FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

ASSIGNMENT OF BACHELOR’S THESIS

Title:	Machine learning for earthquake prediction and forecasting
Student:	Georgii Korostii
Supervisor:	MSc. Juan Pablo Maldonado Lopez, Ph.D.
Study Programme:	Informatics
Study Branch:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	Until the end of summer semester 2019/20

Instructions

Earthquake prediction is a notoriously complicated problem, but at the same time, it has enormous and long-lasting consequences in the affected regions. In this work we survey the methods for earthquake prediction (specification of time, location and magnitude) and forecasting (probabilistic assessment of earthquake hazard).

1) Study the problem of earthquake prediction and forecasting.
2) Discuss advantages and drawbacks of different algorithms.
3) Explore various proposed approaches (e.g. neural networks).
4) Compare the performance (classification/training time/scoring time) of the proposed approaches.
5) Consider possible optimizations and improvements.
6) Implement the improved method and analyze achieved results.

References

Will be provided by the supervisor.

Ing. Karel Klouda, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague February 4, 2019

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor’s thesis

Machine Learning for Earthquake Prediction and Forecasting

Georgii Korostii

Department of Applied Mathematics
Supervisor: MSc. Juan Pablo Maldonado Lopez, Ph.D.

May 15, 2019

Acknowledgements

I would like to thank my supervisor Juan Pablo Maldonado Lopez for assistance during the work on the current thesis, for helping in all hardships that have met me on my way and for the great support.

Declaration

In Prague on May 15, 2019 . . . .

Citation of this thesis

Korostii, Georgii. *Machine Learning for Earthquake Prediction and Forecasting.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2019.

Abstrakt

V rámci této bakalářské práce bude definován a popsán problém predikce a prognózování zemětřesení. Hlavním cílem je prezentace našeho přístupu k řešení tohoto problému. Také zjistíme, jak fungují základní algoritmy strojového učení a zanoříme se do experimentování s neuronovými sítěmi, které budou použity pro prognózování budoucích signálu a také pro predikci zemětřesení.

Klíčová slova průběh signálu, predikce zemětřesení, prognózováni zemětřesení, konvoluční neuronové sítě, rekurentní neuronové sítě, výkon, matice záměn

Abstract

In the current thesis, the problem of earthquake prediction and forecasting will be defined and described. The main goal is to present our approach to solving that. In this paper we will describe how basic Machine Learning algorithms work and dive into experimenting with neural networks, that further will be used for forecasting the signal readings and earthquake prediction.

Keywords waveform, earthquake prediction, earthquake forecasting, convolutional neural networks, recurrent neural networks, performance, confusion matrix

Contents

Introduction **1**

Goals of the Thesis **3**

1 Theoretical background **5**

1.1 Theory Behind an Earthquake 5
1.2 Earthquake Prediction and Forecasting 6
1.3 Classification 6
1.4 K-Nearest Neighbors 7
1.5 Naive Bayes 10
1.6 Decision Tree Classifier 11
1.7 Random Forest Classifier 12
1.8 Rotation Forest Classifier 13
1.9 Logistic Regression 14
1.10 Performance measure 15
1.11 Neural Networks 17

2 Solution of the Problem **23**

2.1 Gathering Required Data 23
2.2 Baseline Modeling 26
2.3 Neural Networks 26
2.4 Performance 30

Conclusion **33**

Bibliography **35**

A Acronyms **39**

B Contents of enclosed flash disk **41**

List of Figures

1.1 K-NN in action. 8
1.2 Logistic function 14

List of Tables

2.1 The first letter of channel deciphered. 24
2.2 The second letter of channel deciphered. 25
2.3 The third letter of channel deciphered. 25
2.4 Accuracies of the different baseline models. 26
2.5 Prediction accuracy. 31
2.6 Forecasting performance. 31

Introduction

First documented earthquakes are known to have occurred thousands of years ago and they keep happening nowadays. Most of them are not significant enough to cause severe damage but some had caused terrible and often deadly repercussions. Therefore, the problem of their prediction remains actual throughout the whole human history.

Many attempts have been made to finally create an algorithm to put an end to the suddenness of these events and prepare for the serious earthquakes. Since the lack of data, the issue seems to be difficult to solve. The problem is, that serious earthquakes happen a lot more rarely, hence, there is not much information we can retrieve about circumstances of this phenomenon.

Scientists keep digging into this issue. Whilst ones say that all discoveries that have been made are the product of juncture, others remain optimistic and invent more ways to achieve the result. In attempts to determine causal relationship between earthquakes and factors that lead to them, experts explore seismicity patterns, weather conditions, electromagnetic fields and even animals behavior (according to Grant and Halliday [1] toad *Bufo bufo* is believed to have pre-seismic anticipatory behavior).

In the next chapter the goals of the thesis will be discussed. This paper is based on the research of all possible algorithms for earthquake prediction and forecasting that results in personal implementation. In order to conceptualize conclusions, it is vital to have appropriate knowledge about the problem, which will be acquired in Chapter 1 - chapter about the theoretical background. Next, in Chapter 2 some practical approaches will be discussed alongside with data gathering. In conclusion, the last chapter will put all pieces together and tells if work that has been performed achieved all goals that were outlined.

Goals of the Thesis

The goal of this thesis is to explore various possible solutions for earthquake prediction and forecasting resulting in personal realization and performance assessment. Different algorithms will be discussed alongside with their advantages and disadvantages.

After understanding all problems with mentioned algorithms, personal solution will be suggested together with data and performance analysis. Also, the rationality of offered solution will be covered in the end of current paper.

Chapter 1

Theoretical background

This chapter gives an understanding of the processes behind earthquakes and approaches that different researchers tried studying earthquakes prediction and forecasting. Also, it contains various theoretical information about all algorithms that have been used for experiments and explains our approach to solving this difficult problem.

1.1 Theory Behind an Earthquake

Earthquakes happen every day in different places on Earth. Major of them are weak and/or occurring far away from inhabited areas. In these cases, they may cause damage that is not serious enough to draw concern. However, occasionally destructive ones happen that can take many lives and cause billion dollars material losses for the countries' economies.

People started to study earthquakes long ago. Our ancestors had various explanations for ground movements – giant snakes or turtles that live underneath the ground create these movements, they believed. But thoughout the whole human history, scientists moved forward in understanding this phenomenon. [2]

Earthquake is shaking caused by the rupture and displacement of rocks beneath the Earth's surface. [3] When stress occurs, rock deforms and when deformation strength is too high, the rock breaks and two sides slide past each other. Most earthquakes take place near plate boundaries but not necessarily right on the boundary. The magnitude of an earthquake depends on the extent of the area that breaks and the average amount of displacement. Rupture does not happen at once, it starts from a particular point and spreads rapidly from there.

Earthquake, after it had ended, can cause a series of aftershocks. They are earthquakes just like any other but they can only be triggered by stress transfer from a preceding earthquake. There can be hundreds of them but they mostly have a much lower magnitude than the main shock, but sometimes they can have higher. Aftershocks can happen seconds or minutes after the preceding earthquake as well as hours, days, months or even years. [3] As was already mentioned, earthquakes cause a different kind of damage. They can cause buildings to be destroyed; they can cause fires after having damaged power lines; they can cause floods after breaking dams; they can cause tsunamis that take the lives of thousands of people. Because of all of that, it is extremely important to study earthquakes and find ways to predict them.

1.2 Earthquake Prediction and Forecasting

In this section, let's take a look at some theory behind earthquake prediction and forecasting. Generally, prediction or predictive modeling is a statistical method that is made to predict outcomes. It can be used to predict events in the future or any particular unknown event regardless of when it occurred. In our case, we are going to use predictive modeling to *classify* whether currently read signals show that the earthquake took place.

While for many people prediction and forecasting add up to the same, or precisely *almost* the same, definition, we are going to distinguish these two mentioned notions. Forecasting in our case is making *a prediction* about the future based on past and/or present data.

Seismologists often perceive prediction as determination of location, date, time, and magnitude of the alleged earthquake. But the issue with this definition is that according to Geller et al. [4] earthquakes cannot be predicted. This article suggests that an earthquake results from a sudden slip on a geological fault. Such fracture and failure problems are notoriously intractable. Although other seismologists claim to have reached success in this area of study. For example, Varotsos, Alexopoulos, and Nomicos [5] designed the method called VAN. Scientists found that special signals known as Seismic Electric Signals (SES) are observed before an earthquake and are caused by transient variations of the Earth electric field. Authors claimed their prediction to be successful when an earthquake occurred within several days to several weeks after recording SES. Despite the fact that the algorithm succeeded in 60% of all observed events in Greece, it was widely criticized and thought to be coincidental [5].

1.3 Classification

Considering all the facts, all definitions will be used as they were stated in the previous section of the current chapter.

As mentioned before, one of the methods of predictive modeling is classification which can be defined as the problem of identifying to which subcategory a new observation belongs. There are several well-known algorithms for machine learning classification:

1. K-Nearest Neighbors
2. Naive Bayes
3. Decision Tree Classifier
4. Random Forest Classifier
5. Rotation Forest Classifier
6. Logistic Regression

In the next parts of the current chapter will be the brief introduction to every mentioned model and the algorithms that stand behind these models will be explained.

1.4 K-Nearest Neighbors

This is one of the examples of one of the simplest supervised learning algorithms. [6] Given the scatter plot and distance function that allows computing the distance between any two points on that scatter plot, it is possible to classify each new entry. In order to do so, it is required to have some amount of data already classified – data that are used for training purposes. Depending on K, when a new entry is being classified, the algorithm looks at K nearest points on a scatter plot and based on distance metric decides which class this entry belongs to as can be observed in Figure 1.1

Let's take a look at distance function.

Distance or **metric** in the set χ is a function $d : \chi \times \chi \to [0, +\infty)$ such that for each $x, y, z \in \chi$ it holds [7]

1. *positive definiteness:* $d(x, y) \geq 0$, a $d(x, y) = 0$ iff $x = y$
2. *symmetry:* $d(x, y) = d(y, x)$
3. *triangle inequality:* $d(x, y) \leq d(x, z) + d(z, y)$

Popular option for distance function is **Euclidean distance**, which is also called L_2. It is represented as:

$$\|x - y\|_2 = d_2(x, y) = \sqrt{\sum_{i=0}^{p-1}(x_i - y_i)^2},$$

for two points $x = (x_0, x_1, \ldots, x_{p-1}) \in \mathbb{R}^p$ and $y = (y_0, y_1, \ldots, y_{p-1}) \in \mathbb{R}^p$. Cosine distance is also often used:

$$\frac{x \cdot y}{\|x\|_2 \|y\|_2}$$

It is crucial to choose suitable K. It should be small enough for not to pick up irrelevant neighbors, but big enough to enclose enough data points to get a meaningful sample. It is recommended to run tests for different values of K to determine the appropriate one.

To sum it up, this algorithm has deserved right to be called one of the simplest supervised learning algorithms because all it does is check K nearest to the new entry points on the scatter plot and let the majority of them decide which category this entry belongs to. Although it sounds very easy, this algorithm is extremely powerful. Despite all that, it has its drawbacks: the curse of dimensionality.

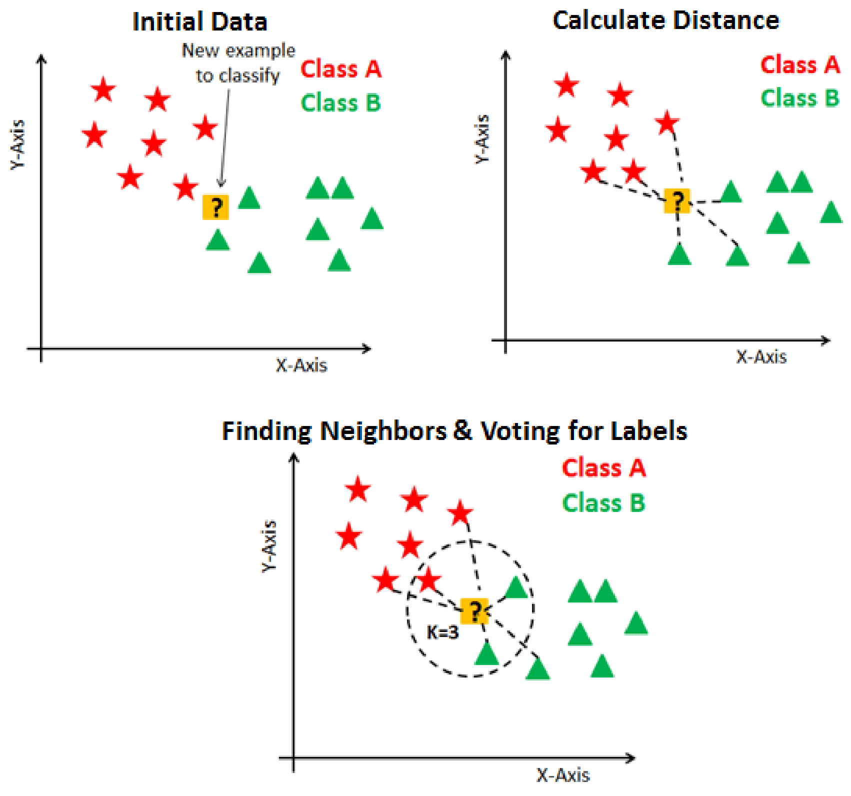

Figure 1.1: K-NN in action.

1.4.1 The Curse of Dimensionality

K-NN is known to perform better with a lower number of features than with higher. [8] The more dimensions (attributes) there are, the more data entries required and this dependence grows exponentially because an increase of dimensions leads to a problem known as overfitting [1]. It has been shown that with a larger number of attributes, Euclidean distance is not effective and other measures, such as cosine similarity, are preferred. In order to eliminate the mentioned issue by preserving the maximum amount of variance, principal component analysis (PCA) is used.

1.4.2 Principal Component Analysis

In order to understand what PCA is, it is essential to introduce a few definitions. Let's use $\mathbf{X}_{:i}$ for ith column of matrix $\mathbf{X}$ that contains measured values of attribute X_i, where $\mathbf{X}_{:i} = (x_{1,i}, \dots, x_{N,i})^T$. [7]

- *Sample covariance* of attributes X_i and X_j, which is estimation of $cov(X_i, X_j)$, is:

  $$\widehat{cov}(X_i, X_j) = \tfrac{1}{n-1} \textstyle\sum_{k=1}^{N} (x_{k,i} - \bar{\mathbf{X}}_{:i})(x_{k,j} - \bar{\mathbf{X}}_{:j}),$$

  where $\bar{\mathbf{X}}_{:i} = \frac{1}{N} \sum_{k=1}^{N} x_{k,i}$ stands for sample mean of ith attribute.

- *Covariance matrix* is matrix which has $cov(X_i, X_j)$ as its (i, j)th element.

- *Sample covariance matrix* is a matrix which has $\hat{cov}(X_i, X_j)$ as its (i, j)th component and is in fact point estimation of covariance matrix. Although further in this paper the notion of a **covariance matrix** will be used for the **sample covariance matrix**.

Covariance matrix is symmetric and on diagonal has non-negative values - sample variances. Let's assume that data is centered, i.e. $\mathbf{X}_{:i} = 0$, then covariance matrix can be expressed as $\frac{1}{N-1}\mathbf{X}^T\mathbf{X}$. Since covariance matrix is symmetric, it is also diagonalizable and using corresponding eigenvectors it is possible to create an orthonormal basis of $\mathbb{R}^p$. Final so-called eigendecomposition is

$$\tfrac{1}{N-1}\mathbf{X}^T\mathbf{X} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T,$$

where

- $\mathbf{\Lambda} \in \mathbb{R}^p$ is a diagonal matrix with eigenvalues in diagonal arranged in descending order according to size, $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_3$
- $\mathbf{V} \in \mathbb{R}^p$ is an orthogonal matrix which columns contain corresponding eigenvectors producing orthonormal basis.

[1] [9] The production of an analysis which corresponds too closely or exactly to a particular set of data, and may, therefore, fail to fit additional data or predict future observations reliably

Then projection to d dimensional space can be performed as

$$\mathbf{X'} = \mathbf{X}\mathbf{W}_d,$$

where $\mathbf{W}_d \in \mathbb{R}^{p,d}$ is a matrix containing first d columns of matrix $\mathbf{V}$.

Let's consider ith new attribute $X_i^{'} = v_{i,1}X_1 + \ldots + v_{i,p}X_p = \mathbf{X}^T\mathbf{v}_i$ that is called ith principal component.

$$\bar{X'}_i = \tfrac{1}{N}\textstyle\sum_{k=1}^{N}(v_{i,1}x_{k,1} + \ldots) = \tfrac{1}{N}v_{i,1}\textstyle\sum_{k=1}^{N} x_{k,1} + \ldots + v_{i,1}\tfrac{1}{N}\textstyle\sum_{k=1}^{N} x_{k,p} = 0,$$

$$\widehat{var}X_i' = \tfrac{1}{N-1}\textstyle\sum_{k=1}^{N}(v_{i,1}x_{k,1} + \ldots + v_{i,p}x_{k,p})^2 = \tfrac{1}{N-1}\textstyle\sum_{k=1}^{N}(\mathbf{x}_k^T\mathbf{v}_i)^2 = \\ \tfrac{1}{N-1}\mathbf{v}_i^T\mathbf{X}^T\mathbf{X}\mathbf{v}_i = \mathbf{v}_i^T\lambda_i\mathbf{v}_i = \lambda_i.$$

That implies that variance of ith component equals value of ith eigenvalue λ_i and that, by choosing d main components, directions where data have the greatest variance are selected.

In other words, PCA simply takes high-dimensional data and then trying to preserve the most variance projects data to a lower-dimensional space. In our case we are dealing with time-series data and, obviously, it has many attributes and PCA will be used to reduce the number of dimensions while still preserving or increasing an accuracy rate.

1.5 Naive Bayes

In order to understand how Naive Bayes Classifier works, it is crucial to understand the Bayes' Theorem and how it is used for real-life cases. The next section will give a brief introduction to this.

1.5.1 The Bayes' Theorem

Let's consider two probabilistic events A and B. Then we will have:

$$P(A \cap B) = P(A|B)P(B)$$

$$P(B \cap A) = P(B|A)P(A)$$

Considering the intersection is commutative, $P(A \cap B) = P(B \cap A)$, left sides are equal, too. Then it is possible to derive the Bayes' theorem: $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$ Firstly, let's take $P(A)$. This denotes how probable the target event is. It is called apriori probability because it is determined by mathematical considerations. $P(B|A)$ is called likelihood and $P(B)$ is considered to be normalization constant. Often, it is written as α and the overall equation looks as follows:

$$P(A|B) = \alpha P(B|A)P(A)$$

Lastly, it is important to consider the case where there are $n \in \mathbb{N}$ concurrent conditions which is more plausible for real-life observations:

$$P(A|C_1 \cap C_2 \cap \ldots \cap C_n)$$

Considering the assumption that the effects of every cause are independent (*conditional independence*), it is possible to outline following expression:

$$P(A|C_1 \cap C_2 \cap \ldots \cap C_n) = \alpha P(C_1|A)P(C_2|A)\ldots P(C_n|A)P(A) \text{ [10]}$$

1.5.2 Naive Bayes Classifier

Now, after defining Bayes' Theorem, it is possible to explain how Naive Bayes' Classifier works. It is called in this way for a reason: it is based on a naive condition where the conditional independence of all causes is implied. In practice it is hardly possible to imagine – all events seem to be correlated with one another and as a result, can perform with low accuracy quite often.

Let's consider the dataset:

$$X = \{\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n\} \text{ where } \bar{x}_i \in \mathbb{R}^m$$

Every feature vector will be represented as:

$$\bar{x}_i = \{x_1, x_2, \ldots, x_m\}$$

Then, we also have a target dataset:

$$Y = y_1, y_2, \ldots, y_n \text{ where } y_i \in \{0, 1, 2, \ldots, N\}$$

and each y can belong to one of N different classes. Considering Bayes' theorem under conditional independence, we can write:

$$P(y|x_1, x_2, \ldots, x_m) = \alpha P(y) \textstyle\prod_i P(x_i|y)$$

The values of the marginal apriori probability $P(y)$ and of the conditional probability $P(x_i|y)$ are obtained through frequency count. That means that given an input vector x, the predicted class is the one which a posteriori probability is maximum. [10]

1.6 Decision Tree Classifier

Decision tree itself is a specific arrangement of data in a tree structure where at each node data is separated into two branches according to the value of the attribute at this node. [11] In order to understand the algorithm, it is required to define some notions:

- *Information entropy* of the probability space is

$$E(S) = -p_1 log_2(p_1) - \ldots - p_n log_2(p_n),$$

where p_i is the probability an element i would be chosen from the probability space, $i \in \{1, 2, \ldots, n\}$.

- *Information gain* for an attribute A is defined as follows:

$$IG(S, A) = E(S) - \sum_{v \in values(A)} \frac{|S_v|}{S} E(S_v),$$

where S_v is a set with elements of S that have the value v for the attribute A and $values(A)$ denotes the set of all possible values of attribute A.

All previous definitions will be the basis of the algorithm that is going to be discussed further - ID3 algorithm.

1.6.1 ID3 Algorithm

This algorithm is used to construct a decision tree from data based on information gain. Each element in the set S has attributes $A_1, A_2, \ldots, A_m$ and it is possible to partition this set according to any of the possible attributes. This algorithm divides the set starting with the attribute that yields the highest information gain.

The tree starts with the root and goes all the way down and splits the data up into different groups based on attributes' information gain that was counted before. In can be finished either when we ran out of attributes or even earlier – when until all data at the node have the same class of our interest.

Once the tree was constructed, it can be used to classify new entries. The main advantage of the decision trees is that they do not require data preprocessing and normalization, which comes in handy, but it has a few drawbacks, such that it is inclined to overfitting a model when the depth of the tree is not limited to a particular level. [6]

1.7 Random Forest Classifier

Random forest is built by combining the predictions of several decision trees that have been trained in isolation. [12] It is a supervised learning algorithm. Decision trees are trained with the help of *bagging* method which general idea is that the combination of learning models increases performance. The biggest advantage of random forests is that it is possible to use it for classification and regression problem and there are the most popular ones nowadays. In the current paper, we are talking only about the classification part, since the data we have can belong to one of the two classes.

We already know how the decision tree algorithm works and random forest works very similar, but instead of looking for the most important feature while splitting the node, it looks for the best feature among a random subset of features. However, the main drawback of random forests is that large number of trees makes the model slow and ineffective for real-time predictions. Random forests are fast to train but are relatively slower when making predictions. Lastly, the more accurate prediction requires more trees and that as it has been already mentioned results to a slower algorithm.[13]

1.8 Rotation Forest Classifier

The main difference from the previous tree algorithm is that the Rotation Forest does not require as many trees to be created to achieve impressive accuracy. Hence, it spares much time on creating trees which is fairly time-consuming. Interestingly, authors of the algorithm claim that underlying estimator can be not only a tree but anything other as well. Although what remains unchanged is that it still uses bagging as one of the basic techniques. [14] What is also very important that in comparison to different algorithms such that AdBoost (that hasn't been used in this paper), simple bagging and Random Forest on 33 datasets, Rotation Forest outperformed all of them. [15]

1.8.1 Algorithm

A number of the trees should be specified by the user. When it is done, the algorithm looks like:
For each tree T, perform the following:

1. Split the attributes in the training set into K non-overlapping subsets of equal size

2. For each of K datasets with k attributes perform the next steps:

    - Bootstrap 75% of the data and use the bootstrapped sample for further steps.
    - Run PCA on the ith subset in K and save all components For every feature j in the Kth subset, we have principal a principal component, a. We are going to denote it as a_{ij}, where it is the principal component of the jth attribute in the ith subset.
    - Store the principal components for the subset.

3. Create a rotation matrix of size $N \times N$, where N is the total number of attributes. In the matrix, each principal component should match the position of the feature in the original training dataset.

4. Project the training dataset on the rotation matrix using the matrix multiplication

5. Build a decision tree with the projected dataset

6. Store the tree and rotation matrix.

1.9 Logistic Regression

This is another important model that is used for cases where the target is categorical. We could have used linear regression, but the problem is that we would need to set some threshold where output value is considered to be denoting one class or another. For instance, linear regression model gives the output of 0.45 whereas our threshold value is 0.5 so we would assign a label to be 0 and not one (in terms of our data) which could lead to serious problems in real time. Linear regression is unbounded and here comes the logistic regression model that strictly ranges from 0 to 1. [16]

The key feature of the logistic regression is that it uses the logistic function to place results of a linear equation between 0 and 1 Logistic function or *sigmoid* is defined as [17]

$$logistic(\mu) = \frac{1}{1+\exp(-\mu)}$$

Graph of this fuction is plotted in the Figure 1.2. In the linear regression the

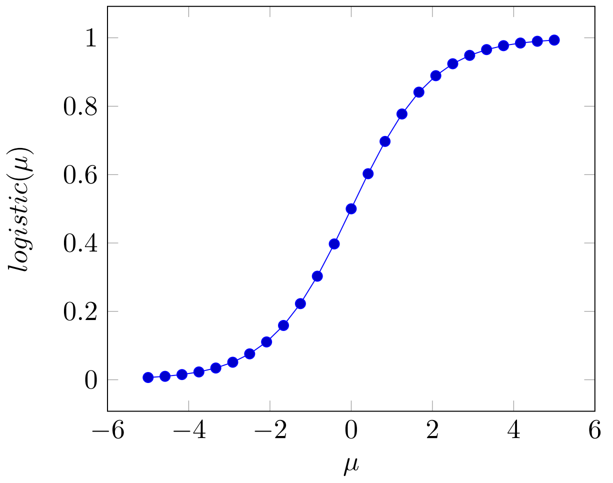

Figure 1.2: Logistic function

relationship between the outcome and features is modeled as:

$$\hat{y}^{(i)} = \beta_0 + \beta_1 x_1^{(i)} + \ldots + \beta_p x_p^{(i)}$$

For the logistic regression it is required to have probabilities between 0 and 1, so right side of the function would be wrapped into the logistic function that will force the output to be in the range from 0 to 1:

$$P(y^(i) = 1) = \frac{1}{1+\exp(-(\beta_0+\beta_1 x_1^{(i)}+\ldots+\beta_p x_p^{(i)}))}$$

When it comes to weight interpretation in logistic regression, it is different than in linear regression. Since we have probability on the left vside of equation, weights do not influence it linearly any longer.

$$\log\left(\frac{P(y=1)}{1-P(y=1)}\right) = \log\left(\frac{P(y=1)}{P(y=0)}\right) = \beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p$$

Ratio inside of a logarithm function is called *odds* and with logarithm wrapped around it, it is called log odds. We want to know how the prediction changes when one of the features is changed by 1 unit. We are going to perform following modification on the previous expression:

$$\frac{P(y=1)}{1-P(y=1)} = odds = \exp(\beta_0 + \beta_1 x_1 + \ldots + \beta_p x_p)$$

Then we compare what happens when we increase one of the features x_j by 1 and we are going to look at the ratio of the two predictions:

$$\frac{odds_{x_j+1}}{odds} = \frac{\exp(\beta_0+\beta_1 x_1+\ldots+\beta_j(x_j+1)+\ldots+\beta_p x_p)}{\exp(\beta_0+\beta_1 x_1+\ldots+\beta_j(x_j)+\ldots+\beta_p x_p)}$$

After applying the rule

$$\frac{\exp(a)}{\exp(b)} = \exp(a-b),$$

we will get much simplified expression:

$$\frac{odds_{x_j+1}}{odds} = \exp(\beta_j(x_j+1) - \beta_j x_j) = \exp(\beta_j)$$

After these modifications, we can say that a change in feature by one unit changes the odds ratio by a fuctor of $\exp(\beta_j)$.

1.10 Performance measure

After a model is trained, performance measure should be considered to learn how effective particular algorithm was. Of course, there are some methods to do so, such as:

1. Classification accuracy
2. Confusion matrix
3. Mean squared error
4. Receiver Operator Curve and Area Under the Curve

1.10.1 Classification accuracy

The term itself means the ratio of the number of correct predictions to the total number of input samples:

$$Accuracy = \frac{NumberofCorrectpredictions}{Totalnumberofpredictions}$$

The problem is, that it is working well only when we have the same or almost the same number of samples belonging to each class. For example, if we have 90% of samples from one class and 10% from a different class, we can easily reach the accuracy of 90% just simply predicting that all samples to be of the first class. [18]

1.10.2 Confusion matrix

It is a matrix that describes the overall performance of a model. For example, a signal in our case can be classified as 0 (for no earthquake) or 1 (for the case when an earthquake did happen). Matrix encapsulates 4 core notions to understand how well the model performed: [18]

- True Positives (TP) – cases that were predicted as 1 and actually were labeled as 1.
- True Negatives (TN) – cases that were predicted as 0 and actually were labeled as 0.
- False Positives (FP) – cases that were predicted as 1, but actually are labeled as 0.
- False Negatives (FN) – cases that were predicted as 0, but actually were labeled as 1.

Accuracy of the matrix can be calculated in the following way: [7]

$$Accuracy = \frac{TruePositives + TrueNegatives}{TotalNumberofSamples}$$

1.10.3 Mean Squared Error

Mean Squared Error (MSE) is an average of the square difference between the original values and the predicted values. Since we use the square of the error, the impact of larger errors becomes more visible than impact of smaller ones, hence it gives us space to concentrate on bigger errors. [18] It is calculated in the following way: [7]

$$MSE = \frac{1}{N}\sum_{j=1}^{N}(y_j - \hat{y_j})^2$$

1.10.4 Receiver Operator Curve and Area Under the Curve

Receiver Operator Curve (ROC) is a graph that shows the performance of classification model and plots two parameters:

- True Positive Rate (TPR) or Recall and it is defined as

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

- False Positive Rate (FPR) and it is defined as

$$TPR = \frac{FP}{FP+TN}$$

Area Under the Curve measures entire area under ROC from $(0,0)$ to $(1,1)$ These methods are the most popular in acquiring deeper knowledge on performance and are most frequently used among data scientists.

There are also a couple of details that are worth mentioning considering the data that we are working with. We are working with time-series waveforms and there are specific methods for handling this type of data. The simplest way to manage it is to use raw time-series waveforms partitioned into windows of equal time-steps. [19]. Another way is to take advantage of Fast Fourier Transform algorithm for feature extracting [20]. However, the feature extraction has proven itself not being efficient due to the fact that features are not informed by the prediction variable of interest and may cause noise whilst reducing dimension.

Regarding forecasting part of the current paper, everything is vague, because the state-of-the-art algorithm, as is consequent from previous paragraphs, does not exist. But according to the studies [21], neural networks are a possible solution for prediction, while still having weaker performance than classification for obvious reasons. In this paper we will try these algorithms out:

1. 1D Convolutional Neural Networks
2. Recurrent Neural Networks

1.11 Neural Networks

The neural networks are inspired by successfully working biological systems, that consist of numerous nerve cells and have the *capability to learn.* As a result, neural networks can generalize and associate data: after a training procedure, they can find meaningful solutions for similar problems of the same class that were not trained. [22]

Although there are different types of neural networks, here only two of them will be used. Let's discuss their architecture and characteristics.

1.11.1 1D Convolutional Neural Networks

Convolutional Neural Networks, or CNN, are a special type of neural networks that are used for processing data that has grid-like topology. One-dimensional are specifically used for time-series classification at regular time intervals. The name "convolution" denotes that the network uses operation called *convolution*, which is a kind of linear operation. CNN just use convolution instead of general matrix multiplication. [23]

1.11.1.1 The Convolution Operation

The convolution operation is defined in a following way:

$$(x * w)(t) = \int x(a)w(t-a)da,$$

where $x(t)$ is referred to as *input* and $w(t)$ as the kernel. The output is sometimes referred to as a *feature map*. Usually, we work with cases when time is discretized and data are provided in regular intervals. If we assume that time index t can take only integer values and x and w are defined only for integer t, we can define discrete convolution:

$$(x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a)$$

In ML algorithms, the input is usually a multidimensional array of data and kernel is a multidimensional array of the parameter being adjusted during the learning process. These arrays will be referred to as tensors.

1.11.1.2 Features of CNN

Neural networks layers use matrix multiplication by a matrix of parameters. Convolutional neural networks, however, have *sparse interactions*. This means that the kernel is smaller than the input. It is possible to achieve that using meaningful features detection. Because of that, we need to store fewer parameters and this helps us occupy less memory and increases statistical efficiency.

Another important feature is *parameter sharing*. It refers to using the same parameters for more than one function in the model also called *tied weights*. Each member of the kernel is used at every position of the input. It is extremely beneficial feature since it causes to learn only one set of parameters instead of learning a separate set for every location. Still, it does not affect the runtime, rather it reduces the storage requirements.

Another feature of the layer is *equivariance*. This means that when input changes, the output changes in the same way. Mathematically, a function $f(x)$ is equivariant to a function g if $f(g(x)) = g(f(x))$. [23]

1.11.1.3 Pooling

A typical layer of CNN consists of three stages:

1. Convolution stage: several convolutions in parallel to produce a set of linear activations
2. Detector stage: each linear activation is run through a nonlinear activation function
3. Pooling stage: modification of the output of the layer

Pooling replaces the output of the network at a certain location with a summary statistics of the nearby outputs. Pooling is used to make the representation *invariant* to small translations of the input. It means that translation of the input by a small amount, the values of most of the pooled outputs do not change. [23]

1.11.2 Recurrent Neural Networks

To understand Recurrent Neural Networks, or simply RNN, it is important to know how feed-forward networks work. All of them are named to represent the way that information goes through. While feed-forward neural networks transfer the information through its nodes, the recurrent networks cycle it through a loop.

Feed-forward network trains on labeled instances until it minimizes the error of classification. After having finished the training process, the first unknown instance that it will have to classify, could or could not affect the way it classifies the second. To sum up, basic feedforward networks has no notion of time and work only with current input without being affected by previous ones.

That makes RNN stand out since they take current input having *memory* about what they had seen in the past. It finds correlations between events separated by many moments. These correlations are called “long-term dependencies”, because the event that comes next depends on one ore more that came before. [24]

1.11.2.1 Architecture

Recurrent Neural Networks are different from other Artificial Neural Networks because they contain at least one feed-back connection, so the activations flow in a loop. Hence, they are suitable for sequence recognition/reproduction or prediction that is incredibly helpful for earthquake forecasting problem.

The most common type of RNN has standard Multi-Layer Perceptron (MLP) with extra loops. These loops are used to create a very powerful feature of the neural networks, that is the memory. Others have every neuron connected to all the others and may have stochastic [2] activation functions.

For simple architectures and deterministic activation functions, learning is achieved through similar gradient descent procedures that are used in back-propagation algorithm for feed-forward networks. When a random activation function is used, simulated annealing processes are more suitable. [25]

1.11.2.2 Features of RNN

Fully recurrent neural networks in its simplest form is an MLP that has hidden units activations fed back into the network along with the inputs. Time has to be divided into different time steps, i.e. be discretized, whereas activations are updated at each time step. To hold activations until they are processed exists delay unit.

An extremely important feature of the RNN is the state space model. Let us say, that input and output vectors are $x(t)$ and $y(t)$ respectively, the three weight matrices $\mathbf{W}_{IH}$, $\mathbf{W}_{HH}$ and $\mathbf{W}_{HO}$, and the hidden and output unit activation functions f_H and f_O. Then we can describe the behavior of RNN as a dynamical system using the pair of non-linear matrix equations:

$$h(t) = f_H(\mathbf{W}_{IH}x(t) + \mathbf{W}_{HH}h(t-1)) \; y(t) = f_O(\mathbf{W}_{HO}h(t))$$

The definition of the state of a dynamical model is a set of values that summarizes all the information about the past behavior of the system that is required to determine the description of its future behavior. The state is defined by the set of hidden unit activations $h(t)$. This implies, that apart from input and output spaces, there is another space that is called *state space*. Lastly, each dynamical system has its *order* that is defined as the dimensionality of the state space. [25]

1.11.2.3 Backpropagation Through Time

The Backpropagation Through Time, or BPTT, is an extension of standard backpropagation and it performs gradient descent on a complete unfolded network. Let's imagine, that training sequence starts at time t_0 and ends at time t_1, the total cost function is the sum over time of the standard error function $E_{sse/ce}(t)$ at each time step:

$$E_{total}(t_0, t_1) = \sum_{t=t_0}^{t_1} E_{sse/ce}(t)$$

and the gradient descent weight updates have contributions from each time-step:

$$\Delta w_{ij} = -\eta \frac{\partial E_{total}(t_0,t_1)}{\partial w_{ij}} = -\eta \sum_{t=t_0}^{t_1} \frac{\partial E_{sse/ce}(t)}{\partial w_{ij}}$$

[2] random

The partial derivatives $\frac{\partial E_{sse/ce}}{\partial w_{ij}}$ now have contributions from the multiple instances of each weight $w_{ij} \in \{\mathbf{W}_{IH}, \mathbf{W}_{HH}\}$ and depend on the input and hidden unit activations at previous time steps.

Usually, all updates are made in online fashion. That requires storage of history of inputs and previous network states at earlier times. In order to make it computational feasible, it requires truncation at a certain number of time steps.

Truncation of the unfolded network to one-time step reduces it to Simple Recurrent Network, or Elman network. In this case, each set of weights appears only once, therefore it is possible to use standard backpropagation over BPTT. This means, that it will be difficult for the network to use data from far back in time.

The other type is very effective for time-series prediction which is quite useful in our case. It is called Non-Linear Auto-Regressive with Exogenous inputs (NARX) model. It has a single input and output with a delay line on the inputs and the outputs are fed back to the input by another delay line (MLP structure). [25]

1.11.2.4 Vanishing Gradients

As it is already known, each weight receives update based on partial derivative of an error function that corresponds to a particular weight in each iteration of training. Here comes the problem, where this gradient obtained can be extremely small and it causes the weight to cease changing its value. As a result, it can completely stop the training process of the neural network. [24]

1.11.2.5 Long Short-Term Memory Units

Long Short-Term Memore Units, or LSTM, was proposed by Sepp Hochreiter and Juergen Schmidhuber in the mid-90s as a solution to vanishing gradient problem. It helps to preserve the error that is being backpropagated through time and layers. This algorithm maintains constant error so the networks can continue to learn even over many time steps.

LSTM have their own memory unit to store information outside the recurrent network. They store it in special cells. Information can be stored, accessed for writing or reading. The cell itself decides what to store when to allow other actions, and when to open or close gates. The gates are similar to neural networks nodes as they act on the signals they receive. They block or pass information based on its strength via their own set of weights. Those weights are adjusted via recurrent networks learning process. [26]

Chapter 2

Solution of the Problem

In this section, the approach for solving this problem will be discussed alongside with the advantages and disadvantages of using particular methods. Firstly, we need to discuss how the data were obtained.

2.1 Gathering Required Data

Since there are not many various data sets prepared for this topic, the problem of getting data to work with was very acute. All datasets found before had a questionable origin and most importantly they did not contain vast numbers of data and moreover these datasets had a rather poor amount of cases when earthquake did occur.

However, the solution to this problem was found. The choice fell on ObsPy – Python framework for seismology that has embedded methods for downloading data from seismic stations. It is required to find a certain amount of positive events and other things are taken care of by the framework. It connects to the various different seismic stations and gets the data using embedded methods and saves it to the file system.

2.1.1 About the Data

For purposes of the thesis, 15 different earthquakes were chosen. They were downloaded using method *download* from the class *MassDownloader*. All the data were downloaded from the single provider "GFZ" from one channel in order to maintain consistency. The method requires just the coordinates of a particular event and time when it occurred. For example, resulting dataset contains data of such large and costly earthquakes as Tohoku 2011 earthquake that caused tsunami after as well as some random recent earthquakes that did not invite any victims at all.

For positive cases, we took a time range between 5 minutes before the earthquake occurred and an hour after. Right now it is clear how positive cases were found, but what about negatives? Regarding this part of the assignment, it was decided to do it the same way as with positives. One difference is time range – it was taken between 70 minutes before the event started and right before it starts.

Script for downloading data is located in the project directory and is called “download_data.py”.

2.1.2 Providers and Stations

GFZ is German Research Center for Geosciences and is a national research center for Earth sciences. It does not specialize only on seismology, but on mathematics, chemistry, physics, and biology as well. Since this paper is about earthquakes, only topics regarding them are relevant. This organizations operates data center that is called *GEOFON* that was founded in 1993 and serves its purpose nowadays, too. In 2014, this data center contained 80 active stations on all continents but being highly concentrated in Europe, Mediterranean region and in the Indian Ocean.

Each station uses various channels for reading and transmitting signals. When data are being read it is possible to specify the channel or channels that will be used. Each channel name consists of three letters, one of each serves a different purpose. For example, some options on what the first letter denotes are shown in the table 2.1. [27] Broad and short periods seem to have

Band code	**Band type**	**Sample rate (Hz)**
E	Extremely short period	≥ 80 to < 250
S	Short period	≥ 10 to < 80
H	High broad band	≥ 80 to < 250
B	Broad band	≥ 10 to < 80

Table 2.1: The first letter of channel deciphered.

the same frequency, but they have different corner period – short ones have corner period < 10 sec while broad ones have ≥ 10 sec.

The next letter says about the instrument that was used as a sensor. Basically, it identifies what is being measured. Table 2.2 contains all possible options for seismometers, which were used in this paper. [27]

Finally, the third letter denotes the orientation code for the instrument. It provides a way to indicate the directionality of the sensor measurement. [27] Explanation of all possible options is given in table 2.3.

Instrument code	**Meaning**
H	High Gain Seismometer
L	Low Gain Seismometer
G	Gravimeter
M	Mass Position Seismometer
N	Accelerometer

Table 2.2: The second letter of channel deciphered.

Orientation code	**Meaning**
Z N E	Traditional (Vertical, North-South, East-West)
A B C	Triaxial
T R	For formed beams (Transverse, Radial)
1 2 3	Orthogonal components but non-traditional orientation
U V W	Optional components

Table 2.3: The third letter of channel deciphered.

2.1.3 Creating Datasets

With the retrieved data, two datasets for different purposes were created – one for prediction and another for forecasting. The methods for creating them are a little bit different and will be discussed in the current section.

After having downloaded the data, four directories – two directories for both prediction and forecasting, positive and negative events separately – were created with all signals placed in different files. It is not quite convenient to work with, considering the fact that data should be aggregated in datasets that could have been uploaded into the python environment to start exploring them. Therefore, firstly, the data was accumulated into two separate datasets stored on the disk.

The data was preprocessed in the same way for both prediction and forecasting. The number of attributes was reduced from approximately 200 000 per each signal to 196 in order to fit resulting dataset to the memory. It was achieved by changing every 1 000 of attributes with their mean value. The data was not affected by this change since there was extremely much noise in downloaded signals.

After these modifications, dataset for prediction was stored and ready to be used. However, forecasting dataset was not – in order to work with it, we need to have so-called “past” and “future” cases to train and test models. Therefore, the negative event becomes the past event and the positive becomes future since negative event was before positive one. The problem was, that there was an uneven number of them, so we needed to find unpaired ones and remove them. After this, datasets for forecasting was created and ready to be worked with.

Both prediction and forecasting datasets are stored in the "data" folder of the project directory, so it is easy to import them to the Jupyter Notebook.

2.2 Baseline Modeling

For prediction purposes, basic machine learning algorithms were applied, among which are all mentioned above in the previous section. This part is important for maintaining efficient algorithms that can accurately recognize whether current waveform contains data signaling about occurring earthquake or not.

Before creating the model, it is very important to divide data set into training and testing sets. It is usually done using embedded methods from Sklearn module in Python. We need to leave at least 20% for testing purposes and leave this data until the very last moment because we do not want to bias our model. For the next part, we are going to normalize data – to convert entries from a given range to range from 0 to 1. For neural networks algorithms, we are going to convert labels to a binary matrix using label encoder.

Regarding recognition, the best algorithm is K-Nearest Neghbors with accuracy on the test set more than 83%. Second best algorithm is Rotation Forest Classifier with accuracy more than 82%. And the worst performance had Naive Bayes algorithm with accuracy on the test set nearly 70%, although it is not far worse than other algorithms tested. The whole performance of models and highest achieved accuracies are given in the table 2.4.

Model	**Accuracy (%)**
Decision Tree	71.7
Random Forest	73
Logistic Regression	70.7
Rotation Forest	82.1
K-NN	83.2
Naive Bayes	70

Table 2.4: Accuracies of the different baseline models.

2.3 Neural Networks

Now we are going to check performance for classification task using neural networks. Both of these models were fed by waveform data, as well as all previous algorithms.

2.3.1 1D Convolutional Neural Networks

Convolutional neural network (CNN) models [28], known as one of the biologically inspired models, have been used for pattern recognition tasks such as face recognition and hand-written numeral recognition. It is regularized version of multilayer perceptron, which refers to fully connected network – each neuron from one layer is connected to every neuron on another.

According to our experiments, 1D CNN showed great results for classification tasks – accuracy is 81% on the test set. They also showed quite good results (RMSE value is 0.093 on normalized data). These results are denoted in tables 2.5 and 2.6.

2.3.1.1 Model Hyperparameters

For the prediction part, 3 layers of convolutional neural networks were created. The first layer is called *convolutional* consists of 32 filters, or feature detectors, the second layer – of 64 filters, third – of 128 filters. Input height of the dataset equals to the number of attributes and is 196.

Next filter parameter is input width also referred to as depth. In our case, we have got only one because we have only one signal. In the case of a picture, for instance, it could be three – for each color in RGB pattern.

Then we need to enter kernel size, or in other words – width of the sliding window that will be going through all data. In the case of current solution, it is four because it was tested to have the best performance within the current data. Given these hyperparameters, each window will slide through all the data in 193 ($= 196 - 4 + 1$) steps.

The next layer of the CNN is *pooling layer*. It is usually used after convolutional layer in order to prevent overfitting and reduce complexity of output. In our example size of 3 was chosen. It means that the output layer will contain only $\frac{1}{3}$ of the input matrix.

Next comes *activation* function – very important and basically the main part of every neural networks algorithm. Activation functions introduce nonlinear properties to it. Their main purpose is to convert input signals into output signals that are then introduced as the inputs to the next layers.

All layers, except for output, in 1D CNN model for both prediction and forecasting contain activation function called ReLU. This states for Rectified Linear Units. This is very simple function that is defined: $R(x) = max(0, x)$. In other words if x is less than 0, it just passes 0 for x, otherwise, it gives the value of x. It is very simple and efficient.

The output layer, on the other hand, was proven to generate better results when having activation function called "sigmoid" or sometimes "logistic". The graph of this function has been shown in this paper before in Figure 1.2. Just to remind, it is defined as $\frac{1}{1+\exp(-x)}$. The problems it causes was also discussed earlier in the 1D CNN part of theoretical background, such as gradient vanishing. Despite causing various problems, it was proven to work the most effective in the output layer of the current model.

The next layer is *Dropout layer*. Dropout is a technique where randomly selected neurons are ignored during training. [29] This means that their contribution is temporally removed on the forward pass and weight updates are not applied on them on the backward pass. As a result of this process, the network is less prone to overfitting and gives better generalization. In both prediction and forecasting cases, the value of Dropout was set at 0.1. This means that 10% of all neurons were randomly dropped out.

As it has already been mentioned, three convolutional layers were created for prediction part. Each of them follows the same pattern as was described above – same pooling, activation, and dropout. It is a bit different for forecasting part – only two layers were used and dropout technique was not used there because it has given worse results. Also, for pooling layer instead of 3, 2 was used. That means that now output contains only half of the input. The next extremely important part is that the number of filters in both convolutional layes is significantly lower than in the prediction part. Since we have less data for forecasting, a bigger number of feature detectors gave much worse results than lower numbers.

The next and the last layer is the output layer. In both cases, it is done similarly. The only difference is the dimension of output we require for each model. For prediction we need to classify, whether current waveform describes positive or negative case – earthquake or no earthquake. Therefore, we need the output to be either 0 or 1. For this, we use fully connected layer with sigmoid activation. It reduces the output of the previous layer to the size of 2. This is done using matrix multiplication. The output value represents the probability of each of two classes for prediction. In case of forecasting it goes similarly, but we map out output to the 196 different values – because out waveform contains 196 different attributes.

After we defined model, comes the training process. We simply pass our model training set with corresponding labels in order for it to fit this data. The training process is divided into the given amount of epochs during which appropriate weights are assigned and adjusted. It is required to set amount of epochs that corresponds to the number of rows in training dataset in order to reduce probability of both over- and underfitting.

2.3.2 Recurrent Neural Networks

Recurrent neural networks (RNNs) contain cyclic connections that make them a more powerful tool to model sequence data than feed-forward 4neural networks. RNNs have demonstrated great success in sequence labeling and prediction tasks such as handwriting recognition and language modeling [30].

In the case of our problem, RNN required more time to learn than similar CNN model. They were a bit harder to fit and at the same time it was more difficult to find appropriate parameters for each layer. Experimentally it was proven, that the more layers, the higher is the probability of overfitting.

Overall results for prediction part of the task were not as high as with CNN, which can be observed in table 2.5. On the other hand, forecasting had slightly better performance when RNN were applied, rather than CNN. The results are visible in table 2.6. Although, there is a quite high probability that further hyperparameters adjusting could lead to even better result and that the desired model would be chosen only based on personal preferences.

Structure of the model is very similar to the structure we had with CNN, only instead of a convolutional layer, we now use Bidirectional LSTM layer. LSTM was already discussed earlier, only unknown here is "bidirectional". This word means that despite the fact that there seems to be one layer created, actually there are two. They are duplicates and are connected to the same output. One layer gets information from the "past", while another gets it from the "future".

Before this wrapper layer was tried, experiments without this wrapper were performed. They showed that the performance achieved with Bidirectional wrapper was significantly higher than without it.

The next step was adjusting the dimensionality of outputs space. It was experimentally proved that model that has lower dimensionality performs better than with higher number. Also, when higher numbers were entered, model training was extremely time consuming and the process of fitting data was slow.

Every other aspect of the model is the same as with CNN – activation funtcion of hidden layers is ReLU and the one for output layer is sigmoid.

2.3.3 Training process

Before training, the model should be compiled depending on the monitored metrics and loss function. For prediction part, monitored metrics were chosen to be accuracy and for forecasting it was a mean squared error.

Based on task different loss functions are defined. The loss function is another crucial part of neural networks. They are used to compute the difference between the predicted value and actual. The forecasting part is generally regression problem since we need to determine how given signal will progress in the future, therefore mean squared error is suitable loss function. The standard form of MSE loss function can be defined as:

$$\mathbf{L} = \tfrac{1}{n}\textstyle\sum_{i=1}^{n}(y^i - \hat{y}^i)^2$$

$(y^i - \hat{y}^i)$ is named *residual*, i.e. the difference between actual value y^i and predicted $\hat{y}^i$. Loss function has a task to minimize the sum of squared residuals.

Regarding the prediction part, it can be represented as simple binary classification, where labels could be only 1 or 0, i.e. true or false. For these types of problems, there is special loss function called *cross-entropy*. The loss function is denoted as following:

$$\mathbf{L} = -\frac{1}{n}\sum_{i=1}^{n}[y^i log(\hat{y}^i) + (1-y^i)log(1-\hat{y}^i)]$$

If the value of cross-entropy is large, the difference between two distributions is large, too, and other way around – if the difference is small, two distributions are similar.

2.4 Performance

It was already mentioned that performance for forecasting was similar for both RNN and CNN, while for prediction CNN achieved more impressive results. Although what is worth mentioning, is the fact that using standard ML algorithms to solve the prediction task, led to the same performance results (for such algorithms as K-NN) as when using neural networks. Moreover, in case of RNN, all of the standard ML algorithms achieved much better results than RNN. This, of course, could be caused by the lack of data and/or wrong RNN hyperparameter adjusting.

After observing the outcome, the important question arises: "Is it worth using neural networks for prediction instead of standard ML algorithms?" The answer is not simple for the following reasons:

1. The data
2. Hyperparameters adjustment
3. Desired accuracy

In order to have neural networks model that can reach extremely high accuracies it is required to have the corresponding amount of data to train it. Our dataset contains only 1 500 lines of data, which is not enough to create efficient model without it being overfitted. Hence, to have considerably good model, more data need to be gathered and also maintained. The reesulting dataset could be large enough, making maintaining data another problem, which was not the case here. The, after gathering preferred data, model training will be even more time consuming leaving us waiting for a long time until it is trained and validated.

Another problem is to choose the right parameters for our model. Even after it is trained for the first time, rarely it is performing well enough to give impressive results. Often, it is required to spend much time adjusting hyperparameters – in other words, experimenting. This, combined with the problem of time-consuming, leaves us to spend even more time creating anefficient model.

Also, RNN generated an extremely large amount of false negatives. It can be crucial in cases when misclassification can cost people lives. So, it is needed to perform experiments and find either parameters or additional layers that will reduce the number of false negatives to a much smaller amount.

Although after this process is finished, we can generate an extremely accurate model that will almost unmistakeably predict the class of the given waveform. However, as we know, standard ML algorithms are capable enough to achieve competitive and good results. Given all this, it is the choice of each individual.

The forecasting problem, on the other hand, does not leave us choice – it is not possible to solve using standard Machine Learning algorithms. Here, it was decided to experiment with different types of neural networks. The same 3 problems mentioned earlier are applicable here, too. For instance, the lack of data caused the algorithm to output similar results for different instances. However, overall performance can be considered as high and feeding the model more data can make it even higher. Here, both CNN and RNN performed with similar MSE, although RNN was a little bit better.

To sum up, prediction task can be solved relatively efficient using standard ML algorithms and it could be possible to increase it performing different data preprocessing, while neural networks, especially CNN, have the potential for generating even better outcomes. The forecasting task, on the contrary, is not trivial, therefore it can be solved using methods of supervised learning, in our case – neural networks. It requires more data, right parameter adjusting and plenty of time, in order to create an efficient model that can one day save people's lives.

Model	**Accuracy (%)**
1D CNN	81%
RNN	65%

Table 2.5: Prediction accuracy.

Model	**RMSE**
1D CNN	0.093
RNN	0.083

Table 2.6: Forecasting performance.

Conclusion

The aim of this thesis was to explore various algorithms for earthquake prediction and forecasting. Many people have taken their part to develop an effective solution to both of these problems and these solutions were discussed here. The major issue still is that people do not possess enough bits of knowledge to predict time, location and magnitude of the next earthquake. All scientists and researchers that claimed to solve this problem, did not achieve results that are good enough to be considered as a valid solution.

One of the most important parts of this thesis was to find suitable data that could be the basis for all experiments. Performance and results depend not only on models that were used but also on the quality of the gathered data. Different preprocessing techniques were tried in order to reduce noise that was present in the data and the optimal solution was found.

Few Python scripts were created in order to retrieve data and load them into the dataset in an appropriate form that allowed us to perform experiments on them. There are two different scripts that create datasets for prediction and for forecasting and one script that downloads the data from seismic stations using methods from ObsPy framework – the framework specifically created for seismologists.

The approach that was discussed in this thesis differs from the classical approach amongst other researchers. We did not try to find place and time of the next earthquake, on the contrary, it was decided to gather different waveforms and forecast will there be an earthquake in the next hour, based on the data we have right now. And after generating possible waveform for the next hour, predict whether this waveform contains readings signaling about the earthquake.

Different models were created and their performance was measured. Prediction is relatively efficient solved using standard ML algorithms, such as K-NN. Forecasting, on the other hand, can be generated with neural networks and standard ML algorithms are not applicable in this case. Experimentally, it was proven that RNN works a little bit better than CNN, although further hyperparameters adjusting and gathering more data can influence this model to work differently and better.

For further planned work we would outline these points:

1. gather more clean and wide data from seismic stations
2. experiment with data preprocessing, reducing possible remaining noises
3. studying neural layers that activate true positive cases for the prediction problem

Bibliography

1. GRANT, R. A.; HALLIDAY, T. Predicting unpredictable; evidence of preseismic anticipatory behaviour in the common toad. *Journal of Forecasting*. 2010.
2. DAVISON, C. *The Founders of Seismology*. Arno Press, 1978.
3. EARLE, S. Physical Geology. 2015. Available also from: `https://opentextbc.ca/geology/front-matter/preface/`.
4. GELLER, R. J.; JACKSON, D. D.; KAGAN, Y. Y.; MULGARIA, F. Earthquakes cannot be predicted. *Science*. 1997, vol. 275.
5. UYEDA, S. The VAN method of short-term earthquake prediction. *INCEDE newsletter*. 1997.
6. KANE, F. *Hands-on Data Science and Python Machine Learning*. Packt Publishing, 2017.
7. KLOUDA, K.; VASATA, D.; LOPEZ, J. P. M. BI-VZD lectures slides. 2018. Available also from: `https://courses.fit.cvut.cz/BI-VZD/lectures/files`.
8. NAVLANI, A. *K-NN classification using scikit-learn*. 2018. Available also from: `https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn`.
9. SIMPSON, J.; WEINER, E. *English Oxford Dictionary*. Oxford University Press, 2019. Available also from: `https://en.oxforddictionaries.com/definition/overfitting`.
10. STAFF, Packt Editorial. Implementing 3 Naive Bayes classifiers in scikit-learn. 2018. Available also from: `https://hub.packtpub.com/implementing-3-naive-bayes-classifiers-in-scikit-learn/`.
11. TOTH, D. Decision Trees. 2018. Available also from: `https://hub.packtpub.com/decision-trees/`.

12. DENIL, M.; MATHESON, D.; FREITAS, N. de. Narrowing the gap: random forests in theory and in practice. 2014. Available also from: `http://proceedings.mlr.press/v32/denil14.pdf`.

13. DONGES, N. The Random Forest Algorithm. 2018. Available also from: `https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd`.

14. SUBRAMANIAN, G. Rotation Forest – A Classifier Ensemble Based on Feature Extraction. 2015. Available also from: `https://hub.packtpub.com/rotation-forest-classifier-ensemble-based-feature-extraction/`.

15. RODRIGUEZ, J. J.; KUNCHEVA, L. I.; ALONSO, C. J. Rotation Forest: A New Classifier Ensemble Method. 2006. Available also from: `https://ieeexplore.ieee.org/document/1677518/citations?tabFilter=papers`.

16. SWAMINATHAN, S. Logistic Regression - Detailed Overview. 2018. Available also from: `https://towardsdatascience.com/logistic-regression-detailed-overview-46c4da4303bc`.

17. MOLNAR, C. *Interpretable Machine Learning*. Christoph Molnar, 2019. Available also from: `https://christophm.github.io/interpretable-ml-book/`.

18. MISHRA, A. Metrics to Evaluate your Machine Learning Algorithm. 2018. Available also from: `https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234`.

19. PEROL, T.; GHARBI, M.; DENOLLE, M. Convolutional neural network for earthquake detection and location. *Science Advances*. 2018, pp. 8.

20. CHU, S.; MAURER, J. Can machine learning determine physical source properties of earthquakes from a single station? *CS 229 poster*. 2016.

21. IBRAHIM, M. Al; PARK, J.; ATHENS, N. Earthquake warning system: Detecting earthquake precursor signals using deep neural networks. *CS 229 poster*. 2018.

22. KRIESEL, D. *A Brief Introduction To Neural Networks*. 2005. Available also from: `http://www.dkriesel.com/_media/science/neuronalenetze-en-zeta2-2col-dkrieselcom.pdf`.

23. GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

24. GREGOR, K.; DANIHELKA, I.; GRAVES, A.; REZENDE, D. J.; WIERSTRA, D. DRAW: A Recurrent Neural Network For Image Generation. 2015. Available also from: `https://arxiv.org/pdf/1502.04623v2.pdf`.

25. BULLINARIA, J. A. Recurrent Neural Networks. 2015. Available also from: `http://www.cs.bham.ac.uk/~jxb/INC/l12.pdf`.

26. OLAH, C. Understanding LSTM Networks. 2015. Available also from: `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`.
27. TRABANT, C. *SEED Reference Manual.* 2012. Available also from: `https://www.fdsn.org/media/_s/publications/SEEDManual_V2.4.pdf`.
28. CUN, Y. Le; BENGIO, T. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks.* 1995, pp. 255–258.
29. SRIVASTAVA, N.; HINTON, G.; KRIZHEVSKY, A.; SUTSKEVER, I.; SALAKHUTDINOV, R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research.* 2014, vol. 15, pp. 1929–1958. Available also from: `http://jmlr.org/papers/v15/srivastava14a.html`.
30. SAK, H.; SENIOR, A.; BEAUFAYS, F. *Long Short-Term Memory recurrent neural network architectures for large scale acoustic modeling.* 2014. Available also from: `https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/43905.pdf`.

Appendix **A**

Acronyms

CNN Convolutional Neural Networks

RNN Recurrent Neural Networks

LSTM Long Short-Term Memory

MSE Mean Squared Error

K-NN K-Nearest Neighbors

ML Machine Learning

Appendix B

Contents of enclosed flash disk

```
readme.txt..................the file with flash disk contents description
notebooks..the directory with Jupyter notebooks and pretrained models
thesis.................the directory of LaTeX source codes of the thesis
text.........................................the thesis text directory
  thesis.pdf...........................the thesis text in PDF format
```