Data Mining - Exercise 3

Luca Witte

2022-11-11

Task 1: *k*-NN

The k-nearest neighbor (k-NN) algorithm is implemented in *knn.py* using the Euclidean distance function. Its inputs are a training data set, a test data set and the range for the parameter k. For $1 \leq k \leq 10$ the output is:

Table 1: k-NN results returned from the attached python file. k-values from 1 to 10 were implemented.

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
4	0.76	0.80	0.86
5	0.76	0.85	0.79
6	0.81	0.81	0.93
7	0.76	0.80	0.86
8	0.76	0.80	0.86
9	0.71	0.79	0.79
10	0.81	0.81	0.93

The output values are defined as follows:

$$Accuracy = \frac{TP + TN}{TP + FP + FP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

Where TP is the number of true positive classification, TN a true negative, FP a false positive and FN a false negative.

From the way the algorithm works, it is clear that the chosen value for parameter k is substential for a reliable result. Very small values can be sensitive to errors in the training data, as the vote is only conducted between few nearest neighbors. If k is chosen too large, very distant objects might participate in the vote.

Subtask 1.b

To optimize the chosen value for k, quantifications as shown in *tab. 1* are helpful. Though, different metrics show different optima:

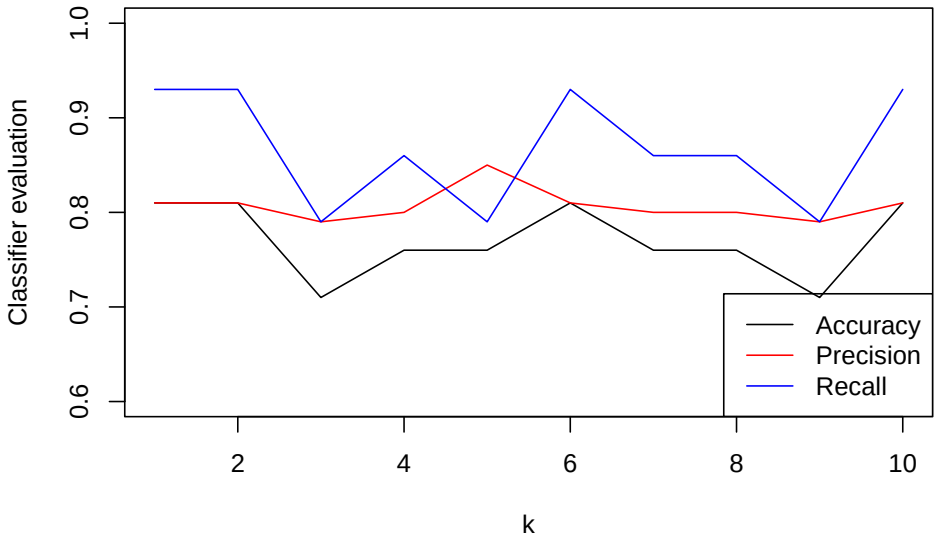

Figure 1: Classifier evaluation for the test-data set.

As an example, when evaluating the precision, one might select $k = 5$, but thereby implement suboptimal accuracy and recall. The measure optimized in classifier evaluation might be selected depending on the desired application.

Precision (true positives out of all positives) might be less important if the positively classified objects are investigated with additional, more accurate approaches. Though, precision should be optimized if the cost of false positives is high. Recall might be optimized to minimize the number of false negatives. This can be necessary if false negatives have severe consequences, e.g. in approaches to detect infectious diseases. Accuracy does take into account both false positives and false negatives and therefore might be used if both are associated with similar costs or consequences.

Therefore, *tab. 1* should be sorted by the most suitable metric depending on the specific case. In the microRNA expression data set, a research question is approached and not a diagnostic question. Therefore, results will be validated in subsequent steps and thereby a higher number of false positives might be corrected. False negatives on the other hand might lead to a loss of relevant information that could influence development of therapeutics or diagnostics. Therefore, precision and/or accuracy might be optimized in this case. In the depicted range, a k-value of 6 might be chosen to optimize both or $k = 5$ to maximize precision and maintain a good accuracy.

Further, besides the hyperparameter k, knowledge about the implemented distance metric is crucial to interpret the results. The choice of metric should be guided by prior knowledge about the data set and the metric itself.

Subtask 1.c

The question refers to the "training step" which is ambiguous to the author of this report. The training data is already processed and thereby no complex operations are conducted on the training set alone. Therefore, storing m objects of length l results in a time complexity of $\mathcal{O}(m * l)$ and a space complexity of $\mathcal{O}(m * l)$.

The actual algorithm (the "testing step") exhibits higher complexity. For each object in the test set (n objects of length l), the distance to all elements of the training set (m objects of length l) is calculated, resulting in a complexity of $\mathcal{O}(n * m * l)$. By considering each test object separately ($n = 1 * l$), this complexity can be approximated by $\mathcal{O}(m * l)$. The results for each test object are sorted. *numpy.argsort* (used in the script) by default implements the quicksort algorithm with a complexity of $\mathcal{O}(m * log(m))$. The complexity could be reduced by not sorting the whole list, but only extracting the k lowest values, resulting in a complexity of $\mathcal{O}(m * l * k)$.

Subtask 1.d

The prediction step is implemented after sorting the distances. Starting from a list of k objects, the label of each is evaluated ($\mathcal{O}(k * 1)$). The number of occurrences for all c possible outcomes are stored. Finding the maximum of the c labels has a complexity of $\mathcal{O}(c)$. For a larger c, the complexity is of the same order as for $c = 2$.

Subtask 1.e

Other distance metrics can be implemented. The only important thing is that within an application of the algorithm, the same distance metric is used to calculate all distances.

The same is true for semimetrics like dynamic time warping. While the interpretation of results might be more difficult than the geometric interpretation of distance metrics, they can be implemented to calculate the distance between test data and training data.

Subtask 1.f

As the k-NN algorithm only assigns a classification based on the training data. This means that a new measurement $x' \notin \mathbf{X}$ will be assigned one of the values ("classes") from $Y = \{y_1, y_2, ...\}$. Meaning that no new value can be extracted (like in a regression), but an existing value is assigned. Most intuitively, for $k = 1$, the new measurement is assigned the Y-value from the closest data point in $\mathbf{X}$. While this might deliver a somewhat accurate approximation, it does not constitute a reliable regression which is able to interpolate between the data points.

Task 2: Naive Bayes

In the second task, the Naive Bayes algorithm was implemented in the *nbayes_summarize_data.py* script. It is applied to a data set of breast cancer patients. The data is labeled either with 2 (benign) or 4 (malign) and for each sample, 4 features were observed and recorded. The program generates the following output for the benign samples:

Table 2: Likelihoods returned from the attached python file.
Results for $P(X = x_j \mid Y = 2)$ with $j \in [clump, uniformity, marginal, mitoses]$

Value	clump	uniformity	marginal	mitoses
1	0.316	0.835	0.821	0.970
2	0.104	0.081	0.080	0.019
3	0.210	0.053	0.067	0.005
4	0.145	0.019	0.011	0.000
5	0.180	0.000	0.007	0.002
6	0.035	0.005	0.009	0.000
7	0.002	0.002	0.000	0.002
8	0.009	0.002	0.000	0.002
9	0.000	0.002	0.002	0.000
10	0.000	0.000	0.002	0.000

It can be seen that the obtained values are in accordance to the expected output from the task description, though a small deviation occurs in some values. This might be caused by storing the data with different floating point precisions inside the script and subsequent rounding errors. The results are used to evaluate classification of a test-point:

Table 3: Test point for which a class label is to be predicted.

clump	uniformity	marginal	mitoses
5	2	3	1

For the two possible class labels, we obtain:

$$
\begin{aligned}
&with: \\
&\quad x = \{clump = 5,\ uniformity = 2,\ marginal = 3,\ mitoses = 1\} \\
&Evidence = P(X = x) \\
&\quad = 0.18265 * 0.06636 * 0.07951 * 0.83486 \\
&\quad = 0.00080 \\
&Evidence\ is\ constant\ for\ all\ classes\ and\ can\ be\ omitted
\end{aligned}
$$

$$
\begin{aligned}
Likelihood_{\ class\ 2} &= P(X = x|Y = 2) \\
&= 0.180 * 0.081 * 0.067 * 0.970 \\
&= 0.000948 \\
Prior_{\ class\ 2} &= P(Y = 2) \\
&= 0.6621 \\
Posterior_{\ class\ 2} &= P(Y = 2|X = x) \\
&= P(X = x|Y = 2) * P(Y = 2) \\
&= 0.000948 * 0.6621 \\
&= 0.00063
\end{aligned}
$$

$$
\begin{aligned}
Likelihood_{\ class\ 4} &= P(X = x|Y = 4) \\
&= 0.188 * 0.037 * 0.105 * 0.574 \\
&= 0.000419 \\
Prior_{\ class\ 4} &= P(Y = 4) \\
&= 0.3379 \\
Posterior_{\ class\ 4} &= P(Y = 4|X = x) \\
&= P(X = x|Y = 4) * P(Y = 4) \\
&= 0.000419 * 0.3379 \\
&= 0.00014
\end{aligned}
$$

From this evaluation, we can conclude that Naive Bayes would classify the test point as class 2 (benign):

$$arg\ max_{yi}\ P(Y = y_i|X = x) = arg\ max_{yi}\ P(X = x|Y = y_i) * P(Y = y_i)$$

$$
\begin{aligned}
&with: \\
&P(Y = 2|X = x) = 0.00063 \\
&P(Y = 4|X = x) = 0.00014
\end{aligned}
$$

$$P(Y = 2|X = x) > P(Y = 4|X = x)$$

Subtask 2.b

Depending on the applied methods, missing data can be a large issue in data science. The Naive Bayes algorithm has a simple, yet efficient way to solve this problem: It skips the respective instance and just calculates the probabilities based on the observed attributes.

The given data set contains missing values (NA's). The number of NA's differs between the different features in the training set:

Table 4: Number of missing values in the given data.

	clump	uniformity	marginal	mitoses
NA == True	657	648	654	654
NA == False	6	15	9	9

To obtain the solutions given in the task description, only the missing values are removed. The NA's are removed in the calculation, which changes the total number of entries in each column and thereby the observed frequency counts. Thereby the probabilities are changed within each column.

An alternative approach is to omit the entire recorded observation if a value is missing. This will decrease the total size of the training data set and therefore is more suited for very large data sets with only a small number of NA's.

Subtask 2.c

"Zero-frequency problem" describes a situation where in a class, one feature does not occur in the training data ($P(X_j = x_j|Y = y_i) = 0$). The classification is performed based on the term:

$$P(Y = y_i) * \prod_{j=1}^{d} P(X_j = x_j|Y = y_j)$$

The probability used for classification is zero if one of the likelihoods is zero. This will happen even in instances where the probabilities of other attributes strongly suggest a classification as the respective class.

A commonly used solution for this issue is adding a small Δ (often $\Delta = 1$) to each individual observation. Referred to as pseudocounts, this will result in a non-zero product from all probabilities without introducing a strong perturbation in the actual values.

Task 3: Bayes' Theorem

Subtask 3.a

From the task description, we can extract the following information:

- Bowl 1 ("A") contains 30 vanilla ("V") and 10 chocolate ("C") brownies
- Bowl 2 ("B") contains 20 vanilla ("V") and 20 chocolate ("C") brownies

To obtain the probability of Bowl 1 ("A") given the drawing of a vanilla brownie ("V"), we can apply Bayes' Theorem:

$$P(A|V) = \frac{P(V|A) * P(A)}{P(V)}$$

$$with:$$

$$P(V|A) = \frac{30}{30 + 10} = 0.75$$

$$P(A) = 0.5$$

$$P(V) = \frac{30 + 20}{30 + 10 + 20 + 20} = 0.625$$

$$therefore:$$

$$P(A|V) = \frac{0.75 * 0.5}{0.625}$$
$$P(A|V) = 0.6$$

Result: the probability that the selected bowl is bowl 1 is 0.6.

Subtask 3.b

Here, we assume that Basel has a maximum of $N_{max} = 1000$ Uber cars. Further, we know that there are at least $D = 60$. We use Bayes' Theorem to calculate the posterior probability $P(N|D)$ for different values of N to find out the most likely number of Uber cars under the given information. The prior $P(N)$ is assumed to be uniform with $P(N) = \frac{1}{N}$. The following python code was used for calculation:

```
N_max = 1000
D = 60
prior = 1/N_max

Vec_out = pd.DataFrame(np.zeros([N_max - D,2]))
Vec_out.columns = ["N", "posterior"]

counter = 0
evidence = 0
for M in range(D, N_max):
    evidence += 1/M * prior

for N in range(60,N_max):
    LH = 1/N # for N >= 60

    post = LH * prior / evidence
    Vec_out.iloc[counter,] = np.array([N,post])
    counter += 1
```

The obtained data is visualized below:

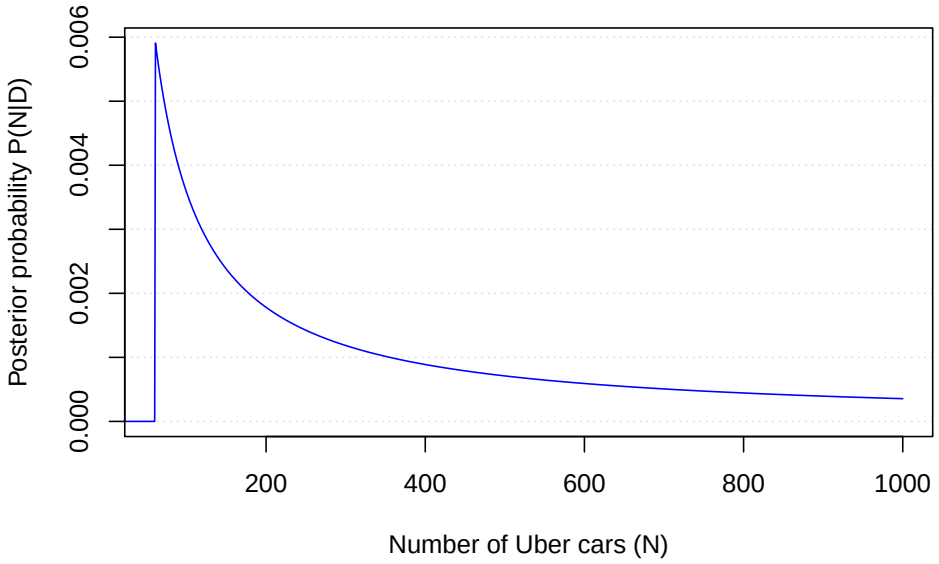

Figure 2: Probability distribution of N given the information in task 3.b.

From *fig. 2* and the raw data obtained using the code above, it can be extracted that the posterior probability is maximal for $N = 60$. As both the prior and the evidence in the given case do not depend on the value of N, the posterior probability only depends on the likelihood with $P(D|N) = \frac{1}{N}$ for $N \geq 60$ and $P(D|N) = 0$ for $N < 60$. The inverse relation $\frac{1}{N}$ is maximal for $N = 60$ in the limits of $60 \leq N \leq N_{max}$.

To calculate the expected value $E[N|D] := \sum_{N=D}^{N_{max}} N * P(N|D)$, the following python code was used:

```
vec_Exp = Vec_out.iloc[:,0] * Vec_out.iloc[:,1]
vec_Exp.sum()
```

It returns an expected value of 333.