Vysoká škola ekonomická v Praze

Fakulta financí a účetnictví

Katedra bankovnictví a pojišťovnictví

Studijní obor: Finance

Application of Artificial Intelligence Techniques in Credit Risk

Autor bakalářské práce: Martin Rýpar

Vedoucí práce: Ing. Milan Fičura

Rok obhajoby: 2018

Prohlášení

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a že jsem uvedl všechny použité prameny a literaturu, ze které jsem čerpal.

V Praze dne 25. května 2018 .....

Podpis studenta

Poděkování

Rád bych poděkoval Ing. Milanu Fičurovi za ochotu a vedení při psaní této bakalářské práce.

Abstrakt

Tato bakalářská práce se zabývá metodami umělé inteligence a jejich využitím při modelování kreditního rizika, konkrétně při modelování pravděpodobnosti defaultu. V teoretické části práce jsou popsány použité metody, tedy logistická regrese, náhodné lesy, support vector machines a neuronové sítě. V praktické části jsou tyto metody implementovány a vytrénovány na datech z online peer-to-peer platformy Lending Club a na datech z online soutěžící platformy Kaggle. V závěru jsou prezentovány výsledné hodnotící metriky, kde je ilustrováno, že metody UI mohou dosahovat lepších výsledků oproti běžně užívanému standardu - logistické regresi.

Klíčová slova

strojové učení, umělá inteligence, neuronové sítě, kreditní riziko, pravděpodobnost defaultu

Abstract

This bachelor thesis describes artificial intelligence methods and their application in credit risk modelling, particularly in probability of default modelling. In theoretical part are described methods used in practical part, namely logistic regression, random forests, support vector machines and neural networks. In practical part are those methods implemented and trained on data from online peer-to-peer platform Lending Club and on data from online competition platform Kaggle. In the end are presented evaluation metrics, where is showed that AI methods can reach better results compared to commonly used standard - logistic regression.

Keywords

machine learning, artificial intelligence, neural networks, credit risk, probability of default

Contents

List of Figures

List of Tables

Introduction

The artificial intelligence experiencing huge amount of attention in recent years. This wave of interest is mainly driven by research and its findings in the field of machine learning. The development in recent years was so rapid, that things considered as futuristic a few years ago, are perceived without surprise today. There are applications or its possibility which widely penetrated not only academic discussion, but also public discussion. Such topics are self-driving cars or dominance of computers in games like Chess or Go. But those are the exposed examples, there exists many different applications which are not so well known within public. Those can be, besides many others, optimization in search engines, software enhancements, programs trading on the financial markets or programs for credit risk assessment.

This thesis focuses on application AI methods only in banking, exactly in a credit risk assessment. In choosing a subject of this thesis, both topicality and mainly personal interest in this field were key motivations for the choice. The first goal is to create several classification models, which can predict whether the loan applicant will repay the debt or not. And the second goal is to compare these models, mainly in terms of significant differences in accuracy of predictions. Particularly, compare the predictive capability of industry standard methods, such as logistic regression, with non-standard methods.

Structure of the thesis is divided into three sections, the first two parts are both theoretically focused, compared to the third part, which is more practical. The first section introduce some of the basic terms of machine learning and artificial intelligence. The second part describes some of the frequently used methods in credit risk assessment. Then in the third part are the theoretically described methods from second part implemented and compared.

1 Short Introduction to Machine Learning and Artificial Intelligence

In this part are introduced some of the very basics of machine learning (ML), artificial intelligence (AI) and its relationship, in order to gain some basic familiarity with the topic. Generally, machine learning algorithms can be divided into three subgroups. The most often used way how to divide them is as follows [1]:

1. *Supervised learning algorithms* experience a dataset of many observations with a number of features, where each observation is associated with a label or target. Also called the learning with a guide. Such examples are classification tasks like image recognition models or credit worthiness of loan applicant and regression problems as well.

2. *Unsupervised learning algorithms* experience a dataset containing many observations with a number of features, but without any label or target. Then algorithm learns useful properties of the structure of the dataset without a guide. Such as clustering tasks, where the goal is to find the datapoints with certain similarities hidden in the data or learning the entire probability distribution that generated the dataset.

3. *Reinforcement learning algorithms* those are algorithms which interact with an environment, so there is a feedback loop between the learning system and its experiences. Self-driving cars or ALphaGo Zero – program playing the game of Go, which learned how to play by playing against earlier version of similar program AlphaGo [2], are examples of reinforcement learning.

It should be noticed, that those terms above are not formally defined and the edges between them are not sharp. Nevertheless, they are frequently used and helps with orientation in the field of ML.

To illustrate relationship between AI and ML, I found myself very useful the Venn diagram showed on Figure 1. From the figure is clear that ML is included in AI, thus the ML can be considered as a discipline of AI or an approach to AI. That applies for representation learning and deep learning as well.

Machine learning refers to an ability of AI system. The ability to acquire knowledge by extracting patterns from raw data. This capability is very beneficial when tackling

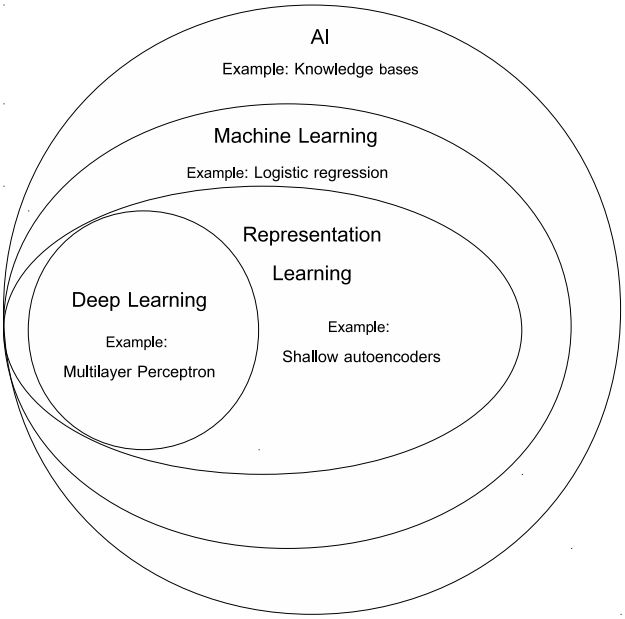


Figure 1: Venn diagram (source: [1])

real-world problems. Well known ML algorithms are Logistic regression, Naive Bayes or Decision Trees. Representational learning is mainly concerned by finding an appropriate representation of data with goal to perform ML task. You can imagine that as having linearly non-separable data on input, then task for representation learning is to find its better representation in order to make them linearly separable. Deep learning is an approach to AI, which by constructing complex model composed of related simpler units poses great power and flexibility. [1] The model as a interconnected hierarchy is represented by neural network to which will be devoted more space in the next part.

2 Methodology

Since the question whether the loan applicant is credible in terms of debt repayment has yes/no answer, we can call it a classification task. In other words, there is a demand on the model to correctly output which of K categories some input belongs to. In case of credit risk assessment or default modelling the output is binary, thus the we can call it a binary classification task. Some of the below described methods are possible to use, with some adjustments, as a regression model as well. In regression task is model asked to output a numerical value for some input, where the output is real number. This section includes a theoretical description of methods which will be implemented in the third part of the thesis.

2.1 Logistic Regression

Logistic regression is very well known and often used method within risk analysts and risk managers, where this approach is considered as a standard, due to its relatively simple interpretation of calculation steps and undemanding implementation. Opposite to classic linear regression, which is usually very easy to use, this method is more appropriate as it is more suitable when analyzing the binary dependent variable. Subsection is mainly based on this source: [3].

In case of probability of default modelling, the logistic regression is used to model probability that vector of features for i-th applicant x_i belongs to the default class Y=1, or formally

$$P(x_i) = P(y_i = 1|x_i) \tag{1}$$

The P(x_i) term represents the conditional probability of default for a given input vector of features x_i. Where under features are coded particular characteristics of given applicant. For the logistic regression is used mathematical function, which is often referred as the sigmoid function and may be denoted as

$$P(z_i) = \frac{e^{z_i}}{1 + e^{z_i}} = \frac{1}{1 + e^{-z_i}}. \tag{2}$$

Since the output of sigmoid function may be interpreted as a probability, see that the same notation P is kept for both conditional probability and sigmoid function. The z_i is

real-valued number and latent credit score of i-th loan applicant, the β^T is transformed vector of coefficients.

$$z_i = \beta^T \cdot x_i. \tag{3}$$

The logistic function offers very convenient properties as it is a S-shape curve which maps any real-valued number into a value between 0 and 1. In a sense it compress the $(-\infty, \infty)$ interval into $(0, 1)$ which may be easily interpreted as a probability. And thus equation (2) can be rewritten into this form

$$P(x_i) = \frac{e^{\beta^T \cdot x_i}}{1 + e^{\beta^T \cdot x_i}} = \frac{1}{1 + e^{-\beta^T \cdot x_i}}. \tag{4}$$

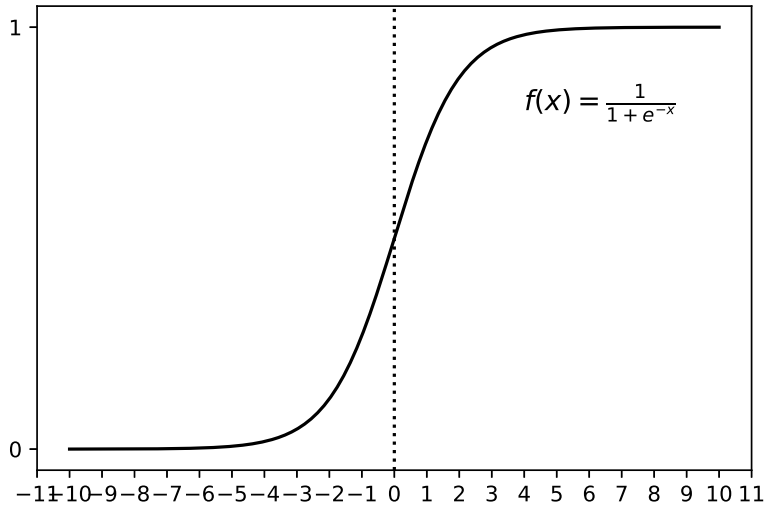


Figure 2: Sigmoid function (source: own processing)

Further is crucial to define odds function as a ratio of defaults and non-defaults, this is often referred as logit function as well

$$\text{odds(x)} = \frac{P(x)}{1 - P(x)}. \tag{5}$$

One can verify that this ratio is after several steps possible rewrite as $e^{x_i \cdot \beta^T}$. With goal to get linear expression on the right side of the previous equation, the whole equation is logarithmized in order to get so called log-odds function

$$\ln \frac{P(x)}{1 - P(x)} = \beta^T \cdot x_i. \tag{6}$$

The parameters of logistic function could be estimated using Ordinary Least Squares (OLS) or Weighted Least Squares (WLS), but the Maximum Likelihood Estimation (MLE) is a better choice. [3] Assuming independence of observations, the estimation b based on the maximization of log-likelihood function is

$$\ln L(b) = \sum_i [y_i \cdot \ln(P(-b^T \cdot x_i)) + (1 - y_i) \cdot \ln(1 - P(-b^T \cdot x_i))]. \tag{7}$$

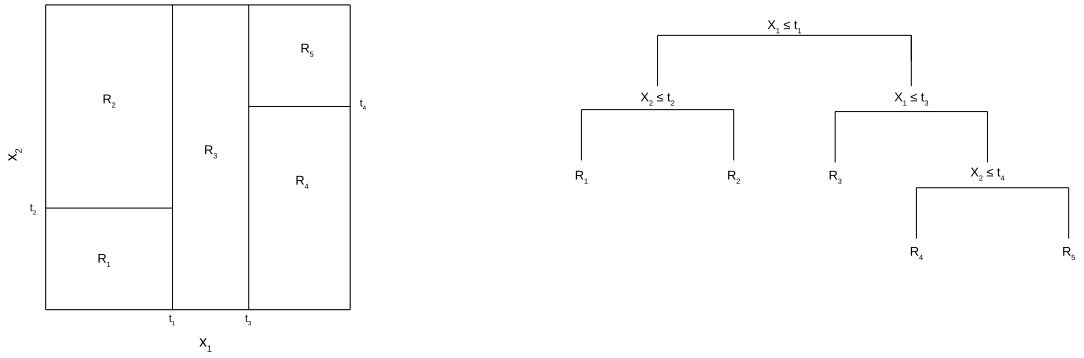
2.2 Random Forests

Random forests, with which Breiman officially came in paper from 2001 [4], is one of the tree-based algorithms with variety of possible applications. But before giving a more accurate description of method itself, there is need to define its building blocks – decision trees and bagging, which will be done in the following subsections.

2.2.1 Decision Trees

Decision trees, the core part of the Random Forest, are supervised learning method which may be used for classification and regression as well. Contrary to logistic regression, decision trees are non-linear method. The aim is to create a model that predicts the value or class of a given variables by learning simple decision rules inferred from the data. The advantage of this approach is its interpretability as the method itself is very close simulation of real-person decision-making process. However, there are naturally some shortcomings of tree-based methods such as high sensitivity to training data, where even small change in the training dataset can result in a very different set of splits. This subsection is particularly based on materials from [5] and [6].

For the description of tree-based methods will be used popular classification and regression tree model or shortly CART [6], even though there are many other variations such as C4.5 or ID3. The CART algorithm provides partition of the features space into a set of rectangles and then fit a simple model (such as constant) in each one. To illustrates a recursive partitioning of the input space hand in hand with the corresponding tree structure, see Figure 3.



(a) Two dimensional input space partition

(b) Decision tree corresponding to partition

Figure 3: CART partitioning (source: [5])

More formally, consider a dataset of N observations and P inputs, which is $(\mathbf{x_i}, y_i)$ and for each observation $i = 1, \ldots, N$ with input vector $\mathbf{x_i} = (x_{i1}, \ldots, x_{iP})$ is given output variable y_i. In case of classification the output variables would look like this, $y_i = 1, \ldots, K$ where K denotes a number of classes, but for now we stay with the regression. Then suppose that the tree is partitioned into M regions $R_1, \ldots, R_M$ and response as a constant c_m in each region is modelled.

$$f(x) = \sum_{m=1}^{M} c_m \cdot I(x \in R_m). \tag{8}$$

Using sum of squares $\sum(y_i - f(\mathbf{x_i}))^2$ as the minimization criteria, one can reach the conclusion that the best constant separator $\hat{c}_m$ is simply the average of y_i in the region R_m. [5]

$$\hat{c}_m = avg(y_i | \mathbf{x_i} \in R_m). \tag{9}$$

The algorithm needs to, besides choosing input variables for each split and appropriate separators, determine the optimal tree structure. Combinatorically large number of possible solutions usually makes minimization using sum of squares error computationally unfeasible. That is reason why is so called greedy optimization employed – starting the growing with a single root node, corresponding to the whole input space, and then one by one node the tree greedy grows. Consider a splitting variable j and split point s, then we can define the pair of half-planes

$$R_1(j,s) = \{X|X_j \leq s\} \text{ and } R_2(j,s) = \{X|X_j \geq s\}. \tag{10}$$

Then below expression is solved by finding optimal variable j and s, and as we already know the sum of squares error is optimized by an average, thus the inner minimization might be rewriten as an average of c_i in the specific region R_m.

$$\underset{j,s}{min}[\underset{c1}{min} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \underset{c2}{min} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2] \tag{11}$$

Finding an optimal separator enables to split the data into two resulting regions and repeat the partition on each of those two regions. In case of applying this process, without any adjustment, to all resulting regions, we might end up with too big and overfitted tree which fails in predicting new observations or otherwise only with a small tree not capable of capturing the important structure. As the tree size is a tuning parameter, representing complexity of the model and which should be adaptively chosen from the data[5], there arise an obvious question. How to ensure that tree reaches the optimal or at least close to optimal size? Generally preferred approach is to grow a large tree T_0 using minimum node size as a stopping criteria and then prune it. The minimal node size is usually another finetuned parameter and refers to minimal number of datapoints in each node. The pruning process, which is described below, balances the residual error against a measure of model complexity.

Let's define a subtree $T \subset T_0$ as any tree that can be obtained by pruning T_0, that is cutting off any number of its internal (non–terminal) nodes. And terminal nodes indexed by m, giving $|T|$ terminal nodes from total number of T nodes. The cost complexity criterion is given by

$$C_\alpha(T) = \sum_{m=1}^{|T|} \sum_{x_i \in R_m} (y_i - \hat{c}_m)^2 + \alpha \cdot |T|, \tag{12}$$

denoting $\sum_{x_i \in R_m} (y_i - \hat{c}_m)^2$ as $Q_m(T)$ we can simply write

$$C_\alpha(T) = \sum_{m=1}^{|T|} Q_m(T) + \alpha \cdot |T|. \tag{13}$$

The above function $C_\alpha(T)$ is minimized by finding optimal tree T and thus we obtain a tree size of the final model as well. The regularization parameter α determines the trade-off between the overall residual sum of squares error and the complexity of the

model measured by the number $|T|$. High α penalizes more the bigger trees (trees with more terminal nodes), thus optimal tree will be relatively smaller and vice versa for small α. To estimate the optimal hyper-parameter α, cross-validation is usually used. [5]

For classification trees, the output variable y_i taking values $1, \ldots, K$ where K denotes a number of classes, as was already mentioned. The process of growing and pruning of classification tree is very similar to regression tree, only the sum of squares error functions is replaced by a more appropriate measure of error. These three measures, are according to [5], better measures of impurity :

$$\textit{Misclassification error} = \frac{1}{N_m} \sum_{i \in R_m} I(y_i \neq k(m)) = 1 - \hat{p}_{mk(m)}, \tag{14}$$

$$\textit{Gini index} = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}), \tag{15}$$

$$\textit{Cross-entropy} = -\sum_{k=1}^{K} \hat{p}_{mk} \cdot \log \hat{p}_{mk}. \tag{16}$$

Where $\hat{p}_{mk}$ denotes the proportion of class k observations in node m, given that node m represents region R_m with N_m observations

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k), \tag{17}$$

The function I or so called indicator function is equal to one if the argument is true. Then the class of all observations in node m will be classified according to the majority class given by $k(m) = argmax_k\ \hat{p}_{mk}$ in the node m.

All of the above measures are similar, though only Gini index and Cross-entropy are differentiable and thus more suitable to numerical optimization. Those two functions are also more sensitive to the node probabilities, which make them advantageous when growing the tree.

2.2.2 Bagging

Bagging or bootstrap aggregation is called a technique for reducing the variance of an estimated prediction function [5]. Decision trees are suitable for bagging as this technique works usually very well for high-variance, low-bias procedures, such as trees. Suppose training data $Z = \{(x_1, y_1), (x_2, y_2), \ldots, (x_N, y_N)\}$ and fitted model to those data predicting $\hat{f}(x)$ at input x. Indicator vector $\hat{f}(x)$ with predictions for K-class response with

single one and K-1 zeroes. Then vector $\hat{f}(x)^b$ is produced for each bootstrap sample Z^b which is drawn from training set, with replacement, and $b = 1, 2, \ldots B$ giving in total B different predictions based on different samples. Breiman in [7] also emphasizes that random inputs and random features produce good results in classification, but less so in regression.

In regression is the bagging estimate produced by averaging all the estimates based on bootstrapped samples, in classification is rather used „votes“ counting. Suppose tree based classifier $\hat{G}(x)$ for a K-class response, such that $\hat{G}(x) = argmax_k \ \hat{f}(x)$. Then the bagged estimate $\hat{f}_{bag}(x)$ is K-vector $p_1(x), p_2(x), \ldots, p_K(x)$ where $p_k(x)$ is equal to proportion of trees predicting class k at x. The selected class is then the one with the highest number of „votes“ picked by bagged classifier from the B trees.

Instead of exact classification, often is required to have the class-probability estimates at x. And even though proportions $p_k(x)$ are tempting to treat as estimates of these probabilities, it simply will not work as it shows in [5]. Suppose two-class classifier and true probability of class 1 at x is 0.75, and each of the bagged classifiers accurately predict a 1. Then $p_1(x)$ will be equal to 1, which is incorrect. However, there exists an underlying function $\hat{f}(x)$ for a number of classifiers $\hat{G}(x)$, which estimates the class probabilities at x. For trees, those are the class proportions in the terminal node. Instead of using the indicator vector, averaging those could improve the estimates of the class probabilities and especially for small B, it tends to produce bagged classifiers with lower variance.

2.2.3 Random Forests

After introducing both crucial parts of random forests, let's focus more on the method itself. The random forests combine or ensemble multiple techniques, which is the reason why it belongs among ensemble methods. The group of algorithms called ensemble has the goal to combine the predictions of several base estimators built with a given learning algorithm in order to improve robustness over a single estimator. Ensemble methods could be separated into two families – *averaging methods* and *boosting methods*. In averaging methods, the aim is to build several estimators independently and then average their predictions. On average, the combined estimator is usually better than any of the single base estimator as Breiman in [4] indicated. The improvement of predictions is caused by reduction in variance, which is often issue accompanying decision tree model. In boosting

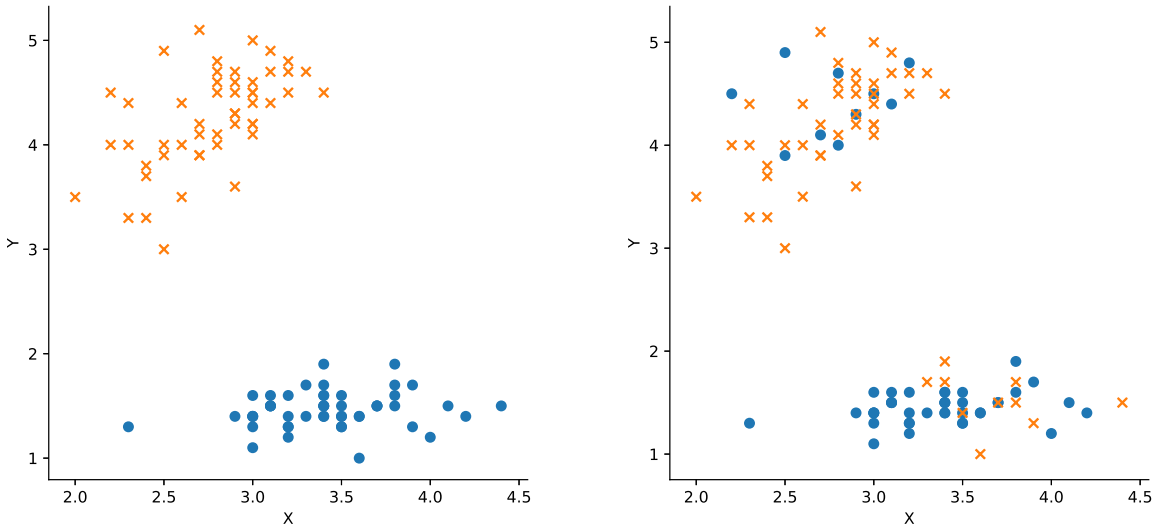
methods, base estimators are built sequentially and in order to reduce the bias of the combined estimator. The motivation is to combine several weak models to produce a powerful ensemble.

Breiman builds on his paper from 1996 [7] and use bagging in order to bootstrap samples out of the training data. Then in [4] follows on that by adding random input selection. Therefore bagging as well as random input selection are used to grown the trees. By adding the random variable selection is, instead of all input variables, used only low number of randomly generated variables to split at each node. Random forests using random variable selection enables to produce classifier relatively robust to outliers and noise, which is at the same time faster than bagging or boosting. Application of bagging has its reasons as well. It can be used to give ongoing estimates of classification error rate of the combined ensemble of trees and also estimates of correlation and strength as Breiman denotes in [4]. All of these estimates are done based on out-of-bag data, which is described further. Out-of-bag estimates are results of out-of-bag classifier trained on the data which was not drawn in the bagging process when training a particular tree. Therefore the out-of-bag estimate for the error rate is the error rate of the out-of-bag classifier on the training set. It is possible to think of it as an alternative to test set and generally cross-validation, which is in this case unnecessary. Breiman in [4] further supports the use of out-of-bag estimates due to that they are unbiased opposite to cross-validation. See that using bagging offers very convenient way of estimating generalization error.

Random forests is large collection of de-correlated trees, where each tree is trained on the bootstrapped data. In each node is selected random subset of input variables until minimal node size is reached. Out of that random subset is selected one variable giving the best split in the node and then the split is executed. When training the random forests classifier there are few parameters whose optimal value should be found by fine-tuning. Those are number of randomly selected inputs m, for classification usually chosen $m = \sqrt{p}$ where p is the total number of input variables or even $m = 1$. Possible, though very unlikely, are even higher values, depending on the situation. [5] Minimum node size, in classification usually recommended to use $n_{min} = 1$, but again should be treated as fine-tuning parameter and its optimal value depends on the problem.

2.3 Support Vector Machines

Support vector machines (SVM) is supervised learning method used for classification, regression and outliers detection. It was extensively developed by Vladimir Vapnik and his colleagues, who came up with generalization of this method. [8] The method was previously applicable only for linearly separable datasets, but due to the improvement, the method produces nonlinear boundaries which makes possible to work with linearly nonseparable data as well. The difference between linearly separable data and linearly non-separable data is illustrated on the Figure 4. In the case of two dimensional linearly separable data, there exists possibility to draw a line, that separates all the observations of the class. In non-separable data that is not possible, the classes overlap. The idea of the method presented in the paper from 1995 is as follows, input vectors are non-linearly mapped to very high dimension feature space. In the feature space is constructed linear decision surface and everything is reversely transformed back to original space.



(a) Linearly separable data

(b) Linearly non-separable data

Figure 4: Data separability (source: own processing)

In the following subsections will be described classifier for case of linearly separable data and its generalization to nonseparable data.

2.3.1 Maximal Margin Classifier

In SVM is operated with one crucial concept – *margin*. Margin is the smallest distance between decision boundary and any of the samples, see Figure 5. In this case, the opti-

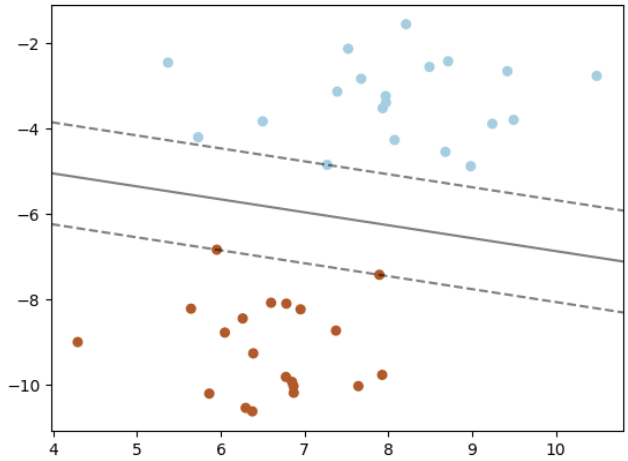


Figure 5: Optimal hyperplane (source: [9])

mization objective is to maximize margin, since bigger margin creates more robust decision boundary. As the motivation is to found maximal possible margin, therefore SVM is also called Maximal Margin Classifier.

Using [5], to denote it more formally. Given training data with N observations pairs $(x_1, y_1), \ldots, (x_N, y_N)$ with $x_i \in \mathbb{R}^{\mathrm{P}}$ and $y_i \in \{-1, 1\}$. Hyperplane defined by

$$f(x) = x^T\beta + \beta_0 = 0, \tag{18}$$

where β is unit vector and thus $||\beta|| = 1$. Then classification function using $f(x)$

$$G(x) = sign(x^T\beta + \beta_0). \tag{19}$$

One can quickly verify that output of $f(x_i)$ is > 0 when the data point i lies above the hyperplane and in opposite $f(x_i)$ is < 0 when the point i lies below the hyperplane. In this case are data linearly separable, thus the function $f(x) = x^T\beta + \beta_0$ for which apply $y_i f(x_i) > 0 \ \forall i$. Hence is possible to find hyperplane maximalizing the margin between the two output classes.

$$\begin{aligned} &\underset{\beta,\beta_0,\|\beta\|=1}{maximize}\ M \\ &\text{subject to } y_i(x^T\beta + \beta_0) \geq M,\ i = 1, \ldots, N. \end{aligned} \tag{20}$$

Or it could be rephrased, due to that margin $M = \frac{1}{\|\beta\|}$ as

$$\begin{aligned} &\underset{\beta,\beta_0,}{minimize}\ \|\beta\| \\ &\text{subject to } y_i(x^T\beta + \beta_0) \geq 1,\ i = 1, \ldots, N. \end{aligned} \tag{21}$$

The second form in equation (21) gives convex function opposite to non-convex function in the equation (20). As convex optimization is more efficient compared to non-convex, preferred is the second form.

This approach works well for linearly separable data, but it is a serious limitation as in many real-world problems linear separation between classes cannot be found. In that case could be used more robust method introduced by Cortes and Vapnik [8], named Soft Margin Classifier. This method tolerates to some extend outliers and noise in the data, but beside that is similar to Maximal Margin Classifier. Getting back to the previous equations, we allow some misclassifications on the training data by relaxing the constraint in equation (20) or (21). Lets define slack variables ξ_i where $\forall i,\ \xi_i \geq 0,\ \sum_{i=1}^{N} \leq constant$. Notice that misclassification occur when $\xi_i > 1$ and total number of misclassification is limited by some *constant* number. We can use those as a further cost and by doing so get

$$y_i(x^T\beta + \beta_0) \geq 1 - \xi_i\ \forall i. \tag{22}$$

Thus we can redefine the classifier from equation (21) as

$$minimize\ \|\beta\| \text{ subject to } \begin{cases} y_i(x^T\beta + \beta_0) \geq 1 - \xi_i,\ \forall i, \\ \xi_i \geq 0,\ \sum \xi_i \leq constant. \end{cases} \tag{23}$$

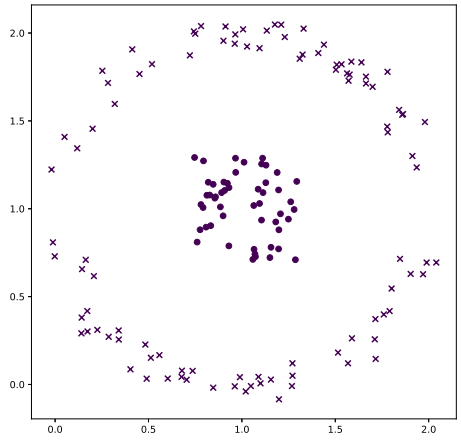
More thorough description can be found in [5], [10] or [8].

2.3.2 Support Vector Machines and Kernels

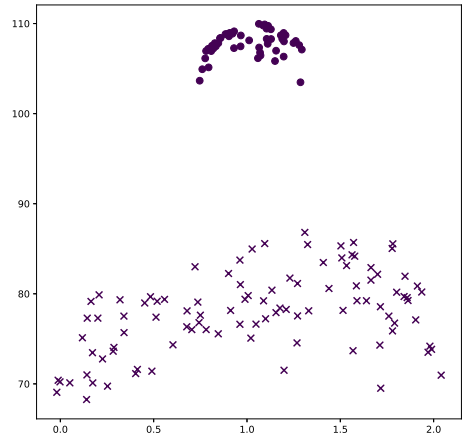
Algorithms covered in the previous section construct hyperplanes in the input feature space. To get a potentially better representation of the data, we can map the data points into an alternative higer-dimensional space. Then we need to transform the *n-dimensional* input vector x into an *N-dimensional* feature vector using some N-dimensional vector function ϕ [8]:

$$\phi : \mathbb{R}^{\mathrm{n}} \rightarrow \mathbb{R}^{\mathrm{N}}. \tag{24}$$

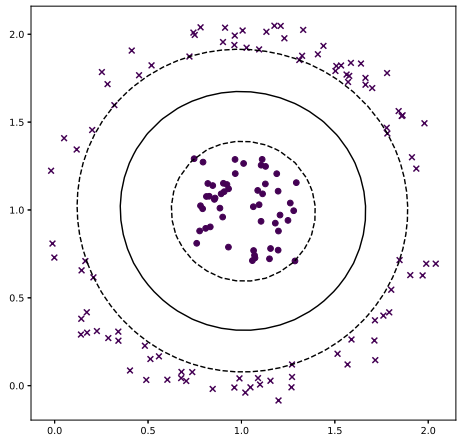
Consider these two classes shown on Figure 6(a), they are linearly nonseparable in the two dimensional space. After execution of transformation into higher-dimensional space, using radial basis function, see Figure 6(b), the separation is possible. The data are separated again by maximizing the margin and then transformed back to original space, Figure 6(c).



(a) Linearly non-separable data



(b) Data transformed to higher dimension using RBF



(c) Separated using SVM

Figure 6: Linearly non-separable example (source: [5]

As is showed in [5], there is no need to specify the transformation ϕ at all. There is only necessary to know the kernel function

$$K(x, x') =< \phi(x), \phi(x)' > , \tag{25}$$

which computes inner products in the transformed space. Worth to mention that K should meet certain conditions – it should be symmetric positive definite or semi-definite. These are popular choices for kernel function according to [5]:

$$\begin{aligned} \textit{dth-Degree polynomial: } K(x,x') &= (1+ < x, x' >)^d, \\ \textit{Radial basis (RBF): } K(x,x') &= exp(-\gamma||x-x'||^2), \\ \textit{Neural network: } K(x,x') &= tanh(\kappa_1 < x, x' > +\kappa_2). \end{aligned} \tag{26}$$

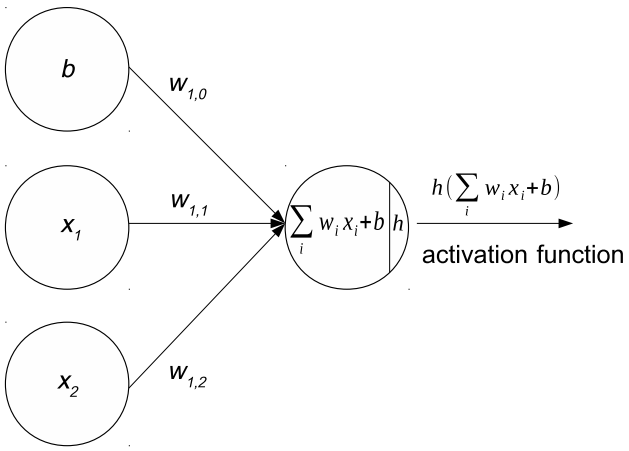
2.4 Neural Networks

Neural networks (NN) becomes more and more frequently used method nowadays as their dominance is spread across many different fields. The causes why are NN so popular today are almost identical within the literature. Jeremy Howard presents three key reasons of high relevance of NN. First, NN are in essence universal approximation machines. That was proved for NN with sigmoid activation function in [11] and in 1991 Kurt Hornik [12] showed that it is not the specific choice of the activation function which gives NN the potential of being universal approximators, but rather the multilayer feedforward architecture itself. Second, using gradient descend and backward propagation enables to fit any parameters given training data to do so. Then any theoretical function could be approximated and applied. Third, speed and scalability. As NN intesively uses matrix operations which could be done very fast using graphics processing units (GPU – part of the graphic card) compared to standard central processing unit (CPU). And mainly due to gaming industry we have solid computational power within the graphic cards for reasonable prices.

Neural networks were originally mainly inspired by the goal of modeling biological neural systems, therefore biological terminology. But achieving good results in machine learning tasks, they gained popularity among engineers too. NN are consisted of neurons (units), which are the basic building blocks. Neuron receives input signal x from other neurons (or as an input feature) and multiply it with its learnable weights w. This product is summed with bias b and if the sum is above certain threshold, then is produced output signal by activation function. See the single neuron model on the Figure 7, which is called a perceptron.

In case of we use sigmoid $\sigma(x) = 1/(1+e^{-x})$ as an activation function and appropriate loss function on the neuron's output, perceptron can be turned into a linear classifier. Then

Figure 7: Single neuron (source: own processing)

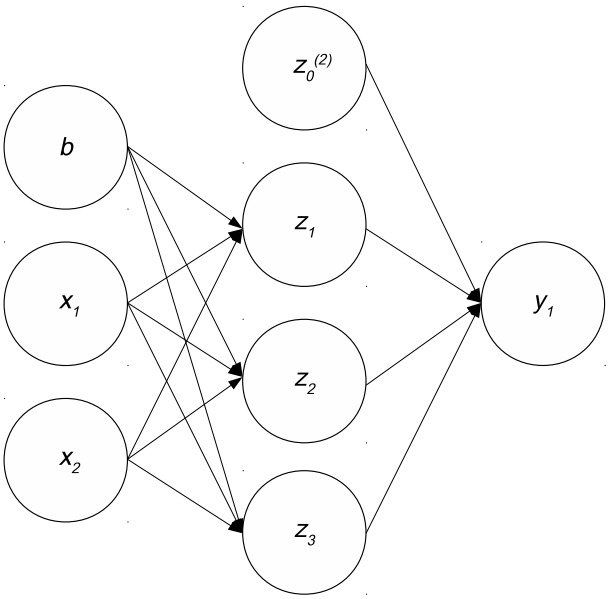


$\sigma(\sum_i w_i x_i + b)$ can be interpreted as the probability of one of the classes $P(y_i = 1|x_i; w)$ and thus probability of the other class would be $P(y_i = 0|x_i; w) = 1 - P(y_i = 1|x_i; w)$, since they must sum to one. Then using cross-entropy loss (mentioned in 2.2.1) and its optimization gives us the classifier. In essence, we recreated here logistic regression using simple NN, or we can use fancier name – Binary Softmax classifier.

But the real power of NN comes with the more complex networks. If we build on a single perceptron and create a whole network of perceptrons, we receive the Multi Layer Perceptron (MPL). That network can be represented by directed acyclic graph see Figure 8. The graph is constructed by layers, where we recognize the three basic types of layers: input layer, hidden layer and output layer. The network can have any number of hidden layers, but only one input and one output layer. More formally, considering the layers as functions enables us to rewrite the model from figure as this chain $f(x) = f^{(3)}(f^{(2)}(f^{(1)}(x)))$. Then input layer is $f^{(1)}$, hidden layer is $f^{(2)}$ and output layer is $f^{(3)}$. The overall length of this chain is depth of NN. Certain architectures can extend to many layeres, therefore is this field called deep learning. [1] Below is ploted a 2-layer fully connected NN (one hidden and one output layer), notice that in naming convention is not count with input layer.

See that flow of informations within both models is only in one way, there are no

Figure 8: A 2-layer Neural network (source: own processing)



feedback connections that would feed the output back into network. Thus, these models are called feedforward neural networks. During training is NN provided by input x with its label y, then goal is to get each output $f^*(x)$ of NN as much close as possible to its true label y. This is done by training the weights inside the network.

2.4.1 Activation functions

Activation functions provides model with non-linearity as it performs certain mathematical operation and computes the hidden layer values. Besides sigmoid, which is already described in the logistic regression part, there exist better choices. For example, tanh $g(z) = \frac{e^{2z}-1}{e^{2z}+1}$ which squashes real-valued number to (-1,1) interval. Tanh is preferred to the sigmoid due to that is zero centered. [13]

However, most widely used is non-linearization called Rectified Linear Unit (ReLU), considered as the default activation function. ReLU has fancy name, though its defined by simple function $g(z) = max(0, z)$. The function, even though is non-linear, is very close to linear transformation and therefore preserves many of its properties. Those properties make optimization using gradient-based methods easier and enable linear models generalize well, which are significant benefits. [1]

Moreover ReLU activation function makes training several times faster, as is shown in.[14]

Figure 9: Tanh (source: own processing)

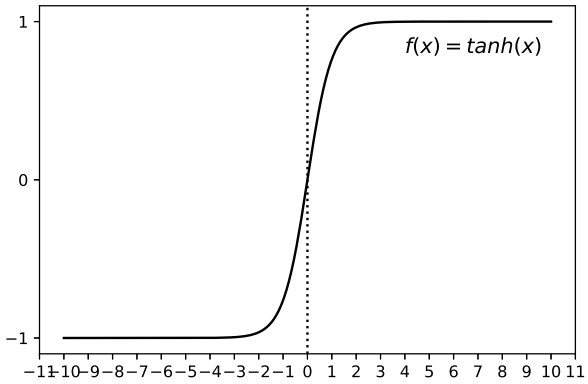
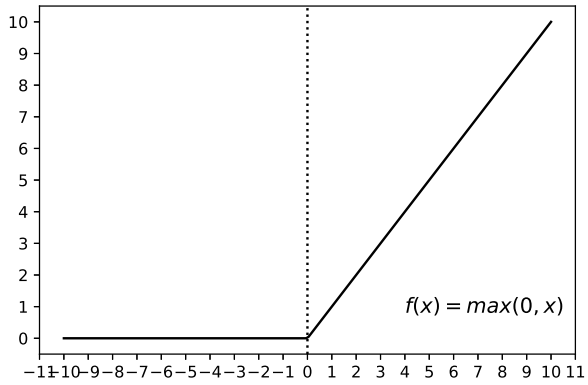


Figure 10: ReLU (source: own processing)



Though disadvantage can be higher sensitivity to training data. As with wrongly set up learning rate, the weights can be updated in such way that neuron will not activate on any given input again. The problem occurs when a substantial part of the network is „dead“. The probability of this issue is minimized by setting appropriate learning rate, which will be discussed in the next part. Ian Goodfellow with colleagues in [15] come with the generalized ReLU function that does not suffer by dying neurons issue. But on the other hand it lacks the simplicity of ReLU when significantly increases the total number of parameters.

2.4.2 Backpropagation

More than once was mentioned a learning of the weights, now lets look how it is done. By learning is meant finding the optimal weights and biases given some optimization criteria.

The term called backpropagation or shortly backprop has within the literature different meanings. In this thesis is term backpropagation used as reference to technique, which propagates the prediction error through the network and adjusts weights and biases, in order to improve the quality of the model.

Backpropagation is composed of two distinct stages. First, the forward stage where derivatives of an error function with respect to the weights and biases are evaluated. That gives us the gradient $\nabla E = [\frac{\partial E}{\partial w^{(1)}}, \frac{\partial E}{\partial b^{(1)}}, \frac{\partial E}{\partial w^{(2)}}, \frac{\partial E}{\partial b^{(2)}}, \cdots, \frac{\partial E}{\partial w^{(n)}}]$. Secondly, the backward stage where are computed adjustments to be made to the weights using the computed derivatives from the forward stage. [16]
Consider the following error function, which is for one observation calculated as squared difference of an output from the output layer L and a desired output y

$$E = \frac{1}{2}(\mathbf{z}^{(L)} - \mathbf{y})^2, \tag{27}$$

where $\mathbf{z}^l$ is vector of outputs in layer l and h is some non-linear activation function

$$\mathbf{z}^{(l)} = h(\mathbf{a}^{(l)}), \tag{28}$$

$\mathbf{a}^l$ is calculated as

$$\mathbf{a}^L = \begin{pmatrix} w_{1,1}^{(l)} & w_{1,2}^{(l)} & \cdots & w_{1,n}^{(l)} \\ w_{2,1}^{(l)} & w_{2,2}^{(l)} & \cdots & w_{2,n}^{(l)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,1}^{(l)} & w_{k,2}^{(l)} & \cdots & w_{k,n}^{(l)} \end{pmatrix} \begin{pmatrix} z_0^{(l-1)} \\ z_1^{(l-1)} \\ \vdots \\ z_n^{(l-1)} \end{pmatrix} + \begin{pmatrix} b_0^{(l)} \\ b_1^{(l)} \\ \vdots \\ b_k^{(l)} \end{pmatrix} \tag{29}$$

or using matrix and vectors notation, where $\mathbf{W}^{(l)}$ is matrix of weights (mapping function) from layer *l-1* to layer l and vector $\mathbf{b}^{(l)}$ is vector of biases in layer l

$$\mathbf{a}^{(l)} = (\mathbf{W}^{(l)}\mathbf{z}^{(l-1)} + \mathbf{b}^{(l)}). \tag{30}$$

In forward pass is for each unit computed a weighted sum of its inputs and summed with bias as showed above. In computation of the gradient is extensively used the chain rule for partial derivative. Gradient of the error function with respect to the weights and biases is computed as follows.

$$\frac{\partial E}{\partial \mathbf{w}^{(l)}} = \frac{\partial E}{\partial \mathbf{z}^{(l)}} \frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{a}^{(l)}} \frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{w}^{(l)}}, \tag{31}$$

and by each part

$$\frac{\partial E}{\partial \mathbf{z}^{(l)}} = (\mathbf{z}^{(l)} - \mathbf{y}), \tag{32}$$

$$\frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{a}^{(l)}} = h'(\mathbf{a}^{(l)}), \tag{33}$$

$$\frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{w}^{(l)}} = \mathbf{z}^{(l-1)}, \tag{34}$$

and with respect to biases in layer l

$$\frac{\partial E}{\partial \mathbf{b}^{(l)}} = \frac{\partial E}{\partial \mathbf{z}^{(l)}} \frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{a}^{(l)}} \frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{b}^{(l)}}. \tag{35}$$

Then applying same logic for previous layers and chaining the above terms, we can obtain the whole gradient ∇E

$$\frac{\partial E}{\partial \mathbf{w}^{(l-1)}} = \frac{\partial E}{\partial \mathbf{z}^{(l)}} \frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{a}^{(l)}} \frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{z}^{(l-1)}} \frac{\partial \mathbf{z}^{(l-1)}}{\partial \mathbf{a}^{(l-1)}} \frac{\partial \mathbf{a}^{(l-1)}}{\partial \mathbf{w}^{(l-1)}}. \tag{36}$$

However, above calculations are illustrated for one observation. Including the whole set of observations, partial derivations are computed as an average over all observations in the set

$$\frac{\partial E}{\partial \mathbf{w}^{l}} = \frac{1}{n} \sum_{k=1}^{n} \frac{\partial E_k}{\partial \mathbf{w}^{l}}. \tag{37}$$

and similar for error function, which will be sum of squared errors

$$E = \frac{1}{2} \sum_{n} (\mathbf{z}_n^{(L)} - \mathbf{y}_n)^2, \tag{38}$$

Here comes the second stage, while having all the partial derivations calculated, one have information how nudging the weights or biases affect the error function. Those informations are contained in gradient ∇E. In propagating the error from output layer L through the network is used those derivatives. Firstly, lets introduce $\delta^{(L)}$ as difference in output of each unit and its desired value

$$\delta^{(L)} = (z_n^{(L)} - y_n) \odot h'(\mathbf{a}^{(L)}), \tag{39}$$

then those errors are propagated as follows

$$\delta^{(l-1)} = (\mathbf{W}^{(L)})^T \delta^{(L)} \odot h'(\mathbf{a}^{(L-1)}). \tag{40}$$

where $\odot$ denotes Hadamard product, defined as

$$\begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} \odot \begin{pmatrix} t_1 \\ t_2 \\ t_3 \end{pmatrix} = \begin{pmatrix} v_1 \cdot t_1 \\ v_2 \cdot t_2 \\ v_3 \cdot t_3 \end{pmatrix}. \tag{41}$$

3 Development of Models

In this section are described the crucial steps of development of models explained in previous part. For the purpose of this thesis, two datasets were used – data from peer-to-peer lending platform Lending Club [17], which contains more than million observations. This dataset was simplified by random sampling down to 200 000 observations, due to substantial requirements on time and computational power. And second dataset from Kaggle - platform for data science competitions - consisting of 150 000 observations [18]. The empirical part was created using Python programming language [19] and its libraries, from which worth to mention Scikit-learn [20], Pandas [21], Numpy [22] and Keras [23]. The source codes are available in github repository along with further description of features from both datasets, both available on this address: `https://github.com/ryparmar/bachelor-thesis`.

3.1 Data Wrangling

When dealing with some real-world dataset, there is in most of the cases necessary to execute certain steps related to data itself before any further work on modelling. Those steps can be cleaning the data from missing values or transformation of the data into different format. A set of those steps is usually called a data wrangling or data munging and it is an inseparable piece of any reasonable work with data.

Usually first, explanatory data analysis is executed in order to get some intuition about the data and uncover potential shortcomings or clear mistakes within the dataset. To reveal those mistakes, outliers detection can be used, as there is usually always some expected range in which should a number get in. An example of clearly incorrect data record can be age of 500, which is several orders of magnitude from expected value and reality as well. Or more general approach is to count missing values per observation and per feature, then if the number is above certain level of threshold remove the whole observation or feature. Those actions result in removing some features with no information value – for example index columns – and observations with evident mistakes in order to improve data quality for modelling.

In the first part are found some missing data in both datasets. In case of Lending club dataset, which have almost 150 features in total were all the features missing more than

20 % of records removed. Similarly, each observation with more than 30 missing values are removed. This approach may be radical, however it is possible. After this cleaning lending club dataset contains approximately 140 000 observations. In kaggle dataset is chosen different approach as the number of features is much lower. If observation contains more than one missing feature then is removed, rest of the missing data are replaced by median. The replacement of missing data by median affects only one feature at the end – monthly income. Total amount of observation from previous 150 000 is after execution of above steps approximately 146 000.

Next step is encoding of qualitative (or categorical) features, into more convenient form for learning algorithms. Each observation of such features is labeled by some discrete number. Consider sex for example, there are two possible values – "male" labeled as 1 and "female" labeled as 0. Though, there is often no ordinal relationship within the features, which apply for sex variable as well. Therefore there is no obvious reason to have higher number for male than for female. A preferred choice is to binarize labeled observations, so the "male" is represented by an array [1, 0] and "female" by [0, 1]. Beside appropriate representation of categorical variables, the machine learning estimators usually requires standardization of quantitative variables. In case of significant difference to normally distributed data for the individual features, the estimators might behave badly and have lower predictive ability opposite to case with normalized data. Therefore quantitative variables in both datasets are standardized into [-1, 1] range.

Since the probability of total repayment converge very fast to zero in case of borrower is past due with his or her payments, one can approach it with certain flexibility and define the default state by himself. In the kaggle dataset is the outcome variable – serious delinquency in two years – defined as whether a person experienced 90 days (or more) past due delinquency or not. In the case of lending club dataset is a default loan defined as a loan which is not up to date on all outstanding payments for more than 120 days. [24]

3.2 Cross-Validation, Regularization and Hyper-parameters

Cross-validation (CV) is called a widely used process for estimating prediction error. The ability of model to be as accurate as possible not only on training data, but more importantly on the yet-unseen data is in real-world applications absolutely crucial. Therefore,

CV can be perceived as a way of simulating exposition to yet-unseen data. The process is as follows – based on data splitting, part of the data is used for fitting each competing model and the rest of the data is used to measure predictive performances of the models by calculating validation errors. The model with the best overall performance is selected.[5]

Executing CV has further benefits, the very significant one is it minimizes a chance of overfitting the model. Consider the situation, that model has very good performance on training data, but poor performance on yet-unseen data, in this situation the model is probably overfitted.

Overfitting is highly undesirable situation as one does not intend to develop a model, which perfectly predicts what is already known, but is useless in case of predicting with new data. In other words, the model should be able to generalize well.

It is a common practice in supervised learning to separate the dataset into a training set and a validation set, where the actual splitting is possible to do in many ways. The simplest approach would be to separate the dataset into two parts, train the model on a first part and calculate the test error on the remaining part. An optimal splitting ratio between train set and test set depends on the situation, usually 70% – 80% training set and rest for testing works fine.

Another often used approach is K-Fold cross-validation, more appropriate choice in case of scarce data. We split dataset into K parts of roughly equal size, for example $K = 10$ and iterate through the data, where in each iteration the test set is changed as depicted below. This approach generates 10 separate folds of predictions with prediction errors, which are averaged in order to get a final test error. **In this thesis were data randomly split to 80% for training and validation set and 20% for testing. For finding hyper-parameters was used 10-fold cross-validation.**

Test set and validation set are sometimes used interchangeably, however it is not completely correct approach. Ideally, the model should be evaluated on a sample, that was not used in training nor the parameter tuning, so it can be reached unbiased performance evaluation. Dividing dataset into training, validation and test set provide that unbiased evaluation. Comprehensible definition of training set, validation set and test set taken from [25] is as follows:

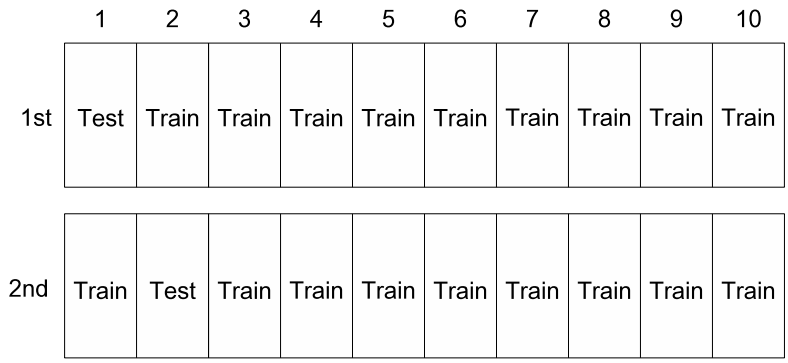
1. *Training Dataset* the sample of a dataset used in training to fit the model

2. *Validation Dataset* the sample of a dataset used to during tuning model hyper-

parameters to provide an unbiased evaluation of a model fit on the training data

3. *Test Dataset* the sample of a dataset used to give an unbiased evaluation of final model fit on the training dataset

Figure 11: 10-fold splitting for first two iterations (source: own processing)



Other possibilities how to avoid overfitting are reducing the number of used features or so called regularization. Regularization keeps all the features, but penalizes the complexity of a model by penalizing weights in proportion to weights size. So the bigger weights are penalized more compared to smaller weights. Consequence is that lower number of features affects the number of weights and thus also model complexity. As a result, an ideal model tends to use all the features a bit instead of using only a few ones a lot. Regularization is deployed in optimization algorithms searching for optimal hyper-parameters and weights of the model. It has form of regularization parameter increasing some cost function in proportion with growing model complexity.

Usually a model have some hyper-parameters, which affect training process and thus can affect predictions as well. There is difference between hyper-parameters and parameters of model, where the hyper-parameters need to be set up before training and can not be changed during training. In case of using same algorithm with different hyper-parameters, there is a solid chance of having different predictions within those two models. Compared to parameters of model, or weights how it is also called, which are changed during a training process based on optimization algorithm.

Hyper-parameters tuning was done using scikit-learn library, particularly by GridSearchCV and RandomizedSearchCV functions. Both functions search a predefined grid of parameters and try either all the combinations (GridSearchCV) or some given number of randomly chosen combinations (RandomizedSearchCV). Based on that process the best performing combination of parameters is chosen for a given model. For neural networks exist very elegant way of reducing overfitting by randomly omitting a certain fraction of the both hidden and visible units. This technique is called dropout and was introduced by Hinton in [26]. Another way is to make architecture simpler by reducing number of units in hidden layers. For tree-based model can be set for example maximal number of features considered when looking for the best split or maximal depth of tree, which help to prevent overfitting.

Regularization parameter, mentioned earlier, can be considered as a hyper-parameter likewise. For logistic regression was tuned parameter C which is inverse of regularization parameter. Similarly for SVM was tuned identical parameter C For random forest model are besides maximal depth of tree and maximal number of features fine-tuned also minimum number of samples required to split an internal node, minimum number of samples required to be at leaf node, number of trees in the forest and measure of quality of a split function.

Since it was not intention to provide description of above techniques in its full complexity, dedicated space to those steps is limited and offer only basic explanation. However the importance of those steps is substantial and their correct execution can have significant effect on final performance of models.

3.3 Variable Selection

One can think of a selecting variables into the model as a marginal part, though it is exactly opposite. Not all the features necessarily bear relevant information, but they can made the predictions even worse. Without feature selection exist higher chance of introducing bias into a model, which can result in overfitting and thus worse performance of model. There are three objectives of feature selection – improve the prediction performance of the model, provide faster and more cost-effective models and provide better understanding of the underlying process that generated the data.[27]

There exist several classes of methods to select important variables. Wrapper methods

– searching through the set of different combinations of features, evaluating and comparing them to each other. As evaluating all the possible combinations might be hardly feasible, depending on the number of features, the searching may use some heuristic method or stochastic method. Often used example of wrapper is recursive feature elimination algorithm. The second class is filtering methods. Those methods select the best features based on their scores in various statistical tests for their correlation with the outcome variable. Last class is embedded methods, which are more complex and combines qualities of the previous methods.

However, not all the methods described in the previous part can benefit from feature selection. Decision trees perform internal feature selection as an integral part of the procedure. Therefore they are resistant to the inclusion of many irrelevant variables. Similarly neural networks find the relevant and irrelevant variables in the training process internally, which is done by learning appropriate weights for them. In this thesis, feature selection was deployed for both logistic regression and SVM where the categorical features were chosen based on filtering methods using Chi-Square test. Numerical variables were chosen using random forest method measuring importance.

3.4 Evaluation Metrics

For the purpose of comparison of the predictive quality of classification models, there exist a number of different, more or less convenient, metrics. Further are covered only the metrics used in this thesis. The very first and basic method is simply calculate accuracy of predictions as

$$accuracy = \frac{\#\text{correct predictions}}{\#\text{total predictions}}. \tag{42}$$

Accuracy is very quick and easy metric, however in case of a classification task, where one of the possible outcomes occurs rarely opposite to another outcome, can lead to distortion. Consider an example of 1000 observations of given loans, where 50 of them are defaulted and the rest are non-defaulted. Then a model which would predicts that all the customers (1000 observations) are non-defaulted, would obtain accuracy of 95%, which is not bad at all. But the model is completely useless as it does not help with the classification of those defaulting customers. Therefore it is more convenient to use different more informative metrics.

More appropriate metrics are so called *recall score* and *precision score*. To illustrate

those terms, confusion matrix is very helpful in doing so.

Figure 12: Confusion matrix (source: own processing)

		Prediction	
		Negative	Positive
Actual	Negative	True negative (TN)	False positive (FP)
	Positive	False negative (FN)	True positive (TP)

Then recall and precision are calculated as

$$recall = \frac{\text{TP}}{\text{total actual positives}} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \tag{43}$$

$$precision = \frac{\text{TP}}{\text{total predicted positives}} = \frac{\text{TP}}{\text{TP} + \text{FP}}. \tag{44}$$

Precision shows what percentage of predicted positives are actual positives and recall denotes percentage of positives which a model predicted correctly out of total actual positives. Hand in hand with recall and precision is usually calculated *F1 score*, which can be interpreted as a weighted average of precision and recall.

$$F1 = \frac{2 * \text{precision} * \text{recall}}{\text{precision} + \text{recall}} \tag{45}$$

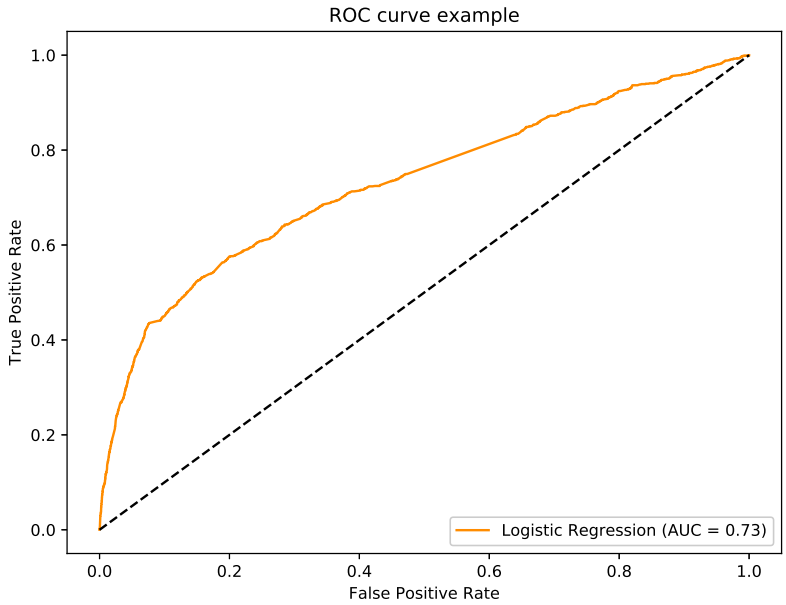
The last used evaluation metrics are Receiver Operating Characteristic (ROC) and Area Under Curve (AUC). A ROC curve is plotting True Positive Rate (TPR) against False Positive Rate (FPR), where

$$TPR = \frac{\text{TP}}{\text{TP} + \text{FN}}, \tag{46}$$

$$FPR = \frac{\text{FP}}{\text{FP} + \text{TN}} \tag{47}$$

and the more the ROC have a leaning towards the upper-left corner where TPR = 1, the better quality of the model. AUC is just the area under the ROC curve, so for a perfect model is AUC = 1 and completely random guessing would have AUC = 0.5.

Figure 13: ROC curve (source: own processing)



For all above mentioned metrics apply that values gain number from interval [0, 1] and the higher the value is the better the model is. Above mentioned metrics are only a few representatives of wide class of evaluation metrics and besides them, it is possible to define evaluation metrics by yourself, in case of need.

3.5 Empirical Results

In this subsection are presented evaluations of trained models. For both datasets apply that data were randomly split in two parts 80% for training and validation and 20% for testing. Then on the first part were models trained and validated using 10-fold cross-validation to obtain hyper-parameters. Following results are calculated on the testing part of the data. After split of the dataset, upsampling algorithm SMOTE [28] was applied in order to get output classes approximately equally represented in the training and validation splits. This technique can improve training on imbalanced datasets that learnt patterns can better differentiate the output classes and thus obtain better prediction accuracy.

Regarding the dataset from Kaggle, which contains only 10 numerical features. For SVM was used linear kernel due to a substantial computational complexity of this algorithm. For feed forward neural network was used fully connected architecture with one

Method	List of optimized hyper-parameters
Logistic Regression	C
Neural Network (MLP)	number of hidden units, weight initialization, dropout, batch size
Random Forest	number of estimators, maximal depth, minimum number of samples to split, minimum number of samples in leaf, maximum number of features
Support Vector Machines	C

Table 1: Optimized hyper-parameters

hidden layer consisting of 512 hidden units, applied 50% dropout and ReLU activation. Output layer with a single output unit and sigmoid activation. Model was optimized using root mean square prop (RMSprop) with binary crossentropy loss function and was trained in 15 epochs.

Method	Accuracy	Precision	Recall	F1 score	AUC
Logistic Regression	76.4	17.4	65.4	27.5	71.3
Neural Network (MLP)	80.9	21.9	69.2	33.2	**75.5**
Random Forest	92.3	42.5	33.5	**37.5**	65.1
Support Vector Machines	63.5	11.7	66.3	19.9	64.8

Table 2: Results on Kaggle Dataset

Regarding dataset from lending club, for SVM, logistic regression and neural network was used feature selection, where was selected top 27 features based on feature importance reached in random forest. Models with notation wo feature selection were trained with 76 features. For random forest is feature selection unnecessary as the method can choose relevant features internally. For SVM was again used linear kernel because of computational complexity. Neural network consists of one hidden layer with 10 units, 50% dropout, ReLU activation function and single output unit with sigmoid activation. Model was optimized using root mean square prop (RMSprop) with binary crossentropy loss function and trained in 15 epochs.

Method	Accuracy	Precision	Recall	F1 score	AUC
Logistic Regression wo FS	67.4	17.9	72.7	**28.8**	**69.8**
Logistic Regression with FS	64.3	16.8	74.5	27.4	68.9
Neural Network (MLP) wo FS	82.0	21.1	38.0	27.2	62.6
Neural Network (MLP) with FS	82.0	21.1	38.4	27.3	62.3
Random Forest	85.7	18.0	30.0	22.5	58.2
SVM wo FS	66.7	17.7	73.4	28.5	69.7
SVM with FS	63.5	16.6	75.3	27.2	68.9

Table 3: Results on Lending Club Dataset

Conclusion

Artificial intelligence (AI) and machine learning (ML) are again widely used and popular terms nowadays. Mainly due to combination of factors like exponential increment of data in the world, hardware development and universal applicability of neural networks. The range of possible applications of AI and ML methods is very wide, thus their penetration into world of finance is not a surprise. This thesis focuses on application of those methods in credit scoring modelling. The score for each applicant can be calculated using age, income, purpose of the loan, previous delinquency and many other variables, then based on a reached score the bank decide whether to lend the money or not.

In this thesis were described some of the methods from wide family of artificial intelligence techniques. Particularly random forest, neural network (multilayer perceptron), support vector machines and industry standard logistic regression. All methods were implemented for two public datasets – dataset from online lending platform Lending Club and dataset from online data science platform Kaggle.

Results of this thesis showed that artificial intelligence methods can outperform logistic regression, however results are not consistent across datasets. Comparison is based on F1 score and AUC metrics. Despite the fact that the results reached in this thesis do not imply any general validity, the artificial intelligence methods should be implemented along with the industry standard methods mainly because of two reasons. Firstly, they can often increase accuracy of predictions and secondly they can work as a quality check of currently using methods.

In future research would worth to examine entity embeddings of categorical variables [29] and its potential in probability of default modelling.

References

[1] Goodfellow Ian, Yoshua Bengio, and Aaron Courville. *Deep Learning*. Available from: http://www.deeplearningbook.org. MIT Press, 2017. ISBN: 978-0-262-03561-3.

[2] David Silver et al. „Mastering the game of Go without human knowledge“. In: *Nature* 550.7676 (2017), p. 354. URL: https://www.gwern.net/docs/rl/2017-silver.pdf.

[3] Witzany Jiří. *Credit risk management - Pricing, Measurement, and Modeling*. Švýcarsko: Springer International Publishing, 2017. ISBN: 978-3-319-49799-0.

[4] Leo Breiman. „Random Forests“. In: *Machine Learning* 45.1 (2001), pp. 5–32. ISSN: 0885-6125. DOI: 10.1023/A:1010933404324. URL: https://link.springer.com/content/pdf/10.1023/A:1010933404324.pdf.

[5] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning : data mining, inference, and prediction*. Springer New York Inc., 2009. ISBN: 978-0-387-84857-0.

[6] Leo Breiman. „Classification and Regression Trees“. In: (1984).

[7] Leo Breiman. „Bagging Predictors“. In: *Machine Learning* 24.2 (1996), pp. 123–140. ISSN: 0885-6125. DOI: 10.1007/BF00058655. URL: https://www.stat.berkeley.edu/%7B~%7Dbreiman/bagging.pdf.

[8] Corinna Cortes and Vladimir Vapnik. „Support-vector networks“. In: *Machine learning* 20.3 (1995), pp. 273–297. URL: https://link.springer.com/content/pdf/10.1007/BF00994018.pdf.

[9] G. Bradski. *The OpenCV Library, Introduction to Support Vector Machines*. [online]. [cit 21-March-2018]. Available from: "www.docs.opencv.org/2.4/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html".

[10] Christopher JC Burges. „A tutorial on support vector machines for pattern recognition“. In: *Data mining and knowledge discovery* 2.2 (1998), pp. 121–167.

[11] G Cybenkot. „Approximation by Superpositions of a Sigmoidal Function“. In: *Math. Control Signals Systems* 2 (1989), pp. 303–314. URL: https://pdfs.semanticscholar.org/05ce/b32839c26c8d2cb38d5529cf7720a68c3fab.pdf.

[12] Kurt Hornik. „Approximation capabilities of multilayer feedforward networks“. In: *Neural networks* 4.2 (1991), pp. 251–257.

[13] Andrej Karpathy. *Stanford University CS231n: Convolutional Neural Networks for Visual Recognition*. [online]. [cit 24-February-2018]. Available from: "https://cs231n.github.io/".

[14] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. „Imagenet classification with deep convolutional neural networks“. In: (2012), pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[15] Ian J Goodfellow et al. „Maxout networks“. In: *arXiv preprint arXiv:1302.4389* (2013).

[16] Bishop Christopher M. *Pattern Recognition and Machine Learning*. Springer New York Inc., 2006. ISBN: 978-0387-31073-2.

[17] Lending Club. *All Lending Club Loan Data*. [online]. [cit 16-February-2018]. Available from: "https://www.lendingclub.com/info/download-data.action".

[18] Kaggle. *Kaggle competition dataset: Give Me Some Credit*. [online]. [cit 16-February-2018]. Available from: "https://www.kaggle.com/c/GiveMeSomeCredit/data".

[19] Python Software Foundation. *Python Language Reference, version 3.6*. [online]. [cit 21-March-2018]. Available from: https://www.python.org/.

[20] F. Pedregosa et al. „Scikit-learn: Machine Learning in Python“. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[21] Wes McKinney. „Data Structures for Statistical Computing in Python“. In: *Proceedings of the 9th Python in Science Conference*. Ed. by Stéfan van der Walt and Jarrod Millman. 2010, pp. 51–56.

[22] Travis E Oliphant. *A guide to NumPy*. Vol. 1. Trelgol Publishing USA, 2006.

[23] François Chollet et al. *Keras*. [online]. [cit 5-March-2018]. Available from: "https://github.com/fchollet/keras".

[24] Lending Club. *What do the different Note statuses mean?* [online]. [cit 10-April-2018]. Available from: "https://help.lendingclub.com/hc/en-us/articles/215488038-What-do-the-different-Note-statuses-mean-".

[25] Jason Brownleen. *What is the Difference Between Test and Validation Datasets?* [online]. [cit 1-March-2018]. Available from: `https://machinelearningmastery.com/difference-test-validation-datasets`.

[26] Geoffrey E Hinton et al. „Improving neural networks by preventing co-adaptation of feature detectors“. In: *arXiv preprint arXiv:1207.0580* (2012).

[27] Isabelle Guyon and André Elisseeff. „An introduction to variable and feature selection“. In: *Journal of machine learning research* 3.Mar (2003), pp. 1157–1182.

[28] Nitesh V Chawla et al. „SMOTE: synthetic minority over-sampling technique“. In: *Journal of Artificial Intelligence Research* 16 (2002), pp. 321–357.

[29] Cheng Guo and Felix Berkhahn. „Entity Embeddings of Categorical Variables“. In: *arXiv preprint arXiv:1604.06737* (2016).

[30] Vincenzo Pacelli and Michele Azzollini. „An artificial neural network approach for credit risk management“. In: *Journal of Intelligent Learning Systems and Applications* 3.02 (2011), p. 103.

[31] Gang Wang et al. „A comparative assessment of ensemble learning for credit scoring“. In: *Expert systems with applications* 38.1 (2011), pp. 223–230.

[32] Bušo Bohumír. *Porovnanie metód machine learningu pre analýzu kreditného rizika*. Diplomová práce. Vysoká škola ekonomická v Praze, Fakulta financí a účetnictví, katedra bankovnictví a pojišťovnictví. Praha, 2016.

[33] Jan Říha. *Artificial Intelligence Approach to Credit Risk*. Diplomová práce. Univerzita Karlova v Praze, Fakulta sociálních studií, Institut ekonomických věd. Praha, 2016.

[34] Rachel Thomas Jeremy Howard. *Fast ai lesson 1 notes*. [online]. [cit 4-March-2018]. Available from: `http://wiki.fast.ai/index.php/Lesson_1_Notes`.

[35] Michael Nielsen. *How the backpropagation algorithm works*. [online]. [cit 22-March-2018]. Available from: `http://neuralnetworksanddeeplearning.com/chap2.html`.