CZECH TECHNICAL UNIVERSITY IN PRAGUE

F3

Faculty of Electrical Engineering
Department of Cybernetics

Bachelor's Thesis

Evaluation of Readers' Reactions to the Content of Media News

Jakub Ambroz
Study program: Open Informatics
Specialisation Artificial Intelligence and Computer Science
ambrojak@fel.cvut.cz

May 26, 2023
Supervisor: Ing. Radek Mařík, CSc.

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Ambroz Jakub** Personal ID number: **499162**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Open Informatics**

Specialisation: **Artificial Intelligence and Computer Science**

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Evaluation of Readers’ Reactions to the Content of Media News

Bachelor’s thesis title in Czech:

Hodnocení reakcí čtenářů na obsah mediálních zpráv

Guidelines:

1) Make an review of methods used to search for phrasesthat determine the content of media reports.
2) Research methods of evaluating the sentiment of a text.
3) Research methods of visualizing results related to natural language processing.
4) Choose an appropriate set of methods and implement the corresponding processing chain.
5) Conduct experiments with both Czech and English texts. Focus on visualization of the results.
6) Discuss the results and identify critical processing points.

Bibliography / sources:

[1] Mitchell, Ryan. 2015. Web Scraping with Python. “O’Reilly Media, Inc.”
[2] Hapke, Hannes, Cole Howard, and Hobson Lane. 2019. Natural Language Processing in Action. Simon and Schuster.
[3] Bird, Steven, Ewan Klein, and Edward Loper. 2009. Natural Language Processing with Python. “O’Reilly Media, Inc.”
[4] Kubat, Miroslav. 2018. Introduction to Machine Learning. S.L.: Springer International Pu.

Name and workplace of bachelor’s thesis supervisor:

Ing. Radek Mařík, CSc. Department of Telecommunications Engineering FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **21.12.2022** Deadline for bachelor thesis submission: **26.05.2023**

Assignment valid until: **22.09.2024**

Ing. Radek Mařík, CSc. Supervisor’s signature	prof. Ing. Tomáš Svoboda, Ph.D. Head of department’s signature	prof. Mgr. Petr Páta, Ph.D. Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor’s thesis, the author must state the names of consultants and include a list of references.

. Date of assignment receipt	Student’s signature

Prohlášení autora práce / Declaration

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, May 26, 2023

....
signed Jakub Ambroz

Abstrakt / Abstract

Tato bakalářská práce poskytuje základní přehled o extrakci dat z webu (web scraping), zpracování přirozeného jazyka (NLP) a analýze sentimentu - se zaměřením na lexikony sentimentu. Byl vytvořen soubor dat s články a jejich komentářovými sekcemi ze stránek `www.seznamzpravy.cz` a `www.idnes.cz`. Z datasetu jsme vytvořili vektorové reprezentace pomocí programu Word2vec. Úpravou přístupu založeného na slovníku (dictionary-based approach) jsme zkoumali, zda by tyto vektory mohou být vhodné pro vytváření lexikonů sentimentu.

Existující lexikon sentimentu byl použit k analýze sentimentu komentářů k článkům zaměřeným na různá témata. Nakonec jsme zkoumali, zda by reakce na tyto komentáře (lajky, dislajky) mohly být použity k měření sentimentu.

Klíčová slova: Python, Zpracování přirozeného jazyka, Analýza sentimentu, Extrakce dat z webu, Analýza novinových zpráv, Word2vec, Lexikon sentimentu, Učení bez učitele, Neuronové sítě

Překlad titulu: Hodnocení reakcí čtenářů na obsah mediálních zpráv

This bachelor's thesis gives a basic overview of web scraping, natural language processing, and sentiment analysis - focusing on sentiment lexicons. A dataset with articles and their comment sections from `www.seznamzpravy.cz` and `www.idnes.cz` was created. From the dataset, we created vector representations by Word2vec. By adjusting the dictionary-based approach, we explored if these vectors could be appropriate for creating sentiment lexicons.

An existing sentiment lexicon was used to analyze the sentiment of the comments on articles focusing on different topics. Finally, we explored if reactions to these comments (likes, dislikes) could be used to measure sentiment.

Keywords: Python, Natural Language Processing, Sentiment Analysis, Web Scraping, News Analysis, Word2vec, Sentiment Lexicons, Unsupervised Learning, Neural Networks

Contents /

Tables / Figures

Chapter 1
Introduction

Before we could perform any analysis, we first had to get the data. In this case, that will be news and readers' reactions to them. We decided to download articles from online news sites with comment sections. Chapter 2 introduces some technical and theoretical background necessary for understanding web scraping. However, most of the relevant information is in 6.2 under implementation. Because in this field, the theory is simple but practical implementation can be challenging.

Then the data is scraped and extracted from the HTML source code. The following text is processed. For this, there is a field of NLP (Natural Language Processing). Chapter 3 describes how to parse natural text into tokens and how to preprocess the text. It then explains how to represent text with vectors.

We look at how Neural Networks can be used in NLP in Chapter 4. We focus on how Neural Networks could help us represent words - Word2vec. Particular focus is given to Sentiment Analysis (Chapter 5), a subfield of NLP. Sentiment Analysis deals with identifying or extracting natural text's sentiment (emotion). We explain what sentiment lexicons are and how to create them.

Chapter 6 describes the tools used and why they were chosen. It also describes the dataset and how it was created. This dataset is further used in Chapter 7 to analyze the sentiment of the comments. The dataset is also used to create Word2vec vector representations of words. Furthermore, the possibility of using this representation for creating sentiment lexicons is explored.

Chapter 2
Web Scraping

Web scraping - also known as screen scraping, data mining, or web harvesting - is a term referring to downloading and extracting useful data from the web or the internet. It is also typically used when using a script or program to do the work instead of the human. Because doing the work manually is too tedious and time-consuming. This may be caused by the structure or size of the data that one is trying to access. If there are better options for getting to data desired, for example, public APIs or public datasets or archives, it may be simpler and faster to use those [Mitchell, 2018, Preface; Zhao, 2017].

2.1 Scraping Static Websites

Every site is some kind of HTML document. HTML stands for Hyper Text Markup Language. Hypertext refers to the ability to link other text that can be accessed, e.g., `www.example.com`. Markup means that the text uses some syntax and tags to note how (color, font, size, etc.) each part should be displayed e.g.,

```
<tag attribute1="value1" attribute2="value2">''text to scrape''</tag>.
```

One can get this HTML document by HTTP GET request at a certain URL (e.g., `www.domain_name.xyz/article/abc`). For this, one can use urllib2 or selenium libraries in Python. From this HTML page, to extract the information one wants BeautifulSoup4 is used. BS4 is a Python library specifically for that. But once can use any tool that has the capacity to read and edit text [HTML - Living Standard , Zhao, 2017].

2.2 Scraping Dynamic Websites

DHTML (Dynamic HTML) pages change how they appear or their content with animations or after some interaction with a user. Many websites today use dynamic loading using JS (Javascript). That means that a single GET request is not sufficient for getting the data. Because they are downloaded afterward (for example, when a user scrolls) using some script and are added to the now modified HTML. For scraping such websites, Selenium is used. Selenium WebDriver can be used with Python (or other languages such as Java, JS) and a browser of your choice - Firefox, Edge, Chrome, Safari, [1] [Mitchell, 2018 pg. 108]

[1] `https://www.selenium.dev/documentation/webdriver/getting_started/install_drivers/`

Chapter 3
Natural Language Processing

Natural Language Processing, commonly shortened to NLP, deals with natural languages (English, Spanish, Czech, etc.) and how they can be processed by computers. Natural languages are harder compared to synthetic languages. For example, programming languages were constructed and designed with rules that are always followed, and the syntax is easily parsed. Natural languages usually have rules, but they are more complex and have more exceptions. Another issue is high context dependence and uncertainty, e.g., homonyms. And finally differences between written text and speech or between formal and informal language [Bird, 2009].

3.1 Text Tokenization

Text tokenization is a process of splitting text into tokens carrying meaning. The most straight forward way would be to split the text into words. This can be achieved by splitting the text by white space. But most tokenizer take into account other parts of setneces - dots, comas, exclamation marks, etc. The tokens can be even smaller. Some tokenizer split words into roots, prefixes and suffixes. This approach is, hwoever, very different in each language. [Hobson, 2019].

3.1.1 Text Normalization

Text normalization refers to a multitude of methods that augment the text before the process of extracting information from it. It may be as simple as *case folding* - making all letters the same case. There will be some loss of information (Bush vs bush). A more complicated approach would be capitalizing only letters at the beginning of the sentence [Manning, 2008].

Removing accents seems like a good idea in English (cliché and cliche, or naive and naïve). In languages like Czech, it may change the meaning ("nos", "noš" meaning "nouse" and "carry!") and clump multiple different words into one. However, there are still people who type in ASCII only. This may be due to habit from earlier more limited computers (where diacritics were not always avaiable or displayed correctly), avoiding potential issues that come with different encodings, or because the keyboard layout without diacritics is better for certain computer tasks (e.g., programming). Removing all diacritics and accents may improve performance due to this. The words with the same spelling may have different meanings even before removing diacritics. So removing diacritics does not introduce a new issue, but it can make it worse. [Manning, 2008].

3.1.2 Stemming

Stemming is a process of removing small parts of a word (for example, endings) and reducing the word to its root. This seems like a good idea in English, where there is an ending '-s' for verbs in the third person singular in the present tense, '-s' for plural nouns, '-en', '-ing' and '-ed' in verb forms. This is a gross oversimplification. For example, Porter's stemmer has much more rules than that.[Hobson, 2019, pg. 58]

pád	mužský rod životný	mužský rod neživotný	ženský rod	střední rod
1.	mladý (malý)	mladý (malý)	mladá (malá)	mladé (malé)
2.	mladého (malého)	mladého (malého)	mladé (malé)	mladého (malého)
3.	mladému(malému)	mladému(malému)	mladé (mladé)	mladému (malému)
4.	mladého (malého)	mladý (malý)	mladou (malou)	mladé (malé)
5.	mladý (malý)	mladý (malý)	mladá (malá)	mladé (malé)
6.	mladém (malém)	mladém (malém)	mladé (malé)	mladém (malém)
7.	mladým (malým)	mladým (malým)	mladou (malou)	mladým (malým)

Figure 3.1. Czech Adjectives Singular, Hard Declension [website mojecestina.cz]

pád	mužský rod životný	mužský rod neživotný	ženský rod	střední rod
1.	mladí (malí)	mladé (malé)	mladé (malé)	mladá (malá)
2.	mladých (malých)	mladých (malých)	mladých (malých)	mladých (malých)
3.	mladým (malým)	mladým (malým)	mladým (malým)	mladým (malým)
4.	mladé (malé)	mladé (malé)	mladé (malé)	mladá (malá)
5.	mladí (malí)	mladé (malé)	mladé (malé)	mladá (malá)
6.	mladých (malých)	mladých (malých)	mladých (malých)	mladých (malých)
7.	mladými (malými)	mladými (malými)	mladými (malými)	mladými (malými)

Figure 3.2. Czech Adjectives Plural, Hard Declension, [website mojecestina.cz]

pád	mužský rod životný	mužský rod neživotný	ženský rod	střední rod
1.	jarní (cizí)	jarní (cizí)	jarní (cizí)	jarní (cizí)
2.	jarního (cizího)	jarního (cizího)	jarní (cizí)	jarního (cizího)
3.	jarnímu (cizímu)	jarnímu (cizímu)	jarní (cizí)	jarnímu (cizímu)
4.	jarního (cizího)	jarní (cizí)	jarní (cizí)	jarní (cizí)
5.	jarní (cizí)	jarní (cizí)	jarní (cizí)	jarní (cizí)
6.	jarním (cizím)	jarním (cizím)	jarní (cizí)	jarním (cizím)
7.	jarním (cizím)	jarním (cizím)	jarní (cizí)	jarním (cizím)

Figure 3.3. Czech Adjectives Singular, Weak Declension [website mojecestina.cz]

pád	mužský rod životný	mužský rod neživotný	ženský rod	střední rod
1.	jarní (cizí)			
2.	jarních (cizích)			
3.	jarním (cizím)			
4.	jarní (cizí)			
5.	jarní (cizí)			
6.	jarních (cizích)			
7.	jarními (cizími)			

Figure 3.4. Czech Adjectives Plural, Weak Declension [website mojecestina.cz]

For comparison, these are rules for Czech adjectives[1]. This is only adjectives. The tables for nouns or verbs would look similar. Needless to say, developing own stemmer would be very time-consuming. [Hobson, 2019, pg.59] If there is a need or want to use a stemmer for Czech, use an existing one, e.g., [Zápotocký, 2012].

3.1.3 Lemmatization

Lemmatization is a special type of normalization that uses not only how the word is spelled but also it's meaning. The downside is that it requires information about the meaning of the words. It has to recognize the similarity in meaning in words with completely different spelling (synonyms) and separate words similar in spelling and different in meaning [Hobson, 2019, pg. 60].

Both lemmatization and stemming reduce the vocabulary and increase the ambiguity of the text. They may be useful for certain use cases with a limited amount of data where they perform better on information retainment in some cases. But with a sufficient amount of data, this is not the case. Some authors ([Manning, 2008]) even ignore these entirely because the improvements are negligible.

[1] https://www.mojecestina.cz/article/2009092802-sklonovani-pridavnych-jmen

3.2 Bag of Words

The term *bag of words* refers to representing a sentence or text as a collection of words that are order independent. For each text, we create a vector. This vector is the size of the number of all words in our entire corpus. And each vector has on position i: 1 if the i-th word is present in the text or 0 if the word is not present. This representation is very crude because it doesn't take into account the context or the order of the words. But thanks to this simple technique, we have a vector representation of text. A huge mathematical apparatus was already developed for vectors. And now we can use it with words. These vectors can also serve as an input for Neural Networks or other machine learning models [Hobson, 2019].

3.3 TF-IDF Vectors

Term Frequency - Inverse Document Frequency is a single number where the Term Frequency is divided by Document Frequency. TF-IDF is calculated for a term (word or a few words together) and a document. Term frequency is the number of times the term occurred in a given document, and Document Frequency is the number of documents the term occurred in. Both of these numbers are typically logarithmed. The logarithmization is used because of word frequencies in natural text. What one will see is that the most common word is two times as likely to occur in a text than the second most common. Moreover, four times more likely than the fourth most common. This idea proved useful in accuracy. A nice side effect is that we get rid of the potential problem of numerical stability that may arise in a large corpus [Hobson, 2019].

The logic behind TF and DF is that TF is a really rough but simple measurement of how important the term is to the document. However the most common words (pronouns, prepositions, verb be, etc.) would be important for all documents (due to the distribution outlined above). Dividing by the number of documents the term occurs in allows us to lower the weight of the most common words significantly and increases the importance of words that are relatively rare in common text and are thus specific to the respective document and its topic [Hobson, 2019].

The next step is doing the process described above for every word in the dictionary - that is, for every word present in our corpus. The result is TF-IDF vector that has its length equal to the size of the dictionary. This vector could be considered a simple representation of the meaning or topic of the document. Documents with similar TF-IDF vectors (that is, vectors that are near each other in vector space) probably have similar topics. But they can theoretically be about very different things if the text use homonyms (words spelled the same or similar way but have different meanings) heavily. Synonyms (words with the same meaning but different spelling) are another problem, because it could result in vectors far apart for documents with close topics. These can be partially addressed with text normalization methods described above, such as stemming and lemmatization [Hobson, 2019].

3.4 Semantic Analysis

Semantic Analysis is a subfield dealing with the meaning of the text. TF-IDF vectors are not good enough. But there are methods for transforming TF-IDF vectors into so-called *topic vectors* or for making them directly from the corpus.

3.4.1 Latent Semantic Analysis

LSA (Latent Semantic Analysis) is based on a well-known technique for reducing dimensionality from linear algebra, SVD (Singular Value Decomposition). This technique decomposes a TF-IDF matrix (made out of TD-IDF vector for each document) into 3 simpler matrices. We get the original TF-IDF matrix by multiplying them. This technique has many applications not only in computer science but engineering in general. Therefore there are many implementations with efficient algorithms.

$$M_{TF-IDF} = USV^T$$

From the point of view of semantic analysis, the most important matrix is U. In the full (not reduced) form, it has its width and height equal to the number of words in the dictionary. Individual numbers express correlation between the respective words.
S is a diagonal matrix called singular. In the case of semantic analysis, these numbers express how much information is captured in each dimension in the new vector space of the topic. This can be used in dimensionality reduction, so we drop only the dimensions with the least information. Therefore we replace the smallest values in S with zero. Reducing as many dimensions as one likes while keeping the information loss as small as possible.
V^T is transposed matrix. In this context, it measures the similarity of documents. It can be used as a control if we have some labels of the documents [Hobson, 2019].

3.4.2 Linear Discriminant Analysis

Linear Discriminant Analysis (LDA) is a simple method that splits the data into two categories, e.g., spam and ham. LDA is a supervised learning method. Therefore it is necessary to have already labeled data. The first step is calculating the centroids of both categories. That means the 'average' of data with one of the labels. Data here are the TF-IDF vectors. In the classification phase, the TF-IDF vector of the new document is calculated. Then it is decided to which of the centroids it is closer. The new document is then classified as the label of the closest centroid [Hobson, 2019].

3.4.3 Latent Dirichlet Allocation

Latent Dirichlet Allocation (shortened LDA or LDiA to avoid confusion with Linear Discriminant Analysis) is an alternative to LSA that performs slightly better in some situations. Unlike LSA which uses simple linear algebra, LDiA uses the assumption of Dirichlet distribution of words that is closer to reality. Another assumption is that each document is a linear combination of a certain number of topics. This number is a hyperparameter that can be optimized in training [Hobson, 2019].

Chapter 4
Neural Networks and their Applications in Natural Language Processing

The possibilities of using Neural Networks (NN) are large and still developing area in machine learning. What follows is an introduction to the basic mechanism behind them. This can be used in understanding what types of Neural Networks are useful for NLP.
Artificial Neural Networks are inspired by the structure of biological neural networks. These are made out of neurons that are connected to other neurons by synapses. In biology, the structure of these connections is genuinely complicated. The artificial kind uses more simple structures that can be more easily computed and represented mathematically [Kriesel, 2007].

4.1 Perceptron

Perceptron is one of the most basic types of artificial neurons. It has n inputs. Let the i-th one be x_i. Next, we use a vector notation for all the inputs $\vec{x}$. Each input will have some weight w_i. And the last thing is *bias*. It can be thought of as a weight that influences the neuron regardless of the values of the input. When we combine it into one equation, it looks like $f(\vec{x})$

$$f(\vec{x}) = \vec{x}\vec{w} + b = \sum_{i=0}^{n} x_i w_i + b$$

The output of this operation is a number that would serve as an input to the activation. The neuron does not activate if the activation is not high enough. That means the number must be higher than a certain threshold t in order to activate the neuron. If it is higher, than the output of the neuron will be one; otherwise, it will be zero. The activation function $g(x)$:

$$g(x) = \begin{cases} 1 & \text{for } x \geq p, \\ 0 & \text{for } x < p \end{cases}$$

The output of the neuron is $g(f(\vec{x}))$. There are many alternatives to threshold function, described above, that can be used as activation functions: the sigmoid function, hyperbolic tangent, and ReLU (Rectified Linear Unit). Some can improve the performance of Neural Network.

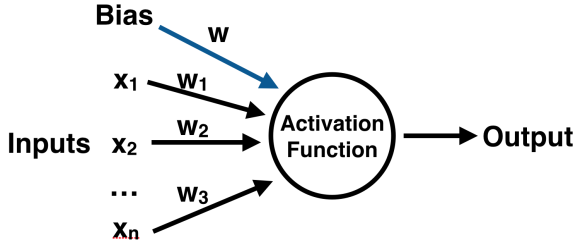

Figure 4.1. Perceptron, [Portilla, 2017]

4.2 Feedforward Neural Networks

Feedforward is one of the most straightforward architectures NN use. It is composed of several layers. The first one is called the *input layer*, next there are several *hidden layers* followed by an *output layer*. The layers are connected in one direction only. There are no shortcuts to further layers or loops back into a previous one. They are called *fully connected* if there are connections for each neuron in one layer into all neurons in the proceeding layer [Kriesel, 2007].

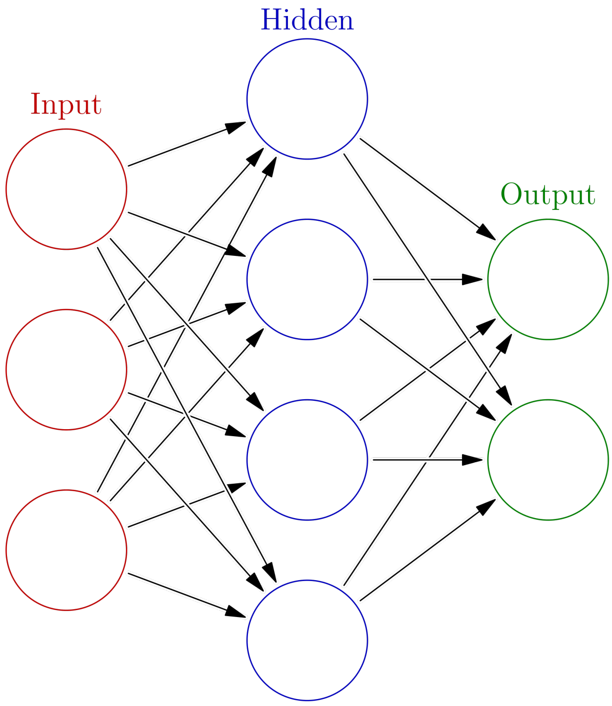

Figure 4.2. Simple Feedforward Neural Network, [Portilla, 2017]

The learning uses a process called *backpropagation*. It uses differentiation and chain rule. It backpropagates the differential from output back to input. Afterward the weights are adjusted accordingly [Kriesel, 2007].

4.2.1 Word2vec

Word vector refers to any vector that can be used to represent words. The simplest is called *one-hot encoding*. It is an empty vector filled with zeros and only a single one. This vector has the length of the vocabulary, and the one is in the position respective to the position of the word in the dictionary.

Word2vec is an unsupervised learning model which tries to get better *word vectors*. This should be a vector representation of a word that captures its meaning. And idealy it also reduces the dimension. Thus the process should turn the one-hot vector into a *dense vector* - a vector filled with float numbers.

Because it is unsupervised learning, we need just a lot of data, but the data do not have to be labeled. However, the creation of this model is very computationally demanding. Luckily there are publicly available models from Google or Facebook. They are pre-trained on gigantic amounts of data.[1]

[1] https://github.com/facebookresearch/fastText

The basic model (later changed and improved) consists of input, output, and one hidden layer. There are 2 approaches to what are input and output data. In the so-called *skip-gram* approach, the input is one word, and the neural network tries to predict its surroundings. First step is the training phase. When it finishes, we can get the desired *word vector* the weights of the hidden layer. The input is the word to which the word vector is wanted. The word vector is, therefore, the same size as the hidden layer. So it can theoretically be as large or small as the application requires because this number is chosen y humans before starting the training process. Alternatively, it can be looked at as a hyperparameter to optimize.

The alternative approach is the *continuous bag of words* in which the model tries to predict the missing word from words in their surroundings [Hobson, 2019].

4.3 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are focused on images. However, they can also be used in NLP. There are three types of layers in this architecture: convolutional, pooling, and fully connected layer. Fully connected is a simple layer where each neuron is connected to neurons in adjacent layers. Pooling layers are used to reduce dimensionality, and therefore proceeding layers have fewer parameters to optimize. For example, max-pooling selects and outputs the largest element. This is not done on the entire input but only on some small window called *kernel*. For example, a 2 by 2 pixels kernel slides across the image, 2 pixels at each step. The number of pixels it moves in each step is called *stride*. This scales down the image's dimension to half of the original size [O'Shea, 2015].

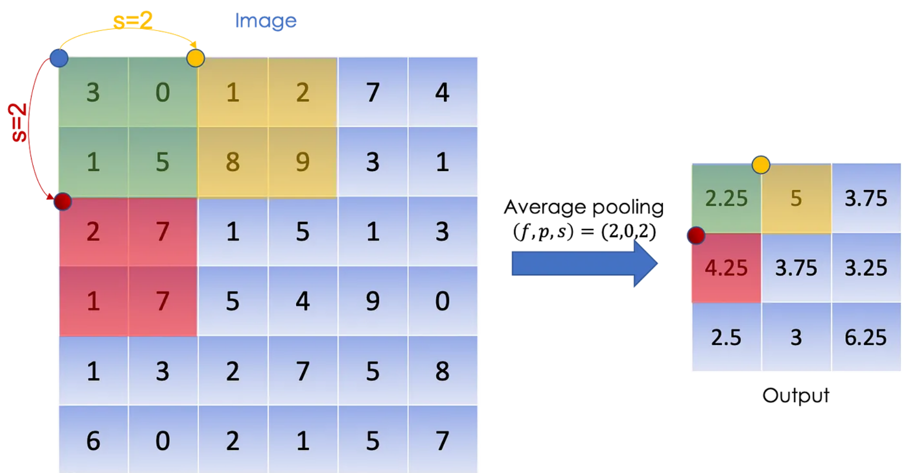

Figure 4.3. Applying a pool layer (average pool) on a simple image represented as a 2D matrix of integers [Mebsout, 2020]

And the most important is the convolutional layer. It has kernels that slide (or convolve) over the input. In each step, there is a different part of the image in the kernel. Afterwards, the scalar product is calculated. The output is known as *activation*. For a 2D-image, the output of one of those kernels would be a 2D-activation map. These would be stacked on top of each other and produce the output of the convolutional layer. There are hyperparameters to optimize, such as the size of the kernel, stride (how much the kernel moves in each step), padding (for example, with zeros outside the original input image) [O'Shea, 2015].

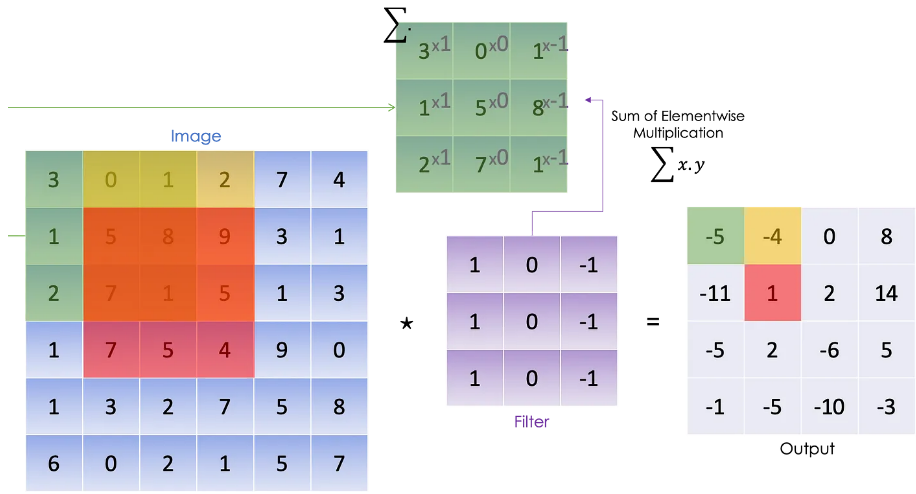

Figure 4.4. Applying a filter on a simple image represented as a 2D matrix of integers [Mebsout, 2020]

This strategy is also useful in NLP when one wants to analyze sentences or documents. The kernels will move across the words in the document. This allows the net to capture word order and word proximity. Both of which are important in natural languages. One issue is that the lengths of the documents are predetermined by the dimensions of the input layer. Shorter texts must be padded, and longer ones must be truncated [Hobson, 2019].

The bigger problem is that one can't use words (or other strings) as inputs for neural networks. It requires a numerical representation. There are two major approaches: One-hot encoding and Word2vec. But Word2vec requires data and computational time to train in order to work. Unlike the simpler one-hot encoding [Hobson, 2019].

CNNs can be used for the classification of some labeled text. For example, written reviews and corresponding ratings approved, disapproved; recommend, don't recommend; fresh, rotten; rating out of five (or 10) stars [Hobson, 2019].

4.4 Recurrent Neural Networks

First RNNs (Recurrent Neural Networks) were introduced in the 1980s for learning strings of characters. Development and research of RNNS and their other application continued in the 1990s. The key feature is using closed-loop connections. There are different architectures for RNNs. It can be *fully connected RNN*, where each node is connected to every other node (even to itself) [Medsker, 2001].

Or much simpler RNNs are modifications of Feedforward NN where certain layers are giving information (feedback) back to proceeding layers. There are two major ways of doing this. The vector given to the input layer includes the output of either the hidden or output layer. The training data is, for example, some strings of characters. The net gets a character and is taksed with predicting the proceeding character. These networks can be trained to either predict the next character or to check and control the correctness of text string [Medsker, 2001].

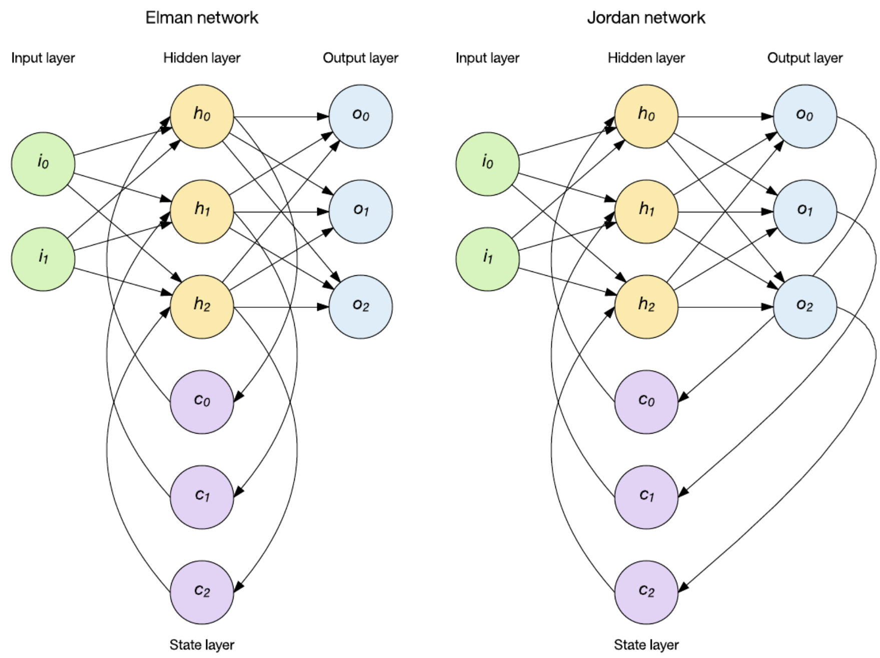

Figure 4.5. Two common architectures of Recurrent Neural Networks [Ramos, 2020]

RNNs can learn dependencies from sequential data. That is important in natural languages, where the data is a sequence of words (or characters). It is important to know what words came before the current word. The recurrent (loop-back) signal can also be looked at as a sort of memory for the network. RNNs can take as the input texts of variable length (unlike CNNs). However, very long ones can cause problems such as *vanishing* (or *exploding*) gradients. This happens because the weights are reused in each step, and they can multiply the signal into infinity or into zero [Salehinejad, 2018, Hobson, 2019].

Another issue with RNNs being very deep networks is that they are hard to train compared to other models. They can, however, improve performance. RNNs models were an important breakthrough for modeling natural languages as sequences of characters [Hobson, 2019, Salehinejad, 2018].

4.4.1 Long Short-Term Memory Networks

LSTMs (Long Short-Term Memory Networks) have further improved RNNs because they can retain the knowledge of long-term dependencies better than a *hidden state*. This state refers to the output of the hidden layer that is then passed to the input layer in the next step in RNNs. LSTMs introduce the *memory state* for the hidden layer. This allows it to retain information longer on top of the more recent memory that comes from the architecture of RNNs, where the output of the hidden layer is added to the input in the next step [Hobson, 2019, Salehinejad, 2018].

For controlling memory, three gates are used. *Forget gate* is a simple feedforward NN. Its output is a vector that determines which values in the memory should be (and how much) forgotten in the memory vector. This vector has values from 1 (value stays in memory) to 0 (value is completely zeroed). The modified memory state continues to the*candidate gate*. This one has two separate parts: candidate choice and candidate values. The first represents how important it is to remember (output values from 0 to 1). And the latter has values from −1 to 1. These vectors are multiplied elementwise and added to the memory state. That is the second update to the memory. And the last gate - the *output gate* - uses this memory to modify the output. The output would otherwise correspond to RNN with the same dimensions. But just before outputting, it is multiplied elementwise by a mask created from the memory state. Thus, memory has a crucial influence on the output of the LSTM layer [Hobson, 2019, Salehinejad, 2018].

LSTMs are more resilient toward the issue of vanishing (or exploding) gradients than RNNs. They, however, require more memory, and they further increase computational complexity. GRUs (Gated Recurrent Units) can be used instead of LSTMs in order to decrease memory demands. GRUs also have gating units but without memory cells. They are better at certain tasks and worse at others compared to LSTMs [Salehinejad, 2018].

Chapter 5
Sentiment Analysis

When tasked with analysing opinions of written text, we turn to sentiment analysis (SA). This field is also known as emotion analysis, opinion extraction, sentiment mining, and others. Or rather, they are focused on slightly different problems from different approaches. However, they are all around the same area of problems that consist of capturing and analysing opinions, sentiments and emotions of people from what they have written [Liu, 2012].

Very little research was done prior to the year 2000. This changed with the rise of the internet. Suddenly there was a huge amount of text that could be used for sentiment analysis. Furthermore, this analysis has commercial applications. And thus, there is a greater financial incentive to improve current methods of SA and develop new ones. Examples of applications include analysing reviews; social media posts (or blogs) for opinions on products, people, political parties, companies, etc. The analysis can be used to predict stock market changes and improve trading strategies. Or internally by the company to distinguish well-liked products from hated ones [Liu, 2012].

5.1 Levels of Sentiment Analysis

The simplest way is to do sentiment analysis at a *document level*. This takes the entire document and outputs the measured sentiment. This may be very useful for corpora where the documents are focused on a single thing. An excellent example are reviews which only focus on the thing reviewed. However,even here, an issue with entities arises, e.g., "The video game was really great. But the movie adaptation was terrible". When analysed as a whole, it is difficult to tell (even for a human) whether this has positive or negative sentiment because there are two things with two different sentiments. Another key piece of information is the domain of the document. If it was a movie review, it would have a very negative sentiment, but as a video game review it is more positive [Liu, 2012].

A different approach is to use a *sentence level* analysis. This splits the document into sentences which are analysed individually. It tries to differentiate sentences that are *objective* (describe something, carry factual information) and *subjective* sentences that carry opinions and thoughts [Liu, 2012].

Aspect level or *Entity level* tries to solve the issue with multiple things in one sentence, as in the review example. It tries to extract not only the opinions or sentiments in the text but also to which entity (i.e., movie, game) this sentiment relates. This is even more challenging than document or sentence level analysis, and these are already quite hard problems [Liu, 2012]

5.2 Sentiment Lexicons

The most important carriers of meaning are words. Those that carry (positive or negative) sentiment are called *sentiment* (or *opinion*) *words*. If we focus on those (and ignore the sentence structure), we can create *sentiment lexicons*. There are sentiment words and their respective sentiment in these lexcions. Additionally common phrases or idioms can also be used as "words" in the lexicon. Sentiment lexicons are a necessary step in sentiment analysis [Liu, 2012 ??].

However, they are certainly not ideal and have several issues. Firstly the same words do not necessarily have the same sentiment in different contexts. The difference may be due to sarcasm or several meanings of the word. Some sentences may have no opinion words and imply an opinion, e.g., "This car uses ten times more fuel than my previous one.". And others may have an opinion word (e.g., nice) and carry no sentiment, e.g., "What movies from the last decade are really *nice*?" [Liu, 2012].

5.2.1 Creation of Sentiment Lexicons

There are several general-purpose publicly available lexicons already created. Depending on the context, this may be sufficient for the desired application. However, certain domains and contexts are more challenging. Thus creating a domain-specific lexicon may improve additional steps in SA. And in languages that lack publicly available lexicons, creating one is the only option [Liu, 2012].

The simplest is the *manual approach*, but its disadvantage is high labor requirement and time consumption. Its usually used in combination with automated approaches [Liu, 2012].

Dictionary-based approach uses already existing dictionaries for the language. The dictionary used must contain synonyms (that is common in most dictionaries) and antonyms. We start with a few words that have known sentiments (good, bad, terrible, awesome, etc.) predefined manually in the lexicon. Next, their synonyms and antonyms are added to the lexicon with the corresponding sentiment. For newly discovered words, we repeat the process until no new words are added.

The final lexicon can be manually inspected, and unfitting words can be removed. This can also be looked at as traversing a graph where the notes are individual words, and the edges are connections (synonym, antonym). These edges may have different weights for synonyms and antonyms. But the absolute weight should be lower than one. This makes the words separated by many connections (and therefore the most different from the original set) have lower sentiment values [Liu, 2012].

Corpus-based approach is used when adapting an already existing general-purpose lexicon for a specific domain. It can aslo be used to generate a general-purpose lexicon with enough data and sufficient diversity in the data. However, for constructing general-purpose lexicons, the dictionary-based approach is typically better [Liu, 2012].

Chapter 6
Implementation and Tools Used

We chose to implement everything in Python because of its vast collection of libraries. Python can be used for scraping the web, extracting text from HTML source code, and Natural Language Processing. Very little preprocessing of the natural language test was done after it was extracted from HTML. This is due to the loss of information that occurs and because all of these processes are harder to do accurately in Czech than in English. And even in English, there is a trend of using less preprocessing and adding more data and computational time to allow the model to capture more information.

6.1 Toolkit Overview

6.1.1 ZipFile

Due to the way file systems work, it is difficult for them to have a large number of (even small) files in a single directory. Easier is to have one large file. And when creating large datasets (in this case, thousands of articles, each having tens, hundreds, or even (low) thousands of comments), this may be a problem. Modern operating systems have some techniques to mitigate this issue, but from personal experience, these are not sufficient.

Folders with thousands of files operate slowly and sometimes even slow down other tasks. For this reason, we decided to batch the files into larger archives. Tar files are an inferior alternative because they compress all the files together (better compression but slower reading and writing). This makes it hard to random access or append new files. Zip ,on the other hand, compresses the files individually. Therefore, changing, adding, and deleting files within the archive is easy. Text files are not very large (even an entire HTML of a website is usually just a few kilobytes). Additionally, we do not care that much about compression - it can even be turned off for even faster access. This makes using zip files the best choice.

6.1.2 Sqlite

For managing what links were already scraped and what are yet to be scraped, some form of database is ideal. Sqlite was chosen because it is free to use, stable, small, simple, and fast [1]. For interacting with it from Python code, sqlite3 library is used.

6.1.3 NLTK, gensim

Natural Language ToolKit (NLTK) is a Python library that has wide range of tools for computational linguistics. It also has some beginner-friendly interfaces to several corpora. However, it is mainly designed for the English language, and for the Czech language, some things (stemmer, lemmatizer, tagging) are useless. The most useful is the tokenizer because the punctuation and word separation work almost the same way in both languages.[2]

[1] `https://www.sqlite.org/index.html`

[2] `https://www.nltk.org`

The *Gensim* library is designed for NLP tasks. It has several models for unsupervised document analysis - Word2Vec, FastText, LSA, LDiA. In our case, we will use its Word2vec model. [3]

6.1.4 BS4

Beautiful Soup (BS4) is a Python library for HTML parsing. It can be used both for finding and changing information in HTML documents. It can do almost anything with a given HTML file[4]. However, it has no capabilities for downloading them from the web. For this, one must use requests or urllib2 libraries.

6.1.5 Selenium, Chromedriver

This tool was first developed for testing the functionality of websites when they are being developed. It allows for running JS code and interacting with the webpage like a user - scrolling, clicking, filling in forms, etc. [Selenium Docs]. This gives the ability to automate almost any repetitive web-based task. It is an ideal candidate for web scraping anything.

Selenium WebDriver is a simple programming interface that drives the browser effectively.[5] It creates an instance of the browser that is controlled by the programmer's script. This means that the website has to be rendered. This can make it slower compared to BS4 with requests. Therefore we use it only where these are not sufficient, and rendering the website in the browser is necessary for the site to function correctly.

6.1.6 Polyglot, Maplotlib

For sentiment analysis, we use `polyglot`'s sentiment lexicon. Polyglot is a Python package based on [Chen, 2014]. To visualize the results, we use Python's matplotlib library as it is the most common one to use for plotting and has sufficient capabilities for most types of data.[6]

6.2 Web Scraping

The most crucial step is a good selection of what news sites are best to scrape. For machine learning is essential to have huge amounts of data, so it cannot be some obscure site. Ideally, it would also have big comment sections with a diverse set of readers. And from a scraping perspective, it would be convenient if the articles and their respective comment sections have a simple layout that can be scraped and parsed easily.

In Anglospere, comment sections in online news have been slowly disappearing. This is in part due to spam and the difficulty of moderation [7]. They are arguments for [8] or against [9] this. However, this trend did not arrive (yet?) in Czech media, and there are several news websites with thriving comment sections under popular articles.

[3] `https://radimrehurek.com/gensim/intro.html`
[4] `https://beautiful-soup-4.readthedocs.io/en/latest/`
[5] `https://www.selenium.dev/documentation/webdriver/`
[6] `https://matplotlib.org`
[7] `https://www.getfoundquick.com/why-are-comment-sections-disappearing/`
[8] `https://www.theguardian.com/science/brain-flapping/2014/sep/12/comment-sections-toxic-moderation`
[9] `https://www.techdirt.com/2015/09/23/trend-killing-news-comment-sections-because-you-just-really-value-conversation-stupidly-continues/`

6.2.1 iDNES.cz

At `https://www.idnes.cz`, we can find a news site with a simple layout of both articles and comment sections. The pipeline looks roughly like this. First, we scrape some individual sites on the domain using requests for downloading. We use Sqlite for managing what was already scraped.And we use BeautifulSoup4 for extracting links from the HTML file.

- Download a website from idnes.cz and mark it scraped in the database
- Save it into zip file
- Find all links to the same domain
- Add links to the database
- Fetch a new unscraped link from the database
- Return to step 1 or stop if a sufficient amount was scraped

The next step is selecting all articles and extracting their headline, opener, and text using bs4 and saving the extracted part for future use. This severely reduces the size of the dataset (for example, 2.5GB into 50MB) so it can be easily transferred elsewhere. A link to the discussion is extracted and saved. And they are scraped and saved correspondingly to articles. The difference is every comment has its individual file in zip file. And every comment has its rating saved. The ratings are simple counts of `pluses` and `minuses` given by other users. This can be a signal used further in sentiment analysis.

6.2.2 Seznam Zprávy

This site (`https://www.seznamzpravy.cz/`) has a more complex design, but articles can be scraped with relative ease. The process is very similar to the process outlined in 6.2.1. The issue comes with the comment section. It is difficult to scrape because it uses on-demand loading that requires clicking. For this reason, it is necessary to use Selenium.

One of the first issues comes with cookies, specifically the popup that covers the screen. For humans, it is easy to click. In Selenium, this is a bit trickier. One has to locate the button in the HTML source code. Find some way to identify it so it cannot be mistaken with other elements - ideally `id` but `class` can also be sufficient in certain situations. Next, the element is selected with the method of the Selenium driver: `driver.find_elements()`. After that follows a simple call `element.click()`. And in most cases, this is a completely sufficient approach. In this case, however, the cookies popup is behind:

```
#shadow-root (closed)
```

“The ShadowRoot interface of the Shadow DOM API is the root node of a DOM subtree that is rendered separately from a document’s main DOM tree.”[10]. From a scraping perspective,this is not a big problem because Selenium has methods for accessing these special types of elements. The real issue is in `(closed)`. This makes the internals inaccessible to Javascript [11].

Selenium cannot operate within these. It is not a bug but an intentional design because these should not be accessible for scripts[12]. In this case, the cookies window could be removed by deleting the parent element. And the site remained functional.

[10] `https://developer.mozilla.org/en-US/docs/Web/API/ShadowRoot`
[11] `https://developer.mozilla.org/en-US/docs/Web/API/ShadowRoot/mode`
[12] `https://github.com/SeleniumHQ/selenium/issues/5869#issuecomment-388821755`

This however, well illustrates the type of issues that can be encountered when webs craping. Their complexity and difficulty depend on the type of website that is being scraped.

6.3 Natural Language Processing

Some parts of the source code use samples from [Hobson, 2019] and their examples at respective GitHub. On some versions of Python and libraries, one may encounter an error about Mapping not existing. This can be easily solved by rewriting the files of the respective library. It is necessary to change

```
from Collections import Mapping
```

into

```
from Collections.abc import Mapping
```

6.4 Word2vec

For constructing the Word2vec representation of the words, we use the gensim library. It expects the data already split into words. For this, we use a tokenizer from the NLTK library. It was designed for the English language, but Czech has similar rules for punctuation, so it works fine. It compares favorably to using a simple tokenizer that uses only the default string split method in Python.

6.5 Dataset

The contents of the dataset that was created will not be made public due to potential legal issues. In Table 6.1 we see an overview of the size of the dataset that was used in further steps. The `Size Extracted` refers to natural text only without all the HTML tags present in `Size`. It is missing in `Seznam Zprávy Comments` because it was extracted dynamically to save some hard disk space and to efficiently use the time that would otherwise be spent waiting.

In the `Articles` column is the number of articles, and in the bracket is the number of individual comments. An unfortunate decision was made to give each comment its own file in the zip archive when extracting iDNES comments. This made the write time into the archive slow and the file unnecessarily large due to a lot of long filenames.

A Better alternative was used in `Seznam Zprávy Comments` - saving all comments from one article into one file. One comment per line format requires replacing any new-line characters with different white spaces. But that is a negligible loss of information.

Name	Articles	Size	Size Extracted
Seznam Zprávy Articles	6772	3.1 GB	41.6 MB
Seznam Zprávy Comments	1825 (237303)	-	42.5 MB
iDNES Articles	12296	3.6 GB	47.7 MB
iDNES Comments	11795 (225797)	1.3 GB	98.3 MB

Table 6.1. The four parts of the dataset. The number in brackets refers to the number of individual comments.

Chapter 7
Experiments and Discussion

7.1 Word2vec

This section was inspired by [Wójcik, 2019] which uses vector representation of words (word2vec). It clusters all these vectors into two groups. Afterwards, it looks at the words closest to centroids (the centers of the clusters). One group has more positive and the other has more negative sentiment. The weight of the sentiment of each word depends on the distance from the closest centroid (closer ones have more sentiment).

The advantage of this approach is that it requires no labeling of the words or documents. It only needs enough textual data. But the disadvantage is that it is not precise nor accurate [Wójcik, 2019]. As there is no solid justification for why the distance would depend more on sentiment than other characteristics. For example it, could separate the clusters into verbs and not-verbs.

The first experiments used only some parts of the collected dataset to create word vectors. Then we looked at words (adjectives) that carry sentiment (good, nice, awesome, bad, terrible, awful). Afterwards, we looked at the words most similar to them (smallest distance between vectors); see 7.1 for first five words.

dobrý 1.0	můj 0.8625	špatný 0.8466	pěkný 0.8099	slušný 0.8058	nápad 0.8044
dobré 1.0	těžké 0.8545	důležité 0.7891	nutné 0.765	vhodné 0.7638	špatné 0.7502
dobrá 1.0	hezká 0.8863	skvělá 0.8583	pěkná 0.8534	krásná 0.8313	taková 0.823
hezké 1.0	pěkné 0.8922	svinstvo. 0.857	těžké. 0.8515	zavádějící, 0.8392	pěkné, 0.8354
super 1.0	super, 0.8083	blbost 0.7454	pěkná 0.7129	liga. 0.7114	ok, 0.7074
špatné 1.0	důvody 0.8137	silné 0.807	skutečné 0.806	rozumné 0.7977	hezké 0.7965
strašné 1.0	zřejmé. 0.8695	zbytečné. 0.8679	hrozné. 0.8638	šílené 0.8625	trapné 0.8562
hrozný 1.0	strašnej 0.8883	Hřib 0.8828	krásný 0.8821	slušnej 0.8818	nádherný 0.8798

Table 7.1. 5 Most similar words in Word2vec model made from only comments from only `www.idnes.cz` with length 200

These results show that similar words are usually the same part of speech (adjectives) or are used together in common phrases ("dobrý nápad" or "pěkné svinstvo"). However, a lot of the words are close in meaning. And this could hopefully be used in sentiment lexicon generation similar to*Dictionary-based generation* (see 5.2.1). The difference would be that instead of prelabeled connections in dictionary (synonyms, antonyms), it would use connections found without supervision by looking for most similar vectors in word2vec representation.

Further experiments with different parts of the dataset and different parameters were done. Unsurprisingly the more data was used and the more diverse it was (articles and comments), the better the representation looked, e.g., 7.2. Some of the others can be seen in Appendix B.

dobrý 1.0	špatný 0.792	skvělý 0.7898	hezký 0.7548	těžký 0.7449	výborný 0.734
dobré 1.0	správné 0.7161	důležité 0.7004	špatné 0.695	skvělé 0.6778	rozumné 0.6532
dobrá 1.0	skvělá 0.8196	špatná 0.8015	zajímavá 0.7739	pěkná 0.7571	důležitá 0.7417
hezké 1.0	pěkné 0.8475	skvělé 0.7558	fajn 0.7519	příjemné 0.7438	zábavné 0.7203
super 1.0	fajn 0.7494	parádní 0.6965	perfektní 0.6638	pecka 0.6499	bezva 0.6423
špatné 1.0	správné 0.7091	dobré 0.695	nepříjemné 0.6554	rozumné 0.6459	složité 0.6429
strašné 1.0	hrozné 0.7868	úžasné 0.7218	šílené 0.7217	děsivé 0.7172	marné 0.6988
hrozný 1.0	strašný 0.7946	neskutečný 0.7334	dobrej 0.7228	blbý 0.7151	vtipný 0.7003

Table 7.2. 5 Most similar words in Word2vec model made from both comments and articles from both `www.seznamzpravy.cz` and `www.idnes.cz` with length 200

This is an improvement. However, the table is severely lacking declensions (3.1) and other parts of speech like adverbs. And the biggest issue is that there are still words with exactly opposite meanings ('good' - 'dobrý'; 'bad' - 'špatný'). Which is not that surprising because they are used similarly. However, it is difficult to deal with when trying to create a sentiment lexicon from these relations.

Next, we look at different declensions ('dobrých',dobří'','dobrými' are all translated as good) and different parts of speech. Instead of adjectives ('good' - 'dobrý'; 'bad' - 'špatný') we use adverbs ('well' - 'dobře'; 'badly' - 'špatně'). In Table 7.3, we see that the most similar words to declined adjectives are typically declined in the same form. And that adverbs also have adverbs close to them instead of other parts of speech.

Another issue can be noticed. Some words that appear to have no relation are listed as similar. The reason may be that they are very rare words. And thus, there are very few contexts in which they appear in the dataset. This issue is, however, negligible because these words are very rare. Therefore they have statistically very little influence on the measured sentiment of a long text (e.g., an article) or multiple short texts (e.g., comments on an article).

dobrých 1.0	špatných 0.7401	úspěšných 0.7264	skvělých 0.7044	zajímavých 0.7039	odlišných 0.703
dobří 1.0	silní 0.8528	chytří 0.84	hloupí 0.8352	špatní 0.8312	takoví 0.8259
dobrými 1.0	původními 0.6973	vybranými 0.6873	cizími 0.6841	notebookem 0.6784	klukama 0.6728
špatnou 1.0	podobnou 0.7534	dobrou 0.745	zajímavou 0.7229	jistou 0.7039	těžkou 0.6955
špatně 1.0	dobře 0.6553	blbě 0.6398	správně 0.5618	špatné 0.5341	obráceně 0.5299
dobře 1.0	špatně 0.6553	skvěle 0.6388	hezky 0.584	slušně 0.5714	lépe 0.5484
hroznými 1.0	radama 0.6709	krabicemi 0.6642	písečným 0.6635	Seznámí 0.6612	kvantovou 0.6612

Table 7.3. Adjectives Declensions, Adverbs, 5 Most similar words in Word2vec model made from both comments and articles from both `www.seznamzpravy.cz` and `www.idnes.cz` with length 200

7.2 Creating Custom Sentiment Lexicon

In section 5.2.1, we described a dictionary-based approach for creating sentiment lexicons. This requires some dictionary with synonyms and antonyms. We decided to adapt this approach to data that have no labels. First, we created a short (cca 25) list of positive and negative words, see Table C.5. This includes some adverbs, adjectives, and their declined forms.

Afterward, we added words that were closest to them in word vector representation. Their sentiment is a product of the sentiment of the original word (1 positive, -1 negative) and similarity (1 - identical, 0 - least similar word possible). We repeat this process for newly added words. This is repeated a predefined number of steps or until a certain limit of the sentiment value is reached. These hyperparameters can be further optimized.

When creating the sentiment lexicon for this case, the minimal similarity of the word in order to be considered is 0.78. The absolute sentiment value cannot be lower than 0.5. And the words must be reached within 4 steps. With hyperparameters set as described, we create a lexicon with 63 negative words and 283 positive words. The negative words seem more precise even if there are fewer in number. As can be seen in a random slice of the lexicon in Table 7.4.

hloupí	0.8351936936378479	nácek	-0.6510593286438748
špatní	0.8312200903892517	exot	-0.6502488769822783
takoví	0.8258762955665588	ubožák	-0.633749006781585
pracovití	0.8197283744812012	chlapík	-0.6324555639875804
skvělí	0.8151135444641113	kašpar	-0.6286448012458017
šikovní	0.8002576231956482	psychopat	-0.6277053543282918
spokojení	0.7989729046821594	buran	-0.626571123618124
neschopní	0.7932671308517456	šmejd	-0.6464964859506495
úspěšní	0.7898635864257812	gauner	-0.6260299209316145
líní	0.7857365608215332	neuvěřitelně	-0.5633571806819127
nemocní	0.7827789187431335	parazit	-0.510118376605857
blbí	0.7820891737937927	narcis	-0.5103853842711382
slabí	0.7809988856315613	klaun	-0.5265567455240019
krásná	0.7980693578720093	nacista	-0.5137524240229452
výborně	0.8387442231178284	udavač	-0.5102098096754933
zajímavá	0.6541010589807996	papaláš	-0.5099974623679102
výborná	0.6530309216136096	ožrala	-0.508373308152948
horšímu	0.6832842449469823	komediant	-0.5171368901037915
lepšímu.	0.6548449585454001	zmetek	-0.5123328541837703

Table 7.4. Random slice of words that were added to the sentiment lexicon

7.3 Comparing Created Lexicon with Other Sentiment Measures

7.3.1 VADER and translation

VADER (Valence Aware Dictionary and sEntiment Reasoner) is a tool for analysing sentiment described in [Hutto, 2014]. It uses more than just sentiment lexicons. It was designed to handle text from social media and is free to use. [1]

This makes it seem like an ideal tool to compare with our simple sentiment lexicon. But the issue is that it is designed for English texts only. This is the issue of most publicly available tools for sentiment analysis.

One solution is to use machine translation on comments collected in the dataset. This is certainly not ideal, as information would be lost in translation, and some things would be translated incorrectly. We have tried translating using several Python libraries (`translate`, `Translators`, `googletrans`, `dl-trans`), but none of them was capable of translating the entire dataset due to its size. And all of the public APIs of online translators limit the amount (and size) of responses.

The second option was to translate only the words present in the corpus and then replace these when evaluating sentiment in a different language. This would make any more advanced analysis than a simple count of positive (and negative) words inapplicable because the word order can be very different in Czech and English.

We used `translate` - Python library that uses Google Translate as a backend. Even this proved to be too long, and only part of the vocabulary was translated. Given the poor quality of what was translated (see 7.5), we decided not to continue in this direction. The translated text would be too littered with incorrect (or weird) translations.

[1] `https://vadersentiment.readthedocs.io/en/latest/`

Czech	English
než	The total
ani	ani
podle	Sort by
ještě	yet.
až	Euro up to
jsme	tied up
být	Could
bylo	I
toho	its
jeho	his
pak	then
tím	Tím

Table 7.5. Random Slice of Translations of Words in Corpus by `translate`

7.3.2 Polyglot and Sentiment Lexicons Comparison

One of the few publicly available sentiment lexicons with support of the Czech language is in `polyglot`. It is a Python library that implements [Chen, 2014]. To compare our sentiment lexicon created above with this a simple measure was used. We took an entire comment section and looked at the word distribution. The ratio of postive (negative) words to all words was used.

In the positive words (see 7.1), these two measures do not seem to be correlated. This distribution can probably be explained by the fact that our lexicon does not have exhaustive list of positive words. So most texts end up having very few sentiment words in general, and the data is squished near the x-axis.

If we were to compare the absolute counts of positive and negative words, some relation can be seen (D.2), especially for positive words D.1. However, this is caused by long texts simply having more words (of all sentiments). Both of these measures are dependent on the length of the text. Therefore, the correlation is mostly caused by this. As we do not see a similar relationship in relative counts.

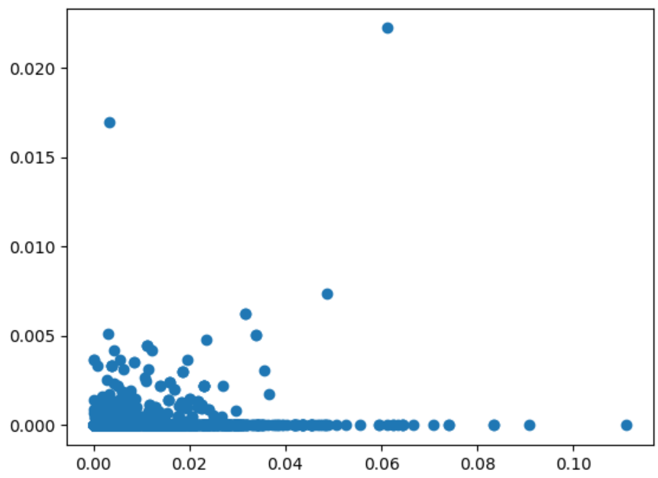

Figure 7.1. Comparison of ratios of positive sentiment words in Polyglot (x-axis) and our lexicon (y-axis)

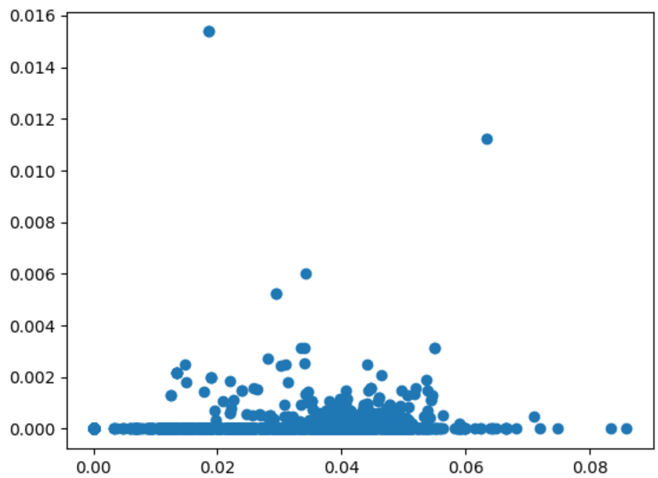

Figure 7.2. Comparison of percantage of negative sentiment words in Polyglot (x-axis) and our lexicon (y-axis)

7.3.3 Distribution of Sentiment in Dataset

Now we look at our dataset through the lens of polyglot sentiment analysis. In Figure 7.3 we see that there are many articles with comment sections that have more positive comments than negative ones. However, most of the data clusters towards more negative than positive.

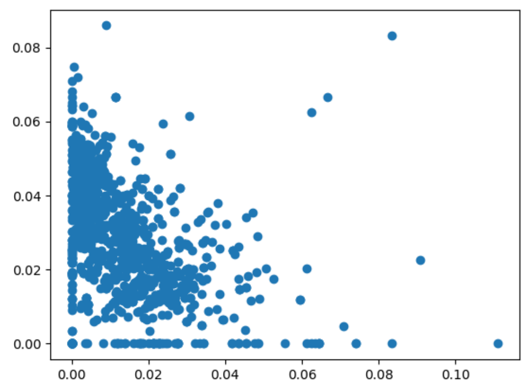

Figure 7.3. Ratios of positive words (x-axis) vs ratios of negative word (y-axis)

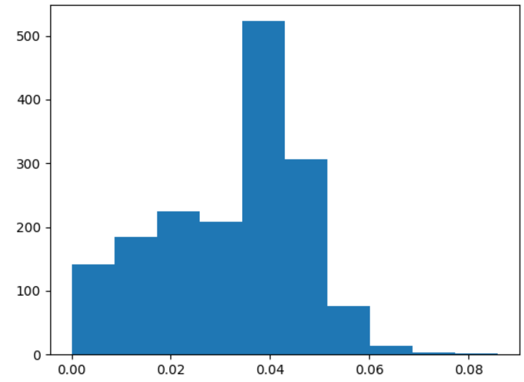

Figure 7.4. Histogram of ratios of negative words

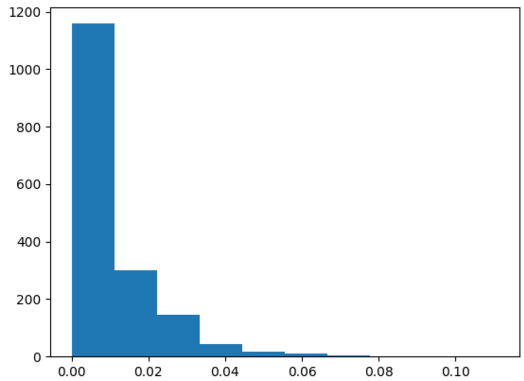

Figure 7.5. Histogram of ratios of positive words

This can also be seen in histograms (Figures 7.5, 7.4). Where most sections are located in the 0% to 2% bins for positive words. On the other hand, negative words have much more even distribution. This is not surprising as most news are somewhat related to politics. And these discussions can get pretty heated. Another reason may be that people head to comment when they want to discuss the article, and this happens more when there is something they do not like or disagree with.

These distributions are different for different topics. Politically charged topics such as 'Ukraine' or 'Presidential Elections' tend to have the distribution skewed toward more negative words. See Figures D.3, D.6. Topics that are more neutral (for example articles about advances in science) have distribution skewed towards less negative words, see Figure D.9 for topic 'Universe'.

7.3.4 Likes versus Sentiment

As we have seen above, sentiment analysis with unlabeled data is difficult. In the case of analysing reviews, one can use the rating(positive, negative; recommend, do not recommend) as a class. This makes it a supervised problem, and other methods can be used, such as NN, for classification as they are described in 4.3.

Figure 7.6. Example of iDNES comment

In our case, we do not have a label associated with the comment. We have only other commentators' reactions to the comment (e.g., 7.6). In iDNES, these are simple likes and dislikes. To see whether they are related to the sentiment, we use ratios again. We divide the number of positive (negative) reactions by the total number of reactions. To see if there is any relation to sentiment, we use once again the ratios of positive (negative) words to all words.

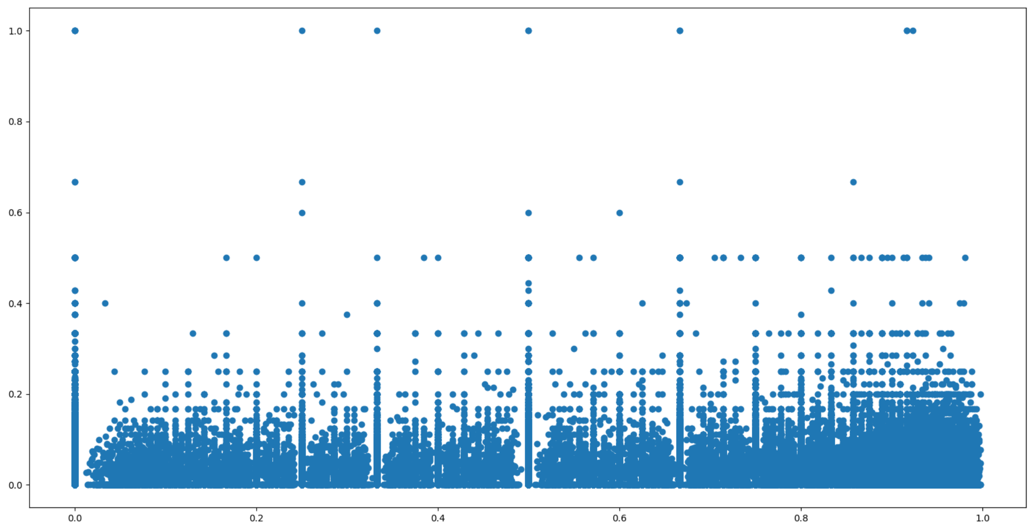

Figure 7.7. Scatter plot of the ratio of positive reactions (x-axis) versus ratio of positive words (y-axis)

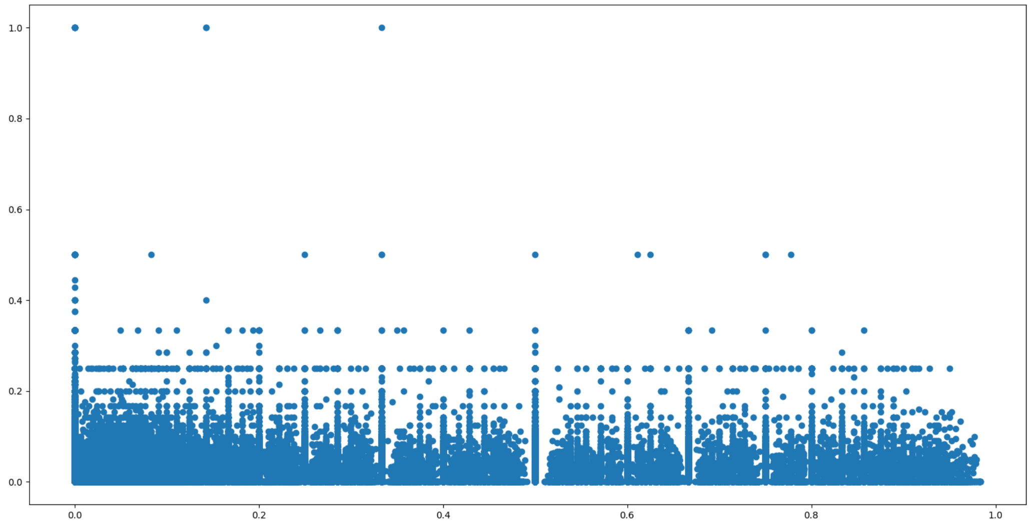

Figure 7.8. Scatter plot of the ratio of negative reactions (x-axis) versus ratio of negative words (y-axis)

We plot them together in Figures 7.7, 7.8. Note these weird horizontal and vertical lines are common fractions ($\frac{1}{2}, \frac{1}{3}, \frac{2}{3}$, etc.) that are over-represented in the dataset as most comments do not get much attention (both likes and dislikes). From these images, it is apparent that there is very little to no relation between these two measures. The densities in the areas are not surprising given the histograms D.12, D.13.

This is the expected result because the likes capture sentiment *to* the comment. And sentiment words capture the sentiment *of* the comment. This can also be seen in the example comment (Figure 7.6), where the comment expresses negative sentiment of the article. Other commenters agree (have positive sentiment towards the opinion) and upvote the comment and create a dichotomy between negative sentiment and positive rating.

Chapter 8
Conclusion

In the Implementation Chapter (6), we have looked at word vectors created by Word2vec. Most similar vectors appear to have similar meanings or are used in similar contexts (declined in the same form, are used next to each other). When creating these vectors, we optimized hyperparameters and used more (and more diverse) data. Next, we tried to select the best representation, where similar vectors have similar meanings.

Then we adapted the *dictionary-based approach* for creating sentiment lexicons described in 5.2.1. Instead of using antonyms and synonyms, we used the distance to other vectors in word vector representation. We started with a list of a few positive (26) and a list of a few negative (25) words. We looked for similar word vectors and added them to these lists. By repeating for a few steps, we had a sentiment lexicon bigger than the original one - 283 positive and 63 negative words.

Afterwards, we compared this created lexicon to an already existing sentiment lexicon in the Python package `polyglot`. The clear winner was `polyglot` lexicon. Mainly because our custom lexicon still had way too few words. The performance of our model could be improved by extending the starting lists, but this would make it more labour intensive and closer to the *manual approach*. Thus we conclude that this approach is worse than the normal *dictionary-based approach*. However, when no dictionary with synonyms and antonyms is available, this may help speed up the otherwise *manual approach*.

In section 7.3.3, we analyzed how the sentiment is distributed. Showed that different topics have different distributions of positive and negative words present in comments. For example, more divisive topics (like politics) have more negative words than neutral topics (like science).

We saw that unsupervised learning did not work well for sentiment analysis and sentiment lexicon creation. Therefore as the last thing, we looked at if the reactions (likes, dislikes) to comments could be used as labels, thus changing this into a supervised problem. However, we reasoned and showed that there does not seem to be a relationship between how positive are the reactions to the comment and how positive is the comment itself.

References

The HTML syntax.
`https://html.spec.whatwg.org/multipage/syntax.html`. Accessed: 20.01.2023.

Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with python.* Sebastopol, CA: O'Reilly Media, 2009. ISBN 978-0-596-51649-9.

Yanqing Chen, and Steven Skiena. *Building Sentiment Lexicons for All Major Languages.* In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers).* Baltimore, Maryland: Association for Computational Linguistics, 2014. 383–389.
`https://aclanthology.org/P14-2063`.

Lane Hobson, Howard Cole, and Hapke Hannes. *Natural Language Processing in Action.* New York, NY: Manning Publications, 2019. ISBN 9781617294631.

C. Hutto, and Eric Gilbert. VADER: A Parsimonious Rule-Based Model for Sentiment Analysis of Social Media Text. *Proceedings of the International AAAI Conference on Web and Social Media.* 2014, 8 (1), 216-225. DOI 10.1609/icwsm.v8i1.14550.

David Kriesel. *A Brief Introduction to Neural Networks* . 2007 . available at `http://www.dkriesel.com` .

Bing Liu. *Sentiment Analysis and Opinion Mining.* Morgan & Claypool, 2012 .

Christopher D Manning, Prabhakar Raghavan, and Hinrich Schutze. *Introduction to Information Retrieval.* Cambridge, England: Cambridge University Press, 2008.

Ismail Mebsout. *Convolutional Neural Networks' mathematics — towardsdatascience.com.*
`https://towardsdatascience.com/convolutional-neural-networks-mathematics-1beb3e6447c0`. 2020. Accessed 24-May-2023.

Larry R Medsker, and LC Jain. Recurrent neural networks. *Design and Applications.* 2001, 5 64–67.

Ryan Mitchell. *Web Scraping with Python, 2e.* Sebastopol, CA: O'Reilly Media, 2018. ISBN 978-1-491-98557-1.

Jose Portilla. *A Beginner's Guide to Neural Networks in Python.*
`https://www.springboard.com/blog/data-science/beginners-guide-neural-network-in-python-scikit-learn-0-18/`. 2017. Accessed: 24.05.2023.

Keiron O'Shea, and Ryan Nash. *An Introduction to Convolutional Neural Networks.* 2015.

Raúl Ramos, and Julián Arias. *Fundamentos de Deep Learning — rramosp.github.io, 5.1 Recurrent Neural Networks;.*
`https://rramosp.github.io/2021.deeplearning/content/U5.01%20-%20Recurrent%20Neural%20Networks.html`. 2020. Accessed 24-May-2023.

Hojjat Salehinejad, Julianne Baarbe, Sharan Sankar, Joseph Barfett, Errol Colak, and Shahrokh Valaee. Recent Advances in Recurrent Neural Networks. *CoRR.* 2018, abs/1801.01078

Selenium Docs.
https://www.selenium.dev/documentation/. Accessed: 04.05.2023.

Skloňování přídavných jmen, mojecestina.cz.
https://www.mojecestina.cz/article/2009092802-sklonovani-pridavnych-jmen. Accessed 24-May-2023.

Rafał Wójcik. *Unsupervised Sentiment Analysis.*
https://towardsdatascience.com/unsupervised-sentiment-analysis-a38bf1906483. 2019. Accessed 25-May-2023.

Bo Zhao. *Web Scraping.* In: 2017. 1-3. ISBN 978-3-319-32001-4.

Petr Zápotocký. *Stemmer pro češtinu.* 2012.
http://hdl.handle.net/11025/5537.

Appendix A
Glossary

AI	Artificial Intelligence
ANN	Artificial Neural Networks
BOW	Bag of Words
BS4	Beautiful Soup 4
CNN	Convolutional Neural Networks
CSS	Cascading Style Sheets
DF	Document Frequency
DHMTL	Dynamic HTML
DOM	Document Object Model
HTML	HyperText Markup Language
IDF	Inverse Document Frequency
JS	JavaScript
LDA	Linear Discriminant Analysis, in other sources it can aslo refer to Latent Dirichlet Allocation
LDiA	Latent Dirichlet Allocation
LSA	Latent Semantic Analysis
LSTM	Long Short Term Memory
ML	Machine Learning
NLP	Natural Language Processing
NLTK	Natural Language ToolKit
NN	Neural Networks
POS	Part of Speech
RNN	Recurrent Neural Networks
TF	Term Frequency
TF-IDF	Term Frequency - Inverse Document Frequency
VADER	Valence Aware Dictionary and sEntiment Reasoner

Appendix B
Tables of Word Similarities with Different Word2vec Representations

dobrý 1.0	skvělý 0.8483	špatný 0.8408	výborný 0.7885	hezký 0.7772	slušný 0.7766
dobré 1.0	správné 0.7371	rozumné 0.7324	důležité 0.7265	špatné 0.7198	těžké 0.7175
dobrá 1.0	špatná 0.8589	skvělá 0.8388	pěkná 0.8132	zajímavá 0.8022	důležitá 0.7686
hezké 1.0	pěkné 0.88	skvělé 0.8073	příjemné 0.7898	fajn 0.7884	krásné 0.7551
super 1.0	fajn 0.735	parádní 0.7156	perfektní 0.6627	skvělá 0.6542	pěkná 0.652
špatné 1.0	správné 0.7683	dobré 0.7198	rozumné 0.7128	nepříjemné 0.6962	hloupé 0.6932
strašné 1.0	hrozné 0.8282	marné 0.7747	úžasné 0.7713	děsivé 0.768	směšné 0.735
hrozný 1.0	strašný 0.7989	neskutečný 0.7714	neuvěřitelný 0.7423	děs 0.7281	dobrej 0.7266

Table B.1. 5 Most similar words in Word2vec model made from both comments and articles from both `www.seznamzpravy.cz` and `www.idnes.cz` with length 100

dobrý 1.0	můj 0.8185	jasný 0.8142	správný 0.7816	náš 0.7712	jediný 0.7658
dobré 1.0	důležité 0.8164	těžké 0.8093	správné 0.7495	jednoduché 0.7257	složité 0.7236
dobrá 1.0	špatná 0.8377	zajímavá 0.8085	složitá 0.7822	citlivá 0.7702	správná 0.7679
hezké 1.0	zašifrované 0.8099	všanc 0.8056	fajn 0.7775	vzdělanější 0.7707	iluze 0.7641
super 1.0	smutné 0.7979	vázán 0.7939	mrtvý 0.7924	fascinující 0.7767	rozený 0.7721
špatné 1.0	zkrátka 0.7293	nebezpečné 0.7187	správné 0.7145	složité 0.7041	špatně 0.7038
strašné 1.0	neuvěřitelné 0.7916	fajn 0.7683	hezké 0.762	zašifrované 0.7587	přehnané 0.7585
hrozný 1.0	beton 0.8318	Jacinto 0.8194	Zrušili 0.8184	Čaroděj 0.8115	interaktivního 0.8087

Table B.2. 5 Most similar words in Word2vec model made from only articles from only `www.seznamzpravy.cz` with length 200

dobrý 1.0	skvělý 0.8827	můj 0.8259	takový 0.8193	tohle 0.8102	tenhle 0.8089
dobré 1.0	důležité 0.8467	špatné 0.8335	správné 0.8235	těžké 0.8059	složité 0.7914
dobrá 1.0	špatná 0.8771	obrovská 0.8624	taková 0.8594	správná 0.854	důležitá 0.8505
hezké 1.0	fajn 0.8462	příjemné 0.845	krásné 0.8368	pěkné 0.8313	neuvěřitelně 0.8302
super 1.0	fajn 0.8641	hodný 0.8179	úžasné 0.8146	strašné 0.8127	skvělá 0.8046
špatné 1.0	dobré 0.8335	těžké 0.8094	správné 0.7936	takové 0.7868	složité 0.7865
strašné 1.0	pravidlem 0.8561	úžasná 0.8505	šťastné 0.8409	hrozné 0.8388	smutné 0.8377
hrozný 1.0	sebestředný 0.7961	skvostně 0.7843	šílený 0.7819	tvůj 0.7814	bezúdržbový 0.7807

Table B.3. 5 Most similar words in Word2vec model made from only articles from only `www.idnes.cz` with length 200

dobrý 1.0	skvělý 0.8603	špatný 0.8334	můj 0.783	silný 0.7811	tenhle 0.7792
dobré 1.0	důležité 0.7911	správné 0.7763	těžké 0.7592	složité 0.7206	špatné 0.7121
dobrá 1.0	špatná 0.8602	zajímavá 0.8519	skvělá 0.8136	důležitá 0.8118	obrovská 0.8114
hezké 1.0	fajn 0.8564	příjemné 0.841	krásné 0.8038	skvělé 0.7983	úžasné 0.7925
super 1.0	fajn 0.8081	hezké 0.7456	neskutečně 0.7393	hrozné 0.7348	zábava 0.7343
špatné 1.0	nepříjemné 0.7549	správné 0.7527	složité 0.7469	výjimečné 0.7372	nebezpečné 0.7295
strašné 1.0	hrozné 0.7997	smutné 0.7855	náhoda 0.7808	šílené 0.7745	nepochybné 0.772
hrozný 1.0	starostlivý 0.7835	hlupák 0.7809	střízliví 0.7783	blbý 0.7765	vegetariánem 0.7696

Table B.4. 5 Most similar words in Word2vec model made from only articles and both sources, with vector of size 100

Appendix C
Sentiment Lexicon Creation Table

dobrý	1.0	špatný	-1.0
dobrá	1.0	špatně	-1.0
dobré	1.0	špatná	-1.0
dobrými	1.0	špatné	-1.0
dobrého	1.0	špatnými	-1.0
dobrému	1.0	špatnou	-1.0
dobrém	1.0	hrozně	-1.0
dobrým	1.0	hrozné	-1.0
dobrou	1.0	ošklivé	-1.0
skvělý	1.0	špatným	-1.0
úžasný	1.0	příšerný	-1.0
perfektní	1.0	špatných	-1.0
výborný	1.0	mizerný	-1.0
příjemný	1.0	hnusně	-1.0
hezký	1.0	hnusné	-1.0
dobří	1.0	odporné	-1.0
chytrý	1.0	odporný	-1.0
úžasná	1.0	odporná	-1.0
krásné	1.0	hnus	-1.0
vynikající	1.0	odpad	-1.0
skvělými	1.0	blb	-1.0
dobře	1.0	idiot	-1.0
skvěle	1.0	zlý	-1.0
pěkně	1.0	zlé	-1.0
krásnou	1.0	zlá	-1.0
super	1.0		

Table C.5. All words that were used as the base sentiment.

Appendix D
Other Figures

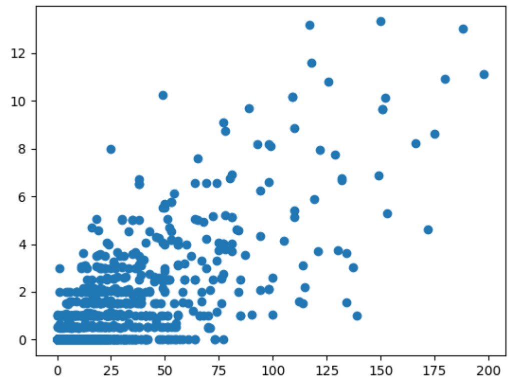

Figure D.1. Comparison of absolute counts of positive sentiment words in Polyglot (x-axis) and our lexicon(y-axis)

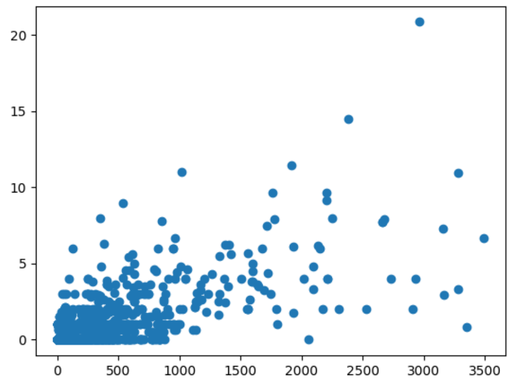

Figure D.2. Comparison of absolute counts of negative sentiment words in Polyglot (x-axis) and our lexicon(y-axis)

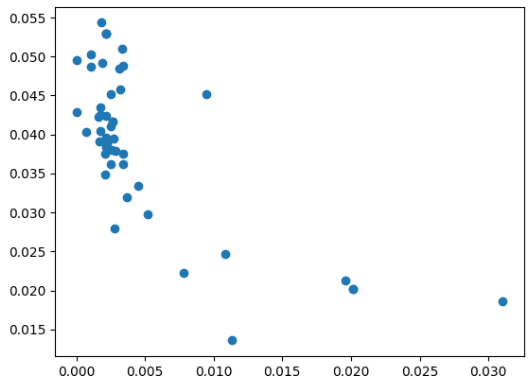

Figure D.3. Ratio of positive words (x-axis) vs ratio of negative word (y-axis), topic "Ukrajina"

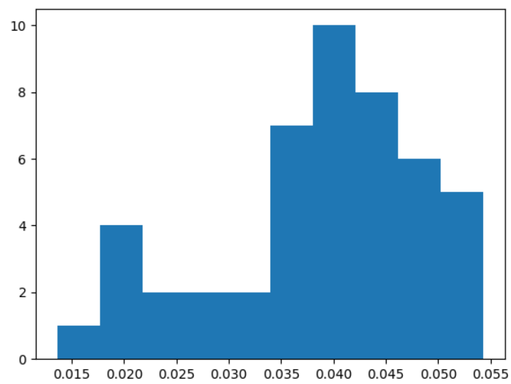

Figure D.4. Histogram of rations of negative words, topic "Ukrajina"

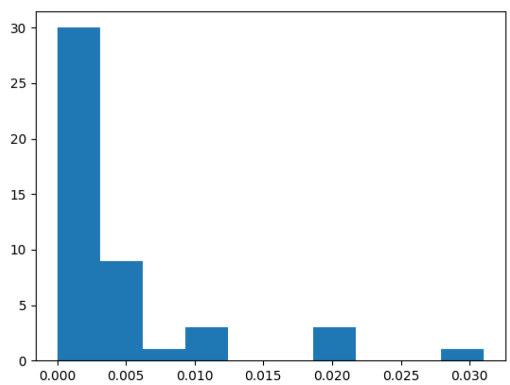

Figure D.5. Histogram of ratios of positive words, topic "Ukrajina"

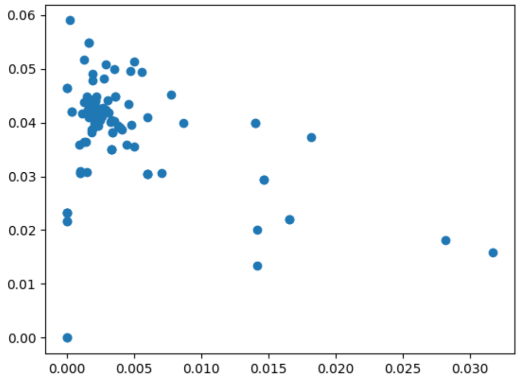

Figure D.6. Ratio of positive words (x-axis) vs ratio of negative word (y-axis), topic "Prezidentské volby"

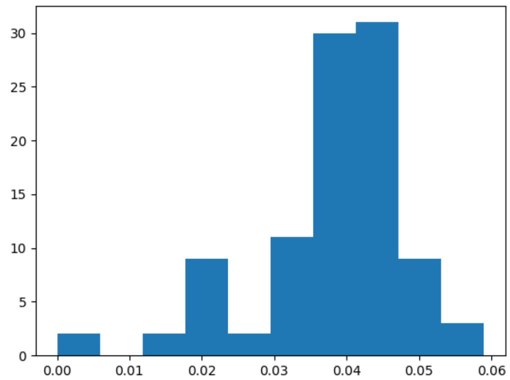

Figure D.7. Histogram of rations of negative words, topic "Prezidentské volby"

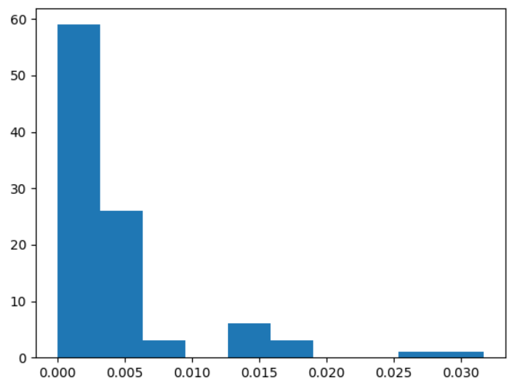

Figure D.8. Histogram of rations of positive words, topic "Prezidentské volby"

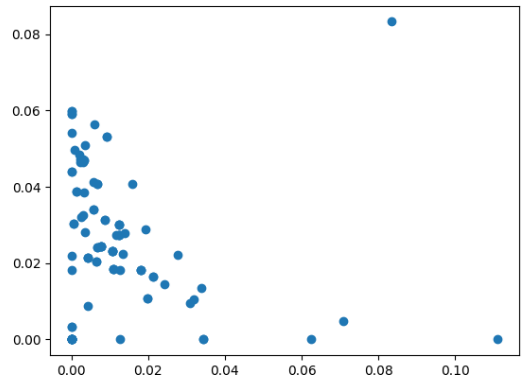

Figure D.9. Ratios of positive words (x-axis) vs ratio of negative word (y-axis), topic “Vesmír”

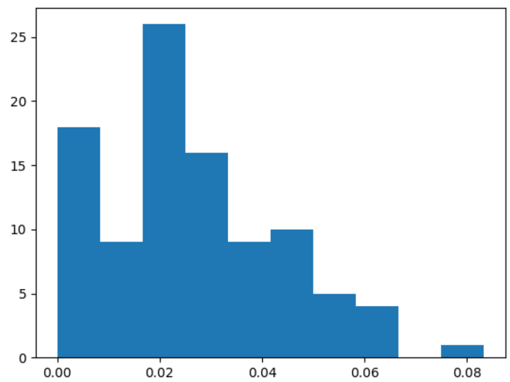

Figure D.10. Histogram of ratios of negative words, topic “Vesmír”

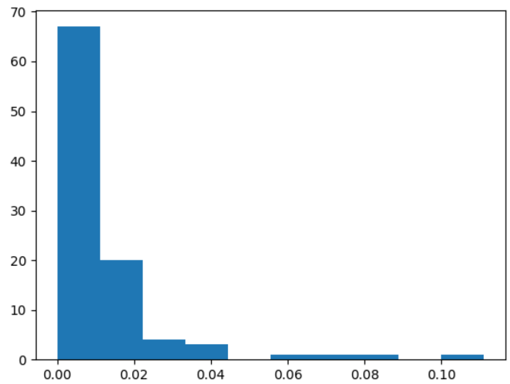

Figure D.11. Histogram of ratios of positive words, topic “Vesmír”

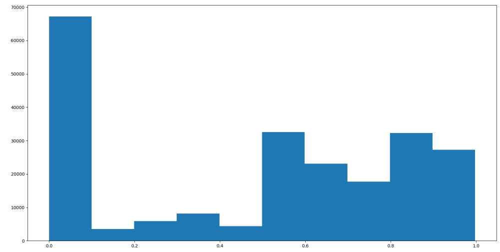

Figure D.12. Hisogram of positive reactions.

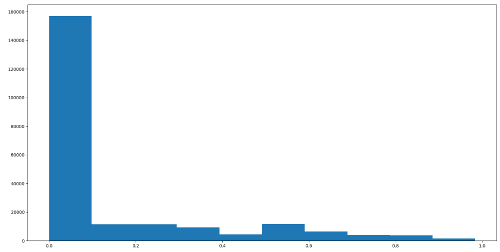

Figure D.13. Hisogram of negative reactions.