Homework 3: *k*-Nearest Neighbour and Naive Bayes

Dr. Juliane Klatt
juliane.klatt@bsse.ethz.ch

Giulia Muzio
giulia.muzio@bsse.ethz.ch

Dr. Bastian Rieck
bastian.rieck@bsse.ethz.ch

Prof. Dr. Karsten Borgwardt
karsten.borgwardt@bsse.ethz.ch

Submission deadline: 11.11.2020 at 08:00

Objectives

The goals of this homework are:

- to understand and implement the *k*-Nearest Neighbour algorithm
- to understand the Naive Bayes classification algorithm.
- to explore theoretical aspects of both algorithms.

Problem overview

You will work with two classifiers in this homework. In Part 1, you will implement the *k*-Nearest Neighbour (*k*-NN) algorithm and apply it to a microRNA expression dataset of breast cancer patients. In Part 2, you will analyse a different breast cancer dataset and use the Naive Bayes classifier to predict whether a tumour should be considered malignant or benign.

Homework Part 1: *k*-NN

Introduction

You will implement the *k*-NN algorithm in Python. Your program will accept a series of command-line arguments that will specify all execution parameters. It will receive data files as input (tab-separated text files) and will create an output file with the results. Additionally, you will be asked to address some theoretical questions about *k*-NN.

patientId	miRNA_1	miRNA_2	...	miRNA_d
X99	99.9	99.9	99.9	99.9

Figure 1: Format of the file `matrix_mirna_input.txt`. Individual columns are separated by a TAB character.

patientId	ERstatus
X99	+/-

Figure 2: Format of the file `phenotype.txt` file. Individual columns are separated by a TAB character.

Dataset

You will work on a real dataset of microRNA expression profiling obtained from a study of breast cancer patients [1]. The original data were downloaded from the Gene Expression Omnibus with id=GSE22216[1]. The patients in this study were divided into two groups: estrogen receptor positive (ER+) and estrogen receptor negative (ER-). In simple terms, ER+ and ER- tumours show different molecular patterns in terms of cell differentiation, proliferation, survival, invasion and angiogenesis. This translates into a better prognosis and treatment of ER+ patients compared to ER- patients. For each patient, the expression levels of d microRNAs were measured. The microRNA expression data can be found in the file `matrix_mirna_input.txt` and its format is described in Figure 1.

The ER status of each patient is stored in a separate file named `phenotype.txt` as shown in Figure 2. The column `ERstatus` constitutes the label of each patient and indicates with the symbol + if the patient is ER+ or with - if ER-.

To test the classification performance of your algorithm, the original data have been divided into two sets. In the `data/part1` directory you will find two subdirectories named `train` and `test`. Each subdirectory contains the files `matrix_mirna_input.txt` and `phenotype.txt` described above. The `train` and `test` subdirectories contain the training and test data respectively.

Exercise 1

Exercise 1.a You will implement the k-NN algorithm in Python. Your script, which has to be named `knn.py`, will accept a series of command-line arguments that will specify all execution parameters. In your implementation of k-NN, use the standard Euclidean distance as distance measure between two data points:

$$\mathrm{d}(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\| = \sqrt{\sum_{i=1}^{d} (x_i - x'_i)^2} \tag{1}$$

Use the files in the `train` subdirectory to train your algorithm and test it with the contents of the `test` subdirectory. Your program will classify all data points in `test` and generate an output file named `output_knn.txt` as shown in Figure 3. The minimum

[1] `http://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE22216`

and maximum values of k will be user-supplied parameters as discussed in the next section of command-line arguments.

For even values of k, it is possible that ties may occur when counting the number of neighbours for each class. Thus, *for instances with ties*, please compute the prediction using $k-1$ neighbours in order to break the tie.

Value of k	Accuracy	Precision	Recall
1	0.81	0.81	0.93
2	0.81	0.81	0.93
3	0.71	0.79	0.79
⋮	⋮	⋮	⋮

Figure 3: Format of output file `output_knn.txt`. Columns are tab-separated. Values are centred within columns just for illustration purposes. The first three lines show the expected output of your program. Ensure that your program also generates a header and uses the correct number of decimal digits.

For this exercise, you are given 3 scripts:

1. `knn.py`: main script of your program, i.e. the one to invoke
2. `knn_classifier.py`: script where you will implement the k-NN algorithm
3. `evaluation.py`: script with the auxiliary functions for implementing the metrics useful to evaluate k-NN performance.

You can implement your solution in the blank parts of the given scripts, but you are not required to. However, your main script has to be named `knn.py`, and its invocation has to be consistent with what reported in the next section of command-line arguments.

If you use the given skeleton, you might find the following things useful:

- `knn_classifier.py` contains the implementation of the class object `KNNClassifier`, which implements the methods of a k-NN classifier.
- `from knn_classifier import KNNClassifier` imports the class object.

- `knn = KNNClassifier(X_train, y_train, metric = 'euclidean')` allows to create an instance of the object `KNNClassifier` with the specified input.
- `knn.set_k(k)` permits you to set the value of k.
- `y_pred_i = knn.predict(X_test_i)` calls the `KNNClassifier`'s method `predict`, i.e. the function in `knn_classifier.py` where to put your implementation of the k-NN algorithm.

Exercise 1.b Assume that your program is executed in a way that explores many different values of k. An expert will argue that by looking at the output file `output_knn.txt`, we cannot determine what the best value of k is. How do we need to structure our data and what process do we need to follow in order to be able to determine the best k?

Exercise 1.c Using 'big O' notation (i.e. Landau symbols) and assuming all data have been preprocessed, what is the time complexity of the training step in k-NN? What is the space complexity?

Exercise 1.d Now focusing on the prediction step, is its complexity different in a problem with c classes, where $c > 2$?

Exercise 1.e In your implementation, you used the *Euclidean distance*, which is a metric. Does k-NN work with other metrics such as the Manhattan distance as well? Moreover, does it also work for semimetrics, i.e. functions that do not satisfy the triangle inequality, such as DTW? Briefly justify your answer.

Exercise 1.f So far, you have used k-NN for *classification*. Is it possible to use k-NN for *regression* as well? More precisely, suppose you have data points $\mathbf{X} := \{\mathbf{x}_1, \mathbf{x}_2, \dots\}$ with associated measurements $Y := \{y_1, y_2, \dots\}$, where each $y_i \in \mathbb{R}$. Given a new measurement $x' \notin \mathbf{X}$, can you predict its measurement y'?

Command-line arguments

Your program will receive 5 command-line arguments:

`--traindir path`: is the path to the directory where the training data are stored.

`--testdir path`: is the path to the directory with the test data.

`--outdir path`: is the path to the output directory where the output file will be saved.

`--mink 99`: the minimum value of k on which k-NN algorithm will be invoked.

`--maxk 99`: the maximum value of k on which to run k-NN. This parameter, in conjunction with `--mink 99` is used to run the algorithm for multiple values of k.

For example, an invocation of the program will be:

```
$ python knn.py --traindir /home/hwk3/data/part1/train \
                --testdir /home/hwk3/data/part1/test \
                --outdir /home/hwk3/part1/output \
                --mink 1 --maxk 10
```

Homework Part 2: Naive Bayes

Introduction

You will implement the Naive Bayes algorithm in Python. Your program will accept a series of command-line arguments that will specify all execution parameters. It will receive data files as input (tab-separated text files) and will create output files with the results. Additionally, you will be asked to address some theoretical questions about Naive Bayes.

Dataset

You will work on a dataset of 699 breast cancer samples[2] obtained from the UCI Machine Learning repository (Wisconsin breast cancer dataset) [2]. The data can be found in the `data/part2` directory. Here, clinicians measured a specific a set of features and assigned a class label to each of them. The features and class are:

- Clump Thickness
- Uniformity of Cell Size
- Marginal adhesion
- Mitoses
- Class

The class label is: 2 for benign and 4 for malignant. The other features have a value between 1 and 10. This information is stored in a file named `tumor_info.txt`. The columns in the file are tab-separated in the order listed above. As it was the case in Part 1, the data have been divided into two sets in two subdirectories: `train` and `test`.

Exercise 2

Exercise 2.a Create a Python script named `nbayes_summarize_data.py` which creates a summary of the probabilities for each feature/value and class in the `train` data. Your program will generate an output file named `output_summary_class_<label>.txt` as shown in Figure 4 where `<label>` is either 2 or 4.

Use the probabilities reported in `output_summary_class_<label>.txt` to predict the class label of the following data point:

```
[ clump = 6, uniformity = 2, marginal = 2, mitoses = 1 ]
```

Include details of the calculations and the probabilities in the same way as it was done during the lecture with slides 93–102 of "Part 2: Classification Algorithms". It is sufficient to *count* the values of features here; there is no need to fit a more complicated distribution in this example.

Exercise 2.b The data contain missing values. How do these affect the computation of the probabilities in Figure 4?

[2] The original data were altered for this assignment

Value	clump	uniformity	marginal	mitoses
1	0.315	0.836	0.821	0.970
2	0.103	0.081	0.080	0.019
⋮	⋮	⋮	⋮	⋮
10	0.000	0.000	0.002	0.000

Figure 4: Format of the output file `output_summary_class_<label>.txt`. Columns are tab-separated. The first three lines show the expected output of your program for class 2; you can use this to debug your implementation. Ensure that your program also generates a header and uses the correct number of decimal digits.

Exercise 2.c What strategy can you suggest to overcome the problem known as 'zero-frequency'? This occurs when you need to compute the probability of a feature/value and class but there are zero instances of it in the training data, i.e. $P(X_j = x_j | Y = y_i) = 0$ for a given i and j.

Command-line arguments

Your program will receive 2 command-line arguments:

`--traindir path`: is the path to the directory where the training data are stored. Only these data are used to create the summary files.

`--outdir path`: is the path to the output directory where the output files will be saved.

For example, an invocation of the program will be:

```
$ python nbayes_summarize_data.py \
                --traindir /home/hwk3/data/part2/train \
                --outdir /home/hwk3/part2/output
```

Homework Part 3: Bayes' Theorem

Introduction

Since Bayes' theorem is of fundamental importance in data mining, this exercise will help you deepen your understanding of its implications. The following exercises can be solved either in writing or by writing some code. Exercise 3.b is supposed to be followed like a tutorial. Students who are already familiar with Bayesian statistics can skip the longer introduction and go directly to the tasks.

Exercise 3

Exercise 3.a Suppose you are invited to a Halloween party. There are two opaque bowls of candy, and you are being told that one contains 30 vanilla brownies and 10 chocolate brownies (bowl 1), while the other contains 20 vanilla brownies and 20 chocolate brownies (bowl 2). You pick one bowl at random and draw a vanilla brownie.

Use Bayes' theorem to calculate the probability that the bowl you selected is bowl 1. Give your answer as precisely as possible and show how you arrived at it.

Exercise 3.b Suppose you are taking an Uber ride home after the Halloween party. You notice that your driver's car has the number $D = 60$ stencilled onto it. Assuming that Uber numbers its cars sequentially and that there are no more than $N_{\max} = 1000$ Uber cars in Basel, what would be a probabilistically-motivated "good guess" for the total number of Uber cars? Bayes' theorem makes approaching these questions simple. We are interested in the *posterior* distribution $P(N \mid D)$, which states the conditional probability of Uber having N cars in Basel, provided that we seen one of them with number D. For example, we have $P(N' \mid D) = 0$ for $N' < 60$, because we already *know* that at least 60 cars have to exist! Bayes' theorem states

$$P(N \mid D) = \frac{P(D \mid N)P(N)}{P(D)}. \tag{2}$$

$P(N)$ is the *prior probability*, i.e. the probability of Uber having N cars. By defining the prior, we can model our own assumptions about the problem. A suitable initial prior would be the *uniform* prior, i.e. $P(N) = \frac{1}{N_{\max}}$. It would encode our lack of knowledge in the sense that we consider *every* number to be equally likely. The second term, $P(D \mid N)$, is also referred to as the *likelihood*. It measures how *likely* our observed data are under the hypothesis that N cars exist. Since every number on a car is equally likely to be observed, we have $P(D \mid N) = \frac{1}{N}$, provided $N \geq 60$. As a last ingredient, we need to calculate $P(D)$, the *evidence*, i.e. the probability of observing the data that we observed. This is the sum over all joint probabilities $P(D, N)$ for all $N \in [D, N_{\max}]$ (we do not have to sum smaller values of N because their probabilities are zero), which measure how likely it is that there are N cars and we observe a car with the number D. According to the rules of probability, we have $P(D, N) = P(D \mid N)P(N)$, so we can re-use the calculations from before and get

$$P(D) = \sum_{N=D}^{N_{\max}} P(D, N) = \sum_{N=D}^{N_{\max}} P(D \mid N)P(N). \tag{3}$$

Task: Calculate the posterior probability $P(N \mid D)$ for all valid values of N (you can do this by writing a small Python script that implements the previous equations). For which N does the posterior distribution attain its maximum?

Submission format and grading

You are *required* to upload your solution to the exercises in the following format:

```
homework3_<lastname>.zip
├─ solution.pdf (pdf containing your written answers to the questions)
├─ knn.py (script written to answer Exercise 1.a)
├─ knn_classifier.py (k-NN class object for Exercise 1.a. Optional)
├─ evaluation.py (auxiliary functions for Exercise 1.a. Optional)
├─ nbayes_summarize_data.py (script written to answer Exercise 2.a)
├─ nbayes_uber.py (script written to answer Exercise 3.b. Optional)
└─ further files or directories your code might need in order to run.
```

Submissions which do not confirm to this structure **will be penalised in the grading process.** This homework is worth a total of 100 points. Table 1 shows the points assigned to each exercise.

Scripts that do not run will not be corrected and result in zero points. When working on exercise 3, ensure that you detail the steps you used to arrive at the answer.

Acknowledgements

This exercise sheet was created by Damian Roqueiro and Karsten Borgwardt and extended by Bastian Rieck.

References

[1] F. M. Buffa, C. Camps, L. Winchester, C. E. Snell, H. E. Gee, H. Sheldon, M. Taylor, A. L. Harris, and J. Ragoussis. microRNA-Associated progression pathways and potential therapeutic targets identified by integrated mRNA and microRNA expression profiling in breast cancer. *Cancer Research*, 71(17):5635–5645, 2011. doi: 10.1158/0008-5472.CAN-11-0489. URL http://cancerres.aacrjournals.org/content/71/17/5635.abstract.

[2] D. Newman, S. Hettich, C. Blake, and C. Merz. UCI Repository of machine learning databases, 1998. URL http://archive.ics.uci.edu/ml/datasets.html.

Table 1: Grading key for Homework 3

60 pts.	**Exercise 1**	
	45 pts.	Exercise 1.a
	5 pts.	Exercise 1.b
	2 pts.	Exercise 1.c
	3 pts.	Exercise 1.d
	2 pts.	Exercise 1.e
	3 pts.	Exercise 1.f
30 pts.	**Exercise 2**	
	20 pts.	Exercise 2.a
	5 pts.	Exercise 2.b
	5 pts.	Exercise 2.c
10 pts.	**Exercise 3**	
	5 pts.	Exercise 3.a
	5 pts.	Exercise 3.b