Identifying sentiment bearing sentences for reviews in Norwegian

Mateo Caycedo Alvarez

Thesis submitted for the degree of
Master in Informatics: Language and Communication
60 credits

Department of Informatics
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Spring 2019

Identifying sentiment bearing sentences for reviews in Norwegian

Mateo Caycedo Alvarez

Identifying sentiment bearing sentences for reviews in Norwegian

http://www.duo.uio.no/

Printed: Reprosentralen, University of Oslo

Abstract

In this work, we tackled the task of identifying sentiment bearing sentences for product reviews in Norwegian. We have created a set of automatically labeled datasets that classify sentences in terms of how relevant they are to the reviews' overall sentiment and also in terms of their sentiment polarity. We leveraged authors' annotations in the form of positive and negative keyphrases, called pros and cons, to provide distant supervision. Then, we used the created datasets to train a sentence identification system using both feed-forward and convolutional neural network models, and pre-trained word embeddings. We also performed a detailed hyperparameter search for our convolutional architecture. The performance of the models was analyzed with regards to product categories and a thorough manual error analysis was performed on the system's output. Our results demonstrate the usefulness of pros and cons to capture the overall sentiment of a review and our convolutional model outperformed all baselines. Our analysis illustrates how task-specific hyperparameter tuning is beneficial for training high performing models for sentence classification.

Acknowledgements

First, I would like to thank my supervisors Samia Touileb and Erik Velldal for their guidance and feedback. I'm specially grateful to Samia for assisting on the laborious task of manually examining results.

Thanks also to the Språktek squad for the moral support, both virtually and in the flesh. Thanks to the BDF for always being there no matter the distance.

Finally, I want to thank my dear Pecosa for her constant support, encouragement and for keeping me well fed throughout this process.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Product reviews published online have become very important for both service providers and customers. Service providers can use them as a way to obtain direct feedback on their products or services, as well as an indicator of how they will do on the market. Potential customers use reviews to understand the characteristics of products and make a final decision of whether to purchase them or not. Reviews are also helpful to highlight the differences between similar products competing for the same share of the market. However, there is such a large number of reviews that some form of automatic summarization is needed to process the amount of information that is available about products and services.

Traditional text summarization methods applied on product reviews do not yield satisfying results. This is because the summaries tend to be too general and focus on aspects such as topics and categories, which are not central to product reviews (Yu, Huang, Shi, & zhu, 2016). The central theme in reviews is the sentiment towards a product or service, an aspect often missing from text summarization techniques.

In this thesis we seek to develop a sentence identification system that focuses on identifying sentences that represent the main characteristics of a product but also capture the author's sentiment towards the product from reviews in Norwegian. This task has also been called *opinion reason identification* (S.-M. Kim & Hovy, 2006) and it is defined as extracting sentences that answer why the author likes or dislikes the product being reviewed. Our work differs in important ways from previous studies because relevant sentences for reviews are a combination of opinions and facts, and thus identifying them constitutes a distinct problem from subjective opinionated sentence identification or keyphrase extraction.

Identifying sentiment bearing sentences can be interpreted as a sentence-level classification task. Supervised machine learning methods have proven useful at solving multiple natural language processing (NLP) tasks, including sentence classification (Goldberg, 2017). However, in order to utilize supervised learning methods annotated data is necessary.

To date, there is no annotated data for this task in Norwegian. Hence, another aim of this project is to create a dataset with annotated sentences in Norwegian for identifying sentiment bearing sentences in reviews. Labeling each sentence manually is a time-consuming and costly task. Thus, to create this dataset we experiment with leveraging reviews that already contain keyphrases annotated by review authors to label sentences automatically. Our hypothesis is that these keyphrases provide a good summary of the reviews' sentiment. We are the first to use the pros/cons corpus, a corpus containing annotations of positive and negative phrases by professional reviewers. In this thesis we refer to positive keyphrases as pros and negative keyphrases as cons.

We evaluate a variety of approaches to use pros and cons as distant supervision, with the aim of automatically labeling each sentence from the reviews present in the corpus. Furthermore, We build upon the matching technique introduced by S.-M. Kim and Hovy (2006), and seek to extend this framework by exploring alternative ways of matching sentences to pros and cons.

Because keyphrases are central to the creation of our "silver" standard dataset, a dataset with automatically generated labels, we also review previous works about automatic keyphrase extraction. The goal of automatic keyphrase extraction is to identify a set of phrases that are related to the main topics of a given document (Hasan & Ng, 2014).

Many different classification methods have been used for keyphrase extraction and sentence classification tasks. Most examples found in literature for supervised approaches include naive Bayes, decision trees, maximum entropy classifiers, multi-layer perceptrons, and support vector machines (Goldberg, 2017; Hasan & Ng, 2014). In recent years artificial neural networks have shown promising results in a wide range of NLP tasks, including sentiment analysis and sentence classification. These neural networks use word embeddings as input representations because they have been shown to capture rich semantic and conceptual information about words (Goldberg, 2017). We will therefore focus

on neural network architectures, specifically convolutional neural networks using word embeddings as our input representations.

1.1 Overview

The remainder of this thesis is structured as follows:

Chapter 2 provides a description of the corpora used for creating our relevance dataset. We also provide a theoretical overview of methods that exploit keyphrases as part of summarization or sentiment analysis tasks. Special consideration is given to the methods and datasets used in previous works. Moreover we outline the basics of neural networks and their uses for NLP tasks.

Chapter 3 describes in detail the process of creating the "silver" standard datasets. These automatically annotated datasets were used to train our sentence identification system.

Chapter 4 details how we reformulate the sentence identification task as two distinct supervised classification sub-tasks: Relevance classification and Polarity classification. It establishes the baseline models for our classification tasks using feed forward neural networks and the development of a convolutional model, including detailed hyperparameter exploration. This chapter also presents development evaluation results.

Chapter 5 presents the evaluation results of our system on the held-out test sets for both our sub-tasks, relevance classification and polarity classification, and the final sentence identification results. We also provide a manual analysis of our system's final output.

Chapter 6 provides a summary and conclusion of the thesis, as well as possible directions for future work.

Chapter 2

Background

Commonly used text summarization methods produce sub-par results when applied on product reviews. An alternative to provide better summarization of reviews is to use keyphrases (Yu et al., 2016). One of the ways of utilizing keyphrases as a summarization technique is automatic keyphrase extraction.

The main goal of automatic keyphrase extraction is to select a set of phrases that are representative of the main topics of a given document. Keyphrases are useful for a variety of information retrieval (IR) and NLP tasks such as document classification and clustering, opinion mining, web mining and text summarization. Search engines can also use keyphrases to supplement full-text indexing and assist users in formulating queries (Merrouni, Frikh, & Ouhbi, 2016). However, there is no standardized definition of what constitutes a keyphrase and there are varied ways to evaluate if a group of keyphrases are relevant, or not, to a document.

There are two general approaches to identifying keyphrases: extractive and abstractive methods. Extractive methods select relevant phrases present in the text (S.-M. Kim & Hovy, 2006) while abstractive methods can generate phrases that correspond to some semantic properties even though they are not found verbatim in the source text (Branavan, Chen, Eisenstein, & Barzilay, 2009), with many approaches falling somewhere in between the two.

What constitutes a keyphrase also varies in the literature. Some authors consider only pairs of noun-adjectives (Hasan & Ng, 2014), some include also single words (S. N. Kim, Medelyan, Kan, & Baldwin, 2013) and others extend the definition to encompass also complete sentences (Berend, 2011). For the purposes of this paper the term *keyphrase extraction* will refer generally to both extractive and

abstractive methods and the term *keyphrase* refers to anything from a single word to a sentence.

The differences in how authors have tackled keyphrases are often times a product of the type and the amount of data available. Authors with access to metadata like URLs, geographical location, or time of creation, can use that information to find and predict keyphrases (Sullivan, 2008). In other applications the definition of what is a valid keyphrase can be limited to the larger task at hand, for example finding phrases that are relevant to other NLP tasks such as sentiment analysis (Liu & Seneff, 2009).

Sentiment analysis, the task of identifying the subjective attitude or sentiment of the author, has been used in conjunction with keyphrases to analyze large amounts of data, particularly in relation to online content analysis (Berend, 2011). Identifying sentiment bearing sentences in a text can be a powerful tool to perform sentiment analysis because it restricts the set of sentences to analyze, namely only those that might capture the essence of the topics in question. When dealing with large amount of user-generated data, like on-line reviews, combining both relevant sentence extraction and sentiment analysis can provide a good overview of the general opinion about a particular product, as well as the product's main characteristics.

For this project the goal is to exploit the information contained in keyphrases to identify sentiment bearing sentences for reviews in Norwegian. To do so we reviewed different techniques and systems that have attempted to solve a similar problem and we identified the aspects that are relevant to the structure and content of the corpora we have available. The ideal approach would be one that takes advantage of the corpora's properties, and is able to produce a set of results that can be useful as a summarization of the reviews' content and polarity.

In this chapter we will describe the pros/cons and the NoReC corpus in Section 2.1. We will discus related work in Section 2.2. Special attention will be given to the corpora used and the classification methods. Section 2.2.7 describes different challenges present in evaluating keyphrase extraction and sentence identification systems, followed by a summary of the chapter in Section 2.3.

2.1 The corpus

The pros/cons corpus consists of reviews from DinSide.no, a website that provides information, advice and reviews on different products and services such as vehicles, electronics, and other categories. These reviews are written by professional reviewers. Additionally the reviews have a "thumbs up" section with words, sentences or phrases that the reviewer identified as positive aspects, and a "thumbs down" section that describes the negative aspects of the product or service in question, as seen in Figure 2.1. I will refer to these collectively as the "pros/cons section" henceforth.

Figure 2.1: Example of a "pros/cons section" from DinSide.no, including the review's score 4 out 6. Known in Norwegian as "terningkast". The phrases under the green "thumbs up" icon are the review's pros and the ones under the red "thumbs down" icon are the cons.

Some of these reviews have a score, given by the author, known as "terningkast" (dice roll) that is widely used in Norway to indicate the rating of a review, scoring products and services from 1 to 6, with 6 being the best possible score. Most reviews with a "terningkast" score are also part of the Norwegian Review Corpus (NoReC) (Velldal et al., 2018). Each review is labeled with a score of 1–6, provided by the "dice roll" rating of the original author.

NoReC is distributed using the CoNLL-U format, pre-processed using UDPipe (Straka & Hajic, 2016), along with a rich set of metadata. The corpus was made as a tool for document-level sentiment analysis in Norwegian, being part of the Sentiment Analysis for Norwegian Text Project (SANT). NoReC consists of reviews of literature, movies,

```
# language = nb
# newdoc id = pros-200001
# text = Aktiv støydemping som gjør jobben.
# newpar id = pros-200001-01
# sent_id = pros-200001-01-01
1 Aktiv aktiv ADJ _ Definite=Ind|Degree=Pos|Number=Sing 2 amod _ _
2 støydemping støydemping NOUN _ Definite=Ind|Gender=Fem|Number=Sing 0 root _ _
3 som som PRON _ PronType=Rel 4 nsubj _ _
4 gjør gjøre VERB _ Mood=Ind|Tense=Pres|VerbForm=Fin 2 acl:relcl _ _
5 jobben jobb NOUN _ Definite=Def|Gender=Masc|Number=Sing 4 obj _ SpaceAfter=No
6 . $. PUNCT _ _ 2 punct _ SpaceAfter=No
```

Figure 2.2: Raw CoNLL-U file of the pros section of a review.

video games, restaurants, music and theater, in addition to product reviews across a wide range of categories. The reviews in NoReC do not necessarily have a "pros/cons section".

The pros/cons corpus is also stored in the CoNLL-U format, using the same pre-processing as NoReC. An example of a raw file from pros/cons can be seen in Figure 2.2. Although there is some similarity with NoReC, the pros/cons corpus has each review divided into two files, one consisting of the pros from the "thumbs up" section, and another for the cons from the "thumbs down" section. NoReC, on the other hand, has one file for each review.

The reviews in pros/cons exhibit some degree of structural consistency due to the fact that they are written by professional reviewers. However, the content of the "pros/cons section" is not consistent across reviews, not even in the same category. In some reviews the "pros/cons section" can be a single fully formed sentence (2.1), a list of phrases (2.2), a list of keywords (2.3) or some combination of the above. Additionally some reviewers use phrases present in the body of the review in the "pros/cons section", while others use different words ranging from synonyms and antonyms to completely new phrases.

The following examples were taken from the "thumbs down" section of three different reviews:

(2.1) *Knappene på pekeplaten er ikke gode*
The buttons on the touchpad are not good

(2.2) *Trist utseende, uspennende interiør*
Sad appearence, unexciting interior

(2.3) *Dyr,* *betjening*
Expensive, maintenance

Generally speaking, keyphrases from reviews can be divided in two categories: opinion-bearing expressions and facts (S.-M. Kim & Hovy, 2006). Opinion-bearing expressions say something about whether the author's opinion is positive or negative. Facts are simply information about the product such as screen size or battery time. Depending on the product described some facts can also provide information about the reasons for a particular score being high or low. For example big components could be detrimental to the score of mobile devices but have no impact on a different product. We are interested in both types of keyphrases.

2.2 Related Work

A typical keyphrase extraction system typically has two main steps: generating candidate phrases and selecting or classifying the phrases as relevant or not to the document. For the first step it is common to use heuristics that exploit the structure of the text to generate candidates (Hasan & Ng, 2014). Previous work in this area has been mostly done on academic corpora using keyphrase extraction to generate automatic tags (S. N. Kim, Medelyan, Kan, & Baldwin, 2010; S. N. Kim et al., 2013; Witten, Paynter, Frank, Gutwin, & Nevill-Manning, 1999).

Keyphrase extraction applied to reviews is slightly different from keyphrase extraction as a summarization technique. Traditionally, for summarization tasks, relevant phrases are those which can help differentiate one document from another within a corpus. Having different keyphrases for each document facilitates search and ranking of relevancy (S. N. Kim et al., 2013). In this project we are not concerned with differentiating reviews from one another at a corpus level. Instead we look to identify relevant sentiment bearing sentences at a document level, meaning that having similar sentences for different reviews is not an issue.

While traditional machine learning systems have been successful at extractive summarization, recent developments in the field of neural networks has made abstractive summarization more viable (Meng et al., 2017). For this reason we will first describe the traditional machine learning systems that deal with semi-structured reviews and that

have some connection to opinion mining or sentiment analysis. We will focus on the type of corpus used, the candidate generation step, and the classification step. Finally we will describe neural network architectures as an alternative to other machine learning models. The methods and architectures differ so greatly from other models that they will be covered in detail in their own section.

2.2.1 Used Corpora

Sullivan (2008) used reviews exclusively for GPS devices from the website buzzillions.com, run by PowerReviews. Those reviews are very structured, the pros and cons of the corpus used belong to a limited set of predefined keywords. There are fields containing metadata belonging to each review like "creation date", "location", and "author". Finally each author of the review could choose tags that summarized the type of consumer the author of the review represents. These tags were called *affinities* and could be selected from a predefined set or specified individually by the writer of the review.

Sullivan's (2008) goal was to predict the set of tags that belonged to a review, these tags being pros, cons, and *affinities*. In order to achieve this the system took the most frequently inputted tags as classes and attempted to classify each tag from the available information in the review. Here the potential tags were not extracted from the review text but their presence was predicted using the different features of the review including the previously mentioned metadata.

Other works have also dealt with the labeling of pros and cons in reviews. Berend (2011) specifically defined keyphrases as "phrases that make the opinion-holder feel negative or positive towards a given product". The reviews used were crawled from epinions.com and are free-text annotations that the author characterized as "ill-structured" and "extremely heterogeneous".

Berend (2011) conducted his experiments on two domains from epinions.com, mobile phone and movie reviews. The reviews were made by users, the data was noisy, with inconsistent punctuation and some grammatically incorrect sentences. Additionally the listed pros and cons ranged from full sentences to token-long phrases with different ways to separate between them. Although the reviews in the DinSide.no were written by professionals as opposed to users, the pros and cons in our corpus exhibit similar characteristics to the reviews used by Berend (2011).

For candidate generation, Berend's system extracted phrases of at most 4 tokens beginning with a non-stopword adjective, verb, or noun and ending with a non-stopword noun or adjective (Berend, 2011). The candidates were normalized further by lowercasing them and applying Porter-stemming to each of the lemmatized forms of the tokens. Finally the stems were sorted alphabetically. This process allowed the system to deal with orthographically different phrases in the same way.

Berend (2011) presented an alternative way of normalizing the phrases using the synsets of WordNet (Miller, 1995). Instead of Porter-stemming the tokens, the representation used was the most frequent word form of the synset for that token. The intuition behind this approach was to capture the semantic similarity expressed in the synset.

Other systems have used phrase extraction as an initial step to try and capture more abstract semantic properties in the text (Branavan et al., 2009). This system formed clusters of keyphrases at training time indexed by topic. Each topic mapped to a keyphrase cluster. These topics are what the authors called *semantic properties* and each property indexed a language model. These models were used to predict relevant properties of unannotated documents.

The data set used by Branavan et al. (2009) was downloaded from epinions.com and consisted of reviews of mobile phones and restaurants. The authors found that review authors often omit properties from the list of keyphrases that are present in the text of the review.

Attempts to use keyphrases to perform a more in depth sentiment analysis have also been made by Liu and Seneff (2009). In this case restaurant reviews from citysearch.com which contained pros, cons, and free text annotations were used. Furthermore, they used a hierarchical representation of the surface strings called linguistic frame. These frames encode different layers of semantic dependencies. Only sets of related adverbs, adjectives, and nouns were selected as keyphrase candidates. The fact that a linguistic frame was created for each sentence made it easier for the system to preserve long distance dependencies. The second part of the system, called "paraphrase", generated noun phrases from the sets of related words.

S.-M. Kim and Hovy (2006) designed a system to identify the reasons behind pros and cons in online reviews. Their reviews, much like in DinSide.no, have the pros and cons fields as part of the review. The pros and cons were used to created labeled data to train a maximum entropy

classifier. The labels used were “pro”, “con” and “neither”. The latter used for sentences that were not connected to a particular keyphrase. They implemented an automatic labeling process to generate training data by cross referencing the pros and cons with the body of the review. This process labeled every sentence in their dataset as being related to a “pro”, “con” or “neither”.

S.-M. Kim and Hovy’s (2006) approach is strictly extractive regarding candidate generation because it considers only the sentences in the body of the review. The main objective of their system is to identify the reasons of the overall sentiment of the review, not just any opinionated sentence. This means that there can be opinionated sentences in the text that are not considered relevant because they are not connected to a particular pro or con.

2.2.2 Classification

The task of selecting the right candidate sentences or phrases for a given document is often formulated as a classification problem. Specially with datasets like the ones described in the previous section, tags or labels, such as pros and cons, can be used to perform supervised classification or distant supervision using a part of the corpus as a training set (Hasan & Ng, 2014). However, even though many authors have chosen this method, the kind of features used for classification vary.

The KEA features are often taken as a starting point or to define a baseline (Hasan & Ng, 2014). These features are the TF-IDF score of a phrase and the first occurrence, “calculated as the number of words that precede the phrase’s first appearance, divided by the number of words in the document.” (Witten et al., 1999, p. 8). The KEA features are purely lexical and represent a good starting point for summarization as they attempt to capture what phrases characterize a particular document.

Berend (2011) extended these features by adding phrase length, defined as the number of non-stopword tokens of a phrase candidate. An alternative for these lexical features is the use of n-grams, more specifically unigrams, bigrams, trigrams and 4-grams (S.-M. Kim & Hovy, 2006; Sullivan, 2008).

These sets of lexical features explore two different intuitions about summarization. On the one hand checking for relative positions within the document can be useful for scientific publications which contain an abstract at the beginning which is in itself a summary of the document.

Professional reviews can also have something similar to a conclusion or summary, but the structure is not as rigid or predictable as in academic papers. On the other hand n-grams try to capture those words that occur frequently assuming that those might be representative of the semantics of the document.

Even though short reviews might not contain a summary at document level it is not unreasonable to think that the first and last sentence of a paragraph can act as a summary or present the main ideas of that chunk of text. S.-M. Kim and Hovy (2006) explore this intuition by adding a feature that indicates the first, the second, the last and the second last sentence in a paragraph. Sullivan (2008) does this at sentence level by including the 4-grams that begin and end each sentence.

Part of speech (POS) tags are often included with some small variations. Some include the POS tags of the previous word of an occurrence of a candidate phrase in the text (Sullivan, 2008). Others like Berend (2011) include the POS tag sequence of a phrase candidate preserving the order in which they appear as a phrase. Even systems that do not perform supervised classification attempt to capture the syntactic structure of a phrase combined with the POS tags (Branavan et al., 2009; Liu & Seneff, 2009). This is not surprising, as certain constructions, for example "adjective-noun", seem to be common keyphrases.

Identifying opinion-bearing or sentiment-bearing words are also relevant features (S.-M. Kim & Hovy, 2006). As features they can be as straight-forward as simply including the surface form of opinion-bearing words, or one can include information on whether a particular token is positive, neutral or negative. In order to include sentiment scores some authors use external resources (Berend, 2011; S.-M. Kim & Hovy, 2006) like WordNet (Miller, 1995), Wikipedia, and SentiWordNet (Baccianella, Esuli, & Sebastiani, 2010).

2.2.3 Neural networks

Neural Networks (NNs) are a branch of machine learning that has seen a rise in popularity in recent years. The use of NNs has increased for classification tasks that used to be performed by statistical models, like the ones described in Section 2.2.2 (Goldberg, 2017). Even though the name originally was inspired by the way computation works in the brain, it is more accurate to describe the actual computations in NNs

as “learning of parameterized differentiable mathematical functions.” (Goldberg, 2017, p. 16).

One of the main reasons for the widespread use of NNs is that they are not only able to make predictions based on past observations, like other machine learning models, but they can create better representations of the data. These presentations are, of course, also useful for making correct predictions (Goldberg, 2017). The mathematical notation to describe NNs is based on the work presented by Goldberg (2017). Bold uppercase letters are used to represent matrices and bold lowercase letters to represent vectors. Finally vectors are assumed to be row vectors.

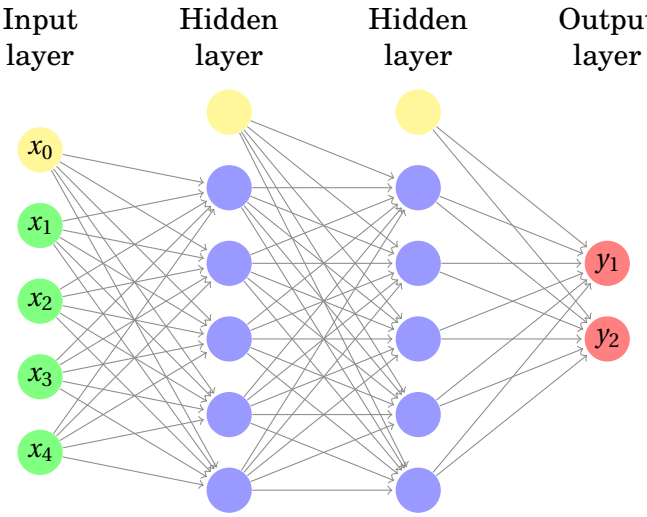


Figure 2.3: Illustration of a feed-forward Neural Network. This is the most basic architecture within NNs. In this example the four green nodes, or neurons, represent the inputs to the network. The blue nodes correspond to the two hidden layers, with five neurons each. The output layer has two output neurons. The yellow nodes at each layer are the bias terms. This kind of architecture with two output neurons could be used for binary classification.

The cornerstone of NNs is the idea of a neuron as the basic computational unit. This idea of an artificial neuron was introduced already in 1943 (McCulloch & Pitts, 1943). An artificial neuron takes an input and performs a transformation of the input, often called an activation function. This activation function is what makes the NNs able to represent complex functions (Goldberg, 2017).

A neural network without an activation is called a perceptron and it is a simple linear model (Goldberg, 2017), as shown in equation 2.4. Where $\mathbf{W}$ is the weight matrix, $\mathbf{x}$ is the input vector and $\mathbf{b}$ is a bias

term. d_{in} denotes the input dimension and d_{out} the output dimension, together these terms inform the size of the weight matrix.

$$\begin{gathered}\text{Perceptron}(\mathbf{x}) = \mathbf{xW} + \mathbf{b} \\ \mathbf{x} \in \mathbb{R}^{d_{in}}, \mathbf{W} \in \mathbb{R}^{d_{in} \times d_{out}}, \mathbf{b} \in \mathbb{R}^{d_{out}}\end{gathered} \tag{2.4}$$

Networks using activation functions to perform non-linear transformations are often called fully-connected feed-forward networks. In Figure 2.3 we can see an example of this kind of architecture. It is fully-connected because each node is connected to every node in the following layer. The feed-forward name is used because the input travels "forward" to the next layer.

A network like the one shown in Figure 2.3 can be mathematically expressed as:

$$\begin{aligned}\text{NN}(x) &= \mathbf{y} \\ \mathbf{h^1} &= g^1(\mathbf{xW^1} + \mathbf{b^1}) \\ \mathbf{h^2} &= g^2(\mathbf{xW^2} + \mathbf{b^2}) \\ \mathbf{y} &= \mathbf{h^2 W^3}\end{aligned} \tag{2.5}$$

In equation 2.5 $\mathbf{h^1}$ represents the first hidden layer, and $\mathbf{h^2}$ the second hidden layer. $\mathbf{y}$ is the output of the network. g represents the activation function. The superscript numbers denote which matrix or vector belongs to each layer. The matrices $\mathbf{W}$ and the bias terms $\mathbf{b}$ are the parameters of the network. Training consist in setting their values such that the network's predictions are correct. Different functions can be used as activation functions to introduce non-linear transformations in a neural network. Figure 2.4 shows the plots of commonly used activation functions and how they relate to one another.

The sigmoid function, described in equation 2.6, was the commonly used function for NNs, because it could model the idea of a neuron firing by transforming each value into the range 0 to 1. A variation of the sigmoid, the hyperbolic tangent or TanH (2.7), maps each value into the range -1 to 1. Figure 2.4 shows how these two functions relate to each other. For both of these equations x is a scalar input and e is the natural logarithm base:

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}} \tag{2.6}$$

$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{2.7}$$

While the sigmoid and TanH functions are still used nowadays in some specific architectures of Recurrent Neural Networks, these functions are considered to be deprecated for most NNs configurations (Goldberg, 2017). The main reason is known as the vanishing gradient problem: the fact that the gradients become so small for very high or low values, that they can stop the network from learning.

The rectified linear unit or ReLU, described in equation 2.8, has become the activation function of choice for most NNs with several layers. It avoids some of the problems of the signmoid and TanH functions and has shown good empirical results (Goldberg, 2017). x is a scalar input and e is the natural logarithm base:

$$\text{ReLU}(x) = \begin{cases} 0 & x < 0 \\ x & \text{otherwise.} \end{cases} \tag{2.8}$$

Some experimentation has also been done with a "smoothed" version of ReLU called the softplus function, shown in equation 2.9. The theoretical advantage this function has over ReLU is that the derivatives for negative values are not hard zeroes. This is due to the fact that the derivative of the softplus is actually the sigmoid function. However ReLU is often reported as having better performance than the softplus function (Zhang & Wallace, 2017). x is a scalar input and e is the natural logarithm base:

$$\text{Softplus}(x) = \ln(1 + e^x) \tag{2.9}$$

For the output layers of the network the softmax function (equation 2.10) is commonly used in classification tasks. The softmax function transforms the values given so they become positive and their sum equals 1. One can interpret these values as a probability distribution between the classes. Note that for equation 2.10 $\mathbf{x}$ is the output vector, e is the natural logarithm base and i and j represent the indices for the values of $\mathbf{x}$.

$$\text{softmax}(\mathbf{x}_i) = \frac{e^{\mathbf{x}_i}}{\sum_j e^{\mathbf{x}_j}} \tag{2.10}$$

Recalling the notation used for a layer of a NN in equation 2.5 we

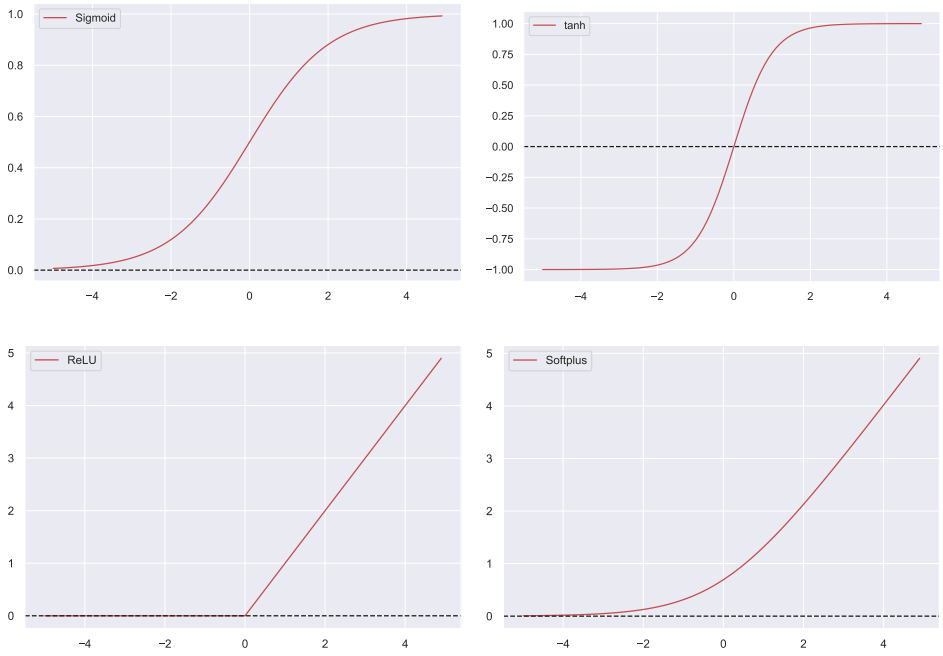


Figure 2.4: Plots of commonly used activation functions for Neural Networks. The top two functions, sigmoid and Tanh have the same shape except that the range of outputs for Tanh is extended to -1. The two bottom functions ReLU and softplus have a similar relationship in that softplus allows for some negative output values. Additionally the sigmoid function is the derivative of the sotfplus function.

can then consider the output $\hat{y}$ for the classification task to be:

$$\begin{aligned} \hat{y} &= \text{softmax}(\mathbf{xW} + \mathbf{b}) \\ \hat{y}_i &= \frac{e^{(\mathbf{xW}+\mathbf{b})_i}}{\sum_j e^{(\mathbf{xW}+\mathbf{b})_j}} \end{aligned} \tag{2.11}$$

2.2.4 Word embeddings

When it comes to applying NNs for NLP tasks a big shift happened in the choice of input representations. For the models described in section 2.2.2 a lot of time and effort was put into determining and choosing which categorical features to use for classification. This task of feature engineering was not only difficult but it also greatly increased the dimensionality of the models when attempting to capture complex linguistic features. An alternative was introduced in the form of dense representations by Firth (1935) and Harris (1954).

In a dense representation each feature is a vector that contains all the information of said feature in the values of the vector. This not only reduces the dimensionality problem mentioned above but it also allows information to be shared between features. Although the use of dense representations did not eliminate the need to think about how information is presented to the models for training, it did allow for a very good method to represent words as vectors to be developed.

Based on the distributional hypothesis of language (Firth, 1935; Harris, 1954), word embedding models try to infer the meaning of words from the contexts in which they are used. While distributional models still result in sparse representations several algorithms have been developed to create dense representations. These dense word vectors are called word embeddings. These word vectors make it easy to compute semantic similarity or to simply use the raw vectors as inputs for NNs. The most prominent algorithms for creating word embeddings are word2vec (Mikolov, Chen, Corrado, & Dean, 2013), fastText (Bojanowski, Grave, Joulin, & Mikolov, 2016), and Glove (Pennington, Socher, & Manning, 2014).

2.2.5 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) were originally developed for image analysis, where they showed great success recognizing objects from a specific category (LeCun & Bengio, 1995). Because of these origins, most of the terminology regarding CNNs reference terms regarding images. This architecture proved to be very powerful at detecting features from an image, understood as a 2-dimensional array of pixels. The different layers would learn to identify different features such as edges or changes in color (LeCun & Bengio, 1995).

The idea of convolution can also be applied to a 1-dimensional input, in other words a sequence. Extracting features from sequences can be very useful for NLP tasks, for example capturing relationships between words, and how these are combined to form phrases or sentences. Because 1-dimensional CNNs are very good at capturing the local ordering of words these are sometimes called n-gram detectors (Goldberg, 2017).

CNNs use two basic operations: convolutions and pooling. A convolution in NLP consists of applying a non-linear function over n words over a sentence. This function is called a filter and transforms a window of n words into scalar values. Multiple filters can be applied

in order to compute important properties of the words in the windows resulting in a vector. Then a pooling operation is used to combine the vectors into a single vector of a specific dimension. The main ways to pool these vectors is to take the maximum or the average value for each dimension over the different windows (Goldberg, 2017). Figure 2.5 illustrates this process.

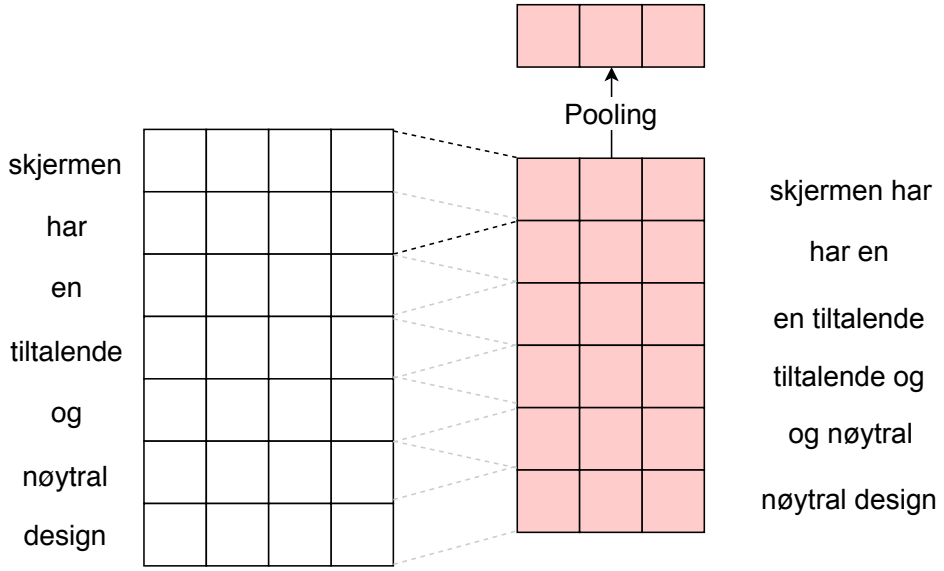


Figure 2.5: Illustration of a narrow convolution in vector-stacking notation. Here the convolution has window size 2 and dimensional output 3. Finally a pooling operation results in a 3 dimensional vector.

A convolution layer for a sequence of words $w_1, ..., w_n$ each with a word embedding w_i of dimension d_{emb} is defined as moving a sliding-window of size k over the sequence and applying this filter to each window in the sequence. The filter is a dot-product with a weight vector $\mathbf{u}$ followed by an activation function g, as shown in equation 2.12. (Goldberg, 2017).

$$p_i = g(\mathbf{x_i} \cdot \mathbf{u}) \tag{2.12}$$

The different filters can be arranged into a matrix $\mathbf{U}$. k denotes the window size and d_{emb} the dimensionality of the word embeddings. Adding the bias vector $\mathbf{b}$ gives us:

$$\begin{aligned} \mathbf{p_i} &= g(\mathbf{x_i} \cdot \mathbf{U} + \mathbf{b}) \\ \mathbf{p_i} \in \mathbb{R}^l, \mathbf{x_i} &\in \mathbb{R}^{k \cdot d_{emb}}, \mathbf{U} \in \mathbb{R}^{k \cdot d_{emb} X l}, \mathbf{b} \in \mathbb{R}^l. \end{aligned} \tag{2.13}$$

The amount of the resulting $\mathbf{p_i}$ vectors is given by the length of the

sequence and the window size. For a sequence of length n and a window size k we get $n-k+1$ $\mathbf{p_i}$ vectors. Alternatively one can pad the sequence with $k-1$ padding-tokens to each side, which gives $n+k+1$ $\mathbf{p_i}$ vectors (Goldberg, 2017). The former is called a narrow convolution while the latter is known as a wide convolution.

As mentioned previously there are multiple pooling strategies. The most common one known as 1-max pooling, picks up the most indicative value from each feature map where $\mathbf{P_{i_j}}$ is the jth component of $\mathbf{p_i}$:

$$c_j = \max_{1<i\leq m} \mathbf{P_{i_j}} \tag{2.14}$$

Alternative pooling strategies attempt to capture positional information that might be lost with 1-max pooling. One such variation is k-max pooling, which retains the best k values sorted by the order in which they appeared in the sequence (Kalchbrenner, Grefenstette, & Blunsom, 2014) .

Another variant is dynamic pooling. With this strategy the vectors $\mathbf{P_{i_j}}$ are split into different groups and pooled separately. The resulting vectors are then concatenated. This is specially useful for document classification as different parts of the document might contain different kinds of useful signals (Goldberg, 2017).

To summarize, CNNs are sensitive to both the identity and order of words within an n-gram regardless of its position in the sentence. This makes them useful for keyphrase extraction as they can identify their presence within a sentence. However, there are other neural architectures that have been used for keyphrase extraction and related tasks.

2.2.6 Neural sequence to sequence models

Recent models have moved away from the extractive methods described in Section 2.2.2, partly due to their inability to cope with absent keyphrases. Cho et al. (2014) and Sutskever, Vinyals, and Le (2014) kick-started this approach with an extension of sequence to sequence or encoder-decoder models to solve machine translation problems. The models use Recurrent Neural Networks (RNN) to encode the input sentence into a vector and then use another RNN to produce a new output sentence.

Models based on neural networks move away from the need to engineer lots of different representations, and in the case of RNN the

output is often at the same level of complexity. For keyphrase extraction this means that the inputs are often sentences and the outputs are sentences or keyphrases. The input representations commonly used for language data are word embeddings, as previously mentioned in Section 2.2.4.

The problem of the so-called "absent keyphrases" is not trivial, some corpora of scientific papers have between 32% and 52% of their keyphrases not matching any contiguous subsequence of the source text. One of the main advantages of sequence models is the ability to generate new sequences of keyphrases or sentences based on textual input that are not limited by the fact that some keyphrases or keywords might be absent in the input text (Meng et al., 2017).

The intuition behind these abstractive methods of summarization is that by being able to generate novel words and phrases not featured in the source text the model produces something closer to a human-written summary (See, Liu, & Manning, 2017). They are, however, still limited by a fixed input and output vocabulary. Some models combine deep supervised learning with reinforcement learning to improve the readability of the summaries created (Paulus, Xiong, & Socher, 2017). In order to cope with representations of out-of-vocabulary words some approaches use a copying mechanism to copy parts of the source text (Meng et al., 2017).

The copying mechanism reintroduces some aspects of extractive summarization by adding a probability for a term to be copied from the source text. The phrases to be copied are weighted by their positional and syntactic characteristics. This mechanism allows the RNN to correctly generate out-of-vocabulary terms. A potential downside is that the model gives priority to the words in the text. Meng et al. (2017) do not see it as a big problem because most keyphrases tend to appear in the source text.

There are, however, some disadvantages about using these particular training setups of RNN-models for product reviews. The previously mentioned models were developed mainly as pure summarization tools for scientific texts. Even in the cases where the models output keyphrases, those phrases are closer to "tags" used to categorize a paper than to the pros or cons found in reviews. The difference is that "tags" are usually more general and describe the field, the theories or the methods used in an academic publication, as seen in this example from Meng et al. (2017):

(2.15) **Title**: Towards content-based relevance ranking for video search
keyphrases: Video search, relevance ranking, video metadata, integrated ranking, video retrieval, video indexing, contentbased ranking, video segmentation.

The category of the reviews we are dealing with is already known and the sentences we are interested in say something more specific about the product in question. In other words, the representations resulting from the aforementioned RNNs seem to be at a level of abstraction that is too high for our task.

2.2.7 Evaluation

Regardless of the feature set or the model used, the problem of evaluating the results of automatic summarization models is often discussed (Hasan & Ng, 2014). Leaving out a portion of the data as a test set is common practice (S.-M. Kim & Hovy, 2006; Sullivan, 2008), if such data is available. However, if the annotations are keyphrases the data is usually noisy as discussed in Section 2.2.1. Additionally keyphrase segmentation and sentence boundaries can differ between reviews or even within the same review. In some cases additional comments that are neither sentences nor keyphrases, such as “none”, can appear in the test set.

An alternative method is to use human annotated data. However, some authors have criticized this approach because of the subjectivity of the task (Berend & Vincze, 2012). Using human annotations is not only expensive and time consuming, but different people can focus on different aspects of the same review and weigh some aspects more heavily than others.

One solution to the subjectivity problem is to measure the agreement score between human annotators using the kappa coefficient (S.-M. Kim & Hovy, 2006) or give precise guidelines as to how they should make the annotations (Berend, 2011). A different approach is to consider the union of all the annotations as the gold standard to cover a wider range of interpretations. It is also noteworthy that the phrases provided by the annotators and by the authors of a review can vary greatly (Berend & Vincze, 2012).

Regardless of how the gold standard is generated there is still uncertainty about automatically evaluating the predicted result with the gold standard. Some claim that doing exact matching between the

expected and predicted results can give misleading results because of small differences due to synonymy. For example, the phrases “tiny keys” and “small keys” would not be matched.

Accepting partial matches or results that are semantically similar might be more representative, but it is not trivial when performing automated evaluations because of ambiguous polar expressions and adjectives. A word like “economical” could show positive polarity in some contexts or be neutral in others. Words such as “like” can be used as adjectives, verbs, etc. depending on the sentence, and thus their sole presence in a keyphrase might not be enough to determine if it captures the meaning of the source text.

Other kinds of noise in the data such as errors in the gold annotations can also lead to overly negative evaluation results (Berend, 2011). Hasan and Ng (2014) estimate that 7-10% of the overall error for keyphrase extraction systems can be attributed to these kind of evaluation errors.

For abstractive methods that output sentences, other considerations such as readability are also important. Sometimes models that get high numerical scores in different metrics produce results that are hard to read (Meng et al., 2017). Extractive methods are usually constrained by the kind of phrases they output. Since the phrases have to be present in the input text, and do not need to be concerned about readability in the same sense.

2.3 Summary

We have reviewed different approaches to summarization and sentiment analysis using keyphrases. We covered methods ranging from traditional feature-based classification models to neural networks using word embeddings. Based on the similarity of the dataset used with the corpus we intend to use, reviews from DinSide.no with annotated keyphrases in the form of pros and cons, the most similar setup to ours is the one presented by S.-M. Kim and Hovy (2006). Their task is also similar to ours because their definition of “reasons” for particular pros or cons is very close to the kind of meaning we intend to capture from the pros/cons corpus.

Nevertheless, using word embeddings and neural networks like the ones described in Section 2.2.6 have produced better results than traditional machine learning methods and avoid some of the pitfalls

resulting from poor feature engineering (Goldberg, 2017). Given the size of our dataset, encoder-decoder methods might not be suitable as they need larger amounts of data to be trained properly. Some tuning of the output might also be needed to keep the output consistent to the kind of sentences and phrases present in the reviews from DinSide.no.

For this project we combined two approaches with a setup similar to S.-M. Kim and Hovy (2006) where we consider sentences in the review text as reasons for the pros and cons. However, we used word embeddings as an input instead of lexical and semantic features, and used a convolutional network as our feature-extractor. This method will still not be able to handle absent keyphrases, but the effects on the overall performance of the model can be mitigated during the candidate generation step by not relying on strict match of the keyphrases but on a degree of token overlap and by using a global list of keyphrases. This process will be covered in Chapter 3. We used a modified version of S.-M. Kim and Hovy's (2006) automatic labeling system to generate our dataset.

Chapter 3

Creating a relevance dataset

In order to build sentence identification system we decided to divide the task into two sub-tasks, **relevance** and **polarity**. The first task is to identify which sentences of a review are relevant to the author's attitude towards the product. The second task is to determine the sentiment polarity of the sentence. Each task was formulated as a supervised classification problem, where the system takes a sentence as its input an predicts a label for each sentence. Unfortunately, we did not have a dataset in Norwegian that had sentences labeled in terms of relevance to the overall sentiment of the review or in terms of whether the sentence was positive or negative. Thus, we needed to create our own datasets from the reviews available from DinSide.no.

In this chapter we will describe the process of creating a labeled dataset for each of our sub-tasks based on NoReC and the pros/cons corpus. A labeled dataset was needed in order to use supervised learning in our classification tasks. Even though the corpus is of relatively small size, we did not have the resources to manually label 127893 sentences. Therefore we needed to find a way to automatically label the sentences for training and also to have a "silver" standard for our development and test sets. A gold standard in machine learning refers to a set of true values that are created using human annotators, a "silver" standard is a similar set except that the values can be automatically generated.

This chapter will outline and explore the different approaches we tried in order to automatically create the labeled dataset, using pros and cons for distant supervision. Section 3.1 explains how we obtained

the keyphrases from the pros/cons corpus to guide our labeling process. Section 3.2 details the different matching strategies we experimented with to label our dataset, including the relevant statistics that informed our design choices. In Section 3.3 we describe how we manually annotated some parts of the dataset to produce a more robust test set. Finally, we include a summary of the chapter in Section 3.4.

3.1 Keyphrases from pros/cons

The keyphrases from the pros/cons corpus contain a lot of useful information, as was described in Section 2.1. However, the actual text of the review was not present in that corpus at the moment of writing this thesis. The texts for most reviews could be found on the NoReC corpus.

Because most of the reviews used in making these datasets could be found in a subset of NoReC, we decided to keep the same split when dividing the dataset into training, development and test sets. In other words, the reviews that belong to the training set in NoReC were also placed in our training set, and the same was done with the development and test sets. Even though we mirror the splitting done in NoReC, our subset contains less product categories because DinSide.no covers less categories than NoReC, which has reviews from different sources. To balance the datasets we tried to keep the same spread of categories throughout the three splits, Figure 3.1 shows the distribution of categories in our dataset. The basic corpus counts are presented in Table 3.1.

	Training	Development	Test
# Documents	2322	254	225
# Sentences	127574	17165	15896
# Tokens	75392	20908	18981
Average # sentences	55	67	71
Average # tokens	957	1162	1255

Table 3.1: Basic corpus counts for each split of the dataset.

In order to see how the pros and cons relate to the review text we needed to cross reference information between these two corpora, namely pros/cons and NoReC. To facilitate working with both datasets we decided to compile a dictionary with some information about the keyphrases and metadata for ease of reference.

This “keyphrase dictionary” was saved in JSON format. Each document had a unique ID, the same ID as the corresponding ID in

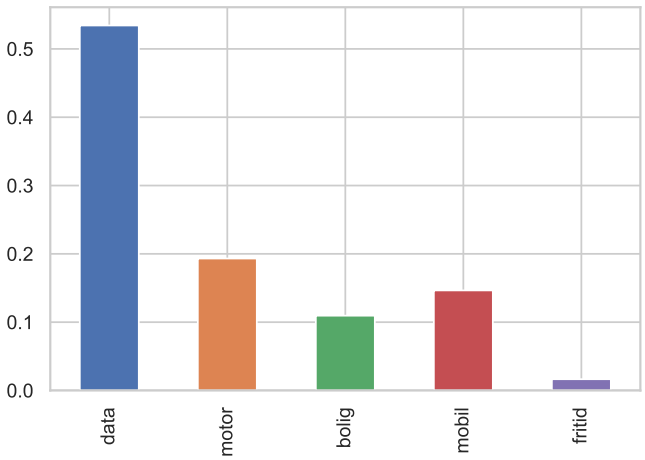


Figure 3.1: Distribution of categories present in all splits of the dataset. The percentages for *autofil* (car lover), *økonomi* (economy) and *reise* (travel) are not shown in the graph because they account for less than 0.01% of the documents in the dataset.

NoReC, or an automatically assigned digit if the NoReC ID was missing. The same ID was used to identify the keyphrases in pros/cons. Other frequently used information such as the pros and cons themselves, their lemmatized forms and the POS-tags sequences were also included. The full attribute list is described in Table 3.2.

Attibute	Description
newdoc id	The file names
nored-id	The id in NoReC
pros	List of the pros from the review
pros_Lemma	List of the lemmatized pros from the review
pros_POS	List of POS-tags sequences of the pros
cons	list of the cons from the review
cons_Lemma	List of the lemmatized cons from the review
cons_POS	List of POS-tags sequences of the cons

Table 3.2: Atributes present in the keyphrase dictionary

The first problem we faced while compiling the phrases from the pros and cons sections was that phrase boundaries were not always clear. Phrase boundaries for the "pros/cons section" were not standardized and varied greatly from one author to another. The most common case was to use commas to separate individual phrases. Example 3.1 shows the "thumbs up" section of a single review:

(3.1) *Kompakt utforming, god bildekvalitet, kontinuerlig fokus*
Compact design, good image quality, continued focus
på video, meget brukervennlig, god og skarp skjerm
on video, very user friendly, good and clear screen

Unfortunately, other authors used different punctuation marks, often inconsistently, such as mixing dashes and commas, using parentheses and backslashes and other combinations. Here are some examples of such cases:

(3.2) *ingen dvd / optisk media*
no dvd / optical drive

(3.3) *morsomt design (for noen)*
fun design (for some)

(3.4) *Tåler (nesten) alt: Vann, støv og støt -*
Withstands (almost) everything: water, dust and shock -
har innebygget lommelykt
has built in flashlight

(3.5) *Stor skjerm; stor mengde tredjeparts programvare*
Big screen; big number of third party software
tilgjengelig - helt gratis
available - completlely free

In other words, we were not able to make rules that could account for all cases. We therefore decided to account for some of the word boundaries and ignore the ones that seemed to lead to ambiguities. After close analysis, we decided to remove parentheses altogether because in the majority of cases they were just adding information to the phrase they appeared in. In some border cases the parentheses were actually used to provide a counter point to a con, or pro, such as:

(3.6) *Vanskelig å løsne batteriet (men kan lades*
Difficult to remove the battery (but can be charged
mens batteriet er i maskinen.)
while the battery is in the machine.)

After removing the parenthesis we split the remaining phrases by comma, colon or semicolon. If none of these were present the phrases were split by dashes, if present. A random selection of a 100 documents were manually inspected to verify the splitting. There were very few cases of erroneous splitting with this approach. The resulting splitting of a pro like example 3.4, results in three different phrases:

(3.7) *Tåler* *nesten* *alt*
Withstands almost everything

(3.8) *Vann*
Water

(3.9) *Støv* *og* *støt* *har* *innebygget* *lommelykt*
Dust and shock has built in flashlight

This case highlights some of the shortcoming of this approach as the word *Vann* (Water) does not really represent an independent pro of this product. The remaining elements *støv og støt* (dust and shock), which are part of the first phrase, are also cut off and carry no useful meaning on their own. The last phrase is also a lot longer because of the inclusion of the actual second pro, namely the reference to the flashlight.

Example 3.9 showcases one of the difficulties of automatically determining phrase boundaries. Fortunately these border cases were very few and most of the splitting worked well. The results in the majority of cases were self-contained descriptions of relevant aspects of a product, such as:

(3.10) *Flott* *skjerm*
Great screen

(3.11) *Hurtigtaster* *til* *de* *fleste* *innstillinger*
Hotkeys for most settings

(3.12) *Filmer* *i* *1080p*
Records in 1080p

(3.13) *God* *bildekvalitet*
Good image quality

(3.14) *Rask* *autofokus*
Quick autofocus

Even with an acceptable splitting strategy, we found that some pros and cons lists contained variations of phrases like *ingen* (none), *ikke noe spesielt* (nothing special), *ingenting negativt* (nothing negative) and *ingenting* (nothing). The issue here is that the authors meant to say that there were no pros or cons, not that these phrases represented qualities or defects of a product.

Given that the occurrence of these phrases was less than 1% of all phrases, we decided to simply remove all occurrences of the

aforementioned words on their own from the dataset. This was done partly to avoid having the same words as negative and positive signals, and mainly to avoid confusion during the labeling process. These words are fairly common in Norwegian and can appear in sentences that are not relevant to the sentiment of the review.

One could argue that the same issue could arise with words meaning *alt* (all/everything), and similar variations. Examining the pros/cons corpus, however, we found no such instances. No author used *alt* by itself in either pros or cons. All the occurrences of *alt* are used to describe other functions or characteristics. For example:

(3.15) *Har alt av funksjoner ellers*
Has all the features otherwise

3.2 Automatic matching

Our strategy for making an automated labeling process was mostly inspired by the work of S.-M. Kim and Hovy (2006). As mentioned in Section 2.2, their goals and dataset were very close to ours. The main idea behind the labeling process is to find sentences that match either a pro or a con in the same review. As an an example, for the pro *plass* (space) a matching sentence would be:

(3.16) *Det er plass som er disse bilenes fremste*
It is space which is these cars' premier
salgsargument, og der har Sharan mye å tilby.
selling point, and there has Sharan lots to offer.

Finding the presence of a keyphrase in a sentence is very straightforward with single-word phrases. For longer phrases S.-M. Kim and Hovy describe the aim as finding "a sentence that covers most of the words in the phrase" (S.-M. Kim & Hovy, 2006, p. 485) . It was not entirely clear how to measure this type of coverage for the phrases in our dataset. We therefore decided to try different overlap strategies to investigate which one would give the best coverage.

Since our main aim was to produce a useful dataset for training and ideally also testing, we looked at two metrics to decide which strategy was the best suited for this task: document coverage and sentence coverage. Document coverage is the percentage of documents for which there was at least one sentence-phrase match. Having a high

document coverage translates into having a larger dataset because more documents, and their corresponding sentences, would be labeled.

Sentence coverage measures the percentage of sentences in one document that had a sentence-phrase match. Having high sentence coverage means that a large number of sentences match a keyphrase in the same document. While having a high sentence coverage means having more training data, the summarization aspect becomes reduced as more sentences are marked as relevant.

Norwegian definite forms for both nouns and adjectives are expressed by adding suffixes. This complicated the matching process when only using the words' surface forms. For example the sentence:

(3.17) *godt hjulpet av den skarpe skjermen*
well helped by the sharp screen

Will not match against the keyphrase:

(3.18) *skarp skjerm*
sharp screen

To overcome this issue we decided to use the lemmatized tokens for both the review text and the keyphrases. For the sake of readability all examples will use the token's surface forms and not the lemmas.

3.2.1 Exact match

The first method we tried, dubbed "exact match", consisted in checking whether the keyphrase, in its entirety, was present in a sentence. This is essentially sub-string matching limited around word boundaries. Phrases that were sentence-like, very long or full sentences had very few matches. This matching strategy had the lowest results as shown in Table 3.3.

The exact match strategy had a document coverage of 23% and a sentence converage of 1.11%. The low scores on both coverage metrics were not surprising, as review authors rarely repeat the phrases used in the main text in the "pros/cons section", as discussed in Section 2.1. Norwegian has the added difficulty of compound words like *skjermstørrelse* (screen size). Meaning that the individual words *skjerm* (screen) and *størrelse* (size) will not get any matches, even though it is very likely that a sentence containing these words is relevant for this particular characteristic.

Strategy	Document coverage %	Sentence coverage %
Exact match	23.0	1.11
Ordered overlap	53.0	2.39
full BoW overlap	54.0	2.61
Partial(2) BoW overlap	80.0	7.80
partial(1) BoW overlap	98.0	52.74
Ordered overlap (G)	98.0	31.30
partial(1) BoW overlap (G)	98.0	66.80
partial(2) BoW overlap (G)	98.0	43.23
full BoW overlap (G)	98.0	61.70

Table 3.3: Metrics for the different overlap strategies. Document coverage = percentage of documents with at least one sentence-phrase match. Sentence coverage = percentage of sentences per document that had sentence-phrase matches. (G) denotes the use of global keyphrases.

3.2.2 Ordered overlap

Moving from the most strict criteria to the more lax ones, our next approach, called "ordered overlap" still required all the words of a keyphrase to be present in a sentence. The difference between exact match and this strategy is that this one allows for other tokens to be in between the words from the keyphrase. All the words from the keyphrase must appear in a sentence in the same order as in the keyphrase to produce a match. The idea behind this strategy was to keep some of the syntactical information in hopes that it will carry some of the same relations between the words in question. It also allows for matching to be done against similar sentences that are slight variations of each other. As an example, with this strategy the keyphrase *sær Bluetooth* (strange Bluetooth) can match against:

(3.19) *Fiio er sær på Bluetooth.*
Fiio is strange when it comes to Bluetooth

While this strategy improved document coverage considerably compared to the exact match strategy, achieving a document coverage of 53%. The sentence coverage was still very low, with only 2.39% of sentences per document receiving a match, see Table 3.3. One reason

appears to be the fact that authors tend to avoid using the exact same sentence several times to make the text more varied. One form of variation is to alter the word order or to use a passive voice.

As an example, this strategy will fail to match the keyphrase 3.20 to the sentence 3.21:

(3.20) *Behagelig* *tastatur*
comfortable keyboard

(3.21) *Vi* *opplever* *tastaturet* *som* *behagelig* *å* *skrive* *på*
We experience the keyboard as comfortable to write on

It is clear that the sentence 3.21 is semantically related to keyphrase 3.20. Additionally, this is the kind of sentence that we want our system to identify. Omitting this kind of sentences would lead to many classification errors if we were to train a model using this data. In order to open for these kind of matches we would have to relax our constraint even more, discarding the requirement of having to preserve word order.

3.2.3 Full Bag-of-words overlap

By removing word order as a requirement, the next step was a representation commonly known as a bag-of-words, meaning a set of unordered words. This method of overlap can be thought of as a less strict version of the previous one. Essentially this translates into interpreting the keyphrases and the review sentences as sets and checking if a keyphrase is a subset of a sentence. If a keyphrase is a set $K = \{word1, word2, ..., wordn\}$ and a sentence a set $S = \{word1, word2, ..., wordn\}$ then a sentence is labeled as relevant if all elements of K are elements of the S. In other words, the set K is contained inside the set S:

$$K \subset S \tag{3.22}$$

The idea behind this method of overlap is to try and use the semantic information of the keywords to label sentences, foregoing the syntactic information from word order. In example 3.20 one can see how the keyphrase and sentence overlap in terms of words and meaning, but not in word order. In theory this overlap strategy might also generate spurious matches, but given that at this point our sentence coverage was under 3% we decided that the risk was worth taking in order to increase the number of matches.

The difference between this strategy and the ordered overlap strategy was small in both document and sentence coverage, as shown in Table 3.3. The most concerning aspect was that the document coverage was still just 54%, meaning that the remaining 46% of documents were effectively discarded during the labeling process. We therefore had to consider even less restrictive matching strategies.

3.2.4 Partial Bag-of-words overlap

This strategy is very similar to the previous one. It checks only if a predefined amount of words from the keyphrase set is present in the sentence set. The first variant we tried was called partial(2), meaning that only 2 of the words from the keyphrase had to be present in the sentence. A sentence was labeled as relevant if:

$$word1, word2 \in Keyphrase \wedge word1, word2 \in Sentence \tag{3.23}$$

In the case of keyphrases containing only one word a sentence was also labeled relevant if that word was present in the sentence. This first variant, using minimum two words for matching, increased the document coverage to 80% and the sentence coverage to 7.8%. A document coverage of 80% is acceptable because we can use 80% of the documents in the corpus. The document coverage of this strategy was acceptable because it means that on average a review would have 7.8% of it's sentences as relevant sentences. For our reviews this means that each review would have between 4 and 5 relevant sentences.

To get the document coverage closer to 100% we tried a second variant that labeled sentences only if one word from the keyphrases was present. This was called partial(1). This variant did increase both document and sentence coverage to 98% and 52.74% respectively, as shown in Table 3.3. Seeing as one of the aims with relevant sentences was to provide a degree of summarization of the review's sentiment, the sentence coverage of this particular strategy was pushing the upper boundary for this metric. After a close analysis, we deemed that labeling more than half of the sentences in a review as relevant was excessive.

Additionally this last strategy had lost part of the meaningful connection between the keyphrases and the text that we were trying to capture. For example, this strategy matched pro 3.24 with sentence 3.25 because of the word *mye* (a lot/much), but these two sentences are not semantically related.

(3.24) *mye for pengene*
a lot for the money

(3.25) *Med så mye plass er det ekstremt viktig å finne*
With so much space it is extremely important to find
lettfrem til riktig låt.
easily the right song.

We were unable to examine every single match to verify whether the sentences labeled were in fact relevant. Because of this uncertainty and the high sentence coverage of the partial overlap we decided to try a different strategy.

3.2.5 Global keyphrases

The main idea of using keyphrases to determine if a sentence is relevant and sentiment bearing is that keyphrases themselves act as a summary of the review's sentiment. In this particular case they are already divided in pros and cons, indicating a positive or negative sentiment tied to a particular phrase. As described in Section 3.2.2, it was difficult to match certain phrases to sentences due to small lexical or syntactic variations. In an attempt to avoid this problem, while still using a relatively strict strategy for matching, we decided to collect all the keyphrases from all the reviews into two lists, one for pros and one for cons.

We called these *global lists* because they are a compilation of all the pros and cons found in the corpus. We investigated the results of applying the matching strategies described previously in sections 3.2.1, 3.2.2, 3.2.3 and 3.2.4 using the global lists of pros and cons, instead of only using the pros and cons from each individual review.

The reason why this would be desirable is that the same characteristics of a product are mentioned many times within the same category. For example: There are 1773 instances of phrases describing the quality of a screen and 2138 describing the price of a service or product. Also intuitively, positive and negative characteristics should be universal for most items being reviewed. Most pros describe good quality, low price, usefulness or reliability. Conversely, negative qualities are usually about high prices, poor quality and failure to deliver functions or expectations.

One could argue that a more systematic approach could be to group all the different keyphrases in different lists according to product type,

category or some other characteristic. This would serve mainly to avoid making comparisons with keyphrases belonging to a completely different product, like mentioning the lack of USB ports on a bicycle. However, since we were checking to see if the words were present in each individual review, a case where a pro or con was irrelevant would simply fail to match. In practice, grouping in categories would only serve to save a negligible amount of time during the matching process. We did, however, have to tackle a different problem, single-word pros and cons.

Our claim that keyphrases from a different review can be useful to determine relevancy in another review is difficult to defend when dealing with single-word pros and cons. In the corpus we found numerous instances of single-word keyphrases like *Design* (Design), *Støynivå* (Noise level) and *Stor* (Big). These words taken completely out of context, as is the case in our compiled lists of global keyphrases, can describe positive or negative aspects of a product depending of the kind of product. In addition, lone nouns can match against sentences with both positive and negative sentiments.

Let us take *Design* (Design) as an example. This keyphrase is taken from a review in which it is classified as a pro. However it being in the global list means that both of the following sentences would be classified as relevant and positive:

(3.26) *Lyktene* *foran* *og* *bak* *har* *fått* *ny* *design*
The lights in the front and the rear have gotten a new design

(3.27) *Skriveren* *har* *en* *stor* *og* *klumpete* *design*
The printer has a big and bulky design

Example 3.26 is a case where the product's design was considered a pro, but it was considered a con in example 3.27. The same situation arises with vague descriptors such as *Stor* (Big), where it can be a positive attribute, as in something with big storage space or a negative one like in example 3.27. Several such situations can arise and, in order to minimize the chance of miss matching, we decided to simply remove single-word pros and cons from the global lists. After the removal of single-word phrases we ended up with 7516 global pros and 6785 global cons.

Another difficulty, even after eliminating single-word keyphrases was the fact that in some cases a sentence could match both with a pro and a con. The distribution is shown in Table 3.4. This is an unfortunate byproduct found in all matching strategies. To overcome this issue we considered three alternatives: Having a third label for ambiguous sentences, making a duplicate of each ambiguous sentence so one would match with a pro and the other with a con, or to simply eliminate ambiguous sentences from the dataset.

	Percentage
Sentences matched with pros	32.1
Sentences matched with cons	46.1
Sentences matched with both pros and cons	21.8

Table 3.4: Distribution of labeled sentences in the dataset

There were two problems with the first option. We wanted to keep the polarity classification portion as a binary problem. The reasons for this were numerous. Firstly we wanted to capture the sentences behind pros and cons, and our reviews operate only in those two categories. There is not a third keyphrase category we could match these results to. It also makes little sense to present sentences in an ambiguous category as being representative of the sentiment of the review. Additionally we wanted to stay as close as possible to the experimental setup described in chapter 4 which consists of two stages of binary classification.

The second option, having copies of a sentence with both positive and negative labels would solve the problem of having to add a third category and having to remove sentences from the dataset. It is, however, not sound within the framework of supervised learning to have the exact same sample with different labels, specially when we expect a classifier to only assign one class to each instance. These reasons and the fact that only 21.8% of the sentences could have ambiguous labels led us to make the decision of eliminating the ambiguous sentences from the dataset.

The same overlap strategies were used to label sentences, except that this time the global pros and cons lists were used. The percentages of document coverage and sentence coverage can be seen in Table 3.3. Many of the challenges identified with the overlap strategies using local keyphrases are still present, and the risk of matching the wrong sentence does increase with the use of the global lists, but it is difficult to quantify by how much without looking at the whole dataset.

As previously mentioned in chapter 2.2.7, it is very difficult to automatically evaluate if a sentence is relevant for the overall sentiment of the review or not. In order to have a firmer grasp of what the results of the matching strategies were, we decided to manually examine 30 random labeled reviews for each overlap strategy and try to get an idea of the kind of matches they were producing. We looked only at the precision of each strategy because the sentence coverage metric provided a good estimation of the recall. The results can be seen in Table 3.5.

The same 30 documents were used for each strategy. Because we were interested in seeing the kind of labels they would produce only documents that had at least one match with all strategies were included. This means that the sample 30 documents were drawn from a subset corresponding to 23% of the reviews, because only that subset had at least one match with every strategy. Since we lack a gold standard the process to evaluate precision consisted in a human annotator looking through each sentence that was labeled as relevant and determining if it was relevant or not. The same evaluation was done in terms of sentiment polarity.

Note that the results from Table 3.5 would likely vary if a different person performed the same procedure, as human annotators tend to disagree when evaluating opinionated sentences (Berend & Vincze, 2012). Because of resource constraints we only had one annotator available for evaluating the matching results.

The manual analysis of the labeled sentences revealed that all the bag-of-words strategies, both full and partial overlap, created too many errors in sentiment labeling. The main reason why their precision score was so low was due to negative sentences being labeled as pro and vice versa. The less strict nature of these strategies was more prone to errors even though they had overall "good" numbers in our coverage metrics. In an attempt to strike a balance between coverage and precision we decided to use the ordered overlap strategy with the global pros and cons lists.

Ordered overlap coupled with the global keyphrases provided good document coverage and the highest precision of any of the other strategies using the global lists. We also wanted to keep the syntactic information by respecting word order and potentially avoiding some of the pitfalls of the aforementioned bag-of-words strategies. The "exact match" strategy was not considered viable because of the low number

of sentences that were labeled. The results were acceptable enough for the training and development sets but for our test set we wanted to have something more robust.

	Precision*
Exact match:	100.0
Ordered overlap:	93.3
full BoW overlap:	76.6
partial(2) BoW overlap:	56.6
partial(1) BoW overlap:	23.3
Ordered overlap (Global):	80.6
partial(2) BoW overlap (Global):	33.4
partial(1) BoW overlap (Global):	16.6
full BoW overlap (global):	70.8

Table 3.5: Average precision for sentence labeling after examining 30 random documents. *Precision was calculated only for documents that had matches. All the 30 documents had at least 1 match.

3.3 Manual annotation

A looming concern when dealing with an automatically generated dataset was the fact that there might be some special circumstances that can create wrong labels. Even though we manually examined a small portion of the documents to do a quality check, we could not be certain that any other mistakes did not slip through. In order to supplement this "silver" standard we decided to create a small test set that was partially manually annotated for each task.

Similarly to the sampling done to the results of the overlap strategies these small annotated sets were checked only in terms of precision. The annotator checked all the sentences and corrected the labels to eliminate false positives both in terms of relevance and polarity. 76 documents were manually checked this way. These small sets were reserved for testing as it was the closest we could get to a gold standard.

A total of 5468 sentences were manually checked and only 1043 false positives were found. This also gave us an estimate of the rate of false positives present in our dataset as a whole. While we do acknowledge that having 19.07% miss-labeled sentences is far from ideal, any automatic labeling system is likely to contain some errors. After performing a thorough manual check, as described in the previous

section, we deemed the result to be good enough for the scope of this project.

3.4 Summary

In this chapter we have detailed the process of creating automatically annotated datasets to train a sentence identification system. Because we did not have an annotated dataset we tried to exploit the pros and cons that were annotated by review authors to automatically label sentences that summarize the sentiment of a review. A dataset for each of our sub-tasks was created.

We described different automatic matching strategies and presented the relevant statistics that informed our design choices. We introduced global keyphrases, a compiled list of all the pros and cons from all reviews in the corpus, as a means to overcome lexical and syntactic difficulties present in the matching process while retaining semantic relations between phrases.

We also performed a manual exploration of a sample of documents, to analyze the output of the different matching strategies. Additionally we created manually corrected test sets for each task, eliminating false positives resulting from the automatic labeling process in order to have test data that was closer to a gold standard.

We chose the strategy called “ordered overlap” using global keyphrases to automatically label our dataset. Additionally we eliminated single-word keyphrases and “ambiguous” sentences, meaning sentences that matched both pros and cons, from the final dataset. We found that this strategy, with the mentioned additions, provided good enough examples to use for training and development, and that the sentences labeled as relevant were overall a good summary of the sentiment of the reviews.

Chapter 4

Relevance and polarity classification

The end goal of our system is to identify the sentences that summarize the overall sentiment of the review. These sentences are grouped into *pro sentences*, related to pros, and *con sentences*, related to cons. In order to achieve this we have automatically annotated the sentences from the reviews using keyphrases from the "pros/cons section" as distant supervision. Then, we used the datasets from chapter 3 to train two separate sentence classification models, following the framework proposed by S.-M. Kim and Hovy (2006).

The reason why two models are trained is that we divide the task of identifying the positive and negative sentences into two tasks. The first task, **relevance** consists in finding relevant sentences that reflect the overall sentiment of the review. In the second task, **polarity** we classify the remaining relevant sentences into positive and negative sentences. Figure 4.1 shows a diagram of the process. Dividing the overall task into two reduces the complexity of each task, since we are now only dealing with a binary classification problem for each one.

Class label	Description
0	Sentences not related to pros or cons
1	Sentences related to pros
2	Sentences related to cons

Table 4.1: Classes defined for the classification process and their respective labels for classification.

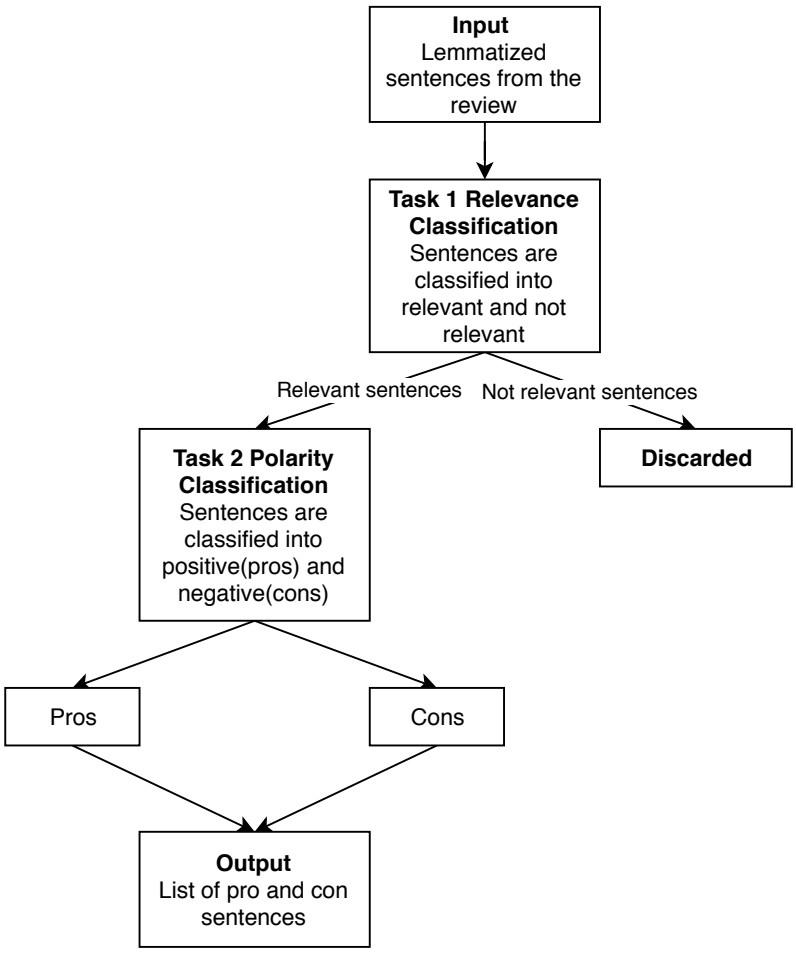


Figure 4.1: Diagram of the complete pipeline for classification of sentences related to pros and cons.

For the first task, in order to classify relevance, the model looks through all the sentences in a review and identifies the sentences that are related to a keyphrase. This can be either a pro or a con but not both. Sentences that represented both pros and cons were removed from the dataset as explained in Section 3.2.5. The sentences labeled as relevant were sent to the second task to be classified in terms of polarity. For the polarity classification task the sentences to be classified were the subset of sentences that were labeled as relevant, meaning related to a pro or a con. Table 4.1 shows the class labels for the three classes we were dealing with. It's important to note that "0" is the majority class and that we were only interested in correctly predicting sentences corresponding to pros or cons.

In this chapter we will experiment with different methods to establish a baseline for each of our classification sub-tasks. Section 4.1 describes our experiments at using a non-machine learning algorithm as a baseline. In Section 4.2 we implement a baseline using a feed-forward neural network and explore different input representations, including word embeddings. Section 4.3 details our experiments with convolutional neural networks. We present the results of our end-to-end experiments in Section 4.4, by end-to-end we mean the results of the pipeline described in Figure 4.1. Section 4.5 presents our methods and results for tuning our convolutional models. Finally, we provide a summary of the chapter in Section 4.6.

4.1 Matching algorithm as a Baseline

Given that we already have automatically labeled the sentences in our corpus to generate the dataset, we believed that a strong baseline for the relevance task could be our own matching algorithm. Our intuition was that we could interpret the matching algorithm, using the ordered overlap strategy described in Section 3.2.2, as a classifier. Since our hypothesis was that the global keyphrases capture the general characteristics of positive and negative sentences we wanted to test if the algorithm could find the right sentences in the test set using only the keyphrases from the training and development sets. The "silver" standard for this baseline was the annotated test set using only local keyphrases.

The reason why the local keyphrases were used for testing this first baseline was that if we allowed the matching algorithm to see all the keyphrases then the baseline and the labeling procedure would be identical, thus making this a meaningless baseline. The other difficulty that arose with this potential baseline was the fact that we could not directly compare the results of any other classifiers because the test sets would be different.

Our dataset for the relevance task has a majority class. As covered in chapter 3 each document possesses in average only 31.3% relevant sentences. This proposed baseline using the matching algorithm did not provide any better results than simply assigning "not relevant" to all instances. Assigning the majority class to all instances gives an accuracy of 0.7, the same result as the matching algorithm.

These results discouraged us from using the matching algorithm as

a baseline. We therefore chose a feed-forward network as our starting point. This network was not tuned and used default values for its hyperparameters unless otherwise noted.

4.2 Feed-forward baseline

A feed-forward network with 3 hidden layers was used as a baseline. ReLu activations were used on all layers except on the output layer where a softmax activation with two output nodes was used. Figure 4.2 shows a general diagram of the network. The loss function used was categorical cross-entropy and the optimizer used was Adam. No regularization methods were used in the baseline model.

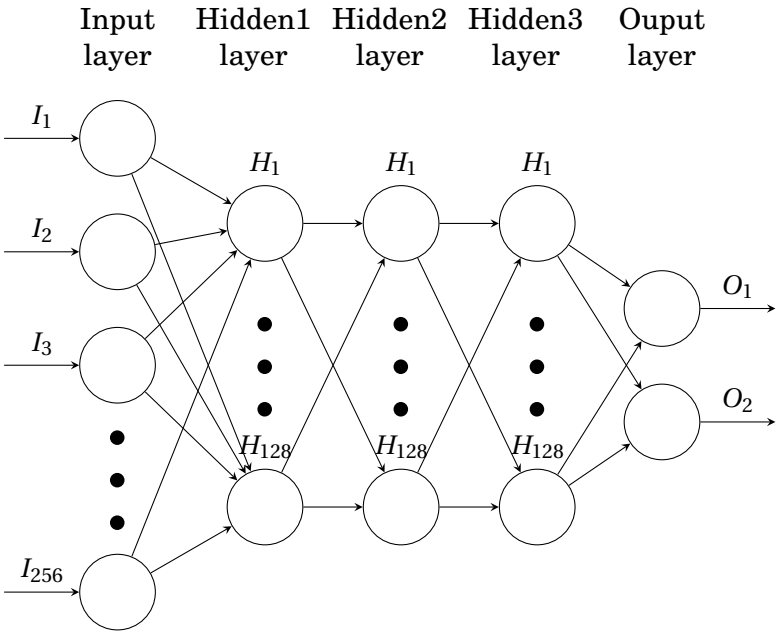


Figure 4.2: General diagram of the baseline model. The input layer has 256 nodes. Each successive hidden layer has 128 nodes each. The output layer has two output nodes, one for each class. Even though the datasets are different for each task both will use the same general architecture shown here.

4.2.1 Implementation details

We used Keras to build our baseline model (Chollet et al., 2015). Keras is a high-level API for running neural networks in Python. While Keras supports different kinds of backends, we used the TensorFlow backend to build the computational graph that represents the network. The

library Gensim was used to handle word embeddings (Řehůřek & Sojka, 2010). One of the main advantages of using Gensim and Keras together is that it is very easy to obtain an embedding layer to use with Keras from pre-trained word embeddings. This embedding layer works as a lookup matrix to find the corresponding vector for each word. All of the activation functions, loss functions and optimization algorithms used in our models are based on Keras' implementations.

The Keras API also provides as set of functions that can be used to access statistics and internal states of the model during training, the functions are collectivelly reffered to as callbacks. We used three of these callbacks to monitor training.

In order to present the best version of each model we used the callback *ModelCheckpoint* to save the best model in each run. The best model in this context in the one that maximizes validation accuracy. Additionally we used *EarlyStopping* to stop the training when the validation accuracy stopped improving. Improvement was defined as an absolute change of at least 0.0001 in validation accuracy. Training was stopped after four epochs without improvement. Finally the callback *ReduceLROnPlateau* was used to dynamically reduce the learning rate when the model began to stagnate. The learning rate was reduced by a factor of 0.1 when the validation loss started to increase.

4.2.2 Accounting for randomness

Feed forward models are not fully deterministic and can provide different results across different runs due to the random initialization of the weight matrices. In order to get an idea of the effect of random initializations we ran the baseline model 20 times with different random seeds and monitored the different results in accuracy, precision, recall and F-score for the task for relevance classification. Table 4.2 shows the maximum, mean and minimum values of each metric alongside the standard deviation. Figure 4.3 shows a plot of the values of the 20 runs.

Metric	Max	Mean	Min	Std
Accuracy	84.10	83.81	83.39	0.0018
Precision	78.49	74.65	70.82	0.0202
Recall	73.38	67.10	59.46	0.0362
F-score	72.43	70.56	67.66	0.0116

Table 4.2: Maximum, mean and minimum values for each of the monitored metrics, including the standard deviation across the 20 runs.

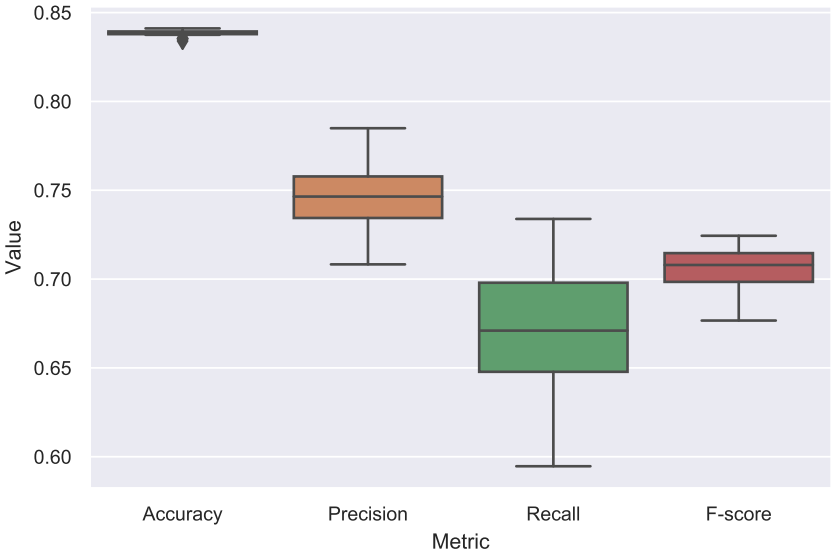


Figure 4.3: Plot of the values of accuracy, precision, recall and F-score after running the baseline model 20 times. Accuracy was the most stable metric across all runs. Recall had the largest variation across runs, still the model showed to be relatively stable.

The results after the repeated runs did show some variation, but the model was relatively stable. The largest fluctuations were observed in terms of recall of all of the monitored metrics. Even though randomness did not seem to play a big role in the model's performance we still wanted to fix the random seed moving forward to preserve repeatibility of the experiments. The chosen seed was the one that showed best performance among the 20 runs.

4.2.3 Pre-processing

All the models presented in this section used the same pre-processing described in chapter 3. All tokens are represented by their lemmatized forms. The token boundaries and their lemmas were taken directly from the CoNLL-U files from the NoReC and pros/cons corpora. All the tokens were kept, including punctuation marks.

4.2.4 Feature representation

We used two methods to represent sentences: Bag-of-words and continuous bag-of-words. We chose these methods of feature representation because they are well established within the field and are simple enough to provide a good starting point.

4.2.5 Bag-of-words

The bag-of-words used is based on the Keras implementation using TF-IDF (Chollet et al., 2015). We chose to use this weighting strategy to help adjust for the fact that some words are more frequent. The weighting strategy is shown in equation 4.1, where $\text{tf}_{t,d}$ is the term frequency of term t in document d belongng to corpus D. The second term, the inverse document frequency is the inverse of the count of distinct documents in the corpus in which the term occurred. The vocabulary size for the bag-of-words was 10000 tokens.

$$\text{TF-IDF}_{t,d} = (\text{tf}_{t,d}) \cdot \log \frac{N}{\text{df}_t} \tag{4.1}$$

4.2.6 Continuous bag-of-words

We wanted to try the same architecture using pre-trained word embeddings. We obtained pre-trained models for Norwegian from the NLPL word embeddings repository[1] (Fares, Kutuzov, Oepen, & Velldal, 2017). In order to isolate the effect of the embeddings during the classification tasks we decided to use models that were similar. A preliminary evaluation showed that our training set contained many out of vocabulary tokens. Table 4.3 shows the number of out of vocabulary tokens found in each model. In order to mitigate the efect of unknown words we chose to work with fastText models because of their ability to infer vectors for unknown tokens (Bojanowski et al., 2016). Due to the fact that we wanted to use lemmas, we also excluded all the models that were not lemmatized.

We chose two models that fulfilled the criteria described above. Both models were trained on four subsets of the Norwegian Newspaper Corpus, Norsk Aviskorpus. The models were, like NoReC and pros/cons, lemmatized using UDpipe (Straka & Hajic, 2016) and the stop words were kept. These models were generated using the fastText Skipgram algorithm with a window size of 5 (Stadsnes, 2018). The only difference

[1] http://vectors.nlpl.eu/repository/

Corpus	# OOV tokens
NAK	49708
NAKs + NoWaC + NBDigital	39491
NAKs + NoWaC	37360

Table 4.3: Number of Out Of Vocabulary(OOV) tokens in the word embeddings models we tested for embeddings in Norwegian. NoWaC = Norwegian Web as Corpus. NAK = Norsk Aviskorpus. NBDigital = National Library of Norway digital corpus. For more details about the word embedding models we refer to (Stadsnes, 2018).

between these models is the vector size, the models have a vector size of 100 and 300 respectively.

To represent a sentence the input vector $\mathbf{x}$ is calculated by the following formula:

$$\mathbf{x} = \frac{1}{|D|} \sum_{i=1}^{|D|} \mathbf{x}^{D_i} \tag{4.2}$$

Sentence D is the sentence containing the word vectors where $\mathbf{D_i}$ is the word vector at position i. All the word vectors are summed together and then divided by the number of vectors in the sentence. This representation is called an *averaged bag-of-words* or *continuous bag-of-words*, because each sentence vector is composed of the averaged sum of the word vectors in the sentence represented as a continuous vector. There are some variations in which the sentence vector is just the sum of all the word vectors. We chose to take the average of the sum to avoid having vectors with large values due to the length of the sentence.

In order to evaluate the influence of word embeddings on the baseline model we trained the baseline model using random word embedding initialization and tuning the weights during training. This model had an extra *Lambda* layer after the embedding layer in which the operation described in formula 4.2 was performed. The reason why this extra layer was needed is because we wanted to keep the mappings to individual word tokens and not to whole sentences. The resulting word vectors had a vector size of 100 and were saved in word2vec format.

4.2.7 Baseline results

The results of the baseline models for task 1, relevance, are presented in Table 4.4. The Bag of Words model outperforms all of the models

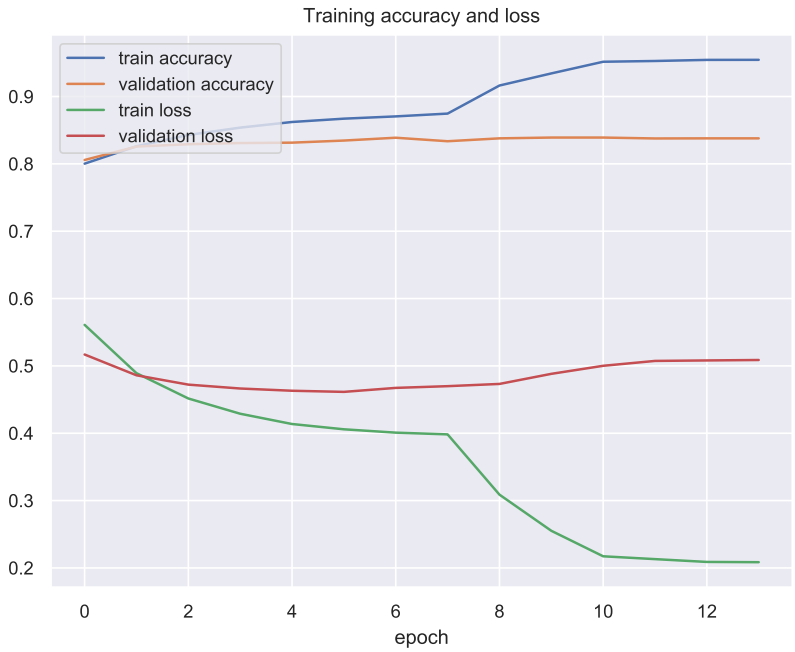
using word embeddings. All of the baselines show an improvement over a majority classifier which has an accuracy of 0.7. A majority classifier for task 1 always outputs "not relevant" as a result.

Model	Emb	VS	Acc	P	R	F1
MC			70.00			
BOW			**84.10**	**69.49**	**63.38**	**66.43**
CBOW	NAK	100	78.25	60.95	30.81	40.92
CBOW	NAK	300	79.56	61.14	44.97	51.82
CBOW	learned	100	80.16	60.73	53.33	56.79

Table 4.4: Baseline results for relevance. Emb = the word embedding model used. VS = the vector size of the embedding model. NAK= Norsk Aviskorpus. MC = a majority classifier that classifier all sentences as "not relevant".

All the models using pre-trained word embeddings had a similar performance. This is not too surprising given that all of them were trained on very similar corpora, and have the same vector size of 100. All of the fastText models we used from the NLPL repository had also the same window size of 5 and were trained using the skip-gram algorithm.

The presence of unknown words does not seem to be a big factor, as random values for embedding initialization yielded similar results to their pre-trained counterparts. Even though these weights were tuned during training the expectation would be that the embeddings trained in much larger corpora would provide a better dense representation of the different words.



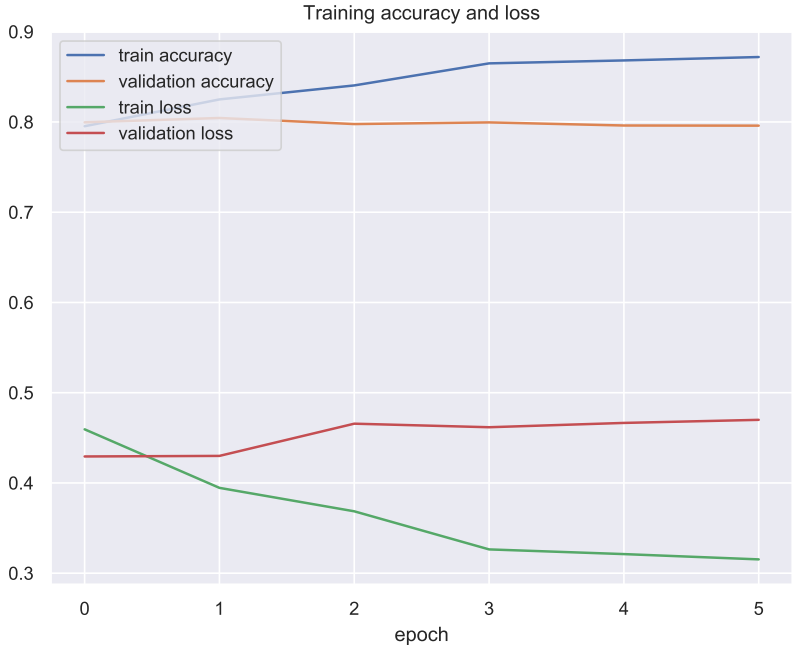


Figure 4.4: Accuracy and loss plot for BOW baseline model (top) and CBOW with learned embeddings (bottom) for task 1 relevance. Both models show similar learning curves.

A common trend in the results of the word embedding models was a low recall score. It seems that the feature representations used were not able to capture meaningful relations between words for the relevance task. Although some information about individual words is preserved, which is a strong signal in regards to sentiment, neither the bag-of-words or the continuous bag-of-words are able to use local syntactic information to form meaningful representations of word grouping or n-grams. This might be one of the reasons why the model struggles to identify relevant sentences.

There was a slight difference with the results for task 2. Table 4.5 shows that even though the scores are generally similar to task 1, recall results are slightly higher. However, this might just be due to the fact that there is not a big majority class as in task 1.

			Pros				Cons		
Model	Emb	VS	Acc	P	R	F1	P	R	F1
MC			56.24						
BOW			85.20	84.65	90.38	87.42	86.06	78.38	82.04
CBOW	NAK	100	75.50	75.02	85.35	79.85	76.38	62.51	68.76
CBOW	NAK	300	79.93	78.59	88.95	83.45	82.36	68.04	74.52
CBOW	learned	100	78.35	76.04	72.36	75.57	78.92	63.40	73.21

Table 4.5: Baseline results for polarity. Emb = the word embedding model used. VS = the vector size of the embedding model. NAK= Norsk Aviskorpus. MC = a majority classifier that labels all sentences as “Cons”.

The baseline models had a very similar performance during training and validation. Most of them follow the same trend as shown in Figure 4.4. The only exception were the embeddings that were tuned during training. These models’ performance peaked very early after 2 epochs and then stopped improving, likely due heavily over-fitting to the training set as shown in Figure 4.5.

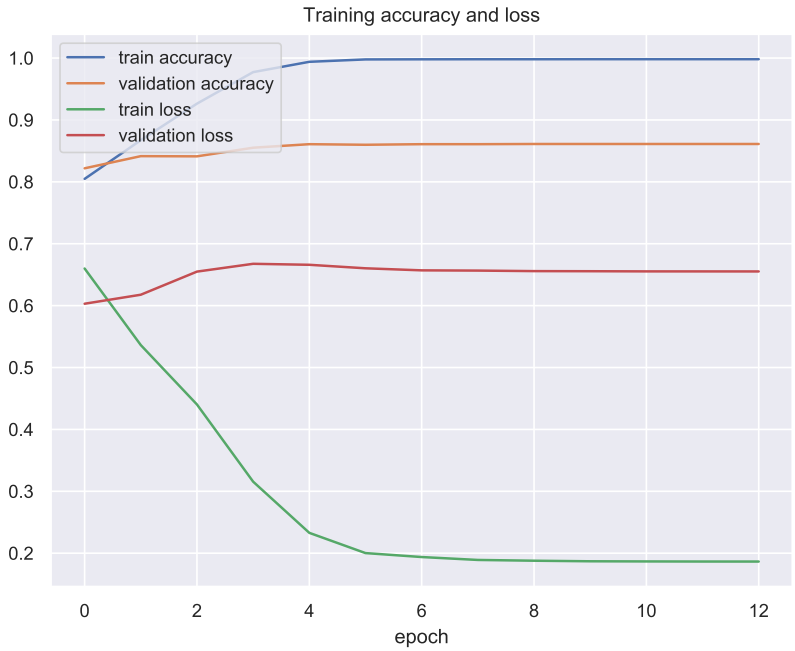


Figure 4.5: Accuracy and loss plot for the baseline model with embeddings learned during training in task 2, polarity. Training accuracy jumps close to 100% after just 4 epochs while validation accuracy stays relatively constant.

4.3 Convolutional neural network models

In order to improve upon the baseline we decided to experiment with a different kind of architecture: Convolutional Neural Netowrks (CNNs). Originally developed for image classification, this architecture has also proven useful to several NLP tasks. One dimensional convolutions are specially useful for analyzing sequences of words. Since we are dealing with sentence classification this kind of approach is very appropriate to both our classification tasks (Goldberg, 2017). For a general description of this architecture see Section 2.2.5.

We used the architectures described by Zhang and Wallace (2017) as a starting point. Their work deals specifically with the problem of sentence classification and they already provide a broad study of the effects different parameters and hyperparameters have on this task.

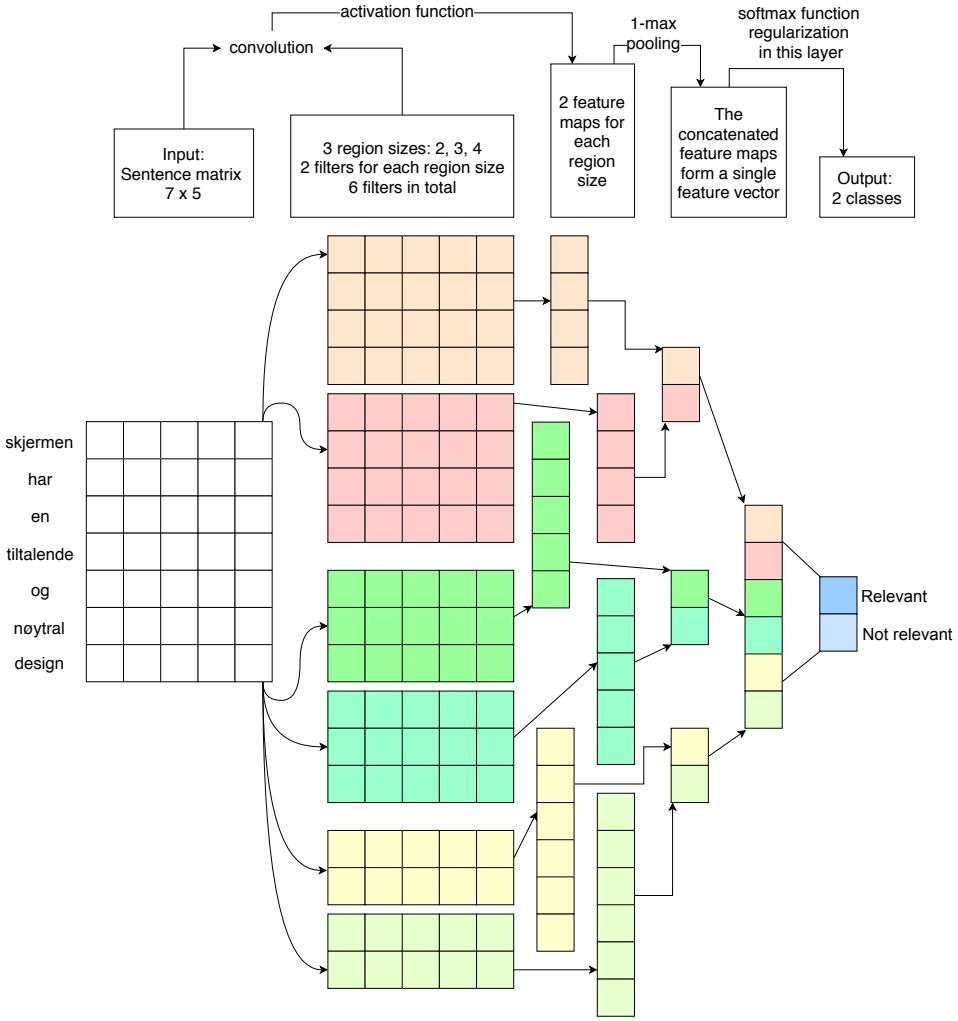


Figure 4.6: Illustration of the baseline architecture suggested by Zhang and Wallace (2017). Three filter region sizes are depicted: 2, 3 and 4. Each region size has 2 filters. Filters perform convolutions on the sentence matrix and generate feature maps of different sizes. 1-max pooling is performed over each map, recording the largest feature from each map. A feature vector with fixed length is generated from all the feature maps. The softmax layer takes this feature vector as its input to classify the sentence. Two output states are depicted because both of our tasks are binary classification problems.

We used the same word embeddings models as described in our baseline as our input features. While it is possible to use several channels with CNNs, such as combining static and non-static embeddings or part of speech tags we focused on a single channel implementation using embeddings for word lemmas. Multiple channels increase memory use and training time significantly, and increasing the channel does not always translate to better performance (Zhang & Wallace, 2017).

The first layer of our CNN is the embedding layer. This layer is followed by three different convolutional layers each with its own region size. Each convolutional layer is connected to a pooling layer and the results of these three layers are concatenated. The last layer is a fully connected layer with a softmax output.

The embedding layer is a matrix of size $V \times d$, where V is the size of the vocabulary and d is the dimensionality of the embeddings. All of the pre-trained embeddings have a vector size d of 100, but the size of the vocabulary V will vary depending on the amount of words that can be inferred from the fastText embeddings. If no vector can be inferred for a specific word a "dummy" character was used to assign a value to that particular token. Gensim provides the functionality of importing the vocabulary from a pre-trained embedding. It is then simply to look up training words to identify the corresponding vector.

Even though it is theoretically possible to have inputs of different lengths for a CNN, having a fixed length helps optimize the training process in batches. If no batches with fixed length are used then each sequence must be processed one by one. Sentences that were shorter than the pre-defined sentence length of 100 tokens were padded with a special token having a value of 0. We did not lose a lot of information by limiting the length to 100 tokens, as only 27 sentences were longer than 100 tokens.

The following layers are three "parallel" convolutions, each with a different region size using ReLU as the activation function. This operations transform a tokenized sentence into a sentence matrix (Zhang & Wallace, 2017). The dimensionality of the sentence matrix is $s \times d$, where s is the length of a sentence and d is the dimensionality of the word vectors. The filters have a filter matrix $\mathbf{W}$ and the matrix contains $h \cdot d$ parameters, where h is the region size. Each convolution is followed by a pooling operation as shown in Figure 4.6.

The final feature vector is created by concatenating the outputs from

each convolution and pooling layer. This final feature vector is fed to a fully connected layer with a softmax activation to generate the final classification. Dropout is applied to the last layer as a regularization strategy by randomly setting values in a weight vector to 0.

4.3.1 Pooling strategies

The most common pooling strategy for text classification is a variant of *max pooling*. Experimentation with different pooling strategies for CNNs report that 1-max pooling consistently have superior performance (Zhang & Wallace, 2017). Preliminary results with local pooling and average pooling with our relevance dataset confirm the findings of Zhang and Wallace (2017). The other pooling strategies showed a clearly lower performance than 1-max pooling.

4.3.2 Filters

CNNs can use multiple filters to learn complementary features. Y. Kim (2014) experimented with three different region sizes (3, 4, 5) with the number of feature maps fixed to 100. Zhang and Wallace (2017) experiment by varying the region size on a single filter and also explore different combinations of number of filters and region sizes. Their results suggest that each dataset has its own optimal filter region size.

Using multiple filters can lead to a moderate increase in performance, however considering multiple filter combinations greatly increases the hyperparameter search space. Zhang and Wallace (2017) limit their search to regions sizes close to the best performing region size in a single filter configuration. The search is further restricted by only looking at adjacent regions sizes, for example (3, 4, 5), except for repeating region sizes, such as (7, 7, 7). However, repeating the same region size is equivalent to simply using a single filter with that given size and then increasing the feature maps accordingly. For this reason we did not experiment with multiple filters with the same region sizes.

4.3.3 Baseline CNN results

In order to determine the effect of the word embedding models and to have a starting point to compare the convolutional architecture to our BOW and CBOW baselines we performed experiments for each classification task using a basic configuration with hyperparameters values taken from the results reported by Y. Kim (2014) and Zhang and

Wallace (2017). The configuration of this network is described in Table 4.6.

Description	Values
input word vectors	Norsk Aviskorpus fastText
word vector dimensionality	100 and 300
filter region size	(2,3,4)
feature maps	100
activation function	ReLU
pooling	1-max pooling
dropout rate	0.5
optimization algorithm	Adam

Table 4.6: The basic configuration for our CNN model. It uses three convolutional layers each with a different region size of 2, 3 and 4. All filters have 100 feature maps and use ReLU activations. 1-max pooling is performed after each filter is applied. A dropout of 0.5 is applied before the softmax layer. Learning was optimized using Adam.

4.3.4 Effect of word embeddings

Tables 4.7 and 4.8 show the classification results of task 1 and task 2 respectively. Both of the convolutional models used were identical except for the word embeddings molds used. The specifics of the word embeddings models are discussed in Section 4.2.4.

Model	Emb	VS	Acc	P	R	F1
BOW			84.10	69.49	63.38	66.43
CNN	NAK	100	86.21	78.77	59.69	67.92
CNN	NAK	300	**87.27**	**80.03**	**63.87**	**71.04**

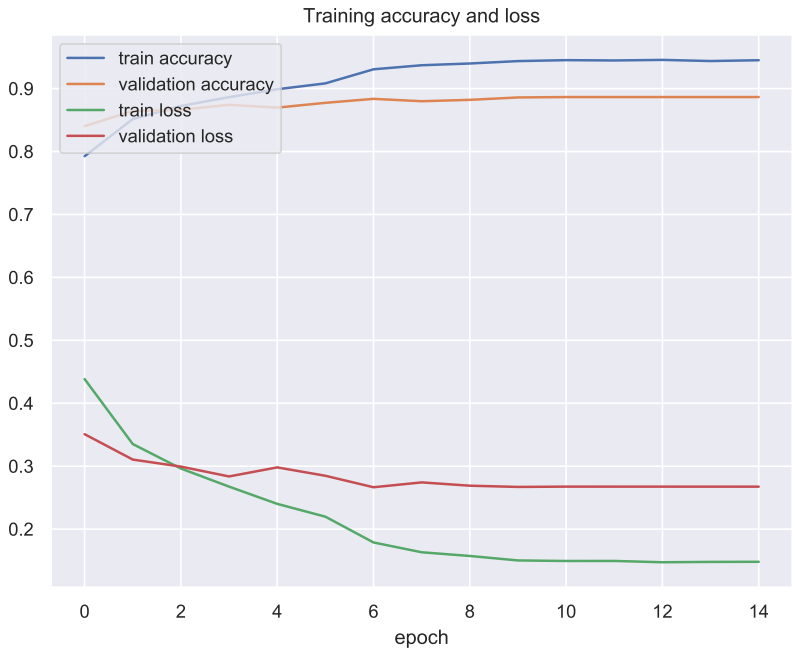
Table 4.7: Baseline results for CNNs task 1. Emb = the word embedding model used. VS = the vector size of the embedding model. NAK= Norsk Aviskorpus. The BOW model is included as a baseline.

These results show that using word embeddings with a larger vector size increases performance across every metric for both tasks. This increase in performance came at a cost in terms of training time, as the models with a higher vector size took roughly double the amount of time to train. The base CNN models also outperform the best performing baseline, the baseline BOW model, in relevance classification. The results are a lot closer for polarity classification, still the CNNs have a slightly better performance in F1 scores.

				Pros			Cons		
Model	Emb	VS	Acc	P	R	F1	P	R	F1
BOW			85.20	84.65	**90.38**	87.42	86.06	78.38	82.04
CNN	NAK	100	85.28	86.59	87.71	87.14	83.50	82.08	82.79
CNN	NAK	300	**86.90**	**88.30**	88.73	**88.51**	**85.04**	**84.49**	**84.76**

Table 4.8: Baseline results CNNs task 2. Emb = the word embedding model used. VS = the vector size of the embedding model. NAK= Norsk Aviskorpus. The BOW model is included as a baseline.

Figure 4.7 shows a comparison of the training curves for the BOW model and the best performing CNN model. While the end performance of both models was close (86.9% for CNN and 85.2% for BOW), we can see from this figure that the gap between the validation and training curves was bigger for the BOW model. Having a tighter gap does not necessarily translate into better performance for a given model, but a large gap could be an indication that the model does not generalize very well.



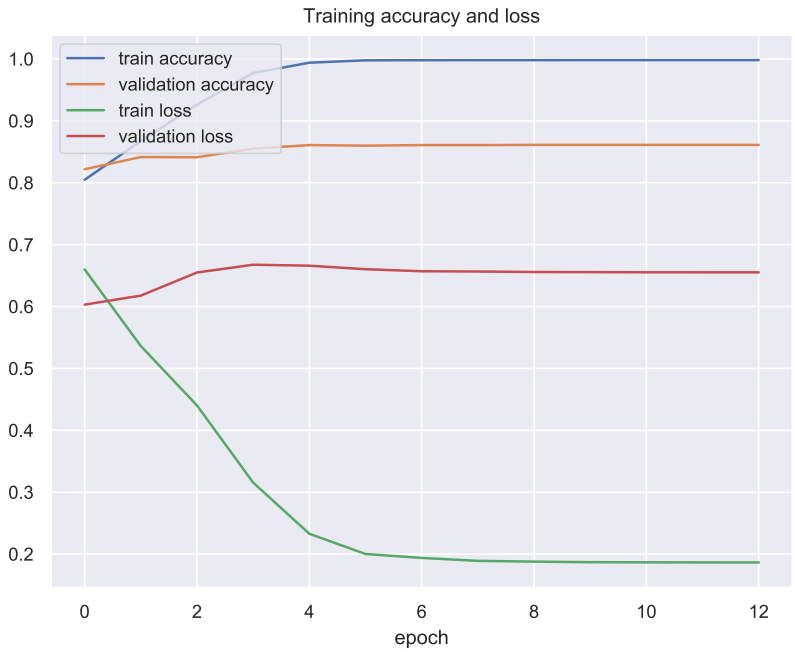


Figure 4.7: Accuracy and loss plot for base CNN with vector size 300 (top) and BOW baseline model (bottom) for task 2.

Both of the CNN and the BOW models had a skew towards the majority class for task 1. This is not surprising because the *relevant* and *not relevant* classes are imbalanced with a $70:30$ ratio in favor of the *not relevant* class. The models also have similar predictions for task 2. Figure 4.8 shows the confusion matrices for task 2. The BOW model tends to identify a few more sentences as pros than the CNN model, but both models show a clear diagonal with a similar distribution for both classes.

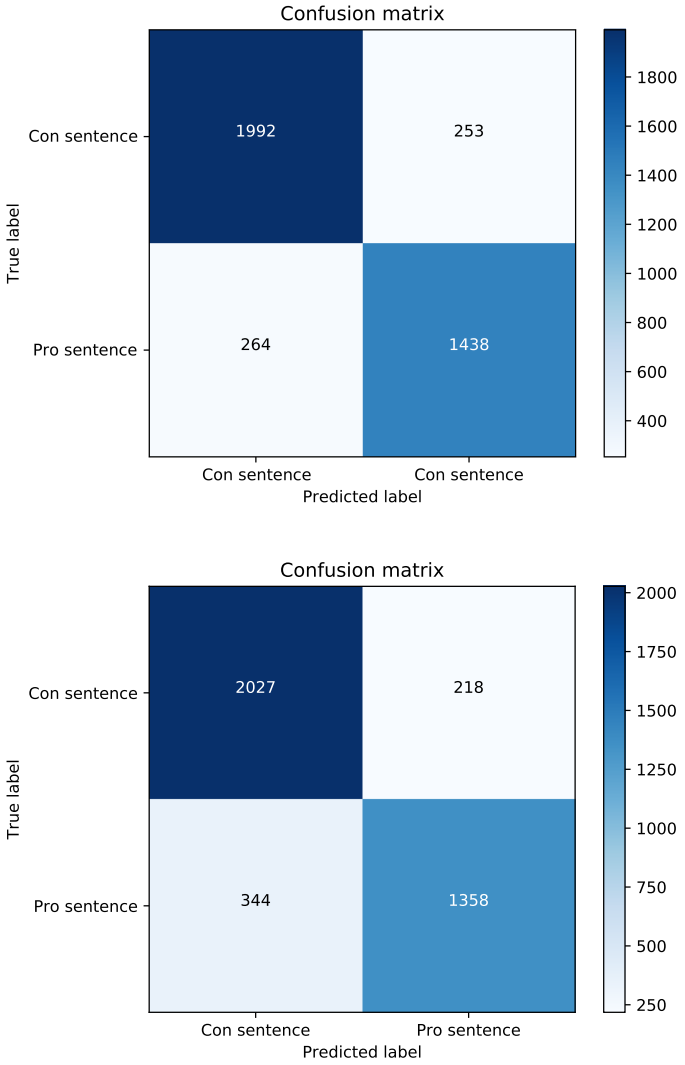


Figure 4.8: Confusion matrices for base CNN with vector size 300 (top) and BOW baseline model (bottom) for task 2.

4.3.5 Performance by product category

Previously in Section 3.3 we mentioned that there are different product categories in the dataset. We observed that the models' performance varied between the different product categories. Figure 4.9 shows both the baseline BOW and CNN models' performance for task 1. The performance of the BOW model is generally better for the larger product categories and slightly worse for the smaller categories, unexpectedly the largest product category *data* (data) was not the best category for either model. The best category for the BOW model was *motor* (motor) with an Accuracy of 87.25%.

The results for the CNN were even more surprising as it's best performing category, *fritid* (leisure), was the smallest one in the dataset. This means the model had very little training data for these kinds of documents and still managed to use the patterns learned from the other categories to perform remarkably well, with an accuracy score of 89.90% in this category.

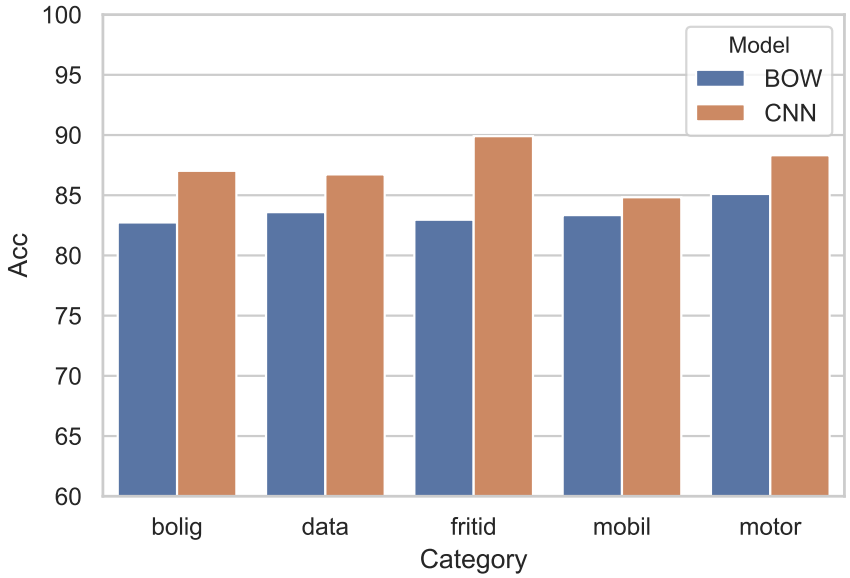


Figure 4.9: Models' performance across categories for task 1 Relevance. BOW performed better for the category *motor* with an Accuracy of 85.10%. CNN had it's top performance in the category *fritid* (leisure) with an accuracy score of 89.90%

For task 2 the BOW model followed a similar trend to the first task with the same category *motor* (motor) showing the best performance with an accuracy score of 87.25%. The biggest difference is that *data*

(data) had the lowest performance in task 2, achieving only 84.24% accuracy. It was more difficult for the model to classify the sentiment of sentences than determining if they were relevant or not to the overall sentiment of the review, and having a larger amount of training data for that category did not appear to have helped performance.

On the other hand the CNN model had a different category as its best performance for task 2. This time *bolig* (residential) performed better achieving an accuracy score of 88.53%. The roles were reversed for the top category in the previous task, while *fritid* (leisure) was the top category for task 1, it was the lowest scoring for task 2. The fact that the smallest category had the lowest score is not surprising, on the contrary it was unexpected that the results from task 1 were different. However, and perhaps more interesting still is the fact that the second lowest performing category was the largest one *data* (data) with an accuracy of 84.88%.

One possible reason for the poor performance of the models in the *data* (data) category might be that the sentences related to pros and cons contain names of devices, for example *USB3.0*. These words can appear in positive and sentences as shown in examples 4.3 and 4.4, where the first sentence is positive and the second sentence negative.

(4.3) *fleksibel med mange ekstra egenskaper for deling av*
flexible with many extra characteristics for sharing
USB3.0 resurser
USB3.0 resources
(positive sentence)

(4.4) *mangler støtte for USB3.0 og gigabit nettverk*
lacks support for USB3.0 and gigabit network
(negative sentence)

In fact, all the other models show a similar difficulty with this particular category. When looking at the embeddings learned by the CBOW model, which learned the embeddings from scratch, we can see that these words have a small cosine distance to sentiment bearing words like *god* (good) or *dårlig* (bad). Both these words have *USB3.0* in their 10 closest neighbors. This is also the case for other technological terms and their cosine distance to sentiment bearing words, being close to words on both sides of the polarity spectrum. It is important to note, however that the mentioned embeddings were trained in the CBOW model and by design that model is not sensitive to the local ordering of

words. This means that although the presence of these words in both contexts might be an indicator of why there is a low performance in that category it might not be the same cause for all the models.

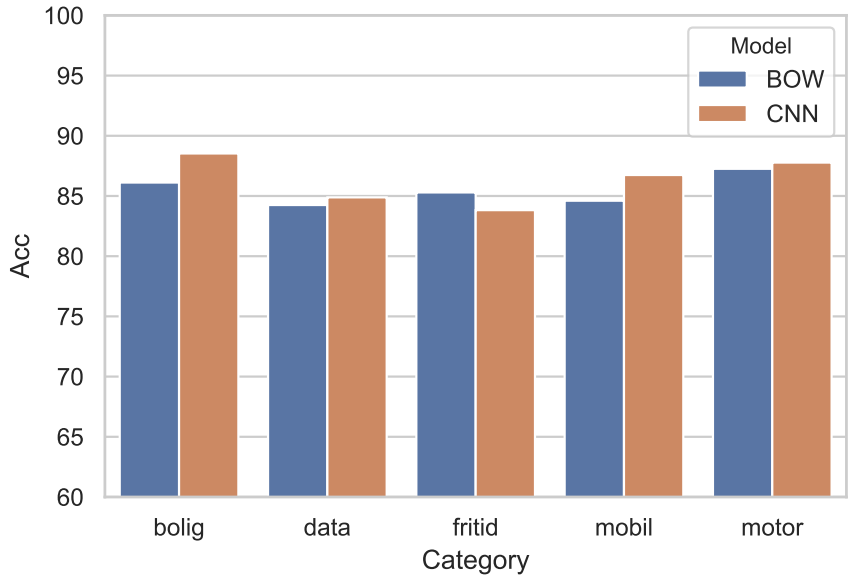


Figure 4.10: Models' performance across categories for task 2 Polarity. BOW performed better for the category *motor* (motor) with an accuracy of 87.25%. CNN had it's top performance in the category *bolig* (residential) with an accuracy score of 88.53%

Another possible explanation for the models' low performance in the *data* (data) category could be the fact that this category is not as homogeneous as the others. While *motor* (motor) and *mobil* (mobile) only deal with the same type of products, cars and mobile phones respectively, *data* (data) includes all kinds of electronic appliances and accessories. The large variety of articles, and the way these are described, might be the why the *data* (data) category is more difficult to classify for our models than the other product categories.

4.4 End to end experiments

In the previous sections we have explored the different models' performance for each task in isolation. As we explained in the beginning of this chapter and illustrated in Figure 4.1, the sentences that are classified as relevant in task 1 are then classified by polarity in task 2. In this section we present the results of the baseline models for both

tasks combined. Because the results of task 2 are dependent on the results of task 1 we called these models "hierarchical models".

Table 4.9 shows the results of the hierarchical models when classifying sentences from the development set. These were the same models trained individually for each task. Because the ranking of performance of all models was identical for both tasks, the same architectures were used for task 1 and then task 2. Combinations of different models did not yield any better results and were therefore omitted. Unsurprisingly all models performed worse than in their individual tasks. This is due to the cascading effect that miss-classification in task 1 carries over to task 2. However the same overall ranking of models was maintained, with the CNN with vector size 300 being the top performer.

				Pros			Cons		
Model	Emb	VS	Acc	P	R	F1	P	R	F1
MC			75.96						
SC			59.81						
BOW			78.14	50.33	49.88	50.10	47.96	59.60	53.15
CBOW	NAK	100	74.75	41.49	18.92	25.99	35.83	28.78	31.92
CBOW	NAK	300	75.24	41.33	24.38	30.67	41.30	40.94	30.67
CNN	NAK	100	80.69	55.31	49.29	52.13	55.07	49.29	52.13
CNN	NAK	300	**81.50**	**57.73**	**56.58**	**57.15**	**56.32**	**60.89**	**58.52**

Table 4.9: End to end results of the hierarchical models. Emb = the word embedding model used. VS = the vector size of the embedding model. NAK= Norsk Aviskorpus. MC= majority classifier that marks all sentences as "not relevant". SC= stratified classifier, makes predictions based on the distribution of labels on the training set.

Neural architectures like the ones we used in the hierarchical models can be very flexible regarding the outputs it can produce for classification tasks. It is trivial to add classes as classification targets and turn a binary classifier into a multiclass classifier. With this in mind we trained two variants of our BOW and CNN models to perform both task 1 and task 2 at the same time. We called these models "joint models". The main difference between the hierarchical models and their joint counterparts is that the joint models were exposed to the whole dataset during training and had to predict all the three classes presented in Table 4.1. For the BOW variants it also meant that a single tokenizer was used for the whole dataset, instead of having one for relevance and one for polarity like the hierarchical BOW.

Adding the extra outputs only added 129 trainable parameters to the BOW model and 300 for the CNN. The training time and resource

usage was virtually the same for training 1 joint model and the 2 models that formed the hierarchical ones. The performance of the joint models is presented in Table 4.10 alongside the best performing hierarchical models. Note that while the joint CNN model had an improvement of 1.28 in accuracy, the F1 scores for both pros and cons were lower than the hierarchical CNN. The joint BOW performed better than its hierarchical counterpart in both accuracy and F1 scores.

				Pros			Cons		
Model	Emb	VS	Acc	P	R	F1	P	R	F1
Hierarchical									
BOW			78.14	50.33	49.88	50.10	47.96	59.60	53.15
CNN	NAK	300	81.50	57.73	**56.58**	**57.15**	56.32	**60.89**	**58.52**
Joint									
BOW			80.96	61.59	46.53	53.01	57.11	53.67	55.34
CNN	NAK	300	**82.78**	**73.33**	38.13	50.17	**71.04**	41.74	52.58

Table 4.10: End to end results of the hierarchical models and joint models. Emb = the word embedding model used. VS = the vector size of the embedding model. NAK= Norsk Aviskorpus.

Figure 4.11 shows the confusion matrices of the hierarchical and joint versions of the CNN models. It appears that even though the hierarchical model had a larger total amount of correctly classified pros and cons, it made a lot more errors distinguishing between positive and negative sentences. The joint model very rarely made errors between positive and negative sentences, but it struggled more distinguishing them from “not relevant” sentences.

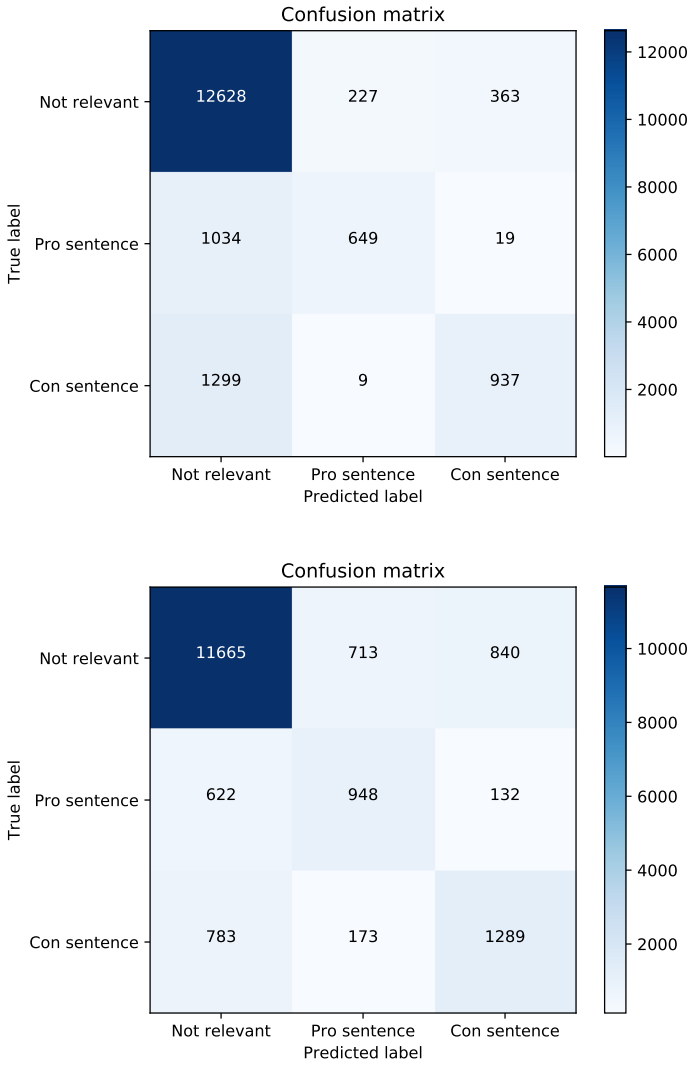


Figure 4.11: Confusion matrices for the joint CNN model (top) and the hierarchical CNN model (bottom).

Having neutral examples, like our “not relevant” sentences, can be important for sentiment classification (Koppel & Schler, 2006). A possible hypothesis of why the joint model is better at distinguishing between sentences in terms of sentiment could be the fact that the model gets presented with neutral examples during training. However it does not seem to identify what distinguishes the pro and con sentences from the rest.

4.5 Hyperparameter tuning

In this section we will describe the process for tuning the hyperparameters of the CNN model and provide an analysis of the results. Table 4.11 shows each parameter with the possible range or categories of values to choose from. However even with the reduced ranges for the chosen hyperparameters the search space becomes too great to perform an exhaustive grid search over all possible combinations.

We followed a similar exploration to the one done by Zhang and Wallace (2017) exploring one hyperparameter at a time while keeping the rest constant. This method has some limitations as it does not account for the interaction between the different hyperparameters. These interactions are often counter-intuitive as simply choosing the best results from each individual run does not guarantee the best performance.

Random hyperparameter search has been reported to provide good results for exploring the hyperparameter space (Bergstra & Bengio, 2012). However this method does not take advantage of the knowledge we have of previous experiments or the dataset. Other sophisticated methods for hyperparameter tuning exist, but these methods require knowledge of which parameters and their values are worth exploring.

The following sections will present the results concentrating on one hyperparameter at a time and keeping the rest of the configuration unchanged. The basic configuration was previously presented in Table 4.6. The performance of models in task 1, relevance, was measured with regards to the F1 score, due to the fact that there was a class imbalance. The performance of models in task 2, polarity, was measured with regards to their accuracy score.

Hyperparameter	Values
Filter region size	1-30
Feature maps	100-300
Pooling	1-max, average
Dropout rate	0-0.9
L2 regularization	0-0.1
Embeddings	Static, Dynamic

Table 4.11: The hyperparameters that constitute our search space.

4.5.1 Filter region size

The filter region size was the most demanding hyperparameter to tune. There are many possible values to choose from and many combinations of possible filters. We performed a line-search over single filter region sizes to find the best performing single region size. After the 'best' region size was identified, we combined multiple filters using region sizes near the single 'best' size. It has been reported that multiple 'good' region sizes tend to outperform using only the the best single size (Zhang & Wallace, 2017).

Table 4.12 shows the results of a single filter with varying region sizes. The ranges for the region sizes were selected from 1 up to the average sentence length in the dataset. Though a convolution with a region size of 1 might seem odd, we wanted to include this filter because CNNs are mostly focused on n-gram features and can lose some important signals coming from single tokens. Filter region size of 3 had the best performance for task 2, while region size 6 had the best performance for task 1. Increasing the size after 10 decreased performance for both tasks, with the exception of filter size 15 that showed good performance for task 1.

Region size	% Acc
1	82.47
2	85.81
3	**86.50**
4	86.27
5	86.32
6	86.07
7	85.58
8	85.58
9	85.41
10	84.85
15	84.61
20	84.24
30	82.52

(a) Polarity

Region size	% F1
1	51.36
2	64.23
3	66.42
4	67.95
5	68.58
6	**69.26**
7	69.04
8	68.71
9	69.61
10	68.83
15	69.15
20	67.48
30	66.02

(b) Relevance

Table 4.12: Effect of a single filter region size with 100 feature maps for each task.

After determining the best single region sizes we explored different combinations of multiple filters. All the possible permutations of the regions sizes from 1 to 5 were tested with 2, 3, 4 and 5 filters. The filters with region sizes over 6 were only tested together with the smaller

filters. The number of feature maps was kept at 100 for all the filters. Additionally we did not test several filters with the same region size, because this is equivalent to simply increasing the number of feature maps. The tuning of the number of feature maps is presented in Section 4.5.2. We explored all combinations for each combination of number of filters and region size with the aforementioned restrictions. The best performing combinations are presented in Table 4.13.

In general we observed that for the polarity task region sizes including 3 and 5 performed better. Combinations including region sizes over 7 performed poorly compared to the lower values. Combinations including region size 1 also tended to under-perform when using one to three filters. However when using four and five filter combinations including region size 1 increased performance. The relevance task showed better results when using larger region sizes, and including sizes 6 and 9 increased performance. Size 1 which helped performance for the previous task was always detrimental to relevance classification.

Multiple region size	% Acc
(3,4,5)	86.79
(2,3,4)	86.90
(4,5,6)	86.98
(3,4,5,6)	87.12
(1,2,3,4)	87.53
(2,3,4,5,6)	87.32
(1,2,3,4,5)	**88.14**
(3,4,5,6,7)	87.79

(a) Polarity

Multiple region size	% Acc
(3,4,5)	71.44
(2,3,4)	71.04
(4,5,6)	71.66
(3,4,5,6)	71.63
(1,2,3,4)	71.14
(2,3,4,5,6)	72.57
(3,4,5,6,7)	72.68
(6,7,9,10,15)	**72.71**

(b) Relevance

Table 4.13: Effect of multiple region sizes. We report only the best combinations for each number of filters.

4.5.2 Number of feature maps

Table 4.14 shows the performance of models in terms of the number of feature maps for each filter, while keeping the rest of the configuration unchanged. While the number of feature maps had little effect on performance, increasing feature maps improved performance for task 1 and decreased performance for task 2.

We also tested different number of feature maps in the range of 50-300 for a single filter with the best performing region size. Zhang and Wallace (2017) reported that the best performance for one of their datasets was to use the best single filter with a high number of feature

maps. This was not the case with either of our datasets. These numbers are not shown as all of the results were worse than using multiple filters.

# Feature maps	% Acc
50	85.79
100	**86.90**
150	86.54
200	86.72
250	86.50
300	86.58

(a) polarity

# Feature maps	% F1
50	69.22
100	71.04
150	70.54
200	71.32
250	71.79
300	**72.13**

(b) relevance

Table 4.14: Effect of the number of feature maps for each task. Larger feature maps improved performance for (b). 100 was the optimal number (a).

4.5.3 Regularization

Regularization was applied to the input of the layer preceding the sotfmax layer as shown in Figure 4.6. We tried both dropout and L2 regularization as these are two common regularization strategies for CNNs (Zhang & Wallace, 2017). Figures 4.12 and 4.13 show the change of each performance metric relative to the dropout rate. The baseline model had a dropout rate of 0.5. There was very little effect by changing the dropout rate and even no dropout performed better than the baseline model. This confirms the findings of Zhang and Wallace (2017). L2 regularization only hurt performance and smaller values performed better as shown in Table 4.15. We also experimented by applying L2 regularization to the convolution layers but this also led to a large decrease in performance.

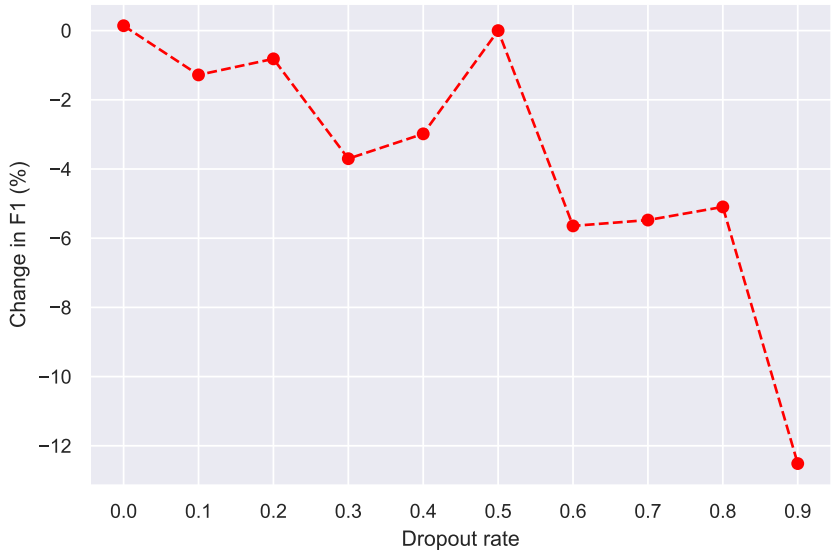


Figure 4.12: Effect of the dropout rate compared to the baseline for task 1. The baseline value was 0.5.

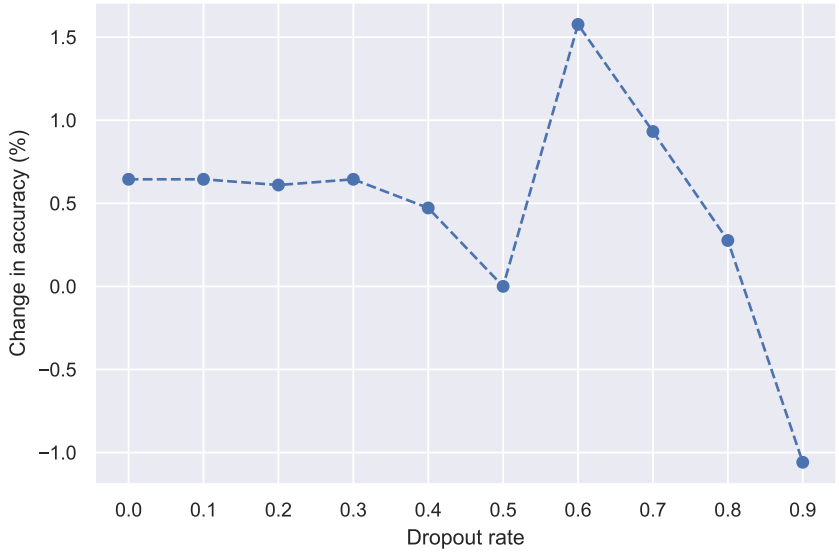


Figure 4.13: Effect of the dropout rate compared to the baseline for task 2. The baseline value was 0.5.

L2	% Acc
0.01	**83.56**
0.02	82.02
0.03	81.05
0.04	80.57
0.05	79.83
0.06	78.31
0.07	79.07
0.08	79.05
0.09	77.38
0.10	73.22

(a) Polarity

L2	% F1
0.01	**67.20**
0.02	65.42
0.03	64.31
0.04	63.76
0.05	62.90
0.06	61.16
0.07	62.03
0.08	62.01
0.09	60.08
0.10	55.30

(b) Relevance

Table 4.15: Effect of L2 regularization for each task.

4.5.4 Static vs Dynamic embeddings

The results of using dynamic embeddings vs static ones are presented in tables 4.16 and 4.17. Dynamic embeddings add a significant amount of parameters to the network, increasing training time considerably. The average time for a static model was 15 minutes, while a dynamic model took several hours. Due to time and resource constraints we did not test every single combination of parameters described in this section with both static and dynamic variants. We are aware that increasing the model's number of parameter has different interactions with other hyperparameters, like regularization strategies. This means that while it is theoretically possible that dynamic embeddings might not be optimal for some configurations, we found that for both our baseline and optimal configurations further tuning the embeddings improved the models' performance. For our datasets the results achieved by dynamic embeddings are better than using their static counterparts. However, the results are comparable to the other hyperparameters tested.

Model	Acc	P	R	F1
CNN Dynamic	**88.21**	**81.77**	**65.69**	**72.85**
CNN Static	87.27	80.03	63.87	71.04

Table 4.16: Effect of dynamic embeddings for task 1, relevance.

		Pros			Cons		
Model	Acc	P	R	F1	P	R	F1
CNN Dynamic	**88.28**	**89.59**	**89.71**	**89.64**	**86.50**	**85.08**	**85.78**
CNN Static	86.90	88.30	88.73	88.51	85.04	84.49	84.76

Table 4.17: Effect of dynamic embeddings for task 2, polarity.

4.5.5 Best configurations

Tables 4.18 and 4.19 show the best configuration of the CNN models for each classification task. Any other parameters not mentioned in these tables were left at their default values. The models differ in the number of feature maps for each filter and the filter region sizes. All the other parameters are identical.

Parameter	Values
word vector dimensionality	300
filter region sizes	(6,7,9,10,15)
feature maps	300
activation function	ReLU
pooling	1-max pooling
dropout rate	0.4
optimization algorithm	Adam
L2 regularization	False

Table 4.18: The best configuration of the CNN model for relevance classification. It uses 5 convolutional layers each with a different region size of 6,7,9,10,15. All filters have 300 feature maps and use ReLU activations. 1-max pooling is performed after each filter is applied. A dropout of 0.4 is applied before the softmax layer. Learning was optimized using Adam.

Parameter	Values
word vector dimensionality	300
filter region sizes	(1,2,3,4,5)
feature maps	100
activation function	ReLU
pooling	1-max pooling
dropout rate	0.4
optimization algorithm	Adam
L2 regularization	False

Table 4.19: The best configuration of the CNN model for polarity classification. It uses 5 convolutional layers each with a different region size of 1,2,3,4 and 5. All filters have 100 feature maps and use ReLU activations. 1-max pooling is performed after each filter is applied. A dropout of 0.4 is applied before the softmax layer. Learning was optimized using Adam.

4.6 Summary

The results presented in this chapter show that the convolutional architecture outperforms all other baselines, including bag-of-words, with even the simplest configurations. We also compared our hierarchical structure of dividing our problem into two binary classification tasks with a joint structure that attempts to do both tasks in one pass. The hierarchical models achieved slightly better F1 scores, but the analysis of the results showed that each structure makes different kind of mistakes.

The hierarchical models showed that each task has it's own behavior regarding performance across the different review categories and hyperparameter configurations. In general our results align with the findings of Y. Kim (2014) and Zhang and Wallace (2017). Dropout had a very small effect and L2 regularization generally hurt performance across both tasks. Tuning the number of filters and the filters' regions size proved to be very difficult and time consuming. Each task had also different optimal filter regions. The advantage to our hierarchical approach was that we could fine tune these parameters for each task.

One point where our findings differed from Zhang and Wallace (2017) was the effect of the number of feature maps. While they reported single filters with large feature maps as having better performance, we found that combining different region sizes with a smaller number of

feature maps worked better for our tasks.

For both our tasks word vector dimensionality had a big impact on the models' performance, with larger vector sizes resulting in better accuracy and F1 scores. Using dynamic embeddings showed a small increase in performance, but was by far the most costly alternative in terms of memory use and training time.

Chapter 5

Final evaluation

In this chapter we present the evaluation results of the tuned convolutional models on the held-out test set. So far we have only evaluated the models' performance on the development set. We will also include the results of the best performing baseline, the Bag-of-words feed-forward network (BOW), for reference. As previously described in chapter 3, the labels of our dataset were automatically generated, but in order to create a test closer to a gold standard we manually checked a random subset of documents from the test set and corrected the labels by eliminating false positives for both relevance and polarity. For more details about the manual annotation process see Section 3.3.

The aforementioned process meant that we had two tests sets, the *A test set* that was larger but contained labels that were automatically generated, and the *M test set* that was smaller but had been manually corrected. It is important to note that false positives were corrected, but not false negatives. In other words we made the classification task intrinsically harder as we just made the set of possible correct predictions smaller. Table 5.1 shows a comparison of the basic counts between the A test set and the M test set.

	A test set	M test set
# Documents	225	76
# Sentences	15896	5468
# Tokens	18981	9632
Average # sentences	71	72
Average # tokens	1255	1288

Table 5.1: Basic corpus counts comparison between the automatically labeled test (A test set) set and the manually-corrected test set (M test set).

We have previously discussed the different challenges present in evaluating sentence identification and keyphrase extraction systems in Section 2.2.7. Some of these include labeling errors, not accounting for synonymy and disregarding results that are syntactically different but semantically equivalent. In order to avoid some of the pitfalls mentioned we conducted both automatic evaluation, meaning calculating traditional metrics like accuracy and F1-scores, and a manual analysis of some of the sentences extracted.

A summary of the models' performance on the *M test set*, including scores from the development set is presented in Table 5.2. All models' performance was lower in the test set, as is normally the case, but the performance drops were relatively high for relevance and polarity. We attribute this mainly to the fact that the model was trained on automatically labeled data, but tested on manually-corrected data. Results were also lower for the end-to-end task, which combines both relevance and polarity tasks, because the errors made in the first task carry over to the second.

		Accuracy		F1	
Model	Task	Dev	Test	Dev	Test
BOW	Relevance	84.10	79.30	66.43	59.72
BOW	Polarity	85.20	74.90	84.73	74.85
BOW	End-to-end	78.14	75.65	63.31	59.11
CNN	Relevance	88.02	82.26	72.37	63.51
CNN	Polarity	88.17	76.99	87.96	76.99
CNN	End-to-end	82.51	78.65	68.68	61.81

Table 5.2: Evaluation of the baseline BOW model and the best performing CNN model in both the development set and the M test set.

Even though the end-to-end task is the most difficult task of the three, it also had the lowest drop in accuracy between development and test. This result can be slightly misleading due to the nature of our manual test set. As mentioned in the previous chapter most of the mistakes on this task were made during the relevance task, and the models tended to classify most sentences as "not relevant". Because we corrected false positives, sentences that were erroneously labeled as "relevant" were now "not relevant", meaning that the raw number of "not relevant" sentences increased. This is also the reason why the majority classifier also had a higher score for the relevance and end-to-end tasks. Results including evaluation on the A test set will be presented in the coming sections.

We will first present the results of task 1, relevance, in Section 5.1, followed by the results of task 2, polarity, in Section 5.2 and the results of the end-to-end task in Section 5.3. We will present the results of our manual analysis of the models' output in Section 5.4. Finally we touch on sentence boundaries and readability in Section 5.4.2 and close with a summary of the chapter in Section 5.5.

5.1 Relevance classification

Table 5.3 shows the evaluation results for task 1 on both test sets for the BOW baseline and the convolutional model. Scores across all metrics were, as expected, higher for the automatically labeled test set. The CNN model had the largest difference in performance across all metrics compared to the BOW model. The largest difference being in the precision score. This is also an expected result given that false positives generated by the automatic labeling process were corrected in the M test set. Thus, the precision score had to necessarily decrease, as well as the overall performance. The majority class, "not relevant", was also larger in the M test set, which translated into better accuracy for the majority classifier. Both of our trained models outperform the majority class classifier.

The CNN model generally outperforms the BOW, with the exception of recall on the automatically labeled test set. However, differences in recall are generally small, booth between models and between test sets. The results on the A test set are comparable to the results on the development set, for a summary of the development results see Table 5.2.

Model	Acc	P	R	F1
M test set				
MC	74.31			
BOW	79.30	55.38	64.79	59.72
CNN	**82.26**	**61.92**	**65.17**	**63.51**
A test set				
MC	70.00			
BOW	84.58	71.87	**65.53**	68.56
CNN	**88.04**	**84.45**	65.40	**73.72**

Table 5.3: Final evaluation results for relevance. M test set is the manually corrected test set, A test set is the test set that was automatically labeled. BOW is the bag of words baseline. CNN is the best configuration of the convolutional model. MC is a majority classifier that only precicts "not relevant".

5.2 Polarity classification

Table 5.4 shows the evaluation results for task 2 on both test sets for the BOW baseline and the convolutional model. The CNN models outperforms the BOW baseline in all metrics but two: precision for pros and recall for cons. However, the difference in these scores was very small. Note that this is the case in both test sets, so it is unlikely to be a result of the different labels resulting from our corrections.

Even though the CNN had better accuracy and F1 scores than the baseline, it did also suffer the largest performance drop between the automatic test set and the manual one across all metrics compared to the BOW. Like the previous task, the main difference was in the precision score, which was the only curated metric during manual correction of the test labels for the M test set. An important difference with the previous task was that the majority class for polarity changed after our manual corrections, from "con sentences" to "pro sentences". This is the reason why the majority classifier has a lower score in the M test set as it only predicts "con sentences", which is the majority class of the training set. Both of our trained models outperform the majority classifier

The difference in precision scores between the two test sets for this task as also lower than for the relevance task. The polarity results on the A test set are comparable to the results on the development set which are shown in Table 5.2.

		Pros			Cons		
Model	Acc	P	R	F1	P	R	F1
M test set							
MC	45.32						
BOW	74.90	**86.48**	64.12	73.64	67.01	**87.90**	76.05
CNN	**76.99**	85.47	**69.77**	**76.83**	**70.15**	85.69	**77.15**
A test set							
MC	56.24						
BOW	85.66	**87.37**	78.59	82.75	84.55	**91.16**	87.73
CNN	**88.12**	87.27	**85.29**	**86.27**	**88.75**	90.32	**89.53**

Table 5.4: Final evaluation results for polarity. M test set is the manually corrected test set, A test set is the test set that was automatically labeled. BOW is the bag of words baseline. CNN is the best configuration of the convolutional model. MC is a majority classifier that only predicts "con sentence".

5.3 End-to-end results

We have divided our sentence identification task into two sub-tasks, relevance and polarity. The end-to-end task refers to performing both tasks sequentially, first classifying sentences as relevant and then classifying relevant sentences in terms of sentiment polarity. Figure 4.1 in chapter 4 illustrates this process. We also refer to this process as a "hierarchical model", because any errors in the first task carry over to the second task.

While we also experimented with a single "joint" model that was trained performing both tasks simultaneously, results during development showed better F1 scores when using a separate model for each task, for a comparison of "joint" versus "hierarchical" see section 4.4.

Table 5.5 shows the evaluation of the end-to-end results on both test sets for the BOW baseline and the convolutional model. Figure 5.1 shows the confusion matrix of the best CNN model. The CNN models outperforms the BOW baseline in all metrics except recall for pros. Note that this is the case in both test sets, so it is unlikely to be a result of the different labels resulting from our corrections.

Even though the CNN had better accuracy and F1 scores than the baseline, it did also suffer the largest performance drop between the A test set and M test set one across all metrics compared to the BOW, as was the case with the previous tasks. The large drop in performance

found in the precision score for cons can be attributed to the fact most of the labeling errors during the automatic dataset generation consisted in labeling non relevant sentences as cons.

		Pros			Cons		
Model	Acc	P	R	F1	P	R	F1
M test set							
MC	76.31						
BOW	75.65	51.26	**42.94**	46.73	36.44	57.24	44.53
CNN	**78.65**	**53.76**	41.38	**46.77**	**43.28**	**60.31**	**50.39**
A test set							
MC	75.96						
BOW	78.16	51.56	**52.39**	51.97	50.22	62.49	55.69
CNN	**82.09**	**55.81**	50.24	**52.88**	**63.10**	**68.36**	**65.62**

Table 5.5: Final evaluation end-to-end results. M test set is the manually corrected test set, A test set is the test set that was automatically labeled. BOW is the bag of words baseline. CNN is the best configuration of the convolutional model. MC is a majority classifier.

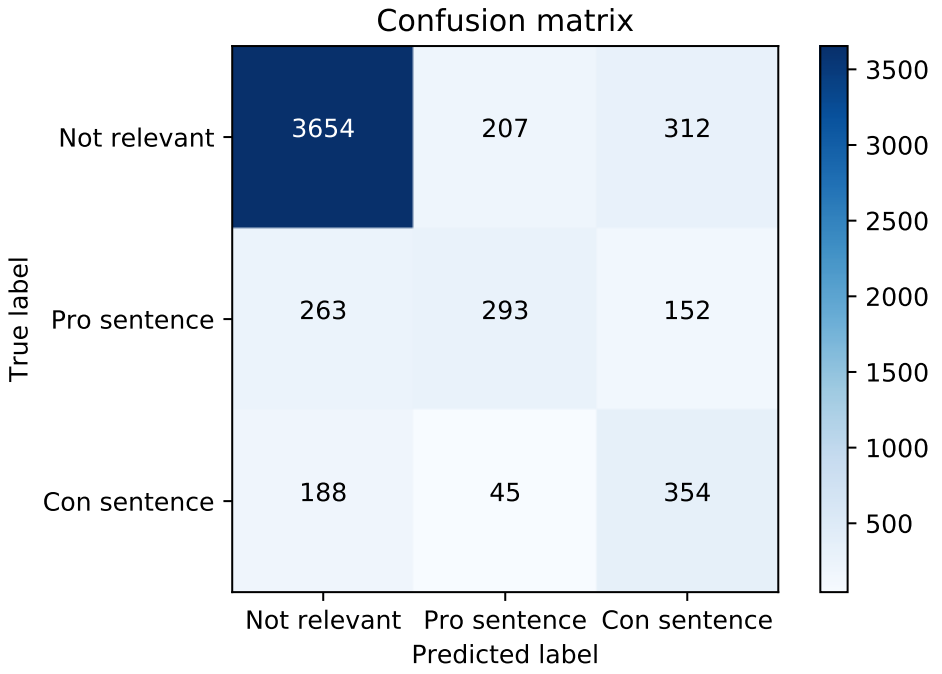


Figure 5.1: Confusion matrix of the best CNN configuration after being evaluated in the manually corrected test set. More sentences were wrongly classified in terms of relevance than polarity.

We also compared the performance of our best CNN configuration across the different product categories in the test set for this task. Figure 5.2 shows performance in each category in terms of accuracy. Category *bolig* (residential) had the best performance with a score of 82.95%. The category with the lowest performance was *mobil* (mobile) with a score of 75.32%.

We expected that performance for each category would be tied to document frequency, because the content belonging to the least frequent categories would be more "out of domain" for the model. However, *mobil* (mobile) had the lowest performance even though it is the third largest category. Conversely *fritid* (leisure) did exceptionally well for a category that represents less than 1% of the whole dataset. Category *data* (data) is the most frequent category, accounting for almost half of the dataset, and was the second lowest performing category. One possible explanation is that the products in the *data* (data) category are more varied than the other categories.

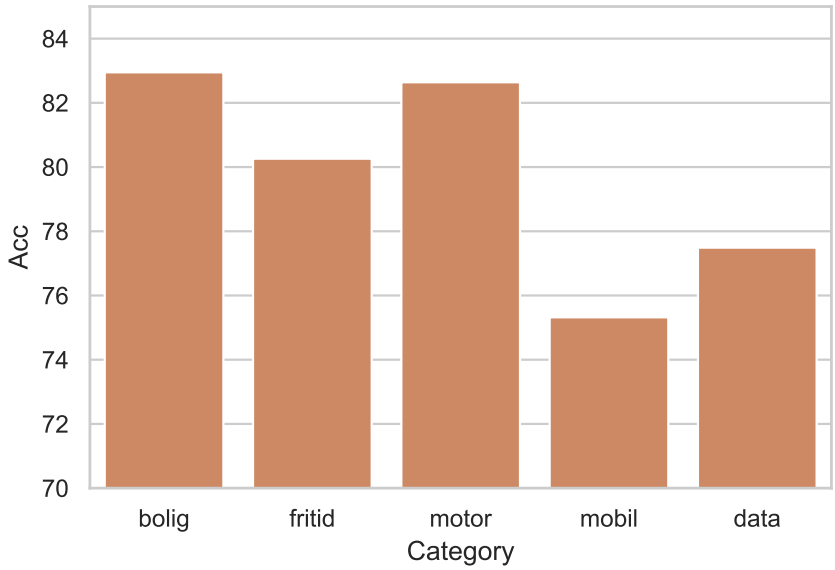


Figure 5.2: CNN Model's performance in terms of accuracy across categories in the test set.

The F1 scores of the best CNN configuration across the different categories of the test set are shown in Figure 5.3. Category *bolig* (residential) is once again the top performer with an F1 score of 66.55%, while *fritid* (leisure) had the lowest F1 score of 55.71%.

Because of the class imbalance present in our dataset the F1 metric is a better indicator of the models ability to identify relevant sentences and their polarity. Also here we found that one of the least frequent categories, *bolig* (residential) outperformed the most frequent ones. This was also the case during development, as explained in Section 4.3.5. A possible hypothesis can be that the writing in this category is more consistent than the others, making it easier for our model to identify relevant sentences and their polarity.

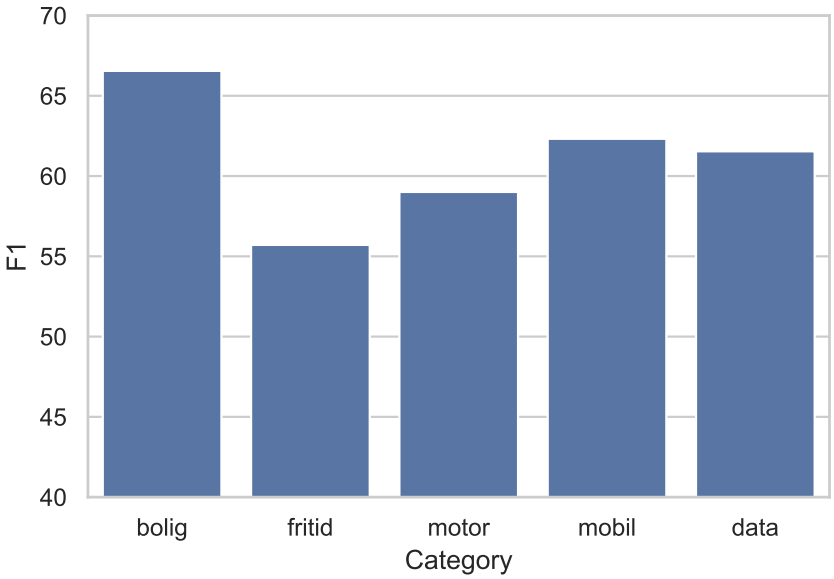


Figure 5.3: CNN Model's performance in terms of F1 score across categories in the test set.

5.4 Manual analysis

Human evaluation for keyphrase extraction can be done in several ways. One method is to look at the keyphrases outputted by a model and determine whether or not they represent the content of the text. S.-M. Kim and Hovy (2006) defined this task specifically for reviews, proposing that good keyphrases are those that can answer the question: *"What are the reasons that the author of this review likes or dislikes the product?"* (S.-M. Kim & Hovy, 2006, p. 483). Another method, proposed by Berend and Vincze (2012) consists of comparing the keyphrases produced by the system to the keyphrases provided by the authors of the text or manual annotators. In our case this means comparing the sentences identified by our system to the pros and cons in the "pros/cons section" of the reviews. We will consider both approaches when discussing the output of our model. In this section we will also present some examples of the output of the system, and look at the kind of classification mistakes the model makes.

Even though Berend and Vincze (2012) recommend to use multiple human evaluators to check the performance of a model, we only had a single evaluator available to perform this analysis. Thus, we cannot

completely eliminate the element of subjectivity that is present in opinionated sentences. The human evaluator was also the same person who corrected false positives in the test set.

In order to see how our model's predictions match the pros and cons determined by the author of the review we manually compared the results of our system with the "pros/cons section" of the review. Due to time and resource constraints we took a sample of 30 random documents from our manually corrected test set to perform this analysis.

Examples of some of the negative sentences identified by the model in a review for a wearable fitness tracker[1] are shown below:

(5.1) *Etter å ha latt oss begeistre over flere Fitbit-produkter de siste årene, som Charge og Charge HR, er vi litt skuffet denne gangen.*
After being excited about several Fitbit products in recent years, such as Charge and Charge HR, we are a little disappointed this time around.

(5.2) *skjermen man må tappe på ikke alltid reagerer.*
the screen you have to tap does not always react.

(5.3) *Og ikke minst lite brukervennlig.*
And not least little user-friendly

These sentences do answer why the author dislikes the product, meaning that the model is in fact capturing the main negative points of the review. However, when comparing the output to the author's defined cons we find that not all represent the same content directly. Some of the author's cons were:

(5.4) *Tungvint å styre skjermen*
Screen was cumbersome to use

(5.5) *upraktisk design*
unpractical design

While sentence 5.2 matches the content of con 5.4, both referencing the screen of the product, the author does not directly references user-friendliness as one of the cons. The "con sentences" identified by the model do not reference the product's design directly either. We cannot then conclude that the model captures all the same key points that the author defined in this review.

A similar situation to the one described above happens with the pros of this review. They do answer the question about why the author liked

[1] https://www.dinside.no/mobil/test-fitbit-alta/60950180

the product, raised by S.-M. Kim and Hovy (2006), but we do not find that the content of all of the author's defined pros and cons was always represented. However, we do find situations in other reviews where the same content is present. Below are some examples from a different review[2], where sentence 5.6 was extracted by the model and phrase 5.7 was defined by the author.

(5.6) *Denne har skøyhøy oppløsning og kan ta bilder med 23 megapiksler, selv om sensoren effektivt er 24,7 megapiksler.*
It has soaring resolution and can take 23 megapixel images, even though the sensor is effectively 24.7 megapixels.

(5.7) *meget godt kamera med god vidvinkel*
very good camera with good wide angle

Another example of the identified sentences having the same content as author defined keyphrases can be found in examples from another review[3]. The model extracted sentence 5.8, which corresponds to the author's keyphrase 5.9:

(5.8) *Menyene er ikke så enkle og brukervennlige som vi kanskje kunne ønsket oss, selv om vi skjønner at kineserne hensikten er å dekke nesten ethvert tenkelig bruksområde for en musikkspiller.*
The menus are not as simple and user-friendly as we might have liked, even though we realize that the Chinese intend to cover almost every imaginable application for a music player.

(5.9) *Kronglete menyer*
cumbersome menues

These examples are illustrative for the output found for other reviews. In general we find that the model manages to answer the question raised by S.-M. Kim and Hovy (2006) reasonably well. On the other hand it lacks recall for all of the author's defined phrases, and it sometimes identifies sentences that are only tangentially related to author's pros and cons.

5.4.1 Error analysis

Even though simply looking at the confusion matrix shown in Figure 5.1 gives us a good idea of the kind of classification errors the model

[2] https://www.dinside.no/mobil/endelig-har-sony-kameraet-blitt-bedre/62437221

[3] https://www.dinside.no/data/denne-musikkspilleren-gjor-det-meste-riktig/65389686

makes, we also looked at some examples of wrongly classified sentences from the same sample of 30 documents. Looking at the model's output beyond just the classification label can give some additional insights into the kind of sentences the model is having trouble classifying.

Most of the polarity classification errors we found in the sample were due to long sentences referencing different aspects of a product or a rebuttal within the same sentence. For example the following sentences were classified wrongly in terms of polarity and they include both positive and negative points:

(5.10) *På tross av sine relativt små, men like fullt nokså irriterende feil og mangler, gjør FiiO X1 2nd gen jobben den er beregnet til godt.*
In spite of its relatively small but still rather annoying faults and defects, FiiO X1 2nd gen does the job it is intended for well.

(5.11) *Joda, det er pent nok, og spesielt produktbildene ser flotte ut, men den svarte utgaven vi fikk inn til test synes vi fortsatt ser litt traust ut.*
Yes, it's pretty enough, and especially the product pictures look great, but the black version we got for testing, we still think it looks a bit boring.

(5.12) *Sensoren er 1/2,3", som er blant de aller største på mobilkamerafronten, men med den svært høye oppløsningen, blir pikselstørrelsen rundt 1,05µm, som er ganske lite sammenlignet med mange andre.*
The sensor is 1/2.3", which is among the largest on the mobile camera front, but with the very high resolution, the pixel size is around 1.05µm , which is quite small compared to many others.

Sentence 5.11 is particularly tricky because the author lists one of the pros as being: *Stiligste treningsarmbåndet fra Fitbit til nå* (The most stylish training band from Fitbit to date). On the one hand the pro listed by the author makes it seem like it is a stylish product, on the other the text of the review says it is boring but better than it's predecessors. It is impossible for a model that looks at sentences in isolation to be able to map complex relationships such as this one.

In terms of relevance our models show a weaker performance than in the polarity task. This was also the case for the development set, as shown in Table 5.2 at the beginning of the chapter. Determining if a sentence is relevant or not to the sentiment of the review is a

more subjective task than determining polarity. Examples 5.13 and 5.14 below show some sentences that were classified as relevant by the model, but had "non relevant" as their true label:

(5.13) *Denne musikkspilleren gjør det meste riktig.*
This music player does almos everything right.

(5.14) *Vi føler likevel at produsenten har ofret for mye funksjonalitet og ikke minst brukervennlig til fordel for en penere design, som ikke nødvendigvis er noe bedre.*
We still feel that the manufacturer has sacrificed too much functionality and, not least, user-friendliness in favor of a nicer design, which is not necessarily any better.

It is not difficult to imagine that a different human annotator could consider these sentences to be relevant. This is not to say the errors made by the model are solely due to mislabeling of the dataset, but it is still an important part of the overall evaluation. Hasan and Ng (2014) estimate that 7-10% of the reported errors in keyphrase extraction tasks are due to labeling errors in test sets.

Errors were also made by classifying relevant sentences as not relevant. Examples 5.15, 5.16 and 5.17 showcase relevant sentences, both positive and negative, that were labeled as "not relevant":

(5.15) *klar og ren lyd*
Clear and clean sound

(5.16) *Ikke sømløs avspilling*
No seamless playback

(5.17) *Den er riktignok 700 kroner dyrere enn sin forgjenger, men allikevel er prisen på 6500 kroner helt ålreit sett i forhold til konkurransen, spør du oss.*
It is certainly NOK 700 more expensive than its predecessor, but nevertheless the price of NOK 6500 is quite alright in relation to the competition, if you ask us.

These examples are generally illustrative of the performance of the model. In general polarity errors are due to difficult edge cases, such as sentences with rebuttals or containing both positive and negative aspects. Relevance errors are not only more frequent but the system also fails in some "easy cases" such as example 5.15. Given the hierarchical nature of the model the errors in the relevance task

carry over to the final output of the model. However this is also an advantage of the hierarchical setup because future tuning of the model for relevance, or even using a completely different classifier, will not affect the models performance in the polarity task.

5.4.2 Sentence boundaries

Good sentiment summarization systems should not only have good classification accuracy but also produce readable outputs (Meng et al., 2017). In order to achieve this goal our system outputs the same sentence it takes as input. Looking at the output we found sentences that were difficult to understand on their own. The corpus used for developing the system was already pre-processed as described in Section 2.1, and the sentences were already split. We fed the sentences one by one respecting the segmentation that was done when creating the corpus. This meant that any errors made during sentence splitting carried over to our output. For example sentence 5.4.2:

(5.18) *Detaljnivået er også i en helt annen liga enn hva tilfellet er for X Performance, noe du ser i 100%-beskjæringene nederst.*
The level of detail is also in a completely different league than the case for X Performance, which you see in the 100% crops at the bottom.

Was split into 5.19 and 5.20:

(5.19) *Detaljnivået er også i en helt annen liga enn hva tilfellet er for X*
The level of detail is also in a completely different league than the case for X

(5.20) *Performance, noe du ser i 100%-beskjæringene nederst.*
Performance, which you see in the 100% crops at the bottom.

While this issue does not directly affect the overall performance of the model, some of the sentences extracted can be difficult to understand or interpret on their own.

5.5 Summary

The results presented in this chapter show that the convolutional architecture outperforms the bag-of-words baseline in all tasks in terms of accuracy and F1 scores. This was the case for both the automatically

labeled test set and the manually corrected one. These results are consistent with the performance shown during development. The relevance task had lower F1 scores than the polarity task. Because relevance was the first task in our hierarchical model it also affected the scores during end-to-end evaluation. Since we do not have gold standard annotations for our dataset, a manual error analysis was performed in a small sample of documents. In general the results observed during the manual analysis were consistent with the scores obtained during evaluation.

The sentences that constitute the final output of the model were in general able to answer the question *"What are the reasons that the author of this review likes or dislikes the product?"* (S.-M. Kim & Hovy, 2006, p. 483). However they did not always directly represent the same content found in the pros and cons defined by the author. There is also room for improvement when determining sentence boundaries during pre-processing.

Chapter 6

Conclusion

In this thesis we have created a sentence identification system for reviews in Norwegian. The system identifies sentences that are relevant to the sentiment of the review and classifies each sentence as a positive or negative standpoint about the object of the review. Our work differs in important ways from previous studies regarding keyphrase extraction because relevant sentences for reviews are a combination of opinions and facts. Thus, identifying sentiment bearing sentences constitutes a distinct problem from subjective opinionated sentence identification or keyphrase extraction. Additionally the system also classifies the sentiment polarity of each relevant sentence. We divided our main task of identifying sentiment bearing sentences into two sub-tasks, which we called *relevance* and *polarity*. Both tasks were formulated as sentence-level binary classification problems.

Furthermore, we have created three datasets of labeled sentences from Norwegian reviews where each sentence is labeled according to its relevance towards the review's overall opinion and its polarity. We were the first to utilize the pros/cons corpus, a corpus consisting on professional reviews from the website DinSide.no where each review is annotated with keyphrases divided into positive and negative phrases. These phrases are referred to as pros and cons. Our hypothesis is that these pros and cons provide a good summary of the reviews' sentiment. We used these pros and cons as distant supervision to automatically label each sentence from the corpus.

We built upon the work done by S.-M. Kim and Hovy (2006) to generate our dataset. In order to determine if a sentence was relevant or not our labeling process checked if the words from a keyphrase were present in a sentence. We dubbed this process *matching*. We experimented with different strategies to *match* sentences to cons or

pros. We also extended the matching process by compiling a list of all the pros and a list of all the cons from all reviews, called *global keyphrases*, and checked for *matching* against these global lists.

The most successful strategy was *ordered overlap*, where all the words from a pro or a con must be present in the sentence and must appear in the same order. This strategy worked better when the *global keyphrases* were used. The resulting datasets were named after each of the tasks, namely *relevance*, *polarity* and *end-to-end*. The *relevance* dataset has each sentence labeled as "relevant" or "not relevant" as a representation of the overall sentiment toward the product being reviewed. The *polarity* dataset contains only "relevant" sentences and each sentence was labeled as a "pro sentence" or a "con sentence", due to the fact that they represent the same content as the pros and cons of the review.

The *end-to-end* dataset is different from the previous two because it contains three labels. Sentences are labeled as "not relevant", "pro sentence" and "con sentence". In other words, these are the labels for the complete sentence identification task without being divided into our two sub-tasks.

In chapter 3 we detailed the creation of these datasets for our tasks. Even though the result is not as robust as a human-annotated dataset, it is a good alternative as a starting point for languages that lack labeled data for sentiment-related tasks. A subset of documents from each of the test sets was manually corrected, eliminating false positives, in order to create test sets that were closer to a gold standard. This process showed that 80.93% of the sentences were labeled correctly in terms of both relevance and polarity.

Our *matching* strategies for automatic labeling struggled with sentences that matched with both pros and cons. However, manual inspection revealed that these sentences were also difficult for us to classify under one of the two categories. There is a non-negligible degree of subjectivity when determining if a sentence like *"Sono PlayBase gir deg praktisk, men ikke fantastisk lyd til stua."* (Sono PlayBase gives you practical, but not amazing sound for the living room) is positive or negative. Still, we show that excluding these ambiguous sentences from the dataset provides a good dataset for relevance and sentiment polarity classification with little data loss.

We established strong baselines based on neural network architectures and analyzed the performance of these models both on the dataset

as a whole and also for each product category separately. Finally, we fine-tuned a convolutional neural network and explored a wide range of values for each hyperparameter in order to determine which had the highest impact. Our results show that well performing configurations for other sentiment datasets are not necessarily optimal for our dataset.

We showed how the task of identifying sentiment bearing sentences can be split into two sub-tasks, which we called *relevance* and *polarity* classification. Even though most machine learning methods can tackle the problem as one task our results indicate that different parameters are better suited for each sub-task. We explored different model architectures for each task, and our baseline model used a feed-forward neural network with bag-of-words as its input.

We also experimented using fastText word embeddings as our input in order to better account for out of vocabulary tokens. Each sentence was represented as the averaged sum of the word vectors, a strategy also known as a continuous bag-of-words. Different pre-trained word embeddings were used to evaluate the effect of using word embeddings as input representations. Randomly initialized embeddings that were tuned during training were also used. The bag-of-words feed-forward network was a strong baseline that performed well above simple non-trained baselines. The non-trained baselines used were a majority class classifier and a stratified classifier, a classifier that makes random predictions based on the distribution of labels on the training set.

A convolutional neural network classifier was also implemented. We performed a thorough exploration of the hyperparameter space focusing primarily on the number of filters and the filter region size. Other parameters such as the number of feature maps, regularization strategies and different embeddings as input representations were also explored. Most of our findings match other reports of hyperparameter tuning for sentence sentiment classification found in the literature. The best performing models for each of the sub-tasks used different filter region sizes and different number of feature maps, showing that it was beneficial to divide the larger task and tune different models individually. The convolutional model outperformed all other models, including the bag-of-words baseline.

The final evaluation confirmed the results obtained during development. The confusion matrices showed that determining whether a sentence is relevant to the authors opinion of the product was more difficult than classifying relevant sentences as positive or negative. Our results

also showed that our system performed well in the smaller product categories of the dataset, indicating that the models managed to generalize reasonably well for product categories that constituted less than 1% of the dataset. Our results were positive considering that we did not have direct access to gold labels to train our models.

Our manual analysis showed that, in general, the system outputs sentences that provide a good indication of the reasons why the review's author likes or dislikes a product. One shortcoming was that not all the identified sentences matched directly with the pros and cons written by the authors. For some of the reviews our model failed to recognize some sentences that we believed to be "easy cases" of relevant sentences, such as: *"Vanskelig å bruke skjermen"* (difficult to use the screen). On the other hand, most of the pro and con sentences identified were found to be informative and useful summaries.

6.1 Future work

A sentence identification system like the one presented in this thesis can be used as part of an automatic summarization system for reviews. Additionally the sentences extracted by our model can complement content-based recommender systems, different kinds of information retrieval systems and document categorization systems. Our system can also help simplify the task of creating fine-grained annotations for sentiment analysis by filtering out the sentences that are relevant to annotate.

Identifying positive and negative sentences can be specially useful for online reviews written by users to complement opinion mining systems. Online stores can use these sentences to provide short summaries to their users based on large amounts of reviews to make the content more readable and easier to comprehend.

We have found the optimal values for several hyperparameters of a convolutional neural network classifier for each of our subtasks. However, exploring the different interactions between all of the hyperparameter space was beyond the scope of this project. Using the ranges that we have identified for each parameter in isolation, more sophisticated methods for hyperparameter tuning can be used such as random search or Bayesian optimization frameworks.

Extensions to our proposed convolutional architecture can also be made by adding multiple channels for different kinds of inputs. Due to

time and resource constraints we limited our inputs to word lemmas. It would, however, be interesting to explore different channels using word surface forms or part of speech tags. A natural next step would also be to use hierarchical convolutions to create intermediary representations for sentences.

Recurrent neural networks have shown promising results in many NLP tasks. It would be interesting to see how recurrent networks perform in both the *relevance* and *polarity* classification tasks in comparison to convolutional architectures. One of the advantages of our approach is that if the recurrent models outperform ours in only one of the tasks both types can be used in conjunction to produce a better end result. Additionally both convolutional and recurrent architectures could be combined by having first a one-dimensional convolution with pooling over a sentence as a feature extractor and then feed the consolidated features to a recurrent network.

As mentioned in Section 5.4.2 a possible improvement for the pre-processing portion of our experimental setup would be to have better defined sentence boundaries to avoid using incomplete sentences as inputs. In this project we utilized the pre-split sentences from NoReC without any additional processing. The phrase boundaries for the pros and cons from the pros/cons corpus could also be improved, as the inconsistent use of punctuation, among other things, made it difficult to extract individual keyphrases from the "pros/cons section" of the review.

Finally, our setup could be transformed from a sentence identification task into a pure keyphrase extraction task. Instead of classifying each sentence independently, a different approach for task 1, *relevance*, could be that the model receives the whole review as its input and attempts to extract relevant phrases from the text. The selected phrases can then be classified as negative or positive individually, as we did in task 2, *polarity*.

References

Baccianella, S., Esuli, A., & Sebastiani, F. (2010). SENTIWORDNET 3.0: An Enhanced Lexical Resource for Sentiment Analysis and Opinion Mining. In *Proceedings of Language Resources and Evaluation Conference* (p. 5).

Berend, G. (2011). Opinion Expression Mining by Exploiting Keyphrase Extraction. In *Proceedings of 5th International Joint Conference on Natural Language Processing* (pp. 1162–1170). Retrieved from http://www.aclweb.org/anthology/I11-1130

Berend, G., & Vincze, V. (2012, July). How to Evaluate Opinionated Keyphrase Extraction? In *Proceedings of the 3rd Workshop on Computational Approaches to Subjectivity and Sentiment Analysis* (p. 5). 99–103: Association for Computational Linguistics. Retrieved from http://dl.acm.org/citation.cfm?id=2392963.2392984

Bergstra, J., & Bengio, Y. (2012). Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*, *13*(Feb), 281–305. Retrieved 2019-04-13TZ, from http://www.jmlr.org/papers/v13/bergstra12a.html

Bojanowski, P., Grave, E., Joulin, A., & Mikolov, T. (2016, July). Enriching Word Vectors with Subword Information. *arXiv:1607.04606 [cs]*. Retrieved 2019-04-08TZ, from http://arxiv.org/abs/1607.04606 (arXiv: 1607.04606)

Branavan, S. R. K., Chen, H., Eisenstein, J., & Barzilay, R. (2009). Learning Document-Level Semantic Properties from Free-text Annotations. *Journal of Artificial Intelligence Research 34*, *34*, 569–603. Retrieved from https://dblp.org/rec/bib/journals/corr/BranavanCEB14 doi: 10.1613/jair.2633

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014, June). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *arXiv:1406.1078 [cs, stat]*. Retrieved 2018-05-14TZ, from http://arxiv.org/abs/1406.1078 (arXiv: 1406.1078)

Chollet, F., et al. (2015). *Keras*. Retrieved from https://keras.io

Fares, M., Kutuzov, A., Oepen, S., & Velldal, E. (2017). Word vectors, reuse, and replicability: Towards a community repository of large-text resources. In *Proceedings of the 21st Nordic Conference on Computational Linguistics, NoDaLiDa, 22-24 May 2017, Gothenburg, Sweden* (pp. 271–276). Linköping University

Electronic Press, Linköpings universitet.

Firth, J. R. (1935). The Technique of Semantics. *Transactions of the Philological Society*, *34*(1), 36–73. Retrieved 2019-04-01TZ, from https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-968X.1935.tb01254.x doi: 10.1111/j.1467-968X.1935.tb01254.x

Goldberg, Y. (2017). *Neural network methods for natural language processing* (Vol. # 37). San Rafael, California.

Harris, Z. S. (1954). Distributional Structure. *WORD*, *10*(2-3), 146–162. Retrieved from https://doi.org/10.1080/00437956.1954.11659520 doi: 10.1080/00437956.1954.11659520

Hasan, K. S., & Ng, V. (2014). Automatic Keyphrase Extraction: A Survey of the State of the Art. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (pp. 1262–1273). Association for Computational Linguistics. Retrieved 2018-03-20TZ, from http://aclweb.org/anthology/P14-1119 doi: 10.3115/v1/P14-1119

Kalchbrenner, N., Grefenstette, E., & Blunsom, P. (2014, June). A Convolutional Neural Network for Modelling Sentences. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (pp. 655–665). Baltimore, Maryland: Association for Computational Linguistics. Retrieved 2019-04-01TZ, from https://www.aclweb.org/anthology/P14-1062 doi: 10.3115/v1/P14-1062

Kim, S.-M., & Hovy, E. (2006). Automatic identification of pro and con reasons in online reviews. In *Proceedings of the COLING/ACL on Main Conference Poster Sessions* (pp. 483–490). Association for Computational Linguistics. Retrieved 2018-03-20TZ, from http://portal.acm.org/citation.cfm?doid=1273073.1273136 doi: 10.3115/1273073.1273136

Kim, S. N., Medelyan, O., Kan, M.-Y., & Baldwin, T. (2010, July). SemEval-2010 Task 5 : Automatic Keyphrase Extraction from Scientific Articles. In *Proceedings of the 5th International Workshop on Semantic Evaluation* (pp. 21–26). Uppsala, Sweden: Association for Computational Linguistics. Retrieved 2018-04-10TZ, from http://www.aclweb.org/anthology/S10-1004

Kim, S. N., Medelyan, O., Kan, M.-Y., & Baldwin, T. (2013, September). Automatic keyphrase extraction from scientific articles. *Language Resources and Evaluation*, *47*(3), 723–742. Retrieved 2018-06-22TZ, from https://link.springer.com/article/10.1007/s10579-012-9210-3 doi: 10.1007/s10579-012-9210-3

Kim, Y. (2014, August). Convolutional Neural Networks for Sentence Classification. *arXiv:1408.5882 [cs]*. Retrieved 2019-03-21TZ, from http://arxiv.org/abs/1408.5882 (arXiv: 1408.5882)

Koppel, M., & Schler, J. (2006). THE IMPORTANCE OF NEUTRAL EXAMPLES FOR LEARNING SENTIMENT. *Computational Intelligence*, *22*(2), 100–109. Retrieved

from https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8640.2006.00276.x doi: 10.1111/j.1467-8640.2006.00276.x

LeCun, Y., & Bengio, Y. (1995). Convolutional networks for images, speech, and time-series. In M. A. Arbib (Ed.), *The handbook of brain theory and neural networks*. MIT Press.

Liu, J., & Seneff, S. (2009). Review sentiment scoring via a parse-and-paraphrase paradigm. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1* (pp. 161–169). Singapore: Association for Computational Linguistics. Retrieved 2018-03-20TZ, from http://portal.acm.org/citation.cfm?doid=1699510.1699532 doi: 10.3115/1699510.1699532

McCulloch, W. S., & Pitts, W. (1943, December). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, *5*(4), 115–133. Retrieved 2019-03-29TZ, from https://doi.org/10.1007/BF02478259 doi: 10.1007/BF02478259

Meng, R., Zhao, S., Han, S., He, D., Brusilovsky, P., & Chi, Y. (2017, April). Deep Keyphrase Generation. *arXiv:1704.06879 [cs]*. Retrieved 2018-04-06TZ, from http://arxiv.org/abs/1704.06879 (arXiv: 1704.06879)

Merrouni, Z. A., Frikh, B., & Ouhbi, B. (2016, October). Automatic keyphrase extraction: An overview of the state of the art. In *2016 4th IEEE International Colloquium on Information Science and Technology (CiSt)* (pp. 306–313). doi: 10.1109/CIST.2016.7805062

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013, January). Efficient Estimation of Word Representations in Vector Space. *arXiv:1301.3781 [cs]*. Retrieved 2019-04-01TZ, from http://arxiv.org/abs/1301.3781 (arXiv: 1301.3781)

Miller, G. A. (1995, November). WordNet: A Lexical Database for English. *Commun. ACM*, *38*(11), 39–41. Retrieved 2018-06-22TZ, from http://doi.acm.org/10.1145/219717.219748 doi: 10.1145/219717.219748

Paulus, R., Xiong, C., & Socher, R. (2017, May). A Deep Reinforced Model for Abstractive Summarization. *arXiv:1705.04304 [cs]*. Retrieved 2018-04-06TZ, from http://arxiv.org/abs/1705.04304 (arXiv: 1705.04304)

Pennington, J., Socher, R., & Manning, C. D. (2014). GloVe: Global Vectors for Word Representation. In *Empirical Methods in Natural Language Processing (EMNLP)* (pp. 1532–1543). Retrieved from http://www.aclweb.org/anthology/D14-1162

See, A., Liu, P. J., & Manning, C. D. (2017, April). Get To The Point: Summarization with Pointer-Generator Networks. *arXiv:1704.04368 [cs]*. Retrieved 2018-04-06TZ, from http://arxiv.org/abs/1704.04368 (arXiv: 1704.04368)

Stadsnes, C. (2018). *Evaluating Semantic Vectors for Norwegian* (Master's thesis). Retrieved 2019-03-18TZ, from https://www

.duo.uio.no/handle/10852/61756

Straka, M., & Hajic, J. (2016). UDPipe: Trainable Pipeline for Processing CoNLL-U Files Performing Tokenization, Morphological Analysis, POS Tagging and Parsing. In *Proceedings of the International Conference on Language Resources and Evaluation* (pp. 4290–4297).

Sullivan, T. (2008). Pro, Con, and Affinity Tagging of Product Reviews. *Stanford Technical Report*, 14.

Sutskever, I., Vinyals, O., & Le, Q. V. (2014, September). Sequence to Sequence Learning with Neural Networks. In *Advances in neural information processing systems* (pp. 3104–3112). Retrieved 2018-05-14TZ, from http://arxiv.org/abs/1409.3215 (arXiv: 1409.3215)

Velldal, E., Øvrelid, L., Bergem, E. A., Stadsnes, C., Touileb, S., & Jørgensen, F. (2018). NoReC: The Norwegian Review Corpus. In *Proceedings of the International Conference on Language Resources and Evaluation* (pp. 4186–4191). Retrieved from http://www.lrec-conf.org/proceedings/lrec2018/pdf/851.pdf

Witten, I. H., Paynter, G. W., Frank, E., Gutwin, C., & Nevill-Manning, C. G. (1999). KEA: Practical Automatic Keyphrase Extraction. In *Proceedings of the Fourth Association for Computing Machinery Conference on Digital Libraries* (pp. 254–255). New York, NY, USA: ACM. Retrieved 2018-06-17TZ, from http://doi.acm.org/10.1145/313238.313437 doi: 10.1145/313238.313437

Yu, N., Huang, M., Shi, Y., & zhu, x. (2016, December). Product Review Summarization by Exploiting Phrase Properties. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers* (pp. 1113–1124). Osaka, Japan: The COLING 2016 Organizing Committee. Retrieved 2018-04-06TZ, from http://aclweb.org/anthology/C16-1106

Zhang, Y., & Wallace, B. (2017, November). A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)* (pp. 253–263). Taipei, Taiwan: Asian Federation of Natural Language Processing. Retrieved 2019-03-21TZ, from http://www.aclweb.org/anthology/I17-1026

Řehůřek, R., & Sojka, P. (2010, May). Software Framework for Topic Modelling with Large Corpora. In *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks* (pp. 45–50). Valletta, Malta: ELRA.