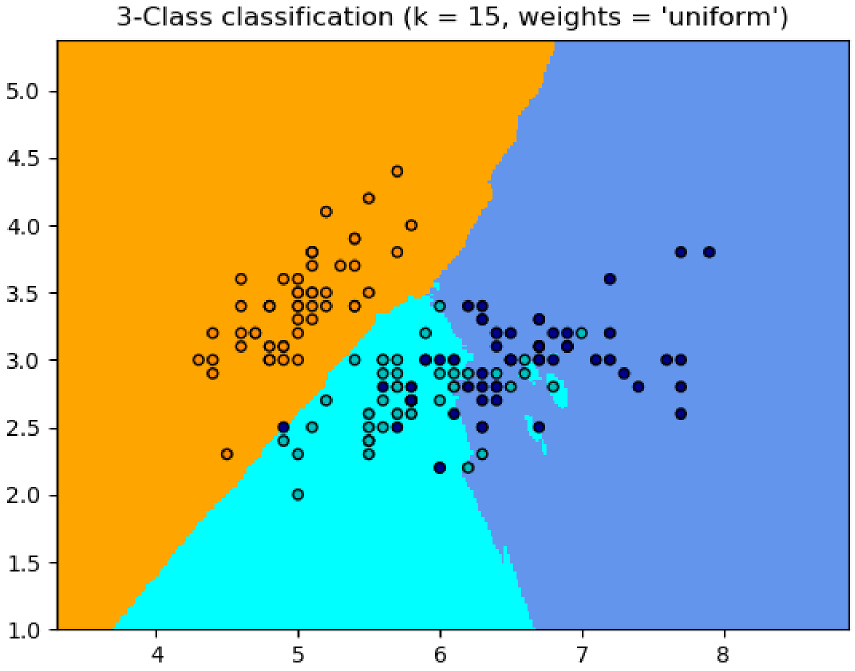

Figure 3.1: Example visualization for feature 5.