ADAM MICKIEWICZ UNIVERSITY
FACULTY OF MATHEMATIICS AND COMPUTER SCIENCE

Łukasz Pawluczuk
album number: 362663

Automatic Summarization of Polish News Articles by Sentence Selection

Automatyczna sumaryzacja artykułów prasowych w języku polskim metodą selekcji zdań

Master Thesis in:
COMPUTER SCIENCE
INTELLIGENT SYSTEMS
Supervisor:
prof. dr hab. Krzysztof Jassem

Poznań 2015

Keywords

Automatic Text Summarization, Sentence Selection, Corpus-based Methods, Machine Learning, Neural Networks

I would like to express my sincere gratitude to my Master's Thesis
advisor prof. dr hab. Krzysztof Jassem, for
his patience, motivation, enthusiasm, and
immense knowledge. His guidance helped me
in all the time of research and writing of this thesis.

I would also like to thank Monika, who
always believed in me and encouraged me in difficult times.

Contents

Declaration

Poznań, June / Czerwiec 2015

I, the undersigned Łukasz Pawluczuk, student of Faculty of Mathematics and Computer Science at Adam Mickiewicz University in Poznań, Poland, declare that the submitted thesis titled *Automatic Summarization of Polish News Articles by Sentence Selection* was written by me. This means that in writing this work, except for the necessary consultations, I did not use the help of others, and in particular I did not commission development of the paper or any part of it, as well as I did not copy the thesis or its part from other people.

I also certify that the copy of the work in print is in line with the copy in electronic form.

At the same time I acknowledge that if these statements were false, the decision to grant me a diploma will be revoked.

[YES]* - I agree to share this Thesis in the reading room of UAM Archives

[YES]* - I agree to share this Thesis to the extent necessary to protect my right to authorship or rights of third parties

Ja, niżej podpisany Łukasz Pawluczuk, student Wydziału Matematyki i Informatyki Uniwersytetu im. Adama Mickiewicza w Poznaniu, oświadczam, że przedkładaną pracę dyplomową pt. *Automatyczna sumaryzacja artykułów prasowych w języku polskim metodą selekcji zdań* napisałem samodzielnie. Oznacza to, że przy pisaniu pracy, poza niezbędnymi konsultacjami, nie korzystałem z pomocy innych osób, a w szczególności nie zlecałem opracowania rozprawy lub jej części innym osobom ani nie odpisywałem tej rozprawy lub jej części od innych osób.

Oświadczam również, że egzemplarz pracy dyplomowej w formie wydruku komputerowego jest zgodny z egzemplarzem pracy dyplomowej w formie elektronicznej.

Jednocześnie przyjmuję do wiadomości, że gdyby powyższe oświadczenie okazało się nieprawdziwe, decyzja o wydaniu mi dyplomu zostanie cofnięta.

[TAK]* - wyrażam zgodę na udostępnienie mojej pracy w czytelni Archiwum UAM

[TAK]* - wyrażam zgodę na udostępnienie mojej pracy w zakresie koniecznym do ochrony mojego prawa do autorstwa lub praw osób trzecich

Łukasz Pawluczuk

Abstract

This thesis describes the automatic summarization system developed for the Polish language. The system implements trainable sentence-based extractive summarization technique, which consists in determining most important sentences in document due to their computed salience. Neural networks were used as a machine learning algorithm. A structure of the system is presented, as well as the evaluation methods and achieved results. The presented attempt is the first summarization project evaluated against the Polish Summaries Corpus, the standardized corpus of summaries for the Polish language.

Niniejsza praca opisuje system zdolny do automatycznego streszczania tekstu napisanego w języku polskim. System implementuje metodę automatycznego streszczania, polegające na selekcji zdań. Metoda ta ma na celu wybranie najważniejszych zdań z tekstu, na podstawie ich obliczonej ważności. Do stworzenia systemu wykorzystano algorytm uczenia maszynowego — sieci neuronowe. Praca przestawia zarówno strukturę systemu, jak i sposoby oraz wyniki jego ewaluacji. Opisane podejście prezentuje pierwszy system automatycznego streszczania, do którego ewaluacji wykorzystano Polish Summaries Corpus. Jest to zbiór ręcznie stworzonych streszczeń dla tekstów napisanych w języku polskim.

Introduction

Automatic text summarization is a very active research field in recent years. Its purpose is to reduce a text document, by extracting its most important parts in order to create more condensed, but still human-readable form, known as summary. The task consists in the creation of an appropriate computer application and a framework for testing and evaluation.

The scope of this work includes:

- developing a sentence-based extractive summarization system, which would be able to create summaries of newspaper articles,
- implementing well-known automatic summarization techniques, as well as introducing experimental ones,
- tuning neural networks algorithm in order to achieve satisfactory performance,
- using the Polish Summaries Corpus, a resource created by Ogrodniczuk and Kopeć in [16] to train the automatic summarization system, what has not been done so far,
- evaluating the performance of the system using well-known standardized evaluation method, based on the ROUGE summarization evaluation package introduced at Document Understanding Conference (DUC) in 2004, by Chin-Yew Lin [10],
- performing comparison of different evaluation methods for automatic summarization.

The thesis is organized as follows: Chapter 1 briefly describes the aim of the summarization task and main methods in the field. Chapter 2 introduces the neural networks theory. Chapter 3 provides a review of already existing summarization systems for the Polish language. In chapter 4 Polish Summaries Corpus is described in detail. Chapter 5 outlines our solution, it's overall framework, as well as the employed set of features. Chapter 6 introduces the evaluation methodology and presents executed experiments and their results. Eventually, Chapter 7 contains some conclusions and the outline for future work.

CHAPTER 1

Automatic Summarization

This chapter briefly describes the automatic summarization task. The main aims of automatic summarization are pointed out as well as basic definitions are introduced. Most important methods of automatic summarization are introduced too.

1.1. Aim of automatic summarization

Modern digital technologies, including World Wide Web, result in information excess. Everyday brings vast amount of new on-line information of various type. Processing this continuously growing information databases is not possible by a single human. Automatic summarization is an attempt to confront information processing needs. It is based on the assumption that a computer system can read all data quickly and present its condensed from.

In fact, different forms of summarization are processes, which occur in everyday life in many areas. In the midst of a huge number of examples are such as [12]:

- Newspaper headlines, which can be found as some form of summaries,
- summaries in the body of news stories,
- show previews or trailers,
- abstracts of scientific articles,
- conference programs,
- stock market bulletins.

A summary may concern many different multimedia forms, such as: text, picture, movie or audio segment. Summary input may be represented in one of these forms or in a combination of them. However, this broad description of summarization is hard to be adopted in automatic process, where strict formal and intuitive definitions are needed. This thesis covers automatic document summarization, where input, as well as output form a coherent text. This form of summarization can be formalized as follows [Lloret]:

Definition 1. *(Summary). Summary can be defined as a text that is produced from one or more texts, that contains a significant portion of information in the original text(s), and that is no longer than half of the original text(s).*

Following on this definition of summary, a concept of **summarizer** is defined [12]:

Definition 2. *(Summarizer). Summarizer is a system whose goal is to produce a condensed representation of the content of its input for human consumption.*

These two definitions may form the basics for the automatic document summarization field, indicating its main goals and outlines.

1.2. Basic notions of summarization

Automatic text summarization may be classified according to program's input or output. As regards input, summarization may concern one document (**single-document summarization**) or multiple documents (**multi-document summarization**).

As regards output, three essential distinctions should be introduced. These distinctions are not exclusive as they refer to different aspects of summarization.

Firsly, one may distinguish [12]:

Extract

A summary that entirely consists of text copied from the input. Extracted text can contain paragraphs, sentences, phrases, terms or even single nouns. These summaries are obviously easier to obtain, since the extracted elements of original text are not modified at all.

Abstract

A summary that consists of at least some material that is not present in the input. Source text may be changed, so a more condensed form is possible in this case.

Secondly, summaries may be distinguished according to information they contain [Lloret]:

Indicative

A summary that indicates source text's topics and gives a brief idea of what the original text is about. This type of summarization may provide some references for more in-depth reading of original documents.

Informative

A summary that covers the whole topics in the source text. Mani ([12]) states that the aim of the informative summary is to cover "all the salient information in the source at some level of detail". Informative summarization can in fact be viewed as a proper subset of indicative ones.

Critical evaluative abstract

A summary that takes into account the abstractor's view on the quality of the source

document. An example of an abstract is a review. In fact, this type of abstracts is beyond the automatic summarization as it provides information, which is not present in the source document, as: opinions, recommendations of feedback.

The last essential distinction refers to the type of the user the summary is intended for. In this case, one can distinguish:

Generic summaries

A summary that is created for broad audience, with no particular, precise aim. Often, it serves as a replacement of original document, although it can be indicative as well.

User-focused (a.k.a. query-driven) summaries

A summary tailored to the requirements of a particular user or group of users [12]. It may respond to user's information needs expressed as topic or query [Lloret]. In some cases a user-focused summary may take into account some kind of user model or profile to adjust the summary, without the query.

Another distinction may refer to the language of the text and the summary. The following summarization types can be determined:

Monolingual

All documents, as well as results are in the same language.

Multilingual

Input documents are in different languages and the results are in the same languages as its sources.

Cross-lingual

Input documents are in different languages, but the result is in a single language, which may even be different than input ones.

A summary may be also restricted to a particular **sublanguage**, for example the technical language.

Another important notion in summarization is the **coherence**. It indicates how various text segments combine into an integrated whole. If the text segments are disjointed, refer to the same ideas, have gaps in reasoning or in general are not well organised, then the text is regarded as **incoherent**. Random collection of sentences is an extreme example of incoherent text [12]. The need for coherence depends of the summarization purpose and type of summarized documents.

All the distinctions mentioned above should be taken into account by the summarization system. They can be summed up and complemented as follows: [12]:

- Compression rate — summary length/source length

- Relation to source — extract/abstract
- Function — informative/indicative
- Audience — user-focuses/generic
- Coherence — coherent/incoherent
- Span — single-document/multi-document
- Language - monolingual/multilingual/cross-lingual
- Genre — type of the text: technical reports, news stories, email messages etc.
- Media — media types concerned (text, audio, pictures)

A definition of **salience** is crucial for summarization methods [12]:

Definition 3. *(Salience). Salience is the weight attached to information in a document, reflecting both the document content as well as the relevance of the document information to the application.*

This quite loose, theoretical definition indicates the requirement of summarization methods to indicate the relevance of parts of document.

1.3. Methods of summarization

Literature often considers automatic summarization a three-stage process. Lloret (2006) names the following steps of the process:

1. **interpretation** of the source text in order to obtain a text representation,
2. **transformation** of the text representation into a summary representation,
3. **generation** of the summary text from the summary representation.

As stated by Mani ([12]), the idea of multi-dimensional linguistic charts is useful to visualise possible relationships in summarized **elements** and **levels** of linguistic analysis. This conception is presented at Figure 1.1. Vertical axis represents various text parts, such as words, phrases, sentences or paragraphs. Horizontal axis — position, indicates the ordering of the elements in the input. Possible level of linguistic analysis of these elements are assigned to in-depth axis. There are various levels of linguistic analysis: morhological, syntactic, semantic and discourse. Interpretation is then viewed as a process of going from shallow to deeper level of analysis and generation goes the reverse direction.

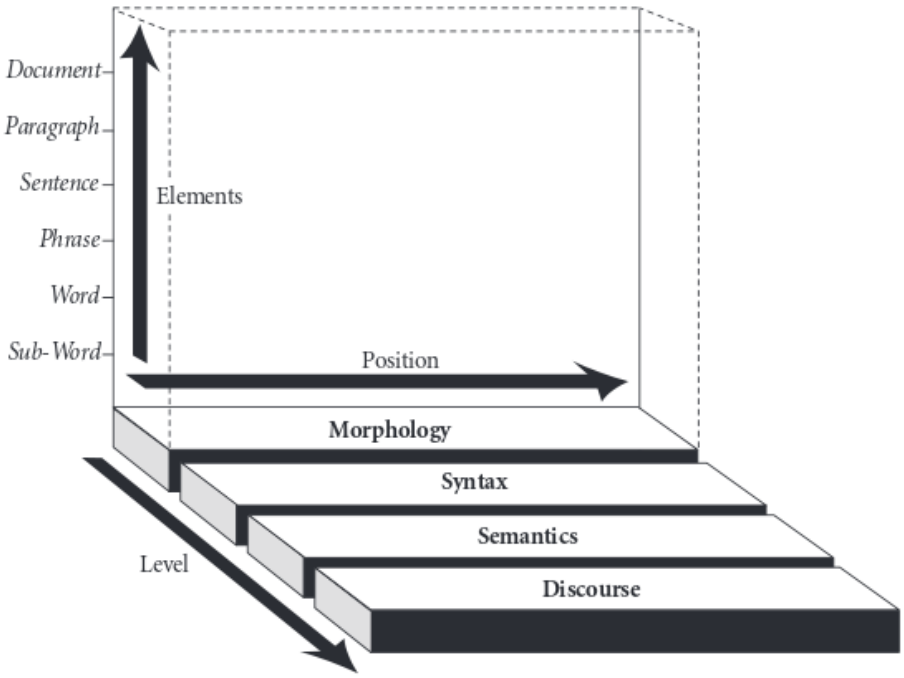

Figure 1.1. The Linguistic Space (source: [12])

According to the linguistic space, methods of text summarization may differ as far as the level of processing is concerned: *surface*, *entity*, or *discourse* levels [14]. It is worth noting that there exist systems, which adopt hybrid-approaches.

Surface-level approaches make use of shallow features to analyze information included in a text document. Usually, these features are combined together into a salience function used to extract information. Examples of such features are:

- Thematic features — based on term frequency analysis and statistically salient terms,
- Location features — based on position in text, paragraph or section depth,
- Background features — based on presence of title or headings terms, or a user's query,
- Cue words and phrases — based on presence of special 'bonus' or 'stigma' terms.

Entity-level approaches are based on the internal representation of text. They model text entities and their relationships across a document. Examples of such relationships between entities are:

- Similarity — e.g. vocabulary overlap,

- Proximity — distance between text units,
- Co-occurrence — words occurring in common contexts,
- Thesaural relationship among words — e.g. synonymy, hypernymy,
- Coreference — e.g. anaphora, cataphora, noun phrases,
- Logical relations — e.g. agreement, contradiction, entailment, consistency,
- Syntactic relations — e.g. relations based on parse trees,
- Meaning representation-based relations — e.g. predicate-argument relations.

Discourse-level approaches model the global structure of text, and its relation to communicative goals. Examples of such structures are:

- Format of the document,
- Threads or topics as they are revealed in the text,
- Rhetorical structure of the text.

1.4. Extraction

This thesis applies the extraction method of automatic summarization. Extraction approaches tend to use shallow linguistic analysis. Sentence semantics is often avoided. Smaller segments, such as words, are taken into account. These methods usually avoid deep text understanding, emphasizing simplicity, ease of implementation and computing time [24]

Extractive summaries are created by extracting key text segments from the text. The most important part of these methods is to determine salient text units. This can be done by looking at the text unit's lexical and statistical relevance or by matching phrasal patterns [6]. The resulting summary is a concatenation of original parts of the source. Sentences are quite popular choice as a size of extraction units. Usually, the particular text unit's salience is determined by the linear weighting model, which can be represented by formula (1.1), where $A(u), B(U), \ldots, Z(U)$ are calculated values of features (thematic, locational, etc.), and $\alpha, \beta, \ldots, \omega$ are weights assigned to them.

$$Salience(U) = \alpha * A(U) + \beta * B(U) + \cdots + \omega * Z(U) \tag{1.1}$$

Selection of features as well as the weight tuning may be done manually. There are also attempts to create supervised machine learning algorithms, which are based on a set of documents and their manually created summaries (**corpus**). In such cases the weights are determined by these algorithms. It must be emphasized that the importance of features often varies with the system. Mani ([12]) states, that discovering good features for a classification

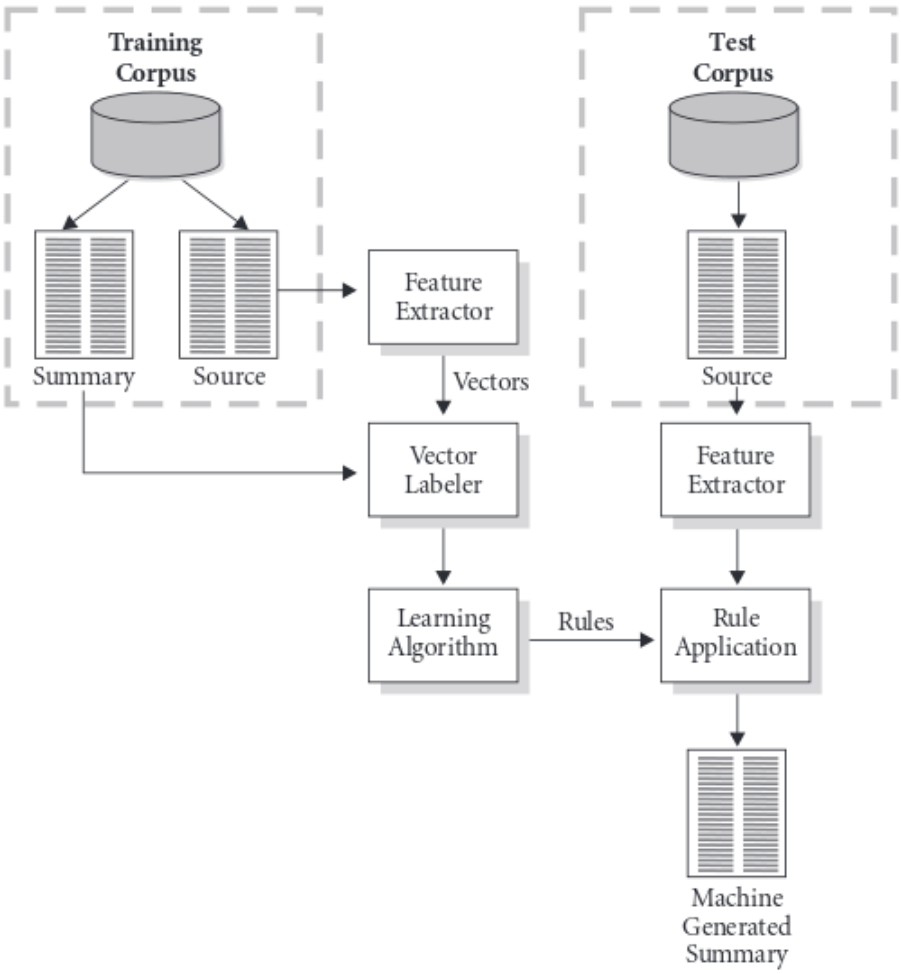

Figure 1.2. Corpus-based approach to sentence extraction (source: [12])

is a crucial, yet not easy, task. Usually, a system employs a mix of 'well-established' features discussed in literature, along with new experimental features.

The conception of a trainable corpus-based extractive system is presented in Figure 1.2. Mani describes this conception as follows [12]:

> A corpus of documents and their summaries are required for training the summarizer. Elements (usually sentences) from each source document are extracted, analysed in terms of the features of interest, and then labelled by comparison with the corresponding summary. The labelled feature vectors are then fed to a learning algorithm, which emerges with rules that can be used to classify each sentence of a test document as to whether it should be extracted into a summary. The accuracy of the classifier can then be measured against *held-out test data*, i.e., data the classifier has not been trained on.

Several key aspects have been introduced in the above quotation. These can be clarified as follows:

Labelling is a procedure of comparing the source document sentence with the summary. It is a trivial task if the summary is an extract, not an abstract. The goal is to analyse the sentences from the source for the features, which make them belong in the extract. Analysing abstract summaries in order to create extracts is much harder to do and will not be covered here. In fact, labelling may be a $\{0, 1\}$ function or a continuous one.

Learning representation depends on the machine learning algorithm used, as well as on the system implementation. The result of learning may be represented as a set of rules or mathematical functions and it should have the machine-readable form.

Evaluation of an automatic summarization system is a complex task and will be discussed in Chapter 6. The main goal is to measure the accuracy of the system against a reference summary (or a set of summaries). Evaluation regards such issues as agreement between references and adequate measurement techniques.

CHAPTER 2

Neural Networks

This chapter will introduce one of the most common approaches to machine learning, namely neural networks. Machine learning is an artificial intelligence discipline, which takes into account autonomous learning of computer software. Neural networks are an attempt to provide a mathematical model of human brain and its computing structure. It is worth noting that neural networks are only a theoretical model which does not translate precisely to brain's complexity of cognition.

According to Stuart Russel's and Peter Norvig's "Artificial Intelligence: A Modern Approach" book:

> We do know that the **neuron**, or nerve cell, is the fundamental functional unit of all nervous system tissue, including the brain. Each neuron consists of a cell body, or **soma**, that contains a cell nucleus. Branching out from the cell body are a number of fibres called **dendrites** and a single long fibre called the **axon**. Dendrites branch into a bushy network around the cell, whereas the axon stretches out for a long distance—usually about a centimeter (100 times the diameter of the cell body), and as far as a meter in extreme cases. Eventually, the axon also branches into strands and substrands that connect to the dendrites and cell bodies of other neurons. The connecting junction is called a **synapse**. Each neuron forms synapses with anywhere from a dozen to a hundred thousand other neurons. [19]

Figure 2.1 presents a visual conceptualisation of the above description. Complicated electrochemical reactions are used to propagate signals in this network. Chemical transmitter substances enter the dendrite, altering electrical potential of the cell body. If the potential reaches a threshold, an electrical pulse is released through the axon and transmitted to other neurons. This creates the most basic cognition mechanism of the brain. Different patterns of stimulation in a network can cause a long-term changes in the strength of connections and even sometimes entire collections of neurons can change its structure significantly. These phenomenon, called neroplasticity, is believed to form the basics for learning in the brain [19]. The following sections present the theoretical background for the artificial neural networks, which try mimic this structure.

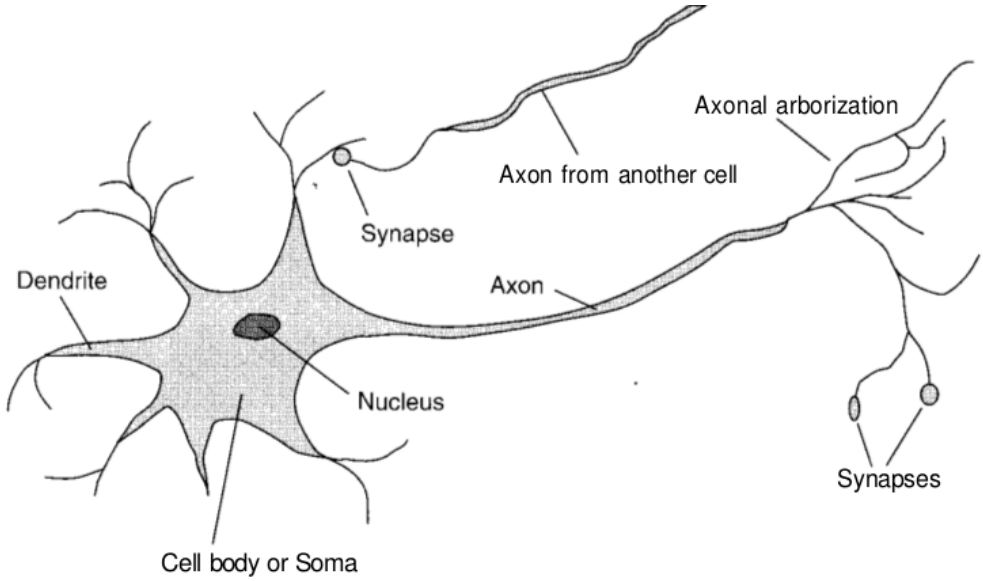

Figure 2.1. The parts of a neuron (source: [19])

2.1. Theoretical background

Figure 2.2 presents an abstract neuron, called also a computing unit. It has n inputs, where each input i can transmit a real value x_i altered by the corresponding weight w_i. In general, these inputs are the outputs of other neurons. All these n values are an input for the primitive function f computed in the body of the neuron, resulting in value y [18].

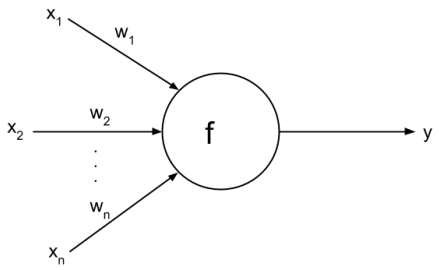

Figure 2.2. An abstract neuron with n inputs and their corresponding weights.

This is simple abstraction of a single node in neural network. Rojas states that: "If we conceive of each node in an artificial neural network as a primitive function capable of transforming its input in a precisely defined output, then artificial neural networks are nothing but networks of primitive functions" [18]. This idea can be formalized as follows [8]:

Definition 4. *(Neural Network). A* ***neural network*** *is a sorted triple* (N, V, w) *with two sets* N*,* V *and a function* w*, where* N *is the set of neurons and* V *a set* $\{(i,j)|i,j \in \mathbb{N}\}$ *whose*

elements are called ***connections*** *between neuron i and neuron j. The function $w : V \to \mathbb{R}$ defines the* ***weights****, where $w((i, j))$, the weight of the connection between neuron i and neuron j, is shortened to w_{ij}.*

Each neuron in a network is desired to do some computations based on delivered input and a given set of weights. Usually, a neuron needs to compute a sequence of functions in order to return the value. These functions include [8]:

Propagation function — for a neuron j this function receives the outputs $o_{i1}, \ldots, o_{in}$ from other neurons $i_1, \ldots, i_n$ (which are connected to given neuron), and transforms them in consideration of the connecting weights $w_{i,j}$ into the *network input* net_j. Formally:

Definition 5. *(Propagation Function and network input). Let $I = \{i_1, \ldots, i_n\}$ be the set of neurons, such that $\forall z \in \{1, \ldots, n\} : \exists w_{i_z,j}$. Then the network input of j, called net_j, is calculated by the propagation function f_{prop} as follows:*

$$net_j = f_{prop}(o_{i1}, \ldots, o_{in}, w_{i_1,j}, \ldots, w_{i_n,j}) \tag{2.1}$$

The weighted sum is a very popular choice for the propagation function. It is computed as follows:

$$net_j = \sum_{i \in I} (o_i \cdot w_{i,j}) \tag{2.2}$$

Activation function — determines the reaction of neuron j to the input values. It indicates a neuron's activity in the network. To present formal definition of activation function, a concept of the threshold needs to be provided::

Definition 6. *(Threshold value in general). Let j be a neuron. The threshold value Θ_j is uniquely assigned to j and marks the position of the maximum gradient values of the activation function.*

In simple terms, the threshold value represents the threshold at which a neuron starts firing.

Definition 7. *(Activation function). Let j be a neuron. The activation function is defined as*

$$a_j(t) = f_{act}(net_j(t), a_j(t-1), \Theta_j) \tag{2.3}$$

Therefore, activation function is a function of a time. It transforms the network input net_j, as well as the previous activation state $a_j(t-1)$ of the neuron into a new activation state $a_j(t)$, with the threshold value Θ. The activation function is often defined globally for all neurons in the network and only the threshold values are different for each neuron.

There are some popular choices for the activation function, which include:

- binary threshold function (Heaviside function) — the activation state remains constant, unless the input is above a certain threshold. Therefore, it can take only two values.
- logistic function (Fermi function) — it maps the input to the range of (0,1). It is computed as in (2.4) formula.

$$\frac{1}{1+\mathrm{e}^{\frac{-x}{T}}} \tag{2.4}$$

  T is a temperature parameter. The smaller it is, the more does it compress the function on the x axis.
- hyperbolic tangent — it maps the input to the range of (-1,1).

These functions are presented at Figure 2.3.

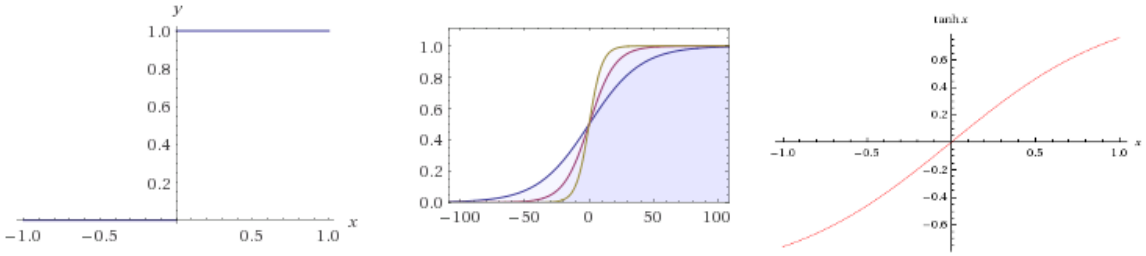

Figure 2.3. Example activation functions. From left to right: Heaviside function, Fermi function and hyperbolic tangent. Fermi function plot illustrates the function for different T values (source: wolframalpha.com)

Output function — calculates the values which are transferred to the other neurons connected to the given one. Formally:

Definition 8. *(Output function). Let j be a neuron. The output function*

$$f_{out}(a_j) = o_j \tag{2.5}$$

calculates the output value o_j of the neuron j from its activation state a_j.

The output function is also defined globally. Unless explicitly specified otherwise, it is the identity function, i.e. the output of the neuron is equal to its activation function.

Different network topologies can be applied to different problems. In general, the topology refers to the arrangement of the elements (nodes and connections) of a network. Generally, neural networks are classified into **feed-forward** and **feed-back** networks.

Feed-forward network

Christopher Bishop states in ([1]), that "In general we say that a network is feed-forward if it is possible to attach successive numbers to the inputs and to all of the hidden output units such that each unit only receives connections from inputs or units having a smaller number". Therefore, is feed-forward networks the signals can only travel in one direction. In feed-forward neural networks neurons are grouped in **layers**. There is one **input layer**, a number of **hidden layers** and one **output layer**. Input and output layers act as a communication with the outside world. Data is forwarded to the input layer, which performs calculations. The results of this calculations become then the new input values for the next layer. This process continues until it has gone through all the layers and the network's results are returned through the output layer. Hidden layers are invisible from the outside of network. Neurons placed in this layers are called **hidden neurons**. In this topology, each neuron in one layer has only directed connections to the neurons of the next layer [8]. **Completely linked** layers are ones in which every neuron is connected to all neurons of the following layer. Figure 2.4 illustrates an example of completely linked feed-forward network. This thesis uses the feed-forward networks.

Feed-back network

In feed-back network the signals can travel in both directions and loops are possible. In general, all possible connections between neurons are allowed. This creates an internal, temporal states of the network. These networks do not always have explicitly defined input or output neurons [8]. Feed-back networks are also called the **recurrent neural networks**.

There is a popular practice to replace a threshold value with a **bias unit** in such networks. Is is realized as an extra neuron in the layer and its activation function is always 1. Through the weight assigned to this neuron, bias behaves as a threshold. This is a big facilitation in the implementation of a network, as well as its performance.

2.2. Learning

Learning algorithms can be divided into **supervised** and **unsupervised** methods. In case of unsupervised learning only the input patterns are given during network training. The network then tries to identify similar patters and classify them into similar categories. On the other hand, supervised learning assumes that the training set consists of input patterns, as well ass desired outputs of the network. Therefore, for each training set, the network modifies its weights in order to not only identify given input patterns in the future, but provide correct outputs to unknown, similar input patterns as well. The **reinforcement**

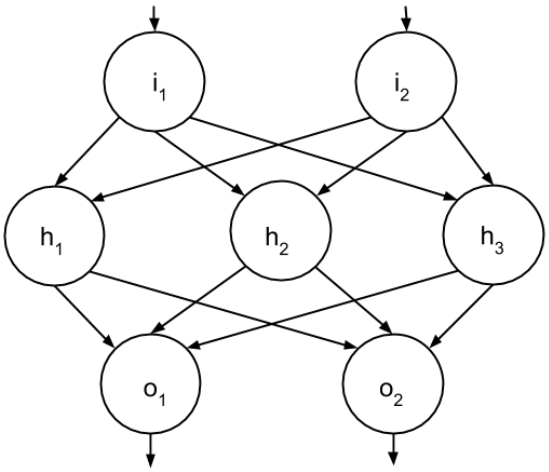

Figure 2.4. Feed-forward Neural Network. It has two input neurons, three hidden neurons and two output neurons.

learning is introduced as a type of learning, where instead of desired outputs, the network receives a measure whether the result was right or wrong and, possibly, how right or wrong it was [8].

The **backpropagation algorithm** is used in layered feed-forward neural networks in case of supervised learning. This algorithm indicates that the network's signal is propagated forward and then the errors are propagated backwards. The error in this case is the difference between actual and expected results. The aim of this algorithm is to reduce this error during training. Usually, the training begins with random weights at connections in network and after the training is done, these weights are adjusted in order to return minimal error [5].

The **error function** for the output of each neuron can be defined as in the (2.6) formula. The a_j is the activation state of the jth neuron and d_j is the desired activation state. It is the square of the difference, because it is always positive and it will be greater if the difference is big, and lesser if the difference is small [5].

$$E_j(d) = (a_j - d_j)^2 \tag{2.6}$$

Then the error of the network can be defined as the sum of the errors of all neurons in the output layer. It is presented at (2.7) formula.

$$E_j(d) = \sum_j (a_j - d_j)^2 \tag{2.7}$$

The **gradient descent** method is defined as in (2.8) formula. It is used to adjust the weights in the network during training.

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} \tag{2.8}$$

Therefore, the adjustment of each weight w_{ij} is equal to the negative constant η multiplied by dependence of the previous weight on the error of the network. The size of the ajdustment depends on η and the contribution of the weight to the error [5].

Lastly, the **linear separability** is defined as follows [18]:

Definition 9. *(Linear Separability). Two sets A and B of points in an n-dimensional space are called absolutely linearly separable if there exist real numbers* $w_1, \ldots, w_{n+1}$ *, such that every point* $(x_1, \ldots, x_n) \in A$ *satisfies* $\sum_{i=1}^{n} w_i x_i \geq w_{n+1}$ *and every point* $(x_1, \ldots, x_n) \in B$ *satisfies* $\sum_{i=1}^{n} w_i x_i < w_{n+1}$.

2.3. Parameters

From a practical point of view, there are several parameters that must be taken into account when designing a neural network. These parameters affect significantly the network's performance, learning capabilities and accuracy. It is necessary to adjust them depending on the considered problem and the training dataset. Wrong adjustment of these parameters may lead to **underfitting** or **overfitting** the network. Underfitting means that the network will be too simple to predict output properly for a complicated dataset. Overfitting occurs when network's information processing capacity is much bigger than the amount of information, which can be provided by the dataset during training. Network is then more accurate in fitting data from the dataset, but less acurate in predicting new data. Neural network's parameters include:

Number of layers

Number of hidden layers is one of the most important parameters in neural network. Appropriate selection of the network topology is crucial in achieving satisfactory training results. Neural networks with different numbers of hidden layers are capable of solving different types of problems. According to Jeff Heaton ([7]) it is to possible to define an overall rule of determining the number of hidden layers, as presented in Table 2.1. In general, neural network with two hidden layers can represent a function with any kind of shape. Therefore, there is no reason to use any more than two layers.

Number of neurons per layer

Number of neurons in hidden layers is also very important and should be carefully selected. If there are too few neurons in the hidden layers, the underfitting may occur. Using too many neurons in the hidden layers may cause a problem of overfitting, which means, that not all neurons will be trained sufficiently. However, if the amount of data is sufficient, training network with too many neurons may take too much time [7].

Number of hidden layers	Result
none	Only capable of representing linear separable functions or decisions.
1	Can approximate any function that contains a continuous mapping from one finite space to another.
2	Can represent an arbitrary decision boundary to arbitrary accuracy with rational activation functions and can approximate any smooth mapping to any accuracy.

Table 2.1. Determining the number of hidden layers (source: [7])

Number of epochs

Number of epochs is a number of learning iterations over training data. It can also represent maximum number of iterations, if training uses minimum error parameter. Again, using too many epochs may lead to overfitting. On the other hand, using too few epochs may cause underfitting.

Learning rate

This parameter controls the size of weight changes during learning of the training algorithm. It can have a big impact on learning time, as well as a possibility of fitting the dataset.

Momentum

Momentum parameter m adds a fraction m of the previous weight update to the current one. It is used to prevent the network from converging to a local minimum. If momentum is high, it can increase the speed of convergence of the network, but can create the risk of not finding the minimum. On the other side, too low momentum may slow down the training.

Minimum error

Desired error function value of the training iteration. Network's training stops if this value is reached.

2.4. Tools

2.4.1. PyBrain

PyBrain is a modular Machine Learning library for Python [20]. It consists of a number of easy-to-use machine learning algorithms, including neural networks. PyBrain is an abbrevi-

ation for Python-Based Reinforcement Learning, Artificial Intelligence and Neural Network Library. It is a flexible and effective tool. The library is open-source and free to use (licensed under BSD Software License). Some examples of using PyBrain are below:

Creating neural network

Listing 2.1 ilustrates an example of how to use the shortcut function *buildNetwork* to create the neural network. This simple call returns a network that has two input, three hidden and a single output neuron. More sophisticated networks can also be created using this function. Its optional parameters can be used to set hidden or output layers type, or weather to use bias or not. In fact, one of the library's biggest strengths is the possibility of creating very sophisticated network topologies.

Listing 2.1. Creating neural network with PyBrain.

```
>>> from pybrain.tools.shortcuts import buildNetwork
>>> net = buildNetwork(2, 3, 1)
```

Activating a network

To calculate the network's output, there is the *.activate()* method, as ilustrated at Listing 2.2. It is possible to activate the network without training, as it is initialized with random values. This method expects a list, tuple or an array as an parameter. It corresponds to the network's input and its length should be the same as the number of input neurons in the network. The array returned by the network represents the output neurons its length is the same as number of output neurons. Its interpretation depends on the training data.

Listing 2.2. Activating neural network with PyBrain.

```
>>> net.activate([2, 1])
array([-0.98646726])
```

Examining the structure

Each layer in PyBrain has its own class, which determines its behaviour. Each layer has also its unique name, which distinguishes it from the other layers in the network. It is possible to access different parts of network using their names as illustrated at Listing 2.3. The *buildNetwork* function uses some names, as well as layer's classes by default. The default class for input and output layer is the *LinearLayer*, which is the name for the identical function in PyBrain. The default function for all hidden layers is the *SigmoidLayer*, which is some special case of logistic function.

Listing 2.3. Examining the neural network's structure with PyBrain.

```
>>> net['in']
<LinearLayer 'in'>
>>> net['hidden0']
<SigmoidLayer 'hidden0'>
>>> net['out']
<LinearLayer 'out'>
```

Different network's topologies

PyBrain allows to create very flexible network topologies. Each layer in network has assigned class, which indicates their neurons activation function. Listing 2.4 illustrates a simple example of customized network. In this case the TanhLayer is used for the hidden layers, what indicates using hyperbolic tangent for them and the SoftmaxLayer, a normalized exponential, is used for the output layer.

Listing 2.4. An example of creating neural network with non default topology with PyBrain.

```
>>> from pybrain.structure import SoftmaxLayer
>>> net = buildNetwork(2, 3, 2, hiddenclass=TanhLayer, \
                                    outclass=SoftmaxLayer)
```

PyBrain uses its own dataset model, which is available in the *pybrain.dataset* package. The following examples presents the *SupervisedDataSet* class usage. It is a standard class used for supervised learning:

Creating the dataset

Listing 2.5 illustrated an example of creating the datasets. The *SupervisedDataSet* constructor creates dataset object with the specified size of input and target values. In the example above, the dataset has two dimensional inputs and one dimensional target.

Listing 2.5. Creating the dataset with PyBrain.

```
>>> from pybrain.datasets import SupervisedDataSet
>>> ds = SupervisedDataSet(2, 1)
```

Adding samples

To add new samples to the dataset, the *addSample()* method is used as presented in Listing 2.6.

Listing 2.6. Adding samples to the dataset with PyBrain.

```
>>> ds.addSample((0, 0), (0,))
>>> ds.addSample((0, 1), (1,))
>>> ds.addSample((1, 0), (1,))
>>> ds.addSample((1, 1), (0,))
```

Examining the dataset

An existing dataset can be examined in a few different ways. To check how many samples are included in a dataset, the python's *len()* function can be used as presented in Listing 2.7. In turn, Listing 2.8 illustrates how to iterate over the dataset. Input and target fields can be accessed directly as well, as presented in Listing 2.9.

Listing 2.7. Dataset size with PyBrain.

```
>>> len(ds)
4
```

Listing 2.8. Iterating over the dataset with PyBrain.

```
>>> for inpt, target in ds:
...     print inpt, target
...
[ 0.  0.] [ 0.]
[ 0.  1.] [ 1.]
[ 1.  0.] [ 1.]
[ 1.  1.] [ 0.]
```

Listing 2.9. Direclty accesing the dataset with PyBrain.

```
>>> ds['input']
array([[ 0.,  0.],
       [ 0.,  1.],
       [ 1.,  0.],
       [ 1.,  1.]])
>>> ds['target']
array([[ 0.],
       [ 1.],
       [ 1.],
       [ 0.]])
```

Lastly, in order to train the created neural network, the *trainers* module is used. Trainers take a neural network module and a dataset and trains the module to fit the data in the dataset. A training module can be used as follows:

Creating trainer

Listing 2.10 illustrates the creation of a backpropagation trainer. The *net* is the previously created neural network module and *ds* is a dataset. In fact, the dataset in this example is built for XOR function. Using optional parameters, it is possible to set the learning rate, as well as momentum values.

Listing 2.10. Creating a trainer with PyBrain example.

```
>>> from pybrain.supervised.trainers import BackpropTrainer
>>> trainer = BackpropTrainer(net, ds)
```

Training

In PyBrain there are two simple ways to train the neural network. Listing 2.11 presents an example of training the network for one full epoch using the *.train()* method. This method returns a double number proportional to the error. In turn, *.trainUntilConvergence()* method allows for training the network until convergence. This is presented in Listing 2.12. The last method returns a tuple containing the errors for every training epoch.

Listing 2.11. Training with PyBrain.

```
>>> trainer.train()
0.31516384514375834
```

Listing 2.12. Training until convergence t with PyBrain.

```
>>> trainer.trainUntilConvergence()
...
```

2.4.2. FANN

Fast Artificial Neural Network Library is a neural network library implemented in C. It is open-source and free to use, even commercially, as it is licensed under LGPL. It is easy to use, versatile, well documented, and fast. FANN has cross-platform execution in both fixed and floating point modes. It contains bindings to more than 20 programming languages, as well as the graphical interface. The following section presents an example usage of FANN with Python binding:

Preparing the dataset

A dataset for FANN library needs to be prepared in a separated file, using a specific format. Taking again XOR function as an example and assuming that -1 is a false value and 1 indicated true, the file saved as xor.data looks as in Listing 2.13. The explanation of this format is following: the first line contains three numbers: number of samples, number of input values for each sample and number of output values for each sample. In the example above, there are 4 samples, 2 input values per sample and one output. The rest of the file contains samples, where the inputs are placed interchangeably with target lines.

Listing 2.13. Dataset example with FANN.

```
4 2 1
-1 -1
-1
-1 1
1
1 -1
1
1 1
-1
```

Creating the network

Listing 2.14 illustrates a way of creating the neural network. The *.neural_net()* creates the neural network. Each *.create_sparse_array()* creates the network's topology. In this case the connection rate is 1, so each neuron from one layer is connected with each

neuron in the consecutive one. Other parameters can be set too using appropriate setter methods.

Listing 2.14. Creating the network using example parameters with FANN.

```
from pyfann import libfann

connection_rate = 1
learning_rate = 0.7

connection_rate = 1
learning_rate = 0.7
num_input = 2
num_hidden = 4
num_output = 1

ann = libfann.neural_net()
ann.create_sparse_array(connection_rate, \
                        (num_input, num_hidden, num_output))
ann.set_learning_rate(learning_rate)
ann.set_activation_function_output( \
                        libfann.SIGMOID_SYMMETRIC_STEPWISE)
```

The training

The training part may be performed as presented in Listing 2.15. xor.data is the previously saved dataset file. The *.save()* method is used for saving trained network to file.

Listing 2.15. Training the network using example parameters with FANN.

```
desired_error = 0.0001
max_iterations = 100000
iterations_between_reports = 1000

ann.train_on_file("xor.data", max_iterations, \
                  iterations_between_reports, desired_error)
ann.save("xor.net")
```

Activating the network

Listing 2.16 illustrates how to activate the neural network in order to calculate its output. Just two simple methods are used in this case: *.create_from_file()*, which loads previously trained network from file and *.run()*, which activates the networks. The output of this run is 1, which is the correct answer.

Listing 2.16. Activating the network with FANN.

```
ann = libfann.neural_net()
ann.create_from_file("xor.net")

print ann.run([1, -1])
```

CHAPTER 3

Review of experiments on summarization of Polish Texts

This section covers experiments on automatic summarization for the Polish language, resulting in theoretical works, as well as working implementations. All of them apply extractive methods of summarization.

3.1. PolSum2 (S. Kulikow)

The first attempt at automatic text summarization for the Polish language was made by Ciura et al. [2] and resulted in the *PolSum* system, which then evolved into *PolSum2*. The system is still available at `http://las.aei.polsl.pl/PolSum/`. *PolSum2* is an extractive system. It performs various kinds of text analysis (morphological, syntactic, semantic) in order to extract most important sentences from an input document. The system also recognizes anaphora, which results in better coherence between selected sentences.

PolSum2 performs in three stages of summarizing[2]. The first stage, called 'Calling remote analyzer' is intended to call the remote server, which performs text analysis. The *Linguistic Analysis Server* (LAS) is used for this purpose. This tool, created by the same authors, performs linguistic analysis on the levels of: morphological, syntactic and semantic analysis. The syntactic analysis builds a parse tree on the basis of Syntactic Group Grammar for Polish (SGGP) [22]. The system also performs the analysis of anaphoric relations. The second stage of summarization process is 'Selecting the essential sentences'. There is no concrete information on the criteria for sentence weighting. The last stage is called 'linearization'. It is designed to create coherent output. Proper forms of words are generated and placed in proper places in sentence. The system also performs homonyms reduction and anaphora substitution for better result reading.

The papers that describe the system do not provide any information about evaluation results.

3.2. Lakon (A. Dudczak)

Adam Dudczak's *Lakon* is another automatic text summarization system created for the Polish language [3]. It is available on-line at `http://www.cs.put.poznan.pl/dweiss/research/`

`lakon/`. The system was developed as a result of author's Master Thesis, whose one of main goals was to compare effectiveness of some popular extractive methods for the Polish language. Three methods were developed. They were based on the following heuristics:

- $tf \times idf$ and *Bm25 Okapi* — assumes that words occurrence frequency determines the sentence's salience
- sentence's position in text — assumes that most important sentences are often at the beginning of paragraphs,
- lexical chain — assumes that relations across sentences determine their salience.

The system was evaluated on the corpus created from 10 manually summarized newspaper articles. 60 volunteers manually created totally 285 summaries of these articles. Evaluation results indicated that the most effective features were words occurrence frequency and sentence's position. The lexical chains method was proved to be worse than the others.

3.3. Summarizer (J. Świetlicka)

Świetlicka's Summarizer [23] is the latest tool created for Polish. It is available on-line at `http://clip.ipipan.waw.pl/Summarizer`. This solution is the most similar to the one proposed here. It uses various machine learning methods for training an extractive summarizer based on a set of sentence's features. These features include:

- LLR — *Log Likelihood Ratio*,
- $tf \times idf$,
- Sentence's centrality,
- Occurrence of characteristics phrases — bonus and stigma words, popularity of one or two first words of a sentence,
- Similarity to the title — indicating occurrence of words from the title in a sentence,
- Number of words starting with uppercase — indicating Named Entities,
- Number of tokens that are not proper words — i.e. punctuation or numbers,
- Localization — position of the sentence in paragraphs and position of the sentence in the whole text,
- Length of sentence,
- Length of paragraph,

- Length of text,
- Type of sentence — based on the last token: declarative, interrogative, imperative.

A number of tests were performed on different subsets of these features. The author of the experiment used about 13 different machine learning algorithms in order to compare their effectiveness. The corpus was created by the author on her own and contains 102 newspaper articles for training and 67 articles for evaluation.

Świetlicka's Summarizer also performs simple summary linearization. It consists of three steps. Firstly, sentences are sorted in the order of their appearance in the document. Secondly, fragments in parentheses are removed in order to make sentences shorter. Lastly, some special words, such as therefore, moreover or however, are removed from the beginnings of sentences.

The discussed work contained the following conclusions:

- localization-based features, particularly sentence position in the paragraph and the whole document, tend to be the most important ones,
- sentence centrality feature is also very effective,
- cue words feature are not so effective,
- machine learning algorithms tend to be an effective solution for automatic summarization. Using a set of features result in better quality than using each feature separately.

CHAPTER 4

Polish Summaries Corpus

Polish Summaries Corpus is a resource created by Ogrodniczuk and Kopeć in 2014 [16]. Its aim is to provide a high quality corpus containing manual summarization examples. The corpus forms a significant facilitation for researchers, who can build their own summarization tools based on this corpus, as well as evaluate them. Ogrodniczuk and Kopeć notice that previous works on automatic summarization in the Polish language lacked a common corpus and a common evaluation method, therefore their results are not comparable.

Rzeczpospolita corpus — a collection of articles from the Web archive of a Polish newspaper [Weiss] was used as the base corpus for Polish Summaries Corpus. As stated by Ogroniczuk and Kopeć ([16]), this corpus consists of 190 379 pseudo-HTML files, dating from 1993 to 2002. Each file in the corpus contains different number of articles or even some non-textual content. The Rzeczpospolita corpus has been made available by Presspublica, who are the owners of the newspaper.

Polish Summaries Corpus contains 569 text documents divided into 7 categories: Society and Politics, Sport, Economy, Culture news, Law, National news and Science and Technology. All these texts have been manually summarized by independent annotators. All 569 documents have the extractive summaries and 154 have also the abstractive summaries. Size of each category is presented in Table 4.1. For each document in the corpus 5 independent propositions of summarization have been created. Each proposition of summarization contains 3 summaries of a given text of the approximate length of 5%, 10%, 15% of the original, respectively. The summaries are included in one another: 10% summary contains only fragments from previously selected 20% summary and so on. Therefore, the extractive corpus size is 8355 summaries.

Text domain	Abstractive summaries	Extractive summaries
Social and political	22	393
Sport	22	36
Economy	22	34
Cultural news	22	32
Law	22	26
National news	22	24
Science and technology	22	24
Total	**154**	**569**

Table 4.1. Selected domains sizes (source: [16])

CHAPTER 5

The proposed solution

The solution presented here implements sentence-based extractive summarization. It consists of two main components: linguistic analysis and summarization application. The latter component selects essential sentences and generates the result summary. The summarization component applies neural networks as a machine learning algorithm. Its conceptual framework is presented at Figure 1.2 in Chapter 1. Demonstration of the implemented system is available at `http://summar.pl/`.

5.1. Methodology

This section provides a detailed description of the tasks performed by each component of the system.

Linguistic analysis

The linguistic analysis component performs various kinds of text analysis. The input document is processed into the internal model and transferred to the summarization component. Most important natural language processing operations are performed by PSI-Toolkit [9], which is an open source system, integrating different NLP tools for the Polish language. These operations include dividing input document into paragraphs, sentences and tokens. Subsequently, lemmatization is performed and parts of speech are determined. Named entities are recognized using the Named Entity Recognition tool called NERf [25]. Lastly, short sentences, with no last punctuation mark are marked as headers. The conceptual visualisation of the linguistic analysis pipeline is presented at Figure 5.1

Summarization

The summarization component works as a three-stage process. These stage are:

1. Computation of feature values for each sentence in the document. This stage is desired to translate the input sentence into the corresponding vector of features values.
2. Sentence weighting based on the previously trained machine learning model and these computed features. As a result of this stage, each sentence in document has a weight, which indicates how worthy is this sentence to be included in the summary.

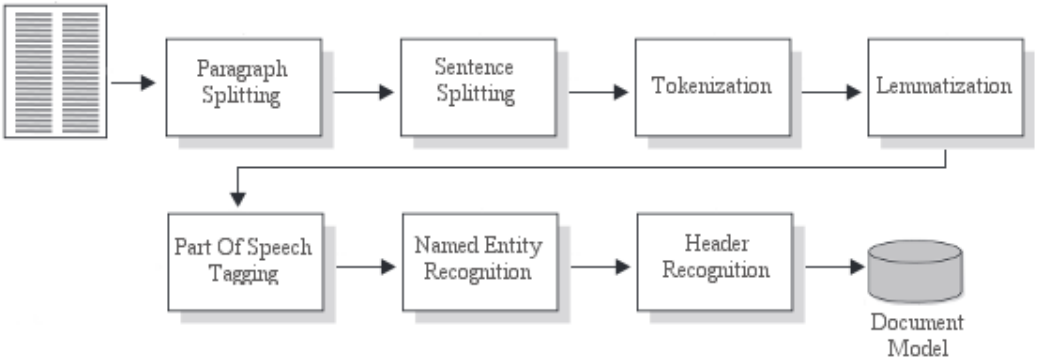

Figure 5.1. Linguistic analysis pipeline.

3. Summary preparation according to the obtained sentences weights. The summary contains of n sentences with the biggest weights, where n is the desired number of sentences to be included. This stage includes the sorting of result sentences, according to their order in the original document.

5.2. Description of features

This section describes each feature used in the system. Selection of the features was based on literature [Lloret, 14, 12, 3, 23] as well as a few new ideas. The complete list of used features includes:

TfIdf

It is a sum of **term frequency – inverse document frequency** ($tf \times idf$) value for every word in sentence. This measure is calculated for each word and is indicating the importance of this world in the whole document. The (5.1) formula is used to calculate $tf \times idf$ value.

$$tf \times idf = tf_{w,d} \cdot idf_w \tag{5.1}$$

In fact, it is a product of two measures, which are:

term frequency (tf) — assumes that a measure of word's importance is the number of its occurences in text. This measure in not sufficient on its own, because not all frequently occurring words are important. There are many common words in text, which are very frequent, such as prepositions or pronouns. To calculate tf value, the (5.2) formula is used.

$$tf_{w,d} = \frac{n_{w,d}}{\sum_k n_{k,d}} \tag{5.2}$$

In (5.3) $n_{w,d}$ is a number of occurrences of term t_w in document d_d and the denominator indicates the number of occurrences of all terms in the document d_d.

inverse document frequency (idf) — assumes that the most characteristic words for a given document are these which are the least frequently occurring. This measure is also not sufficient on its own, because rare words, like surnames, may turn out to be too precise to put them in summary. To calculate this measure, we use the (5.3) formula.

$$idf_w = \log \frac{N}{N_w} \tag{5.3}$$

N indicates the number of all considered documents and N_w is a number of documents containing the word w. In the case of summarization the sentence scale is used.

Centrality

Centrality is the arithmetic average of the sentence's similarity to all other sentences in the document. This measure assumes that the sentences that are the most similar to the others best reflect the content of the document. The (5.4) formula is used to calculate the centrality.

$$centrality(s) = \frac{1}{N-1} \sum_{x \neq s} sim(x, s) \tag{5.4}$$

N is a number of sentences in the document and *sim()* refers to cosine similarity. If a sentence is presented as a vector of $tf \times idf$ values of its subsequent words, the cosine similarity represents the similarity between two sentences as a cosine of this vectors. The (5.5) formula calculates the cosine similarity.

$$sim(s_1, s_2) = \frac{\sum_{w \in s_1, s_2} tf_{w,s_1} tf_{w,s_2} (idf_w)^2}{\sqrt{\sum_{w \in s_1} (tf_{w,s_1} idf_w)^2} \times \sqrt{\sum_{w \in s_2} tf_{w,s_2} idf_w)^2}} \tag{5.5}$$

When two sentences are identical, cosine similarity value is 1. On the other hand, if there is no common word in two sentences, its value is 0.

Location Features

Location features are based on sentence's location in the document. They assume that sentences with different salience may occur in different parts of the document. The location features implemented in the experiment:

SentLocPara — position of a sentence in the paragraph: in the first, second or third of equal parts,

ParaLocSection — position of the paragraph in the document: in the first, second or third of equal parts,

SentSpecialSection — occurrence in a special section such as the beginning (introduction) or ending (conclusion) of document.

Part Of Speech Features

This features are based on the Part of Speech tags of the sentence's words. These include:

Verb — existence of the final verb,

Nouns — number of nouns in sentence,

Pronouns — number of pronouns in sentence,

Named Entities Features

This features are based on the Named Entities recognized in the sentence. These include:

SentInHighestPname — number of Named Entities in the sentence as found by a naive method, recognizing Named Entity as a word starting with capital letter,

NER — number of Named Entities in the sentence as found by NERf Named Entities Recognition tool [25],

NERTf — sum of each Named Entity frequency in the whole document, occurring in given sentence,

PersNameNE — number of recognized NE of the "person" type,

OrgNameNE — number of recognized NE of the "organization" type,

PlaceNameNE — number of recognized NE of the "place" type,

DateNE — number of recognized NE of the "date" type,

GeogNameNE — number of recognized NE of the "geography" type,

TimeNE — number of recognized NE of the "time" type.

Other Features

The other features include:

SentInHighestTitle — number of words from heading or title in the sentence,

ParaLength — paragraph length: short (up to 1 sentence), average (2–5 sentences) or long (more than 5 sentences),

SentLength — sentence length: short (up to 7 words), average (7–14 words) or long (more than 14 words),

AvWordLength — the average of words lengths in sentences,

SentType — type of the sentences, based on its last punctuation mark: declarative, interrogative or imperative.

MetaInfo — sentences not referring to the document content, i.e.: an information about document's author or photo signatures.

It is worth noting that all the features are normalized. The features applied by the author of this thesis, which were not mentioned in the referred works, are:

- **MetaInfo**,
- **AvWordLength**,
- **Verb**,
- **Nouns**,
- **Pronouns**,
- **NER**,
- **NERTf**,
- **PersNameNE**,
- **OrgNameNE**,
- **PlaceNameNE**,
- **DateNE**,
- **GeogNameNE**,
- **TimeNE**.

CHAPTER 6

Evaluation

This chapter introduces methods aimed at evaluating automatic summarization systems. Firstly, main issues are discussed. Next, evaluation methods used in the developed system are described in detail. Lastly, the performed experiments and their results are presented.

6.1. Evaluation issues

Automatic summarization evaluation is not a trivial task. There exist a large number of evaluation methods and metrics. However, there are some well recognized challenges, which need to be considered by any evaluation method. These challenges include [13]:

- There are serious problem in determining the desired output in automatic summarization. An automatically generated summary may differ from any human summary used for evaluation and yet be considered as a good one.
- Human judgement may be required for some specific systems. That increases the expense of an evaluation experiment and makes it not easily repeatable.
- Since summarization involves different compression rates, which should be considered in the evaluation process. This makes the evaluation task much more complex.
- Usually, the summarization method used in an information system takes into account its desired application and user's needs. This should be considered in evaluation as well.

The most basic division of evaluation methods is the following:

Intrinsic

In the intrinsic evaluation methods, automatically generated summaries are compared to the manually created reference summaries. There is an assumption that the bigger is the overlap between these summaries, the better is automatically generated one.

Extrinsic

The extrinsic methods evaluate summaries based on a specific task. For example, human may be asked to answer a set of question about the content of the original documents on the basis of automatically generated, as well as human generated summaries. These answers are then compared with each other, as well as with the answers based on the original documents and then the quality of the system is determined.

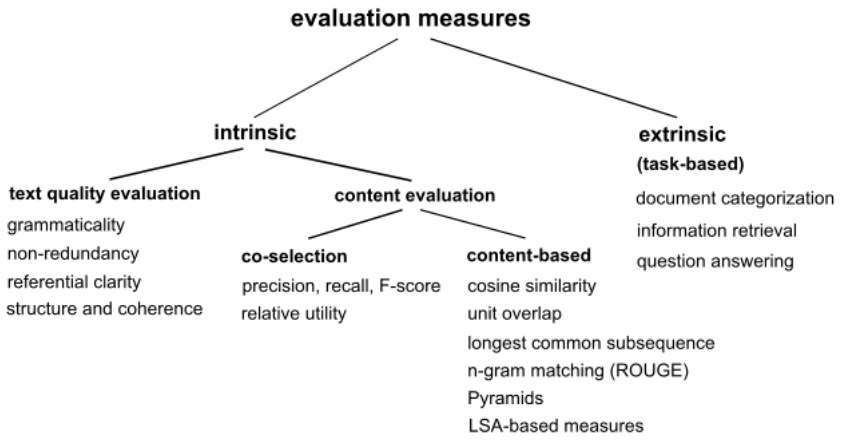

Figure 6.1. The taxonomy of summary evaluation measures (source: [21])

Figure 6.1 illustrates more specific summary evaluation taxonomy, as well as some concrete methods [21]. Intrinsic methods measuring **text quality** are often assessed by human annotators. In case of this thesis, the most important measures are grouped as the intrinsic **content evaluation**. These methods are often based on comparison with an ideal summary. They may be based on **co-selection** (compares ideal summary sentences with automatically selected ones) or **content-based** (compares the actual words in sentences). The extrinsic methods are usually **task-based**, which means that the used metrics refer to performance of using the generated summaries for a certain task.

6.2. Evaluation methods

Precision/Recall

The common information retrieval metrics based on **Precision and Recall** measures can be used to evaluate automatic summarization systems based on sentence-selection. Automatically selected sentences are evaluated against the manually created summaries, considered to be the gold standard. According to Nenkova and Keown ([15]):

Recall refers to the fraction of sentences chosen by a human that were also correctly identified by the system. The (6.1) formula is used to calculate recall.

$$Recall = \frac{|\text{system-human select overlap}|}{|\text{sentences selection by human}|} \tag{6.1}$$

Precision is the fraction of system sentences that were correct. The (6.2) formula is used to calculate precision.

$$Precision = \frac{|\text{system-human select overlap}|}{|\text{sentences selection by system}|} \tag{6.2}$$

Lastly, the **F-1 score** is defined as the harmonic mean of precision and recall as shown in the (6.3) formula.

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{6.3}$$

However, precision and recall measure are not the best choice for automatic summarization evaluation, due to several important issues [15]. There is a high degree of inconsistency in choosing sentences by different annotators for the same text. Using different human extracts to evaluate summarization system may result in significant changes in precision/recall results, not corresponding to the actual system's quality. Therefore, it seems that in case of automatic summarization evaluation recall might be more adequate than precision. As Nenkova and Keown state in ([15]): "Precision is overly strict — some of the sentences chosen by the system might be good, even if they have not been chosen by the gold standard creator. Recall, on the other hand, measures the overlap with already observed sentence choices". Another issue refers to semantic equivalence of different sentences in text, which is very common in news. If the system chooses one of equivalent sentences and human the other one, it is seen as a wrong answer. Due to this issues more sophisticated metrics need to be introduced.

ROUGE

ROUGE stands for Recall-Oriented Understudy for Gisting Evaluation [10]. It was introduced by Chin-Yew Lin at Document Understanding Conference (DUC) in 2004 and since then it has become the standard method for the evaluation of automatic summarization systems. It provides a set of measures to automatically determine the quality of summary in comparison to ideal summaries created by humans. The measure is based on overlapping units such as n-grams, word sequences and word pairs. ROUGE has been proved to be highly correlated with human judgements. This section describes ROUGE-N methods, which were proved to work well in single document summarization tasks.

$$\text{ROUGE-N} = \frac{\sum_{S \in \{ReferenceSummaries\}} \sum_{gram_n \in S} Count_{match}(gram_n)}{\sum_{S \in \{ReferenceSummaries\}} \sum_{gram_n \in S} Count(gram_n)} \tag{6.4}$$

ROUGE-N is an n-gram recall between a candidate summary and a set of reference summaries [10]. It is computed using the (6.4) formula, where n stands for the length of the n-gram $gram_n$, and $Count_{match}(gram_n)$ is the maximum number of n-grams co-occurring in a candidate summary and a set of reference summaries. It is worth noting that the denominator of (6.4) increases if more than one reference documents are used. Moreover, larger weight is assigned to matching n-grams occurring in multiple references, so if words are shared by more references, ROUGE-N favors them.

Kappa

Another important measure is the **Cohen's kappa** coefficient, which is widely used in the natural language processing annotation work [17]. In case of automatic summarization, it is used to measure agreement between annotators in a summary corpus. The kappa coefficient controls agreement P(A) by comparing the actual agreement compared to random agreement P(E) [17]. It is calculated using the (6.5) formula.

$$K = \frac{P(A) - P(E)}{1 - P(E)} \tag{6.5}$$

Kappa increases with the degree of agreement among annotators, taking $K = 1$, when the agreement is perfect. $K = 0$, when there is no agreement. In case of agreement lower than the one expected by random, Kappa can also be negative. Its worth noting that Cohen's kappa measures agreement between only two annotators.

Fleiss' kappa is a similar measure of agreement, used where there are more than two annotators. It allows even that different documents may be annotated by different annotators.

Let N be the number of sentences in the document, let n be the number of annotators, who annotated the document, and let k be the number of possible categories into which assignments are made (in this case there are two categories: "worthy" or "not worthy" of being included in summary). Then, let n_{ij} be the number of annotators who assigned the i-th sentence to the j-th category. To calculate Fleiss' kappa the following procedure is used [4]:

1. Calculate p_j, which is the proportion of all assignments which were to the j-th category. It is calculated using the (6.6) formula.

$$p_j = \frac{1}{Nn} \sum_{i=1}^{N} n_{ij} \tag{6.6}$$

2. Calculate P_i, which is the extent to which annotators agree for the i-th sentence. It is calculated using the (6.7) formula.

$$P_i = \frac{1}{n(n-1)} [(\sum_{j=1}^{k} n_{ij}^2) - n] \tag{6.7}$$

3. Compute $P(A)$, the mean of the P_i 's, using the (6.8) formula and compute $P(E)$, using the (6.9) formula.

$$P(A) = \frac{1}{N} \sum_{i=1}^{N} P_i \tag{6.8}$$

$$P(E) = \sum_{j=1}^{k} p_j^2 \tag{6.9}$$

4. Calculate the kappa coefficient using calculated $P(A)$ and $P(E)$ and the (6.5) formula.

In case of automatic summarization evaluation kappa coefficient may by be used as a metric. Let k be the number of manually created references. Measure of agreement among k annotators is considered as a baseline. During evaluation a random reference summary is replaced with automatically generated summary and the new value of kappa is calculated. This allows to determine the impact of the automatically generated summary on the measure of agreement.

6.3. Experiments and Results

A number of experiments were performed in order to evaluate results. PyBrain and FANN were used as neural networks libraries. Different subsets of features were used in order to achieve the best results in summarization. These subsets are presented in the Table 6.1. Therefore, every experiment was performed on a combination of Sub1 or Sub2 with one of Ner subsets.

Subset Name	**Features**
Sub1	{TfIdf, ParaLength, Centrality, SentType, SentSpecialSection, SentInHighestTitle, SentLength, SentLocPara, ParaLocSection}
Sub2	Sub1 ∪ {Pronouns, MetaInfo, Verb, Nouns, AvWordLength}
Ner1	{SentInHighestPname}
Ner2	{NER, NERTf}
Ner3	{OrgNameNe, GeogNameNe, DateNe, PlaceNameNe, PersNameNe, TimeNe}

Table 6.1. Subsets of features used in experiments.

Each learned model for each feature subset was evaluated with three different methods. These include:

- ROUGE (i.e. ROUGE-1, ROUGE-2, ROUGE-3),
- Precision and Recall,
- Kappa coefficient.

Two baselines were used as a useful reference point for comparison of results:

RANDOM — random summarization,

FIRST — first n sentences are considered as a summary, where n is desired summary length,

Additionally, **Kappa** coefficient was used as a human upper bound. Therefore, agreement between annotators in the test corpus is considered as an optimum, for which agreement between computer generated summary and human references should be as close as possible.

Parameters for each neural network library were selected individually in order to achieve the best results. Different combinations of parameters were tried. Evaluation results presented in this thesis are calculated for the best attempted combination. These are as follows:

PyBrain

Activation function: logistic function

Number of hidden layers: 1

Number of neurons per layer: 450

Number of epochs: 1

Learning rate: 0.0005

Momentum: 0.99

FANN

Activation function: logistic function

Number of hidden layers: 1

Number of neurons per layer: 20

Number of epochs: 100

Learning rate: 0.7

Momentum: 0.99

There are several significant differences between parameters combination for each library. Firstly, PyBrain was configured to perform only 1 training iteration, where FANN needed as much as 100 iterations to draw stable results. Secondly, the learning rate is much higher in case of FANN. Training with many iterations performed better with learning rate of 0.7. Lastly, using any more than 20 neurons in hidden layers for the FANN library did not resulted with significantly better results. In case of PyBrain, in order to achieve better performance up to 450 neurons were needed.

Evaluation results for ROUGE measure for the PyBrain library are shown in Table 6.2. These results can be summed up as follows:

1. Almost every subset of features used in experiments gave quite similar results in ROUGE, which were about 15% better than the RANDOM baseline and only a few percent better than the FIRST baseline, according to the F-1 score.

2. For the RANDOM baseline, ROUGE-2 and ROUGE-3 scores have decreased significantly, compared to ROUGE-1. For FIRST baseline and experiment's results this scores where more stable for each ROUGE-1, ROUGE-2 and ROUGE-3 scores.

3. No feature subset performed clearly better than the others.

4. Feature subsets, which used naive NER method (*Ner1*) more often reached the best result, than the subsets, which used more sophisticated NER methods (*Ner2* or *Ner3*).

5. Feature subsets without NER information included (*Sub1* and *Sub2*) did not fare much worse than feature subset with NER information included.

6. *Sub1* $\cup$ *Ner2* feature subset gave the worst result among all.

7. Using the bigger feature subset *Sub2* gave more stable evaluation results according to ROUGE-2 and ROUGE-3 compared to ROUGE-1.

Evaluation results for ROUGE measure for FANN library are shown in Table 6.3. These results can be summed up as follows:

1. The results achieved using the FANN library seem to be slightly higher than the ones achieved using PyBrain according to ROUGE measure.

2. Almost every subset of features used in experiments gave quite similar results in ROUGE.

3. Experiments using FANN library seem also to confirm that using naive NER method gives slightly better evaluation results.

4. Feature subsets without NER information included (*Sub1* and *Sub2*) still have very high evaluation results.

5. There is no subset, which performed much worse than the others according to the ROUGE measure.

6. The best results were achieved for the *Sub1* $\cup$ *Ner1*, as well as *Sub2* $\cup$ *Ner1* feature sets.

7. The results are nearly the same according to ROUGE-2 and ROUGE-3 measures, no matter of the subset used.

	ROUGE-1			ROUGE-2			ROUGE-3		
	R	P	F-1	R	P	F-1	R	P	F-1
RANDOM	0.29	0.34	0.30	0.14	0.17	0.15	0.13	0.15	0.13
FIRST	0.36	0.50	0.41	0.28	0.37	0.31	0.27	0.36	0.30
Sub1	0,46	**0,44**	0,44	0,35	0,32	0,33	0,34	**0,31**	0,32
Sub1 ∪ Ner1	**0,50**	0,43	**0,46**	**0,38**	**0,32**	**0,34**	**0,38**	**0,31**	**0,34**
Sub1 ∪ Ner2	0,47	0,39	0,42	0,33	0,26	0,29	0,32	0,25	0,28
Sub1 ∪ Ner3	0,49	0,43	0,45	0,37	0,31	**0,34**	0,37	0,30	0,33
Sub2	0,48	0,43	0,45	0,37	**0,32**	**0,34**	0,36	**0,31**	0,33
Sub2 ∪ Ner1	0,49	**0,44**	**0,46**	**0,38**	**0,32**	**0,34**	0,37	**0,31**	0,33
Sub2 ∪ Ner2	0,48	**0,44**	0,45	0,36	**0,32**	**0,34**	0,36	**0,31**	0,33
Sub2 ∪ Ner3	0,49	0,43	0,45	**0,38**	0,31	**0,34**	0,37	0,30	0,33

Table 6.2. ROUGE evaluation results for PyBrain. R = Recall, P = Precision

	ROUGE-1			ROUGE-2			ROUGE-3		
	R	P	F-1	R	P	F-1	R	P	F-1
RANDOM	0.29	0.34	0.30	0.14	0.17	0.15	0.13	0.15	0.13
FIRST	0.36	0.50	0.41	0.28	0.37	0.31	0.27	0.36	0.30
Sub1	0,51	0,43	**0,47**	0,40	0,32	0,35	0,39	0,31	0,34
Sub1 ∪ Ner1	**0,52**	0,43	**0,47**	**0,41**	0,32	0,35	**0,40**	0,31	**0,35**
Sub1 ∪ Ner2	0,51	0,43	0,46	0,39	0,32	0,35	0,39	0,31	0,34
Sub1 ∪ Ner3	0,51	0,43	0,46	0,39	0,32	0,35	0,38	0,31	0,34
Sub2	0,50	**0,44**	0,46	0,39	0,32	0,35	0,38	0,31	0,34
Sub2 ∪ Ner1	0,51	**0,44**	**0,47**	0,40	**0,33**	**0,36**	0,39	**0,32**	**0,35**
Sub2 ∪ Ner2	0,51	0,43	0,46	0,40	0,32	0,35	0,39	0,31	0,34
Sub2 ∪ Ner3	0,50	**0,44**	0,46	0,39	**0,33**	0,35	0,38	**0,32**	0,34

Table 6.3. ROUGE evaluation results for FANN. R = Recall, P = Precision

Evaluation results for Precision and Recall measure for PyBrain are presentend in Table 6.4. These results can be summed as follows:

1. Precision and Recall seem to be much less sensitive measure than ROUGE. All results are almost the same in this experiment.
2. *Sub1* ∪ *Ner2* feature subset still gave the worst result.
3. *Sub2* ∪ *Ner1* performed the best among all.

Evaluation results for Precision and Recall measure for the FANN library are shown in Table 6.5. These results can be summed up as follows:

1. The results achieved using the FANN library seem to be slightly higher than the ones achieved using PyBrain according to Precision and Recall measure.
2. Precision and Recall measure are almost the same for different subsets. Therefore, as in PyBrain experiments, this measure seems to be much less sensitive than ROUGE.
3. There is no feature subset, which performed significantly worse than the others.
4. Including NER information does not change the results.

	R	**P**	**F-1**
RANDOM	0.29	0.34	0.30
FIRST	0.36	0.50	0.41
Sub1	**0,29**	**0,27**	0,27
Sub1 ∪ Ner1	0,28	**0,27**	0,27
Sub1 ∪ Ner2	0,23	0,22	0,22
Sub1 ∪ Ner3	**0,29**	**0,27**	0,27
Sub2	0,28	**0,27**	0,27
Sub2 ∪ Ner1	**0,29**	**0,27**	**0,28**
Sub2 ∪ Ner2	**0,29**	**0,27**	0,27
Sub2 ∪ Ner3	0,28	**0,27**	0,27

Table 6.4. Precision and Recall evaluation results for PyBrain. R = Recall, P = Precision

Kappa evaluation results for PyBrain are shown in Table 6.6. These results can be summed as follows:

1. Kappa evaluation results are also nearly the same in this experiment.
2. All results are very close to the kappa upper bound.

	R	P	F-1
RANDOM	0.29	0.34	0.30
FIRST	0.36	0.50	0.41
Sub1	**0,31**	**0,29**	**0,29**
Sub1 ∪ Ner1	**0,31**	**0,29**	**0,29**
Sub1 ∪ Ner2	0,30	0,28	0,28
Sub1 ∪ Ner3	0,30	**0,29**	**0,29**
Sub2	0,30	0,29	**0,29**
Sub2 ∪ Ner1	**0,31**	**0,29**	**0,29**
Sub2 ∪ Ner2	0,30	0,28	**0,29**
Sub2 ∪ Ner3	0,30	0,28	**0,29**

Table 6.5. Precision and Recall evaluation results for FANN. R = Recall, P = Precision

3. Just as in previous experiments, *Sub1* ∪ *Ner2* feature subset gave the worst result among all.

4. The difference between best obtained results and the kappa upper bound is only 0.02. This means that agreement between human references and computer generated summaries is almost the same as the agreement between only human references.

Kappa evaluation results for the FANN library are shown in Table 6.7. These results can be summed up as follows:

1. The results obtained using the FANN library seem to be slightly higher than the ones achieved using PyBrain according to kappa measure.

2. The results obtained using the Fann Library are also quite closer to the kappa upper bound.

3. There is no subset, which performed significantly worse than the others.

4. The difference between these results and the kappa baseline is only 0.02 in as much as 4 cases.

5. *Sub2* ∪ *Ner1* feature set performed the best, achieving the kappa value of 0,27.

	Kappa
Human	0.28
Sub1	0,25
Sub1 ∪ Ner1	0,25
Sub1 ∪ Ner2	0,23
Sub1 ∪ Ner3	**0,26**
Sub2	0,25
Sub2 ∪ Ner1	0,25
Sub2 ∪ Ner2	**0,26**
Sub2 ∪ Ner3	0,25

Table 6.6. Kappa evaluation results for PyBrain.

	Kappa
Human	0.28
Sub1	0,25
Sub1 ∪ Ner1	0,26
Sub1 ∪ Ner2	0,25
Sub1 ∪ Ner3	0,26
Sub2	0,26
Sub2 ∪ Ner1	**0,27**
Sub2 ∪ Ner2	0,26
Sub2 ∪ Ner3	0,25

Table 6.7. Kappa evaluation results for FANN.

CHAPTER 7

Summary

The experiments carried out in the area of automatic summarization for the Polish language preformed in this thesis are summarized in this chapter. Conclusions from the experiments are presented and possible paths of development are be pointed out.

7.1. Conclusions from the experiments

In this thesis, a document summarizing approach for the Polish language has been presented. It is based on sentence extraction and applies neural networks as a machine learning algorithm. This approach seems to be promising in achieving an acceptable summarizing method for the Polish language, however there are some difficulties in choosing the proper features set and tuning machine learning algorithm. Several conclusions may be drawn from this experiment, which may be helpful in future research. These conclusions include:

1. The basic set of features, which includes: $tf \times idf$, centrality, location features, paragraph and sentence length, sentence type and title keywords, seems to be sufficient in achieving satisfactory results. However, using more features, like Part Of Speech tags, resulted in better results while examining longer n-grams in ROUGE. That suggests it is worth using larger feature sets.

2. Using as a feature a number of Named Entities in the sentence as found by a naive method (which recognizes Named Entity as a word starting with capital letter), seems to be sufficient in machine learning task. Using sophisticated NER methods seems pointless.

3. The Kappa coefficient evaluation method used in experiments gives very valuable information about the automatic summarization system. It turns out that a set of human summaries containing one computer generated summary seems to have almost the same agreement measure as a set of summaries, which are entirely made by human. The other methods have evaluated the computer generated summary compared to the gold standard, which is some ideal summary concluded from all references.

4. Precision and Recall measures do not seem to be very good evaluation methods in case of automatic summarization, because they do not differentiate results of experiments.

7.2. Possible paths of development

There is still much work to do in the field. The list of interesting considerations for the future research at the task of automatic summarization for Polish is listed below.

1. There is still need to develop new features, which would improve machine learning. One of possible path is to develop simpler features, for example binary features, which would contain only one information (yes/no) or features containing clear, not processed information, like index of the sentence in document. Such features may be more useful for training algorithm.

2. Other machine learning algorithms should be examined as well. Especially, some simpler algorithms, like linear regression should be used. Using linear regression instead of neural networks may be also useful for determining the impact of single features on the learning process. In case of neural networks determining the impact of single features is not possible, because of the algorithm's complicated structure.

3. Dividing the corpus may give some better evaluation results. One possible direction is to choose documents form corpus, which have the biggest agreement measure and use them as the dataset for training. Another idea is to perform different trainings for different categories of documents.

4. The presented attempt may serve as the baseline for future solutions, as it is the first summarization project evaluated against the Polish Summaries Corpus, the standardized corpus of summaries for the Polish language.

Bibliography

[1] Bishop, C. M. (1995). *Neural Networks for Pattern Recognition*. Oxford University Press, Inc., New York, NY, USA. 20

[2] Ciura, M., Grund, D., Kulików, S., i Suszczanska, N. (2004). A system to adapt techniques of text summarizing to polish. In Okatan, A., editor, *International Conference on Computational Intelligence, ICCI 2004, December 17-19, 2004, Istanbul, Turkey, Proceedings*, pages 117–120. International Computational Intelligence Society. 31

[3] Dudczak, A., Stefanowski, J., i Weiss, D. (2008). Automatyczna selekcja zdań dla tekstów prasowych w języku polskim. Technical Report RA-03/08, Institute of Computing Science, Poznan University of Technology, Poland. 31, 37

[4] Fleiss, J. i inni (1971). Measuring nominal scale agreement among many raters. *Psychological Bulletin*, 76(5):378–382. 44

[5] Gershenson, C. (2003). Artificial neural networks for beginners. *CoRR*, cs.NE/0308031. 21, 22

[6] Hahn, U. i Mani, I. (2000). The challenges of automatic summarization. *Computer*, 33(11):29–36. 13

[7] Heaton, J. (2008). *Introduction to Neural Networks for Java, 2Nd Edition*. Heaton Research, Inc., 2nd edition. 22, 23, 57

[8] Kriesel, D. (2007). *A Brief Introduction to Neural Networks*. 17, 18, 20, 21

[9] Krzysztof, J. (2013). Psi-toolkit - an open architecture set of nlp tools. In Zygmunt, V., editor, *Proceedings of the 6th Language and Technology Conference*. 36

[10] Lin, C.-Y. (2004). Rouge: A package for automatic evaluation of summaries. In *Proc. ACL workshop on Text Summarization Branches Out*, page 10. 7, 43

[Lloret] Lloret, E. Text summarization: An overview. [on-line] `http://www.dlsi.ua.es/~elloret/publications/TextSummarization.pdf`. 8, 9, 10, 37

[12] Mani, I. (2001a). *Automatic Summarization*. Natural language processing. J. Benjamins Publishing Company. 8, 9, 10, 11, 12, 13, 14, 37, 58

[13] Mani, I. (2001b). Summarization evaluation: An overview. 41

[14] Mani, I. i Maybury, M. (1999). *Advances in Automatic Text Summarization*. MIT Press. 12, 37

[15] Nenkova, A. i McKeown, K. (2011). Automatic summarization. *Foundations and Trends in Information Retrieval*, 5(2-3):103–233. 42, 43

[16] Ogrodniczuk, M. i Kopeć, M. (2014). The polish summaries corpus. In Chair), N. C. C., Choukri, K., Declerck, T., Loftsson, H., Maegaard, B., Mariani, J., Moreno, A., Odijk, J., i Piperidis, S., editors, *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, Reykjavik, Iceland. European Language Resources Association (ELRA). 7, 34, 35, 57

[17] Radev, D. R., Teufel, S., Saggion, H., Lam, W., Blitzer, J., Qi, H., Çelebi, A., Liu, D., i Drábek, E. (2003). Evaluation challenges in large-scale document summarization. In Hinrichs, E. W. i Roth, D., editors, *ACL*, pages 375–382. ACL. 44

[18] Rojas, R. (1996). *Neural Networks: A Systematic Introduction*. Springer-Verlag New York, Inc., New York, NY, USA. 17, 22

[19] Russell, S. J. i Norvig, P. (2003). *Artificial Intelligence: A Modern Approach*. Pearson Education, 2 edition. 16, 17, 58

[20] Schaul, T., Bayer, J., Wierstra, D., Sun, Y., Felder, M., Sehnke, F., Rückstieß, T., i Schmidhuber, J. (2010). PyBrain. *Journal of Machine Learning Research*. 23

[21] Steinberger, J. i Jezek, K. (2009). Evaluation measures for text summarization. *Computing and Informatics*, 28(2):251–275. 42, 58

[22] Suszczańska, N. i Lubiński, M. (2001). Polmorph, polish language morphological analysis tool. In *19th IASTED International Conference APPLIED INFORMATICS - AI'2001, Innsbruck (Austria)*, pages 84–89. 31

[23] Świetlicka, J. (2010). Metody maszynowego uczenia w automatycznym streszczaniu tekstów. praca magisterska, University of Warsaw. 32, 37

[24] V., G. i G.S., L. (2010). A survey of text summarization extractive techniques. *Journal of Emerging Technologies in Web Intelligence*, 2(3):258–268. 13

[25] Waszczuk, J., Głowińska, K., Savary, A., i Przepiórkowski, A. (2010). Tools and methodologies for annotating syntax and named entities in the National Corpus of Polish. In *Proceedings of the International Multiconference on Computer Science and Information Technology (IMCSIT 2010): Computational Linguistics – Applications (CLA'10)*, pages 531–539, Wisła, Poland. PTI. 36, 39

[Weiss] Weiss, D. Korpus rzeczpospolitej. [on-line] http://www.cs.put.poznan.pl/dweiss/rzeczpospolita. 34

List of Tables

List of Figures