UCL Université catholique de Louvain

École polytechnique de Louvain (EPL)

epl

Large scale person re-identification: error rates minimization under time constraint

Dissertation presented by
Victor HAMER

for obtaining the Master's degree in
Computer Science and Engineering

Supervisor(s)
Pierre DUPONT

Reader(s)
Marco SAERENS, Xavier MORELLE , Vincent BRANDERS

Academic year 2017-2018

Abstract

Given a population of individuals, the task of person re-identification consists of identifying the person inside the population to which corresponds the test data, if this person exists. In a large scale setting, realistic time and space constraints have to be posed, both for the learning and decision processes.

In this thesis, the person re-identification problem is solved by a threshold nearest neighbour search. The decision threshold minimizing the error rates has been shown to be dependent on the population size. With such an adaptive decision rule, the error rates increase sublinearily with the number of individuals, which allows to tackle large populations. In order to meet the time constraint, the decision rule has to be adapted, which degrades the identification performance. It is thus crucial to develop search algorithms, but also metrics allowing for an efficient search. A Mahalanobis metric learning scheme balancing between identification performance and computational efficiency will be proposed. The compromise is done by casting a convex optimization problem, that can be solved using a simple gradient descent algorithm. For some parameter values, this method is showed to be equivalent to standard metric learning for person re-identification.

Our method has shown a drastic improvement of the search efficiency on partially artificial data, compared to standard margin maximization metrics. In presence of a hard time and space constraint, this leads to a significant enhancement of the identification performance.

Keywords: person re-identification, metric learning, nearest neighbour search, large-scale, time constraint.

Acknowledgments

I would like to express my gratitude to my supervisor, Professor Dupont, for his guidance as well as his flawless availability throughout the year. His rapidity to answer my emails was remarkable.

Thanks also to M. Morelle who proposed the very interesting subject of this thesis, without whom this work would not have been possible.

I would also like to thank my family and in particular my dad for their precious help. Finally, I must express my deep thankfulness to Kyara Hallet, for her priceless support.

Contents

List of Symbols

I	Set of all individuals
R	Set of all noisy instantiations of these individuals
E	Set of stored examples, also used to denote the expected value.
l	Labelling function
P	Set of known individuals, called known population
q	Test or query example
d	Distance function between examples
t	Decision threshold
n	Number of data points
x	Known population size
β	Parameter of the F-score
α	Parameter of the F-score, equal to $\frac{\beta^2}{\beta^2+1}$
D_M	Mahalanobis metric parametrized by the matrix M
T	Expected decision time
η	Upper bound on the decision time
η_d	Upper bound on the number of distance computations
S_w	Space bound, in words of 32 bits.
B	Set of base prototypes
n_p	Number of base prototypes
D_{tot}	Total number of distance computations
N	Number of stored examples per individual

List of Acronyms

NNS	Nearest Neighbour Search
FAR	False Acceptance Rate
FRR	False Rejection Rate
FCR	False Classification Rate
BK-tree	Burkhard-Keller tree
AESA	Approximating and Eliminating Search Algorithm
LAESA	Linear AESA
TLAESA	Tree LAESA
MMC	Mahalanobis Metric for Clustering
ITML	Information Theoretic Metric Learning
LMNN	Large Margin Nearest Neighbour
NCA	Neighbourhood Components Analysis
MCML	Maximally Collapsing Metric Learning

List of Figures

List of Algorithms

Introduction

In some African countries, no universal means to identify people such as Id cards exists. This fact poses numerous problems. In particular, medical monitoring, which consists of recording the medical history of different patients, is very challenging in such countries. As an accurate medical history allows for more reliable treatments, it is critical to identify people correctly. Savics SPRL is currently trying to implement such a medical monitoring system by using alternative ways of identification. They use a device developed by the company Fujitsu which takes an infra-red picture of the hand of a patient. It analyses its veins network and decides whether or not the owner is already known in the database. Unfortunately, this system is not reliable enough. In particular, the number of unknown individuals being wrongly recognized is too high. To reduce the error rates, the hand-recognition system will be coupled with a machine learning algorithm using personal data, such as name and address. This machine learning algorithm, which address the person re-identification problem, is the main subject of this thesis.

The aim of person re-identification is to determine to which known person the test example[1] belongs, if this person exists. Person re-identification can be seen as a particular case of extreme classification[3] where the number of labels is huge (one per known individual). In such a context, time and space constraints have to be posed, both for the learning and the decision procedures.

Person re-identification has already attracted a lot of attention from the scientific community[4, 5, 6]. However, their work focuses on a sub-problem of person re-identification. Only the binary task of deciding whether or not two examples belong to the same person is addressed. The aspect of searching the individual among the known population is not tackled. In such a case, time and space constraints are irrelevant. To evaluate their methods, they feed them with pairs of examples from the same or from different individuals and record the number of classification errors. By doing so, they give an equal importance to false positive and false negative predictions[2]. This is inadequate, as for a correct identification to occur, such a binary classifier has to predict correctly negative on every example of different individuals whereas only one correct positive prediction is required.

In this thesis, this gap will be filled. All the aspects of the person re-identification problem will be studied. First, the problem will be formally defined, as well as the performance metrics used to evaluate the solution. As this work is based on partially artificial data, the process of data generation and perturbation will be discussed. A simple threshold nearest neighbour algorithm will then be proposed to answer the person re-identification problem. The optimal decision threshold will be found by minimizing the F-score of the error rates while respecting a

[1] An example is a noisy instantiation of the personal data of a patient. For instance ("Vctor", "Hamre").

[2] For the non-initiated reader, a false positive (negative) prediction happens when two examples of different (the same) individual(s) have been predicted as belonging to the same (different) individual(s).

time constraint on the decision time.

Different search algorithms will be analyzed in order to reduce this decision time as much as possible. An algorithm with an asymptotic logarithmic complexity in terms of distance computations will be proposed. Then, the unconstrained optimal identification performance will be studied. In particular, the evolution of the latter with the population size will be discussed. After that, the effects of the time constraint as well as an eventual space constraint on the identification performance will be assessed.

Like most machine learning algorithms, nearest neighbour methods highly depend on the metric used. Numerous schemes for nearest neighbour metric learning have been proposed[1, 7, 8]. However, they focus on the classification performance of the algorithm, and not its computational efficiency. In a large scale setting, where a realistic time constraint has to be posed, the decision time required cannot be neglected. In this thesis, the focus is on learning a metric in possibly non Euclidian metric spaces, as the personal data of the patients are essentially made of character strings. For such spaces, nearest neighbour algorithms generally make use of the triangular inequality to prune the search space as much as possible, what reduces the decision time. This pruning is completely dependent on the metric used. While some metrics can have excellent identification performance, if their decision time is too large, they will be of little use for large populations. For this reason, a Mahalanobis[9] metric learning method for nearest neighbour algorithms balancing between identification performance and efficiency at decision time will be proposed. Our method will be shown to be a generalization of a standard algorithm for person re-identification.

The effect of keeping several examples per individual inside the population will also be studied. This will have an important effect on the error rates. First, the case where a constant number of examples per individual is kept will be addressed. Then, variations between individuals will be allowed. The impacts on the time and space constraints will also be discussed.

Finally, different approaches to incorporate the data from the hand-recognition system directly, or indirectly, into the nearest neighbour algorithm will be studied and compared. As the hand recognition system is supposed to be very slow, the challenge will be to take advantage of the additional information it provides without considerably slowing down the decision process.

Throughout this thesis, some important hypotheses will be made in order to provide more interpretable results. The effect of such assumptions will be analyzed and their validity discussed. In particular, as the prime goal of this thesis is to build an identification system for medical monitoring, which must have small error rates, this work focuses on sufficiently small populations, such that the error rates are indeed kept low.

Chapter 1

Problem setting

This chapter poses the basis of the rest of this thesis. To begin with, the specifications of the person re-identification problem will be formally defined. Then, the general algorithm used to solve it will be presented, as well as the performance metrics and the data used for its evaluation.

1.1 Specifications

I is defined as the set of all individuals. R is defined as the set of all noisy instantiations of these individuals, or individual realizations. Realizations are obtained by adding perturbations to the personal data of the individuals. Every realization $r \in R$ originates from an individual $i \in I$. We pose the function $origin(r)$ which returns the individual from which r was generated. $E \subset R$ is the set of examples that are stored in the database. Every stored example consists of a realization annotated with a label that uniquely identifies the individual it belongs to. The injective labelling function $l : E \to \mathbb{N}$ takes a stored example and returns its label. The known population P is defined as the set of all individuals that are represented by at least one example in E, $P = \{origin(e) : e \in E\}$. Individuals that are (not) in P are said to be (un)known. The input of the algorithm is a new realization $q \in R$, the set of stored examples E and the labelling function l.

The output should be:

$$O(q, E, l) = \begin{cases} l(e) \text{ if } \exists e \in E : origin(e) == origin(q) \\ unknown \text{ otherwise} \end{cases}$$

In case of recognition, q may, or may not be added to E, as an example belonging to the same individual as q is already stored. In case of a rejection, q is assigned a new label and is added to E. Labels are assigned by extending the domain of the labelling function.

The algorithm must have small training and decision times, even when dealing with millions of individuals. From now on, individual realizations will be referred to as examples or points. The input example q will be called the query.

1.2 Error types

In this section, the different kinds of classification errors are introduced. They are the following:

- False acceptance (FA): an unknown individual is recognized.

- False rejection (FR): a known individual is not recognized at all.
- False classification (FC): a known individual is recognized but as another individual.

Note that each misclassification belongs to exactly one of the error types above. For each kind of error, the corresponding error rate can be defined.

- False acceptance rate (FAR): $P(\text{FA}|origin(q) \notin P)$ that can an be estimated by $\frac{N_{FA}}{N_{unk}}$
- False rejection rate (FRR): $P(\text{FR}|origin(q) \in P)$ that can an be estimated by $\frac{N_{FR}}{N_{kno}}$
- False classification rate (FCR): $P(\text{FC}|origin(q) \in P)$ that can an be estimated by $\frac{N_{FC}}{N_{kno}}$

where N_{unk} and N_{kno} are the number of test examples that belong to an unknown and known individual, respectively.

1.3 The Data

This work is based on partially artificial data for several reasons. First, as personal data is sensitive, no online data set containing the names of the individuals as well as relevant personal information was found. The names have then to be extracted from other sources. Second, **noisy** personal data is needed, which is even harder to find. As a consequence, the data will have to be perturbed artificially. This will allow the generation of an unlimited number of examples per individual, which will be helpful for the measurements of the error rates. The two next sections detail the methodology used to build the data sets.

1.3.1 Data extraction and generation

Using different public repositories (UCL[10], ULB[11],...), the first and last names of 30.000 people were extracted. The entities for which the staff of UCL works has also been retrieved. One entity has been assigned randomly to every individual, following the proportions. These first three attributes will be combined to some attributes of the UCI Adult Data Set[12]. This data set contains census data on 50.000 people. Its original goal was to train a classifier that predicts whether individuals have an income superior to 50.000 dollar per year or not. Some of these attributes are perfectly relevant for our task. The selected ones are the age (translated into a birth year), the gender, the country, the race, the marital status, the work type and the education level. To complete the birth date, random numbers for the day and month of birth were generated. Obviously, the real distribution of these is uniform and they are uncorrelated to the other attributes.

Furthermore, randomly selected mother's first name and father's first name were added. They were also taken from the UCL repository, following the proportions. Finally, a random phone number was generated.

All the names along with the entity are textual attributes. The year, month and day of birth, as well as the phone number, are integers. However, as the phone number is subject to the same kinds of error as textual variables, it will be considered as such. The gender, country, race, marital status, work and education are all categorical (even binary for gender). The feature space is summarized by table 1.1. Given the extraction process, several attributes are correlated to each other. Such attributes are depicted in the same color in table 1.1.

first	last	mom	dad	entity	num	y	m	d	sex	land	race	status	work	educ
T	T	T	T	T	T	I	I	I	C	C	C	C	C	C

Table 1.1: Feature space. Attributes displayed in the same color are correlated.

1.3.2 Data perturbation

Starting from the original data set, several noisy data sets were generated.

For the textual attributes, several common mistakes exist:

- The insertion of a letter that is not supposed to be there.
- The deletion of a letter that was supposed to be present.
- The swap of two adjacent letters.
- The replacement of a letter by another, incorrect one.

According to Damereau (1964), these standard mistakes represents 80% of transcriptions errors[13]. The probabilities of such mistakes in a word are assumed to be proportional to the number of letters of the word. The perturbation algorithm goes through each letter of the word. At each position, there is a non-zero probability to add, remove, substitute the letter at this position or invert it with the next letter. Those probabilities vary for different attributes.

Each categorical attribute was assigned a probability to be swapped with another random value in the domain of the attribute. For integers, the probability to swap the original value with another one decreases exponentially with the absolute difference between both. Different individuals may have a stronger or weaker tendency to make mistakes. To model that, a uniformly distributed random noise coefficient was assigned to each person. This coefficient varies from 1 to 3.

From now on, attributes will be assumed to be non-evolving (such as the names, the date of birth, ...). The only possible changes of the attributes between two examples of the same person are due to random perturbations (such as encoding mistakes). In other words, the set of all individuals I (and hence the stored population P) does not change in time. Different individual realizations (belonging to the same individual) generated at different moments in time, will all originate from the same data. This assumption is not always true. For example, the phone number can evolve with time.

Assumption 1 - Random noise: The noise in the data is purely random. Recurring mistakes and evolving attributes are not modeled.

This assumption was made in order not to break the correlation between attributes. For instance, there is a strong correlation between the year of birth and the marital status. Without knowledge about the real evolution of the latter, the correlation between both would have been broken by modifying durably the marital status attribute.

1.3.3 Data separation

The full data set (30.000 individuals) was split into a train set of 5.000 individuals and a test set of 25.000 individuals. The metrics that will be learned will be trained exclusively on individuals

in the train set. The error rates will be estimated on individuals of the test set. Several noisy examples of the same individuals will be generated, both for the training and testing processes. This is required to counter-balance the small sizes of the data sets. The searching experiments will be also be done on the test set.

1.4 Nearest neighbor classification

The person re-identification problem can be seen as a standard label classification problem.

The input corresponds to the different features of an example. The label of an example is then the ID of the individual they originate from. When the system knows $|P|$ individuals, the classification problem has $|P| + 1$ labels. One label per individual and a particular label, *unknown* which has no example. The other labels are represented by at least one example. Of course, every time an example is classified as unknown, the system will assign it a new label id. This means that the system must be evolving. The model has to change, at least every time a new individual is found (hence the small training time required). This task differs from standard machine learning problems on two important aspects. First, the number of labels is huge (of the same order of the number of examples). Second, there will always be the special label *unknown* which has no example assigned to it.

A well known classification algorithm is the nearest neighbor search (NNS). Let's assume that a distance function $d : (R, R) \rightarrow \mathbb{R}^+$ that computes the distance between two examples is available. A standard NNS would always assign the label of the nearest example (for a 1-NNS). For person re-identification, a new label has to be assigned to a new individual. That can be done using a threshold. If there exists an example in E that is closer to the query example than the threshold, its label will be predicted. Otherwise, the prediction will be the special label *unknown*.

Given a query $q \in R$, a set of stored examples $E \subset R$ and a threshold $t \in \mathbb{R}$, all examples $e_i \in E$ such that $match(q, e_i) \triangleq d(q, e_i) \leq t$ have to be found. E_t is defined as the set of examples that are closer to the query point than the threshold t: $E_t = \{e_i \in E | match(q, e_i)\}$. The decision algorithm is summarized by algorithm 1.

Algorithm 1: Threshold nearest neighbour classification

Predict (E, t, q, l)
- **inputs :** Set of stored examples E, threshold t, query q, labelling function l
- **output:** Predicted label
- $E_t \leftarrow \{e_i \in E | match(q, e_i))\}$
- **if** $E_t == \emptyset$ **then**
  - **return** *unkwown*
- **else if** $\forall e_i \in E_t : l(e_i) == label$ **then**
  - **return** *label*
- **else**
  - **return** $choose(P_t)$

When no example is found, the label *unknown* is predicted, which means that the individual is not recognized. If all found examples have the same label, this label is predicted. Otherwise, several examples with different labels are found. In this case, the *choose* function will select

a label. Different choices could be to select the label that appears the most, or the one of the closest example. However, as will be seen in section 3.1, the implementation of the *choose* function is not essential. The distance function d and the threshold t also have to be defined, which will be done in section 1.5 and 1.6, respectively.

One advantage of NNS algorithms is that they have no learning time. As the number of labels changes every time a new individual is found, it is very convenient. However, the decision time will increase with the size of the population, as there are more examples to examine. This will lead to the formulation of a time constraint on the decision time, first expressed in section 1.6.

1.5 Distance metrics

In this section, the distance function used by the NNS algorithm will be defined. In particular, different possible metrics for textual, categorical and integer attributes will be studied.

1.5.1 Metric properties

In non-Euclidian metric spaces[1], nearest neighbour algorithms make use of the triangular inequality to prune the search space, as will be explained in section 2.1. We then impose our distance function d to be a metric. For d to be a metric, it has to satisfy the following properties:

- non-negativity:
$$\forall x, y, d(x, y) \geq 0$$
- symmetry :
$$\forall x, y, d(x, y) = d(y, x)$$
- identity of indiscernibles:
$$\forall x, y, d(x, y) = 0 \Rightarrow x = y$$
- **triangle inequality** :
$$\forall x, y, z, d(x, y) \leq d(x, z) + d(z, y)$$

The first condition is implied by the others. If the third one is not respected, we speak about a pseudo-metric, which will be enough for our application.

A Mahalanobis[9] metric will be learned

$$D_M(x, y) = \sqrt{\mathbf{v}(x, y)^t M \mathbf{v}(x, y)}$$

$\mathbf{v}(x, y) = [d_0(x_0, y_0), d_1(x_1, y_1), ..., d_n(x_n, y_n)]$ will be referred to as the distance vector of x and y. This distance vector contains the pairwise distances between the attributes of x and y. D_M is a distance metric (technically a pseudo-metric) if the parametrization matrix M is definite semi-positive (all eigenvalues are positive or zero)[1][14]. This constraint is noted by $M \succeq 0$. Obviously, for D_M to be a metric, all d_i must be metrics as well. In the rest of this section, these d_i will be defined. Then, a simple matrix M will be proposed as baseline.

For categorical variables, the most obvious distance metric is

[1] As was seen in section 1.3.1, the feature space is clearly non-Euclidian.

$$d_c(x,y) = \begin{cases} 0 \text{ if x==y} \\ 1 \text{ otherwise} \end{cases}$$

This distance function trivially satisfies the symmetry and identity of indiscernible properties. Furthermore, the triangle inequality also holds. Indeed, if $d_c(x,y) = 0$, then $d_c(x,y) \leq d_c(x,z) + d_c(z,y)$ because $d_c(x,z) \geq 0$ and $d_c(x,z) \geq 0$. If $d_c(x,y) = 1$, then $x \neq y$. So $x \neq z \vee y \neq z$ which implies $d_c(x,z) == 1 \vee d_c(z,y) == 1$. Finally, $d_c(x,z) + d_c(z,y) \geq 1 = d_c(x,y)$ is obtained $\square$.

For the integer variables, the standard Euclidean distance $d_{int}(x,y) = |x - y|$ will be used, for which all the properties holds.

Concerning the textual attributes, the choice of the metric is more open. Different alternatives are[15]

- Hamming distance: counts the number of positions at which the characters are different.
- Levenshtein distance: counts the number of insertions, deletions or substitutions needed to transform the first string into the second.
- Damerau-levenshtein distance: also allows inversions of two adjacent characters.

The Damerau-levenshtein distance seems the more appropriate to deal with human error. Moreover, it does satisfy the triangular equality, as it is an edit distance. However, these four kinds of errors were explicitely used in the noise generation process. In real data sets, there will be noise that cannot be directly reflected by the string distance function. As a consequence, the levenshtein distance will be used for string comparison.

One problem with the levenshtein distance is that it is not normalized. People with longer names will make more errors in them and therefore, the levenshtein distance between their examples will be bigger. As a consequence, they will be more often rejected than people with shorter names. This will have a negative impact on the FRR. Li and Liu (2007) proposed a normalized levenshtein distance that satisfies the triangular inequality[16].

$$d_{norm}(x,y) = \frac{2 * d_{lev}(x,y)}{|x| + |y| + d_{lev}(x,y)}$$

Unfortunately, it was experimentally observed that the performance of this normalized distance is in fact worse than the performance of the standard levenshtein distance. The comparison is illustrated in Annexe A.

Finally, as all d_i do not have the same ranges of values, each d_i has been scaled such that the mean distance between the attributes i of different individuals is equal to 1.

1.5.2 Default metric

Now that the distance functions between textual, integer and categorical attributes have been defined, the optimal matrix M has to be learned. As mentioned in the introduction, finding the best metric is not trivial and requires knowledge that was not discussed yet. As a consequence, a very simple, and understandable metric will be first developed.

We will restrict ourselves to a particular form a Mahalanobis distance. We want the final distance to be equal to a weighted sum of the pair-wise distances between attributes.[2] This is the case if M can be decomposed into $M = ww^t$ with w a weight column vector. Then,

$$D_M(x,y) = \sqrt{\mathbf{v}(x,y)^t M \mathbf{v}(x,y)} = \sqrt{\mathbf{v}(x,y)^t w w^t \mathbf{v}(x,y)} =$$

$$\sqrt{(w^t \mathbf{v}(x,y))^2} = w^t \mathbf{v}(x,y) = \sum_i w_i * d_i(x_i, y_i)$$

Each weight w_i should depend on how reliable, and how distinctive is the information provided by the attribute i. If the variable i is very noisy, its weight should be small. Also, if it allows to discriminate well the examples, the weight should be large. For example, if a binary attribute is nearly always true, the information about the patient learned by looking at this attribute is small. Hence, the small weight.

A straightforward weighting scheme would then be to assign attribute i the weight $w_i = \frac{E[d_i(x_1,y)]}{\sum_i d_i(x_1,x_2)^2}$ where x_1 and x_2 belong to the same class, whereas y does not. The numerator favors attributes with big distances between examples of different classes. The denominator favors attributes that are subject to small amounts of noise. Note that this metric is scaled. If we pose $d_i \leftarrow x * d_i$, w_i will be divided by x such that $w_i * d_i$ remains unchanged. The distance distributions obtained with this metric can be seen on figure 1.1(b), compared to the case where all w_i are equal (figure 1.1(a)). For both metrics, the weights have be scaled such that $\sum_i w_i = 1$. They were learned on the same training set consisting of 250.000 pairs of examples sharing the same label and 250.000 other pairs that do not. They are then evaluated on a test set consisting of 250.000 other pairs of points for each category.

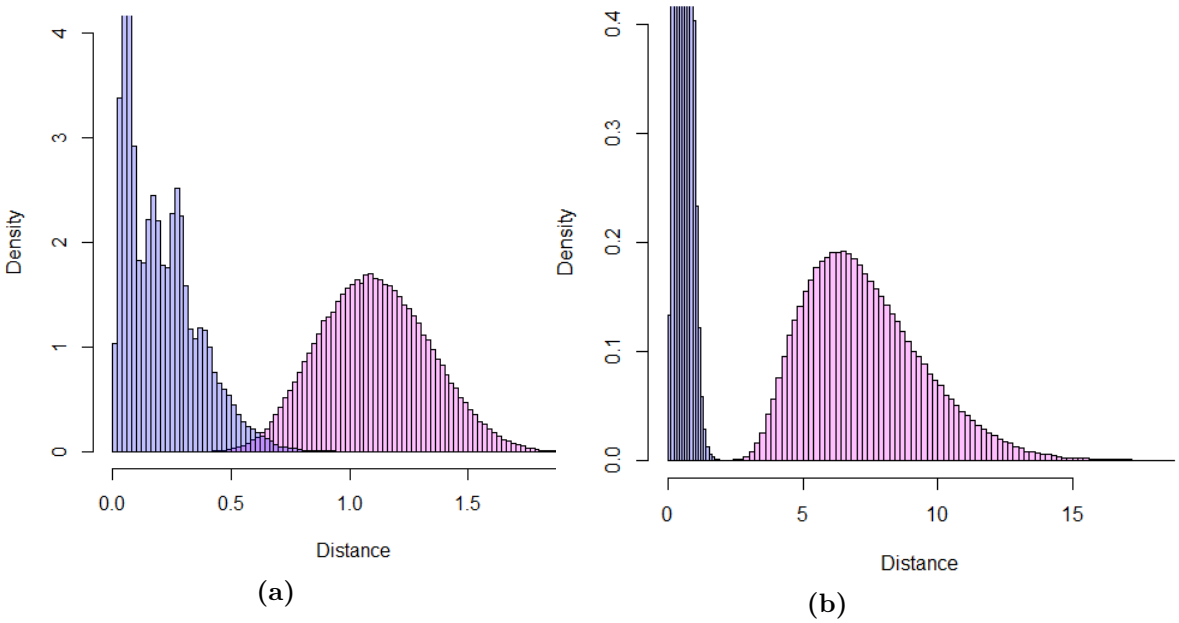


Figure 1.1: Distance distributions on the test set without proper metric learning (a) and with the default metric (b). The blue curve corresponds to distances between examples of the same individual. The red curve represents the distances between two examples of different individuals.

[2] That was chosen rather than $M = diag(w)$ because, experimentally, the latter gives slightly worst results.

As can be seen, without a proper metric learning, the two curves are largely crossing. With the custom weights, they are separated a lot better. Nonetheless, the careful reader may have noticed that this default metric is flawed. Indeed, the weight given to an attribute with zero noise would be infinite. Under no circumstances should this metric be utilised in an end application. It will only be used as a starting point. Most general results will be developed using it. Thanks to these results, it will be easier to define more precisely what a good metric should be. A more advanced metric learning scheme will then be proposed in chapter 5, which will improve the results that will be obtained with this default metric.

1.6 Constrained optimization problem

The NNS algorithm that will be used was presented in section 1.4. Its decision rule depends on a decision threshold that has not been fixed yet. In this section, the latter is expressed as the solution of a constrained optimization problem: the error rates are minimized while satisfying an expected time constraint.

Let FAR(x,t) be the FAR of the NNS algorithm on a population of size $x = |P|$ with a threshold t. FRR(x,t) is defined in the same way. Let T(x,t) be the expected decision time for a threshold t and a population of size x. A common choice when two error rates have to be balanced[3] is to maximize their F-score[17]. We are then looking for $t^*(x)$ that maximizes:

$$F(x,t) = (1+\beta^2)\frac{(1-FRR(x,t)) * (1-FAR(x,t))}{\beta^2 * (1-FAR(x,t)) + (1-FRR(x,t))}$$

subject to $T(x,t) < \eta$ where η is the time bound on the expected waiting time.

$T(x,t)$ will be analyzed in details in chapter 2 while the behaviour of $FAR(x,t)$ and $FRR(x,t)$ will be studied in chapter 3. Chapter 4 will then assess the effects of the time constraint. A possible space constraint will also be formalized.

[3] As will be seen in section 3.1, the FCR is bounded by the FAR, and hence, is not considered explicitly.

Chapter 2

Searching

In this chapter, different searching algorithms, aiming at finding every example closer to the test example than the threshold, will be analyzed. As will be shown in section 4.2, the time constraint will have disastrous effects on the error rates. It is thus essential to reduce its impact, by developing search algorithms as efficient as possible. First, the general filtering equation will be developed. Section 2.2 will be devoted to the understanding and measurement of the pruning, as well as the prototype selection algorithms. Then, two different data structures allowing for an efficient pruning will be analyzed. The assumption that the time is measured by the number of distance computations will be made. In such a case, a simple algorithm with an asymptotic logarithmic time complexity will be derived.

2.1 Filtering equation

For the decision time to be small, the query example cannot be compared to every example in the database. A filtered, or pruned example is a stored example to which the distance to the query does not have to be computed.

Suppose that the distance function d is a metric. It thus verifies the triangular inequality

$$\forall x, y, z, d(x, y) \leq d(x, z) + d(z, y)$$

Let p be an example for which the distance to every other example in the database is known. Such an example is called a prototype. As the algorithm searches only for examples that are closer than the threshold t from the query q, all examples e satisfying

$$d(e, p) > d(p, q) + t \vee d(e, p) < d(p, q) - t \tag{2.1}$$

can be filtered. Indeed, equation 2.1 implies $d(e, q) > t$. The proof is sketched below.

From the triangular inequality (using e as the intermediate example), $d(e, q) > d(p, q) - d(p, e)$ can be found. Supposing the right part of equation 2.1 , $d(e, q) > t$ is well obtained[1]. Using q as the intermediate example, $d(e, q) > d(p, e) - d(p, q)$. Supposing the left part of equation 2.1, $d(e, q) > t$ is found again $\square$. The filtering condition for a set of prototypes $B \subset E$ is then

$$\exists p \in B : t < |d(e, p) - d(p, q)| \tag{2.2}$$

as there needs to be only one prototype $p \in B$ satisfying equation 2.1 in order to prune example e from the search.

[1]The symmetry property of a metric distance was also used.

2.2 Data analysis

In this section, the filtering possible will be analyzed and different prototype selection algorithms will be studied.

The evolution of the search candidates for a particular query can be seen on figure 2.1(a-b-c). The white histogram corresponds to the distance distribution between the query and all examples. Each sub-histogram has been obtained by filtering all examples that satisfy equation 2.2 with different series of random prototypes and $t = 1$. The fourth histogram (d) was obtained with the same sequence of prototypes as the first one, but with another query.

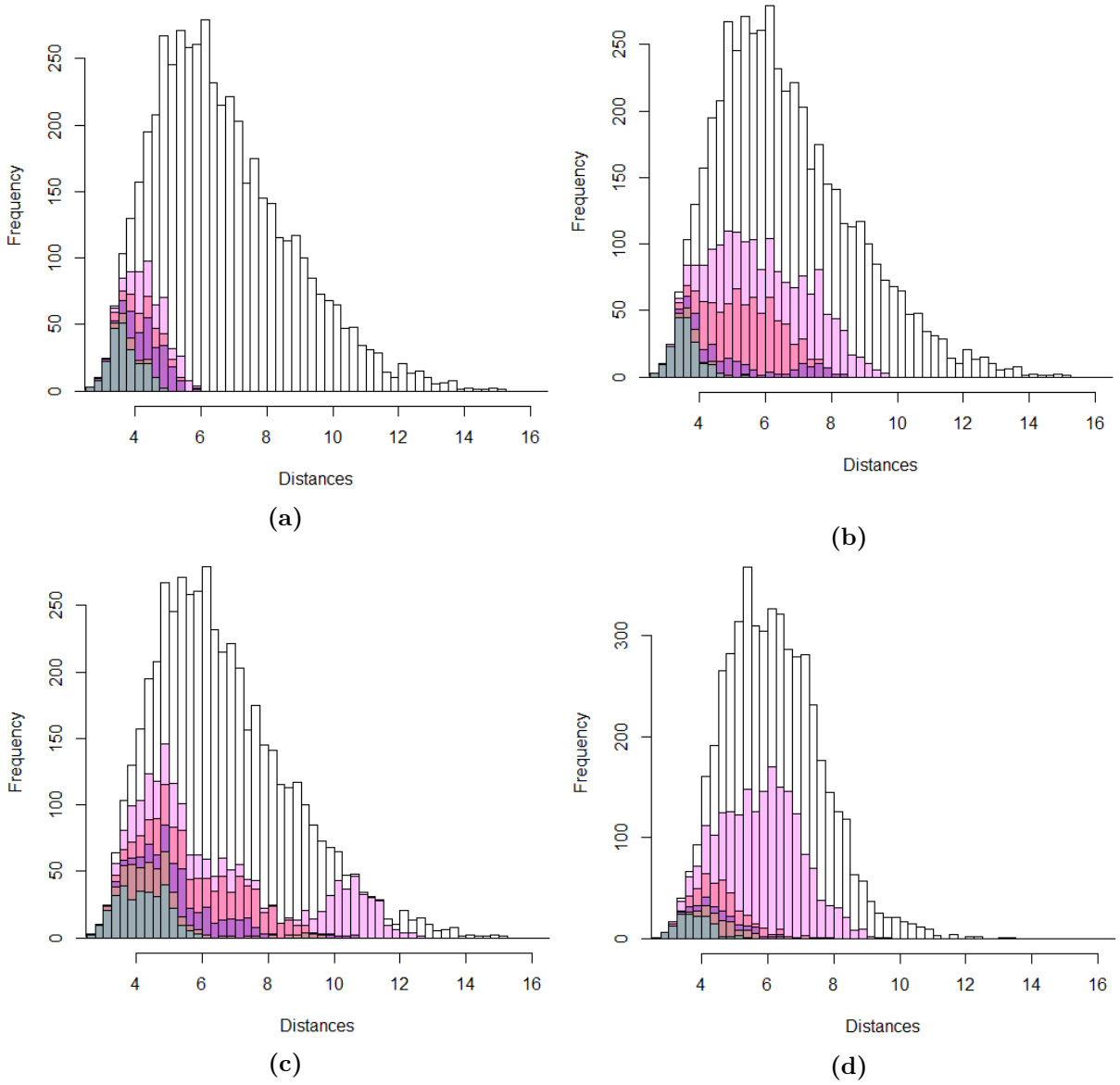


Figure 2.1: Evolution of the distance distribution between a query and the search candidates after 0 to 5 filtering steps. The histograms (a-b-c) are obtained with the same query, but with different series of prototypes. The last one (d) was built with the same prototypes as the first one, but with another query.

Some observations can easily be made:

- The filtering efficiency critically depends on both the chosen prototypes and the query.
- Unfiltered examples tend to be the ones that are the closest to the query.
- The filtering efficiency rapidly decreases with the number of iterations.

In the rest of this section, each of these observations will be justified. Different algorithms for prototypes selection will also be proposed and compared.

Prototype points are taken from the same space as the queries. As a consequence, they have the same distance distribution (white histograms of figure 2.1). Recall that all examples e such that $d(e,p) > d(q,p) + t$ or $d(e,p) < d(q,p) - t$ can be filtered. As the distance distribution behaves roughly like a Gaussian, more examples will be pruned if $d(q,p)$ is either very small, or very big. In the worst case, $d(q,p)$ will be equal to the mean of the distance distribution of p, which will minimize the number of examples that satisfy $t < |d(e,p) - d(p,q)|$. This would not be the case if the distance distribution was uniform, for example.

Specifically, the best prototypes for a query q are the ones such that $|d(q,p) - mean_p|$ is large, where $mean_p$ is the average distance between the prototype and all other examples. This justifies observation number one. As, we cannot have a separate set of prototypes for each possible query, our set(s) of prototypes should contain points with a distance distribution with high variance.

The proportion of 1000 test examples filtered by 200 different prototypes (averaged on 50 queries), in function of the variance of the distance between these examples and prototypes can be seen on figure 2.2. Clearly, prototypes with higher variance filter the best. The results are sensibly the same for different thresholds, except that the pruning is better for the smallest ones, as implied by equation 2.2.

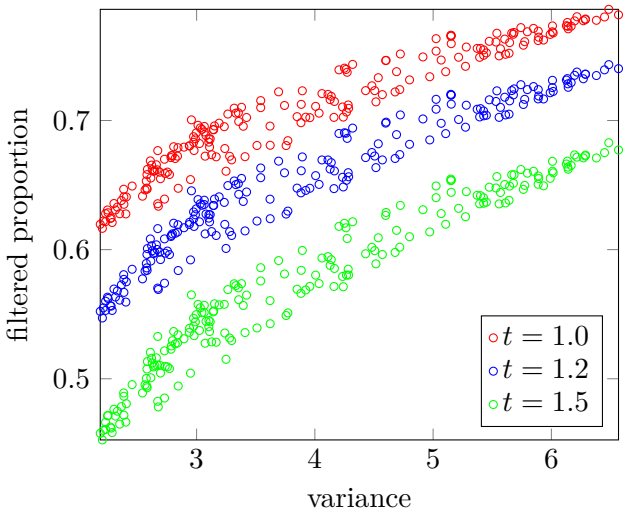


Figure 2.2: Fraction of 1000 examples that are filtered by 200 different prototypes, in function of the variance of the distance between the prototypes and the examples. The results are averaged for 50 different queries.

From the triangular inequality, the following relation can be obtained:

$$d(p,q) - d(e,q) \leq d(e,p) \leq d(p,q) + d(e,q)$$

Intuitively, $d(e,p)$ is constrained into a box with center $d(p,q)$ and side $d(e,q)$. If $d(e,q)$ is small, $d(e,p)$ is tightly constrained. Unfortunately, to filter e, $d(e,p)$ must be outside the box with same center and side t. This means that, when $d(e,q)$ decreases, the intervals of $d(e,p)$ that would cause a filtering reduce themselves in size. Specifically, to filter e we must have either $d(p,q) + t < d(e,p) \leq d(p,q) + d(e,q)$ or $d(p,q) - d(e,q) \leq d(e,p) < d(p,q) - t$. Intuitively, points that are close to the query will be approximately at the same distance to other points as the query is, and hence, will be harder to filter. As a consequence, after pruning, the remaining points will be close to the query (observation number two). For the first few filtering steps, it can be observed on figure 2.1 that some points at a particular distance from the query remain unfiltered. Those points are close to the prototype, and not the query. Points close to different prototypes will not be always the same, so they will eventually get filtered.

Figure 2.3(a) was obtained by sampling randomly 200 examples, computing their distance to a given query and the mean of the difference in distance to 1000 other examples.

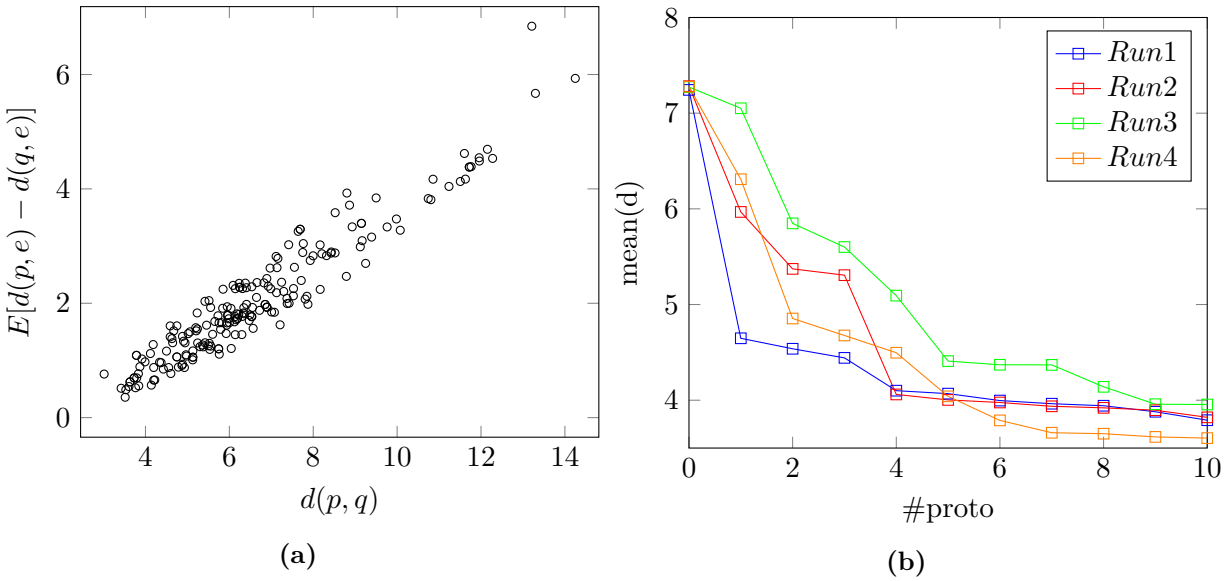


Figure 2.3: Mean of the difference of the distances between a query and 1000 examples and the distances between 200 prototypes and the same 1000 examples, in function of the distance between the prototypes and the query (a). Evolution of the mean distance between unfiltered points of figure 2.1, in function of the number of filtering steps (b).

The tendency is clear. The more a point is far from the query, the more its distance to another example will be far from the distance between the query and the same example. Points at the bottom left will the hardest to filter.

Again due to the triangular inequality, points that are close to the query will tend to be close to each other. On figure 2.3(b), the mean distance between unfiltered points of the four runs of filtering of figure 2.1 can be observed.

2.2.1 Filtering efficiency

If the filtering was equally probable for every point (given a particular query), a fixed fraction of the examples would be pruned at each filtering iteration. The number of non-filtered points would then decrease exponentially with the number of filtering steps. However, as can be seen on figure 2.1, this is not the case. This is due to the fact that some examples (see figure 2.3(a)) are harder to filter than others. After the first few steps, nearly all of the far away examples will already be filtered. As the far examples get pruned, remaining examples will get closer and closer to the query, and so will become harder to filter. The fraction of examples filtered at each step decreases (observation number three).

On figure 2.4 can be seen the filtering probabilities of a sample of 600 points, for a particular query and a threshold of 1.2, in function of the distance between the examples and the query. This confirms well that close examples are harder to filter.

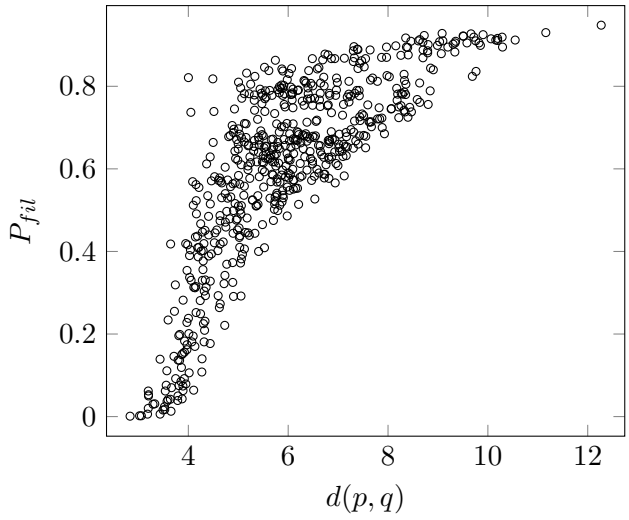


Figure 2.4: Filtering probabilities of 600 examples for a particular query and a threshold of 1.2, estimated with 1000 prototypes, in function of the distance between the filtered examples and the query.

The expected amount of examples that are not filtered after n_p filtering steps (when prototypes are chosen randomly) is equal to

$$Frac_t(n_p) = \sum_{i=1}^{|E|}(1 - p_i)^{n_p} \tag{2.3}$$

When n_p grows, equation 2.3 behaves more and more like an exponential. Indeed, the dominant term (the minimum probability to be filtered) will be more and more dominant. At the limit,

$$\lim_{n_p \to \infty} \sum_{i=1}^{|E|}(1 - p_i)^{n_p} = (1 - p_{min})^{n_p} \tag{2.4}$$

which is an exponential. Figure 2.5(a) depicts evolution of $Frac_t(n_p)$, up to 400 prototypes. The results are averaged for 50 queries. For each query, the filtering of 5000 examples was recorded.

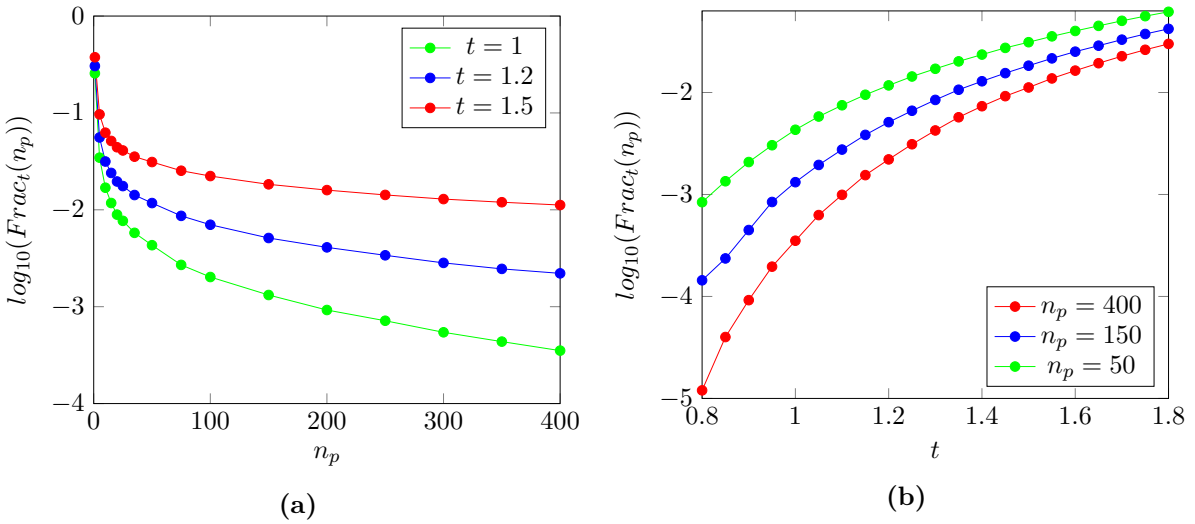


Figure 2.5: Expected fraction of unfiltered examples for up to 400 filtering steps (a). (b) represents the same fraction, but in function of the threshold, for 3 different numbers of prototypes.

As can be seen, $Frac_t(n_p)$ is not really an exponential, but tends to it. On figure 2.5(b) is shown the reduction in pruning that arises when the threshold is increased. This reduction decreases when t increases. A fixed difference in threshold will be more significant for small thresholds than for large ones. It will take its importance later, when the maximum allowed thresholds given a time constraint will be computed (section 4.2).

Figure 2.6 depicts more clearly the exponential behaviour of $Frac_t(n_p)$ for a larger number of prototypes.

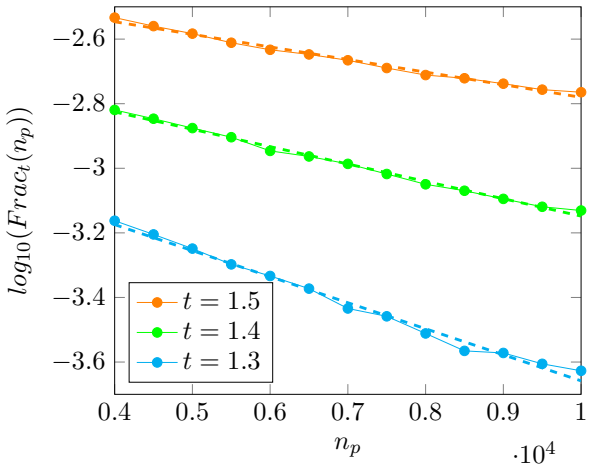


Figure 2.6: Evolution of the fraction of unfiltered examples for a number of prototypes between 4000 and 10000.

Finally, as points that are close together have a higher chance to be all approximately at the same distance from other points, prototypes that are close together will filter the same points. To have a good chain of filtering, prototypes should be far from each other. Suppose that every point that has been filtered for a query q, a prototype *proto* and with a threshold of 1.2 has been recorded. Figure 2.7(a) shows the proportion of points filtered by another prototype p that are

also filtered by *proto*, in function of $d(proto, p)$. Clearly, close prototypes will filter massively the same points, whereas far prototypes filter different points. On figure 2.7(b) is depicted the fraction of points that are filtered by p and that were not filtered by *proto*. As can be seen, far prototypes do a better job than close ones.

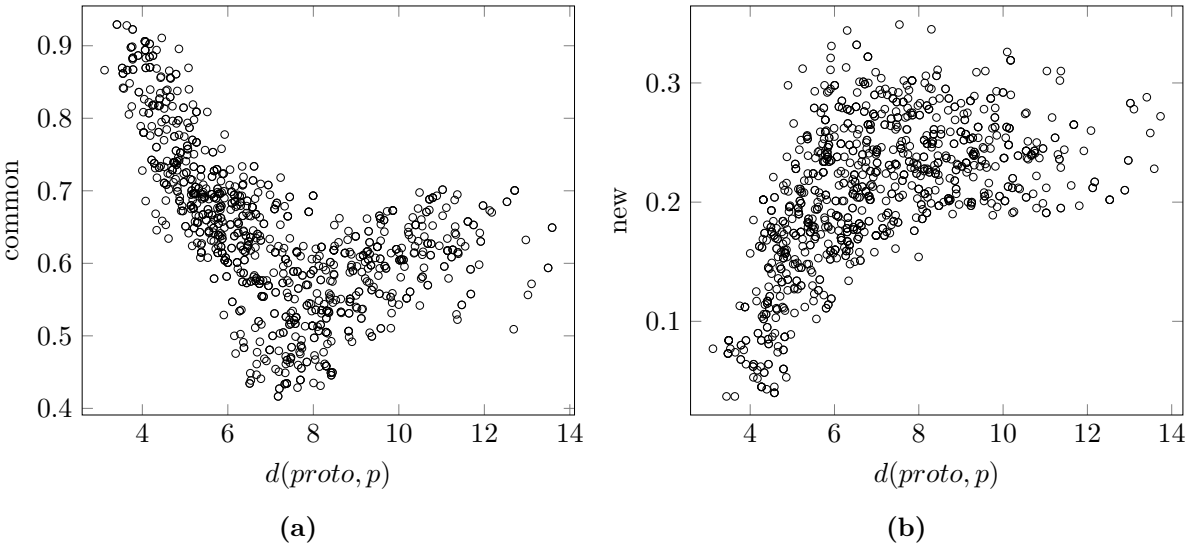


Figure 2.7: Fraction of examples that are filtered by both *proto* and new prototypes, in function of the distance between these new prototypes and *proto* (a). Fraction of examples that are filtered by the new prototypes and not by *proto*(b).

2.2.2 Prototype selection

In order to find a set of prototypes that are far from each other, the first prototype can be chosen randomly. Then, the following prototypes are chosen via this greedy rule [18]:

$$p_i = argmax_{e \in E-B} \sum_{k=1}^{i-1} d(e, p_k) \tag{2.5}$$

where E is the set of stored examples and B the set of prototypes already chosen. The results can be seen on figure 2.8 for a threshold of 1.5. The prototypes were selected among 10.000 points, distinct from the 50 queries and 1000 test points. In blue is the number of unfiltered points after a certain number of filtering steps, using selection rule 2.5. The green and orange curves were obtained with randomized selection, starting with the same random initial prototype. As you can see, the algorithm outperforms a random selection but only slightly.

The red curve was built with another selection algorithm. The latter uses a set of data points and queries and selects at each step the prototype that would lead to the maximum filtering. The prototypes are taken from the same set as previously. In such a case, the first prototype does not have to be selected randomly. In purple is represented the pruning obtained when this selection rule has been tested on its training set. As can be seen, this method brings better results. Unfortunately, it is more computationally intensive.

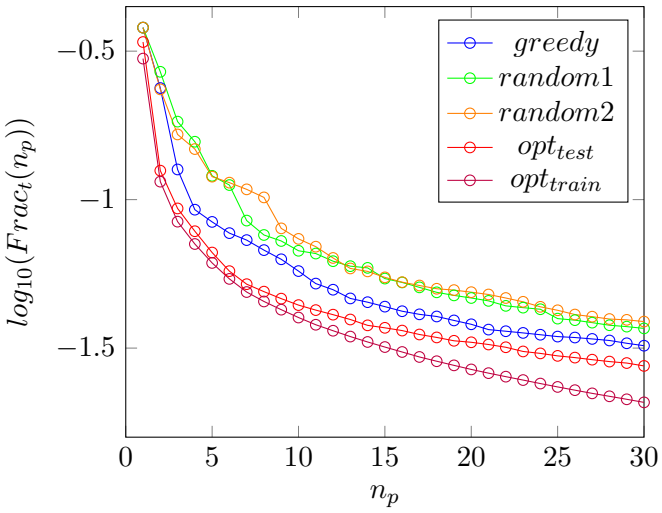
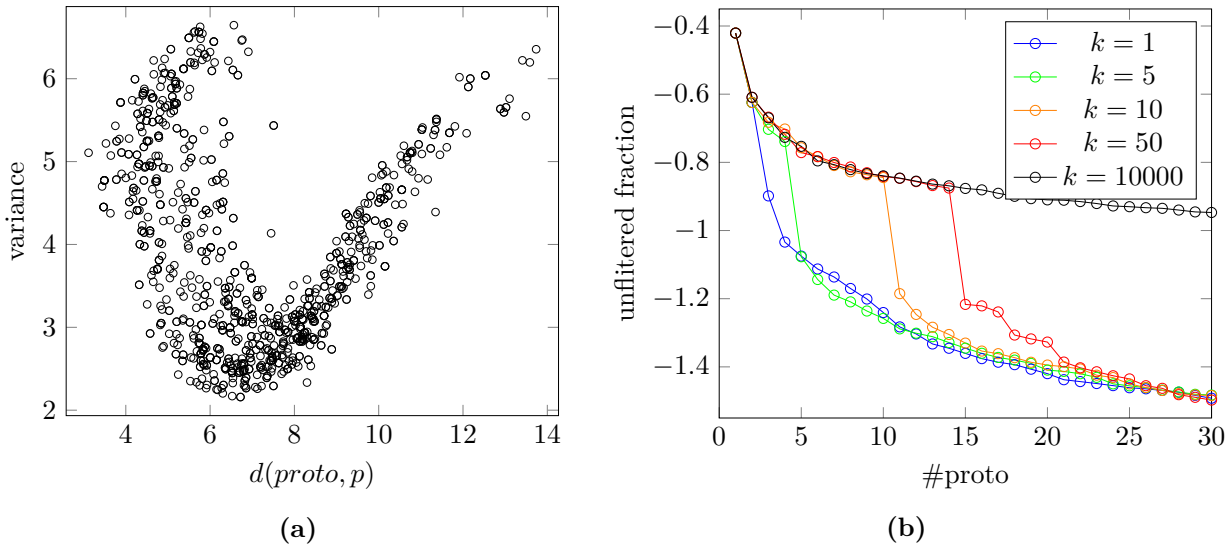


Figure 2.8: Fraction of unfiltered examples (among 5000), after up to 30 filtering steps. In blue are the results obtained with the greedy prototype selection among 10 000 possible prototypes. The green and orange curves were obtained with randomized selection. The results are averaged for 50 queries, with a threshold of 1.5. The red curve is obtained via the more optimal selection, evaluated on the same test set and trained with 1000 other points and 50 other queries. In purple are the results of this strategy on its train set.

Previously, it has been seen that prototypes with high variance filter better. That was not taken into account in the greedy rule 2.5. However, this has only a small effect, as prototypes that are far away from the current prototypes[2] will also have a high variance, as can be seen on figure 2.9(a). And indeed, the filtering efficiency barely varies when using a modified selection rule: the prototype with the highest variance among the k best prototypes found with rule 2.5 is selected. The results are displayed in figure 2.9(b), and are indeed equivalent to rule 2.5 (k=1). In black is the filtering obtained with k=10000 (the prototype with the highest variance is selected at each iteration). The results are really poor. It turns out that all the selected prototypes are really close together. This is also the reason why the filtering is bad for the first filtering steps with a lower k.

Until now, in order to filter any example e, the distances between e and the prototypes have to be looked at directly. For the search to be complete, every data point has to be processed individually, which can be harmful for large data sets. The two next sections will develop different data structures that will allow the pruning of bunches of examples at once.

[2]Or close, but close ones will filter the same points



(a) **(b)**

Figure 2.9: Variance of 1000 prototypes in function to their distance to *proto* (a). *proto* is the same as in figure 2.7. Results of the modified greedy rule (b). The experimental procedure is the same as for figure 2.8. The variance of the prototypes were estimated with 1000 examples, distinct from the test set and the queries.

2.3 BK trees

Our first attempt to filter efficiently examples will use the data structure known as the BK tree[19] [20] [21]. Every node of such a tree contains an example of the population. The edge between a child node and its parent is labeled by an integer, the distance between the example of the child and the example of the parent. Nodes that have the same parent are all at the same distance from its ancestors. An example inside a node that has some children is called a key. The keys of the tree can play the role of prototypes, as every distance to their children is known. A toy BK tree can be seen on figure 2.10.

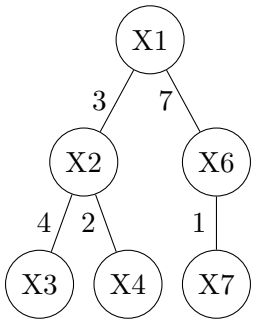


Figure 2.10: Toy BK tree containing 6 examples.

Example X2 is at distance 3 from X1, X3 is at distance 4 from X2 and at distance 3 from X1 (like X2) because the node of X3 is a child of the node of X2.

Suppose that all examples closer than 2 to a new example, X8, have to be found. This is called a range query. First, $d(X1, X8)$ is computed. Let's say it is 4. Then X6 and X7 can be pruned because $7 > 4 + 2$. Now, $d(X2, X8) = 1$ is computed. Again X3 can be filtered, but not

X4. If X4 is closer than the threshold (2), it is added to the return set.

Adding a new example to the BK tree is easy and can be done in O(log(n)) time (where n is the total number of examples, or nodes). Existing edges are followed until there is no more edge with the correct label. Then, a new node is created and the example is assigned to it. The new edge is labeled by the distance between the example of the node's parent and the added example. The space taken by the tree is in O(n).

BK trees suffer from several major problems. First, the distance metric must take only integer values. Second, when the distance between two examples can be big, the tree will be extra flat. Each new layer gives a new opportunity to filter some examples, so a tree with only few layers is not advised. To overcome these difficulties, we propose a generalization of the BK tree which we call the boxed BK tree.

2.3.1 Boxed BK trees

The principle is simple. The edges are no longer labeled by integers, but by intervals, or boxes. Large boxes will increase the height of the tree, whereas small boxes will decrease it.

Unfortunately, the larger are the boxes, the less effective the filtering at each level will be. Indeed, to filter a box, every possible distance inside the box must be allowed to be pruned. Actually, the minimum and the maximum of the box are the only values that have to be checked, but the problem remains the same: how to choose the optimal box size ? We will restrict the problem to the one where every box has the same width.

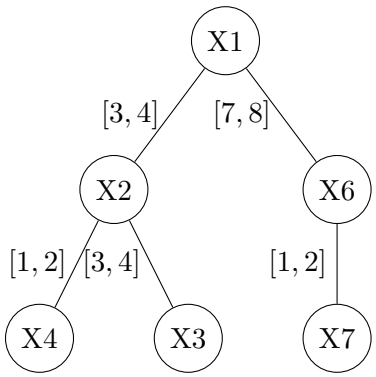


Figure 2.11: Toy boxed-BK tree containing the same example as the figure 2.10. The box width is set to 2.

On figure 2.11 is the boxed version of the toy BK tree presented before, with a box width of 2. Unfortunately, in this case, only X6 and X7 will be filtered. Indeed, recall that $d(X6, X8) = 4$ and $t = 2$. As $4 + 2 < 7$ (the minimum of the box), X6 can be filtered. However, X3 cannot be filtered anymore because $1 + 2 \geq 3$. The searching algorithm is described more precisely by algorithm 2.

Algorithm 2: Searching in boxed BK trees

```
RangeQuery (r, t, q)
    inputs : root r, threshold t, query q
    output: E_t ≜ {e_i ∈ r | d(q, e_i) ≤ t)}
    set ← ∅, d_r ← d(q, r.example)
    if d_r ≤ t then
        set ← set ∪ {r.example}
    foreach child in r.children do
        if d_r ≤ child.box.max + t  &  d_r ≥ child.box.min − t then
            set ← set ∪ RangeQuery(child, t, q)
    return set
```

2.3.2 Performance of BK trees

The filtering performance of the boxed BK trees has been measured with different box sizes. The measurements were done on a tree containing 1000 examples, with 100 distinct queries. Without filtering, 100% of the tree would be explored.

As can be seen on figure 2.12(a), when the boxes are too big (3), or too small (0.05), the filtering efficiency decreases. Boxes too big decrease the filtering possible at each node, but, as already mentioned, they increase the depth of the tree, which allows for more filtering attempts. Small boxes will have an optimal filtering at each node, but the tree will have few levels. Here it seems that a box width of 1 is a good choice. Nonetheless, the performance difference is not really critical.

Using this 1 box size, the evolution of the number of distance computations and node explorations in function of the size of the tree is displayed on figure 2.12(b,c,d) (plain lines). The results are obtained by averaging the measurements for 50 different queries that are not present in the tree. The plain lines are obtained with the standard BK trees. Unfortunately, the keys of the BK trees along a path (from the roof to a leaf) get closer and closer together when the depth increases. Indeed, they are at the same distance from an increasing number of points and that, as explained in section 2.2, implies they will be close to each other as well. As was seen in section 2.2.1, close prototypes will filter the same points, reducing the pruning effectiveness.

A modified version of BK trees, called fixed-queries BK trees [21] was used to solve the issue. In such a tree, every key of the tree at a given depth is the same. This way, only a small number of keys is kept, and they can be chosen to maximize the expected filtering. The distance from the query to every key can be computed only once, before starting the search. When using such a tree, the distance to an actual data point is computed only when a terminal node is reached. This amount will be reduced as the filtering will be more efficient. However, a lot of nodes will be explored (all non-terminal ones) without computing distances. This can be harmful if distance computations are computationally easy.

As can be seen on figure 2.12 (dotted lines), the filtering is much more efficient with optimized keys. However, the total number of node explorations is bigger, as predicted. The height of the tree is logarithmic in function of its number of examples, so the number of keys to find is small. As a consequence, the optimized rule developed in section 2.2.2 was used. The results obtained with the random selection rule are displayed with dashed lines. It seems that the

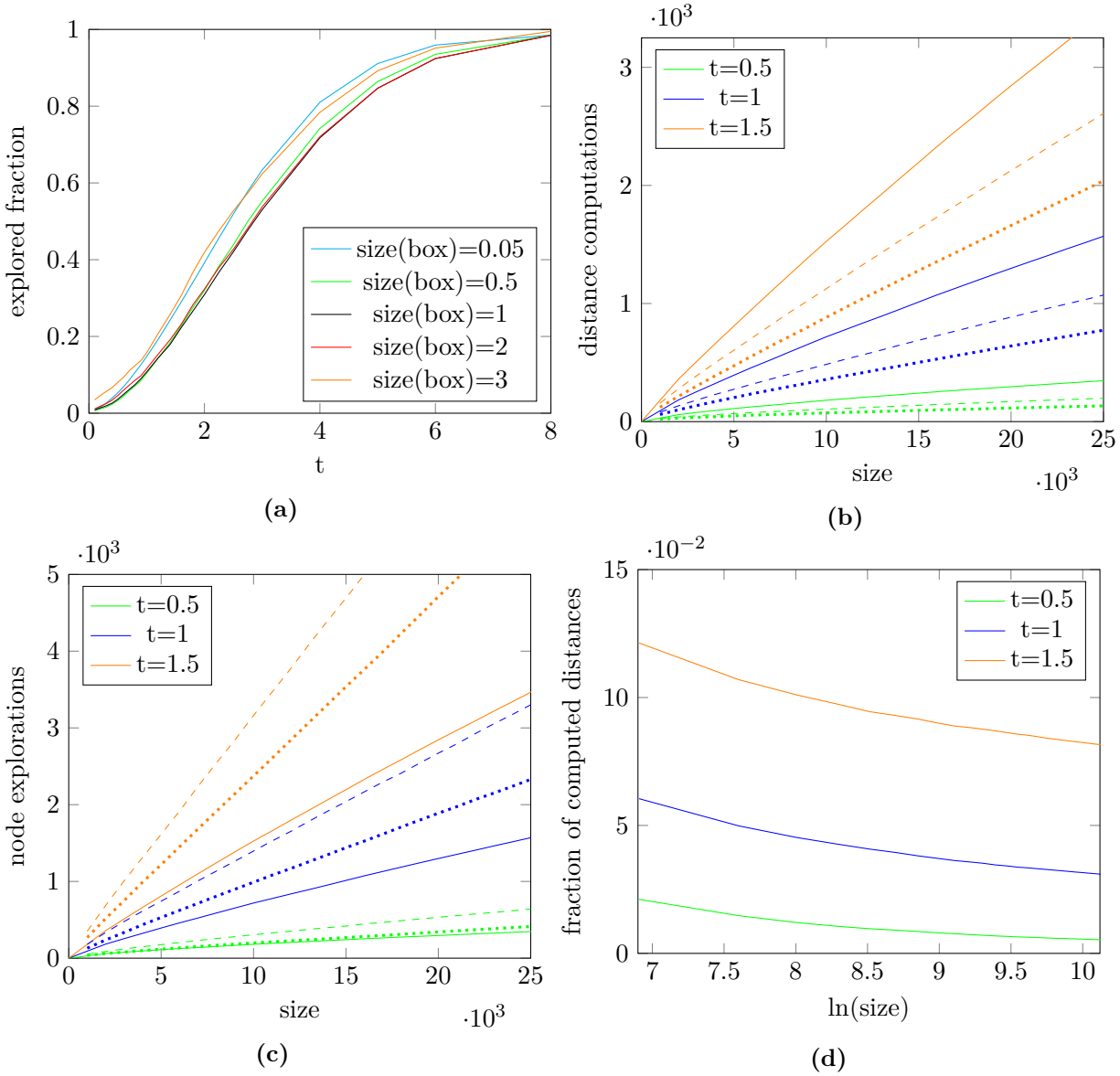


Figure 2.12: Explored node fraction in function of the threshold, for different box sizes (a). The measurements were done with a set of 1000 points, with 100 distinct queries. Number of distance computations (b) and node explorations (c) in function of the size of the tree. The plain lines corresponds to the standard BK-trees. The dashed and dotted lines are obtained with fixed-queries trees using a random prototype selection and the optimized rule of section 2.2.2, respectively. The results were averaged for 50 queries, not present in the tree. Fraction of the distances that are computed, for fixed trees, using the efficient selection rule, in function of the natural logarithm of the size of the tree (d).

time complexity is linear for both the node explorations and distance computations. In fact, it is sub-linear, but it can be barely noticed. The fraction of the points to which a distance is computed decreases well, as can be seen on figure 2.12(d), with optimized prototypes.

The complexity of BK trees is hard to define, due to their extreme unbalance. However, they provide efficient filtering. The fixed box size is probably a limiting factor.

2.4 Vantage point trees

BK trees weakness is their unbalance. Vantage point trees[22] solve this problem.

At each node, the data is split into two parts. The first part will be closer to the node's example than a given distance. The other part will be farther. The median distance will be used as threshold in order to keep the tree as balanced as possible. To compute the median, leaf nodes have to contain a set of examples, called a bucket. When a new example is added, the algorithm goes through the tree until it reaches a leaf. Then, the new example is added to the bucket. If the bucket is now full, the examples are split into two nodes. An example is picked at random and its median distance towards the other examples is computed. The left child node contains the examples that are closer than the median. The right child node contains the ones that are farther. By definition, both nodes contains the same number of nodes (+- 1). Picking an example of the bucket as key of the tree is a bad idea as explained before. As for BK trees, pre-selected prototypes are used to split the data. A toy vantage tree is represented in figure 2.13.

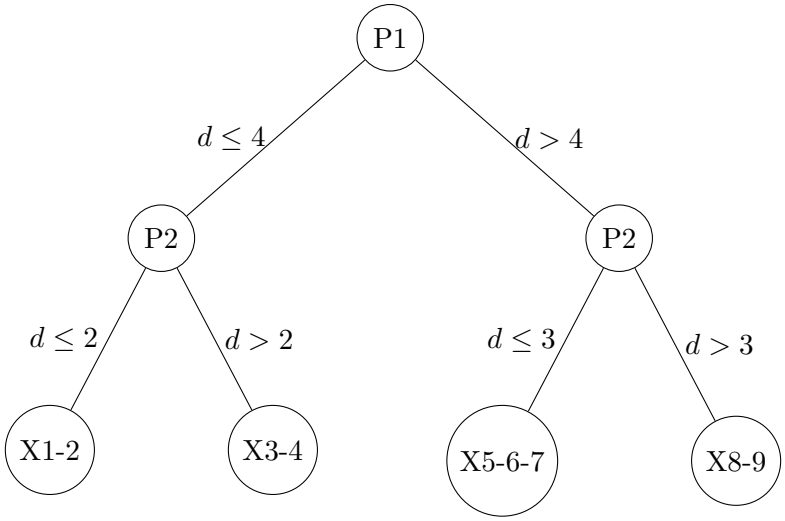


Figure 2.13: Toy vantage trees with two prototypes and 9 stored examples. The bucket size is 3.

During the search, if the split distance is greater than the sum of the search threshold and the distance between the test example and the key of the current node, the right branch can be pruned. The same is valid for the left branch if the split distance is lower than their difference. The searching procedure is summarized by algorithm 3.

Algorithm 3: Searching in fixed vantage trees

```
RangeQuery (r, t, q)
    inputs : root r, threshold t, query q
    output: E_t ≜ {e_i ∈ r | d(q, e_i) ≤ t)}
    set ← ∅
    if is.bucket(r) then
        foreach e in r.bucket do
            if d(q, e) ≤ t then
                set ← set ∪ {e}
        return set
    else
        d_r ← d(q, r.key)
        if d_r ≤ r.split + t then
            set ← set ∪ RangeQuery(r.left, t, q)
        if d_r ≥ r.split − t then
            set ← set ∪ RangeQuery(r.right, t, q)
        return set
```

Assuming that there is a fixed probability p to explore both branches at each node[3], a recurrence equation giving the expected number of visited terminal nodes can be developed. Let $C_h(h)$ be the expected number of visited **terminal** nodes of a tree with height h+1. Then:

$$\begin{cases} C_h(0) = 1 \\ C_h(h) = (p+1)C_h(h-1) \end{cases} \tag{2.6}$$

Indeed, each visited terminal node will be split into two child nodes when the height of the tree is increased by one. At least one of them will be visited, and the other one will have a probability p to be as well. This gives as exact solution $C_h(h) = (p+1)^h$. Let $i = 2^h$, the number of terminal nodes of the tree (of height h+1). Then,

$$C(i) = (p+1)^{log_2(i)} = i * (\frac{p+1}{2})^{log_2(i)} \tag{2.7}$$

with $C(i)$ the expected number of visited terminal nodes of a tree with i terminal nodes. This complexity is neither logarithmic, nor linear. Indeed, $\frac{\partial C}{\partial i} = log_2(2a) * a^{log_2(i)}$ with $a = \frac{p+1}{2}$. As $a < 1$, $\lim_{i\to\infty} \frac{\partial C}{\partial i} = 0$. This is not the behaviour of a linear function. Furthermore, WolframAlpha[23] indicates that $\lim_{x\to\infty} x * a^{log_2(x)} - log_2(x) = \infty$ when $0.5 < a < 1$ which means that the complexity is not logarithmic either. Formally, the height of the tree has to be taken into account, as the distance to every prototype has to be computed. However, this term is logarithmic in the size of the tree and becomes quickly negligible.

On figure 2.14(a) is depicted the total number of distance computations, divided by the tree size, for a $t = 1$ and different bucket sizes, using fixed-queries trees. As can be seen, the complexity is well sub-linear. A big bucket will trigger a tree with a smaller height and hence fewer computations of distances to prototypes. Moreover, it brings a better approximation of the real median value. The tree will then be more balanced. However, as the height of the tree will be smaller, the number of filtering steps will decrease. As can be seen, smaller buckets seem to be better, in terms of distance computations.

[3] And hence a probability $1 - p$ to filter one of the branches.

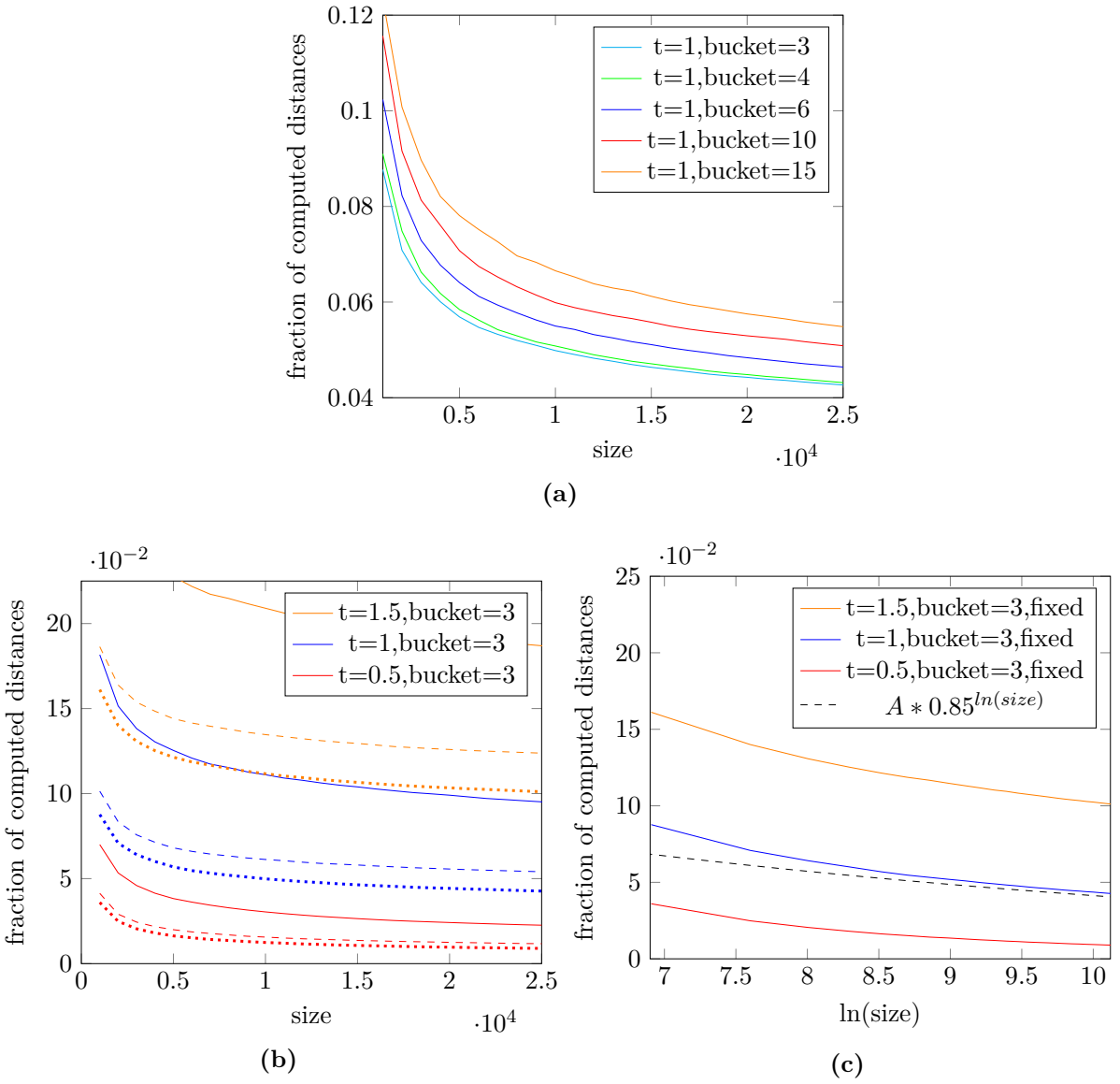


Figure 2.14: Fraction of distance computations for fixed vantage trees in function of the size of the tree as well as the bucket size (a). Fraction of distance computations for fixed trees (dashed: random selection, dotted: optimized selection) or standard ones (plain), for different thresholds (b). (c) represents the same data of fixed trees (optimized) as (b), but in function of the natural logarithm of the size of the tree. The curves are compared to the theoretical complexity.

The difference between fixed-queries trees (dashed and dotted) and standard ones (plain) can be seen on figure 2.14(b). Fixed vantage trees offer a better performance because the keys at each nodes are optimized (via the same rule as for figure 2.12 for the dotted lines, and via random selection for the dashed lines). However, they can't exactly match the filtering efficiency of BK trees, at least for sizes below 25000. On figure 2.14(c), the comparison between the experimental complexity and the theoretical one (dashed lines) is made. If $C(i) = i * (\frac{p+1}{2})^{log_2(i)}$ and $n = i * b$ is size of the tree (with b the bucket size), then $\frac{D(n)}{n}$, the fraction of computed distance (what is plotted) is equal to

$$\frac{D(n)}{n} = \frac{C(\frac{n}{b}) * b}{n} = \frac{n * b}{n}(\frac{p+1}{2})^{log_2(\frac{n}{b})} = A * (\frac{p+1}{2})^{log_2(n)} \tag{2.8}$$

with A a constant. Based on the dashed line for t=1, $\frac{p+1}{2} \simeq 0.85$, so $p \simeq 0.7$. It seems that there is a $1 - p \simeq 0.3$ probability to prune one branch of the tree, at each node. The expected pruning at each layer is then $\frac{p}{2} \simeq 0.15$, as the tree is supposed to be balanced. However, as was seen in section 2.2.1, the filtering efficiency decreases when the number of prototypes increases. As a consequence, the probability to prune a branch will decrease with the depth as well. Equation 2.8 gives the complexity only for the best case scenario where the pruning does not decrease with the depth of the tree.

2.5 AESA algorithms

If distances can be calculated efficiently, BK trees or vantage trees could offer a good solution. However, when computing the distances require high computational time, they will offer poor performances, as the amount of distance computations still remains too high. The number of filtering steps (height of the tree) is too small. In this section, the assumption that the distances are heavy to compute (as levenshtein distance is) is made and some existing algorithms that try to limit the number of such computations are reviewed.

An important note is that algorithms that will be discussed in this section look for the nearest neighbor, not all neighbors inside a range. While we used a constant threshold for pruning, they start with a big threshold and iteratively decrease it as they find points closer and closer to the query point.

The Aproximating and Eliminating Search Algorithm[24], or AESA, pre-computes all distances between train examples. AESA iteratively selects prototypes from the unfiltered points, aiming at finding prototypes as close as possible to the query. This is required to decrease the search threshold as much as possible, for a more efficient filtering. AESA can, on average, answer a query using only a constant number of distance computations. However, this requires to compute the distances between all train examples. As a consequence, the pre-processing time, as well as the space used, is in $O(n^2)$ with n the number of data points. This makes it unusable for large data sets, which is clearly the focus here. Moreover, the time required to answer a query is in $O(n)$.

LAESA[25], or Linear AESA, achieves the same expected constant number of distances computations, with a preprocessing time of $O(n)$ and a space complexity of $O(n)$. They keep a set of base prototypes to which they compute the distances to all data points. Like AESA, LAESA heuristically estimates the identity of the closest base prototype to the query. Prototypes that can be filtered are eliminated from the set of prototypes.

TLAESA[18], or Tree LAESA, uses branch and bound techniques to reduce the time complexity to $O(log(n))$, while keeping the other complexities unchanged. TLAESA computes two things during the pre-processing. First, as LAESA, it selects a set of base prototypes and computes the distance from them to all data points. Second, they build a tree representation of the database. The tree representation proposed in[18] as well as the pruning mechanism will not work well in our case, see Appendix B. Instead, vantage point trees will be used (in section 2.6).

The more data points there is, the closest the nearest neighbour will be. This means that, with more data points, the filtering efficiency of AESA algorithms will increase and counter balance the increase in data points. This can explain the claimed constant number of distances computations. In our case, the threshold value is fixed, so it will not be possible to reach that. A viable alternative to the decision algorithm 1 would have been to always predict the label of the closest example, if the latter is closer than the threshold. In such a case, a range query is not mandatory as only the closest example is sought. However, when the test example belongs to an unknown individual, no example inside the range will be found. In such a case, the search threshold of a standard nearest neighbour request would have been strictly higher during the whole search, what will provide worse performance.

LAESA uses a fixed number of base prototypes. While it may be enough to guarantee an approximately constant number of distance computations for standard nearest neighbour, it will only provide a linear complexity for a range query. For standard nearest neighbour, filtering can still be done after having exhausted the prototypes. Indeed, the threshold can still be reduced by finding points closer to the test example. For a range query, the threshold can never be decreased. As a consequence, for a fixed number of prototypes, a given fraction of examples will remain unfiltered, whatever the order of the prototypes. The number of prototypes will have to grow when the number of examples increases. In the following, the prototypes will be ordered in the same way for every query. Heuristically selecting the closest prototype to the query will not be done as in AESA algorithms. This was useful mostly to reduce the search threshold as soon as possible, which cannot be done for a range query. Reordering the prototypes in order of closeness to the query was even observed to decrease the filtering performance.

In the rest of this section, the evolution of the number of distance computations with the number of base prototypes is studied. The underlying data structure, that stores the database, is irrelevant, assuming the time taken by the algorithm is measured in terms of distance computations. The computation of $d(q, e)$ can be avoided when there exists a prototype p such that $|d(q, p) - d(p, e)| > t$. This is true iff $g(q, e) \triangleq max_{p \in B} |d(q, p) - d(p, e)| > t$ with B the set of base prototypes. With more prototypes, less distances to standard data points will be computed, as $g(q, e)$ will increase. However, the distances between the query point and all prototypes have to be computed before starting the search, increasing the number of distance computations. This means that there exists an optimal number of prototypes, possibly different for each threshold value and database size. The total number of distance computations, D_{tot}, is equal to $D_{tot} = D(n, n_p, t) + n_p$ where $D(n, n_p, t)$ is the expected number of distance computations to unfiltered points, for a threshold t, n_p prototypes and a total of n points.

$D(n, n_p, t)$ can be decomposed into $D(n, n_p, t) = n * Frac(n_p, t)$, where $Frac(n_p, t)$ is equal to the fraction of examples that are not pruned with n_p prototypes and a threshold t.

For a given threshold (the dependence in t is removed), the optimum number of prototypes does depend on the number of points. Indeed,

$$\frac{\partial D_{tot}(n_p, n)}{\partial n_p} = 0 \Rightarrow \frac{\partial D(n_p, n)}{\partial n_p} + 1 = 0 \Rightarrow n\frac{\partial Frac(n_p)}{\partial n_p} = -1 \tag{2.9}$$

The optimal number of prototype is such that $\frac{\partial Frac(n_p)}{\partial n_p} = \frac{-1}{n}$. Intuitively, the number of distance computations that are avoided by adding a prototype should be larger that one. Indeed, before starting the search, an additional distance will be computed.

The behaviour of $Frac(n_p)$ was depicted in figure 2.5. It seems to behave approximately like an exponential for large values. It can then be written $Frac(n_p) = c * e^{-a*n_p}$.

Equation 2.9 gives

$$-c * a * e^{-a*n_p^*} = \frac{-1}{n} \Rightarrow -a * n_p^* = -ln(n) - ln(c * a) \Rightarrow n_p^* = \frac{1}{a}(ln(n) + ln(c * a)) \tag{2.10}$$

So,

$$D_{tot} = n * Frac(n_p^*) + n_p^* = n * (c * e^{-ln(n)}e^{-ln(c*a)}) +$$

$$\frac{1}{a}(ln(n) + ln(c * a)) = n * c * (\frac{1}{n}\frac{1}{c * a}) + \frac{1}{a}(ln(n) + ln(c * a))$$

$$D_{tot} = \frac{ln(n)}{a} + C = O(log(n)), C = \frac{1 + ln(c * a)}{a} \tag{2.11}$$

A logarithmic time complexity is achieved if the following assumptions are met:

- The cost of computing g for every data point is negligible.
- There exists a number of prototype from which the number of distance computations decreases exponentially with the number of prototypes.

Note that $n * Frac(n_p^*) = \frac{1}{a}$. This means that, after filtering, $\frac{1}{a}$ unfiltered examples will remain, on average. This number does not depend on n.

2.6 Final algorithm

In this section, the efficient pruning of vantage trees will be combined with the logarithmic complexity in terms of distance computation developed in the last section.

Vantage trees (as well as all trees) weakness is that they can only have approximately $log_2(n)$ (up to the bucket size) steps of filtering. Usually, $log_2(n)$ will be far less than the optimal number of prototypes, given by equation 2.10. This means that too many distances will be computed. The following approach can be used to solve the problem, as is done by LTAESA (except that another tree structure is used).

First, n_p* prototypes are selected. The vantage tree is built iteratively as before, using the first prototypes as keys. However, when a bucket is reached, the relation $t \geq g(e, q)$ is still checked for every example e inside the bucket, therefore using all prototypes. $d(e, q)$ is computed only if $t \geq g(e, q)$ is true. By doing so, the number of distance computations stays logarithmic. The number of computations of the function g will increase sublinearly (following approximately equation 2.8), as vantage trees are used. However, as the number of prototypes increases, the

cost of a single g computation increases logarithmically (as the number of prototypes does). As a consequence, the overall time complexity is then approximately given by (combining equation 2.8 and 2.11)

$$T(n) \propto n * log(n) * (\frac{p+1}{2})^{log(n)} \tag{2.12}$$

which can be shown to be still sublinear, as $\frac{p+1}{2} < 1$.

For BK trees, as they are extremely unbalanced, estimating the depth of the tree can be really difficult. It cannot be guaranteed that the number of keys in the BK tree will be smaller than n_p*, especially for large box sizes. If it is bigger, too many distances will be computed, which will degrade the performance.

The number of expected distance computations can be seen on figure 2.15, in function of the size of the population.

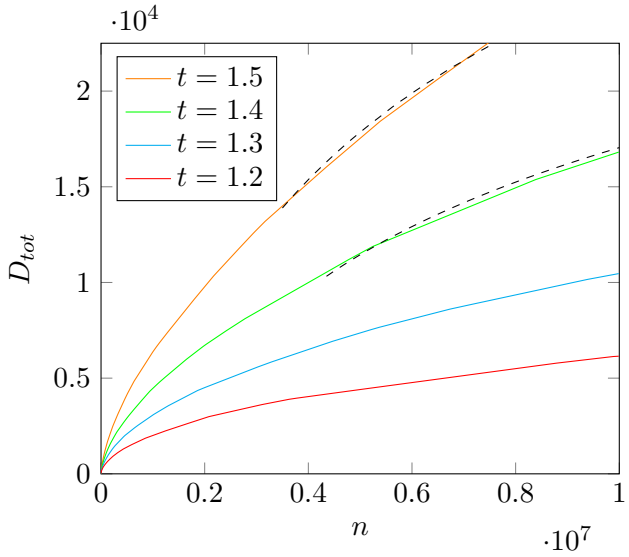


Figure 2.15: Evolution of the number of distance computations for a population of increasing size. The results were average for 50 queries, not present in the population. In black is compared a real logarithm, obtained by using the exponential approximation of $Frac(n_p, t)$ on figure 2.6.

D_{tot} closely matches the behaviour of a logarithm for large population sizes. With this strategy, the expected amount of distance computations is reduced tremendously. However, this does not come without disadvantages. Indeed, the table containing the distances between all prototypes and points has to be stored. This can take a lot of memory and will lead to the formulation of a space constraint in section 4.3.

Chapter 3

Performance analysis

In this chapter, the unconstrained optimization problem defined in section 1.6 will be solved. To do that, the behaviour of the false acceptance and false rejection rates with the decision threshold will be studied. Then, the threshold maximizing the F-score will be determined. The evolution of the optimal error rates as well as the F-score with the population size will then be looked at.

3.1 Error rates

In this section, $FRR(x,t)$ and $FAR(x,t)$ will be studied in details. Before going further, the following assumption is made (it will be released in section 6):

Assumption 2 - One example per individual: Each known individual has exactly one stored example.

From now on, e and e' will refer to stored examples of different individuals while e and e_j will be examples that originate from the same individual. $P(match(e,e'))$ then refers to the probability that two examples of different persons match. Additionally, $P_i(match(e,e'))$ refers to the probability that an example belonging to the individual i matches with an example from another individual.

The larger the database becomes, the harder it will be to recognize correctly someone. Indeed, the probability that the stored population contains an individual with a close profile to the test individual will increase. One may want to define metrics that do not depend on the size of the database. In particular, the FAR could be defined as $P(match(e,e'))$. This modified FAR, which will be referred to as FAR', can be estimated by $\frac{N_{FA}}{N_{unk}*|P|}$. As will be seen later, this is valid only if FAR' is very small, such that the probability to recognize an unknown individual as two other known individuals is negligible. FAR' does not depend on $|P|$. It is equal to the expected FAR of a database containing only one example.

Let X be a random variable equal to the distance between two randomly drawn examples[1] inside E. Using our data, its distribution can be approximated (figure 3.1(a)). The blue curve is the normal distribution with the same mean and variance as X. As can be seen, $X \sim N(\mu,\sigma)$. However, $f_X(x)$ seems to decrease faster than $N(\mu,\sigma)$ for small x and decrease slower for large x. $FAR'(t)$, or the probability that 2 random examples of different individuals match given

[1] They will originate from different individuals, because of the one example per individual assumption

a threshold t, is equal to the fraction of example pairs that are to the left of the threshold distance.

$$FAR'(t) \approx \int_0^t f_X(x)dx$$

As only the evolution of $FAR'(t)$ for very small values is interesting, the normal approximation will not be used. Obviously, $FAR'(t)$ increases monotonically with t. As mentioned in the introduction, this thesis is focused on a low error setting. Assumption 3 can then be made (and will be checked in section 3.5).

Assumption 3 - Small false acceptance rate: $\forall i \in I : P_i(match(e, e')) \ll \frac{1}{|P|}$.

It states that, for all individuals, the probability that one of their examples matches a randomly drawn example from another individual is much less than the inverse of the population size. This has for consequence that no individual has a close to 1 probability to match at least one example. Assumption 3 also implies that, for all individuals, the probability of matching two different examples can be neglected. Indeed[2],

$$P_i(\exists e', e'' \in E : match(e, e') \cap match(e, e'')) = P_i(\exists e'' : match(e, e'')|\exists e' : match(e, e'))*$$

$$P_i(\exists e' : match(e, e')) = P_i(\exists e' : match(e, e'))^2 \ll P_i(\exists e' : match(e, e'))$$

As a consequence,

$$FAR(x, t) \triangleq P(\exists e' : match(e, e')) = \sum_{i \in P} P_i(match(e, e')) \tag{3.1}$$

$FAR(x, t)$ can then be expressed in function of the population size $x \triangleq |P|$ and in function of $FAR'(t) \triangleq P(match(e, e')) \triangleq E_i[P_i(match(e, e'))]$.

$$FAR(x, t) = \sum_{i \in P} P_i(match(e, e')) = x * E_i[P_i(match(e, e'))] = x * FAR'(t) \ll 1 \tag{3.2}$$

$FAR(x, t)$ increases linearly with the population size. As $FAR'(t)$ increases with t, so does $FAR(x, t)$.

Now, let Y be the random variable that represents the distance between two examples of the same individual. Ideally, Y would always be 0. Because of the noise, E[Y] becomes greater than 0. Again, Y's distribution is approximated with our data (figure 3.1(b)).

[2]The independence between false matches has been used. It is valid as e' and e'' belong to different individuals.

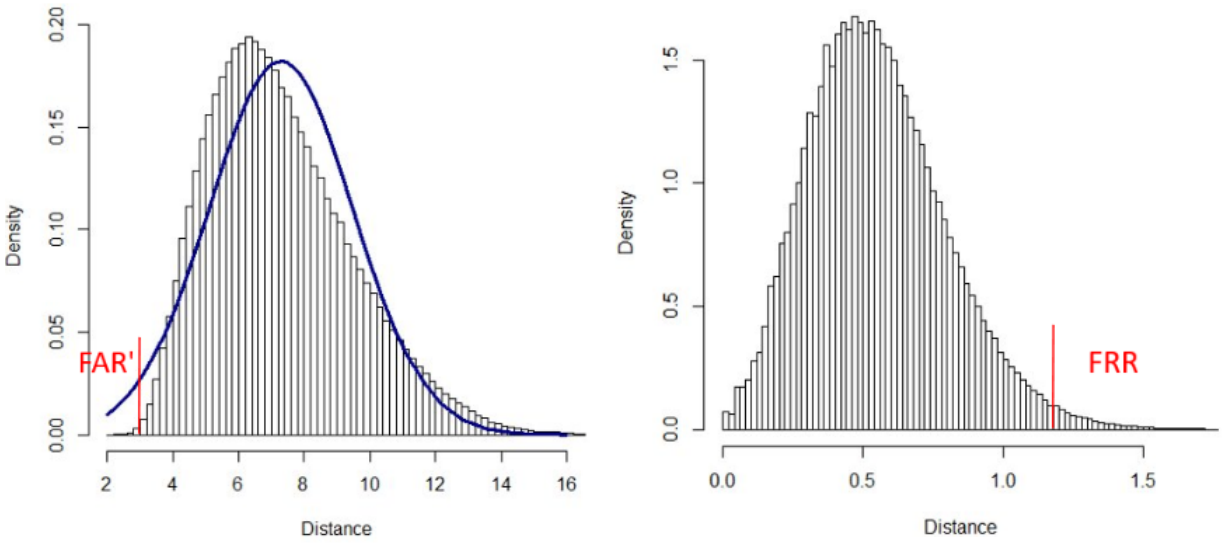


Figure 3.1: Distance distribution between examples of different individuals (a) and from the same individuals (b). Possible thresholds are represented in red. FAR' is equal to the aera to the left of the threshold (a). The FRR is equal to the aera to the right of the threshold (b).

In general,

$$FRR(x,t) \triangleq P(\neg\exists e_i : match(e,e_i) \cap \neg\exists e' : match(e,e')) \tag{3.3}$$

Indeed, for a false rejection to occur, no example with the same label should be matched. Furthermore, no other label should be predicted as well, or it would be a false classification, not a false rejection. Equation 3.3 can be simplified using the small false acceptance rate assumption as well as assumption 4, verified in section 3.5.

Assumption 4 - No strong bias for false rejection in case of false acceptance:
$P(\neg\exists e_i : match(e,e_i)|\exists e' : match(e,e')) \ggg P(\neg\exists e_i : match(e,e_i)))$.

Assumption 4 states that the probability to be falsely rejected is not increased a lot by knowing that the query has been falsely matched. Combined with the small false acceptance rate assumption, this implies that, in most cases, a rejected individual will not be falsely accepted. Then,

$$FRR(x,t) = P(\neg\exists e_i : match(e,e_i)) -$$

$$P(\neg\exists e_i : match(e,e_i)|\exists e' : match(e,e')) * FAR \simeq P(\neg\exists e_i : match(e,e_i))$$

The FRR is then independent of the population size. Finally, using the one example per individual assumption,

$$FRR(t) = P(\neg match(e,e_0)) \tag{3.4}$$

From equation 3.4, the FRR is equal to the proportion of examples pairs that are to the right of the threshold on figure 3.1(b). Indeed, with a distance greater than the threshold between them, the test example would be rejected.

$$FRR(t) \approx \int_t^\infty f_Y(y)dy$$

$FRR(t)$ decreases when the threshold increases, on the contrary of $FAR(x,t)$ which increases with the threshold. This means that the FAR and FRR cannot be set to a given value

independently. Decreasing the FAR will increase the FRR and vice-versa. A compromise between these two quantities has to be made.

Equations 3.2 and 3.4 are crucial because it will make the measurement of the error rates much easier. Concerning the FAR, only FAR' has to be measured once. Then the FAR for all population sizes can be deduced. As explains section 3.2, measuring the FAR once is already computationally intensive. Measuring it for all population sizes would have been impossible. The FRR can also be measured once, as it has been shown to be independent of the population size.

Concerning the FCR, its precise value will depend a lot on the choice of implementation of the *choose* function. Indeed, in the vast majority of cases where the query individual i is known, the example corresponding to i in the population will be within the threshold (supposing assumption 5 below). So, to compute the FCR, the behaviour of the algorithm when two different examples match the test example should be defined. Is the first found returned ? Or rather the closest to p ? For both cases,

$$FCR \triangleq P(\neg match(e, e_0) \cap \exists e' : match(e, e')) +$$

$$P(match(e, e_0) \cap \exists e' : match(e, e')) * P(choose\_wrong) \leq FAR \tag{3.5}$$

Assumption 5 - Small false rejection rate: $FRR(t) \ll 1$.

Equation 3.5 can be simplified using the small false rejection rate assumption. Indeed, the first term of equation 3.5 can be neglected (also using assumption 4):

$$\begin{gathered} FCR \approx P(match(e, e_0) | \exists e' : match(e, e')) * P(\exists e' : match(e, e')) * P(choose\_wrong) = \\ (1 - P(\neg match(e, e_0) | \exists e' : match(e, e'))) * FAR * P(choose\_wrong) \\ \approx FAR * P(choose\_wrong) \leq FAR \end{gathered}$$

If the label is chosen among the found examples at random, then $FCR \approx \frac{FAR}{2}$. Anyway, the FCR is bounded by the FAR. For this reason, the FCR will not be considered explicitly. Note that no assumption is required for the FCR to be bounded by the FAR. Indeed, for a false classification to occur, a false match has to happen as well. Only the relation $FCR \approx FAR * P(choose\_wrong)$ requires assumption 4 and 5. The FCR will be talked about in more details in section 3.5.

On figure 3.2 can be seen the evolution of the FRR and the FAR depending on the threshold, for a data set of 4000, 30k and 1M examples.

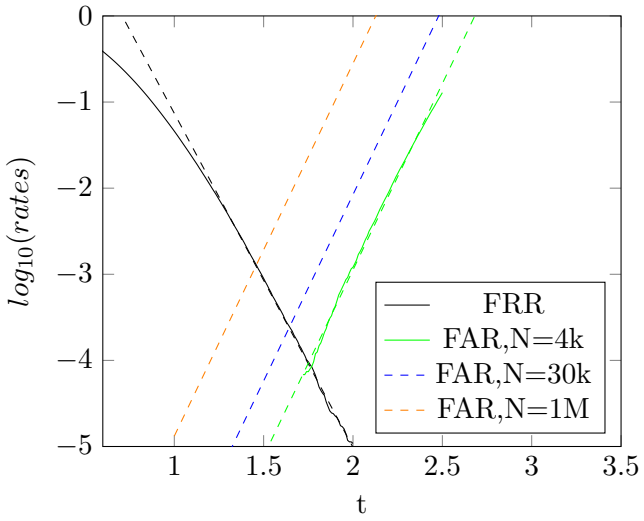


Figure 3.2: False rejection rate and false acceptance rate, in function of the threshold. The dashed lines are the exponential approximations of the error rates. The FAR curves were obtained by measuring FAR' and multiplying it by the population size.

The FRR is independent of the size of the database and decreases with the threshold. The FAR, on the other hand, increases with the threshold and the size of the database. This is a logarithmic graph, so the increasing and decreasing are exponential. As a consequence, the optimal point will be close to their intersection. Increasing the population size will increase both the optimal FAR and FRR, but decrease the optimal threshold.

Exponential rates approximation: $FRR(t) \sim c_{frr}e^{-a_{frr}t}$ and $FAR(x,t) \sim c_{far}x * e^{a_{far}t}$

$FRR(t)$ and $FAR(x,t)$ will be approximated by exponentials (dashed lines on the graph) with $c_{frr}, a_{frr}, c_{far}, a_{far} > 0$.

As can be seen, the FRR and FAR remain low even for a database of 1M examples with this noise intensity. Theoretically, those excellent rates could be achieved. In practice, the time barrier will be a strong obstacle.

3.2 Measuring the FAR

Measuring the FRR for a large variety of thresholds is easy. To find a false rejection, only one couple of examples that are farther away than the threshold has to be found. With only one distance computation, either a false rejection, or a correct classification is obtained. For the FAR it is completely different. Given one example and a population, in order to determine whether or not a false acceptance occurs, the distances between the test example and every stored example (supposing no filtering) have to be computed.

For this reason, it is impossible to measure the FAR accurately for small thresholds. There is no other way than using the exponential approximation and extrapolating the FAR for such small thresholds, even if a lot of data is available. In our case, it is more the small size of the test set that prevents us from measuring the FAR more accurately.[3]

[3]The experiment still took 4 hours on my hardware, using the Cpp package to boost R performance.

3.3 The Euclidian representation

This section develops a visual representation of the error rates, using figure 3.3.

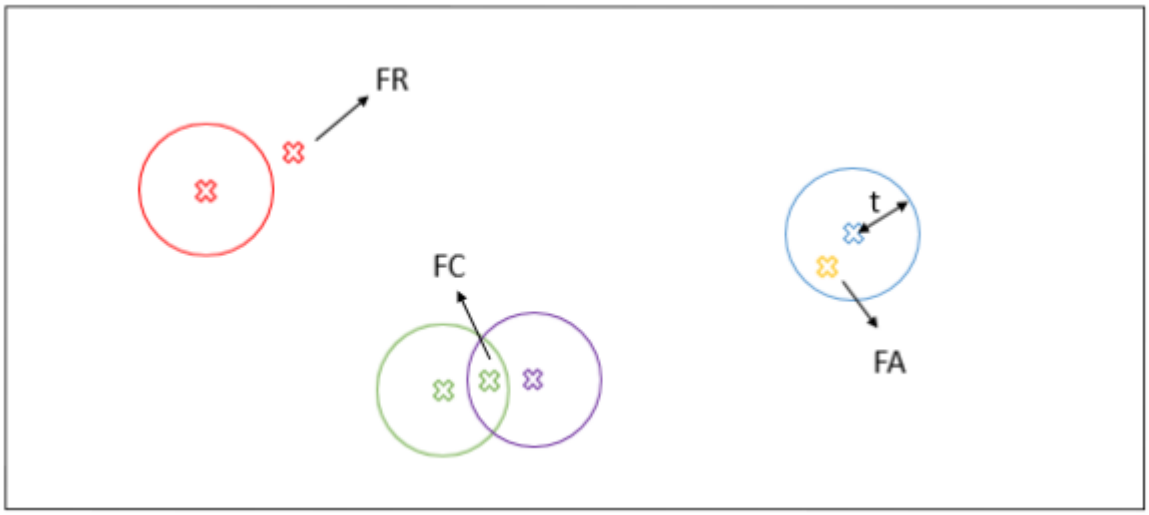


Figure 3.3: 2-D Euclidian representation of the feature space. The circles of radius t corresponds to the space claimed by each known individual. Different individual are represented with different colors.

Each different individual is represented with a different color. Individuals already registered have claimed part of the space. This part is a circle (one example per individual assumption), centered on their stored example with a radius equal to the threshold. A false rejection occurs when a registered individual generates a new example that falls outside of his circle and any other circle. If the threshold is increased, and so the radius of the circles, the false rejection rate will decrease. A false acceptance occurs when an example of an unknown individual falls into the circle of another one. Increasing the radius of the circle would, this time, increase the false acceptance rate. When a registered individual example falls into the circle of another one, a false classification may occur if the label is chosen wrongly.

Increasing the database size will increase the number of different circles. The false acceptance rate is proportional to the space claimed by other individuals. By the small false acceptance rate assumption, only few circles will intersect. The total claimed space will then increase linearly with the population size which implies that the false acceptance rate will do the same.

3.4 Solving the optimization problem

The optimal threshold, t^*, was defined in section 1.6 as

$$t^*(x) = argmin_t(F(x,t))$$

It can be shown that maximizing the F-score or minimizing the weighted sum of the error rates $\alpha * FRR(t) + (1-\alpha) * FAR(x,t)$ yields the same results, for $\alpha = \frac{\beta^2}{\beta^2+1}$, as long as the small error rates assumptions are valid. Indeed,

$$F = (\beta^2+1)\frac{(1-FRR)(1-FAR)}{\beta^2(1-FAR)+(1-FRR)} \approx \frac{\beta^2 - \beta^2 FAR + 1 - FRR - \beta^2 FRR - FAR}{\beta^2 - \beta^2 FAR + 1 - FRR}$$

$$= 1 - \frac{\beta^2 FRR + FAR}{\beta^2 - \beta^2 FAR - FRR + 1} \approx 1 - \frac{\beta^2 FRR + FAR}{\beta^2 + 1} = 1 - (\alpha FRR + (1-\alpha)FAR) \quad (3.6)$$

Then, at the optimum, the following equation has to be verified[4]

$$\frac{\partial F}{\partial t} = 0 \Leftrightarrow \frac{\partial FAR}{\partial t} = -\beta^2 \frac{\partial FRR}{\partial t} \quad (3.7)$$

Using $FRR(t) \sim c_{frr} e^{-a_{frr} t}$ and $FAR(t) \sim c_{far} x * e^{a_{far} t}$, equation 3.7 gives

$$t^*(x) = argmin_t(F(x,t))) = \frac{1}{a_{far} + a_{frr}} ln[\frac{\beta^2 c_{frr} a_{frr}}{c_{far} x a_{far}}] \quad (3.8)$$

The optimal FRR and FAR can then be derived.

$$FRR^*(x) = FRR(t^*(x)) = c_{frr} exp(-a_{frr} t^*) = c_{frr} exp(\frac{-a_{frr}}{a_{far} + a_{frr}} * ln(\frac{\beta^2 c_{frr} a_{frr}}{c_{far} a_{far} x})) =$$

$$c_{frr} (\frac{\beta^2 c_{frr} a_{frr}}{c_{far} a_{far} x})^{\frac{-a_{frr}}{a_{far}+a_{frr}}} = A_{frr} x^{\gamma} \quad (3.9)$$

with $A_{frr} = c_{frr} (\frac{\beta^2 c_{frr} a_{frr}}{c_{far} a_{far}})^{\frac{-a_{frr}}{a_{far}+a_{frr}}}$ and $\gamma = \frac{a_{frr}}{a_{far}+a_{frr}} < 1$

In the same way,

$$FAR^*(x) = FAR(x, t^*(x)) = c_{far} (\frac{\beta^2 c_{frr} a_{frr}}{c_{far} a_{far}})^{\frac{a_{far}}{a_{far}+a_{frr}}} x * x^{\frac{-a_{far}}{a_{far}+a_{frr}}} = A_{far} x^{\gamma} \quad (3.10)$$

with $A_{far} = c_{far} (\frac{\beta^2 c_{frr} a_{frr}}{c_{far} a_{far}})^{\frac{a_{far}}{a_{far}+a_{frr}}}$ and $\gamma = \frac{a_{frr}}{a_{far}+a_{frr}} < 1$

The F-score is then

$$F^*(x) = F(x, t^*(x)) = 1 - x^{\gamma}(\alpha A_{frr} + (1-\alpha) A_{far}) \quad (3.11)$$

All error rates increase sub-linearly with the database size. From the definition of γ, it can be deduced that the evolution of $F^*(x)$ with x depends a lot on the relative value of a_{frr} and a_{far}. If $a_{frr} \gg a_{far}$, then $\gamma \approx 1$, which means $1 - F^*(x)$ will increase linearly with x. On the contrary, if $a_{frr} \ll a_{far}$, then $\gamma \approx 0$, such that $F^*(x)$ will not depend on x at all.

This can be justified intuitively. First, consider the case where $a_{frr} \gg a_{far}$. For a given population size x, the optimal threshold is given by $t^*(x)$. Now, the size of the population is doubled $x \leftarrow 2x$. The optimal threshold is then decreased but only by a very small amount. Indeed, as a_{frr} is big, reducing too much the threshold will make the FRR skyrocket. The modification of the threshold will barely affect $FAR'(t)$, as a_{far} is small. As a consequence, $FAR^*(x)$ will well be doubled. If $a_{frr} \ll a_{far}$, the same transformation $x \leftarrow 2x$ will cause a bigger decrease of $t^*(x)$. This will not harm the FRR too much, as a_{frr} is small. However, this will greatly decrease $FAR'(t)$ and will counter-balance the increase of the population size.

[4]The condition is necessary and sufficient.

For our particular data, $c_{frr} = 561.156$, $a_{frr} = 8.933$, $c_{far} = 6.2 * 10^{-16}$ and $a_{far} = 9.965$. As a consequence, $\gamma = 0.473 \approx 0.5$, so the error rates behave roughly like $\sqrt{x}$. Such a slow increase means that populations of great sizes can be tackled. For example, the population size (for $\beta = 1$) corresponding to $F^*(x) = 0.99$ is $x = (\frac{1-0.99}{0.5*(A_{frr}+A_{far})})^{\frac{1}{\gamma}} = 80.55M$. This size will even be significantly increased in chapter 5, where better metrics will be learned.

As can be seen on figure 3.4(a) ($\beta = 1$), the F-score depends massively on the threshold. A threshold too high or too small will result in a low F-score. The optimal F-score also decreases when the population size increases. Figure 3.4(b) highlights the fact that a higher β will lead to a higher optimal threshold. Also, the optimal performance is better when β gets away from one.

Figure 3.4(c) shows well that the optimal threshold decreases with the database size. The error rates increase both with it (figure 3.4(d)). The FRR always increase when the threshold decreases so it is not surprising. For the FAR, the augmentation of the size of the population overcomes the diminution of the threshold. As a consequence, it also increases. A small β yields a bigger FRR and smaller FAR while a bigger β does the opposite.

Figure 3.4(d) shows that the optimal F-score decreases with the database size ($1 - F^*$ is plotted). Also, the optimal F-score is minimal for $\beta \approx 1$. Obviously, it is easier to increase the F-score if one error rate has less importance. For $\beta = \infty$, for example, the FAR is neglected, so the F-score could be set to 1 with a threshold of $+\infty$. Focusing on a small FAR yields slightly best results as focusing on a small FRR. This is due to the fact that the FAR decreases slightly faster with the threshold.

Trying to choose a right β for a given application may be difficult at first sight. However, it can be shown that

$$\frac{FRR^*}{FAR^*} = \frac{A_{frr}}{A_{far}} = \frac{a_{far}}{\beta^2 a_{frr}} \tag{3.12}$$

Fixing $\frac{FRR^*}{FAR^*}$ is a lot more intuitive than fixing an abstract parameter β. Figure 3.4(f) depicts the relationship between β and $\frac{FRR^*}{FAR^*}$

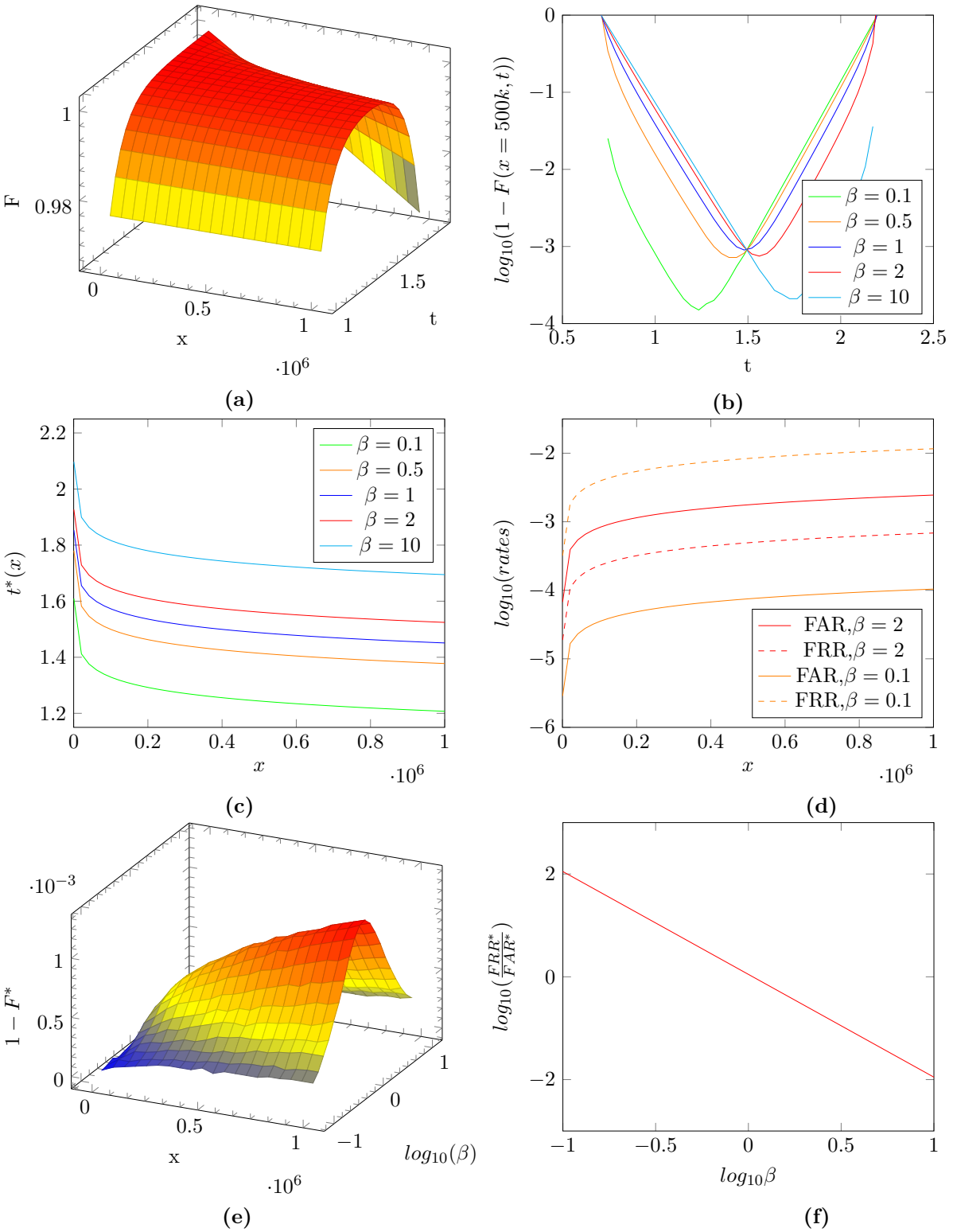


Figure 3.4: Evolution of the F-score with the x of the population and the threshold, for $\beta = 1$ (a). Evolution of the F-score for a population of 500k individuals in function of the threshold, for different β values (b). Dependence of the optimal threshold with the x of the population (c). Evolution of the optimal error rates with the population x (d). 3-D representation of the optimal F-score $(1 - F^*)$ depending on the population x and β (e). Comparison of the two optimal error rates in function of β (f). All graphs are obtained using the exponential approximation.

3.5 Limits

The results presented in this chapter were obtained using the exponential approximation. As can be seen on figure 3.2, it is well valid for the range of thresholds that are relevant (figure 3.4(c)). However, for very large population sizes and especially for a small β, the threshold might decrease enough to go out from the exponential zone of the FRR. If this happens, the FRR will increase less than predicted by the exponential approximation. The predicted F-score will then be lower than the true one. Nonetheless, this will never be the case for $\beta \geq 1$. Indeed, for the limit size of 80.55M and $\beta = 1$, the optimal threshold still remains sufficiently high (1.219).

Concerning the FAR, as explained in section 3.2, its behaviour has to be extrapolated for small thresholds. The failure of finding false acceptance for lower thresholds is a strong clue that the FAR still continues to decrease fast for such thresholds, but there is no real proof of that. The error rates might not always behave exponentially for all possible data sets. This does not make the analysis irrelevant, as all qualitative results are still valid.

As already mentioned, to measure $FAR(x,t)$, $FAR'(t)$ was measured once. Then $FAR(x,t)$ was found for all x using equation 3.2. This is valid only if assumption 3 (small false acceptance rate) is met. The comparison between the true $FAR(x,t)$ and the predicted one using equation 3.2 is made on figure 3.5(a), for a population size of 25.000. For large thresholds, there is a significant difference between them. Indeed, some individuals match with a lot of other individuals. For the true FAR, these multiple matches are counted as only one false acceptance. This explains why the true FAR is lower than the one predicted for large thresholds. However, when the threshold decreases, the difference between the two curves narrows before vanishing completely. As a consequence, equation 3.2 is well correct for thresholds of interests.

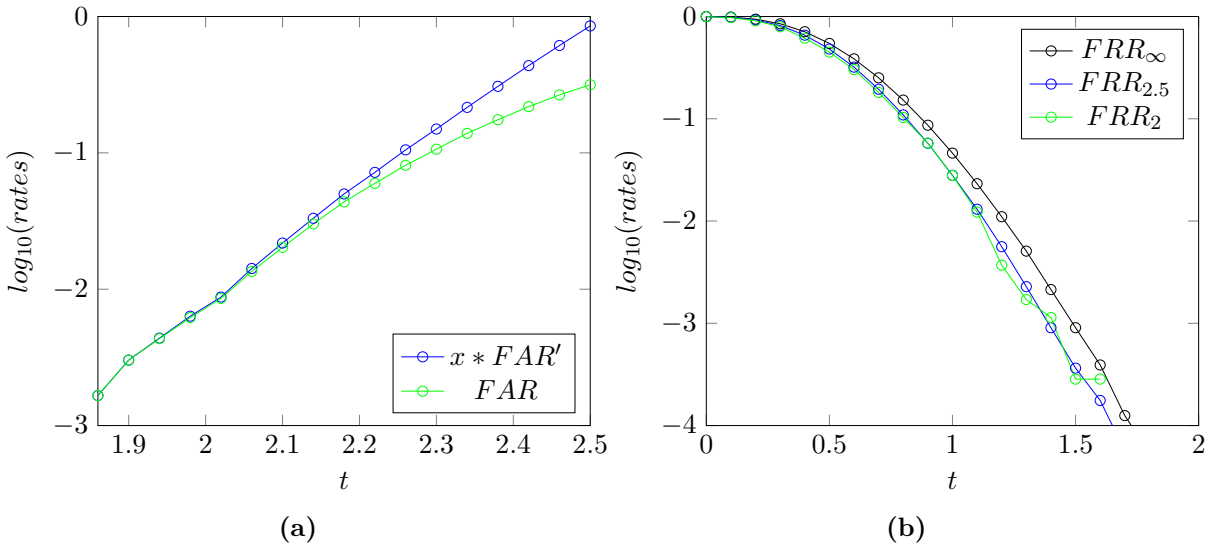


Figure 3.5: Comparison between the true FAR and the FAR predicted by the relation $FAR(x,t) = x * FAR'(t)$ (a). Probabilities of rejection of an example that has been falsely matched with a threshold x, noted FRR_x (b).

Also, the false classification rate was not taken into account. The FCR was shown to be bounded by the FAR. Using assumption 4, its effect on the FRR are also negligible, as the

majority of rejected examples will not be falsely accepted by other individuals. As a consequence, the FRR has been measured using equation 3.4. Figure 3.5(b) justifies the validity of assumption 4. Indeed, as can be seen, the probability of being rejected knowing the query has been accepted is even lower than the standard FRR (FRR_x denotes the probability of rejection of a query than has been falsely accepted with a threshold x).

Supposing that the example with the closest label is predicted (most obvious choice of implementation of the *choose* function), the FCR will be tremendously small compared to the FRR and the FAR. The main difficulty of person re-identification is that unknown individuals must not be recognized. If only known individuals would have to be recognized correctly, the error rates (only the FCR in this case) would be considerably smaller. The probability of the query example to be closer to a foreign example than the example of the same individual can be estimated using $FAR(x,t)$ and $FRR(t)$. In the following, the complete independence between $P(match(e,e_0))$ and $P(\exists e' : match(e,e'))$ will be assumed. From figure 3.5(b), it is not really the case, but the computations will give an idea of the order of magnitude of the FCR (which is too small to be measured with only 25.000 individuals). The results will be bigger than the true FCR, as $P(d(e,e_0) > t | \exists e' : d(e,e') \leq t) < P(d(e,e_0) > t)$ (figure 3.5(b)).

$$FCR^*(x) = P(\exists e' : d(e,e_0) > d(e,e') \wedge d(e,e') \leq t^*(x)) =$$

$$\int_0^{t^*(x)} P(d(e,e_o) > t') * P(\exists e' : d(e,e') \leq t')dt' = x * \int_0^{t^*(x)} FRR(t') * FAR'(t')dt'$$

$FCR^*(x)$ can be seen on figure 3.6 for sizes up to 3M. The increase is quasi-linear but the FCR remains 3 order of magnitudes smaller than the FRR and the FAR for a population of 3M individuals. The true FCR will be even lower than that, as an upper bound has been computed. Nonetheless, as the FCR increases faster than the FRR and the FAR, there exists a population size from which it will no longer be negligible (when the small error rates assumption will cease to be valid).

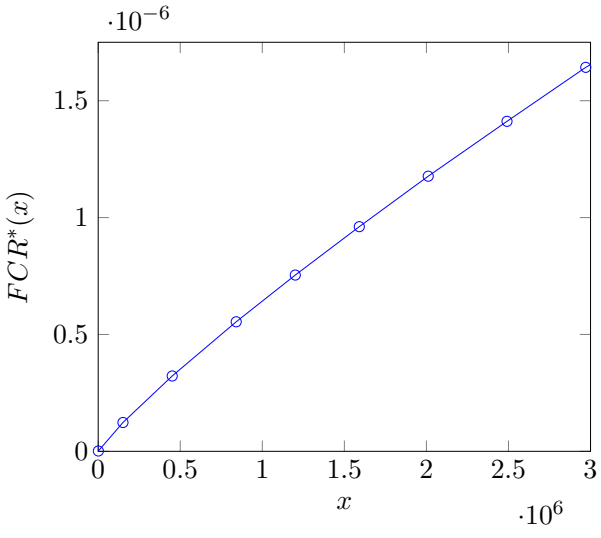


Figure 3.6: Evolution of the FCR with the population size.

Chapter 4

Constraints

In this chapter, a time and space constraint will be posed and their effects on the error rates studied. As already mentioned, nearest neighbour algorithms have no training time, so the time constraint will apply to the decision time.

Throughout the remaining of this thesis, random prototype selection will be used. The reason is that, as will be seen in section 4.2, a lot of prototypes are required and it is costly to find them. Moreover, the prototypes have to be selected from a far larger set, otherwise it would come back to random selection. Not enough data is available, unfortunately.

It was observed in section 2.6 that the number of computed distances has an asymptotic logarithmic behaviour, for a fixed threshold. However, as was seen in section 3.4, the optimal threshold decreases with the database size. As a consequence, the number of computed distances will increase even slower, as can be seen in figure 4.1, with x the population size.

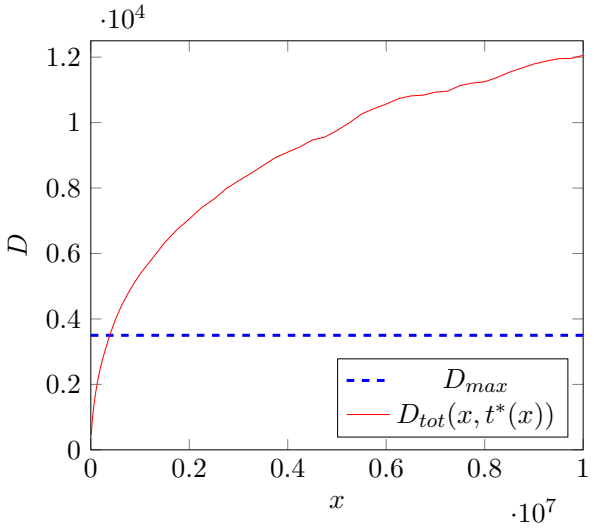


Figure 4.1: Total expected number of distance computations, in function of the population size. The dashed line corresponds to a time constraint with $\eta_d = 3500$.

4.1 Time constraint

As was seen in section 2.2, the smaller the threshold is, the less examples will remain unfiltered. The time constraint then gives an upper bound on the threshold value. The optimum threshold that satisfies this constraint is then given by

$$t^*(x) = min(t : T(x,t) == \eta, argmin_t(F(x,t))) \tag{4.1}$$

As was done in section 2.5, the time taken by the search will be assumed to be only function of the number of distance computations. As $T(x,t) \propto D_{tot}(x,t)$, equation 4.1 can be turned into

$$t^*(x) = min(t : D_{tot}(x,t) == \eta_d, argmin_t(F(x,t))) \tag{4.2}$$

A constraint on the expected number of computed distances is posed. Assuming a time constraint of 3.5 seconds, $\eta_d = 3500$, as the time to compute one distance is 1 msec[1]. This upper bound is represented in dashed on figure 4.1. The maximal threshold $t_{max,\eta_d=3500}(x)$ such that $D_{tot} = 3500$ has to be found.

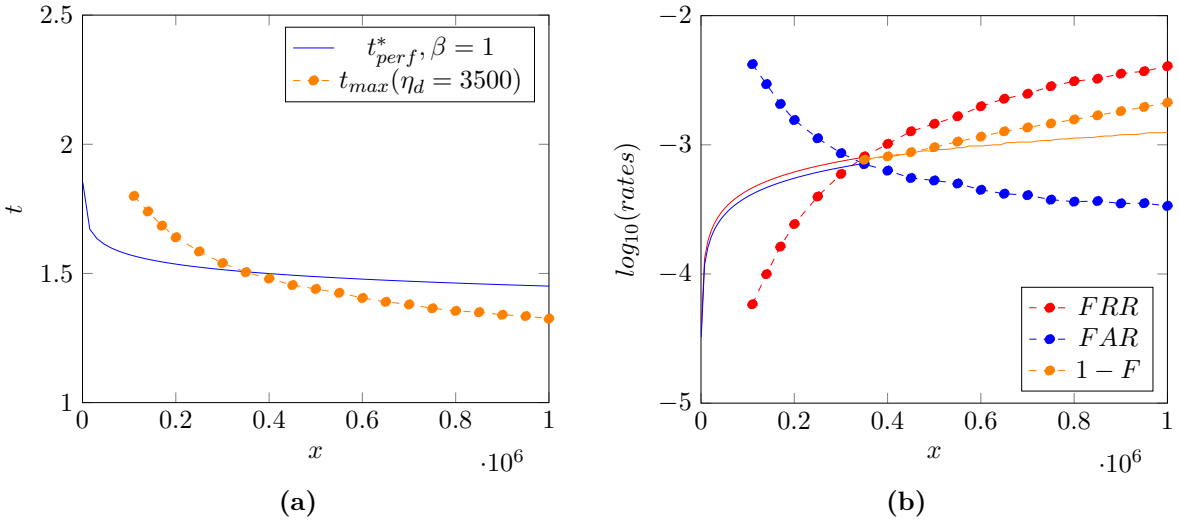


Figure 4.2: Comparison between the optimal threshold (for $\beta = 1$) with the maximal threshold of a time constraint with $\eta_d = 3500$ (a). Resulting error rates when the time constraint $\eta_d = 3500$ is taken into account (b). The plain lines, on the contrary of the dashed lines, indicates that the time constraint has not become limiting yet.

On figure 4.2(a), a comparison between the maximal and optimal thresholds for $\beta = 1$ is made. The time constraint $\eta_d = 3500$ becomes limiting from $x \approx 350.000$. It was observed on figure 3.4 that the higher the β, the higher was the optimal threshold. A large β will then be more affected by the time constraint. On the contrary, a low β will reduce its effects. It is more computationally easy to focus on a low FAR than aiming at a low FRR.

On figure 4.2(b) is depicted the FRR and FAR that will be obtained, for $\beta = 1$, when taking the time constraint into account. The lines are plain when the time constraint has not become active yet. The error rates are then the same as the optimal ones on figure 3.4. As $\beta = 1$, the FRR and the FAR starts roughly equal. Both the FAR and FRR first grow when

[1] On my personal hardware

the size increases. When the time constraint becomes limiting (dashed lines), the threshold starts to decrease more abruptly. From that point, the FAR begins to decrease and the FRR still increases, but faster. As a consequence, the F-score will be dominated by the FRR. If $\alpha * FRR >> (1-\alpha) * FAR$, then from equation 3.6

$$1 - F = \alpha * FRR = \frac{\beta^2}{\beta^2 + 1} * FRR \tag{4.3}$$

It is essential to have a small FRR when a time constraint is involved. A low FRR will decrease the optimal threshold, pushing away the point where the time constraint becomes limiting. A low FRR will also reduce the F-score when the constraint has become active.

4.2 Space constraint

The effect of a space constraint will now be studied. There are mostly two things that uses memory, the set of stored examples and the table containing the distances between all examples and prototypes. A space constraint can then be formulated

$$Space_{tot} = |P| * size_w(e) + |P| * n_p = |P| * (size_w(e) + n_p) < S_w \tag{4.4}$$

where e denotes a stored example and the index w indicates sizes in words (32 bits).

As there is only one stored example per person, the only thing that can be tuned is the number of prototypes. The space constraint then provides an upper bound on n_p:

$$n_p < \frac{S_w}{|P|} - size_w(e) \tag{4.5}$$

A space constraint has only negative effects when there is also a time constraint. Indeed, the number of prototypes used does not influence the optimal error rates, only the speed of the algorithm.

From equation 4.5, the maximum number of prototypes decreases with the number of individuals. Unfortunately, the optimal number of prototypes increases with this number, as was observed in section 2.5. However, the optimal threshold decreases, so it has to be taken that into account. On figure 4.3(a) is depicted the evolution of n_p^* for different threshold values. The optimal number of prototypes decreases with the threshold. So when the size of the population is increased, two counter-acting effects appear. On the one hand, the optimal threshold will decrease, which in turn will decrease the optimal number of prototypes. On the other hand, as the size increases, the latter will increase as well.

As can be observed on figure 4.3(b), the optimal number of prototypes increases with the population size despite the reduction of the threshold. The dashed lines corresponds to a space constraint of 20 Go.

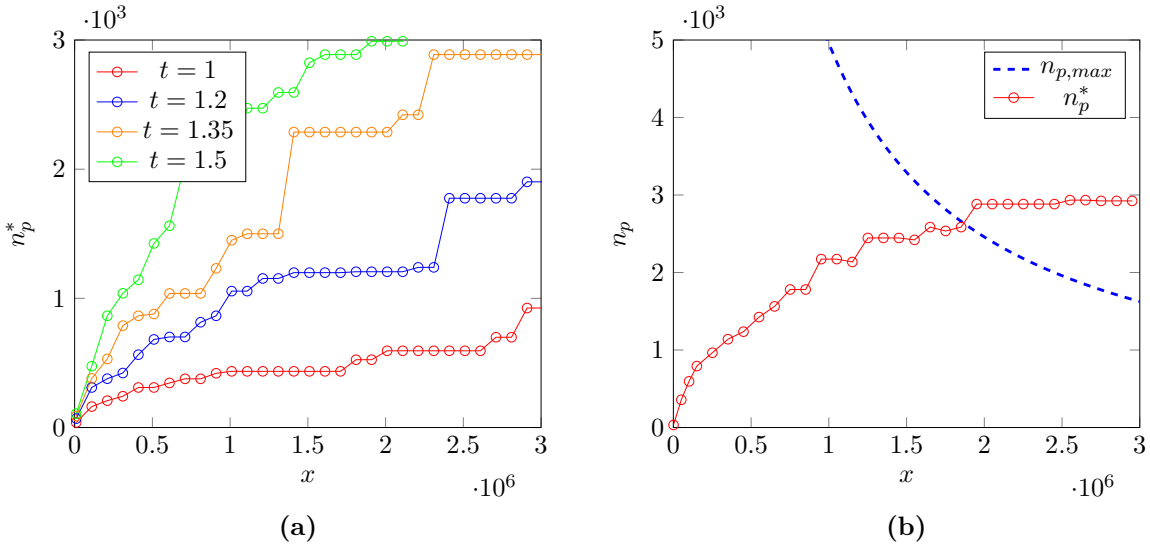


Figure 4.3: Evolution of the optimal number of prototypes with the population size, for different thresholds (a). Comparison between the maximum number of prototypes allowed by a space constraint of 20 Go and the optimal number of prototypes (at the optimal threshold) (b).

Once the space constraint becomes limiting, the search algorithm will have to use less prototypes than the optimal number. As a consequence, to satisfy the expected time constraint, the threshold will have to be reduced even more, leading to a harsher degradation of the F-score, and in particular, the FRR. Figure 4.1 was obtained without bound on the number of prototypes. For a population of 10M individuals, the optimal number of prototypes is approximately 4.000. The size of the table containing the distances between the prototypes and the stored examples would then be $4 * 10^6 * 4000 = 160Go$, which might be too much. Figure 4.4 compares the total number of distance computations for different space constraints.

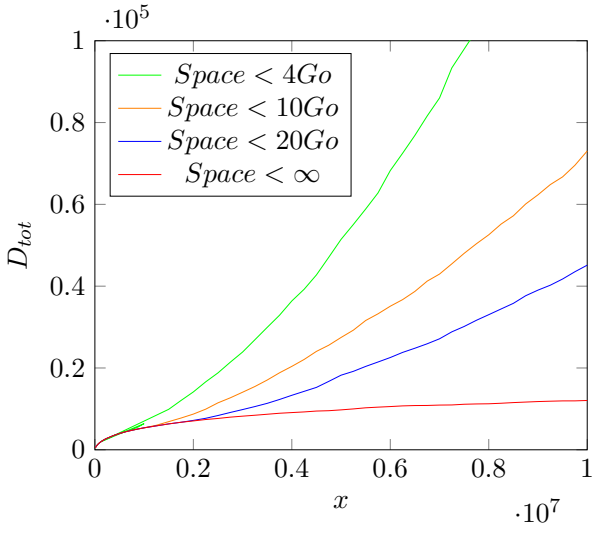


Figure 4.4: Total expected number of distance computations, in function of the population size, for different space constraints.

Clearly, the number of distance computations skyrockets when the space constraint becomes active. Time constraints that were irrelevant with unlimited space can become nearly impossible

to meet when the space used is tightly constrained.

4.3 Limits

The time constraint discussed in section 4.2 is an expected waiting time constraint. If the constraint is respected, on average, the number of distance computations will be less than η_d. However, this does not fix an upper bound on this number. There is absolutely no guarantee on the maximal number of distance computations that could happen in the worst case. In a real life system, such a constraint may be insufficient. A better constraint would be of the form: $y\%$ of the queries must be answered with less than η_d computations. For the expected time constraint, there is a single optimal number of prototypes for each threshold and population size. This optimal number will minimize the total expected number of distance computations. For the new formulation, it is not as simple. Indeed, increasing the number of prototypes beyond the optimal one will reduce the variation among the number of distance computations. As a consequence, it is more complicated to find the maximal threshold satisfying the constraint.

Also, the assumption that the time is measured by the number of distance computations will not always be true. This hypothesis is true when distance are computationally heavy, which is the case where a time constraint is the most relevant. However, the number of distance computations increases less with the population size than the number of explored nodes, given by equation[2] 2.12. As a consequence, there will always be a population size from which the hypothesis will no longer be valid. In this case, the increase of the decision time will be sharper, as it will follow equation 2.12. Then, the maximal threshold allowed by the time constraint will decrease even more, making the degradation of the F-score more abrupt.

[2]Assuming vantage trees are used.

Chapter 5

Metric Learning for large scale person re-identification

All results of the previous chapters were obtained with the default metric, developed in section 1.5.2. In this chapter, a more advanced metric learning scheme is proposed. Two important aspects have to be taken into account while learning the metric. First, the optimal identification performance, that was the subject of chapter 3, has to be maximized. Second, the pruning effectiveness, discussed in chapter 2, has to be maximized as well, such that the effect of the time constraint on the F-score, discussed on chapter 4, remains as low as possible. In this chapter, a Mahalanobis metric learning scheme balancing between identification performance and computational efficiency will be proposed. The compromise will be done by casting a convex optimization problem, that can be solved using a simple gradient descent algorithm. First, previous work on metric learning for nearest neighbours and person re-identification will be talked about. Second, the objectives that have to be balanced will be formalized. Then, the new metric learning scheme will be built iteratively, starting from a well known method. Finally, the choice of the right learning meta-parameters, given a particular time and space constraint, will be discussed.

5.1 Previous work

Most metric learning techniques aim at satisfying different types of constraint on the training set[14]:

- Must link constraint: $S = \{(x_i, x_j) : x_i \text{ and } x_j \text{ should be similar}\}$.
- Cannot link constraint: $D = \{(x_i, x_j) : x_i \text{ and } x_j \text{ should be dissimilar}\}$.
- Relative constraint: $R = \{(x_i, x_j, x_k)\} : x_i \text{ should be more similar to } x_j \text{ than to } x_k\}$.

They then define some loss function on these constraints that has to be minimized. Generally, S contains points of the same label and D points of different labels. R contains triplets of points such that only the two first points share the same label. There is a fundamental difference between must/cannot link constraints and relative constraint.

Must link constraints simply imposes to points having the same label to be close to each other while Cannot link constraint enforces points with different labels to be far. Some metric learning approaches use only these kinds of constraints[1, 26]. MMC (Xing *et al.*, 2003)[1] maximizes the sum of the distances of points in D while constraining the sum of squared of

the distances between points in S. ITML (Davis *et al.*, 2007)[26] poses soft constraints for each pair of points in S or in D. Points in S should be closer than a given distance, while points in D should be farther than another specific distance. The violations of these constraints is then minimized.

Relative constraints (notably used by [7, 27]) enforce points having the same label to a given point x to be closer to x than points with different labels. This type of constraint can be used to explicitly tackle the k-nearest neighbour problem. Indeed, they can ensure that, for each training point, impostors are farther away than points sharing the same label. A nearest neighbour algorithm will then predict the correct label for these training points. For instance, LMNN (Weinberg *et al.*, 2006) defines relative constraints such that the k-closest points of each training points share all the same label[7]. NCA (Goldberge *et al.*, 2005) is another metric learning method that is driven by the nearest neighbour prediction rule[8]. Indeed, NCA tries to maximize the leave-one-out accuracy of a stochastic nearest neighbour classifier. In a related manner, MCML (Globerson and Rowe, 2005) tries to collapse each class into a single point, while pushing the other classes as far as possible[28]. From the definition of the loss function of these three methods, the cost of having two points with different labels close to each other will depend on the presence or the absence of points with the same label in the neighborhood of these points. That is the main reason why nearest neighbour driven approaches, and more generally relative constraints, are ill-posed for person re-identification.

Indeed, for person re-identification, a **threshold** NNS algorithm is used. A threshold NNS differs from a standard NNS because an individual can be rejected. A threshold common to the whole database is maintained. Standard nearest neighbour performance highly depends on the presence of impostors in the neighborhood of the test points, which is not the case for a threshold NNS [1]. That observation will make any NN-driven metric learning sub-optimal for person re-identification. For example, consider points that have no near foreign points in their neighborhood. The utility to bring their similar points closer to them is limited. As a consequence, the probability of rejection of these points will be bigger because the threshold is learned globally. To summarize, similar points should be close to each other, independently of the presence of impostors in their neighborhood, which can be done with must and cannot link constraints. Besides, as the FCR has been shown to be negligible in section 3.5, the training set will probably contain only few, if any, impostors. Posing relative constraints is then irrelevant.

As a consequence, no relative constraint will be posed and only the two sets S and D will be used. S will contain couple of examples of the same individuals and D will be composed of examples of different individuals.

Work has also been done in the specific context of person identification[4, 5, 6]. Usually, they try to match pictures of the same person together. While the feature space is substantially different, the principle of the decision making remains the same. They learn a distance function and a threshold and decides that two pictures matches if their distance is smaller than the threshold. However, they do not provide a way to learn efficient metrics and generally learn or pose the threshold while learning the metric. As mentioned in the introduction, they only focus on discriminating positive and negative pairs of examples. That is the reason why they can pose a fixed threshold. In our case, an individual has to be recognized among a large population. As discussed in section 3.1, this requires a modifiable threshold to balance between the error rates when the size of the population varies. As a consequence, learning the threshold is useless for us.

[1] The FCR has been shown to be negligible in section 3.5.

Mignon and Juri (2012) fix the threshold to 1 and minimize the hinge loss[6]

$$\sum_{(p_i,p_j)\in S\cup D} max(0, y_{ij}(D_A^2(p_i,p_j) - 1))$$

with $y_{ij} = 1$ if p_i and p_j are from the same individual and $y_{ij} = -1$ if they are from different individuals. Our proposed framework will be showed to be equivalent to minimizing this hinge loss, for some parameter value. Xiong *et al.* learn a global metric but a local decision rule[4]. The local decision rule is required in standard person re-identification as the threshold should depend on the difference of posture, angle etc of the person. In our case, the threshold cannot vary as the triangular inequality has to be verified in order to prune the search space. Typically, they evaluate their algorithms by feeding them with positive and negative example pairs and record the number of misclassified pairs. This is completely different from our interest, as negative has to be predicted on every stored example with a different label.

Another approach would be record linkage[29]. This is a probabilistic framework that matches record pairs belonging to the same person together. However, it does not provide a way to prune examples and every single record has to be examined for matching. While it may give better performance, it is unpractical for large scale application, when possibly millions of records have to be matched.

5.2 Objectives

As already mentioned, the two sets S and D will be used to pose must and cannot link constraints. The following two opposing objectives have then to be balanced in order to learn good metrics:

- Reducing the number of misclassifications. This is obviously done by keeping points in S close together and points in D far from each other. More precisely, the distance distribution of points in S, noted f_S, and the one of points in D, noted f_D, have to be separated by a large margin[2].
- Increasing the pruning efficiency in order to meet the time constraint. This is achieved by flattening f_D, while maintaining f_S tight. Indeed, the filtering condition for a stored example e, a query q and a set of prototypes B is given by equation 2.1: $\exists b \in B : |d(q,b) - d(b,e)| > t$. A tight f_S will allow the use of smaller thresholds, and hence will increase the filtering. A flat f_D will increase $|d(q,b) - d(b,e)|$, which will also increase the pruning.[3]

Reducing the extent of f_S will benefit to both of the objectives. However, there is a direct opposition between a large separation and a flat f_D. As an illustration, consider this simple scenario: the distance between points of the same individuals is always 0 while the distance between examples of different individuals is always 1. Formally, $f_S(x) = \delta(x)$ and $f_D(x) = \delta(x-1)$ where δ is the Dirac function. This would obviously lead to perfect unconstrained classification, for all thresholds $0 < t < 1$. However, $\forall q, b, e \in D : |d(q,b) - d(b,e)| = |1-1| = 0 < t$. As a consequence, no pruning would be possible, which will make this scenario, at first sight easy, impossible to tackle for large populations. In the next section, the progression that led to the formulation of the proposed method will be explained.

[2] f_S corresponds to the blue curve of figure 1.1 while f_D is represented by the red curve.

[3] As points that are difficult to filter are closer to the query, f_D has to be flat especially for small distances, which is in direct opposition with a good separation.

5.3 Constrained global distance metric learning

To start with, the approach proposed in [1] was implemented. The following constrained optimization problem has to be solved.

$$M = argmax_A \sum_{(e,e')\in D} D_A(e, e')$$

such that

$$\sum_{(e_i,e_j)\in S} D_A^2(e_i, e_j) \leq 1, A \succeq 0 \tag{5.1}$$

Examples of different patients should be pushed as far as possible, while maintaining examples of the same patient close. We pose $N_s = |S|$ and $N_d = |D|$.

The projected gradient algorithm described in [1] was implemented. The problem is convex, so there is no local minimum that is not a global minimum as well. On figure 5.1 can be seen the comparison between the default metric and the metric learned by solving the optimization problem 5.1. The metric has been trained on a training set with $N_s = N_d = 25.000$. The test set will be the same for all metric learning experiments, with $N_s = N_d = 250.000$. The training set is obtained with the train individuals while the test set is built using test individuals, as discussed in section 1.3.3. The optimal M has been scaled to the same level as our default metric, based on the sum of the squared distances between points in S.

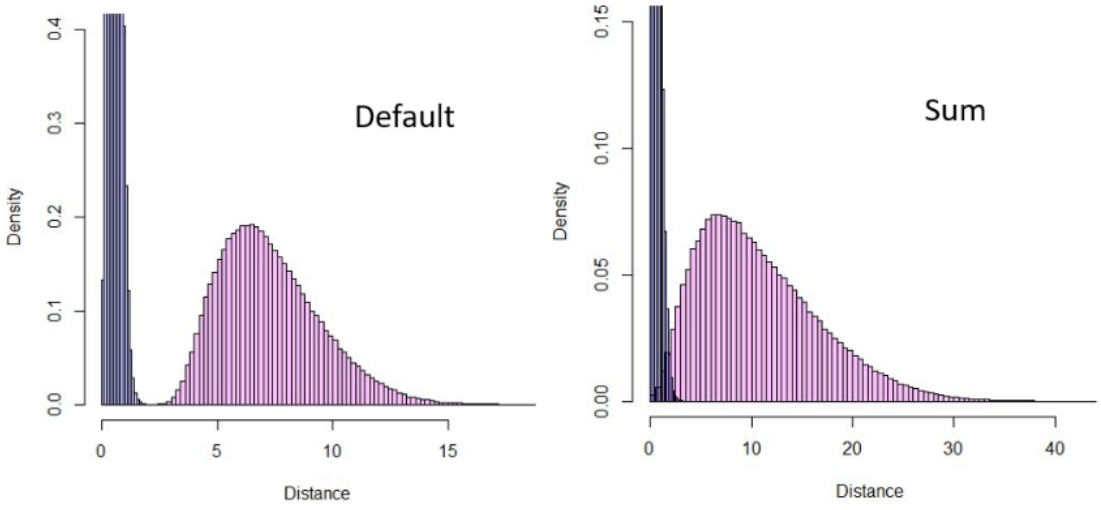


Figure 5.1: Comparison of the distance distributions of the default metric and the metric learned with the method developed in [1].

The results are catastrophic. As only sums of distances are looked at, the intersection of f_S and f_D is not prevented. The default metric will give far better identification performance. However, the width of f_D has largely increased, which satisfies the second objective. This leads to an interesting property:

Pushing away points in D with the same intensity, independently of the distance between them, extends the with of f_D.

Thanks to this property, one may want to define some objective function that allows to modulate the weight given to points in D. We then propose the following objective function:

$$M = argmin_A g(A) = \sum_{(e,e')\in D} e^{-k*D_A(e,e')} \tag{5.2}$$

with $k > 0$. As this function is convex, the optimization problem stays convex. This objective function gives more weight to pairs of examples that are close to each other. The meta-parameter k allows the modulation of the difference of importance of examples in D. For $k \to 0$,

$$g(A) \approx \sum_{(e,e')\in D} 1 - k * D_A(e,e') = N_d - k * \sum_{(e,e')\in D} D_A(e,e')$$

which is equivalent to solving problem 5.1. Indeed,

$$M = argmin_A g(A) = argmax_A \sum_{(e,e')\in D} D_A(e,e')$$

On the contrary, when $k \to \infty$, the dominant term in the sum will be the one with the lowest distance. As a consequence, the optimal M will be given by

$$M = argmax_A[min_{(e,e')\in D} D_A(e,e')]$$

Finally, as the constraint $\sum_{(e_i,e_j)\in S} D_A^2(e_i,e_j) \leq 1$ fixes the scale, the value of k should depend on the number of points in the training set. To prevent that, we pose $k = \sqrt{N_s} * k'$ such that k' can be fixed independently of the number of training examples.

On figure 5.2 is displayed the results for $k' = 0.1, 0.25, 0.5, 1, 2.5$ and 5 on the test set. As can be seen, the separation between f_S and f_D increases when k' grows. However, the extent of f_D is reduced which should affect the pruning capacity. This illustrates again the difficulty of balancing the two objectives.

One major problem of this approach is that f_S and so the FRR is not restricted enough. The sum of the squared distances between points in S of all metrics is equal to the one of the default metric. However, it can be spotted that the width of f_S is larger for the new metrics. Decreasing the width of f_S will decrease the error rate but also increase the filtering. It is thus essential to reduce the extent of f_S as much as possible. The FRR of these metrics is compared with the FRR of the default metric on figure 5.3. Clearly, controlling the sum of squares does not control the FRR enough.

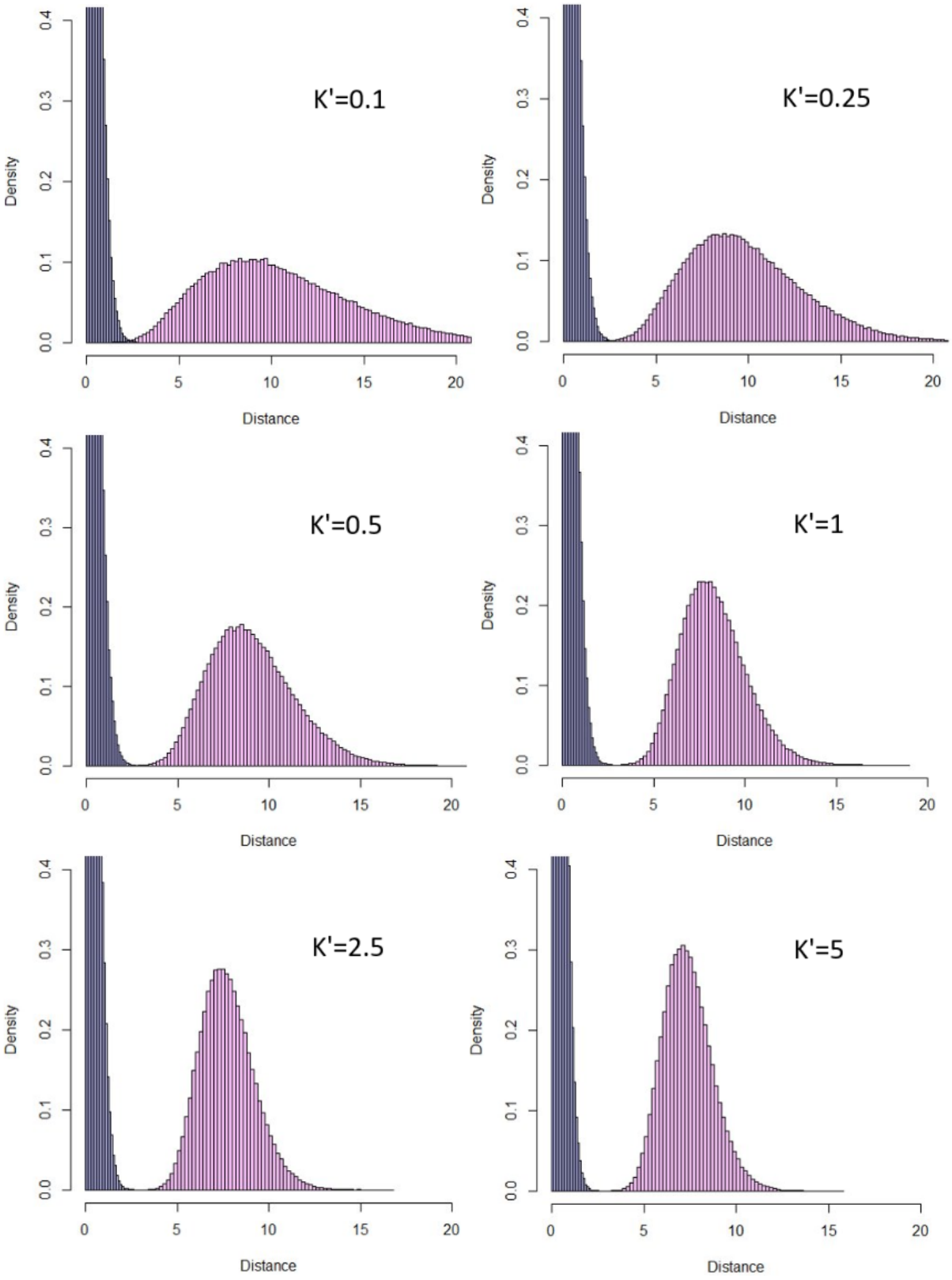


Figure 5.2: Distance distributions f_S and f_D, evaluated on the test set, using metrics learned by solving the optimization problem 5.2 for different k' values.

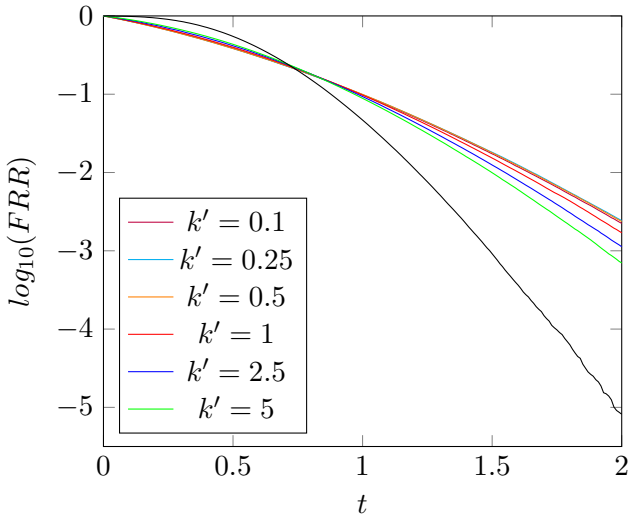


Figure 5.3: Comparison of the default FRR with the FRR of the metrics learned by solving the optimization problem 5.2

5.4 Exponential separation

To solve this problem, we propose the following optimization problem:

$$M = argmin_A g(A) = \frac{1}{|S|} \sum_{(e_i,e_j)\in S} e^{-k_s*(1-D_A(e_i,e_j))} + \frac{1}{|D|} \sum_{(e,e')\in D} e^{-k_d(D_A(e,e')-1)} \tag{5.3}$$

subject to $A \succeq 0$

f_S and f_D are separated around 1. Points close to 1 will bring a stronger penalty that points farther. The evolution of this penalty is dictated by the two meta-parameters k_s and k_d. Setting 1 to c would just multiply every distance by c and M by c^2.

The optimal M can be found by a simple gradient descent algorithm. Like the two previous ones, this optimization problem is convex. To ensure $M \succeq 0$, M has to be projected into the set of PSD matrixes at each step while minimizing the difference in Frobenius norm before and after the projection[1].

$$M = argmin_{M'} \|M - M'\|_F : M' \geq 0$$

This projection is done by decomposing $M = X^T \lambda X$ where λ is a diagonal matrix containing the eigenvalues of M. Then, M is set to $M = X^T \lambda' X$ where λ' is again a diagonal matrix but containing $max(\lambda_i, 0)$ on its diagonal.

Changing the value of k_d will allow the compromise between identification performance and pruning efficiency. k_s, however, can be set once and for all, as there is no disadvantage of keeping the FRR as small as possible. Modifying k_d will also tune the weight of points in D in comparison of the weight of points in S. This will slightly change the scale of the solution.

In the case where $k_s = k_d$, our framework is equivalent to the standard margin maximization formulation proposed by [6][4]

$$M_{margin} = argmin_A g_m(A) = \sum_{(e_i,e_j)\in\{S\cup D\}} max(0, y_{ij}(D_A(e_i,e_j) - 1))$$

with $y_{ij} = 1$ if e_i and e_j are from the same individual and $y_{ij} = -1$ if they are from different individuals.

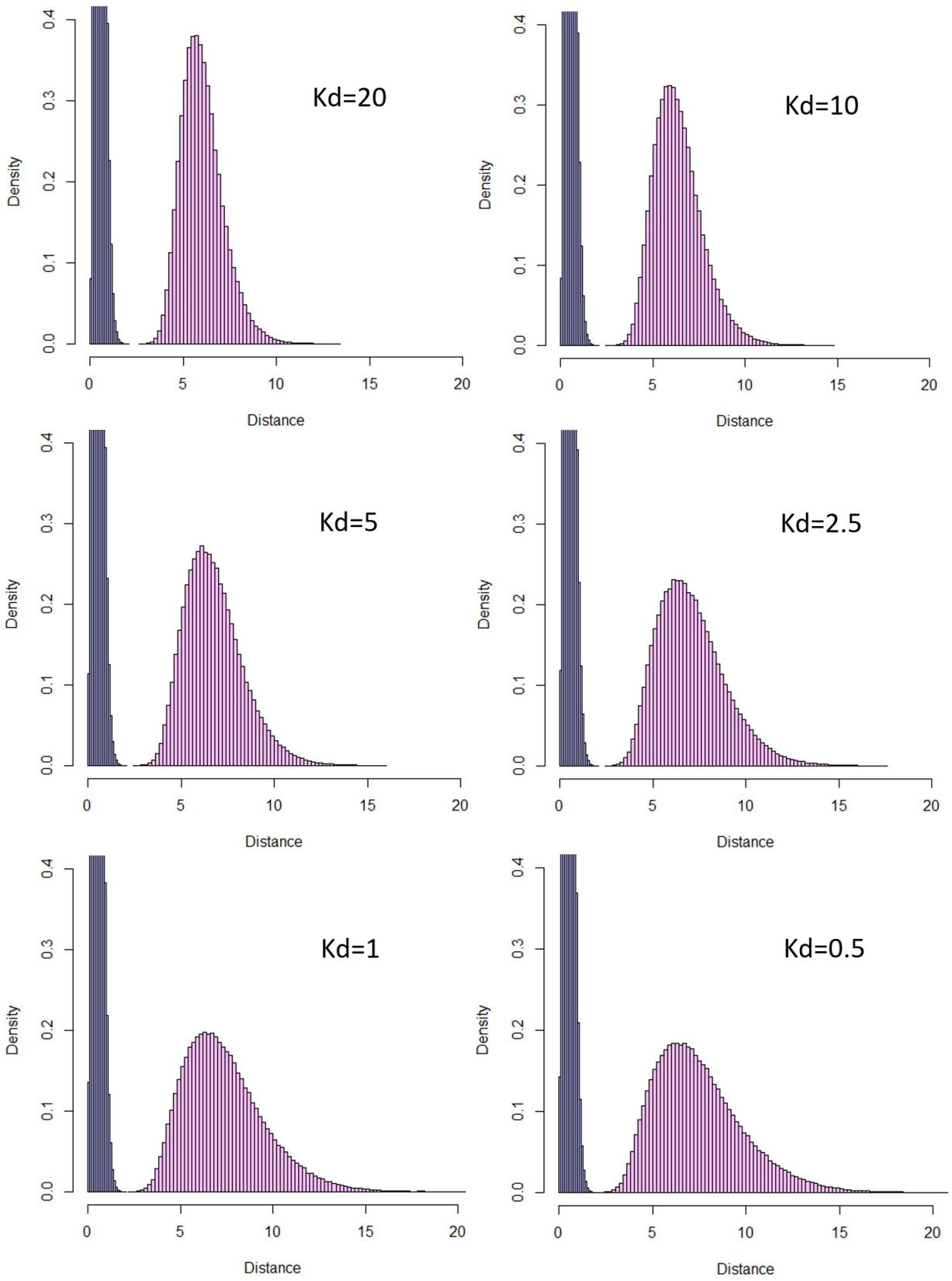
Indeed, the hinge loss $h(x) = max(x, 0)$ is not derivable and a popular approximation is the generalized logistic function[6] $l_\beta(x) = \frac{1}{\beta}(1 + e^{\beta x})$. $l_\beta(x)$ tends to the hinge loss when β, called the sharpness parameter, increases: $\lim_{\beta\to\infty} l_\beta(x) = h(x)$. Also, $l_\beta(x) \propto e^{\beta * x}$ in the region $x \ll 1$. As the margin is being maximized, f_S and f_D will probably not intersect[5], and $l_\beta(x)$ will stay in the exponential zone $x \ll 1$. The results obtained by minimizing the hinge loss would then be the sames as the ones of our framework for $\beta = k_s = k_d$. Using the hinge loss would give a special status to the threshold 1 because the behaviour of the hinge loss changes radically at $x = 1$, unlike simple exponentials. As already mentioned, we do not try to minimize the number of misclassification for this threshold, as the used threshold will be significantly less. Hence, there is no need to minimize the hinge loss.

A train set with $|S| = |D| = 250.000$ has been used[6]. The test set is still the same. First, k_s and k_d are posed to be equal, in order to maximize the margin. Using a validation set, we found that $k_s = k_d \approx 20$ provides the biggest margin. Under 20, the care is not enough on the border points. Above 20, there is too much overfitting. k_d is then progressively reduced in order to learn metrics allowing for a better pruning. The results on the test set are plotted on figure 5.4. The scale of the learned metrics were not identical. To compensate that, they have been scaled such that the threshold at which their FRR is equal to 0.001 are equal to 1.5. This way, they all have approximately the same FRR, and they can be compared uniquely on the separation between f_S and f_D and the extent of f_D.

[4]The author chose to use the squared of the distance $D_A^2(e_i, e_j)$ for convenience.

[5]f_S and f_D won't be close to intersect, even when the metrics are evaluated on the test set (figure 5.4).

[6]As this method is more promising, the effort to implement the gradient descent with Rcpp to boost R performance has been made, such that bigger sets can be used for training.


Kd=20
Kd=10
Kd=5
Kd=2.5
Kd=1
Kd=0.5
Density
Distance
0.0
0.1
0.2
0.3
0.4
0
5
10
15
20

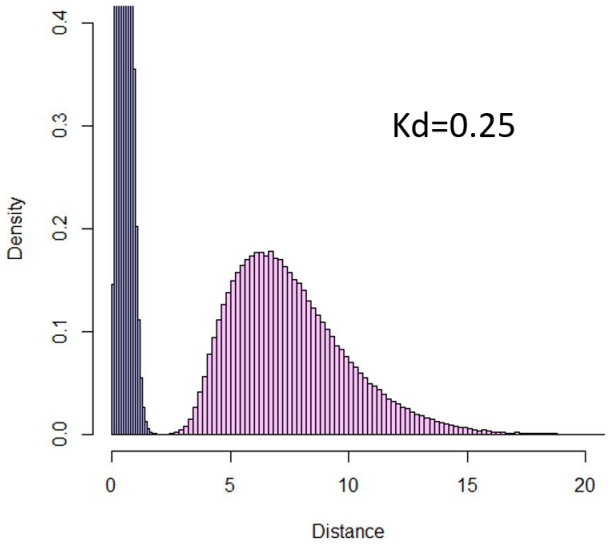


Figure 5.4: Distance distributions on the test set of the metrics learned by the exponential separation method.

As expected, f_S and f_D get less and less separated when k_d is decreased which will cause higher error rates. However, it can be observed that f_D is getting flatter, which will cause a more efficient pruning.

This method provides stronger results that the previous one because the FRR is more constrained. Each iteration of the algorithm is also faster because there is no need to use the inner loop of the projected gradient descent algorithm, as there is only one constraint to impose. However, the number of iteration required for convergence has been observed to be larger, especially for a small k_d.

5.5 Metric choice

In this section, the optimal identification performance as well as the pruning efficiency of the different metrics will be analyzed. The optimal metric will then be shown to depend on the population size, the time constraint and an eventual space constraint.

On figure 5.5(a) is represented the optimal F-score[7] of the metrics for population sizes up to 3M. This would be the F-score of these metrics if no time constraint was involved. As expected, a higher k_d gives a better optimal F-score. On figure 5.5(b) can be seen the optimal thresholds for these metrics. A higher k_d will cause a higher optimal threshold. This makes sens as metrics with a high k_d have a lower FAR[8]. They can then afford to use higher thresholds. In the following, the metric learned with $k_d = x$ will be referred to as M_x.

[7] Note well that we plotted $1 - F^*$

[8] And the same FRR, as they have been scaled based on it.

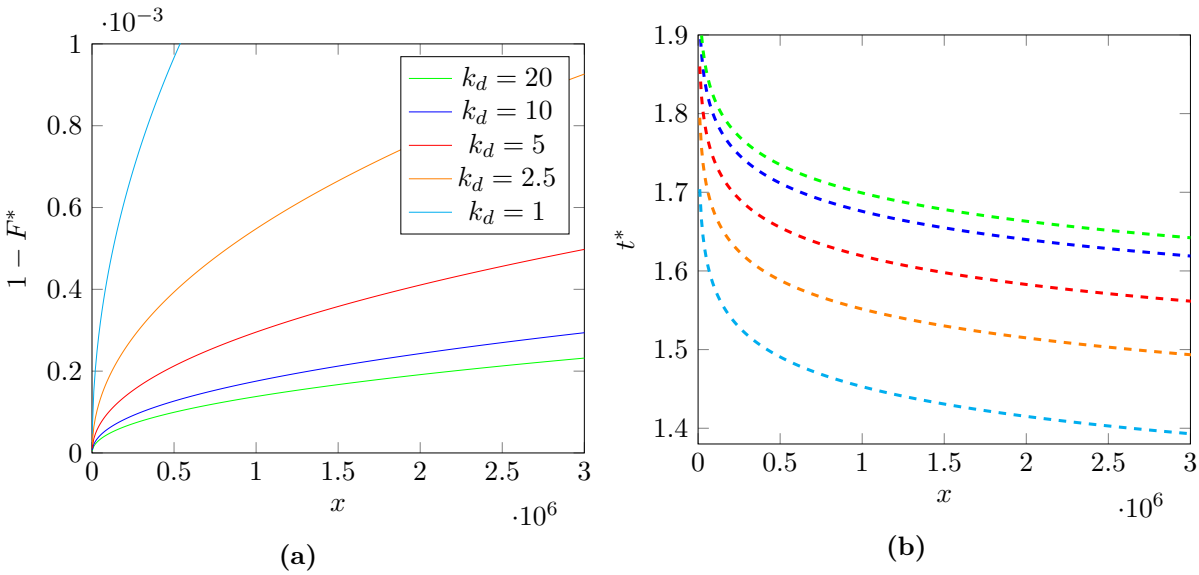


Figure 5.5: Optimal identification F-score of the metrics learned with different k_d (a). Optimal thresholds of these metrics (b).

In presence of a time constraint, the optimal identification performance of the high separation metrics will never be achieved for two reasons. First, the required thresholds are substantially higher. This will cause a huge drop in pruning efficiency. Second, for a given threshold, the pruning of these high separation metrics will be lower than the ones with low separation. On figure 5.6 is depicted the fraction of unfiltered examples, using fixed-queries vantage trees with a random prototype selection (plain) and the optimized rule of section 2.2.2 (dotted), for the different metrics (t=1.5). Low separation metrics are clearly more pruning effective. Nonetheless, the gain obtained by using optimized prototypes is bigger for high separation metrics. In the case where only vantage trees are used, the decision time of M_1 will be more than three times lower than the one of M_{20}, for a constant threshold and optimized prototypes, which is a significant improvement.

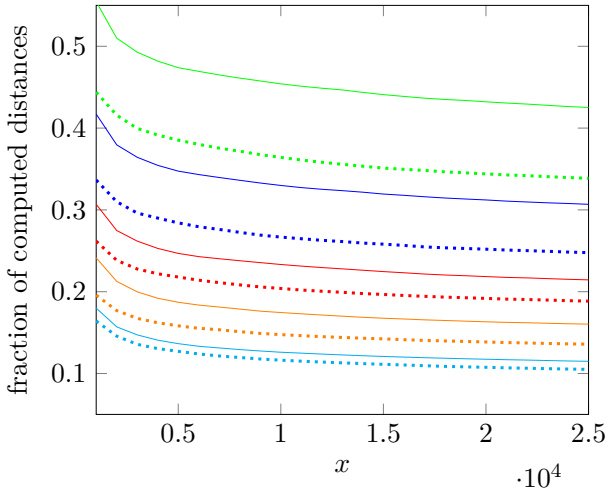


Figure 5.6: Fraction of unfiltered examples of a fixed vantage tree, using the different metrics (t=1.5, random (plain) and optimized (dotted) prototype selection).

In the rest of this chapter, the assumption that the time of the search is measured in distance computations will be made again. The algorithm developed in section 2.6 can then be used. The amount of pruning for all above metrics, for a threshold of 1.5 and up to 500 prototypes can be seen on figure 5.7(a). Clearly, a small k_d will be more efficient. This better pruning will allow the efficient metrics to use larger maximal thresholds when a time constraint is involved. Nonetheless, it can be spotted that there is no more gain from going below $k_d = 1$, even if f_D still gets broader. This can be explained by the fact that unfiltered points will be the points that are the closest to the query. If the separation between the two curves narrows, the closest points will be closer, and hence may be harder to filter (as was seen in section 2.2). This is supported by the fact that a $k_d < 1$ still provides gain for a low number of prototypes. On figure 5.7(b), the maximal thresholds are displayed for a time constraint of 7.500 distance computations and a maximal of 500 prototypes, compared with the optimal thresholds (dashed). Metrics with a good separation have higher optimal thresholds, but lower maximal ones. For both of these reasons, the time constraint will become limiting for them a lot sooner.

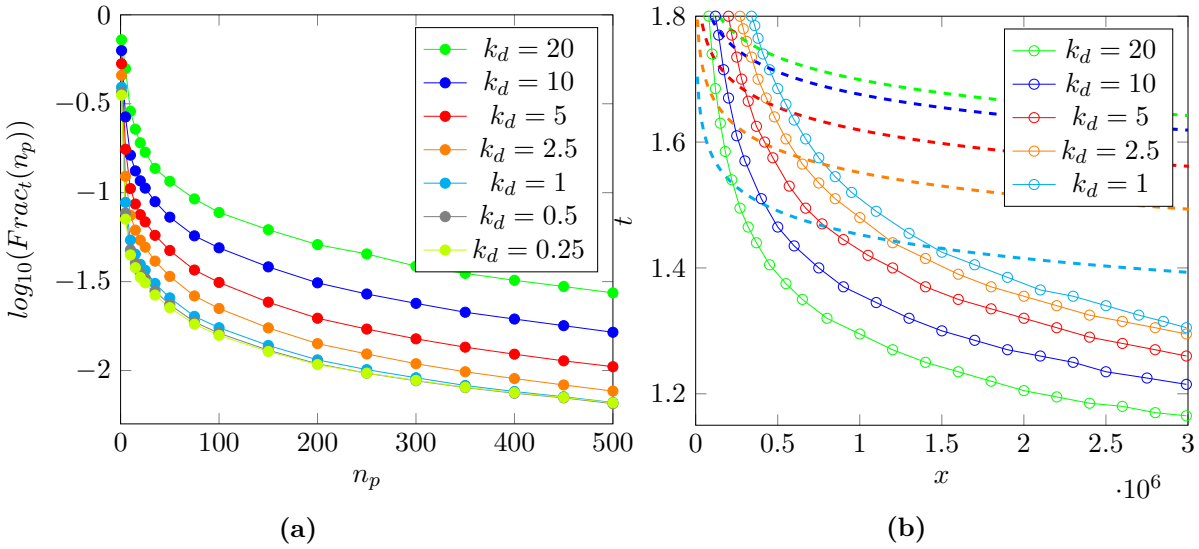


Figure 5.7: Unfiltered fraction of examples, for the different metrics, up to 500 prototypes (t=1.5) (a). Maximum (plain) and optimal (dashed) thresholds of these metrics, for a time constraint with $\eta_d = 7500$ (b).

It was seen on figure 5.5(a) that a higher k_d offers better identification performance. However, when the time constraint mentioned above is added, the results are radically different, as can be observed on figure 5.8. The dashed line represents the performance after the time constraint imposed a lower threshold than the optimal one. It can be seen that $k_d = 20$ is the best only for population up to 100k individuals. Between 100k and 225k, $k_d = 10$ provides the best results. Then $k_d = 5$ between 225k and 610k. Finally, $k_d = 2.5$ has the lower error rates from 610k to 3M. It can be guessed that $k_d = 1$ would eventually overcome $k_d = 2.5$ for populations a little bigger than 3M. This shows well that the best suited metric depends on the size of the population and the time constraint involved.

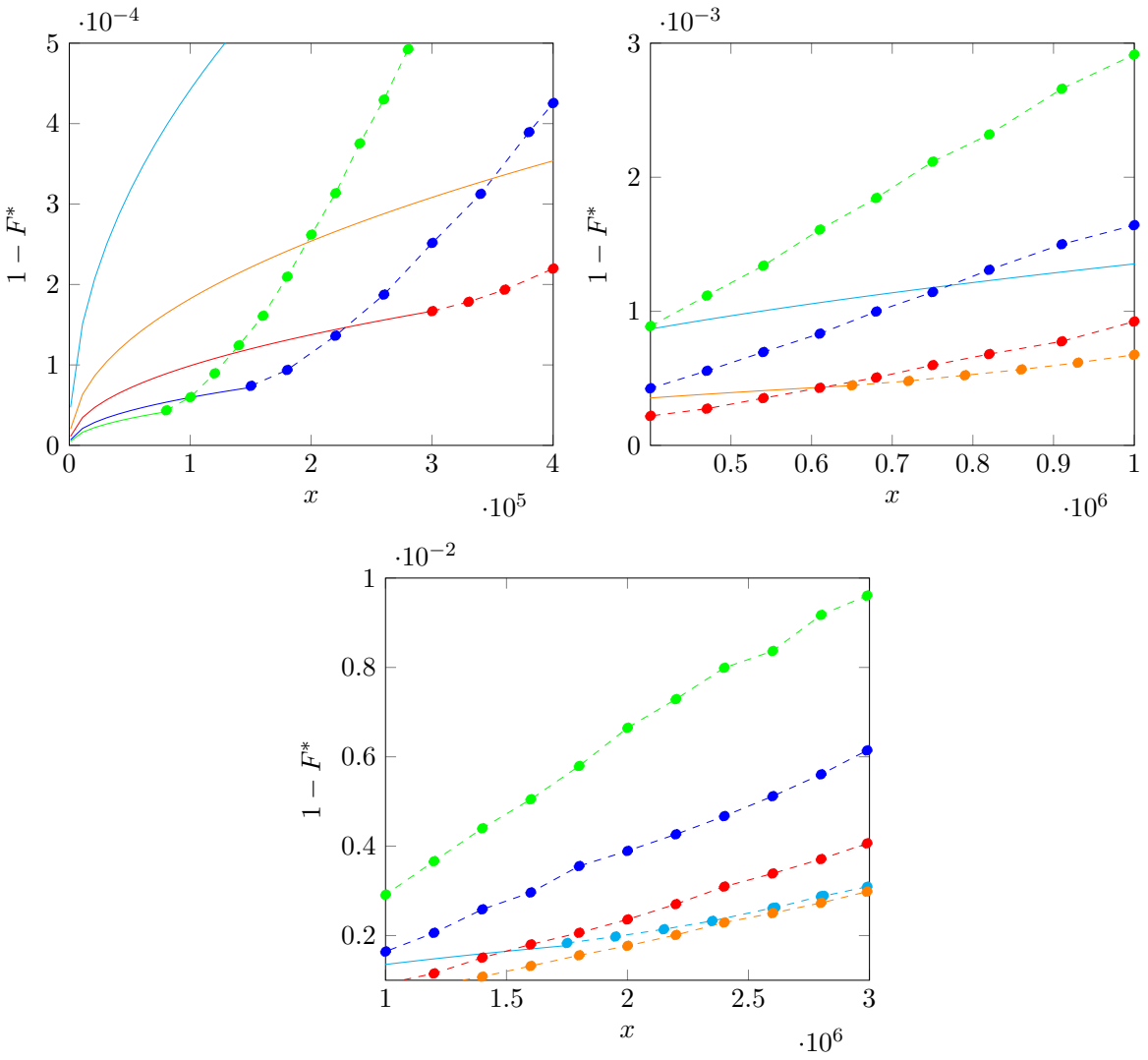


Figure 5.8: F-score of the metric learned with different k_d, with a time constraint of $\eta_d = 7500$, for population sizes up to 3M individuals.

It may not be so apparent on figure 5.7(b), but the difference between maximum thresholds of the different metrics reduces when the population size increases. This is due to the fact that a slight difference of the threshold value will make a bigger difference in pruning for low thresholds than for high thresholds, as was observed on figure 2.5. As the error rates behave exponentially, a fixed difference of thresholds will affect them in the same way whatever the threshold values are. As a consequence, the F-score of metrics with a high k_d is expected to slightly catch up the F-score of metrics with a low k_d for very large population sizes.

5.6 Effect of a space constraint

In the previous section, the metrics were compared under a constraint of a maximum of 500 prototypes. If this constraint is released, the differences between the metrics will narrow. Indeed, high thresholds and hard pruning metrics will use more prototypes to counter-balance their lack of pruning, as was seen on figure 4.3 (for the thresholds). Moreover, even if all metrics use the same number of prototypes, the difference in maximal thresholds allowed greatly depends on this number. Consider an new time constraint with $\eta_d = 150.000$ and a maximum of only 20 prototypes. The corresponding maximal thresholds can be seen on figure 5.9. Clearly, the difference in maximal thresholds has largely increased, which will increase the difference of F-score between the metrics. As a space constraints limits the number of prototypes that can be used, it will also increase the difference between metrics. This difference will increase with the population size, as the number of prototypes allowed by the space constraint decreases with it. The equivalent of figure 5.8 with no bound on the number of prototypes can be found in Appendix C.

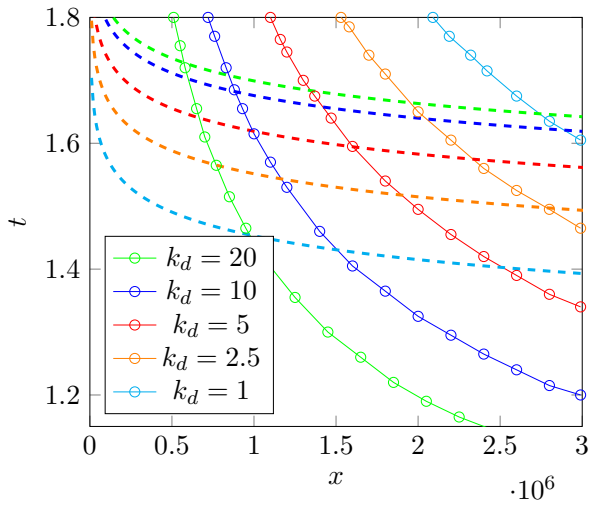


Figure 5.9: Maximum (plain) and optimal (dashed) thresholds of the learned metrics, for a time constraint with $\eta_d = 150.000$ and a maximum of 20 prototypes.

On figure 5.10 is displayed the results obtained with a time constraint of $\eta_d = 50.000$ and a space constraint of 4Go. Figure 5.10(b) depicts the number of used prototypes (plain) before reaching the maximum allowed by the space constraint (dashed). The need to use pruning efficient metrics is even more apparent.

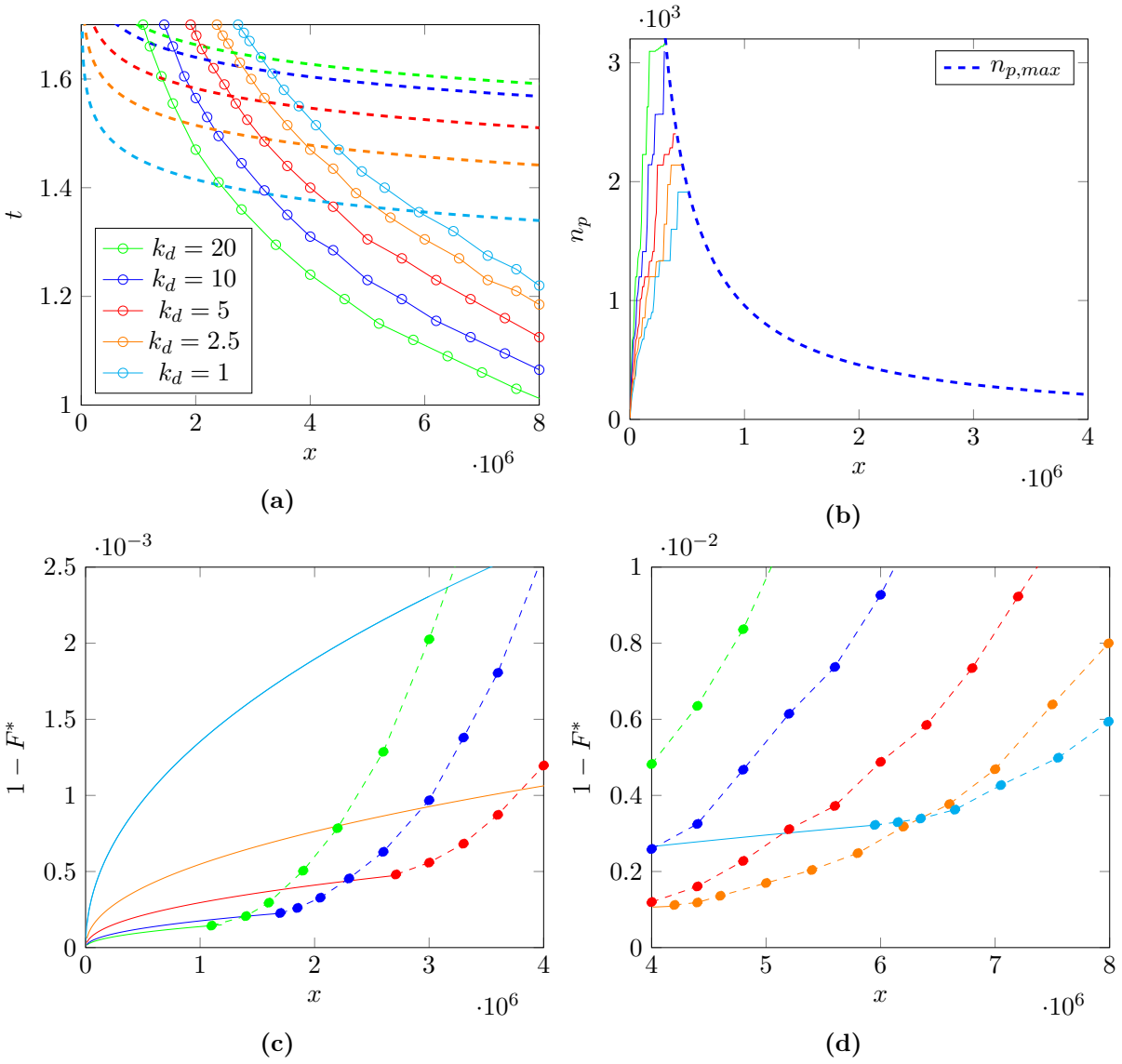


Figure 5.10: Optimal (dashed) and maximal (plain) thresholds for the different metrics given a time constraint with $\eta_d = 50.000$ and a space constraint of 4Go (a). Number of used prototypes (plain) and upper bound on this number given by the space constraint (dashed) (b). Effective F-score of the metric given these constraints, for populations sizes up to 8M (c-d).

5.7 Comparison with the default metric

The default metric, introduced in section 1.5.2 and used throughout this thesis is not able to give as good results as the ones obtained with the metric learning method presented in section 5.4. A metric can be considered strictly superior, noted by $<$, to another metric only if it has lower error rates and a more efficient pruning. By comparing figure 5.7 and 2.5, it can be seen that the pruning offered by the default metric, M_{def}, is inferior to the one of $M_{k_d=2.5}$. Using figure 5.8 and 3.4, it can also be observed that the optimal identification performance of the default metric is also worse. As a consequence, $M_{def} < M_{k_d=2.5}$.

5.8 Extensions

A generalization of our objective function can be obtained by adding another parameter, K_s

$$g_E(A) = \frac{K_s}{|S|} \sum_{(e_i,e_j)\in S} e^{-k_s*(1-D_A(e_i,e_j))} + \frac{1}{|D|} \sum_{(e,e')\in D} e^{-k_d(D_A(e,e')-1)} \tag{5.4}$$

which is then equivalent to

$$g_E(A) = K \sum_{(e_i,e_j)\in S} e^{k_s D_A(e_i,e_j)} + \sum_{(e,e')\in D} e^{-k_d D_A(e,e')} \tag{5.5}$$

with $K = \frac{|D|}{|S|} K_s e^{-(k_s+k_d)}$

Increasing K will give more weight to the FRR and vice-versa. Experimentally, it was observed that, for large k_d, modifying K will just increase or decrease the scale of M, but the optimal solution would be roughly equivalent. For small k_d, changing K has more important effects. This model is able to learn a broader class of metrics, but finding the right choice of parameters might be tedious and superfluous[9]. A simple cross validation protocol cannot be used to find the best choice of parameters, because the metrics cannot be compared only on accuracy. The pruning efficiency has to be taken into account too. Nonetheless, better metrics can probably be found, as there are much more in this framework.

5.9 Limits

One default of the proposed metric learning method is that it is proned to overfitting, especially for large k_s and k_d, as exponentials give a lot more weight to border points. Moreover, the projection step into the set of PSD matrices scales in $O(d^3)$ which will make the algorithm unusable for high dimensional data. The complexity in term of training points is $O(n)$, however. This low complexity can partially solve the overfitting problem by using large training sets, if available.

[9]Different K values were tried without significant improvement in both accuracy and pruning

Chapter 6

N examples per person

In this chapter, the assumption that every known individual has exactly one stored example is released. First, it is generalized into the N examples per individual assumption. The effects on the error rates and on the time constraint will be studied. Later, different number of examples for each individual will be allowed.

Assumption 6 - N examples per individual: Each known person has exactly N stored examples.

6.1 Effect on the error rates

We first define two events, A_j and R_j. A_j occurs when there has been a (false) acceptance with the j^{th} stored example of an individual. R_j happens when the query example has not matched with the j^{th} stored example of the individual it belongs to. We also define $P_i(A_j)$ as the probability of a false acceptance with the j^{th} stored example of individual i and $P_i(R_j)$ as the probability that the query example, knowing it belongs to individual i, does not match with the j^{th} stored example of i. From these definitions[1], $P(A_j) = \sum_{i \in P} P_i(A_j)$ and $P(R_j) = E_i[P_i(R_j)]$. Finally, as $P(A_j)$ and $P(R_j)$ do not depend on j (the probability to match any example of an individual is the same), they will be referred to as $P(A) = FAR_{N=1}$ and $P(R) = FRR_{N=1}$, respectively.

Keeping N examples for each known individual has profound effects on both error rates. For a false rejection to occur, no example belonging to the query individual must be matched. Concerning the FAR, a single false match is sufficient to have a false acceptance. The FRR is then reduced by increasing N whereas the FAR is increased. Formally,

$$FRR_N = P(\bigcap_{j=1}^{N} R_j) = E_i[P_i(\bigcap_{j=1}^{N} R_j)] \leq FRR_1 \tag{6.1}$$

$$FAR_N(t) = \sum_{i \in P} P_i(\bigcup_{j=0}^{N-1} A_j) \geq FAR_1 \tag{6.2}$$

For N=2,

$$FRR_2 = E_i[P_i(R_2 \cap R_0)] = E_i[P_i(R_2|R_1) * P_i(R_1)] = FRR_1 * P(R_2|R_1) \tag{6.3}$$

[1]Using the small false acceptance rate assumption, defined in section 3.1

Indeed, $P(R_2|R_1)$ is equal to the weighted sum of the $P_i(R_2|R_1)$ for every known individual. The weight of each individual i is equal to the probability of a measured false rejection to be his own, or $\frac{P_i(R)}{x*P(R)}$.

$$P(R_2|R_1) = \sum_{i \in P} \frac{P_i(R_1) * P_i(R_2|R_1)}{x * P(R_1)} = \frac{E_i[P_i(R) * P_i(R_2|R_1)]}{FRR_1}$$

Also,

$$FAR_2 = \sum_{i \in P} P_i(A_2 \cup A_1) = \sum_{i \in P} P_i(A_2) + P_i(A_1) - P_i(A_2 \cap A_1) =$$

$$2P(A) - \sum_{i \in P} P_i(A_2 \cap A_1) = 2P(A) - P(A_2|A_1) * P(A_1) = FAR_1 * (2 - P(A_2|A_1)) \quad (6.4)$$

because

$$P(A_2|A_1) = \sum_{i \in P} \frac{P_i(A_1) * P_i(A_2|A_1)}{P(A_1)} = \frac{\sum_{i \in P} P_i(A) * P_i(A_2|A_1)}{FAR_1}$$

The FRR decrease is modulated by $P(R_2|R_1) > FRR_1$. Knowing that the test example of a person has been rejected by one of its stored example will significantly increase his probability to be rejected by other stored examples. The first rejection is indeed likely due to the fact that the person has made a lot of mistakes in the test example. The probability to have other rejections is then increased.

The FAR increase will be maximal when $P(A_2|A_1) = 0$. If $P(A_2|A_1) = 1$, the FAR will not increase at all. In practice, $P(A_2|A_1)$ will be greater than 0. Indeed, knowing that a person has been falsely accepted by some other person will increase his probability to be accepted by other examples of the same person. Indeed, these two persons are likely to have close personal data.

In general, for $N > 1$,

$$FAR_N = FAR_{N-1} + FAR_1 - P(A_N| \bigcup_{j=1}^{N-1} A_j) * FAR_{N-1} \quad (6.5)$$

$$FRR_N = FRR_{N-1} * P(R_N| \bigcap_{j=1}^{N-1} R_j) \quad (6.6)$$

On figure 6.1(a) is shown the FRR for N going from 1 to 5. As can be seen, the FRR is first greatly decreased when N goes from 1 to 2. However, increasing N further gives less and less benefit. This comes from the fact that $P(R_N| \bigcap_{j=1}^{N-1} R_j)$ increases when N increases, as can be seen on figure 6.1(b).

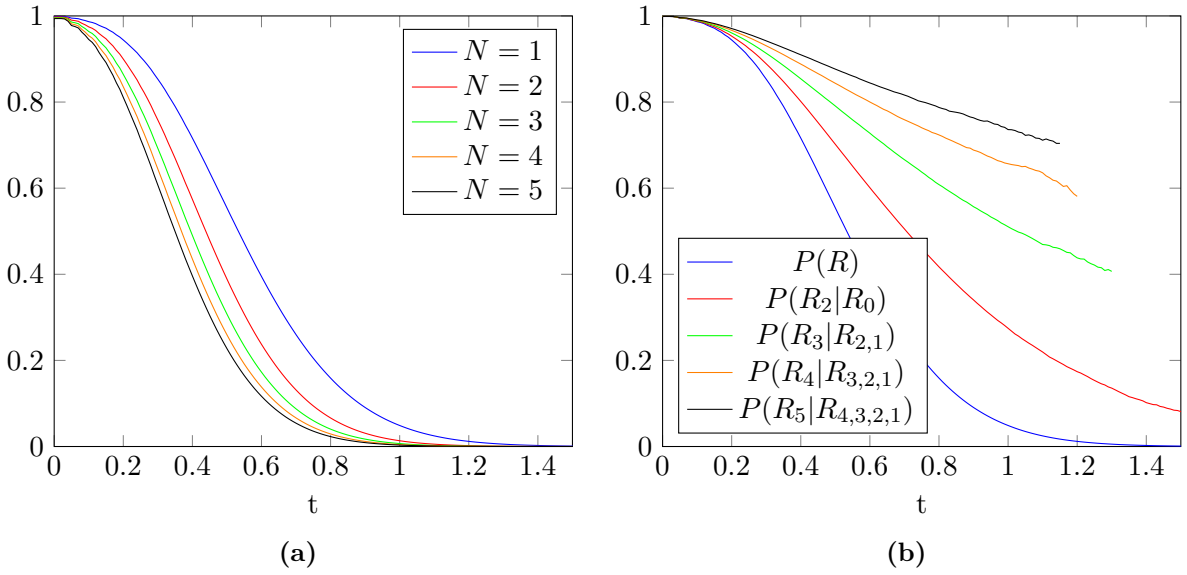


Figure 6.1: Evolution of the FRR with the threshold, for N up to 5 (a). Conditional rejection probabilities (b).

On figure 6.2 is depicted the evolution of $P(A_1|A_0)$ with t. As can be observed, $P(A_1|A_0)$ decreases when t decreases. It seems that it tends to 0 for small thresholds (interesting ones). However, for such small thresholds, the same problem than the one to measure accurately the FAR arises. The left most points of figure 6.2 have a low statistical significance. Only two pairs of examples that have matched for $t < 1.6$ were found. For each of these pairs, twenty noisy examples were generated for both the query and the stored individual in order to measure $P(A_1|A_0)$ as accurately as possible. Despite this low significance, it is clear that the tendency is to the decrease. The conservative approximation $P(A_1|A_0) \approx 0$ will be made. This approximation will give a FAR bigger than the actual, and hence a F-score smaller. This means that a lower bound on the F-score will be computed. In such an approximation, using equation 6.5, the following relation is valid

$$FAR_N(x,t) = N * FAR_{N=1}(x,t) \tag{6.7}$$

Using figure 6.1 and equation 6.7, the F-score for different N can be computed. This latter is depicted on figure 6.3(a), for $\beta = 1$. As can be observed, increasing N is beneficial for the F-score. However, the gain decreases when N increases. Also, a higher or lower N does not favor a big or small database size. The optimal N will not depend on the size of the population. On figure 6.3(b), several things can be spotted. First, for all β, the error rates starts to decrease fast with N. However, the gain levels off and a plateau is reached. After that, the performance starts to worsen. It seems that $N^* \approx 15$. Second, the gain obtained by increasing N is higher for $\beta > 1$, which makes sens as a high β favors a small FRR.

The FRR is reduced because N is increased. This increases the FAR. However, the FAR is decreased more by the reduction of the optimal threshold. This reduction can be seen on figure 6.3(c). Nonetheless, the FAR does not decrease as much as the FRR (figure 6.3(d)). Indeed, as was shown before, $\frac{FRR^*}{FAR^*} = \frac{a_{far}}{\beta^2 a_{frr}}$. Increasing N increases a_{frr} whereas a_{far} is not modified[2]. Hence, $\frac{FRR^*}{FAR^*}$ decreases.

[2]The transformation $c_{far} \leftarrow N * c_{far}$ occurs when N examples per person are stored.

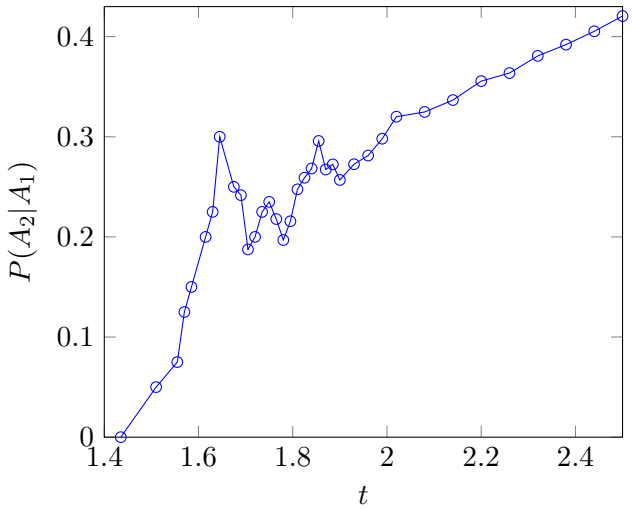


Figure 6.2: Evolution of $P(A_2|A_1)$ with the threshold.

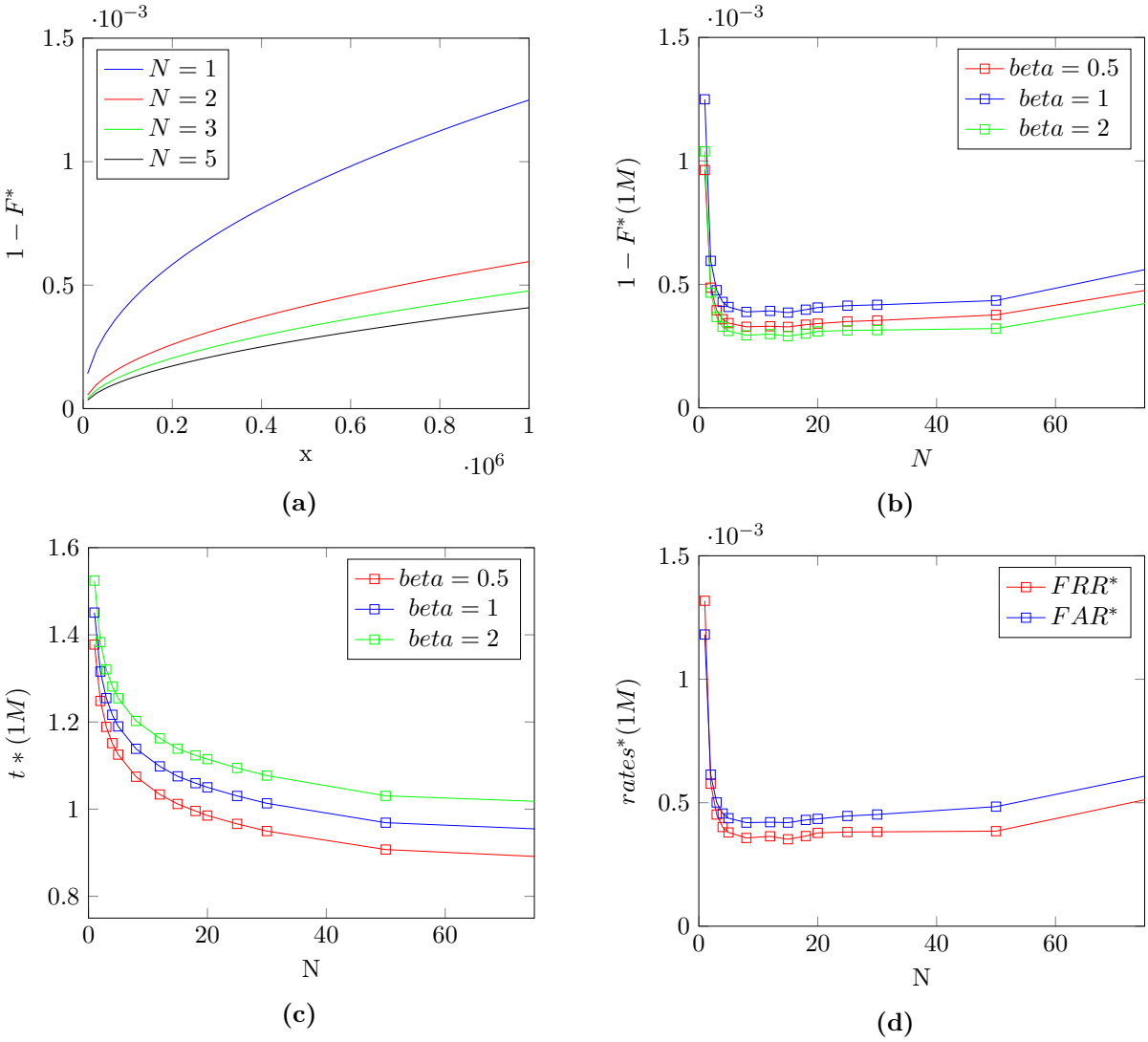


Figure 6.3: Optimal F-score for different N values in function of the population size (a). Evolution of the optimal F-score with N, for a population size of 1M (b). Evolution of the optimal thresholds with N, for a population size of 1M (c). Comparison of the optimal FRR and optimal FAR in function of N (d).

6.2 Time constraint

Increasing N has two opposing effects on the time constraint. First, it increases the number of examples. This means that, for the same threshold, N times as many distances as in the single example case will be computed. Formally, $t_{max}(N = n, x) = t_{max}(N = 1, x * n)$. This will tend to make the time constraint more restrictive. However, as was seen on figure 6.3(c), increasing N decreases the optimal thresholds, which will in turn increase the filtering.

The maximal threshold allowed for the same time constraint as in section 4.2 ($\eta_d = 3500$) as well as the optimal thresholds can be seen on figure 6.4(a). Both the optimal and maximal thresholds for bigger N are decreased. On figure 6.4(b) is depicted the F-score obtained for these N values. It can be seen that the identification performance is less impacted by the time constraint for bigger N, as counter-intuitive as it may seem. The reason is that decreasing the threshold by a fixed amount will increase more the filtering for small thresholds, as was observed on figure 2.5(b). As higher N values deal with lower thresholds, these thresholds have to be lowered less to meet the time constraint.

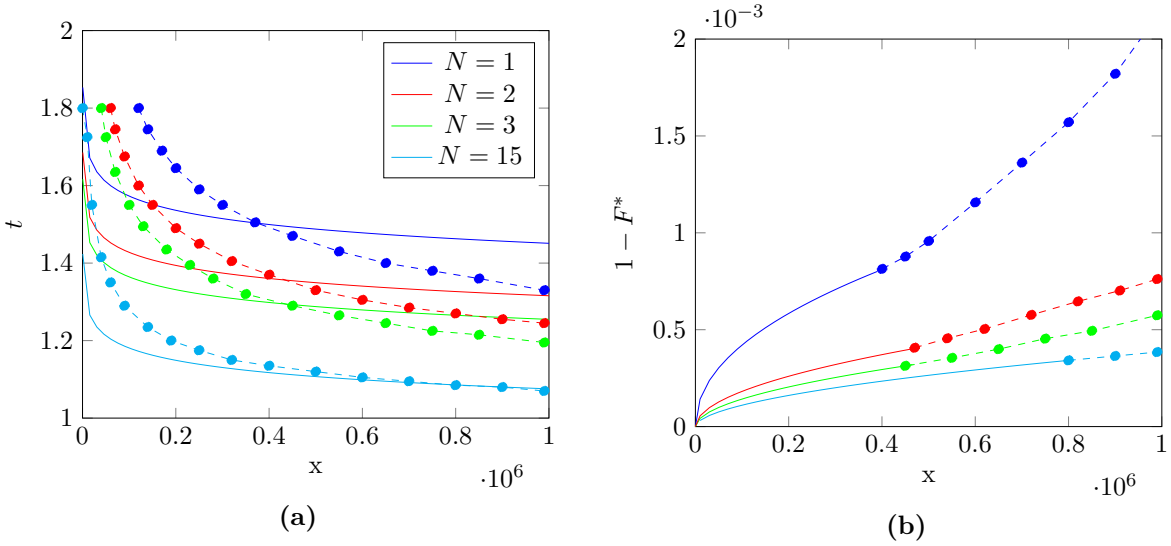


Figure 6.4: Comparison between the optimal and maximal thresholds allowed by the time constraint $\eta_d = 2500$, for different N values (a). Effective F-score obtained, taking into account the time constraint (b). The lines are dashed when the time constraint has become restrictive.

6.3 Space constraint

Unlike a time constraint, a space constraint has disastrous impacts for larger N. Equation 4.5 can be turned into equation 6.8, for $N \geq 1$.

$$n_p(N) < \frac{S_w}{N * |P|} - size_w(e) \tag{6.8}$$

As a consequence, for large n_p:

$$n_{p,max}(N) \approx \frac{n_{p,max}(1)}{N} \tag{6.9}$$

As was observed on figure 4.3(a), the optimal number of prototypes decreases when the threshold decreases. As optimal thresholds for higher N are lower, this will tend to reduce the optimal number of prototypes. However, this effect is counter-balanced by the increasing number of stored examples, which tends to increase the number of prototypes. The optimal number of prototypes for $N = 1, 2, 3$ and 15 can be seen on figure 6.5(a), compared to the maximum number of prototypes allowed by a space constraint of 20Go. There is first a tendency to the decrease for $N = 2$ and $N = 3$, the threshold diminution overcoming the increase of $|E|$. Nonetheless, for larger N, the optimal number of prototypes increases back. Anyway, it is clear that a space constraint becomes more quickly active for larger N and decreases their identification performance more abruptly, as can be seen on figure 6.5(b). The same time constraint with $\eta_d = 3500$ was used. Only $N = 15$ is affected for sizes smaller than 1M. The other curves are identical to the ones in figure 6.4(b).

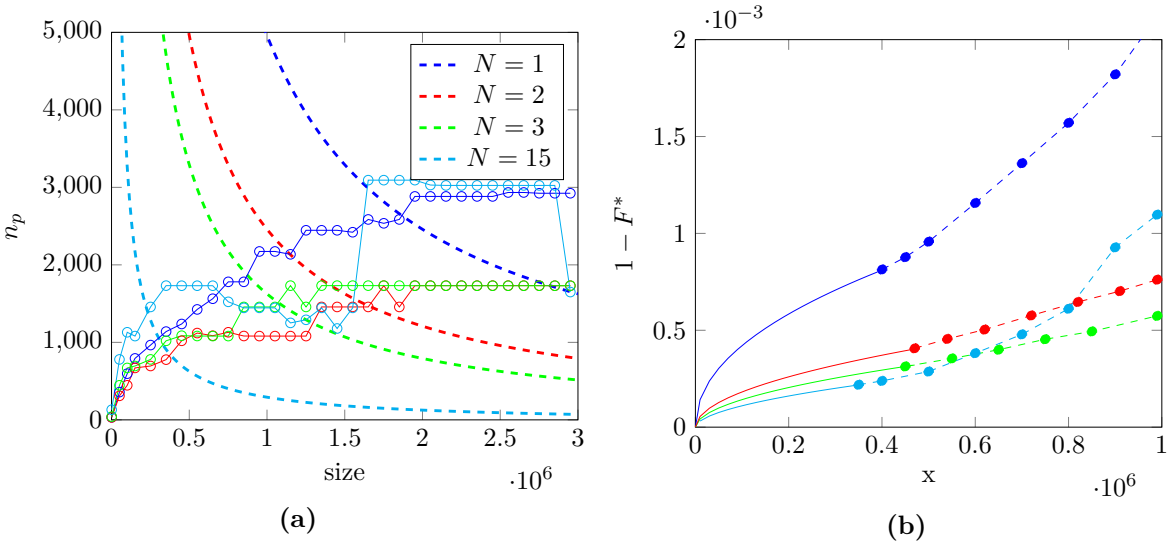


Figure 6.5: Comparison of n_p^* and $n_{p,max}$ for $N = 1, 2, 3, 15$ and a space constraint of 20Go (a). Identification F-score for $N = 1, 2, 3, 15$ taking into account a time constraint with $\eta_d = 3500$ and the space constraint of 20Go (b).

6.4 Weighted FRR

Until now, a false rejection of any patient was supposed to have the same cost. The FRR is then given by the mean of the false rejection rate of every known individual:

$$FRR = \frac{\sum_{i \in P} P_i(\bigcap_{j=1}^{N} R_j)}{x}$$

However, for some reason, a false rejection of a person may be more costly than a false rejection of another one. Or, and that is equivalent, every false rejection has the same cost, but some patients have to be identified more often that others. A weight w_i can be assigned to every individual i such that the FRR becomes

$$FRR = \frac{\sum_{i \in P} w_i P_i(\bigcap_{j=1}^{N} R_j)}{\sum_{i \in P} w_i} \triangleq \frac{\sum_i w_i FRR_{N_i,i}}{\sum_i w_i} \tag{6.10}$$

If every person has to be identified at a frequency $\frac{w_i}{\sum_i w_i}$, equation 6.10 will give the true false rejection rate.

In such a case, keeping the same number of examples for every one would be sub-optimal. Intuitively, it will be interesting to keep more examples of patients with a high weight. Indeed, the diminution of their false rejection rate will decrease more the global FRR. The FAR, on the other hand, is only function of the total number of stored examples: $FAR = FAR' * \sum_i N_i$.

The problem can then be formalized as follows: find t^*, N_i^* such that the F-score is minimized

$$F(N_i, t) = \alpha \frac{\sum_i w_i * FRR_{N_i,i}(t)}{\sum_i w_i} + (1-\alpha) FAR'(t) * \sum_i N_i \tag{6.11}$$

The expected FRR decrease of individual i when his j^{th} example is added, is given by:

$$\delta_{j,i} FRR \triangleq FRR_{j,i} - FRR_{j-1,i} \tag{6.12}$$

As estimating the false rejection rates of each patient will be impossible in practice, we first focus on the case where those rejection rates ($FRR_{j,i}$) are unknown. Because $FRR_{N_i} = E_i[FRR_{N_i,i}]$ for all $N_i \geq 1$, this is equivalent to considering that all patients have the same rejection rates, equal to FRR_{N_i}. Indeed, the expected FRR decrease of a random individual is given by:

$$E_i[\delta_{j,i} FRR] = E_i[FRR_{j,i} - FRR_{j-1,i}] = E_i[FRR_{j,i}] - E_i[FRR_{j-1,i}] = FRR_j - FRR_{j-1} \triangleq \delta_j FRR$$

As a consequence, adding the j^{th} example of individual i at a constant threshold t is expected to reduce the F-score if the following relation is satisfied[3] (using equation 3.7):

$$-\beta^2 w_i \delta_j FRR(t) > FAR(t)' * \sum_i w_i \tag{6.13}$$

Reducing the threshold will have two effects on equation 6.13. First, it will reduce FAR' (right hand side). Second it will increase the absolute value of $\delta_j FRR$ (left hand side) as can bee seen on figure 6.6, at least for interesting thresholds. As a consequence, equation 6.13 will never be made false by reducing the threshold. Then,

$$\forall i, t, t' : t \leq t' \Rightarrow N_i^*(t) \geq N_i^*(t') \tag{6.14}$$

The following relation also holds as a higher w_i increases the left hand side of equation 6.13, without modifying the right hand side:

$$\forall i, j : w_i \geq w_j \Rightarrow N_i^* \geq N_j^* \tag{6.15}$$

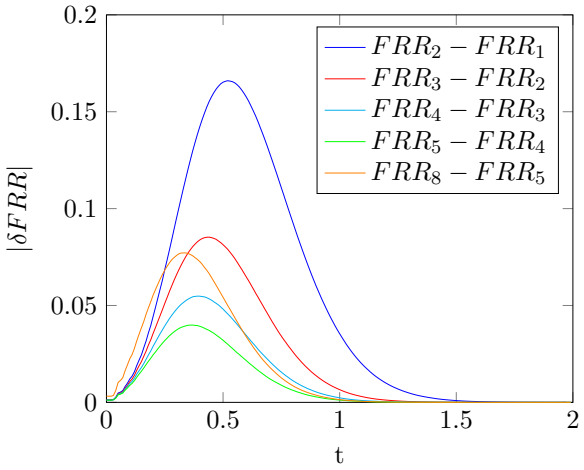


Figure 6.6: Gain in FRR when N is increased.

[3]Note that, as the FRR is decreased when N increases, δFRR is smaller than 0.

Making use of equation 6.13,6.14 and 6.15, algorithm 4 can be used to solve efficiently the weighted FRR problem:

Algorithm 4: Weighted FRR solver

Data: $FAR'(t), FRR(N_i, t), w, \beta, x$
Result: Find t, Ns which provides the maximal lower bound on the F-score
$sort(w, decreasing), counter_{range(N_i)} = 0$
$fs[i] \leftarrow \frac{w[i]}{mean(w)*x}, F_{max} \leftarrow 0, Ns[i] \leftarrow 1$
foreach $t \in [t_{max}, t_{min}]$ **do**
  foreach $n \in range(N_i)$ **do**
    $change \leftarrow true$
    while $counter_n < length(w)$ & $change$ **do**
      $i \leftarrow counter_n, change \leftarrow false$
      if $\beta^2 fs[i] * (FRR(n,t) - FRR(n+1,t)) > FAR'(t)$ **then**
        $counter_n++, Ns[i]++, change \leftarrow true$
      end
    end
  end
  $F \leftarrow F_{score}(Ns, t, w)$
  if $F > F_{max}$ **then**
    $F_{max} \leftarrow F, t_{max} \leftarrow t, Ns_{max} \leftarrow Ns$
  end
end
return t_{max}, Ns_{max}

The algorithm goes through the possible threshold values in a decreasing order. From equation 6.14, this means that no example will ever need to be removed. It also takes advantage of equation 6.15 by sorting the individuals in decreasing order of weight. The variable $counter_x$ contains the index of the individual with the highest weight that has currently $x-1$ stored example. This variable is used to keep track of which examples should be added next.

The results of algorithm 4 are shown on figure 6.7 for the following settings:

- The allowed N_i are 1,2,3,4,5,8,15,25 and 50.
- A sample of 10.000 w_i is taken from the exponential distribution with mean 1.

As can be seen on figure 6.7(a), the optimal F-score has been noticeably decreased, compared to the F-score obtained with the same number of stored example for every individual ($N^* = 15$). The distribution of these N_i is depicted on figure 6.7(b). From equation 6.15, it is clear that the interesting output of the algorithm is not the N_i in themselves, but rather the intervals of w_i that fix these N_i. Thanks to these intervals, the number of examples of any individual can be fixed, based on his weight[4]. For the previous experiment, these intervals are represented on figure 6.8. The only interest of using a larger sample of w_i is to fix these intervals more accurately.

[4]These intervals depend slightly on the size of the population

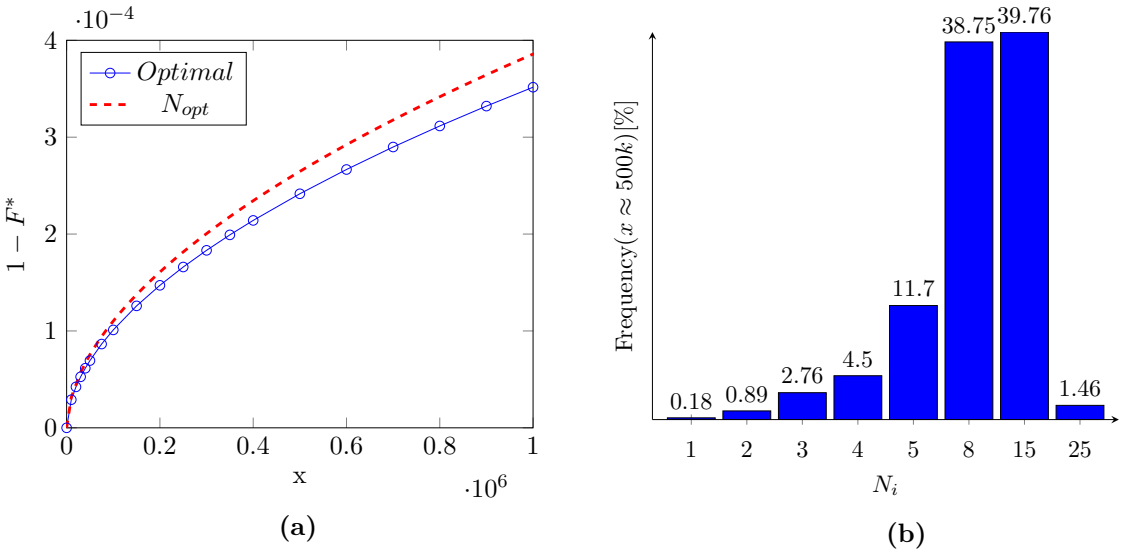


Figure 6.7: Comparison between the optimal F-score when N_i is constant for all individuals ($N_i^* = 15$) and the F-score provided by algorithm 4 (a). Distribution of the N_i among the individuals for a population size of 500k (b).

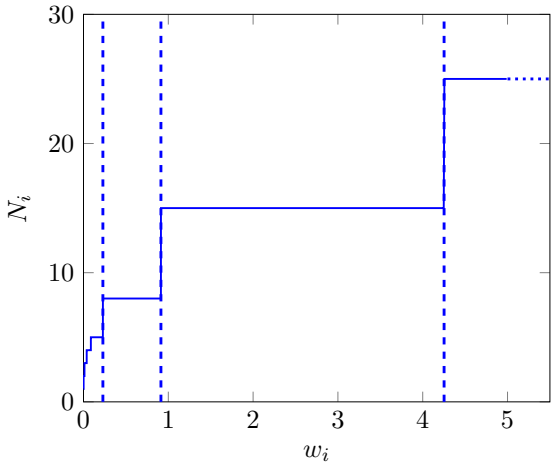
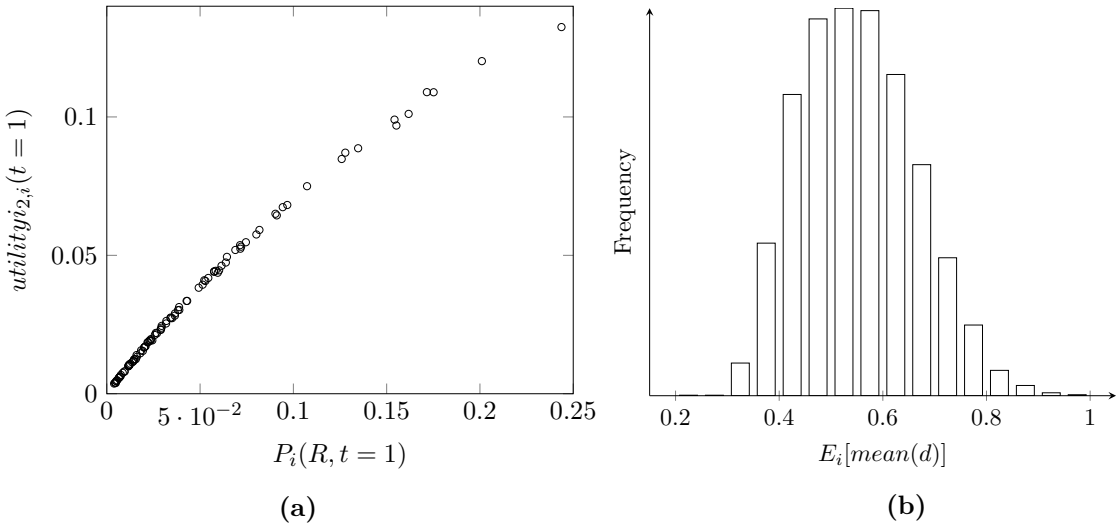


Figure 6.8: Learned function that fixes the number of example N_i of an individual with weight w_i, for a population size of 500k.

6.5 Inhomogeneity

In this section, we assume that the rejection probabilities of every patient are known and we will see if the error rates can be decreased further. For convenience, we define the utility function $utility_{j,i} \triangleq \delta_{j,i} FRR = FRR_{j,i} - FRR_{j-1,i}$. Examples of high utilities should be added in priority as they will reduce the FRR by a larger amount. We first study the relationship between the utility of examples of an individual and his tendency to make encoding mistakes. Then, as in section 6.4, the optimization problem 6.11 will be solved, taking advantage of this additional information.

The utility of the second example of 100 patients, for a threshold of 1, can be seen on figure 6.9(a). As can be seen, keeping a second example of unreliable individuals decreases a lot more their FRR. Their utility is higher. It will then be beneficial to keep more examples of such individuals. Figure 6.9(b) shows the distribution of the mean of the distance between two examples of the same individual. Some individuals have a higher mean distance because their names are longer and/or their noise multiplier is higher.



(a) **(b)**

Figure 6.9: $utility_{i,2}$ in function of $P_i(R)$, for $t = 1$ (a). Distribution of the mean error among individuals (b).

Taking into account this inhomogeneity, equation 6.13 turns into equation 6.16

$$\beta^2 w_i * utility_{j,i} > FAR' * \sum_i w_i \tag{6.16}$$

Figure 6.10(a) shows the utilities of an unreliable person (blue), a randomly selected one (green) and a trustworthy one (red), compared to the FAR of a population of 1M individuals. In the case where $\forall i \in P : w_i = 1$ and $\beta = 1$, the plotted utilities are equal to the left hand side of equation 6.16 whereas the FAR is equal to the right hand side. As the utilities of unreliable individuals are bigger, their examples will be added for higher thresholds. As can be seen on figure 6.10(b), the trend continues for larger N, here $N = 5$, and is even increased.

The optimal F-score obtained[5] with a sample of 100 patients for which the utilities were estimated can be seen on figure 6.11(a). In blue is represented the results for exponentially distributed w_i. The orange curve corresponds to the case where only the utilities are taken into account ($w_i = 1$ for everyone). The dashed line has been obtained by considering only the 100 sample points to estimate the FRR, in order to avoid any bias due to the small size of the sample. In this case, $N_i^* = 8$.

As can be seen, the difference is considerable. The number of examples for each patient is also more evenly separated (figure 6.11(b)). Some patients are so unreliable that they require 50 stored examples. Other patients are very trustworthy and only need two such examples. Unfortunately, in a real life application, evaluating the utilities of each person will be impossible,

[5]As equation 6.15 is no longer valid, algorithm 4 has to be slightly modified.

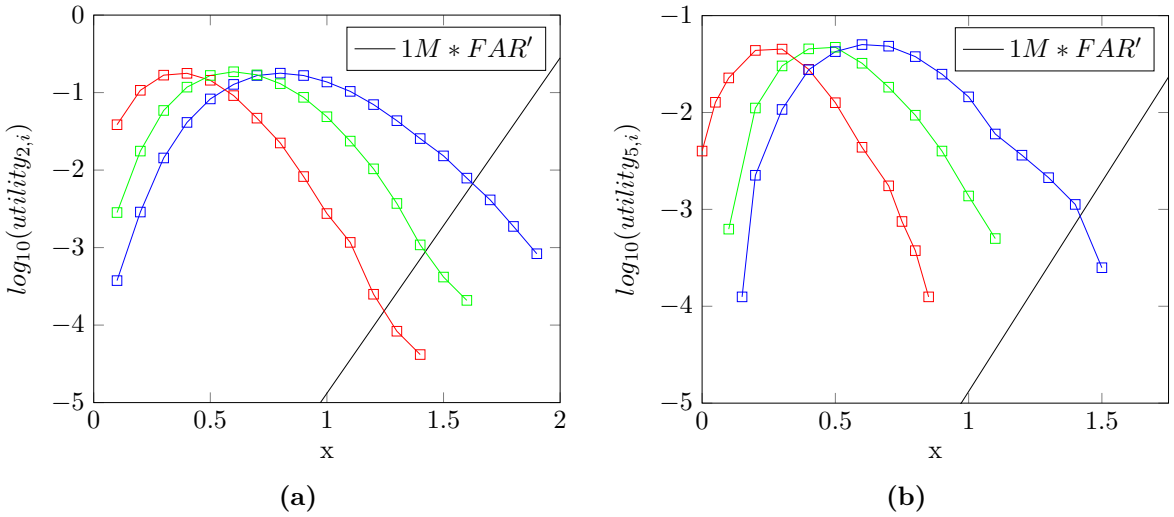


Figure 6.10: $utility_{i,2}$ in function of the threshold for a very unreliable person (blue), a randomly selected one (green) and a trustworthy one (red)(a). Same but $utility_{i,5}$ (b)

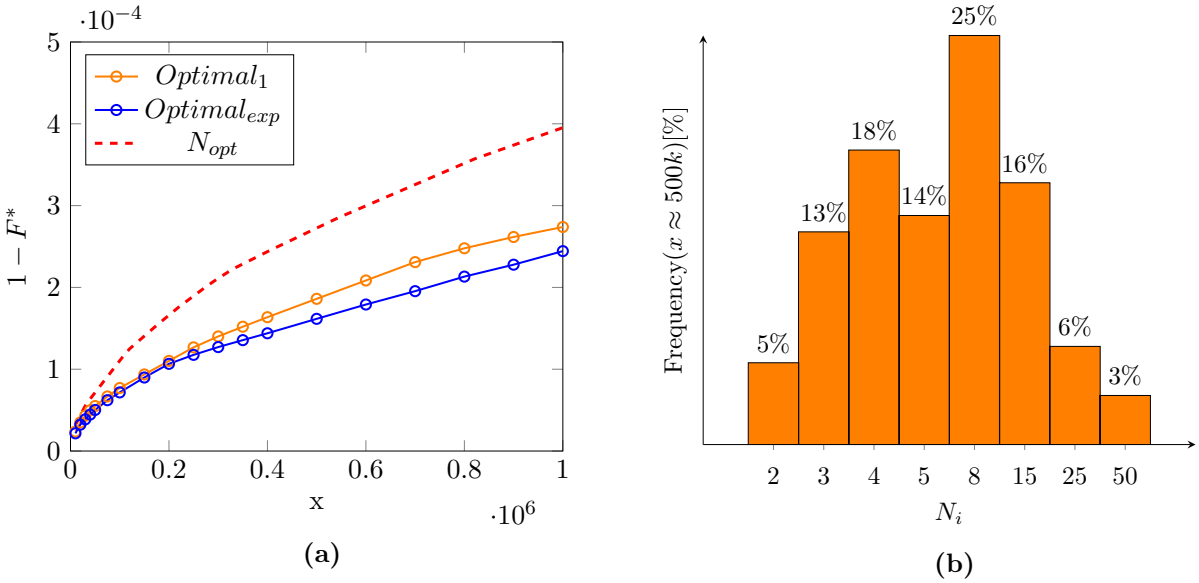


Figure 6.11: Comparison between the optimal F-score with $\forall i N_i = 8$ and with N_i obtained via the modified algorithm 4 (a). Distribution of the N_i among individuals ($\forall i w_i = 1$) (b).

as it requires a lot of data for every person. The results presented are what could be obtained if the number of examples were chosen perfectly, which is impossible in practice. Nonetheless, it is clear that examples of unreliable patients should be added first.

6.6 Limits

As mentioned earlier, what has been computed from section 6.1 are only lower bounds on the F-score. We supposed that every new example increases the FAR by the same amount, FAR'. This assumption is not really true, however.

As can be seen on figure 6.12, there is a clear relationship between the reliability of a person and his $P_i(A_1|A_0)$.

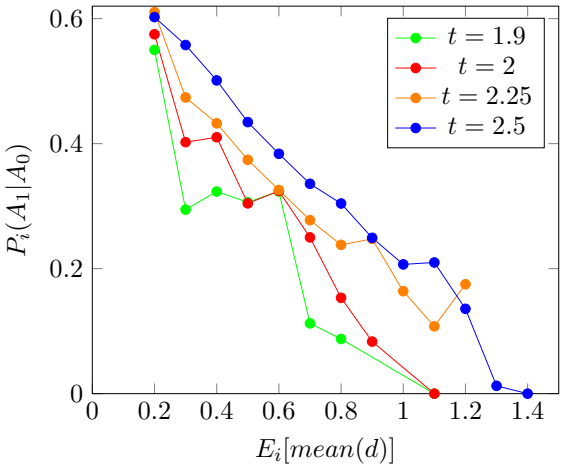


Figure 6.12: Evolution of $P_i(A_1|A_0)$ in function of the mean distance between examples of individual i.

The less trustworthy a person is, the lower his $P_i(A_1|A_0)$ is. As was observed before, adding new examples of unreliable patients will decrease the FRR the most. However, this is those examples which will increase the FAR the most as well (remember equation 6.4). This fact reduces the interest of keeping more examples of unreliable patients. For instance, adding second examples of the most reliable patients will increase the FAR by only $(1 - 0.6) * FAR' = 0.4 * FAR'$ for the range of thresholds that are plotted, as $P_i(A_1|A_0) \approx 0.6$. On the contrary, the least trustworthy individuals will increase the FAR by the full FAR', as $P_i(A_1|A_0) \approx 0$. Nonetheless, the difference in utilities (figure 6.10) between such individuals is much more significant than this difference in increasing FAR. As a consequence, adding more examples of unreliable patients still remains a worthwhile rule.

It can also be observed that, for a same $E_i[mean(d)]$, $P_i(A_1|A_0)$ decreases with t, which justifies the decrease of the global $P(A_1|A_0)$ observed in figure 6.2.

Chapter 7

Incorporating the hand data

In this chapter, three different ways to combine the hand recognition system and the nearest neighbour algorithm will be analyzed. The identification performance of the default metric has already been strongly enhanced by learning high margin metrics in chapter 5. In this chapter, this performance will be increased even further by combining the nearest neighbour search on personal data and the hand recognition system. As a consequence, ridiculous population sizes will be addressable. The reader should not expect any real life system to be able to tackle such sizes, as the number of attributes and the noise intensity among the data were arbitrarily chosen and may not reflect the reality.

7.1 Standard attribute

The first way to combine both systems would be to consider the hand data as another attribute and to define a distance function between hands that verifies the triangular inequality. However, the data representing the hand (veins network) is non-disclosed. The only accessible output from the hand recognition system is whether or not two hands have matched. As the hand data is not available, the effect of such an attribute will be mimicked. This can be done by extending the distance vector (see section 1.5) between two examples with one binary distance. This distance is equal to 0 if the hands of the two patients have matched, and 1 if they have not. The matching is assumed purely random. In particular, this extra distance function does not satisfy the triangular inequality. The function taking two examples and giving the binary distance between their hands is noted $d_h : (E, E) \to \{0, 1\}$. In the future, Savics will have access to an identification confidence between two hands. This score function should replace d_h as it will provide a better precision.

Using the data received from Savics, the FRR of the hand recognition system has been estimated to 0.003. FAR' is assumed to be equal to 10^{-7}, as claimed by Fujitsu. Not enough data was provided to estimate it reliably. As a consequence, d_h will be equal to 1 for 0.3% of pairs of examples of the same patient and 0 for 10^{-5}% of pairs of examples of different patients.

Using the metric learning method developed in section 5.4 with $k_s = k_d = 20$ (in order to maximize the margin), an extended Mahalanobis metric (M is now of size 16) was learned. The results with or without the additional feature on the same test set are compared on figure 7.1(a-b). Clearly, the added attribute separates a lot more the data. The optimal F-score with (dashed) or without (plain) the extra attribute can be seen on figure 7.1(c). As can be observed, the error rates have been significantly decreased.

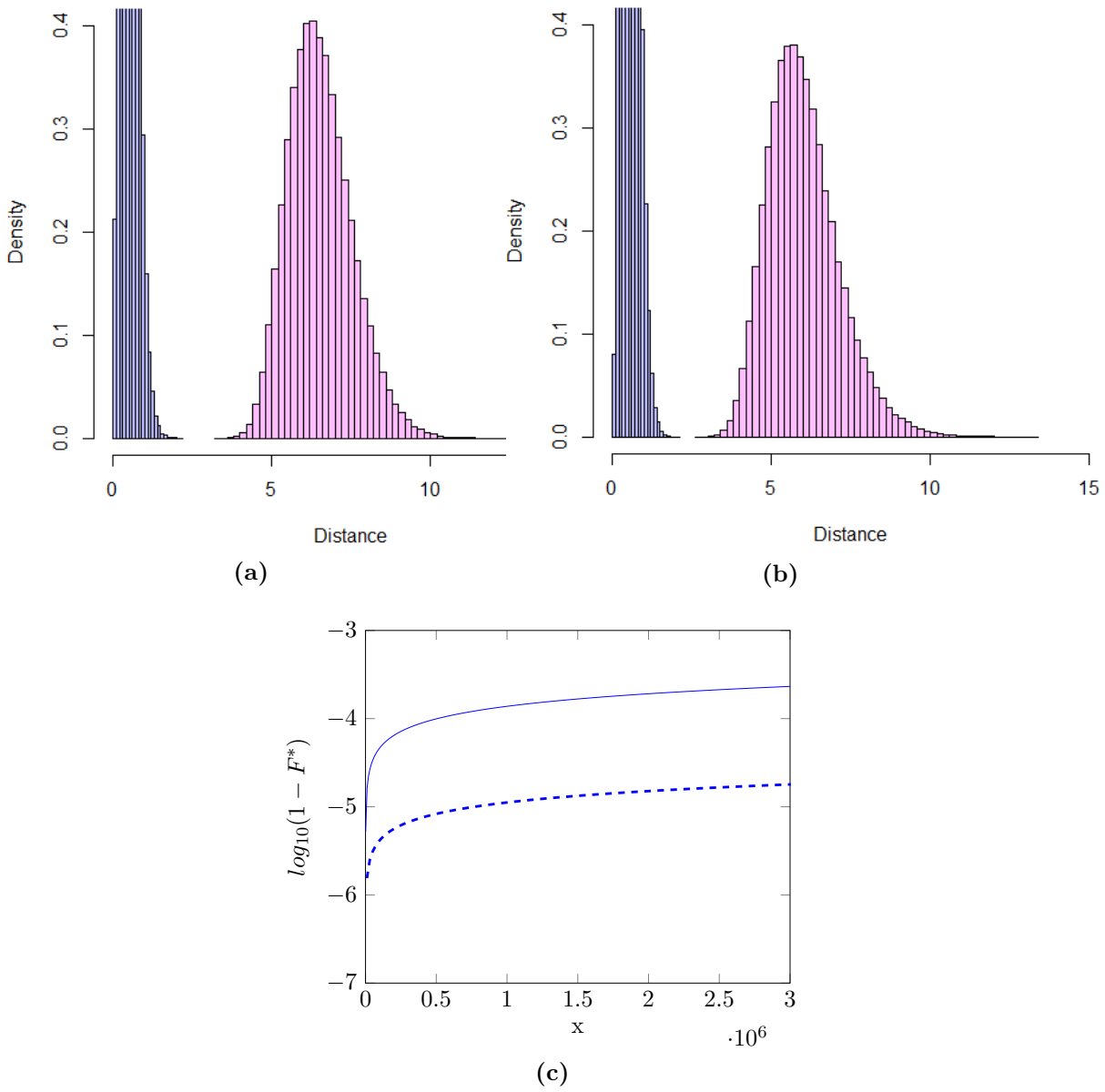


Figure 7.1: Comparison of the distance distributions with (a) and without (b) the extra hand attribute. Comparison between the optimal F-score with (dashed) and without (plain) the extra attribute (b). Both metrics are learned with $k_s = k_d = 20$.

As d_h does not satisfy the triangular inequality, one could think that the distance between every stored point and the query has to be computed. Fortunately, it is not true. As d_h is binary, a violation of the triangular inequality will only occur if two distances among $d_h(q, e), d_h(q, p)$ and $d_h(p, e)$ are equal to 0. As $d_h(e, e')$ will only be extremely rarely equal to 0 for examples of different individuals, the following algorithm will compute barely more distances than in the case where d_h would satisfy the triangular inequality.

Algorithm 5: Pruning with some triangular inequality violations

```
RangeQuery (q, t, B, E)
    inputs : query q, threshold t, prototype set B, set of stored examples E
    output: E_t ≜ {e ∈ E | d(q, e) ≤ t)}
    set ← ∅, E_t = E
    foreach b ∈ B do
        if d_h(q, p) == 1 then
            foreach e ∈ E_t do
                if d_h(p, e) == 1 & t < |d(q, p) − d(p, e)| then
                    E_t ← E_t/{e}
    return E_t
```

Nonetheless, as the hand recognition system is really slow, the time taken by one distance computation will greatly increase. This will make the assumption that the time is measured by distance computations even more true. The maximal number of such distance computations will have to be decreased in order to keep the same expected searching time.

7.2 Systems in series

The NNS algorithm could be used simply to filter as much people with different personal data than the test patient as possible. Then, the set of selected patients is given to the hand-recognition system which would make the final choice (or the other way around). The matching of the two systems are assumed independent. There is no reason why patients close to each other in personal data would have close hand pictures. In such a case,

$$FRR = FRR_{nns} + (1 - FRR_{nns}) * FRR_{hand} \approx FRR_{nns} + FRR_{hand} \tag{7.1}$$

A rejected patient is either rejected by the NNS, or accepted by the NNS and then rejected by the hand recognition system. The FRR is increased by FRR_{hand}.

A false acceptance occurs when both systems accept an incorrect patient.

$$FAR = x * FAR'_{nns} * FAR'_{hand} \approx 0 \tag{7.2}$$

The FAR is decreased by $FAR_{nns} * (1 - FAR_{hand}) \approx FAR_{nns}$.

The disadvantage of this structure is a too heavy FRR. A lying patient will succeed every time in concealing his identity. The advantage is that it will be impossible to pretend to be someone else. Furthermore, if the slowest system is put into the second place, the overall computational efficiency will be dictated by the fastest system. As a consequence, the NNS system should be put in the first place.

As the hand recognition system is a black box with fixed error rate, only the threshold of the nearest neighbour search can be modified, in order to optimize the global F-score. As

$$\frac{\partial FAR}{\partial t} = FAR'_{hand} * \frac{\partial FAR_{nns}}{\partial t}, \frac{\partial FRR}{\partial t} = \frac{\partial FRR_{nns}}{\partial t}$$

, equation 3.7 gives

$$FAR'_{hand} \frac{\partial FAR_{nns}}{\partial t} = -\beta^2 \frac{\partial FRR_{nns}}{\partial t}$$

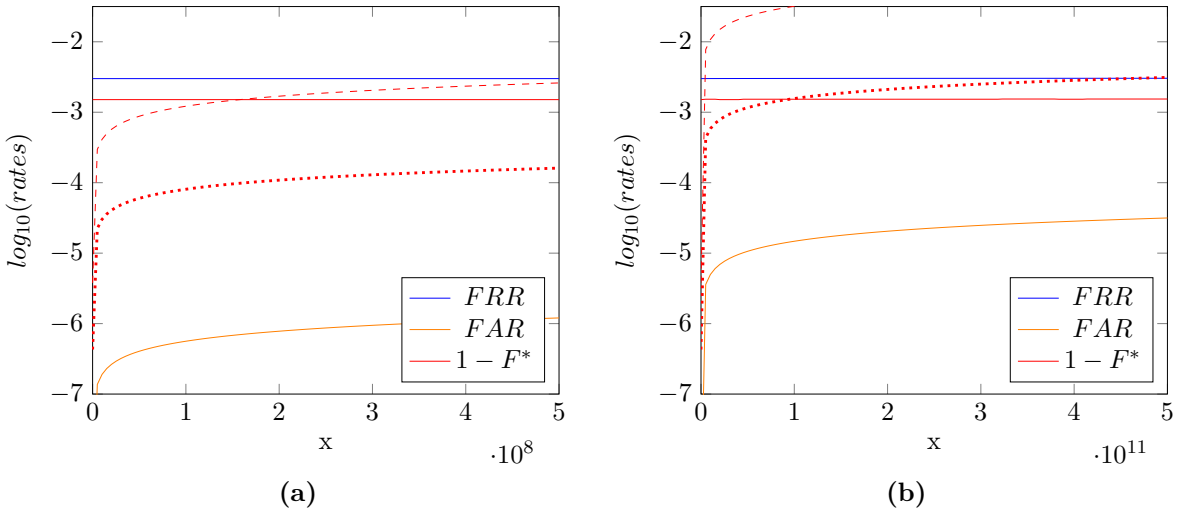


Figure 7.2: Evolution of the FRR, FAR and F-score of the system composed of the NNS algorithm and the hand recognition system in series (plain). Comparison with the nearest neighbour F-score when it is alone, with (dotted) or without the extra attribute (dashed).

which in turn implies

$$t^* = \frac{1}{a_{far} + a_{frr}} ln(\frac{\beta^2 c_{frr} a_{frr}}{FAR'_{hand} x a_{far} c_{far}}) = t^*_{nns} - \frac{ln(FAR'_{hand})}{a_{far} + a_{frr}} \tag{7.3}$$

Assuming $FAR'_{hand} \approx 10^{-7}$, $t^* = t^*_{nns} + 0.835$. With such an increase of the threshold, $FRR_{nns} \approx 0 \Rightarrow FRR \approx FRR_{hand}$. From equation 7.2, $FAR \approx 0$. That implies

$$F \approx 1 - (\frac{\beta^2}{\beta^2 + 1}) FRR_{hand} \tag{7.4}$$

The F-score is approximately independent of the size of the database, as can be seen on figure 7.2. As can be observed, for low populations, the FRR given by equation 7.1 is too big, because of the fixed value of FRR_{hand}. Nonetheless, as the size of the population increases, the in-series configuration outperforms first the NNS algorithm alone (dashed line), and then even outperforms the NNS algorithm with the additional hand attribute (dotted line). Eventually, for extremely large population sizes, the F-score will start to increase.

Unfortunately, the optimal threshold has been strongly increased, which will degrade a lot the speed of the search. If a time constraint is present, FRR_{nns} will be dictated by the maximum threshold allowed by this time constraint. In such a case, using equation 7.1, $FRR \approx FRR_{nns,max} + FRR_{hand} > FRR_{nns,max}$. As the F-score is dominated by the FRR when the time constraint becomes limiting, the F-score of the combined systems will then be strictly lower than the F-score of the nearest neighbour algorithm alone.

7.3 Systems in parallel

Another choice would be to put both systems in parallel. In other words, patients are accepted if they are accepted by at least one of the systems. In such a case,

$$FAR = FAR_{nns} + FAR_{hand} - FAR_{nns} * FAR_{hand} \approx FAR_{nns} + FAR_{hand} \tag{7.5}$$

and

$$FRR = FRR_{nns} * FRR_{hand} \approx 0 \tag{7.6}$$

In this case, it is the FAR that is difficult to control.

The disadvantage of this structure is that patients can pretend to be someone else, based on his personal data, and be accepted. Moreover, the computational efficiency will be dictated by the slowest system, as both have to run on the full population. The advantage is that the system will rarely be fooled by a patient writing different personal data every time, if it does not correspond to another real patient.

This time we have

$$\frac{\partial FAR}{\partial t} = \frac{\partial FAR_{nns}}{\partial t}, \frac{\partial FRR}{\partial t} = FRR_{hand} * \frac{\partial FRR_{nns}}{\partial t}$$

At the optimum,

$$\frac{\partial FAR_{nns}}{\partial t} = -\beta^2 FRR_{hand} * \frac{\partial FRR_{nns}}{\partial t}$$

which implies

$$t^* = t^*_{nns} + \frac{ln(FRR_{hand})}{a_{far} + a_{frr}} \tag{7.7}$$

For $FRR_{hand} = 0.003$, $t^* = t^*_{nns} - 0.301$. This time, the nearest neighbour search will be made substantially faster by reducing the threshold. However, as the slowest system is the hand recognition system, the overall computational efficiency will not be improved by this reduction of the threshold. From equation 7.6, $FRR \approx 0$. As the threshold is greatly reduced, $FAR_{nns} \approx 0$. From equation 7.5, this implies $FAR \approx FAR_{hand}$. As a consequence,

$$F \approx 1 - \frac{FAR_{hand}}{\beta^2 + 1} = 1 - \frac{x * FAR'_{hand}}{\beta^2 + 1} \tag{7.8}$$

In this case, the F-score decreases linearly when the population size increases. As can be seen on figure 7.3, the F-score of the combined system is a lot lower than the one of the nearest neighbour system on its own. This is because the false acceptance rate of the hand recognition system is too big. Furthermore, this configuration is impracticable for any time constraint, as the hand recognition system is supposed to be incredibly slow.

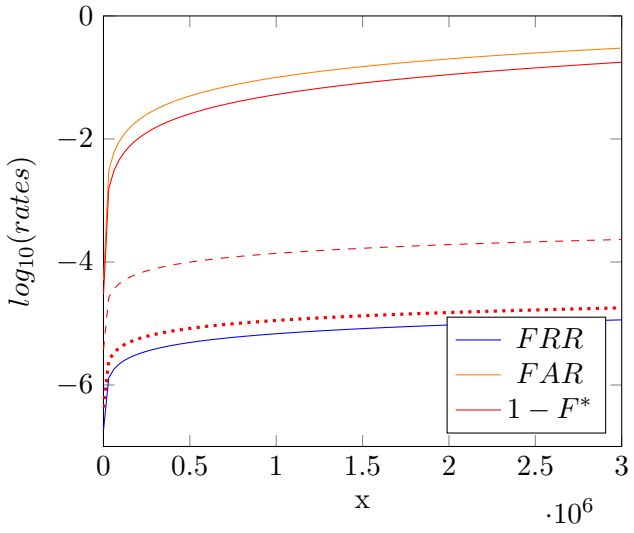


Figure 7.3: Evolution of the FRR, FAR and F-score composed of the NNS algorithm and the hand recognition systems in parallel. Comparison with the nearest neighbour F-score when it is alone, with (dotted) or without the extra attribute (dashed).

7.4 Twofold NNS

For the in-series configuration, once a patient is rejected by one of the system, the other cannot override this decision. The same is true for an acceptance for the in-parallel configuration. In this section, we assume that the hand recognition system is placed first. Its output will be the set of individuals that have matched the test hand. The NNS algorithm will then take this output as additional input and make the final decision. More precisely, it will run on this selected set of patients using a given threshold, t_2, and run on the rest of the population with another threshold, t_1.

A rejected patient is either rejected by the hand recognition system, then rejected by the NNS with the threshold t_1 or accepted by the hand recognition system and rejected by the NNS with the threshold t_2. Concerning the FAR, $x * FAR'_h$ patients will be accepted on average by the hand recognition system. An accepted patient is either from these patients and accepted by the NNS, with t_2, or is from the non-accepted patients and accepted by the NNS with t_1. As a consequence, equation 7.9 has to be minimized in order to maximize the F-score.

$$1 - F = \alpha * FRR + (1 - \alpha) * FAR = \alpha(FRR(t_1) * FRR_h + FRR(t_2) * (1 - FRR_h)) +$$
$$(1 - \alpha) * (x * (1 - FAR'_h)FAR'(t_1) + x * FAR'_h * FAR'(t_2)) \tag{7.9}$$

Equation 7.9 can trivially be rearranged into the sum of two terms, one of them depending on t_1 and the other on t_2. The two thresholds can then be optimized independently. The obtained F-score as well as the optimal thresholds are displayed on figure 7.4. As can be seen, this method outperforms all other techniques. It is equivalent to the in-series configuration for $FAR'(t_1) = 0$ and $FRR(t_1) = 1$ ($t_1 \to 0$) and to the in-parallel configuration for $FAR'(t_2) = 1$ and $FRR(t_2) = 0$ ($t_2 \to \infty$). As the twofold NNS technique explicitly minimize the F-score and is a generalization of both of these techniques, its performance is strictly better, for all

population sizes.[1].

Unfortunately, the twofold NNS technique requires the hand recognition system to be run on the full population. This makes it impracticable.

To summarize, the standard attribute method provides strong performances, but nos as strong as what could be expected. This is probably because the metric learning does not minimize explicitly the F-score. Furthermore, as the hand and personal data are supposed to be independent, the Mahalanobis metric cannot take advantage of any correlation between the two. Finally, the binary nature of the extra attribute is also a limiting factor. Indeed, this latter is so rarely equal to 0 between examples of different individuals than the training set did not contain a single pair of such examples. Nonetheless, as pruning is still achievable, the number of requests to the hand recognition system can be controlled using the metric learning techniques developed in chapter 5. The in-series configuration will provide poor performance for small population sizes but a very good F-score for large populations. It is extremely dependent on FRR_{hand}, however. The number of hand distances computations will be kept very small if the hand recognition system is put in the second place. The in-parallel configuration provides awful performance and is impracticable, as the hand recognition system has to be run on the whole population. The twofold NNS technique provides the best performance for all population sizes, but is also impracticable for the same reason. As a consequence, the standard attribute and the in-series techniques might be the most interesting ones. The best one will mostly depend on the time constraint and FRR_{hand}.

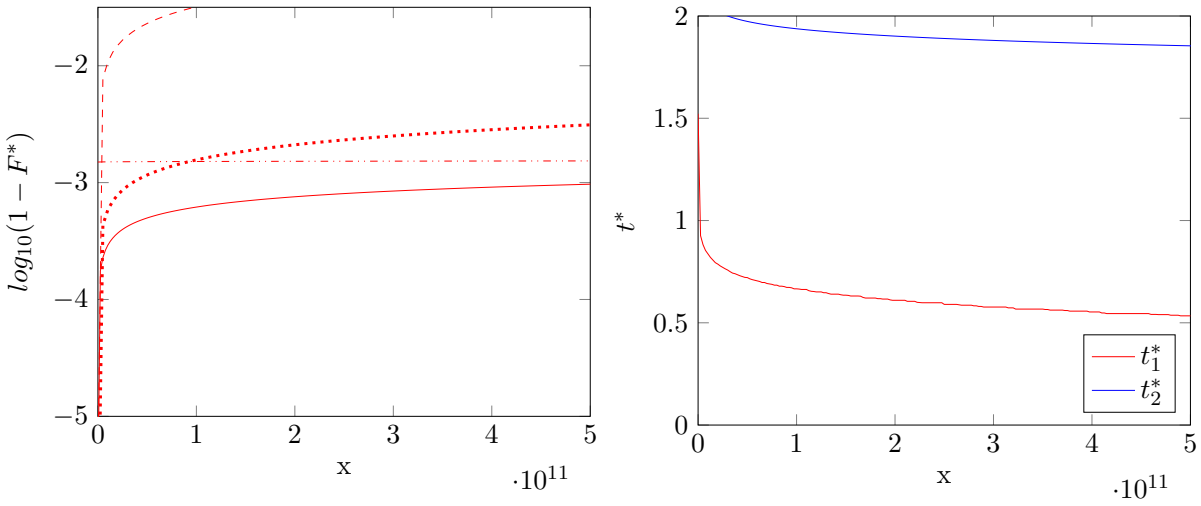


Figure 7.4: Comparison of the F-score obtained with the NNS algorithm only (dashed), with the in-series configuration (dashed dotted), with extended distance function (dotted) and with the twofold NNS method (plain) (a). Evolution of $t_1(x)$ and $t_2(x)$ (b).

[1]Technically, the in-series configuration will eventually outperforms the twofold NNS because $FAR'(0) > 0$, but only for extremely large population sizes, normally out of range.

Conclusion

The task of person re-identification consists in identifying a person inside a population to which corresponds the test data. While significant work has been done on the binary task which aims at predicting whether or not two examples belong to the same person, the full person re-identification problem was left aside. In particular, the key point of searching the test individual among the population has not been addressed yet. In this thesis, this deficiency has been fixed. This has led to an adaptive decision rule as well as the formulation of time and spaces constraints which, in turn, created the need to learn computationally efficient metrics.

In this thesis, the person re-identification problem has been tackled by a simple threshold nearest neighbour algorithm. The label of the closest example to the test example is predicted if both examples are closer to each other than a given distance, called decision threshold. Otherwise, the test individual is unrecognized. Modifying this decision threshold is an efficient way to balance between the FRR and the FAR. Increasing the threshold will decrease the FRR but increase the FAR and vice versa. The FCR has been shown to be negligible compared to the two other error rates.

For that reason, the optimal threshold can be found by maximizing the F-score of the two other error rates only. The FRR does not depend directly on the population size. However, for small error rates, the FAR is directly proportional to it. As a consequence, when the population size increases, the optimal threshold will decrease to counter-balance the rise of the FAR. The decision rule must be dependent on the population size. Nonetheless, despite the modification of the threshold, the optimal F-score still decreases with the population size. This diminution has been shown to be sub-linear, which allows to deal with large populations. The β parameter of the F-score is an efficient way to put some preference towards the FRR or the FAR. A small β will give more importance to the FAR, decreasing the optimal threshold. On the contrary, a large β will favor a small FRR, with a larger optimal threshold.

To predict the label of the closest example, the latter has first to be found. As the distances between examples are heavy to compute[2], comparing the test example to every stored example will quickly become intractable when the known population grows. To prevent that, different searching approaches, aiming at reducing the decision time, were studied. These approaches make use of the triangular inequality and so-called prototype examples, to prune the search space as much as possible. Significant work has already been done in the literature on such techniques. However, they generally aim at finding the closest example instead of all examples inside a range. For that reason, as well as the specificity of our search space, such techniques had to be adapted. An algorithm was then proposed, which has a logarithmic complexity in terms of distance computations as long as the fraction of examples filtered by each prototype tends to a constant. In order to reduce the overall time complexity of this algorithm to sub-

[2]For each distance computation, several levenshtein distances have to be computed.

linear, different tree structures, such as BK and vantage trees, were analyzed. For both trees, the computational performance has been shown to rely heavily on the prototypes used. As a consequence, fixed-queries trees were proposed to allow for an optimized prototype selection. Different selection rules were then studied and compared.

Armed with these results, the effect of the time constraint has been assessed. As the filtering is more efficient with small thresholds, the time constraint puts an upper bound on the threshold. If the time constraint becomes limiting, the threshold will then be smaller than the optimal one. As a consequence, the FAR will be smaller but the FRR will be larger, which will degrade the F-score by considerable amounts. Clearly, a time constraint is a bigger problem for a large β than for a small β, as the optimal thresholds for a large β are higher. Thus, computationally speaking, it is easier to give more importance to the FAR. An eventual space constraint will limit the number of usable prototypes in function of the population size. The optimal number of prototypes increases with the number of individuals, while its maximal value decreases with it. The effect of the space constraint will then worsen when the population gets bigger. However, a space constraint is not a problem on its own, it only exacerbates the impact of the time constraint.

Then, in order to moderate the negative effects of the time and space constraints, a metric learning scheme balancing between identification performance and decision time was proposed. The compromise was done by casting a convex optimization problem, that can be solved using a simple gradient descent algorithm. High margin metrics have been shown to reduce significantly the error rates, compared to the default metric. However, their lack of efficient filtering makes them very vulnerable to a time constraint. On the contrary, metrics with a high pruning capacity will have higher optimal error rates, but will be able to use higher maximal thresholds. As a consequence, they will provide a better overall identification performance for large populations. The choice of the right metric depends on the time and space constraints, as well as the target population size. A tight space constraint will favour pruning efficient metrics, as they tend to use less prototypes.

The effect of keeping more than one example per person in the database was also studied. Doing so decreases the FRR but increases the FAR. This will cause the decrease of the optimal threshold. While the decrease in FRR can be accurately measured, the effect on the FAR cannot be determined precisely. As a consequence, a worst case assumption was made, which allowed the derivation of lower bounds of the F-score. Nonetheless, a significant increase of the F-score has been observed. Concerning the time constraint, adding several examples per patient has two opposed effects. On the one hand, the rise of the population size makes the time constraint more limiting. On the other hand, the decrease of the optimal threshold will improve the filtering, counter-balancing the increased number of examples. On our data, it was shown that the time constraint did not prevent the addition of several examples. On the contrary, a space constraint will much more affect the error rates when a high number of examples is kept.

The noisier the examples of a patient are, the more useful it is to keep several of them in the database. In this regard, unreliable patients should have more examples than trustworthy ones. Allowing a variable number of stored examples per individual significantly improved the F-score. Although this enhancement relies on a precise measure of the FRR for each patient, which will not be available in practice, it is clear that examples of noisy patients should be added in priority.

Finally, different ways to incorporate the hand data in the algorithm were studied. Theoret-

ically, the identification performance can be reduced tremendously by combining both systems. However, the slow execution of the hand recognition system strongly limits this theoretical enhancement. Nevertheless, two approaches can take advantage of the additional information provided by the hand recognition system without suffering too much from it's slowness. First, the feature space can be extended with one additional attribute. The number of requests to the hand recognition system can then be controlled, using the proposed metric learning scheme. While this technique has not provided the best results, it has a lot of potential to unlock. Indeed, using the identification score between hands instead of the binary distance is expected to further reduce the error rates significantly. Nonetheless, if the score function happens to not satisfy the triangular inequality, the pruning mechanism will no longer be usable, which will most likely make this solution impracticable.

The second way to take advantage of the hand data consists in putting the NNS and the hand recognition system in series. In other words, only patients that have been accepted by both systems are accepted by their combination. The F-score of this method is approximately constant with respect to the population size and is equal to the FRR of the hand recognition system. This implies a poor identification performance for small populations but a very competitive one for large populations. Furthermore, placing the NNS in the first place is an efficient way to reduce the number of distance computations between hands.

The contributions of this thesis are multiple. First, it highlights the need of an adaptive decision rule. It also proposes an efficient search algorithm, that minimizes the expected number of distance computations. Most importantly, it brings to light the need of using computationally efficient metrics and proposes a metric learning scheme that can balance between this efficiency and the identification performance. This method showed a drastic improvement of the search efficiency on partially artificial data compared to standard margin maximization metrics. When hard time and space constraints have to be met, this leads to a considerable decrease of the error rates. Problems that are unsolvable using standard metrics can be tackled when some care is given to the computational efficiency in the metric learning process. While this work focuses on person re-identification, other machine learning problems will most likely need the learning of efficient metrics in the future.

Appendices

Appendix A

Comparison between normalized and standard levenshtein distance

In this appendix, the comparison between the standard and the normalized levenshtein distance is made.

While the standard distance clearly puts a bias to reject people with longer string attributes, the normalized form will reject more people with shorter names. This is not trivial to see and we explain it in a restricted domain. Suppose individuals are composed by only one single string attribute. A couple of examples is rejected if their normalized distance to each other exceeds 0.5. Also suppose the only error possible is to substitute one letter by another and that every letter has a probability of 0.1 to be substituted. If the person has a single character name such as "a", one tenth of the times a new example is generated, "a" will be swapped to another letter. This will cause $d = 0.66 > 0.5$. His probability to be rejected is then $1/10$. Now if a person has a long name, such as "christophe" with 10 letters, he will have to make at least 7 errors in order to be rejected. This will happen tremendously less often. The key point here is that the reject distance must be big because the error rates must be small.

On figure A.1 can seen the distance distributions measured on a small training set using metrics learned with the method developed in section 5.4, with $k_s = 20$ and $k_s = 2.5$, for the levenshtein string metric (A.1(a)) and it's normalized form (A.1(b)). Only the six string attributes were used.

As can be seen, the data separation is more or less the same in both cases. Nonetheless, the normalized levenshtein distance contracts a lot more the red curve. Nearly no pruning would be possible (assuming the other attributes are not used). That is the reason why the standard form was preferred. Overall, the standard levenshtein distance obtained the value (to be minimized) 126.853 for the objective function. The normalized levenshtein distance only got 252.03.

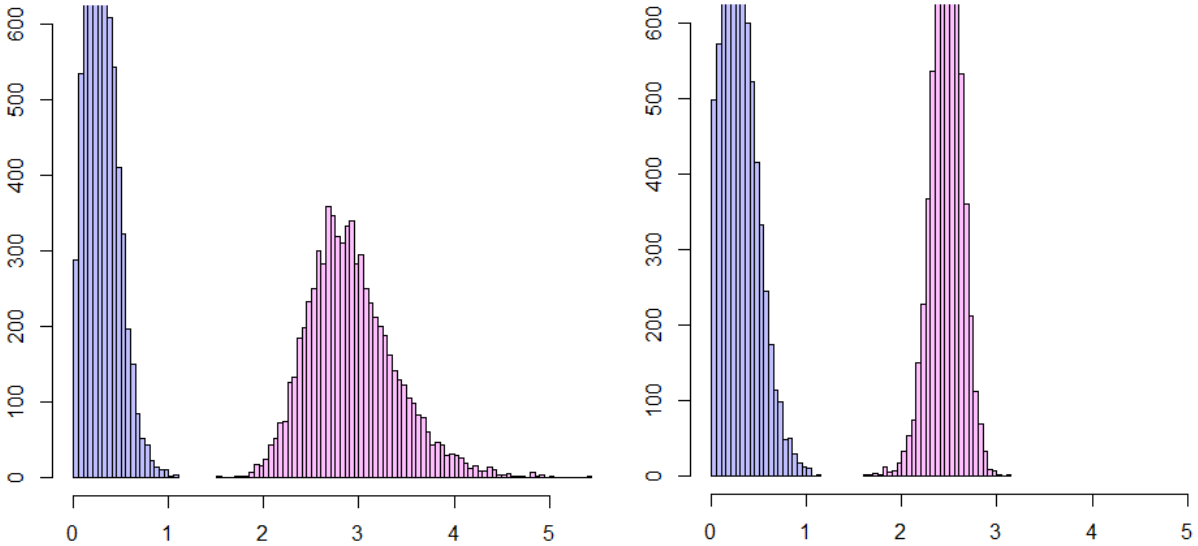


Figure A.1: Distance distribution obtained on a small train set with (b) or without (a) normalized levenshtein distance. Only the 6 strings attributes are used. The metrics are learned by the method developed in section 5.4, with k_s and $k_d = 2.5$.

Appendix B

Pruning mechanism of LTAESA

In this appendix, the tree representation of the database as well as the pruning mechanism used by LTAESA[18] will be studied in more details.

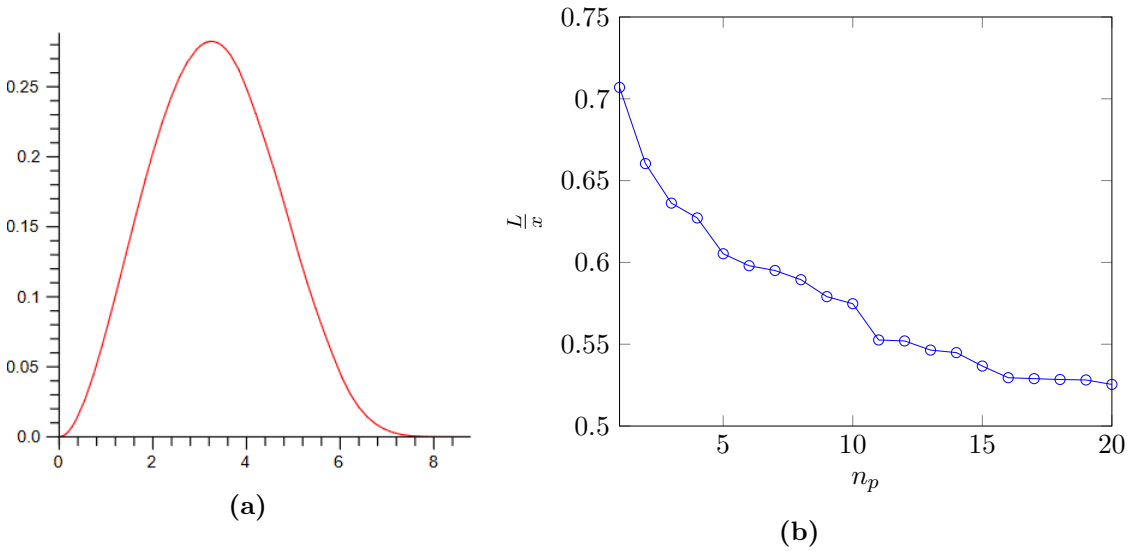
To each node t of the tree is associated a subset of the database S_t. S_t contains only one data point iff t is a leaf. Otherwise, $S_t = S_{t_l} \cup S_{t_r}$ where t_l and t_r are the left and right child of t, respectively. Each node also stores a data point m_t and a radius r_t. The radius is the maximal distance between m_t and every data point in S_t: $r_t = max_{p \in S_t} d(m_t, p)$. The left child of a node has the same representative: $m_{t_l} = m_t$. On the contrary, the right child's representative is chosen following the rule $m_{t_r} = argmax_{p \in S_t} d(p, m_t)$. Finally, $S_{t_l} = \{p \in S_t : d(p, m_{t_r}) < d(p, m_{t_l})\}$. S_{t_l} contains the remaining examples $S - S_{t_r}$.

Due to the triangular inequality, if in node t, the following relation holds $thr + r_t < d(x, m_t)$ with thr the threshold value and x the query, node t and all it's children can safely be pruned from the search. However, we would like to avoid computing $d(x, m_t)$. From the triangular inequality, it is also true that $\forall b \in B, d(x, m_t) \leq |d(x, b) - d(b, m_t)|$ where B is the set of prototypes. This relation implies $d(x, m_t) \leq max_{b \in B} \|d(x, b) - d(b, m_t)| = g(x, m_t)$. This means that if $thr + r_t < g(x, m_t)$, t can be pruned without computing a single distance.

When given a query point, TLAESA first computes the distances between this point and all prototypes. Then, it traverses the tree and computes distances only when it reaches leaves of the tree. There may be more prototypes than level in the search tree. When a leaf is reached, $g(x, m_t)$ is evaluated. The distance between the data point and the query will be computed only if $g(x, m_t) \leq thr$.

The claimed average constant complexity was measured on hypercubes, using the euclidian distance. As an illustration, [2] has computed the distance distribution between random points in a box of sides 4,5 and 6, as can be seen on figure B.1(a). TLAESA will work great on this space because the distance between points can be close to zero. This means that, in the search tree, the radius of nodes deep down in the tree will tend to 0. The pruning will then be maximized. However, consider our space illustrated in figure 1.1(b). In this case, r_t will never be close to zero. As a consequence, the pruning condition $thr + r_t < g(x, m_t)$ will rarely be met.

The fraction of leaf nodes that are explored (noted L) for up to 20 prototypes is depicted in figure B.1(b). This amount is much bigger than the one obtained with vantage trees, measured in figure 2.14 (it was about 0.08 for the same tree size). Even if L will decrease for a bigger number of prototypes, as $g(x, m_t)$ will give a better estimation of $d(x, m_t)$, the pruning efficiency of vantage trees will never be caught.



(a) **(b)**

Figure B.1: Distance distribution between random points inside a box [2] (a). Fraction of explored leaf nodes of the tree (b). This has been evaluated on a tree containing 1000 examples.

Appendix C

Metrics comparison without space constraint

On figure C.1(a) can be seen the maximum thresholds allowed for the metrics learned in section 5.4 by a time constraint with $\eta_d = 5000$. The number of used prototypes is unbounded. The difference in thresholds has decreased from figure 5.7(b). Moreover, the metrics learned with $k_d = 2.5$ and $k_d = 1$ have both approximately the same maximum thresholds. The effects of the time constraint on the F-score can be seen on figure C.1. The main difference is that $k_d = 1$ is the second worst metric, for a population size of 3M. The difference between all metrics are also slightly decreased.

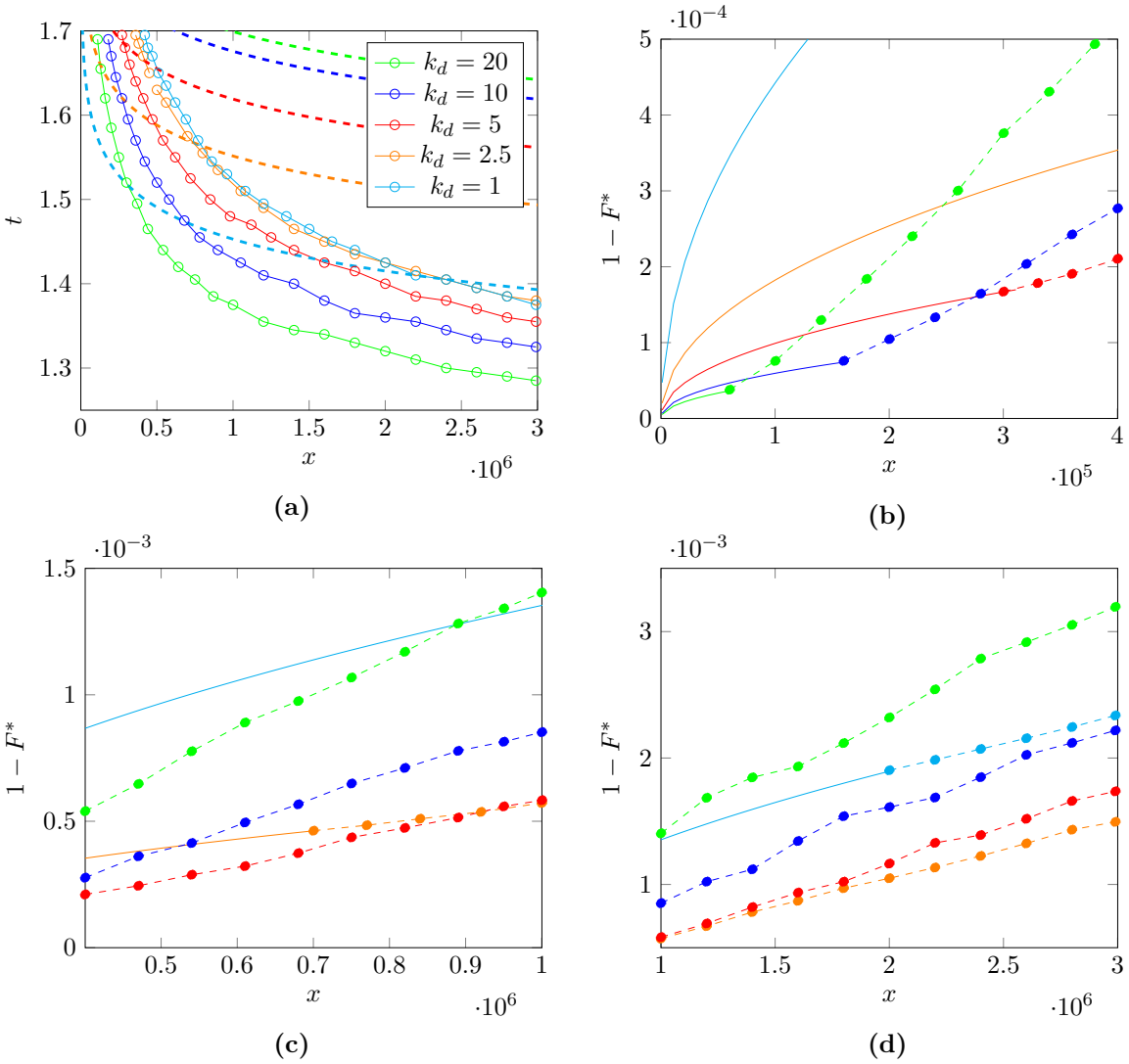


Figure C.1: Optimal (dashed) and maximal thresholds (plain) for the different metrics, given a time constraint with $\eta_d = 5000$ and no bound on the number of prototypes (a). Corresponding F-score for population sizes up to 3M individuals (b-c-d).

Bibliography

[1] Eric P. Xing, Michael I. Jordan, Stuart J Russell, and Andrew Y. Ng. Distance metric learning with application to clustering with side-information. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 521–528. MIT Press, 2003.

[2] Johan Philip. *The probability distribution of the distance between two random points in a box*. KTH mathematics, Royal Institute of Technology, 2007.

[3] Krzysztof Dembczy nski. Extreme classification: Machine learning with millions of labels. [http://www.cs.put.poznan.pl/kdembczynski/pdf/extreme-classification-uam-2017.pdf], May 24, 2017.

[4] Zhen Li, Shiyu Chang, Feng Liang, Thomas S Huang, Liangliang Cao, and John R Smith. Learning locally-adaptive decision functions for person verification. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 3610–3617. IEEE, 2013.

[5] Fei Xiong, Mengran Gou, Octavia Camps, and Mario Sznaier. Person re-identification using kernel-based metric learning methods. In *European conference on computer vision*, pages 1–16. Springer, 2014.

[6] Alexis Mignon and Frédéric Jurie. Pcca: A new approach for distance learning from sparse pairwise constraints. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2666–2672. IEEE, 2012.

[7] Kilian Q Weinberger, John Blitzer, and Lawrence K Saul. Distance metric learning for large margin nearest neighbor classification. In *Advances in neural information processing systems*, pages 1473–1480, 2006.

[8] Jacob Goldberger, Geoffrey E Hinton, Sam T Roweis, and Ruslan R Salakhutdinov. Neighbourhood components analysis. In *Advances in neural information processing systems*, pages 513–520, 2005.

[9] P. C. Mahalanobis. On the generalised distance in statistics. In *Proceedings National Institute of Science, India*, volume 2, pages 49–55, April 1936.

[10] UCL répertoire. [`https://uclouvain.be/fr/repertoires#`].

[11] Annuaire de l'ULB. [`https://www.ulb.ac.be/outils/annuaire/1.html`].

[12] M. Lichman. UCI machine learning repository. [`https://archive.ics.uci.edu/ml/datasets/adult`], 2013.

[13] Fred J. Damerau. A technique for computer detection and correction of spelling errors. *Commun. ACM*, 7(3):171–176, March 1964.

[14] Aurélien Bellet, Amaury Habrard, and Marc Sebban. A survey on metric learning for feature vectors and structured data. *arXiv preprint arXiv:1306.6709*, 2013.

[15] Felix Naumann. Similarity measures. IT Systems Engineering, Universität Potsdam, 2013.

[16] Li Yujian and Liu Bo. A normalized levenshtein distance metric. 29:1091–5, 07 2007.

[17] Marina Sokolova, Nathalie Japkowicz, and Stan Szpakowicz. Beyond accuracy, f-score and roc: a family of discriminant measures for performance evaluation. In *Australasian joint conference on artificial intelligence*, pages 1015–1021. Springer, 2006.

[18] Luisa Micó, Jose Oncina, and Rafael C Carrasco. A fast branch & bound nearest neighbour classifier in metric spaces. *Pattern Recognition Letters*, 17(7):731–739, 1996.

[19] Walter A. Burkhard and Robert M. Keller. Some approaches to best-match file searching. *Communications of the ACM*, 16(4):230–236, 1973.

[20] Ricardo Baeza-Yates and Gonzalo Navarro. Fast approximate string matching in a dictionary. In *String Processing and Information Retrieval: A South American Symposium, 1998. Proceedings*, pages 14–22. IEEE, 1998.

[21] Ricardo Baeza-Yates, Walter Cunto, Udi Manber, and Sun Wu. Proximity matching using fixed-queries trees. In *Annual Symposium on Combinatorial Pattern Matching*, pages 198–212. Springer, 1994.

[22] Peter N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '93, pages 311–321, Philadelphia, PA, USA, 1993. Society for Industrial and Applied Mathematics.

[23] Wolframalpha. [http://www.wolframalpha.com/].

[24] Enrique Vidal Ruiz. An algorithm for finding nearest neighbours in (approximately) constant average time. *Pattern Recognition Letters*, 4(3):145–157, 1986.

[25] Luisa Micó, José Oncina, and Enrique Vidal. An algorithm for finding nearest neighbours in constant average time with a linear space complexity. In *Pattern Recognition, 1992. Vol. II. Conference B: Pattern Recognition Methodology and Systems, Proceedings., 11th IAPR International Conference on*, pages 557–560. IEEE, 1992.

[26] Jason V Davis, Brian Kulis, Prateek Jain, Suvrit Sra, and Inderjit S Dhillon. Information-theoretic metric learning. In *Proceedings of the 24th international conference on Machine learning*, pages 209–216. ACM, 2007.

[27] Matthew Schultz and Thorsten Joachims. Learning a distance metric from relative comparisons. In *Advances in neural information processing systems*, pages 41–48, 2004.

[28] Amir Globerson and Sam T Roweis. Metric learning by collapsing classes. In *Advances in neural information processing systems*, pages 451–458, 2006.

[29] William E. Winkler. The state of record linkage and current research problems. Technical report, Statistical Research Division, U.S. Census Bureau, 1999.

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve **www.uclouvain.be/epl**