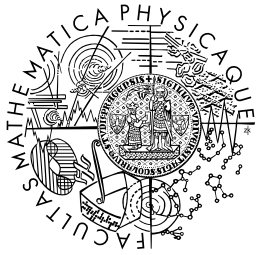

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

**BACHELOR THESIS**

Jan Hanousek

# Neighborhood components analysis and machine learning

Department of Probability and Mathematical Statistics

Supervisor of the bachelor thesis: prof. RNDr. Jaromír Antoch, CSc.
Study programme: Mathematics
Study branch: Stochastics

Prague 2018

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In Prague, 18.4.2018 Jan Hanousek

I would like to thank my advisor Prof. RNDr. Jaromír Antoch, CSc., for this topic, his comments and suggestions and for his constructive remarks.

I would also like to thank my father prof. RNDr. Jan Hanousek, CSc., DSc., for his comments and advices regarding stock market, technical analysis and for being a great father.

Title: Neighborhood components analysis and machine learning

Author: Jan Hanousek

Department: Department of Probability and Mathematical Statistics

Supervisor: prof. RNDr. Jaromír Antoch, CSc., Department of Probability and Mathematical Statistics

Abstract: In this thesis we focus on the NCA algorithm, which is a modification of k-nearest neighbors algorithm. Following a brief introduction into classification algorithms we overview KNN algorithm, its strengths and flaws and what lead to the creation of the NCA. Then we discuss two of the most widely used modifications of NCA called Fast NCA and Kernel (fast) NCA, which implements the so-called kernel trick. Integral part of this thesis is also a proposed algorithm based on KNN (/NCA) and Linear discriminant analysis titled TSKNN (/TSNCA), respectively. We conclude this thesis with a detailed study of two real life financial problems and compare all the algorithms introduced in this thesis based on the performance in these tasks.

Keywords: KNN, NCA, FNCA, kernel trick, TSKNN, TSNCA, classification.

Název práce: Analýza sousedních komponent a strojové učení

Autor: Jan Hanousek

Katedra: Katedra pravděpodobnosti a matematické statistiky

Vedoucí bakalářské práce: prof. RNDr. Jaromír Antoch, CSc., Katedra pravděpodobnosti a matematické statistiky

Abstrakt: V této práci se zabýváme algoritmem NCA, který je modifikací algoritmu k-nejbližších sousedů. Po krátkém úvodu do klasifikačních algoritmů se zaměříme na algoritmus KNN, jeho silné a slabé stránky a co vedlo ke vzniku NCA. Poté prodiskutujeme dvě nejvíce používané modifikace NCA nazvané Fast NCA a Kernel (fast) NCA, které používá takzvaný "Kernel trick". Důležitou částí práce je také nově navržený algoritmus založený na KNN (/NCA) a na LDA nazvaný TSKNN (/TSNCA). V závěru této práce použijeme všechny algoritmy popsané v této práci na vyhodnocení dvou finančních problémů a porovnáme jak byly jednotlivé algoritmy úspěšné.

Klíčová slova: KNN, NCA, FNCA, kernel trick, TSKNN, TSNCA, klasifikace.

# Contents

# 1. K-Nearest Neighbor(KNN)

## 1.1 Classifications based on nearest neighbors

Let us start with a several formal definitions from the field of classification.

**Definition 1.1**
We define the statistical model as $(\mathcal{S}, \mathcal{P})$, where $\mathcal{S}$ denotes the sample space and $\mathcal{P}$ refers to underlying probability distribution defined on $\mathcal{S}$.
In the context of the classification literature, we refer to $\mathcal{S}$ as a (training) data and $\mathcal{P}$ corresponds to the data generating process. △

**Definition 1.2**
Let $X \subseteq \mathbb{R}^p$ be a vector subspace and C $\subseteq$ {1,2,. . . , J} be a set of classes, where J is a constant. Then classifier is a function $f : X \to C$, such that

$$\forall x \in X \ f(x) = j, \ j \in C.$$

△

Basically, the desirable classification function provides the "best predictions possible". This means we seek to find the classification that maximizes the prediction accuracy.

**Definition 1.3**
We define the prediction accuracy PA as

$$PA = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions}.$$

Similarly we define the classification error $CE$ as

$$CE = \frac{Number\ of\ incorrect\ predictions}{Total\ number\ of\ predictions} = 1 - PA.$$

△

There are several different approaches and setups that differ from algorithm to algorithm, but in this thesis we will limit ourselves to k-nearest neighbor algorithm and algorithms based on it.

Nearest neighbor classification is one of the most basic and fundamental classification algorithms. It is a non-parametric method for pattern classification which was first introduced by Fix and Hodges Jr [1951]. It does not require any model specification or assumption about data generation process. We refer to this classification method as the k-nearest neighbor rule (KNN). Recently it has been modified by several authors primarily to address its computational issues (low speed and high memory requirements).

All classification algorithms based on KNN have a very similar structure. At the beginning we divide our data set into two parts. First part is used for training, where we set up parameters, solve various optimization problems, search for an optimal transformation, etc. All of these steps are used to tune up the procedure (its parameters, transformation matrix, etc.) to maximize the prediction

accuracy or minimize the classification error. Many authors describe this phase as "training". In the next step we use the settings discovered during the training session and we evaluate goodness-of-fit of our procedure on the remaining data. It means that we use the remaining data for verification of the tested procedure. There are several ways to evaluate how classifier performs in practice, but we will limit ourselves to cross-validation.

**Definition 1.4**
Consider a training set

$$S = \{(x_i, c_i), i = 1, 2, .., n\},$$

where $n$ is the number of observations, $x_i \in \mathbb{R}^p$ are feature vectors, $c_i \in \{1,2,\ldots, J\}$ are class labels (categories) and $J$ denotes the number of classes. Let k$\in \mathbb{N}$,$k \leq n$ and $S = \cup_{j=1}^{k} S_j$, where $S_i \cap S_j = \emptyset$ for $\forall i, j \in \mathbb{N}$, $i, j \leq k$, $i \neq j$. Then for one round of cross-validation we consider $S \setminus S_i$ as training set and $S_i$ as a validation set. $\triangle$

There are many types of cross-validation, but we will use two most popular methods that are also the most appropriate with respect to KNN-based algorithms.

**Definition 1.5**
K-fold cross validation is a type of cross-validation, where $\forall i \in \mathbb{N}$, $i \leq k$, we define $S \setminus S_i$ as training set and use $S_i$ as a validation set.
In Leave-one-out (LOO) cross-validation $\forall i \in \mathbb{N}$, $i \leq n$ we define $S \setminus \{x_i\}$ as training set and use $\{x_i\}$ as a validation set. $\triangle$

Since KNN is a non-parametric classification method – it does not make any assumption about the distribution of the data – therefore, we will omit general statistical model approach here. We will discuss it more in the Simulation part, when we plan to evaluate classification performance depending on the data generating process.

## 1.2 Setup and algorithm

Consider a classification problem with a training set

$$S = \{(x_i, c_i), i = 1, 2, .., n\},$$

where $n$ is the number of observations, $x_i \in \mathbb{R}^p$ are feature vectors, $c_i \in \{1,2,\ldots, J\}$are class labels (categories) and $J$ denotes the number of classes.
Our aim is to find a classifier

$$f\colon \mathbb{R}^p \to \{1,2,\ldots, J\},$$

which for each vector $x$ maximize its prediction accuracy. (Ideally we want to find a classifier $f$ which assigns every feature vector $x$ its true class $c_x$).

**Definition 1.6**
Let $d(x_i, x_j)$ denote the metric in $\mathbb{R}^p$ we use for measuring the distance between $x_i,\ x_j \in \mathbb{R}^p$. Let $x$ denote the feature vector (data point) we would like to classify and $d_i = d(x_i, x)$ distance of $x$ from each data point $x_i \in S$. Then the KNN classification predicts category of the feature vector $x$ equal to the majority class among its k-nearest neighbors with respect to distance $d_i = d(x_i, x)$. △

For two categories ($J$=2) it is recommended to choose $k$ as an odd number to avoid ties. For the general number of categories $J$ Definition 1.4 does not uniquely define the KNN prediction in case of ties. Hence we will use more precise rule for KNN prediction.

**Definition 1.7**
For any given $x \in \mathbb{R}^p$ we define the set of its $k$ nearest neighbors as

$$K_x = \{i \mid d_i <= d_{(k)}\} ,$$

where $d_{(k)}$ denotes k-th order statistics of $d_1,\ldots,d_N$ measuring the distances of $x_i \in S$ from $x$.
Let $n_i$ denote the number of the points belonging to the $i$-th class from $K_x$.
If $\mathrm{n}_i > \mathrm{n}_j\ \forall\ i \neq \mathrm{j}$ then KNN will predict category $x$ as $i$. In case of ties we assign $x$ into the class of the closest point with the maximal category, which means we assign $x$ into the class of the closest point. In the unlikely event of a tie after the last step we randomly pick one of the closest points and assign $x$ into the same class. For the sake of simplicity we can choose the point with the lowest index in our training set. △

For more detailed description of the KNN algorithm and its pseudo code we refer to Appendix A.1.

In general, all classification algorithms using nearest neighbor idea strongly depend on the used metric [for some discussion see, e.g. Weinberger and Saul, 2009]. The chosen metric is usually related to the specific method, but overall set of metrics ranges from the classical Euclidean to the Mahalanobis distance (for definitions see Appendix A.4).

Nonlinearity of KNN decision rule is accompanied by parameter parsimony – the method employs only a single integer parameter $k$, the number of the nearest neighbors used for the classification. Let us note that parameter $k$ is a proxy for the radius of the distance. As we mentioned above, usually it is recommended to choose $k$ as an odd number to avoid ties in case of two categories. Optimal choice of $k$ for KNN is not conclusive, however, most of the literature agrees on a rule of thumb to choose k approximately as $\sqrt{n}$, where $n$ denotes the number of data points [see e.g. Duda et al., 2012].

Users of the KNN method highly value that by enlarging the training set (i.e., increasing the amount of information used for the training) the quality of prediction is improved and errors in classifications are reduced and surprisingly, the over-training is not the issue here [see e.g. Goldberger et al., 2005].

## 1.3 Problems with KNN

Literature highlights two major problems associated with the KNN. First, because it uses the whole training data to determine the category class of a new observation, we must store the entire training data set. As it was mentioned before, more training data we have, the better classification is obtained. This means that if we want to improve the algorithm, we need to store more data, which causes the algorithm to be very computationally expensive, especially on larger data sets.

Second problem is associated with a proper definition of the "nearest" data point, more specifically what is the optimal distance metric to use. In general we can observe a tendency to favor Mahalabis distance over $L_2$ or $L_1$ norms [see Goldberger et al., 2005, Weinberger and Saul, 2009, among others ].

Therefore, many modifications of the KNN try to avoid data and computational demanding features of the KNN, by narrowing the amount of stored data, by using different metric, by employing optimal transformation matrix and/or by using kernel methods. We will discuss major shortcomings of the KNN on the widely used new methods called Neighborhood components analysis (NCA), Fast Neighborhood components analysis (FNCA) and Kernel FNCA.

# 2. Neighborhood components analysis (NCA)

## 2.1 How does NCA solves KNN problems

Neighborhood components analysis (NCA) was created to solve both shortcomings of the KNN mentioned above [Goldberger et al., 2005]. Since NCA is a modification of the KNN - the setup, main purpose and underlying statistical model are exactly the same as defined in the context of the KNN algorithm. Ideally, we would like to store limited information from our training set and choose the appropriate distance metric that maximizes the performance of the nearest neighbor classification. However, since we do not know the real distribution of the test data, researchers instead aim to optimize the classification algorithm by leave-one-out (LOO) method on the training data.

This means we first consider the classification of a single data point based on agreement of its k-nearest neighbors with respect to a given distance metric. This approach (LOO) is then used to determine the optimal radius of the distance which maximizes the classification accuracy [see e.g. Shao, 1993]. LOO is used to save computation time, since it only requires training and validating the classifier $n$ times, where $n$ is the number of data points in training set $S$.

## 2.2 Setup and algorithm

Consider a classification problem with a training set

$$S = \{(x_i, c_i), i = 1, 2, .., n\},$$

where $n$ is the number of observations, $x_i \in \mathbb{R}^p$ are feature vectors, $c_i \in \{1,2,\ldots, J\}$ are class labels (categories) and $J$ denotes the number of classes.
Our aim is to find a classifier $f : \mathbb{R}^p \to \{1, 2, \ldots, J\}$ which assigns every feature vector $x \in \mathbb{R}^p$ its true class $c_x$.
Let us first consider a classifier (denoted as $f$) such that

1. Assigns "reference point", denoted as Ref($x$), randomly from $S$ to every point $x$ (the probability distribution of selection is defined later in Equation 2.4).

2. Based on the label of reference point Ref($x$) it assigns a class label to $x \in \mathbb{R}^p$.

Notice that this classifier is similar to KNN with $k$=1.

In NCA the reference point is chosen randomly and every point in our training set $S$ has a probability of being chosen, with the probability increasing as the distance between the point and its reference decreasing.
In NCA algorithm we use the following distance function.

**Definition 2.1**
Let $x = (x_1, \ldots, x_p) \in \mathbb{R}^p$ and $x_i = (x_{i1}, \ldots, x_{ip}) \in S$. We define the distance function as

$$d_w(x_i, x) = \sum_{r=1}^{p} w_r^2 |x_{ir} - x_r| \ , \tag{2.1}$$

where $w_r$ are the feature weights- i.e. a number assigned to $x_r$ that reflects its relative importance in the classification process. △

Let us define kernel function

$$k(z) = \exp(-\frac{z}{\sigma}\ ). \tag{2.2}$$

This function is used to reduce memory cost.
Similarly as in Yang et al. [2012b] we assume that

$$P(Ref(x) = x_j \mid S) \propto k(d_w(x, x_j)) \ , \tag{2.3}$$

This means that probability that the reference point $x$ is equal to the data point $x_j$ is decreasing with the distance between $x$ and $x_j$ . It also means that the probability is proportional (denoted by the symbol $\propto$) to the kernel function of this distance.

Since any reference point x is chosen from the training set S, then the sum of P(Ref($x$)= $x_j|S$) must be equal to 1 for all $j$. Hence,

$$P(Ref(x) = x_j \mid S) = \frac{k(d_w(x, x_j))}{\sum_{j=1}^{n} k(d_w(x, x_j))} \ . \tag{2.4}$$

Now that we have defined the probability with which classifier $f$ selects reference point randomly let us consider the application of LOO cross-validation of this classifier. This means using the reduced data set $S^{-i}$ to predict the class label of $x_i$ .We do this to avoid assigning point $x_i$ as its own reference point.

We can now define the probability that a point $x_j$ will be selected as a reference point for $x_i$ as

$$p_{ij} = P\left(Ref(x) = x_j \mid S^{-i}\right) = \frac{k(d_w(x_i, x_j))}{\sum_{j=1, j\neq i}^{n} k(d_w(x_i, x_j))} \ . \tag{2.5}$$

Then we can calculate the probability of correct classification using LOO of the observation $x_i$

$$p_i = \sum_{j=1, j\neq i}^{n} P(Ref(x_i) = x_j | S^{-i}) \times \mathbb{I}[c_j = c_i] \ . \tag{2.6}$$

Then the leave-one-out probability of classifying every observation correctly can be expressed as:

$$F(w) = \sum_{i=1}^{n} p_i \ . \tag{2.7}$$

The goal of NCA is maximize the expected number of correctly classified observations, which is equivalent to maximizing the function $F(w)$ with respect

to $w$. To achieve this Yang et al. [2012b] suggested using regularized objective function

$$F\left(\mathrm{w}\right) = \sum_{i=1}^{n} p_i - \lambda \sum_{r=1}^{p} w_r^2, \tag{2.8}$$

where $\lambda$ is the regularization parameter.

Since we want to find the optimal $\boldsymbol{w}$, we set the parameter $\sigma$ in kernel function $k$ in (2.5) and the task of finding the weight vector $\boldsymbol{w}$ is equivalent to the following minimization problem for given $\lambda$

$$\hat{w} = \underbrace{\operatorname{argmin}}_{w} \; f\left(w\right), \tag{2.9}$$

where $f\left(w\right) = -F\left(w\right)$

Now we have found the weight vector $\hat{w}$ that minimizes the classification error which means that we have maximized the classification accuracy and the NCA algorithm is complete.

It is important to note that in Goldberger et al. [2005, p.514] it is stated that: "However, unlike many other objective functions (where good optima are not necessarily deep but rather broad) it has been our experience that the larger we can drive $f$ during training the better our test performance will be. In other words, we have never observed an "overtraining" effect."

The NCA thus solves the biggest shortcomings of KNN. It finds and uses a distance metric that maximizes the performance of nearest neighbor classification and this distance metric can be restricted to be low rank, hence reducing the dimensionality and reducing computation costs.

# 3. Fast Neighborhood Component Analysis (FNCA)

## 3.1 What lead to fast NCA

While NCA is considered to be one of the best performing metric learning algorithms it still has its limitations [see Weinberger and Saul, 2009, Goldberger et al., 2005, among others]. The computational cost remains to be very high, which leaves the NCA method to be applicable only for moderate data size. The computational challenge is primarily driven by the fact that all data points must be stored, compared and evaluated over the entire training set.

FNCA analysis aims to overcome the major difficulties of the NCA: "*Compared with NCA, FNCA uses a novel probability distribution with less computational cost to determine the reference point and hence significantly improve the training speed.*" [Yang et al., 2012a, p.22]

Since FNCA is a modification of NCA algorithm it uses the same statistical model, Yang et al. [2012a] define distance function different from the one in the NCA algorithm.

**Definition 3.1**
Let x,y $\in \mathbb{R}^p$. We define the distance metric $d$ as

$$d(x, y) = (x - y)^T Q(x - y), \tag{3.1}$$

where Q denotes a positive semi-definite square matrix. △

**Theorem 3.1**
Let Q be positive semi-definite real matrix. Then there exist a Cholesky decomposition of Q such that

$$Q = A^T A \tag{3.2}$$

Proof. See, e.g., Horn and Johnson [1990, p.407]. □

Note that A could be represented in a factor form as lower triangle matrix. [1]Using decomposition (3.2), the distance metric (3.1) could be expressed as

$$d(x, y) = (x - y)^T A^T A(x - y) = (Ax - Ay)^T (Ax - Ay) \tag{3.3}$$

## 3.2 Setup and algorithm

Let us consider again a general classification problem with a training set

$$S = \{(x_i, c_i), i = 1, 2, .., n\},$$

[1] Usually Choleski decomposition is written in the form $AA^T$, where A is lower triangular matrix. In the notation we exchange A and $A^T$ for compact forms of the subsequent formulas.

where $n$ is the number of observations, $x_i \in \mathbb{R}^p$ are feature vectors, $c_i \in \{1,2,\ldots, J\}$ are class labels (categories) and $J$ denotes the number of classes.

We aim to find a linear transformation matrix $L$ (size $r \times d$) which maximizes performance of the KNN algorithm in terms of the classification accuracy. This is equivalent to finding feature weight vector $\hat{w}$ that minimizes classification error in NCA, i.e. both approaches can be used in NCA algorithm and yield the same results [Goldberger et al., 2005, Yang et al., 2012b]. Any effective strategy is linked to the idea of using LOO classification proposed in Goldberger et al. [2005].

However, the error function associated with LOO classification is a discontinuous. That is because generally a very small change in matrix $A$, or weight vector $\boldsymbol{w}$, could significantly alter the neighborhood used for classification. Therefore, a small infinitesimal change in $L$ or $\boldsymbol{w}$ could lead to a jump change in a classification error.

This is why the NCA introduces stochastic neighbor assignment. The main problem associated with the approach used in NCA is that it is not very efficient procedure. In the original NCA algorithm for classification of any point we have to consider all remaining data points as its potential neighbors and calculate probability of whether they will be selected. But as Yang et al. [2012a] mentioned this is not necessary, because to determine the class of a new observation we need only its first $k$ nearest neighbors. This is the key idea for introducing a modification of the NCA algorithm known as the "Fast NCA" [Yang et al., 2012a]. To accommodate this idea the Fast NCA uses different probability distribution function to determine which data points are relevant for the classification of the reference point.

For each reference point $x_i$ let us denote $NS_k(x_i)$ the set of k nearest neighbors from the same class (NS=Nearest Same) and $ND_k(x_i)$ will stand for $k$ nearest neighbors from the different classes (ND=Nearest Different).
Applying this notation, we define the following sets:

$$D_i = \{\mathrm{j} \mid 1 \leq \mathrm{j} \leq \mathrm{n}, x_j \in ND_k(x_i)\} \tag{3.4}$$

and

$$S_i = \{\mathrm{j} \mid 1 \leq \mathrm{j} \leq \mathrm{n}, x_j \in NS_k(x_i)\} \tag{3.5}$$

In other words, $D_i$ contains potential reference points that would lead to misclassification of $x_i$, while $S_i$ contains possible reference points that will provide correct classification. It means that for correctly classified $x_i$ we should have points in $S_i$ being closer to $x_i$ than points in $D_i$.
Using this logic Yang et al. [2012a] define the probability that $x_i$ chooses $x_j$ as its reference point (nearest neighbor) as

$$p_{ij} = \begin{cases} \frac{\exp(-\frac{d_{ij}(\mathbf{L})}{\sigma})}{\sum_{k \in D_i \cup S_i} \exp(-\frac{d_{ik}(\mathbf{L})}{\sigma})} & \forall j \in D_i \cup S_i \\ 0, \quad otherwise \end{cases}, \tag{3.6}$$

where $d_{ij}$ denotes distance between $x_i$ and $x_j$ and $L$ is the linear transformation matrix, see Equation (3.2) which leads to the optimal nearest neighbor classification – the matrix we want to find. Here the parameter sigma defines the kernel width that influences the probability of each sample being selected as the reference point.

Using formula (3.6), the probability of point being misclassified is equal to the sum of assigned probabilities over the "misspecification" set, i.e.

$$p_i = \sum_{j \in D_i} p_{ij} \tag{3.7}$$

Probability of the correct classification is obviously $1 - p_i$. It could also be obtained as a similar sum over the set $S_i$ , which leads to the correct classification of the point $x_i$.
We can then write the approximation of the LOO classification error as

$$\xi_{LOO}(L) = \frac{1}{N} \sum_i p_i = \frac{1}{N} \sum_i \sum_{j \in D_i} p_{ij}. \tag{3.8}$$

However, unlike NCA, fast NCA can suffer from overfitting while minimizing (3.7), because it is employing limited training data set. Yang et al. [2012a] recommend placing some restrictions on the model, to reduce the model complexity by using a standardization or regularization of the norm.The creators of the fast NCA used and recommended popular Frobenius norm as regularizer. Using it we can rewrite the objective function as

$$\xi(\mathbf{L}) = \sum_i \sum_{j \in D_i} p_{ij} + \frac{\lambda}{2} \|\mathbf{L}\|_F^2 \quad , \tag{3.9}$$

where $\lambda$ is a nonnegative regularization parameter, which is tuned using the training data.

## 3.3 Comparing NCA and FNCA

There are three main ways that FNCA and NCA differ. First, FNCA uses different probability distribution to determine nearest points. While in NCA we compare each individual point in the data set to the rest of the entire training data set, in FNCA we constrain ourselves only to the first $k$ nearest neighbors both from the same class and from (all) different classes [Yang et al., 2012a].

Second, NCA and FNCA use different functions to define how much does every distance term contribute to $p_{ij}$. FNCA uses $exp\left(-\frac{d_{ij}(\boldsymbol{L})}{\sigma}\right)$, while NCA uses $exp\left(-d_{ij}^2(\boldsymbol{L})\right)$. The reason for that is that FNCA's goal is to minimize the LOO of the weighted KNN classification error, while NCA only aims to minimize the stochastic variant of the error associated with LOO classification.

Finally, as shown in Yang et al. [2012a], FNCA solves the overfitting issue for small training set by introducing regularization term. On the other hand NCA doesn't solve overfitting, which makes FNCA better more suitable in dealing with high dimensional data with limited training set [Yang et al., 2012a].

To compare performance of the NCA and fast NCA, let us recall the parameters influencing the computational complexity. We consider a training set $S = \{(x_i, c_i), i = 1, 2, .., n\}$ of $n$ training data vectors, where $x_i \in \mathbb{R}^p$ , i.e., $p$ refers to size of the dimensional space and $c_i \in \{1,2,\ldots, J\}$ denotes the possible classes. The linear transformation matrix $L$ of matrix $A$ (3.2) has the size $(r \times p)$, $r < p$.

Using the results of Yang et al. [2012a] we see that the most computational expensive part of the NCA is the gradient computation, with complexity $O(rpn^2)$ each iteration. It means that NCA algorithm efficiency in each iteration is quadratic in $n$ (number of data points) and linear in $r$ and $p$ where $p$ denotes dimensionality of the vector $x$ and $r(< p)$ corresponds to the dimensionality reduction brought by the linear transformation $L$.

In contrast the complexity of Fast NCA is order $O(2krpn) + O(kn^2)$, where $k$ denotes the number of nearest neighbors used for classification. The complexity of the first term reflects the idea of the fast NCA, where we use only $k$ nearest neighbors, while $O(kn^2)$ is the computational cost of finding the $k$ nearest neighbors.

For comparing the complexity of the Fast NCA and NCA we need to compare the largest part of the Fast NCA complexity with the NCA. One can see that when $2rp > n$, then the first term dominates and FNCA is about $n/2k$ times faster than NCA during each iteration. Otherwise second term (the complexity of finding $k$ nearest neighbors) dominates and FNCA is about $rp/k$ times faster in each iteration.

## 3.4 Kernel FNCA

Although FNCA solves many major issues of NCA and as stated: "Experimental results show that, compared to NCA, FNCA not only significantly increases training speed but also obtains higher classification accuracy." [Yang et al., 2012a, p. 31], it is still only suited for linear metric learning. However we can extend FNCA to nonlinear metric learning by implementing the so-called *Kernel trick*.

**Kernel trick**
The goal here is to get linear learning algorithms to learn nonlinear decision boundaries or functions, which we can achieve by transforming our data set into higher dimension. Certain functions $k(x, y)$, which are referred to as kernel function, in our input space $X$ can be expressed as inner product in another space $V$. If we can express the kernel as a feature map $\varphi : X \to V$ , then it satisfies

$$k\left(x, x^{'}\right) = \left\langle \varphi\left(x\right), \varphi\left(x^{'}\right) \right\rangle_{v} \quad , \mathrm{x}, \mathrm{x}' \in \mathrm{X} \quad ,$$

where the main restriction is that $< ., . >_v$ is a proper inner product. If it is a proper inner product, then the map $\varphi$ doesn't have to be explicitly represented, since we will be only computing inner products between all images of every two points.
For application of kernel trick in FNCA see Yang et al. [2012a, p. 33].

# 4. NCA algorithm with LDA filter [Two-step (F)NCA/KNN]

As has been stated before, the NCA may be one of the most successful metric learning algorithms, but the NCA still has its limitations. And while FNCA, both with or without the kernel trick, may alleviate many of previous concerns, the algorithm remains computationally expensive and hard to conduct for larger data sets [Yang et al., 2012a]. In this section I propose a combination of linear discriminant analysis (LDA) and KNN, NCA and/or FNCA algorithms. Main idea is to use the LDA as a starting separation rule and conduct more sophisticated and computationally expensive methods only if the reference point belongs to the area where LDA has a low predictive power. It is first important to note that since we use LDA algorithm we will restrict ourselves to LDA statistical model and its requirements (see Appendix A.3.1).

Let us illustrate the problem graphically. For sake of simplicity we assume that our class set has only two categories. Depending on (linear) separation rule we can define the benchmark for predicted probabilities to clearly assign point $x$ to one of the category. In Figure 4.1, the reference point is denoted by a star symbol.

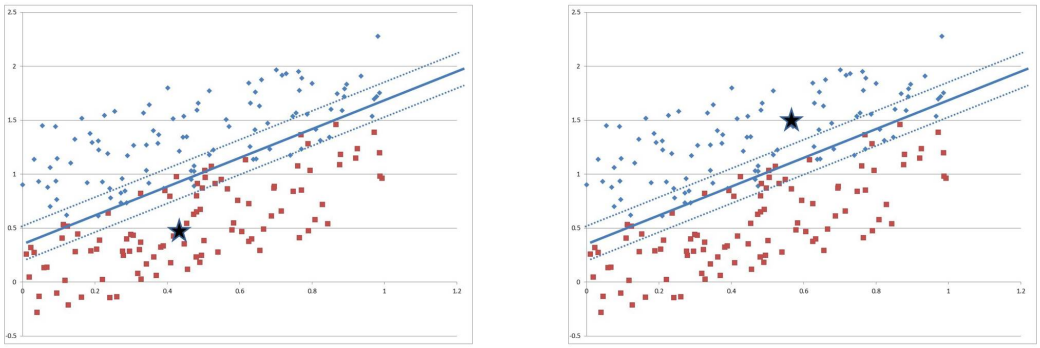

Figure 4.1: Using LDA as a filter for additional KNN, NCA or FNCA methods

As we see if the reference point x (star symbol) is far from the separation curve we can assig x into the corresponding category without running time consuming algorithms as KNN, NCA or FNCA. In terms of computer memory allocation we can keep only the points which are close to the separation curve.

Obviously reference point could belong to a "shadow" zone, i.e., to the area around the separation curve, where predicted probabilities of both classes are close. For illustration, see Figure 4.2.

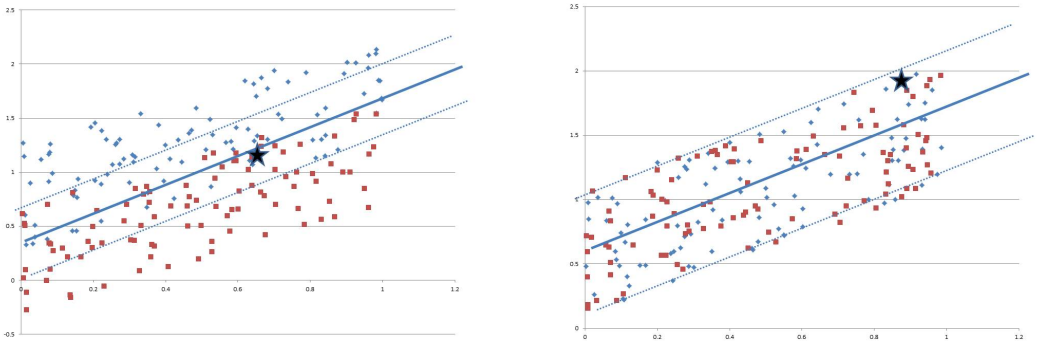

Figure 4.2: Illustrating the case when LDA filter does not help and KNN, NCA or FNCA are needed

Note that the second graph in the Figure 4.2 represents the case when the LDA does not help in separating these two classes. This is the worst case scenario and in this particular case we do not save any computer time or computer memory, since LDA filter is clearly unreliable.

**Setup of Two-step classification procedure:**
Let us consider a classification problem with a training set

$$S = \{(x_i, c_i)\,, i = 1, 2, .., n\}\,,$$

where $n$ is the number of observations, $x_i \in \mathbb{R}^p$ are feature vectors, $c_i \in \{1,2,\ldots,J\}$ are class labels (categories) and $J$ denotes the number of classes. While the algorithm could be set for general number of categories, for the sake of simplicity we will consider only two categories, i.e., $J$=2.

Step 1. Linear discrimination analysis filter

1. We choose benchmarks $p_0$ and $p_1$ which correspond to thresholds for assigning reference point $x$ into the class 0 and 1, respectively.

2. We tune LDA on the training set $S$.

3. To save computer memory we mark the points which have predictive probabilities for 0 and 1 higher or equal to $p_0$ and $p_1$, respectively. Neighborhoods around the predictive probabilities can also be tuned during training session of the entire algorithm.The algorithm works reasonably well with $p_0 = p_1 = 0.65$, but the parameter is likely depending on the quality of the LDA separation rule. This step will produce a reduced training set $S_R$.

4. For the reference point $x$ we compute predictive probability $\hat{p}_0$ and $\hat{p}_1$ that $x$ belongs to the class 0 and 1, respectively.

   (a) If $\hat{p}_0 \geq p_0$ we classify $x$ as 0 and end the algorithm.

   (b) If $\hat{p}_1 \geq p_1$ we classify $x$ as 1 and end the algorithm.

5. Reference point $x$ belongs to the area where LDA does not provide clear classification; we must proceed with additional classification algorithm(s)

Step 2. Perform KNN, NCA or FNCA on a reduced training set

1. We start more computationally demanding algorithms in our case KNN, NCA or FNCA on reduced training set $S_R$. Classification of reference point x is assigned using one of these algorithms.

Introduction, details and mathematical formulation of the LDA can be found in standard statistical books, e.g., McLachlan [2004]. In simulations and application we use and refer to relevant package(s) implemented in R.

# 5. Comparing classification methods

In this section we use several training data sets to compare KNN and algorithms discussed in previous parts expanded by SVM, which is regarded in the machine learning community as a candidate for good and relatively fast classifier. As is the norm, the performance of the algorithms is evaluated by the quality of predictions and speed. This section has two main parts.

In the first part we use large dataset for classification from the machine learning depository at Center for Machine Learning and Intelligent Systems, Bren School of Information and Computer Science, University of California, Irvine.[1] More specifically we use data for predicting individuals with annual personal income above fifty thousand dollars. Size of the dataset is exceeding 48,000 observations and hence allows better evaluation of classification algorithms from the perspective of computing efficiency. Note that because of the nature of the data (we have available individual characteristics including main factors for earning equation) [2] one can expect relatively good performance of our classification algorithms in terms of the prediction accuracy. On the other hand, given the size of the data, it could be a challenge to perform KNN type classification in a reasonable time.

In the second part we use classification algorithms for predicting future stock prices and returns. This exercise is in fact a test of the efficient market hypothesis. Data for this task were downloaded from finance.yahoo.com. Prediction of future returns (five trading days ahead) are then evaluated across our classification algorithms, using five different market indexes over the period 2000-2017. We use two year training window and one year testing window for each index and classifier. This gives us 16 pairs of training and testing sets for each index. In the context of the efficient market hypothesis (EMH) the poor classification results confirm the EMH, i.e., that future market returns are not predictable.

Implementation of algorithms and all computations were conducted in R. For evaluation, testing and training of selected algorithms the library "caret" was used [Kuhn et al., 2008]. Specific algorithms were either part of the caret, and/or used and slightly modified from the libraries "MASS" [Venables and Ripley, 2013], "knn", "AppliedPredictiveModeling" [Kuhn and Johnson, 2013, among others]. Both experiments were conducted on Windows 10 operating system with Intel(R) Core(TM) i7-6700HQ CPU @2.60GHz, 2601 Mhz, 4 Cores, 8 logical Processors, and with 64GB operating memory.

[1] See https://archive.ics.uci.edu/ml/datasets.html. This depository belongs to the best organized and publicly accessible archives for classification and machine learning.

[2] Mincer and Polachek [1974] introduced and discussed various factors influencing individual earnings. This relationship if often called "earnings equation" or "wage equation". Individual earnings are analyzed as a function of education and experience interacting with race and gender, among other factors.

## 5.1 Predicting individuals with annual income over 50,000$

In his part we use large data extracted from US Census for predicting individuals with annual personal income above 50,000 $ (For original data description see, https://archive.ics.uci.edu/ml/datasets/Adult).

This large and wildly cited data set was extracted by Barry Becker from the 1994 Census database, which was compiled by US Census Bureau. It is a data set of 48,842 adults with the goal of predicting whether their yearly income will be under or over 50,000$. Originally there are 6 continuous (e.g. age, capital gain, capital loss, ...) and 8 nominal attributes (e.g. occupation, sex, marital status, education, ...). Some observations have missing values, which we exclude to have fully defined dataset.

To create a good set of predictors we use the classical approach formed in Mincer and Polachek [1974]. Mincer and Polachek [1974] claim that earning function depends on education, experience (nonlinear effect, usually proxied by age and age$^2$), race and gender; the other social characteristics could also improve fit of the wage equation. Which is why we create variable age2=age*age and we group education and occupational variables into a broad set of indicator variables, for example female=1 if gender=female, etc.). For the final dataset the number of variables was reduced and information used for prediction reflects the classical Mincer earnings equation. [3]

### 5.1.1 Classification methods, their speed and prediction accuracy

In following tests TSKNN and TSNCA denote Two-step KNN and Two-step NCA, both of which are based on the algorithm described in Chapter 4. In these tests we use NCA implemented with Kernel trick (Chapter 2 and 3), since it is the most widely used version of NCA.

To better understand computation efficiency of the selected algorithms we conduct a series of classifications. First we randomly select the "starting dataset" which contains a gradually increasing proportion of the main dataset. We randomly draw 10 different sub-samples (no repeats) with increasing sample size. In the randomly drawn sub-samples we keep the same proportion of the individuals with income exceeding 50,000$ to maintain the similar conditions for classification.

We start at 4% of the main data set with the increase, representing 2 percent of the main sample (i.e., about 500 observations). The last dataset used corresponds to the 14 percent of the main sample. Since NCA and SVM in training session

[3] It is usually modeled in a logarithmic form, where earnings are function of individual labor experience, and education (years of schooling).

$$\ln(w) = f(\text{education, experience}) = \alpha + \beta_1 \times s + \beta_2 \times x + \beta_3 \times x^2 + \dots \ ,$$

where $s$ denotes years of schooling and $x$ stands for labor market experience. Later the model has been expanded by other individual characteristics, like gender and race, and age of the individual was used as a proxy for labor market experience.

needed substantial computer time, the computations were stopped at this portion of the main sample.[4]

For the classification exercise we split (randomly) the each sub-set (80:20). This means that in each step the training set contains about 80 percent of the corresponding sub-set and the methods were tested on remaining 20 percent of the subset. Table 5.1 contains the computer time needed to train and tune each algorithm during the training session with 10-fold cross-validation repeated 10 times.

Table 5.1: Relation between training data size and computing efficiency of selected algorithms.

| Computer time in seconds using 10-fold cv, repeated 10 times | | | | | | |
|---|---|---|---|---|---|---|
| Training data size | KNN | NCA | TSKNN | TSNCA | SVM | LDA |
| 1,449 | 14.92 | 447.11 | 6.00 | 85.62 | 352.76 | 1.34 |
| 2,172 | 21.70 | 693.15 | 7.02 | 132.06 | 720.08 | 1.55 |
| 2,896 | 33.54 | 956.70 | 8.45 | 153.49 | 1,216.59 | 1.75 |
| 3,619 | 48.27 | 1,244.36 | 10.03 | 219.12 | 2,021.04 | 2.18 |
| 4,342 | 64.11 | 1,536.86 | 10.38 | 257.33 | 2,983.54 | 2.25 |
| 5,066 | 84.12 | 1,856.30 | 11.39 | 302.09 | 3,933.41 | 2.47 |
| 5,791 | 106.45 | 2,178.60 | 12.87 | 340.28 | 5,451.44 | 2.66 |
| 6,514 | 134.38 | 2,524.45 | 14.24 | 366.61 | 6,671.16 | 3.29 |
| 7,237 | 160.49 | 2,884.73 | 16.81 | 417.59 | 8,237.42 | 3.45 |
| 7,961 | 192.42 | 3,278.42 | 18.97 | 473.97 | 9,870.69 | 3.39 |

We also include a Table 5.2, in which KNN is a benchmark and we show how many times were the algorithms slower than KNN.

Table 5.2: Relative performance of selected algorithms with respect to KNN in training sessions.

| Comparison using 10-fold cross-validation | | | | | | |
|---|---|---|---|---|---|---|
| Training data size | KNN | NCA | TSKNN | TSNCA | SVM | LDA |
| 1,449 | 1.00 | 29.97 | 0.40 | 5.74 | 23.64 | 0.09 |
| 2,172 | 1.00 | 31.94 | 0.32 | 6.09 | 33.18 | 0.07 |
| 2,896 | 1.00 | 28.52 | 0.25 | 4.58 | 36.27 | 0.05 |
| 3,319 | 1.00 | 25.78 | 0.21 | 4.54 | 41.87 | 0.05 |
| 4,342 | 1.00 | 23.97 | 0.16 | 4.01 | 46.54 | 0.04 |
| 5,066 | 1.00 | 22.07 | 0.14 | 3.59 | 46.76 | 0.03 |
| 5,791 | 1.00 | 20.47 | 0.12 | 3.20 | 51.21 | 0.02 |
| 6,514 | 1.00 | 18.79 | 0.11 | 2.73 | 49.64 | 0.02 |
| 7,237 | 1.00 | 17.97 | 0.10 | 2.60 | 51.33 | 0.02 |
| 7,961 | 1.00 | 17.04 | 0.10 | 2.46 | 51.30 | 0.02 |

Graphical comparison of KNN and TSKNN, NCA and TSNCA in Figure 5.1 shows very interesting saving time in classification, where in the first step we

[4] In data training we used for parameter tuning 10-fold cross-validation, repeated 10 times, which with increasing sub-sample leads to relatively excessive computer time spent on training and parameter tuning.

opted for the LDA classifier if the predicted probabilities for each class were high enough ($\geq$0.65).

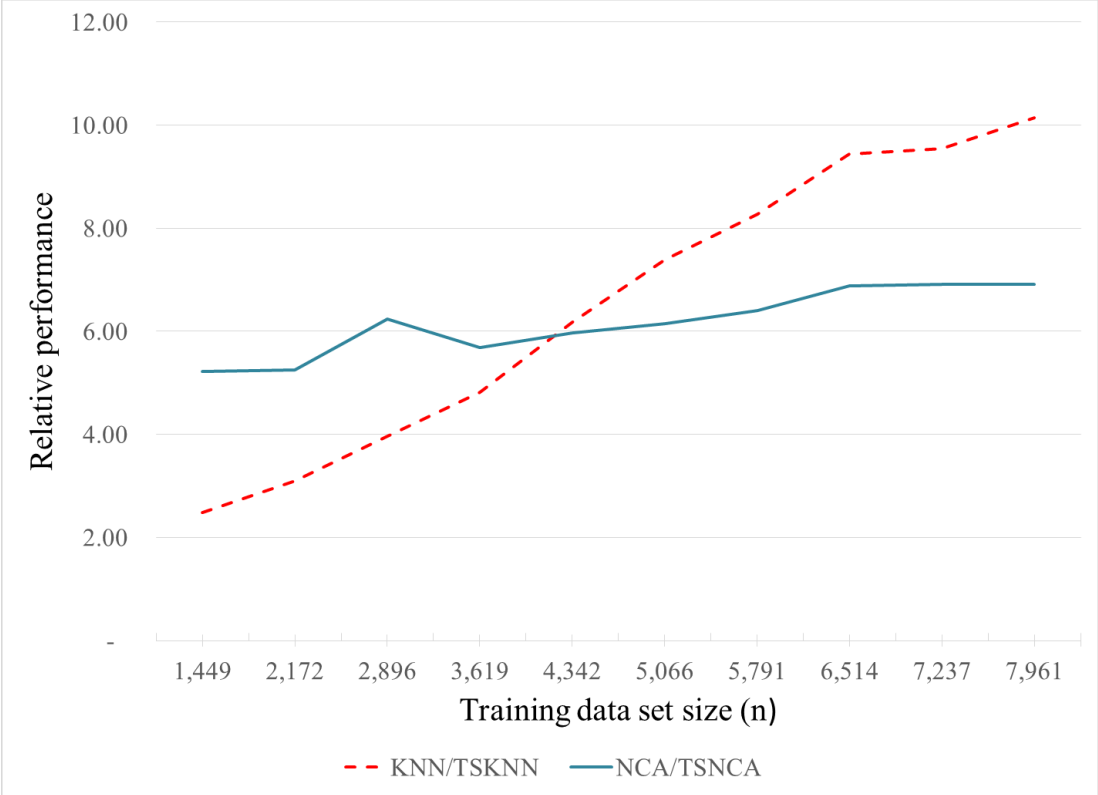


Figure 5.1: Relative performance ratio of computer time in training sessions for KNN and TSKNN, and for NCA and TSNCA, respectively.

Values in the graph correspond to the ratio of time needed to train chosen classifier on the given data size using 10-fold cross-validation, repeated 10 times. It shows how much faster is our two step method with respect to the original classifier, when in the first step we use LDA as a filter, with $p_0$ and $p_1$=0.65.

Also, as one can see from Table 5.2 summarizing the computer efficiency in the training session, our results are quite interesting and do not fully support the claims in the literature that NCA or some other specific-kernel type algorithms could lead to a faster classification compared to the classical KNN. One possible explanation could be the high number of the parameters that require tuning for NCA and SVM during the training session. In our case additional training cost are related to the choice of the kernel. Therefore, we also recorded the computer time needed to perform classification method on the testing data. Results looks much better for kernel-type classifiers, however, they are still much slower compared to the KNN. See Table 5.3 for more details.

Table 5.3: Relative performance of selected algorithms with respect to KNN in testing session.

| Testing data size | KNN | NCA | TSKNN | TSNCA | SVM | LDA |
|---|---|---|---|---|---|---|
| 361 | 1.00 | 1.71 | 0.43 | 0.57 | 2.14 | 0.43 |
| 542 | 1.00 | 1.45 | 0.45 | 0.45 | 2.36 | 0.27 |
| 723 | 1.00 | 1.65 | 0.41 | 0.41 | 2.29 | 0.29 |
| 904 | 1.00 | 1.35 | 0.15 | 0.23 | 2.15 | 0.12 |
| 1,085 | 1.00 | 1.34 | 0.14 | 0.11 | 2.14 | 0.09 |
| 1,266 | 1.00 | 1.50 | 0.13 | 0.17 | 2.20 | 0.11 |
| 1,446 | 1.00 | 1.41 | 0.10 | 0.17 | 2.14 | 0.09 |
| 1,627 | 1.00 | 1.21 | 0.05 | 0.08 | 2.10 | 0.04 |
| 1,808 | 1.00 | 1.29 | 0.07 | 0.08 | 2.12 | 0.05 |
| 1,989 | 1.00 | 1.34 | 0.07 | 0.10 | 1.98 | 0.05 |

We can speculate that additional speed could be reached using parallel computing for evaluating the kernel and/or computing relevant gradients. However, this claim would definitely need deeper analysis, more simulations and testing sensitivity with respect to the performance of implemented algorithms under the parallel computing support, which is why we will not investigate it further.

Table 5.4: Prediction accuracy of tested algorithms in training sessions

| Training data size | KNN | NCA | TSKNN | TSNCA | SVM | LDA |
|---|---|---|---|---|---|---|
| 1,449 | 0.796 | 0.841 | 0.821 | 0.838 | 0.818 | 0.795 |
| 2,172 | 0.812 | 0.872 | 0.833 | 0.855 | 0.846 | 0.807 |
| 2,896 | 0.806 | 0.853 | 0.816 | 0.866 | 0.833 | 0.821 |
| 3,619 | 0.806 | 0.854 | 0.816 | 0.852 | 0.827 | 0.802 |
| 4,342 | 0.805 | 0.858 | 0.806 | 0.857 | 0.832 | 0.807 |
| 5,066 | 0.804 | 0.849 | 0.828 | 0.842 | 0.822 | 0.803 |
| 5,791 | 0.803 | 0.848 | 0.815 | 0.859 | 0.829 | 0.803 |
| 6,514 | 0.801 | 0.858 | 0.829 | 0.847 | 0.824 | 0.803 |
| 7,237 | 0.805 | 0.854 | 0.818 | 0.850 | 0.823 | 0.803 |
| 7,961 | 0.806 | 0.844 | 0.813 | 0.838 | 0.823 | 0.806 |

Note that in training sessions all algorithms significantly dominate the naïve (non-informative rate[5]) estimate. The p-value of the $H_0$ hypothesis that prediction accuracy= non-informative rate was lower than 0.01. For definition of prediction accuracy see Definition 1.3.

[5]Non-informative rate represents the classifier that predicts always the same outcome with the highest occurrence in the data. In our case it always predicts that every individual makes less than 50,000$.

Table 5.5: Prediction accuracy of tested algorithms in testing sessions

| Data size | KNN | NCA | TSKNN | TSNCA | SVM | LDA |
|---|---|---|---|---|---|---|
| 361 | 0.792** | 0.759 | 0.781 | 0.765 | 0.7867 | 0.776 |
| 542 | 0.782 | 0.777 | 0.797*** | 0.793** | 0.804*** | 0.795** |
| 723 | 0.791*** | 0.758 | 0.777 | 0.786** | 0.798*** | 0.789*** |
| 904 | 0.770 | 0.783** | 0.797*** | 0.803*** | 0.806*** | 0.803*** |
| 1,085 | 0.781** | 0.800*** | 0.783*** | 0.792*** | 0.801*** | 0.810*** |
| 1,266 | 0.780*** | 0.785*** | 0.810*** | 0.807*** | 0.809*** | 0.793*** |
| 1,446 | 0.793*** | 0.797*** | 0.786*** | 0.794*** | 0.811*** | 0.809*** |
| 1,627 | 0.797*** | 0.788*** | 0.790*** | 0.803*** | 0.818*** | 0.805*** |
| 1,808 | 0.791*** | 0.782*** | 0.808*** | 0.813*** | 0.809*** | 0.805*** |
| 1,989 | 0.795*** | 0.788*** | 0.797*** | 0.808*** | 0.804*** | 0.790*** |

Note: ***, **, and * denotes cases when we reject $H_0$ (Prediction accuracy=Non informative rate) on 1%, 5% and 10% level of significance, respectively.

It is interesting to observe that sophisticated algorithms like NCA and SVM have a very good prediction accuracy in the training sessions (on average dominating KNN by about 4 percentage points), however their prediction accuracy worsen in testing sessions. Interestingly, KNN was either very close or dominates the NCA in several testing sessions.

Another interesting observation is related to our introduced two-step classifiers TSKNN and TSNCA. As they combine properties of the "parent" estimators, we can see significant reduction of the computer time which is not "paid for" by worse prediction accuracy. Obviously, the prediction accuracy depends on the (linear) separability and parameters $p_0$ and $p_1$ used in the LDA filtering. Nevertheless, it shows that for the large data set this two-step algorithm could be a possible tool to reduce computer complexity with relatively low cost on the prediction side.

## 5.2 Predicting future returns: A test of the efficient market hypothesis

In this part we aim to use classification algorithms to tackle one of the fundamental questions and concepts in finance. Almost every trader has in mind two interesting questions:

- What determines the movement of stock prices?
- Is it possible to forecast future price development?

In the financial literature, these questions are related to the concept of the Efficient Markets Hypothesis (EMH), originally formulated by Fama [1970]. It has later been defined and formalized by Jensen [1978] using the information available at the time $t$: "A market is efficient with respect to information set $I_t$ if it is impossible to make economic profits by trading on the basis of information set $I_t$."[Jensen, 1978]. Basically, in efficient market everything what we can learn from the existing information set $I_t$ is already incorporated in the (current) prices. For an overview of the EMH see, e.g. Malkiel [2003]. Note that the financial theory distinguishes several forms of market efficiency, depending on the definition of the information set:

- Weak form of market efficiency: $I_t$ contains only the history of financial market data (stock prices, volume, high and low prices during the session, etc.)
- Semi-strong form market efficiency: $I_t$ contains all publicly available information available at time t.
- Strong form of market efficiency: $I_t$ contains both public and private information available at time t.

In this section we use classification algorithms to predict return 5 trading sessions ahead to test the weak form of market efficiency. Basically, a direct implication of market efficiency means that it is impossible to develop a forecasting model based on price and volume history, which would generate systematically risk-adjusted expected excess returns.

### 5.2.1 Data descriptio n

We classify and predict future returns using the following market indexes: Dow-Jones industrial Average (DJI), NASDAQ market index (IXIC), Japan market index Nikkei (N225), Hang Seng market index (HSI) and Germany leading market index (DAX).[6]

Time frame for the analysis covers years 2000-2017; we use two years window for training and following one year for testing. The resulting data frame contains 16 pairs of training and testing datasets on which I computed KNN, NCA,

[6] Note that there exist several more leading market indexes, for example London (FTSE), Paris (CEC) etc. Nevertheless, the other indexes were not available in a longer history at the publicly accessible sites.

SVM (radial kernel) and linear discriminant analysis (LDA). Since in this task the speed was not the main concern, TSKNN was not included.

At the beginning the daily data, prices of all major stock market indexes listed above, were downloaded, Publicly available historical data consists of the following variables: Open, Close, Low, High and Volume. We extend this information set using basic concepts of the technical trading, which use several moving averages, short and long, to distinguish whether the market is in up/down trend. For more in-depth analysis see Murphy [1999], Pring [2002], Grimes [2012], among others.[7] Based on these rules, we include in the data set the following moving averages of closing prices: MA(10), MA(20), MA(50), MA(100), MA(200), where MA(x) denotes the moving average of the last x sessions. It is important to note that in the technical analysis, traders compare positions of short and long moving averages, for example MA(50)>MA(200) indicates that prices are in an uptrend, similarly verified by the relationship between MA(20) and MA(100), etc. Which is why the percentage distance between MA(20), MA(100) and MA(50) and MA(200) was also included alongside with the information of the percentage distance of the actual open price from MA(10),. . .,MA(200).

### 5.2.2 Training and testing

For the training data we employ 10 fold cross validation with 10 repeats (other methods yield similar results). Results from the training sets are summarized in the following set of Figures. Because of the restrictions on the length of the thesis, the following testing data windows were selected: pre-crisis (2004-2005), crisis (2007-2008), post-crisis (2010-2011) for Dow Jones Industrial Average (DJI). Other years and indexes, while computed, were not included, because of the space constrains.

Our application highlights one important aspect in learning algorithms-the performance on training samples could differ from performance on testing samples. Below we present a typical comprehensive picture that summarize results of training sessions of our four tested algorithms in selected years for DJI. The results were similar across other years and other indexes.

In the following set of figures representing a pre-crisis, crisis and post crisis training periods, it is clearly visible that SVM shows the best performance, while the KNN was the worst among the tested algorithms. The first column "ROC" denotes the area under the ROC curve (also known as the AUC(Area Under Curve) criterion). Second column called "Sens" refers to sensitivity and the third column "Spec" corresponds to Specificity values.[8] The boxplot was constructed

[7] Technical analysis has been extremely popular in financial markets. Vast majority of investment houses, brokerages and fund managers use technical analysis in some way see, e.g. [Taylor and Allen, 1992, Cheung and Chinn, 2001]. The growing popularity of the technical analysis is also visible from the scope devoted to technical analysis at the popular investment websites (See e.g., www.seekingalpha.com, www.marketwatch.com and www.investopedia.com).

[8] ROC criteron used in classification refers to the area under the ROC curve (Sensitivity versus Specificity). Typically, a random classifier has AUC (area under curve) equal to 0.5. Sensitivity (the true positive rate) measures the proportion of correctly predicted positive outcomes, while Specificity (the true negative rate) refers to proportion of correctly identified negative outcomes.

on resampling the 10-folded cross validation (repeated 10 times).

Again, easy comparisons of the three following graphs show dominance of the SVM algorithm in training sessions, across all three columns/criteria.

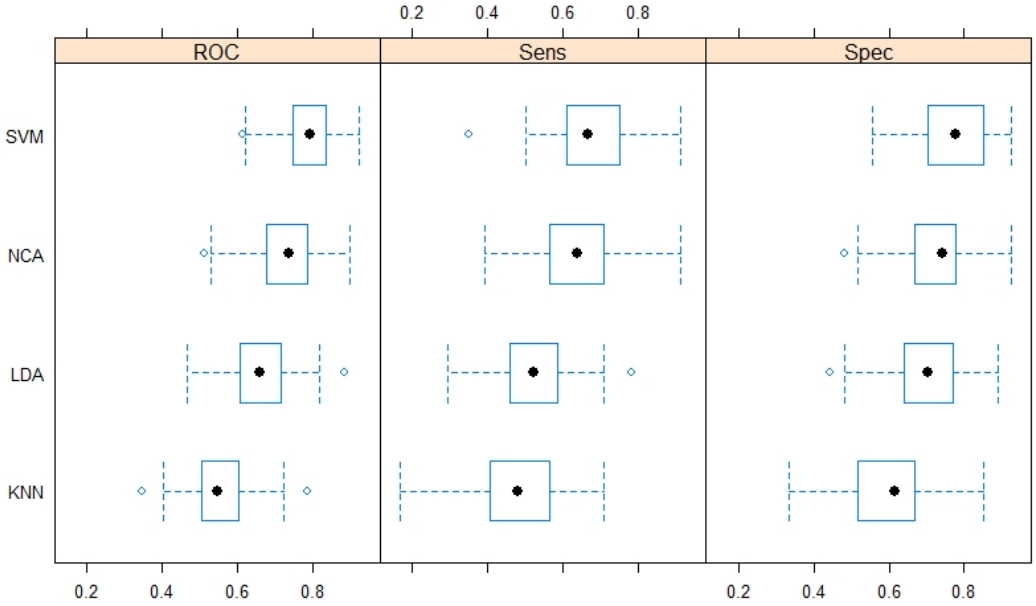


Figure 5.2: Comparison of tuning performance of predicting "buy 5" variable for DJI during pre-crisis period (2004-2005)

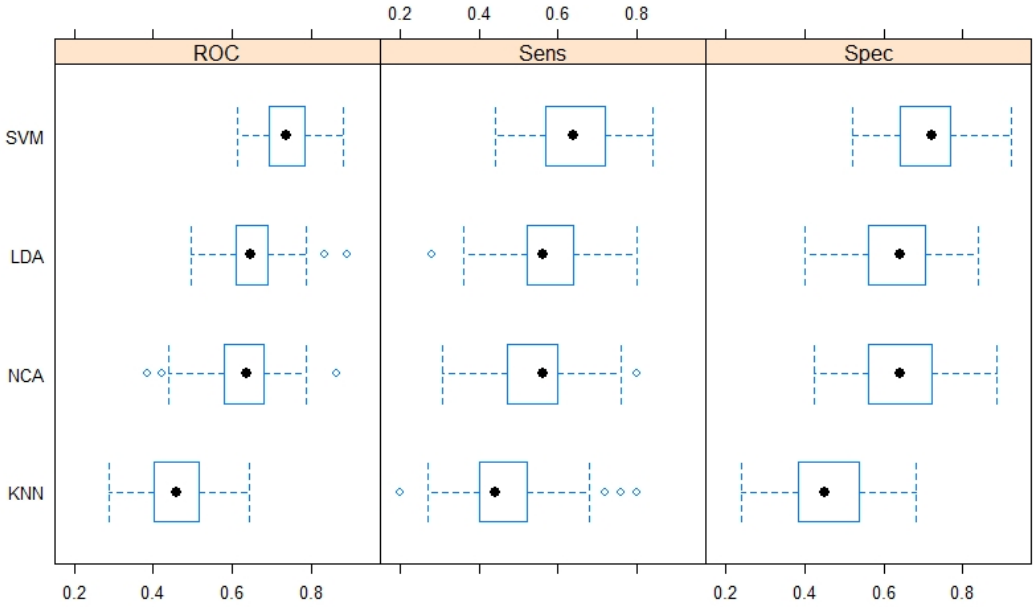


Figure 5.3: Comparison of tuning performance of predicting "buy 5" variable for DJI during crisis period (2007-2008)

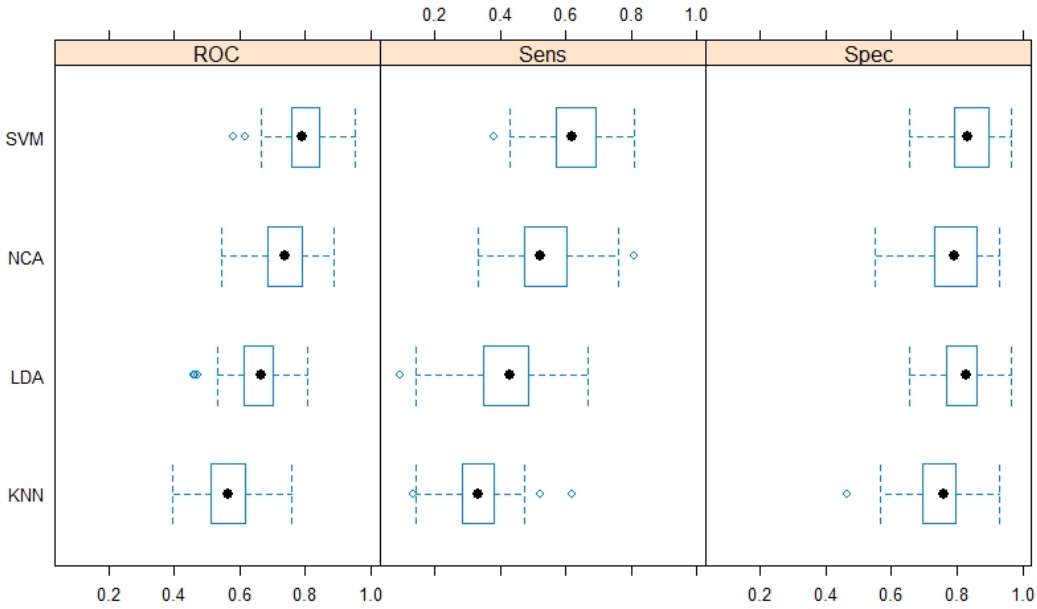


Figure 5.4: Comparison of tuning performance of predicting "buy 5" variable for DJI during post-crisis period (2010-2011)

Nevertheless, when we use trained algorithms to make predictions for the following year, the situation is not so clear, because the KNN performs better. It is however important to note that in most cases the confidence interval greatly overlaps. It is documented in the following table x. that presents the 95% confidence intervals for prediction accuracy for testing periods.

Table 5.6: 95% Confidence intervals for prediction accuracy, DJI.

| Period | KNN | NCA | SVM | LDA |
|---|---|---|---|---|
| 2002 | (0.417, 0.544) | (0.382, 0.508) | (0.382, 0.508) | (0.405, 0.532) |
| 2003 | (0.359, 0.484) | (0.302, 0.424) | (0.336, 0.460) | (0.374, 0.500) |
| 2004 | (0.413, 0.540) | (0.460, 0.587) | (0.460, 0.587) | (0.437, 0.563) |
| 2005 | (0.460, 0.587) | (0.386, 0.512) | (0.425, 0.552) | (0.528, 0.653) |
| 2006 | (0.502, 0.628) | (0.337, 0.462) | (0.546, 0.670) | (0.326, 0.450) |
| 2007 | (0.462, 0.589) | (0.498, 0.624) | (0.462, 0.589) | (0.361, 0.486) |
| 2008 | (0.384, 0.510) | (0.377, 0.502) | (0.427, 0.554) | (0.518, 0.643) |
| 2009 | (0.476, 0.602) | (0.402, 0.528) | (0.325, 0.448) | (0.382, 0.508) |
| 2010 | (0.433, 0.560) | (0.584, 0.706) | (0.572, 0.694) | (0.488, 0.614) |
| 2011 | (0.468, 0.595) | (0.445, 0.571) | (0.425, 0.552) | (0.402, 0.528) |
| 2012 | (0.436, 0.564) | (0.440, 0.568) | (0.389, 0.516) | (0.468, 0.595) |
| 2013 | (0.464, 0.591) | (0.291, 0.412) | (0.429, 0.556) | (0.355, 0.480) |
| 2014 | (0.441, 0.567) | (0.504, 0.630) | (0.544, 0.668) | (0.532, 0.656) |
| 2015 | (0.484, 0.610) | (0.452, 0.579) | (0.417, 0.544) | (0.441, 0.567) |
| 2016 | (0.528, 0.653) | (0.520, 0.645) | (0.452, 0.579) | (0.460, 0.587) |
| 2017 | (0.579, 0.701) | (0.583, 0.705) | (0.599, 0.720) | (0.322, 0.446) |

Table 5.7: Mean returns on investment strategy based on the particular model prediction to buy and sell in 5 trading days

| Index | KNN | NCA | SVM | LDA | Naïve | Random |
|---|---|---|---|---|---|---|
| DAX | 0.155 | 0.108 | 0.105 | 0.083 | 0.163 | (0.087, 0.94) |
| HIS | 0.131 | 0.059 | 0.068 | 0.094 | 0.171 | (0.116, 0.124) |
| N225 | 0.016 | 0.060 | 0.060 | 0.096 | 0.152 | (-0.001, 0.007) |
| IXIC | 0.131 | 0.103 | 0.026 | 0.071 | 0.194 | (0.115, 0.121) |
| DJI | 0.100 | 0.080 | 0.080 | 0.042 | 0.138 | (0.086, 0.092) |

"Naïve" column corresponds to the strategy "always buy". The "never buy" strategy is not represented, since its return is obviously equal to zero. The last column called "Random" represents 95 percent confidence intervals for the mean return of a random identifier which with probability 0.5 buys the index in each session (the confidence interval is based on 100 replications of such a random strategy).

With the exception of Japan (Nikkei 225), the strategy based on KNN predictions generates the best return among the tested methods. Despite the 2008 crisis, during the period 2002-2017 all leading capital markets were in upward trend, hence naïve strategy was (ex-post) dominating our candidates for the prediction. To tackle an effect of purely random identifier, we computed returns for the strategy that buy the index randomly, i.e., with the probability equal to 0.5.

It is interesting that for all indexes KNN identification strategy dominates pure random investment returns. In other words, even the simplified use of the technical analysis rules in case of KNN shows a steady pattern of returns above the 95% confidence intervals for the random strategy and therefore supports the idea that EMH could be potentially questioned with the proper use of the existing historical information.

This result could explain increasing popularity of the technical analysis in predicting trends and helping investors to analyze their positions in the capital market. Nevertheless, is should be interpreted with a caution, because of limited analysis of the training and testing sets, parameter settings and choice of classification algorithms.

On the other hand for prediction we employed only very basic principles of the technical analysis and we did not use recommended trading signals. Finally, it should be noted that trading costs were not taken into account. It does not affect the results, but the expected real returns would be lower, as the trading costs have to be covered.

# Conclusion

The goal of this thesis was to introduce the NCA algorithm, which was created to make KNN algorithm usable in computationally expensive data sets. Following the introduction into classification and defining relevant terms we explained the changes in NCA from KNN algorithm. We then introduced a variant of the NCA algorithm termed FNCA both with and without a kernel trick.

Afterwards we proposed a new two-stage algorithm using LDA in the first step as a filter to reduce the computational cost. We use classification outcomes from LDA for the points with predicted probability above the chosen benchmark. The remaining data points are classified in the second step, which employs more time and memory consuming algorithms. In the second step we use KNN based algorithm (KNN or NCA). This approach results in a two-step algorithm termed TSKNN and TSNCA, respectively. It is however important to note that in the second step we could use any time/memory demanding algorithm. We used the statistical software R to implement these algorithms.

Finally we applied these algorithms on the real data to compare their computing efficiency and classification accuracy. First (large) data set represents cross-sectional data on individual income with the goal to predict individuals with their annual income exceeding 50,000$. Second data set represents time-series data on stock market prices and returns during the period 2000-2017. We use the first data set primarily for demonstration of the computational efficiency and prediction accuracy. The second data set was used to provide a test of the efficient market hypothesis (EMH), using the predictions to buy a market index and sell it five trading days later. Based on the first dataset we concluded that the NCA algorithm is rather demanding in terms of the computer time and it shown to be not applicable on the large data set. Using the second data set we conclude that EMH could be questioned since the investment strategy based on the KNN predictions (using principles of the technical analysis) provided the mean returns above the random identifier.

# Bibliography


Yin-Wong Cheung and Menzie David Chinn. Currency traders and exchange rate dynamics: a survey of the us market. *Journal of International Money and Finance*, 20(4):439–471, 2001.

Richard O Duda, Peter E Hart, and David G Stork. *Pattern Classification*. John Wiley & Sons, 2012.

Eugene F Fama. Efficient capital markets: A review of theory and empirical work. *The Journal of Finance*, 25(2):383–417, 1970.

Evelyn Fix and Joseph L Hodges Jr. Discriminatory analysis-nonparametric discrimination: consistency properties. Technical report, California Univ Berkeley, 1951.

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.

William H Green. Econometric analysis (7th), 2007.

Adam Grimes. *The Art and Science of Technical Analysis: Market Structure, Price Action and Trading Strategies*, volume 546. John Wiley & Sons, 2012.

Roger A Horn and Charles R Johnson. *Matrix analysis*. Cambridge University Press, 1990.

Michael Jensen. Some anomalous evidence regarding market efficiency. 1978.

Max Kuhn and Kjell Johnson. *Applied Predictive Modeling*, volume 26. Springer, 2013.

Max Kuhn et al. Caret package. *Journal of Statistical Software*, 28(5):1–26, 2008.

Burton G Malkiel. The efficient market hypothesis and its critics. *Journal of Economic Perspectives*, 17(1):59–82, 2003.

Geoffrey McLachlan. *Discriminant analysis and statistical pattern recognition*, volume 544. John Wiley & Sons, 2004.

Jacob Mincer and Solomon Polachek. Family investments in human capital: Earnings of women. *Journal of Political Economy*, 82(2, Part 2):S76–S108, 1974.

John J Murphy. Technical analysis of the futures markets: A comprehensive guide to trading methods and applications, New York Institute of Finance, 1999.

Martin J Pring. *Study Guide for Technical Analysis Explained*. McGraw-Hill, 2002.

Jun Shao. Linear model selection by cross-validation. *Journal of the American Statistical Association*, 88(422):486–494, 1993.

Mark P Taylor and Helen Allen. The use of technical analysis in the foreign exchange market. *Journal of International Money and Finance*, 11(3):304–314, 1992.

William N Venables and Brian D Ripley. *Modern applied statistics with S-PLUS*. Springer Science & Business Media, 2013.

Kilian Q Weinberger and Lawrence K Saul. Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research*, 10(Feb):207–244, 2009.

Wei Yang, Kuanquan Wang, and Wangmeng Zuo. Fast neighborhood component analysis. *Neurocomputing*, 83:31–37, 2012a.

Wei Yang, Kuanquan Wang, and Wangmeng Zuo. Neighborhood component feature selection for high-dimensional data. *Journal of Computers*, 7(1):161–168, 2012b.

# A. Appendix

## A.1 K-th nearest neighbor (KNN)

### A.1.1 Inicialization

Let ($X_i$, $C_i$) where i = 1, 2...,n, be data points. For each $i$, $X_i$ is a vector of known characteristics of the observation and $C_i$ denotes its class (category). Let denote by c the number of classes, i.e., $C_i \in \{1, 2, 3, \ldots, J\}$ for all values of $i$. In general we assume that number of classes $J$ is small relative to the number of observations $n$.
Let $x$ be a point for which category $C_x$ is not known, and we would like to find/assign the class of the point $x$ using $k$-nearest neighbor algorithms.

### A.1.2 Algorithm

1. Select the preferable distance between data points $d(x_i, x_j)$, e.g., Euclidean distance.
2. Let $x$ denotes the point for classification.
3. For all data points calculate the distance from the point $x$, $d_i = d(x, x_i)$, $i = 1, 2, \ldots.., n$.
4. Sort distances $d_i$ from the smallest to the largest value (non-decreasing order)
5. Let $k$ be a positive integer, denoting the number of nearest neighbors. We take the first $k$ elements from (4.) which refers to $d_i$ with the smallest distance from $x$.
6. Identify those $k$-points corresponding to these $d_i$ distances from $x$.
7. Result: Identify $x$ with the majority class among its $k$-nearest neighbors.

Formally we define the set of the $k$ nearest neighbors of the point $x$ as

$$K_x = \{i \mid d_i <= d_{(k)}, \text{ where } d_{(k)} \text{ denotes } k\text{-th order statistics}\}.$$

Let $c_i$ denotes the number of the points belonging to the $i^{th}$ class from $K_x$.
If $c_i > c_j \; \forall \; i \neq j$ then $x$ will be put in class $i$. In case of ties we assign $x$ into the same class as the closest point.

### A.1.3 Pseudocode

**function** KNN: KNN(X, C, x) (X,C): training data, size n; X: feature values, C: associated classes of X, x unknown (reference) point to be classified
**for** i=1 **to** n do
    Compute distance d($X_i$, x)
**end for**
Sort (ascending) d($X_i$, x)
Create a set $O$ of $k$ first points in the sort (smallest distance)
**return** majority label in $\{C_i,\ where\ i \in O\}$.

## A.2 Neighborhood component analysis (NCA)

### A.2.1 Inicialization

Let ($X_i$, $C_i$), where i = 1, 2. . . ,n be data points. For each $i$, $X_i$ is a vector of known characteristics of the observation and $C_i$ denotes its class (category). Let us denote by $J$ the number of classes, i.e., $C_i \in \{1, 2, 3, \ldots, J\}$ for all values of $i$. In general we assume that number of classes $J$ is small relative to the number of observations $n$.
Let x be a point for which category $C_x$ is not known, and we would like to find/assign the class of the point $x$ using $k$-nearest neighbor algorithms.

### A.2.2 Pseudocode

**function** NCA: NCA($S,\ \alpha,\ \sigma,\ \lambda,\ \gamma$)
S: training set, $\alpha$: initial step length, $\sigma$: kernel width, $\lambda$: regularization parameter, $\gamma$: small positive constant,$y_{ij}$ is indicator that is equal to 1 if $x_i$ has the same class as $x_j$
**Initialization** $w^0 = (1, ..., 1),\ \epsilon^0 = -\infty$ , $t = 0$
**repeat**
**for** i= 1,. . . , n **do**
Compute $p_{ij}$ and $p_i$ using $w^t$ according to Equations (2.5) and (2.6).
**for** l= 1, . . . , d **do**

$$\Delta = 2\left(\frac{1}{\sigma}\sum_i \left(p_i \sum_{j\neq i} p_{ij}\,|x_{il}-x_{jl}| - \sum_j y_{ij}p_{ij}|x_{il}-x_{jl}|\right)-\lambda\right)w_l^t$$

$$\begin{array}{c} \text{t=t+1} \\ w^0 = w^{t-1} + \alpha\Delta\epsilon^t = \xi(w^{t-1}) \end{array}$$

**if** $\epsilon^t > \epsilon^{t-1}$
**then** $\alpha = 1.01\alpha$
**else** $\alpha = 0.4\alpha$
**until** $|\epsilon^t - \epsilon^{t-1}| < \gamma$
$w = w^t$
**return** w

## A.3 Linear discriminant analysis (LDA) using linear probability model (LPM)

In this thesis we employ a linear discriminant analysis as a first step of our proposed two step classification procedures TSKNN and TSNCA.

While in the statistical literature the linear discriminant analysis is often associated with so called binomial regression models, which cover a broader set of the statistical models, we will build up our classification and predictions on a special case of the linear probability model.

In the *Linear probability model* (LPM) the dependent variable y (taking value 0 or 1) is fitted by a simple linear regression model. The value of $y$ is predicted to be equal to 1 if $\hat{y}$ based on regression fit is greater or equal to chosen threshold (usually 1/2) and 0 otherwise. We can formalize our approach by the following model.

### A.3.1 Model

Consider a classification problem with the training data set

$$S = \{(x_i, c_i), i = 1, 2, \ldots, n\}$$

, where $n$ is the number of observations, $x_i \in \mathbb{R}^p$ are feature vectors, $c_i \in \{1,2,\ldots, J\}$ are class labels (categories) and $J$ denotes the number of classes. For the sake of simplicity we will consider only two categories, i.e., $J$=2.
Let us define variable

$$Y_i = \begin{cases} 1 & \text{if } \mathrm{c}_i = 1 \\ 0 & \text{otherwise} \end{cases} \tag{A.1}$$

We assume that

$$Y_i \ = \beta_0 \ + \beta_1 \times x_{i,1} + \ldots + \beta_k \ \times x_{i,k} + \epsilon_i, \tag{A.2}$$

i.e., $Y_i$ is described by a classical linear regression model. We assume that $\epsilon_i$ are independent random variables, normally distributed with a zero mean.

For the relationship of the feature vectors and outcome we assume either that the feature vectors are not random, or that $\mathbb{E}(\epsilon_i | \, x_i) \ = 0$. Given that $Y_i$ is not continues variables, the variance of $\epsilon_i$ is not constant across outcomes, i.e., $var(\epsilon_i|Y_i = 1)$ is different from $var(\epsilon_i|Y_i = 0)$.[1]
By taking expected value in Equation A.2 we obtain:

$$\mathbb{E}(Y_i \mid x_i) = \beta_0 + \beta_1 \times x_{\mathrm{i,1}} + \ldots + \beta_k \times x_{\mathrm{i,k}} \tag{A.3}$$

and since $\mathbb{E}(Y_i|x_i) \ = \ 1 \times \mathrm{P}(Y_i = 1|x_i) \ + \ 0 \times \mathrm{P}(Y_i = 0|x_i) = \mathrm{P}(Y_i = 1|x_i)$, we obtain

$$p = \ \mathrm{P}(Y_i = 1|x_i) \ = \beta_0 + \beta_1 \times x_{\mathrm{i,1}} + \ldots + \beta_k \times x_{\mathrm{i,k}}. \tag{A.4}$$

[1] Therefore, for the estimation, the ordinary least squares method is not the best linear unbiased estimator, we have heteroskedastic model and we should use Newey-White adjustment to obtain heteroskedastic-consistent standard errors.

Note that for some combination of feature variables we can predict probability larger than 1 or negative, which is a major reason why in application researchers prefer logit or probit models. Nevertheless, the drawback of the LPM that estimated probabilities could get outside of the [0,1] interval does not affect the classification based on the LPM. Simply because we assign $\hat{Y}_i$=1 if $\hat{p} \geq$ B, where B represents selected threshold, usually set to be equal 1/2. For more details, predictions and classification using linear probability model see, e.g. Green [2007].

### A.3.2 Algorithm

1. Select values of classes $\{c_1$ and $c_2\}$ associated with Y=0 and Y=1, respectively. Set the threshold B to be used for assigning class classification according to the predicted probability. Default setting B=1/2.

2. Select the relevant sub-set of feature vectors to be used for modeling linear probability model (A.2).

3. Estimate model (A.2) by the OLS model, obtain estimates of coefficients $\hat{\beta}_0, \ldots, \hat{\beta}_k$.

4. To classify a point $x$, compute predictive probability $\hat{p}$ using equation (A.4) with the estimated regression coefficients.

5. If $\hat{p} \geq$ B classify $Y$=1, otherwise classify $Y$=0. Based on the value of Y assign the corresponding class.

## A.4 Definitions

**Mahalanobis distance**
We define Mahalanobis distance of an observation $\mathbf{x} = (x_1, \ldots, x_N)^T$ from a set of observations with mean $\mu = (\mu_1, \ldots, \mu_N)^T$ and with covariance matrix $\Sigma$ as

$$D_M(x) = \sqrt{(x-\mu)^T \Sigma^{-1} (x-\mu)}$$

**Euclidean distance**
Let $p, q \in \mathbb{R}^n$. We define Euclidean distance $d$ from $p$ to $q$ as

$$d(p,q) = d(q,p) = \sqrt{(q_1 - p_1)^2 + \cdots + (q_n - p_n)^2}$$