Automated Identification of Severe Errors in Speech to Text Transcripts

Frederik Hjelde Rosenvinge

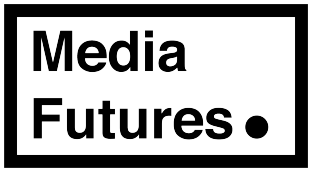
Samia Touileb, MediaFutures
Lubos Steskal, TV 2

Master's Thesis
Department of Information Science and Media Studies
University of Bergen

May 31, 2023

Scientific environment

This thesis is carried out at the Department of Information Science and Media Studies, University of Bergen, Norway. Supervision was conducted as a collaboration between MediaFutures Research Centre for Responsible Media Technology & Innovation and TV 2, with supervisors from both affiliations.



Acknowledgements

Thank you to both of my supervisors, Samia and Lubos, for their excellent supervision and support.

Frederik Hjelde Rosenvinge
Bergen, Norway, 31.05.23

Abstract

In this thesis we explore how problematic misplaced words can be automatically identified in speech-to-text-transcripts. Automatic Speech Recognition systems (ASR) are systems that can automatically generate text from human speech. Because natural language spoken by humans is complex, due to dialects, variations in talking speed, and differences in how humans talk compared to the training data, there might be errors introduced by such ASR systems. Sometimes, these errors are so bad that they become problematic. Post-processing of an ASR system means finding such errors *after* the text has been generated by the system. We want to find out to what degree probabilities of words computed using pre-trained language models can be used to solve this problem, as well as to what degree these probabilities can be used to create a classifier to detect problematic words. We present our solution, where we synthetically introduce problematic words into text documents. Then we compute probabilities of both problematic and non-problematic words in these documents to investigate if they are treated differently by the models. We show that the models generally assign lower probabilities to problematic words and higher probabilities to good words. We train a logistic regression classifier using these probabilities to classify words. Our results show that using probabilities from NorBERT1 and NorBERT2, a logistic regression classifier can accurately detect problematic words. We also show that NB-BERT performs worse than a baseline bigram model.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

If you have ever watched a show with subtitles in recent times, you will likely have seen that the subtitles do not always match what was said. Some words might be spelled wrong, or a different word was used instead of the word you heard, creating an error. Such subtitles are often created using Automatic Speech Recognition technology (ASR). ASR systems recognize the sounds a person makes when talking and translates these into subtitles meant for another person to read. Most of the time, these systems produce the expected words, but in some cases, they create problematic words which are unrelated to the spoken word, which may lead to offensive or otherwise inappropriate texts.

In this thesis, we present our methods for detecting such problematic words in speech-to-text transcripts. We are focusing on post-processing of the generated text created by ASR systems. This means that we are dealing with the text after it has been created. It is possible to solve the issue by improving the ASR system itself if you have access to it, but this is a different problem that we will not explore here.

Our approach to solve the problem is to use Norwegian pre-trained language models to identify problematic words. We focus on three Norwegian language models: NorBERT1, NorBERT2, and NB-BERT, which are all transformer-based models, and have already proven to be effective for many NLP tasks in Norwegian (*Kummervold et al.*, 2021; *Kutuzov et al.*, 2021).

The data we are using together with these models are transcripts of speeches transcribed in meetings at the Norwegian parliament (Stortinget), which is intended to use to improve ASR for Norwegian spontaneous speech and dialects, making it a good fit for our purposes (*Solberg and Ortiz*, 2022). This dataset was created by the National Library of Norway (*Solberg and Ortiz*, 2022). In addition to this dataset, we use a lexicon of bad words to introduce errors synthetically into the data from Stortinget, creating an augmented dataset that can help us with our research. Synthetically introducing errors was necessary due to a lack of errors in the training data.

Large pre-trained language models like the ones we are using, can be used to assign probabilities to words based on their surrounding words. We will compute probabilities of words and use these to classify words as either problematic or non-problematic. After computing probabilities of words, we will use logistic regression to classify them, which will give us insight into how the models treat non-problematic versus problematic words.

Note that in this thesis, when we talk about problematic words, we mean words that

both should not be present in a text, and that are derogatory in nature. When we talk about non-problematic words we mean words that are intended to be in the text. When we describe problematic words, we use terms such as "problematic", "catastrophic", "bad", "severe", interchangeably. This also applies to non-problematic words, where we use terms such as "non-problematic" or "good" interchangeably.

1.1 Problem Statement and Motivation

It is necessary to improve accessibility for people who rely on texting to understand the content of a TV program, primarily those who are deaf or hard of hearing. Transcription of speech is often tedious and time consuming, and making it more efficient through automation is a good way to deal with this. Research wise, it is worthwhile to show the capabilities of modern language model architectures by tackling the specific problem of improving ASR outputs, hopefully advancing our understanding of the limitations and possibilities of such models.

A fair question to ask in response to this is: Why not simply improve upon the existing ASR models that are making these errors, or replace them with better ones that are less faulty? In the case of TV 2 and other actors using them, the ASR models are provided by third-party suppliers, and access is not given to the training data and training regime of the model, making improvements impossible. Severe errors are not frequent, and therefore, using a system that is good enough can still be used even it if introduces errors sometimes. Trying to fix these errors as a post-processing step using pre-trained language models, that have already in many ways been proven to be highly capable is, in our opinion, a reasonable research endeavour.

1.2 Research questions

- **Research Question 1** (**RQ 1**): To what extent can probabilities of words computed using pre-trained language models be leveraged to automatically detect severe errors in speech-to-text transcripts?
- **Research Question 2** (**RQ 2**): To what extent can we develop a system, using pre-trained language models, that can find severe errors in speech-to-text transcripts, using word probabilities computed using pre-trained models?

To answer RQ 1, we will create a dataset with problematic words synthetically inserted into documents, computing probabilities for these being in the documents. Then we will see if the models assign different probabilities to problematic and non-problematic words to such a degree that we can clearly see a difference.

To answer RQ 2, we will use the probabilities computed using pre-trained language models together with a logistic regression classifier in order to detect problematic words.

1.3 Contributions

Our work has three main contributions:

1. We show that when computing probabilities of words using pre-trained Norwegian language models, the models generally assign lower probabilities to problematic words, and higher probabilities to non-problematic words.
2. Using a logistic regression classifier in combination with the probabilities computed using pre-trained Norwegian language models, we are able to show that it is possible to detect problematic words.
3. We show that using different pre-trained language models yield different results. Such that NorBERT1 outperforms NorBERT2, while NB-BERT performs worse than a simple bigram model.

1.4 Thesis outline

The rest of the thesis is structured as follows:

- **Chapter 2 - Background**: Will provide an overview of previous work done in the field of ASR detection and correction as well introduce the models we are using.
- **Chapter 3 - Data**: Will provide details on the data we are using, our reasoning for choosing it, along with its strengths and weaknesses.
- **Chapter 4 - Methods**: Will provide a detailed overview of everything we did that led to our results. We talk about what worked and what did not work, as well as thoughts on what could have been done better.
- **Chapter 5 - Results**: Gives a detailed overview of all results that we achieved from our experiments.
- **Chapter 6 - Conclusion and Future Work**: Will provide a summary of everything we did, as well as future works.

Chapter 2

Background

In this section we will give an overview of the field of ASR detection and correction, and the problem space that we are working within. We will give an overview of the previous works that have been done on automating error detection and correction of transcripts created by automatic speech recognition.

2.1 Automatic speech recognition

Automatic speech recognition **(ASR)** is a field that mainly aims to automatically transform human speech into human readable text. The applications for this technology are everywhere. Speech is a natural way for people to communicate with their smart home appliances, personal assistants, or cellphones, where keyboards are less convenient. Interaction between computers and humans with disabilities can result in difficulties or inabilities in typing or audition. (*Jurafsky and Martin*, 2021). ASR is an important technology, and is a supportive communication tool, making certain scenarios and activities easier and more inclusive.

This helps people such as those who are hard of hearing, or deaf. Blind people can use speech to operate a computer and other voice user interfaces such as Siri (Apple) and Alexa (Amazon), which are some popular examples of ASR. There is thus also an economic incentive for companies like the previously mentioned, as well as for example YouTube, which has millions of videos uploaded to their site every day. These videos get captioned with the use of ASR technology. TV 2, whom we are collaborating with, uses ASR models in the production of their shows. Doctors and medical professionals use ASR to help them create notes during patient consultations, creating a lesser cognitive load on physicians, which leads to better patient care (*Nanayakkara et al.*, 2022a). Driving can also be improved safety-wise, through the use of ASR (*Husnjak et al.*, 2014).

A problem with ASR systems is when they incorrectly recognize speech and create words that are different from what was actually being said by a speaker. These errors create problems for those that rely on correct transcripts, and can hinder accessibility. The first step in trying to improve these systems is to understand what causes problems in the first place. There is not a single cause, of course, but there are certain errors that are more common than others. Errors can be caused by a person talking in a hard-to-understand dialect, with both sounds and words that are unfamiliar to a model, a noisy

environment, or the models themselves being trained poorly.

For this thesis we are not trying to improve the performance of ASR models. Instead, we are researching the capabilities of pre-trained language models with regards to detecting wrong outputs from these ASR models. This is called a post-processing task because we are trying to detect mistakes after they have occurred.

2.2 Factors that affect ASR performance

As mentioned, there are many types of errors and settings that cause errors. *Errattahi et al.* (2018) say that "in general, ASR systems are effective when the conditions are well controlled". The key phrase being "well controlled". According to (*Errattahi et al.*, 2018) there are three factors that affect the performance of ASR. The first of these is **speaker variablities**: a person can, because of varying bodily conditions such as being tired or sleepy, talk differently. Someone who is tired can slur their words, not making them intelligible enough for an ASR system to pick up. The second factor is **spoken language variabilities**: People that live in different places have dialects that sound different in the same language. In Norway, dialects vary a lot from place to place. This is especially important for this thesis as we will be working with Norwegian speech data. It is important to be conscious of the amount of variability within the Norwegian language and how this might cause problems. The third factor is **mismatch factors**: a mismatch in recording setup for the training and testing data. If there is background noise in one of the datasets and not in the other, for example.

If an ASR system captures speech well, and accounts for human errors such as poor pronunciation, human language can still be highly ambiguous in many cases. This ambiguity can also lead to errors. Two sequences of text that on paper look different, can still be spoken more or less similarly. When two words are written differently but spoken similarly, we call these types of words *homophones*. There are an infinite number of ways to express yourself in text, but only a handful of sounds, or *phonemes*, that we can make. For English, words such as "right/write" or "flour/flower" are completely similar in pronunciation, and also completely different in meaning. The sentence "Thanks for the flower/flour" can be perceived in at least two ways. For Norwegian some examples are: "Jul" and "hjul", or "vert" and "verdt", or "vært" and "hvert". This will be exaggerated by different dialects as well.

As mentioned earlier, a problem in ASR systems is ambient noise from the environment and reverberation that interfere with the microphones that capture speech. For the purpose of the thesis, environmental noise was not a big issue, given that the data we are given comes from the Norwegian parliament, referred to as Stortinget in Norwegian, which can probably be relied on to have a good environment for capturing speech. We did notice in the data some that parts of the transcription was not transcribed properly because of inaudible speech, but this is more a problem with the individual speaker, or speakers talking over each other, rather than the environment. We are also focusing on the post-processing part of the ASR system, and dealing with environmental noise is not something we can directly influence on our part, but we wish to highlight how these errors can come from an earlier step in the pipeline.

Another problem that ASR models face are words that are not recognized because they are not in the vocabulary of the model. These are so-called out-of-vocabulary

words (OOV). A Norwegian language model could for example have trouble with French or English words because it wasn't trained with such languages. The model could also have problems with dialect-specific words that are not commonly used.

2.2.1 Evaluating ASR

The point of evaluating ASR systems is to understand how well they perform, and especially how one system performs in relation to another system. According to *Errattahi et al.* (2018) there are two key areas related to ASR errors. The first key area is the reference-recognized alignment which consists of finding the best word alignment between the reference and the automatic transcription (the recognized sequence). In other words, a reference and the transcription are aligned to each other to find out which words have been correctly transcribed and which words were wrongfully transcribed. Aligning two sentences is usually done with the Viterbi algorithm (*Jurafsky and Martin*, 2021). When we know what words correspond to each other in the reference and the transcript, we can evaluate using relevant metrics.

The second key area is the evaluation metrics measuring the performance of the ASR system. The are three types of errors in automatic speech recognition: insertions, deletions, and substitutions. For a given spoken input, an ASR model can either insert a wrong word that was not in the reference, delete a word that should have been included from the reference, or substitute a correct word from the reference with an incorrect word. Word error rate (WER) (*Jurafsky and Martin*, 2021), is one of the most common metrics used to evaluate transcripts, which we will see later when discussing previous works.

WER is defined as the proportion of word errors to words processed. Such that:

$$WER = \frac{S+D+I}{N} = \frac{S+D+I}{S+D+C}$$

where S is the number of substitutions, D is the number of deletions, I is the number of insertions, C is the number of correct words, N is the number of words in the reference, and is defined as: N=S+D+C.

Mccowan et al. (2004) says that an ideal ASR evaluation metric should have four properties. It should be:

- Direct: It should measure the ASR component directly, independently of the application of the ASR component.
- Objective: It should be calculated in an objective manner so that it can be properly used in research.
- Interpretable: The value of the measurement must have an intuitive relationship to system performance.
- Modular: It should be possible to use the metric in an application-independent way.

A problem with WER is that it is not a true percentage, because it has no upper bound. It doesn't tell you *how* good a system is, only that one is better than the other. Imagine an ASR model that outputs one wrong word for each word in the reference sentence. That gives a WER of 100%. An ASR model that outputs two wrong words for each word in the reference gives a WER of 200%, and yet they both convey zero of the intended information. This problem of an upper bound is brought up by both *Morris et al.* (2004) and *Mccowan et al.* (2004). Despite this, WER is the metric most researchers use to evaluate performance in their research, as we will see in the next section.

2.3 Overview of previous efforts/methods

In natural language processing, error correction aims to fix the output of a previous process. Other areas of relevance in addition to automatic speech recognition are for example optical character recognition *Mokhtar et al.* (2018) or neural machine translation *Song et al.* (2020). For our purposes, we are focusing on the error detection of the output of automatic speech recognition. What follows is a non-exhaustive overview of how other works have attempted to go from a bad ASR output to an improved text. Note that some of these methods attempt to both detect and *correct* the wrong output, while in this thesis we focus only on *detection*.

In most cases, you would want a system to be both fast and precise, but in many cases you have to accept either slower speeds (higher latency) with more precision (less errors in the transcript) or higher speeds at the cost of precision.

Leng et al. (2021a) says that previous works on ASR correction, like *Liao et al.* (2020), usually adopt an encoder-decoder based autoregressive generation model to correct the ASR output sentences. Autoregressive models try to predict the next word given the previous words. This creates issues with latency which makes such models hard to use in online ASR services. A solution to latency is to use a non-autoregressive (NAR) sequence generation model, but this has the potential to increases error rates. *Leng et al.* (2021a) proposes "FastCorrect", a novel NAR error correction model that leverages edit alignment (insertion, deletion, and substitution) between the tokens in the source and target sentences to guide error correction. They use a Transformer as the basic model architecture *Vaswani et al.* (2017). FastCorrect is 6 to 9 times faster than an autoregressive model depending on the use of GPU or CPU for testing. FastCorrect got a word error rate of, respectively, 4.16% and 10.27% when tested on an internal dataset and AISHELL-1, an open-source Chinese Mandarin speech corpus [1]. The autoregressive model had a WER of 4.08% and 10.22% on the test set of AISHELL-1 and the internal dataset, respectively. (*Leng et al.*, 2021a). As such FastCorrect does give comparable results on WER with significant increase in speed.

Other researchers are also attempting to create solutions that are fast enough for use in a live setting. *Zitkiewicz* (2022) proposed a post-editing system for editing text made by automatic speech recognition systems that aims to be precise, easily controllable, and data efficient. The system consists of a neural speech tagger and a corrector module. The tagger is trained to recognize which words of an ASR hypothesis should be corrected using a corpus of ASR hypothesis with edit operation tags. In the corrector

[1] https://www.openslr.org/33/

module, only the words with tags are given. A given tag corresponds to a set of operations to perform to correct the word. If a hypothesis word like "cat" is the plural "cats", then a tag "append_s" indicates to add an "s" to the word in the correction module.

They trained and evaluated two tagging model types: a BERT token classification model and a contextual string embeddings model, referred to as "Flair" (*Akbik et al.*, 2019). The Flair models contain BERT models extended with contextual string embeddings *Akbik et al.* (2018), LSTM, and CRF layers. Both the BERT and Flair tagger model comes in three variations: French by *Martin et al.* (2020), German by *Chan et al.* (2020), and Spanish by *Cañete et al.* (2020).

They got relative WER reductions ranging from 21% to 24%. With low inference latency ranging from 14ms to 29ms, the models are fast enough to be suitable for situations such as live streaming. Their inference latency is comparable to that of FastCorrect (*Leng et al.*, 2021a).

Leng et al. (2021b) later created "FastCorrect2" which showed improvements over the previous iteration, FastCorrect. Like FastCorrect, FastCorrect2 uses NAR sequence generation to increase speed. The ASR system used by the researchers, generates several candidates through beam search[2]. This fact, along with the *voting effect* is leveraged by FastCorrect2. The voting effect is when multiple candidate sentences created by the ASR system through beam search can verify the correctness of each other. If there are three sentences: "I have a cat", "I have a hat", "I have a bat", then it is likely that the first two tokens of each sentence are correct. The inconsistencies of the last token could indicate that this token needs to be corrected. They look for what is different across sentences that are otherwise similar. But sentences can vary in length, which is a non-trivial issue if one plans to leverage the voting effect. In this paper they introduce a novel alignment algorithm that tries to align sentences according to token similarity and pronunciation similarity to fix this issues. Compared to FastCorrect, FastCorrect2 shows a 2.55% and 3.22% improvement in terms of WER reduction on AISHELL-1 and the internal dataset respectively. It is 5 times faster than an autoregressive model. (*Leng et al.*, 2021b).

While speed is a crucial element in many settings where these ASR models are being used, their precision is arguably more important. Like any model, their capabilities are reflected in their training data. More data is more often than not, better for performance. A problem with many ASR models is that they don't model language well enough to be used in specific settings, where domain-specific words, such as words in a specific field of study, are not known to the model. This can be the case even if they are trained on a lot of data, that is general in nature.

Mani et al. (2020) showed that third-party ASR systems, such as those created by Google can be optimized for specific domains as a post-processing step if you have access to both the ASR hypothesis and reference text by using domain adaptation and machine translation. Domain adaptation is the process of making the ASR model more robust in certain situations where domain specific words are used. For example a medical consultation that uses medical terms *Enarvi et al.* (2020). They use machine translation to perform a mapping from out-of-domain ASR errors to in-domain terms in the reference text. This is different to our approach. For our case, we are not specifically aiming for domain adaption, focusing instead on using pre-trained language models as

[2] https://en.wikipedia.org/wiki/Beam_search

they are.

Hrinchuk et al. (2019) introduce an efficient post-processing model for correction of ASR outputs. They use a Transformer-based encoder-decoder architecture with the weights from a pre-trained BERT model. They use Jasper, a deep convolutional end-to-end ASR model, as a baseline *Li et al.* (2019). Using the LibriSpeech benchmark *Panayotov et al.* (2015), a corpus of read English speech, their methods show significant improvement in WER over the baseline acoustic model. They tested with various ways of initializing the weights for the model, either initializing them randomly, or using weights from BERT. Testing showed that using a decoder and an encoder both initialized with pre-trained BERT weights from BERT (base, uncased) gave the best results on word error rate.

Post-processing ASR errors means that the text in question has to be fixed. But some researchers are trying to use more than just the text, adding more information by adding additional features. *Du et al.* (2022) propose a cross-modal post-processing system that uses both acoustic and textual features to correct ASR errors. The authors use a model that is based on standard transformer blocks. Before combining acoustic embeddings together with textual features, their model does not perform better on CER (Character Error Recognition) than a BERT model for Chinese, called bert-base-chinese.[3]. After adding acoustic embeddings, their model gets nearly the same CER with 8.2 times the inference speed over BERT. For both single-speaker and multi-speaker speech they got a 10% relative reduction of character error rate, with a latency of about 1.7ms for each token. Thus the system they have created does increase performance slightly while increasing speed over a regular BERT model.

In a similar way to *Du et al.* (2022), *Xu et al.* (2021) tried to do spell checking for Chinese by using the multi-modal information of Chinese characters. Namely: semantic, phonetic, and graphic. This idea, to use more modalities in addition to the raw text to convey context, is probably worth investigating further. For ASR, there is usually texting of vision-based media. Perhaps one could use the information contained within the video in conjunction with the textual ASR output. A cooking show looks different than a show about sports, and shows different contexts. TV 2, with whom we are collaborating with, could possibly use this in their production. We will not attempt this approach, but acknowledge its potential usage here.

DHaro and Banchs (2016) notes that one of the main challenges when working with domain-independent ASR systems is to transcribe out-of-vocabulary words or very rare words. They tried to solve the problem of improving ASR outputs with regards to rare or out-of-vocabulary words, without changing the ASR model itself. We can see this happening in Norwegian when there are no good translations and the speakers chooses to use an English word, which confuses the ASR model. They point out that a common solution to this problem is to adapt the model in order to increase the vocabulary. In some cases, as with this thesis, the problem is that the ASR model is a third-party model that is not possible to change. In their research the authors use machine translation to solve the problem of ASR errors. Their solution is a system that first gets the n-best candidates from an ASR model, secondly, they use a translation model on each candidate to correct them, thirdly, they re-rank the n-candidates after being fed to the translation model. They showed improvements in both WER and character error rate

[3]https://huggingface.co/bert-base-chinese

(CER), with a relative reduction of up to 8% in WER and 7% on CER.

Continuing the discussion on out-of-vocabulary words and domain-specific speech-recognition systems. Some researchers want to solve the issue by creating better domain-invariant speech recognition systems. For example, *Narayanan et al.* (2018) trained a single model with data from different domains, showing that it worked as well as domain-specific models. The data consists of 162000 hours of speech that is a collection of data that is made up of several different sources. Creating a huge dataset from multiple sources is also what *Chan et al.* (2021) did, creating a model called SpeechStew. The multidomain model showed better generalization properties than domain-specific models.

ASR can be used in more critical situations such as during a consultation between a doctor and their patient, where precision and speed matters. ASR resolves the issue of a doctor having to take notes manually, reducing errors. Still, clinical dialogues is a specific domain, and this makes ASR errors more common. *Nanayakkara et al.* (2022b) presents a seq2seq learning approach to correct ASR transcriptions of clinical dialogue that contains errors. The two seq2seq models used are T5 and BART (*Raffel et al.*, 2019), (*Lewis et al.*, 2019). They use public domain medical data to fine-tune the models. They found that fine-tuning a seq2seq model on a mask-filling task leads to better WER scores than fine-tuning on either a summarising task or a paraphrasing task. Comparing with four commercial ASR systems, with this technique, they outperformed three out of four. The four were: AWS Transcribe, Microsoft, IBM Watson, and Google. BART-base model got the lowest word error rate when fine-tuned on a mask-filling task.

Dutta et al. (2022) say that "the outputs of an ASR system are largely prone to phonetic and spelling errors". This makes sense, as similar sounding words, or homophones, are ambiguous. In their research, they fine-tune a pre-trained sequence-to-sequence model called BART (Bidirectional Auto-Regressive Transformer) *Lewis et al.* (2019) to act as a denoising model to improve ASR output. Their approach involves using raw ASR output together with its corresponding phoneme representation to correct errors. Fine-tuning was done using ASR predictions along with the reference transcriptions. They used two variants: with and without including a phoneme representation on the predicted sequences. To introduce errors into the dataset, rather than using ASR systems, they synthetically introduce them using phonetically similar words. Synthesised word errors were created using the SoundsLike python package[4] to find, for a given word, a list of words with the exact same pronunciation and randomly selects a word as a replacement. This is similar to how we are introducing errors. See subsection 4.3.1 in methods. We are using Levenhstein distance to find the most similar words. We could have used Soundslike, but it doesn't have support for any language except for English.

Similar to our task of finding catastrophic words in text, *Sabry et al.* (2022) used T5 (*Raffel et al.*, 2019), which uses the Transformer architecture, for hate language detection. Results showed that T5 outperformed a LSTM (long short-term memory) model (*Hochreiter and Schmidhuber*, 1997), CNN model (convolutional neural network), and a RoBERTa model (*Liu et al.*, 2019). Using data augmentation provided performance gains. Likewise, HateBERT created by *Caselli et al.* (2020a), was trained

[4] https://pypi.org/project/SoundsLike/

on a large dataset called RAL-E (Reddit Abusive Language English dataset) (*Caselli et al.*, 2020a). The RAL-E dataset is made up of comments from Reddit communities banned for being offensive, abusive, or hateful. Results on three datasets, respectively for offensive (*Zampieri et al.*, 2019), abusive (*Caselli et al.*, 2020b), and hateful *Basile et al.* (2019) language, showed that HateBERT consistently outperformed a generic BERT model in detecting each type of phenomena.

Meripo and Konam (2022) propose an end-to-end approach for ASR error detection using audio-transcript entailment. They state that "there should be a bidirectional entailment between audio and transcript when there is no recognition error and vice versa". They use an acoustic encoder and a linguistic encoder to model speech and transcript, respectively. They experiment using HuBERT (*Hsu et al.*, 2021) and Wav2Vec (*Baevski et al.*, 2020) as the acoustic encoder, and BERT as the linguistic encoder. Their method thus involves encoding speech features from audio, and linguistic features from transcript hypotheses, and then compare the two encoded representations to see if they differ or not. If they entail each other, there are no errors. They achieve classification error rates (CER) of 26.2% on transcription errors, and 23% on medical error (specific domain).

Anantaram et al. (2018) showed improvements in word recognition rate (WRR) in attempting to improve the performance of a general purpose automatic speech recognizer (gpASR), such as those made by Google, to handle specific domains and noisy environments as well as account for speaker accents. Their repair process was motivated by Evolutionary Development (Evo-Devo) processes in biology. They treat an inaccurate ASR output as an injured biological cell. They propose a repair mechanism that is able to gainfully repair the output of a gpASR in a way that its accuracy can be improved significantly for domain specific applications. The method "combines bio-inspired artificial development (ArtDev) with machine learning (ML) approaches to repair the output of a gpASR".

One aspect to highlight with ASR error detection is that there are, in many cases, not many errors to begin with. Word error rates are usually low. For a given sequence of words, most of them do not have to be detected or corrected. If an ASR system has a WER of 10%, only 10% of the words in the sequence need to change. This is different from neural machine translation tasks, where in most cases all words need to be changed because we are translating from one language to another (*Leng et al.*, 2021b). *Shen et al.* (2022) presents *MaskCorrect*, a framework that alleviates what they call the "heavy copy phenomenon", where most of the tokens in a transcript are trivially copied. They point out that prior error correction methods take the incorrect sentence and gives it to the encoder and generate the target (correct) sentence through the decoder. Because of usually low WER scores, models learn to correct on limited tokens, and trivially copy on the other correct tokens which according to the authors harms the training of error correction. They solve the issue by masking a certain percentage of the correct tokens. Masking 15% of the tokens gave the best scores on WER, and WERR (Word Error Reduction Rate).

Error detection can be done via alignments between the target sentence and the reference sentence. This can, according to *Leng et al.* (2022) be done either implicitly or explicitly. Explicit error detection can be done by aligning the target hypothesis with the reference sentence together with the edit distance. This makes it explicit what tokens are correctly transcribed and which are not. Explicit alignment was for example

used in FastCorrect1 (*Leng et al.*, 2021a). As for implicit error detection, the authors point to transformer based autoregressive models, where target and source sentences are embedded using decoder-encoder attention. They also point to models based on connectionist temporal classification (CTC) that leverages a CTC loss to align the target with the duplicated source implicitly. The authors state that both implicit and explicit methods have their faults. Implicit detection does not give a clear interpretation of which tokens are incorrect, and explicit detection has low detection accuracy.

Leng et al. (2022) presents *SoftCorrect*, which uses a soft error detection mechanism to avoid the limitations of both explicit and implicit error detection methods. They use multiple ASR candidates from a beam search, or, the voting effect as was presented in *Leng et al.* (2021b). The encoder is used for error detection, calculating probabilities for each token in each candidate. The decoder is used for focused error correction. Compared to FastCorrect1 and FastCorrect2 that uses edit distance (explicit detection), they achieve better character error reduction rate (CERR) while still achieving low latency.

2.4 Transformer

Within natural language processing, the transformer is the current state-of-the art architecture. This architecture have been proven to be better than previous models on tasks such as machine translation, achieving higher BLEU scores than previous state-of-the art models (*Vaswani et al.*, 2017). After their release in 2017, other architectures based around Transformers have been released. One such architecture is called BERT (*Devlin et al.*, 2018).

2.4.1 BERT

BERT stands for **B**idirectional **E**ncoder **R**epresentations from **T**ransformers. It was created by *Devlin et al.* (2018). It is an architecture based on the original Transformer architecture by *Vaswani et al.* (2017). At the time of release, BERT advanced the state-of-the art for eleven NLP tasks, increasing the GLUE score to 80.5% (*Wang et al.*, 2018). The way BERT is different from the base Transformer architecture is that it looks at context in both the left and right direction from a given token, therefore we say that it is *bidirectional*. This gives it a richer understanding of the input text.

In addition, BERT uses a "masked language modeling" task (**MLM**) in pre-training, where a set of tokens are hidden from the model, and a guess has to be made as to which token could be behind the mask. 15% of all tokens were masked and had to be predicted by the model. By looking at both the left and right context, a token can be adequately guessed. This is reminiscent of a Cloze task, according to *Devlin et al.* (2018). This is a task where a participant is asked to fill in words in a sequence where one or more words are left hidden (*Taylor*, 1953).

After the release of BERT, which was trained on English data, a plethora of new multilingual and monolingual models for other languages than English have been trained and released. As we focus on the Norwegian language, we present in the following sections the Norwegian BERT models that we use.

2.5 Norwegian BERT models

In this thesis, the main goal is to find a way to detect errors in speech-to-text transcripts using pre-trained language models for Norwegian. We want to research the capabilities of these models for this specific use case. See research questions in section 1.2. We want to focus on post-processing of transcripts in Norwegian with words that have been wrongly transcribed. Given this, we wish to research the capabilities of three Norwegian BERT models: NB-BERT (*Kummervold et al.*, 2021), NorBERT1, and NorBERT2 (*Kutuzov et al.*, 2021). All three of these language models were created in recent years, and are all based on the original BERT architecture by *Devlin et al.* (2018).Prior to NB-BERT, NorBERT1, and NorBERT2, the language model most appropriate for the task at hand would probably have been a multilingual model such as mBERT (*Devlin et al.*, 2018).

NB-BERT was created by the AI lab at the National Library of Norway (*Kummervold et al.*, 2021). It is one of the best performing models on Norwegian and other Scandinavian languages today. It outperformed mBERT in tasks such as token and sentence classification, named entity recognition, and part-of-speech tagging for both Bokmål and Nynorsk (*Kummervold et al.*, 2021).

The model is trained on 109GB (18,438 million words) of raw deduplicated text. While the size of the dataset is impressive, it does have a lot of noise. This is mainly because of the use of optical character recognition (OCR) technology used in digitizing the books subset of the data. They even discarded data digitized between 2006 and 2008 due to poor quality. They estimate that 84% of the texts are in Bokmål, 14% is in Nynorsk, close to 4% is in English, and the remaining 1% is a mixture of Sami, Danish, Swedish, and a couple of other languages (*Kummervold et al.*, 2021).

The authors tried to create NB-BERT so that it would do well on all types of Norwegian language tasks, from old texts to modern texts with other languages such as English mixed in. They therefore chose to initialize the model with the pre-trained weights from mBERT. mBERT has a vocabulary of 119547 tokens. NB-BERT was able to outperform mBERT and NorBERT1 on tasks such as named entity recognition, part-of-speech tagging and sentiment analysis (*Kummervold et al.*, 2021).

The researchers attribute the performance of NB-BERT in large part due to the size of the corpus it is trained on. They recognize the fact that OCR errors have crept into the corpus. They conclude that they did not see any indication that this negatively impacted performance. They do recognize that further study should be done to be sure of the effects that noise created by OCR errors have on model performance. They also note that it would be interesting to use an all Norwegian vocabulary instead of using the multilingual mBERT vocabulary (*Kummervold et al.*, 2021).

In addition to NB-BERT there is also NorBERT1 and NorBERT2, two models that have come out of the NorLM initiative from the Language Technology Group at the University of Oslo. Prior to the release of NorBERT1, there were several independently released monolingual BERT-based language models for a number of languages such as Swedish, Vietnamese, Persian, Greek, Dutch, and Finnish. (*Malmsten et al.*, 2020), (*Nguyen and Nguyen*, 2020), (*Farahani et al.*, 2020), (*Koutsikakis et al.*, 2020), (*de Vries et al.*, 2019), (*Virtanen et al.*, 2019). Of these previous research efforts, the most important monolingual language model that both the NorBERT models build

upon is the Finnish model, FinBERT by *Virtanen et al.* (2019) as the training setup for NorBERT1 and NorBERT2 builds heavily on this. Before NorBERT1 and NorBERT2, the only language model that could be used for Norwegian was Google's mBERT. NorBERT1 was created roughly at the same time as NB-BERT in 2021, while NorBERT2 was released in 2022.

NorBERT1 was trained on The Norsk Aviskorpus, Bokmål Wikipedia, and Nynorsk Wikipedia which comprises about two billion word tokens in 203 million sentences both in Bokmål and in Nynorsk. NorBERT2 was trained on the Norwegian Colossal Corpus which is 5 billion words, and the C4 web-crawled corpus (Norwegian part), which is a random sample of about 9.5 billion words.[5] NorBERT1 and NorBERT2 both uses a custom WordPiece vocabulary, which stands in contrast to that of NB-BERT, which uses the same vocabulary as mBERT (*Wu et al.*, 2016). WordPiece has better coverage of the Norwegian language, according to the authors. The vocabulary is 30000 words and 50000 word in size respectively for NorBERT1 and NorBERT2, while mBERT's (and thus NB-BERT) vocabulary is roughly 120000 words in size. The lower sized vocabulary of the NorBERT models is of course compensated for by the fact that it is almost entirely composed of Norwegian words. In addition, NorBERT1 includes accented characters in its vocabulary (*Kutuzov et al.*, 2021).

NorBERT1 outperforms mBERT on Binary Sentiment Classification. NB-BERT outperforms NorBERT1 most likely due to the immense amount of training data. But NorBERT1 does outperform NB-BERT on some NLP tasks. The authors believe that this is because NorBERT1 is trained on cleaner data. The task it does perform better on is fine-grained sentiment analysis. While NorBERT1 does not beat NB-BERT in most cases, it does outperform mBERT (*Kutuzov et al.*, 2021).

Norwegian does not have a lot of resources when it comes to language data when compared to languages with drastically more native speakers, such as Chinese, English, or Spanish. It is a so-called mid-resource language. The lack of data is simply due to the fact that there are not many people that use Norwegian (roughly five and a half million) compared to these other larger languages. Given this, the previously mentioned NB-BERT and NorBERT language models are a great contribution for researchers working on Norwegian NLP tasks, as we are doing in this thesis. The work that is being done at the National Library to increase the amount of publicly available language resources available is also commendable.

[5] http://wiki.nlpl.eu/Vectors/norlm/norbert

Chapter 3

Data

This chapter will provide details on two datasets: the Norwegian Parliamentary Speech Corpus (NPSC) created by *Solberg and Ortiz* (2022), and a lexicon of bad words, Hurtlex, created by *Bassignana et al.* (2018). We will provide our reasoning for choosing these datasets, their strength and weaknesses, as well as their intended use in the thesis.

The main goal of this thesis is to identify problematic words in transcripts created automatically by ASR models, using language models to detect such errors and increase the transcript quality. The task is a classification problem: either a word in a sentence is problematic or it is not.

In order for us to solve this problem, we needed a lot of transcription data of high quality that also had errors in it, created by ASR models. Unfortunately, we did not manage to find a dataset of transcripts with enough errors, as they are not very common. As far as we know, there have been no attempts to create such a specific dataset for Norwegian.

We decided to create our own augmented dataset, using an open-source transcribed dataset, the NPSC, into which we will introduce errors from a lexicon of bad words, Hurtlex. We consider this new augmented dataset as silver data, with which we will evaluate our models.

The point of using the NPSC together with Hurtlex is to try and mimick ASR errors as they sometimes occur when using ASR models to transcribe speech. We chose the data from the Norwegian parliament because it is annotated by experts, ensuring a high level of quality. In addition it is large in size and includes many different speakers from Norway. Members of parliament necessarily include people from all over the country, which gives good coverage of dialects in the country. For these reasons, it is a good dataset for many tasks within natural language processing, and thus also for this thesis. The Hurtlex lexicon is also annotated well enough to be of practical use.

3.1 Norwegian Parliamentary Speech Corpus

The Norwegian Parliamentary Speech Corpus (NPSC) was created by the Norwegian Language Bank[1] at The National Library of Norway between 2019 and 2021. The NPSC consists of recordings of speeches from the Norwegian parliament (Stortinget)

[1] https://www.nb.no/sprakbanken/en/sprakbanken

from entire days of plenary meetings between 2017 and 2018, and transcriptions in both Norwegian Bokmål and Norwegian Nynorsk, in addition to metadata about all speakers. All transcriptions were created and proofread manually by trained linguists and philologists (*Solberg and Ortiz*, 2022).

The NPSC is a good dataset to use for this thesis as we want a lot of high quality, textual data. The data is primarily intended as an open-source dataset for ASR development, which makes it a good fit for this thesis (*Solberg and Ortiz*, 2022). In addition to being open-source, it can also be used with no copyright restrictions.

What follows is a description of how the NPSC transcripts were created following these steps:

Audio files of the speeches were run through Googles Cloud Speech-to-Text service, creating an automatically transcribed text in Norwegian Bokmål.

Then the official proceedings text and ASR outputs were compared sequence by sequence using Levenshtein distance[2] (*Jurafsky and Martin*, 2021).

If a sequence in the proceedings and a sequence in the ASR output were highly similar but not identical, then the proceedings sequence was taken to be the correct one. The words in the ASR sequence were then swapped with the corresponding ones in the proceedings text (*Solberg and Ortiz*, 2022).

Transcribers working at the Language Bank (Nasjonalbiblioteket) then looked through the transcriptions while listening to the audio, correcting them where necessary. After this, a second transcriber looked through the transcription while listening to the audio (*Solberg and Ortiz*, 2022).

Members of parliament were cited based on their written Norwegian form, meaning either Bokmål or Nynorsk. The transcribers follow certain conventions. They aim for (1) consistency, meaning that similar linguistic phenomena are treated similarly across transcriptions. (2) Standardized orthography should be followed whenever possible. (3) Faithful rendering of speech. The transcriptions should render the pronunciation as faithfully as the orthographic convention allows for. (4) Flagging of non-standard speech. If the speech deviates significantly from the written standard, then the transcribers flag this and give a standardized semantic equivalent (*Solberg and Ortiz*, 2022).

3.1.1 Structure of the NPSC

The NPSC is structured into different files which are either in JSON or TXT format. The JSON files have mostly the same content with varying degrees of details. There are files for the tokens in the data, both normalized and not, as well as normalized and non-normalized sentences. The TXT files mostly just have the raw textual data with date and speaker information. The JSON files contain metadata such as which split it belongs to, names of speakers and their language (Bokmål or Nynorsk).

Table 3.2 shows that the data is split into an 80-10-10 percent split (training, evaluation, and test). Three features were made similar across splits to keep them balanced: the percentage of Nynorsk, the percentage of female speakers, and the average word length per sentence (*Solberg and Ortiz*, 2022). Table 3.1 shows the total number of **tokens**, which are the total number of words, and the **types**, which are all unique words.

[2]Which is a way of finding out how many edit operations (deletions, substitutions, insertions) one has to do to get from one sequence to another.

Total tokens	Total types
1088934	45035

Table 3.1: The total number of tokens and types in the NPSC. Tokens are all the words in the dataset, while types are all the unique words in the dataset.

	Sentences	Nynorsk	Sentence length (words)
Train	45470	12.8%	18.7
Evaluation	6844	12.7%	18
Test	6355	13%	18
Total	58669	12.8%	18.6

Table 3.2: NPSC split statistics. Total number of sentences, percentage of Nynorsk and average word length per sentence for the train, evaluation, and test set.

For this thesis, we wanted to create documents that were longer than the sentences in the dataset. We did this in order to increase the context around every word because BERT models can more accurately predict a word when there are more words to the left and right of it. We had to make sure that the document sizes did not extend past the window size of BERT which is 512 tokens (*Devlin et al.*, 2018). We preprocessed the data so that sentences that were spoken by the same speaker continuously were merged together into larger documents. When a new speaker started talking, that counted as the start of a new document and the end of the previous document. See the preprocessing section (4.1) of chapter 4 (methods) for details on preprocessing of the NPSC dataset.

3.2 Hurtlex

Hurtlex is a multilingual lexicon that consists of offensive, aggressive, and hateful words in over 50 languages[3] (*Bassignana et al.*, 2018). This lexicon has its roots in the lexicon "Le parole per ferire" created by the Italian linguist Tullio De Mauro. Hurtlex can be used as a resource to analyze and detect hate speech in social media texts, also in a multilingual perspective.

To create Hurtlex, the authors first extracted every word from the previously mentioned Italian lexicon. They extracted 1138 words with 1082 being unique because some were duplicated in multiple categories. Secondly they use the Italian index of MultiWordNet[4] to add all possible part-of-speech for 59.2% of lemmas, annotating the rest manually. Thirdly, they used BabelNets API to link the lemmas of the lexicon with a definition[5]. In total they found definitions for 71.1% of the lemmas, meaning that 28.9% of the lemmas did not have a definition on BabelNet. See table 3.4 for percentage of lemmas from each category not found on BabelNet.

The authors further used BabelNet to translate the lexicon into multiple languages

[3] https://github.com/valeriobasile/hurtlex
[4] https://multiwordnet.fbk.eu/english/home.php
[5] https://babelnet.org/guide#java

Category	Description	Word Count
CDS	Derogatory words	890
AN	Animals	500
QAS	With potential negative connotations	240
DMC	Moral and behavioral defects	209
RE	Felonies and words related to crime and immoral behavior	200
DDP	Cognitive disabilities and diversity	185
PS	Negative stereotypes ethnic slurs	160
SVP	Words related to the seven deadly sins of the Christian tradition	145
ASM	Male genitalia	142
OM	Words related to homosexuality	123
PR	Words related to prostitution	107
PA	Professions and occupations	97
OR	Plants	66
ASF	Female genitalia	56
IS	Words related to social and economic disadvantage	43
DDF	Physical disabilities and diversity	27
RCI	Locations and demonyms	8

Table 3.3: The 17 word categories in Hurtlex with the total number of words in each category.

Category	Percentage
CDS	26.07%
AN	10.07%
QAS	11.03%
DMC	7.45%
RE	4.69%
DDP	8.55%
PS	2.76%
SVP	6.07%
ASM	6.21%
OM	2.76%
PR	1.66%
PA	5.38%
OR	2.34%
ASF	1.66%
IS	1.38%
DDF	1.52%
RCI	0.41%

Table 3.4: Percentage of lemmas from each category not found on BabelNet.

by retrieving all senses of all the words in the lexicon. They then used BabelNet again to get all lemmas in all the supported languages.

Some of the senses from the first step were unrelated to the offensive context, and thus translating them to other languages would generate unlikely candidates for a lexicon of hateful words. They performed a manual filtering of senses before automatic translation by annotating each pair of lemma/sense as either: Not offensive, neutral, or offensive. To check the consistency of annotations a subset of 200 pairs were annotated by two experts, reporting an agreement on 87.6% of the 200 pairs. Next they decided to split the neutral class into two classes which covers senses which are **not literally pejorative** but can be used to insult through semantic shift (metaphorically). The other class is for senses which have a clear **negative connotation** but is not necessarily directly derogatory (the word "criminal" was an example they used for this class).

Finally, they ended up with two different versions of the lexicon for all 53 languages, with one containing only translations of "offensive" senses (conservative), and the other with translations of "offensive", "not literally pejorative" and "negative connotation" senses (inclusive). Every word in the conservative version is also in the inclusive version, but not vice versa. All senses marked "not offensive" were discarded. See table 3.5 for amounts of words labeled as conservative and inclusive. See also table 3.3 which shows how words were classified at a finer level of detail.

Level	Word Count
Inclusive	2005
Conservative	1193
Filtered	176

Table 3.5: Total number of words marked as either inclusive or conservative. Filtered words are the words we use in our experiments.

3.2.1 Hurtlex limitations and filtering

Many words in the Norwegian part of the Hurtlex data are not really bad in any sense of the word, and would not be useful to us, so they had to be removed. We relied on our intuition to decide what was a bad word and what was not a bad word. Many words are also the same, but have a different part-of-speech tags. To filter the lexicon we first chose to create a subset of it where only the words with the conservative tag were kept, as the inclusive tag contains mostly words that are not considered bad or offensive. Then, we looked at all the words in the conservative category, following basic rules. If a word was not Norwegian, or it is not bad, it is simply not kept. All other words are kept. An example of a word that is not kept is the word "fuck", because even if it is a bad word, it is not Norwegian. Another example is the word "tull", which is Norwegian, but would not be considered bad or catastrophic by native Norwegian speakers. After manually filtering the lexicon we were left with 176 Norwegian words. Table 3.5 shows the total number of words after filtration compared to the two initial levels that words belonged to.

Another limitation with the data is that almost all the words are nouns. When introducing errors, it would possibly be unhelpful to change a word in a sentence that is not a noun. It could be bad because then the structure of the sentence is then changed, making the error less natural.

3.3 Intended use of the data

Our goal is to research if Norwegian pre-trained language models can be used as they are to find errors in ASR transcripts, without fine-tuning them. Even if they are pre-trained on large amounts of data, they do not necessarily handle specific use cases very well. Domain specific terms and out-of-vocabulary words are typical problems.

To find out whether or not this is the case we have to train and test our models on ASR transcripts with errors. We intended to use transcripts from TV2, but these did not have enough errors. We will introduce errors artificially into the NPSC from Hurtlex to fix this problem.

Next, in chapter 4 (methods), we will show how we use the dataset with errors to find the probabilities for the words as well as the bad words that will be inserted. We also explain how we insert errors. Later, we will analyze these probabilities to understand how the models performed. Results are shown in chapter 5.

Chapter 4

Methods

This chapter describes the methods we have used to answer our research questions. To begin with, we show how we preprocessed the NPSC dataset and the lexicon of bad words, Hurtlex, to better serve our purpose, as they were not sufficient as they were. See section 3.2.1 for a reminder of how Hurtlex was filtered. We then show how the BERT models were used to compute probabilities of good and bad words in the documents of our dataset. We then describe how bad words were inserted into the documents, while highlighting the challenges that presented themselves to us, and how we chose to overcome these. Finally, we show how we classified the words as either good or bad using logistic regression, based on the probabilities computed earlier.

4.1 Preprocessing of the NPSC

The transcribed speech data from the NPSC is good to use for this thesis because it is both large enough in size and is created by qualified experts, ensuring its quality. However, there was one important issue with it that had to be addressed. The sentences in the dataset were not long enough to best serve our purpose, which is to detect bad words by computing probabilities of a word belonging in a certain position in a sentence. Short sentences are problematic, as they contain little contextual information. When we compute the probability of a word being in a sentence using BERT, the probability of the word is computed based on the entire sentence that the word is in. Using shorter sentences, we can end up with a situation where correct words are given equally low probabilities as the bad words, which would not help us identifying problematic bad words. See table 4.1 for average lengths in the training, evaluation, and test set before we did any preprocessing.

BERT models have a context window with a size of maximum 512 tokens (*Devlin et al.*, 2018). A bigram model for comparison has a context window that consists only of two words.

To deal with the sentences being too short, we decided to create longer documents from the sentences in the dataset. From here on, we refer to *documents* as the results of preprocessing the NPSC data, and when we talk about *sentences* we are talking about the NPSC data before we did any preprocessing. In order to create these longer documents we had to thus preprocess the NPSC data in order to provide more context to the BERT models. Because we have to work on the token level for this, we will in

Dataset	Sentences	Sentence average word length
Train	45470	18.9
Evaluation	6844	18
Test	6355	18

Table 4.1: Total number of sentences and average lengths before preprocessing the NPSC dataset.

Tokenizer	Sentence tokenized	number of of tokens
NB-BERT	['[CLS]', 'Stortinget', '##s', 'm', '##øte', 'er', 'lov', '##lig', 'satt', '.', '[SEP]']	11
NorBERT1	['[CLS]', 'Stortingets', 'møte', 'er', 'lovlig', 'satt', '.', '[SEP]']	8
NorBERT2	['[CLS]', 'Stortingets', 'møte', 'er', 'lovlig', 'satt', '.', '[SEP]']	8

Table 4.2: An example showing how NB-BERT, NorBERT1, and NorBERT2 tokenize the sentence: "Stortingets møte er lovlig satt".

what follows give an explanation of what tokens and tokenizers are.

A token is a linguistic unit that can be a word. A token can consist of sub-tokens, which are sub-parts of tokens. Tokenization is the task of segmenting text sequences, such as a sentence, into its running words, which is done using a tokenizer. Tokenizers are an essential part of language modeling. Tokenizers have two parts: a token learner and a token segmenter. The token learner gets the raw training data and creates a set of tokens, also known as the vocabulary. The new sequences can then be inserted into the token segmenter which separates it into tokens from the vocabulary (*Jurafsky and Martin*, 2021).

For the example sentence: "*Stortingets møte er lovlig satt*", the words get tokenized by the tokenizers of NB-BERT, NorBERT1 and NorBERT2 as shown in table 4.2. Further, this gets encoded as show in table 4.3. Note that the tokens 102 and 103 correspond to the special tokens `[CLS]` and `[SEP]`. `[CLS]` marks the beginning of a sequence like a sentence, and the `[SEP]` token is used to separate sequences to keep track of where one sequence stops and another one begins. It is this encoded text that ultimately gets fed into the models in order to compute probabilities later.

In this thesis, we preprocess the data by combining smaller, individual pieces of text, the **sentences**, together to create longer texts, the **documents**. What follows is a description of how we preprocessed the sentences in the NPSC. The data is structured in a way that allows us to easily create our desired documents.

The NPSC is structured as JSON files with the individual sentences and metadata related to each sentence in self-contained JSON objects (see the data chapter, Chapter 3, for other details). The sentences are of course the most important part of the dataset, but the metadata proved to be necessary for preprocessing the sentences properly. All the texts in the dataset are single sentences spoken by a single member of the parliament during a speech, and each sentence with metadata is found in its own separate JSON

Tokenizer	Sentence encoded	Number of of tokens
NB-BERT	[101, 61084, 10107, 181, 88869, 10163, 49950, 12554, 35293, 119, 102]	11
NorBERT1	[102, 10848, 30259, 3638, 17933, 2415, 26273, 103]	8
NorBERT2	[102, 8710, 1646, 146, 11842, 1806, 737, 103]	8

Table 4.3: An example showing how NB-BERT, NorBERT1, and NorBERT2 encode the sentence: "Stortingets møte er lovlig satt".

object. The NPSC is chronologically structured in a way that all sentences that belong to a given speech follow each other in the dataset, even though they are separate JSON objects. The metadata for each sentence also contains the start and end time for a given sentence for a given speaker, which makes it possible to create longer documents from shorter sentences by keeping track of when a speech starts and ends, which is necessary to keep track of the context. When someone new starts talking that is considered to be a new document and a new context.

The goal we wanted to achieve by preprocessing the data was to get longer documents, in order to increase and enlarge the context of each sentence for the models. As previously mentioned, the maximum amount of tokens allowed to be inserted at a time into all of the models is 512 (*Devlin et al.*, 2018). This is an absolute limit that we can't go past as we have no way of increasing it. Documents can't be longer than this. We thus have an upper limit defined.

However, we had to lower this threshold down to 500 tokens because we at a later stage want to replace words in the documents with a bad word from Hurtlex, and had to take measures to avoid the situation where a single bad word inserted increases the size beyond the 512 tokens. We chose 12 tokens as a buffer, but we arguably could have experimented with different thresholds.

We decided that the lower threshold should be 100 tokens as we felt that this would serve as a good limit that would allow for more context for each document while keeping the amount of documents high.

To create a new document with our now defined limits, we start with an empty string, then add sentences that follow each other from a speech, always checking that the same speaker is in the metadata for the next sentence as the one before it, to ensure that they are part of the same speech. Before adding a sentence to a current document, we tokenize the sentence, and then do a simple check where we count the tokens that make up the sentence to be added to the document. If adding the amount of tokens that sentence contains does not exceed the length of the upper limit of 500, we add the sentence to the document. If adding it does exceed the upper limit, we simply finish creating the current document, and use the non-added sentence as the start of the next document, and so on for all remaining sentences. As an example: if a current document is made up of 400 tokens, and the next sentence to be added is made up of 101 tokens, we stop the creation of the current document at 400 tokens, and start a new document with the one that has 101 tokens.

A subset of sentences will necessarily be lost because of the lower limit. However,

Dataset	Documents	Document average word length
Train	3882	207.4
Evaluation	578	199.75
Test	571	188.5

Table 4.4: Total number of documents and average lengths after preprocessing the dataset. NB-BERT used to count tokens.

as previously argued, we believe that shorter sentences have less context and therefore might be problematic when trying to identify probabilities. These sentences are less than 100 tokens, and they are not part of a longer speech. Although, they are part of a larger parliamentary hearing, which can be regarded as a larger context. These sentences are for the most part intermediary sentences used by the president of the parliament to introduce a new member of parliament when it is their turn to speak. For the training data, there were about 8% of sentences that were lost due to this. This does not affect our results as we will only be using half of the training data. For the test and evaluation set there were respectively 0.08% and 0.074% of sentences that were lost.

We decided to use NB-BERT's tokenizer when checking for token counts, because this tokenizer will tokenize most Norwegian words into more subwords than NorBERT1 and NorBERT2, since it uses a multilingual tokenizer (mBERT). This way we make it possible for all of the documents to be processed by all the models when finding probabilities.

In addition to the issue of making sure the documents were long enough, we also removed special characters in the sentences that were added by the annotators of NPSC. These are characters that mark speech phenomenons that do not correspond to words in text. These are: <*ee*> for vocalic hesitations, <*qq*> for coughs, <*mm*> for nasal hesitations, <*INAUDIBLE*> for inaudible or overlapping speech. These special tokens contain linguistic information that were not useful for our purposes and could potentially interfere with the BERT models' ability to understand the documents context. We therefore decided to remove them. We also removed trailing white spaces at the beginning and end of documents.

A document is thus defined as a set of sentences spoken by the same speaker that follow each other during the same continuous speech in a unique parliamentary session. If a new speaker starts talking, that is considered the end of the current document, and a new document is created starting from the first sentence spoken by the new speaker. Some entries in the original data is lost because of this, due to the current speaker not speaking long enough to make up at least the 100 tokens given as the lower limit for document lengths. To remind the reader, we chose 100 tokens as the lower limit to make sure that the documents had enough context, and to ensure that enough documents were created.

For each sentence, before we add it to a document, we check the number of tokens in the sentence. We do this using NB-BERTs tokenizer. To reiterate, we use NB-BERTs tokenizer here *only* to count the tokens. We do this because the tokens that make up a sentence is the input that will be used by the models when computing probabilities for individual words later on. If we were to use word lengths then that could possibly

Dataset	Documents	Document average word length
Train	3882	237.2
Evaluation	578	241.76
Test	571	213.2

Table 4.5: Table showing the total number of documents and average lengths after preprocessing the dataset if we had used NorBERT1 to count tokens.

Dataset	Documents	Document average word length
Train	3882	244.4
Evaluation	578	249.89
Test	571	218.76

Table 4.6: Table showing the total number of documents and average lengths after preprocessing the dataset if we had used NorBERT2 to count tokens.

result in errors because of the documents being too long when computing probabilities.

After preprocessing was done on all sentences from the test, evaluation, and training set, we were left with 3882 documents (initially 45470 sentences), as can be seen in table 4.4. Context has been increased from the shorter sentences that we had initially, as seen in table 4.1. For reference, one can see from table 4.5 and table 4.6 that the documents would have been longer if we used either the tokenizer of NorBERT1 or NorBERT2.

Due to time constraints, we decided to use a smaller selection of data from the training dataset. We created a random sample of 2000 documents from the 3882 documents. To create this sample, we simply used the built-in sample method from the pandas library[1]. We did not create a sample from the test and evaluation set as these were small to begin with. In the next section, we briefly present out baseline bigram model, and then in section 4.3 we show how we compute probabilities for all the documents in the preprocessed dataset.

4.2 Simple bigram baseline

A good way to analyze results is to have a baseline model to compare against. For the baseline model we used a simple bigram model. A bigram is defined as a two-word sequence of words such as “hello there”, or “thank you”. Bigram models are one of the most basic ways to model language. The probability of a word is based only on the previous word for each word in the entire sequence.

Using an n-gram model to find the probability of a word in a sentence thus means looking at the n-1 words before the given word, also called the history, that precede the word and asking for the probability of that word succeeding the history (*Jurafsky and Martin*, 2021). For a bigram model, for sentences longer than two words, we lose out

[1] https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.sample.html

on important contextual information. This stands in contrast to the BERT models that can see more context.

A problem with bigram models, as with BERT models, are unknown words. You could end up with a probability of zero if the word is unknown to the model. To solve this problem, we use what is called plus one smoothing. This means that for each word that is not seen by the model, we add one to the amount of times it has been seen to avoid the problem of the word having zero probability (*Jurafsky and Martin*, 2021).

In addition to the limitations regarding context and unknown words, the bigram model would only be trained on the NPSC dataset and nothing else, whereas NorBERT1, NorBERT2, and NB-BERT all are pre-trained on much larger datasets. To make the comparison more fair, we decided to not use the NPSC, and instead use a larger dataset to better train it. We use the Norsk Aviskorpus[2], which was part of NorBERT1's training data. The corpus contains around 1.68 billion words for Bokmål and around 68 million words for Nynorsk. The corpus consists of Norwegian newspapers released between 1998 and 2019.

4.3 Finding probabilities of words

The BERT models and the bigram model are used to assign probabilities to the words in the documents. We also assign probabilities to the bad words we insert into the document. This way, we can later analyze the probabilities that the models give to each word to find out how probable the models think all the words are in their respective document (context). Now that each word has a wider context surrounding it, we expect the good words to be given higher probabilities, and the bad words to be given lower probabilities. Next, we show how we computed the probabilities for words in the documents.

Computing a probability for a word in a document using a bigram model is straightforward. One simply has to look up the word given the previous word, and return its probability from the model. The probability is based on the amount of times the model has seen the current word appear after the previous word.

When using the BERT models to find probabilities, we need to tokenize the sentence into individual tokens. Then for each token, we mask only **one** token, leaving the rest of the tokens available to the model, and then we compute the probability of that token being in that specific position in the document.

After tokenizing a document, some of the words will remain as they were because they already appear in the vocabulary as that word. Other words will get tokenized into smaller tokens. Many tokens in the vocabulary of a model are whole, complete words, as they appear in natural languages such as Norwegian or English. Other tokens are what are called *subwords*, which are not words, but can be used to build words together with other subwords. In table 4.2, using NB-BERTs tokenizer, the word "Stortingets" gets tokenized to "Stortinget", and "##s". Here "##s" is a subword. These subwords are the smallest pieces of information that a language model use to represent languages. These can be arbitrary strings, or meaning-bearing units like a morpheme, as in the previous example (*Jurafsky and Martin*, 2021). If a word is made up of several subwords, the subwords are marked in a special way: BERT prepends them with two hashtags

[2] https://www.nb.no/sprakbanken/ressurskatalog/oai-nb-no-sbr-4/

Stortinget	##s	m	##øte	er	lov	##lig	satt
[MASK]	##s	m	##øte	er	lov	##lig	satt
Stortinget	[MASK]	m	##øte	er	lov	##lig	satt
Stortinget	##s	[MASK]	##øte	er	lov	##lig	satt
Stortinget	##s	m	[MASK]	er	lov	##lig	satt
Stortinget	##s	m	##øte	[MASK]	lov	##lig	satt
Stortinget	##s	m	##øte	er	[MASK]	##lig	satt
Stortinget	##s	m	##øte	er	lov	[MASK]	satt
Stortinget	##s	m	##øte	er	lov	##lig	[MASK]
Stortinget	##s	m	##øte	er	lov	##lig	satt

Table 4.7: Example sentence showing how each token is masked and how we compute probability of each token being in its position in the sentence.

(##) to distinguish them from other non-subword tokens. This will be important for keeping track of which subwords belong to a word later.

The vocabularies of state-of-the-art language models are usually quite large and covers many words. NorBERT1 and NorBERT2 have vocabularies of 30000 and 50000 tokens, respectively. NB-BERT has a vocabulary size of 119547 tokens, corresponding to mBERTs tokenizer, which covers more languages. NB-BERT has worse coverage of Norwegian than NorBERT1 and NorBERT2 (*Kummervold et al.*, 2021). This means that it more often than NorBERT1 and NorBERT2 tokenizes a word using subwords. All three use the WordPiece algorithm for tokenization (*Wu et al.*, 2016).

After we have tokenized the words, and before we can actually get the probabilities of a token, we have to mask all the tokens in the document one at at time. This is done using the special token "[MASK]", which is the same token used for all models, but note that this mask token can differ with other BERT models. We have to replace one token at a time with the mask, moving the mask one token over each time to compute the probability of every token. See table 4.7 for an example of moving the mask. To compute the probability of a single token we insert the whole document into to the model, only asking for the probability of the token of interest being hidden behind the mask. Then we do this for all the remaining tokens. This means that at runtime, while run the code, we have to insert a new, modified document with a new mask, for each token. We remind the reader that the input given to the model is the encoded tokens as shown in table 4.3.

There are two situations that can occur when finding probabilities for words: either the word gets tokenized to a single token, or several tokens. Getting probabilities for words that aren't composed of several subwords, *i.e.* words already in the model vocabulary, is straightforward. We ask the model for the probability of the word, then we make sure to keep the probability and the word coupled together in a list so that we can keep track of what words corresponds to what probability.

The other situation, where we are computing probabilities of subwords that make up a word, requires that we keep track of what subwords belong together to form a word while finding the probabilities for every subword. After we have found the probabilities for each subword, we multiply these probabilities together, giving us a single probability for the entire word based on the subwords. Now we have the probability of the word, but the word is broken up into subwords, and we have to detokenize the

subwords to get back to the word so that we can link it together with the probability.

There is no detokenization scheme for BERT that makes it possible to go from the subwords back to the initial word. We had to use simple Python string manipulation to create a detokenizer to get from the tokens that a word is made up of, to the word. Here we use the fact that subwords are prepended with the special characters (##) and the fact that the start token of a word does not have these special characters to construct the word back from the tokens, since we know where we started from.

This method of computing probabilities is inspired by the method found in *Salazar et al.* (2020). Other researchers such as *Nangia et al.* (2020) have used this strategy for finding token probabilities to score sentences on a bias task. *Salazar et al.* (2020) points out, as we have, that computing probabilities requires a sentence (in our case, document) copy, making the number of inference passes dependent on the length of the sentence. In addition, the cost of the Softmax function is also added, which is dependent on vocabulary size. The Softmax function is necessary to use in order to convert the output of the model into a probability between 0 and 1 that can more easily be analyzed later on. Thus the code we ran is fastest when using NorBERT1, and slowest when using NB-BERT, which has the largest vocabulary.

We computed probabilities in the same way with the training, evaluation, and test set, using all three BERT models as well as a baseline bigram model.

4.3.1 Inserting bad words

Since we do not expect that the transcriptions of the NPSC contain bad words, we have to construct a dataset where we introduce problematic words. In a real world scenario, where using output from text to speech systems, some few occurrences of bad words can be found. Because of time and resource limitations, we therefore create artificial NPSC examples where we introduce bad and problematic words. To this end, for each word `x` in a document, we insert a bad word into the position of word `x`, and compute the probability of this bad word. These bad words are from the Hurtlex lexicon that we manually filtered earlier (*Bassignana et al.*, 2018).

As with the reference (good) words we computed the probability for earlier, we have to create a new document for each bad word, and compute the probabilities of the tokens that the bad word is composed of.

The way we determine which bad word to insert is by using the string similarity metric Levenshtein distance[3] (*Jurafsky and Martin*, 2021). This metric computes the least amount of edit operations required to go from one word to another word. The operations are: insertion, deletion, and substitution. We want to find words that are similar, and this is what Levenshtein does, finds the minimally distant word, meaning the word that requires the least amount of any type of the three edit operations. There could be instances where the minimally distant word found using Levenshtein is not very similar, yet is the *most* similar. We could probably have experimented with using other distance metrics, but we chose Levenshtein.

[3]Which is a way of finding out how many edit operations (deletions, substitutions, insertions) one has to perform to get from one sequence (a word in our case) to another.

4.4 Detecting bad words

The main goal of this thesis is to explore if language models can reliably be used to detect the problematic words in text created by ASR model. In this context, a "detection" means that a given word in a sentence has a low enough probability of being in that sentence that it should count as a word that does not fit in the document.

We thought about methods to detect bad words, and thought a threshold could be a good solution. A threshold would be based on probabilities, and if a word is below the threshold, it should be classified as catastrophic.

Initially we thought about creating an absolute threshold, but, saying that a word has to be, for example, at least 30% likely to show up in a given document for any word in any document might be too strict and is also quite arbitrary. We could have created a threshold based on relative probabilities. The word with the lowest probability of being in a document could indicate that this word is wrongly transcribed. But even if a word has a low probability, this does not mean that it is wrongly transcribed. What we decided to do was to visualize the distribution of probabilities for good and bad words, to see if we can determine a threshold from this.

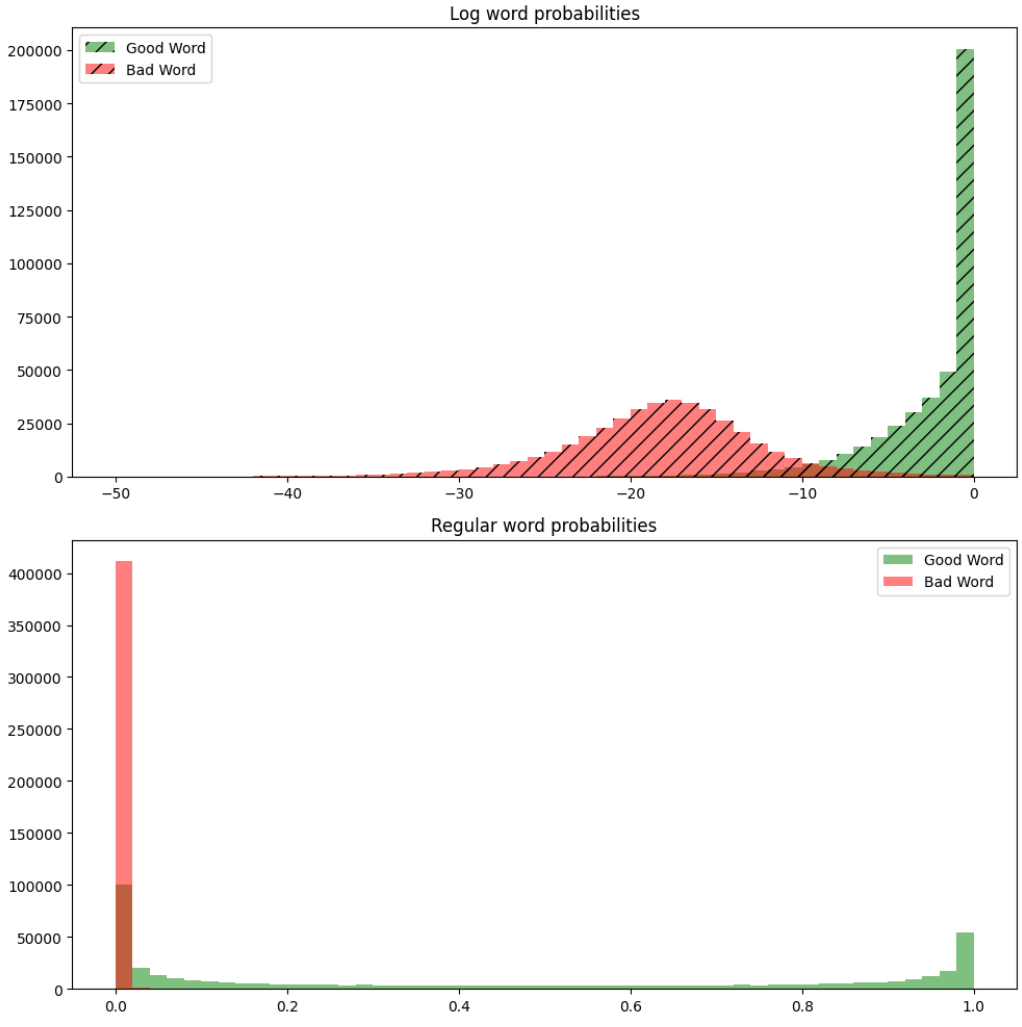


Figure 4.1: Distribution of regular and logarithmic probabilities for good words and bad words using NorBERT1 on the training set.

In figure 4.1 we can see regular and logarithmic probabilities obtained using NorBERT1 on the training set. We see that for regular probabilities a lot of them cluster around zero, both for the good words and especially the bad words. Note that this figure reflects the same pattern seen for NorBERT2 and NB-BERT as seen in Appendix A, which also shows the bigram probability distribution. This clustering makes it hard to interpret the numbers. In logarithmic space, it is easier to see the peaks of both good and bad words, and where they overlap. The graph using logarithms show that the BERT models are indeed able to distinguish bad words from good words.

We could have tried to create a threshold based on what we see in Figure 4.1 and in Appendix A. If we were to do this, we would want this threshold not to produce a lot of false positives, meaning labeling bad words as good. In the figure, this means moving the threshold further to the right, so that bad words have to have quite high probabilities to be counted as good. The trade-off then is that some good words with lower probabilities get labeled as bad (false negatives), but this is preferable to the opposite case.

However, we decided not to create a threshold purely based on the distribution in logarithmic space. Instead of creating a threshold for probabilities of bad words, we decided to train a logistic regression classifier on these logarithmic probabilities to automatically detect which word is out of place.

4.5 Classifying words with logistic regression

Now that probabilities have been computed, we should be able to automatically identify which words have low probability in a given sequence of words.

We use logistic regression to classify the words, and from this we can identify thresholds for detecting bad words. Logistic regression is a machine learning model that can be used to classify an instance of data into one out of two classes. In our case it is a classification of either “good word” or “bad word”.

Logistic regression uses features to classify given data points. We tested three different types of features. First, we ran the logistic regression classifier using only the word probabilities. Second, we used the probabilities together with the variance and the average probability of all the probabilities in a document, and subsequently used all three as parameters in the logistic regression model.

Using only the probabilities as features, we train the classifier using the probabilities in the training data, making sure to label all the bad words’ probabilities with -1 to indicate that they are bad, and a 1 for the good words’ probabilities. Then we use the trained classifier to predict the labels of the words in the testing and evaluation set, based only on their probability. Note that we are using the logarithmic probabilities as shown in figure 4.1 for this as these make it much easier for the classifier to distinguish very unlikely word from the rest.

For the second set of features we use the probabilities as before, and also the mean probability and the variance, and use these as a features for each word.

To make it clear to the reader, we are not feeding any words or documents into the classifier, we are only using probabilities, or probabilities together with mean and variance.

Chapter 5

Results and Discussion

In this chapter, we present the results of using the three BERT models: NB-BERT, NorBERT1, and NorBERT2, in addition to the results of our bigram baseline model.

For the logistic regression classifier, we used two variations of features during training: (i) use only the probabilities of the words, (ii) use these probabilities together with the mean and variance of the probabilities of a document. We first show the results using only probabilities, then with the combined features of probabilities, mean, and variance.

For presentation purposes, we give for each model, on each dataset, a heatmap showing the amount of correct and incorrect classifications. On these heatmaps, darker colors indicate higher values, and lighter colors indicate lower values. We also show a report of the standard evaluation metrics: accuracy, precision, recall, and F1. Finally for each model we also show a table of the five most misclassified words. We also discuss and analyse each model on each dataset.

To start with, we give a brief introduction and description of the standard metrics used. We also explain their importance in the context of our goal of detecting bad words.

Precision (1) measures the percentage of good words that the classifier detected correctly that are in also good in the gold data (*Jurafsky and Martin*, 2021). Recall (2) measures the percentage of good words actually present in the input that were correctly identified by the classifier (*Jurafsky and Martin*, 2021). The F1-score (3) is a harmonic mean that combines precision and recall, and presents the overall performance of a classifier (*Jurafsky and Martin*, 2021). Accuracy (4) shows the percentage of correct classification out of all classifications (*Jurafsky and Martin*, 2021).

Note that using accuracy is useful here because the classes are balanced, but in a more realistic use case there will be significantly less bad words than good words. As an example: if we have a document with 99 good words and 1 bad word, and we classify all of them as good, we get an accuracy of 99%, which is accurate, but not informative.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}} \tag{1}$$

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{2}$$

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{3}$$

$$\text{Accuracy} = \frac{\text{True Positives} + \text{True Negatives}}{\text{True Positives} + \text{True Negatives} + \text{False Positives} + \text{False Negatives}} \tag{4}$$

For our purposes, we are interested in *identifying* the bad words. Because of this, recall for bad words is a good metric to look at as it tells us how many bad words the classifier found. We also want a high precision, meaning that words identified as bad are actually bad. A high precision with low recall is less desirable because that means many bad words go unnoticed. It is also better if the classifier overshoots and classifies good words as bad rather than it classifying bad words as good words, as this will not only miss the bad words but also mislabel them.

5.1 Probabilities as features

Our first experiments focus on using only word probabilities as features for the logistic regression classifier. In what follows we give a detailed analysis of the results for each of our tested pre-trained language models, in addition to our baseline model.

5.1.1 NB-BERT

Results on the evaluation set – Using the evaluation set, we see from table 5.9 a decrease in accuracy of 2% from the test set. The F1 scores are lower than on the test set as well, with 84% and 83% for bad words and good words respectively. Recall for bad words stays the same at 86% with a 3% decrease in precision.

There is an increase in false negatives (22123) from the test set, with a smaller increase in false positives (16194) as figure 5.1(a) shows.

Table 5.1 shows that exactly the same types of words gets misclassified as in the test set as Table 5.2 show, only with differences in amounts. As for the top five false negatives, most of the words are political, like “EØS”, “president”, and “regjeringa”. We can also see the words “president” and “regjeringa” being misclassified for the test set as well. NB-BERT assigns lower probabilities to these words, indicating that they might be less known to it.

Using the evaluation set we see poorer results, but bad words are still retrieved at an equal rate as on the test set. Good words are more often correctly classified on both test and evaluation sets.

Results on the test set – Table 5.10 shows an accuracy of 85% on the test set, with a 15% misclassification rate corresponding to 31233 misclassifications. Table 5.10 shows that the F1 score is 86% for classification of bad words, and 85% for good words, with nearly identical scores for precision and recall. Recall for bad words is the highest with 86%, and recall for good words at 85%. Precision of bad words is 85% and for good words it reaches 86%.

Figure 5.1(b) shows that there were more false negatives (15694) than false positives (15539) using the test set. NB-BERT misclassifies good words more than it misclassifies bad words. This is reflected in the higher F1 score for bad words as well.

The top most misclassified bad words and good words are shown in Table 5.2. Out of the top five false positives, “dum” and “svik” is the least derogatory. All the words are also short in length. It is surprising that the words “faen”, “hor”, and “hore” are present as they are considered to be very derogatory, but NB-BERT still gives them high enough probabilities to be classified as good. Note that the reason the amounts of each false positive is higher than in the false negative cases is because there are fewer types of bad words to misclassify, as they are restricted to the size of Hurtlex (176) as can be seen in table 3.5.

Using the test set, the classifier can find and classify bad words slightly better than good words as is reflected in the higher recall and F1 score for the bad word class.

Summary NB-BERT – Overall, when using only probabilities from NB-BERT as features, the classifier is slightly better at finding bad words, but with a lower precision. It is slightly worse at finding good words, but will classify them more correctly. The fact that very derogatory words such as “faen” gets misclassified as good is surprising, and is the opposite of what would wish for.

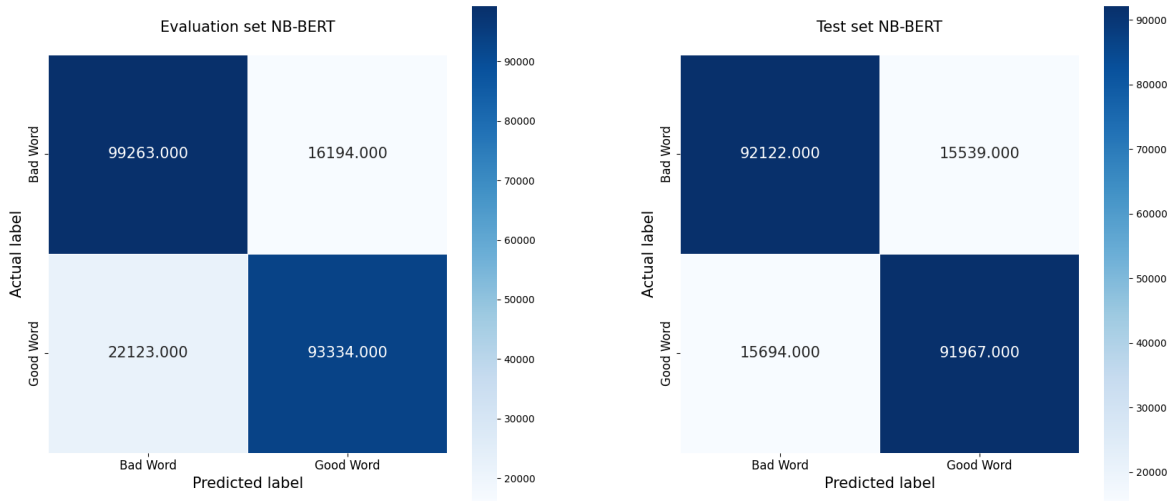


Figure 5.1: Heatmap showing classifications of bad and good words using NB-BERT probabilities on the evaluation (a) and test (b) set. Feature: probability

5.1.2 NorBERT1

Results on the evaluation set – Using the evaluation set, we get the same accuracy of 95%, and also the same scores for precision, recall, and F1 as on the test set, as Table 5.9 and 5.10 shows. However, here we see what we saw on both datasets for NB-BERT and unlike for the test set (see results below) using NorBERT1: there are more false negatives (5819) than false positives (5766) as figure 5.2(a) shows, which is preferable as we don’t want bad words being classified as good.

Bad Words Labeled as Good (FP)	Good Words Labeled as Bad (FN)
hor (5648)	president (585)
dum (4361)	òg (128)
faen (1198)	EØS (114)
svik (582)	regjeringen (92)
hore (465)	komiteen (75)

Table 5.1: Five most frequently misclassified words using NB-BERT probabilities on the evaluation set. Amounts in parentheses. FP = False positives. FN = False negatives. Feature: probability

Bad Words Labeled as Good (FP)	Good Words Labeled as Bad (FN)
hor (6074)	president (530)
dum (2050)	òg (65)
faen (1471)	regjeringa (57)
svik (701)	trenger (56)
hore (445)	sånn (44)

Table 5.2: Five most frequently misclassified words using NB-BERT probabilities on the test set. Amounts in parentheses. FP = False positives. FN = False negatives. Feature: probability

Table 5.3 shows that we get the same misclassified words as when we used the test set, with relatively similar amounts as well. The only difference is in the false negative case where the fifth word "statsråden" in the test set is swapped for the short word "au" in the evaluation set. Again we see that the words are either short or political.

Results on the test set – Table 5.10 shows an accuracy of 95% on the test set. This corresponds to a 5% misclassification rate (10925 in total). Table 5.10 shows that the scores for precision, recall, and F1 are all 95%. The classifier performs equally well in regards to finding good and bad words. Unlike the results for both datasets using NB-BERTs probabilities, there are less false negatives (5284) than false positives (5641) as figure 5.2(b) shows. This means that it more often will classify a bad word as good rather than the other way. This is less ideal as this leads to bad words being classified as good.

Table 5.4 shows that "faen" is the bad word that most often gets misclassified as good, which is surprising as it gets misclassified more than three times as much as the next bad word which is "vås". The word "Faen" is also the most derogatory, being the only swear word. The misclassified good words are, as with NB-BERT, either short such as "s", which is not a word, but is most likely a result of tokenization. Or, they are political, such as "president" and "statsråden". This of course reflects the dataset we use, and it could be that NorBERT1 has problems with these domain specific words as we suspect NB-BERT also do.

Summary NorBERT1 – NorBERT1 performs better than NB-BERT on both datasets, with higher scores on all metrics. It has less misclassifications, with a high precision and recall for good and bad words. It does misclassify good words as bad slightly more than the other way around, which is less ideal. Surprisingly enough it manages

to misclassify the word “faen” as good, more than three times as much as the others, and with higher amounts than NB-BERT on both datasets. However, this is the only swear word in the top five, the others being less derogatory, whereas NB-BERT had more derogatory words. Moving on, we expect that NorBERT2 will perform as well or better than NorBERT1.

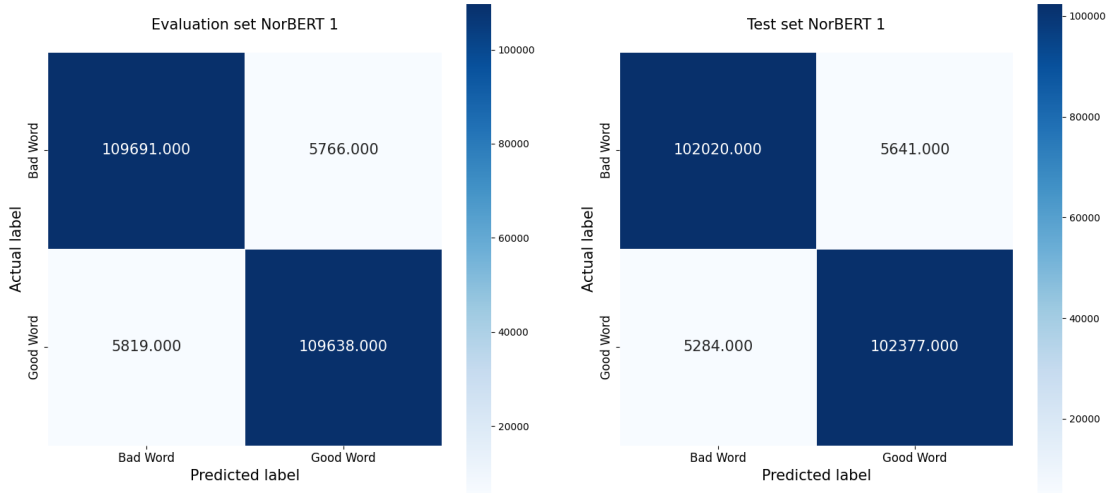


Figure 5.2: Heatmap showing classifications of bad and good words using NorBERT1 probabilities on the evaluation (a) and test (b) set. Feature: probability

Bad Words Labeled as Good (FP)	Good Words Labeled as Bad (FN)
faen (1556)	president (741)
vås (539)	òg (104)
kjeltring (427)	takk (48)
avskyelig (273)	s (45)
stinker (263)	au (41)

Table 5.3: Five most frequently misclassified words using NorBERT1 probabilities on the evaluation set. Amounts in parentheses. FP = False positives. FN = False negatives. Feature: probability

Bad Words Labeled as Good (FP)	Good Words Labeled as Bad (FN)
faen (1613)	president (697)
vås (471)	takk (69)
kjeltring (358)	òg (66)
avskyelig (226)	s (39)
stinker (220)	statsråden (38)

Table 5.4: Five most frequently misclassified words using NorBERT1 probabilities on the test set. Amounts in parentheses. FP = False positives. FN = False negatives. Feature: probability

5.1.3 NorBERT2

Results on the evaluation set – Using the evaluation set, Table 5.9 shows the same accuracy as the test set, 94%, while recall of bad words decreases by 1%, and precision and F1 score for good word classification decrease by 1% compared to the test set.

Figure 5.3(a) also shows the same pattern regarding misclassifications as the test set (see below). There are more false negatives (9179) than false positives (5677).

Table 5.5 shows that we get the same misclassified words as when we used the test set, with relatively similar amounts as well. This is similar to what we saw with NorBERT1.

Results on the test set – Table 5.10 shows an accuracy of 94% on the test set, 1% less than NorBERT1. The F1 score of 94% for classification of both good and bad words is 1% lower than NorBERT1. The recall is 1% higher than NorBERT1 for bad words, meaning that NorBERT2 finds more bad words, but with a 3% lower precision at 92% misclassifies them more often.

Figure 5.3(b) shows more false negatives (8450) than false positives (4561). It will classify good words as bad more often than the other way around.

Table 5.6 shows the same pattern we have seen with NB-BERT and to a lesser degree NorBERT1, that shorter bad words are misclassified as good rather than longer ones. In contrast to NB-BERT and NorBERT1, the word "faen" is missing here, which is an improvement over NorBERT1.

For the false negatives, there is also a similar pattern as with NB-BERT and NorBERT1, with words either being short such as "i" and "det", or political such as "president". The token `[UNK]` is used for words that the tokenizer does not recognize. We know that NorBERT2 has problems with accented letters, and given that the word "òg" has appeared in the top misclassifications using NB-BERT and NorBERT1, we expect the unknown word to be that.

Summary NorBERT2 – We expected NorBERT2 to perform better than NorBERT1, given that it is the most recent model trained with more data and has a larger vocabulary than NorBERT1 (20000 more tokens). NorBERT2 only has better recall for bad words (96% on the test set), and better precision for good words (96% on the test set). It does not have the same problem with the swear word "faen" as NorBERT1, but at the same time the rest of the misclassifications that NorBERT1 make are overall less derogatory than NorBERT2s misclassifications.

With the scores on the metrics, along with the amount and types of misclassifications, NorBERT2 does perform slightly worse than NorBERT1. We are not sure why this is the case, but think that it has something to do with the models tokenization of the documents, and perhaps unknown tokens.

5.1.4 Bigram

Results on the evaluation set Table 5.9 shows that on the evaluation set, the bigram model has an accuracy of 90%, a decrease of 1% from the test set (see discussion bellow). Recall and F1 scores of bad words decrease by 1%, and recall of good words

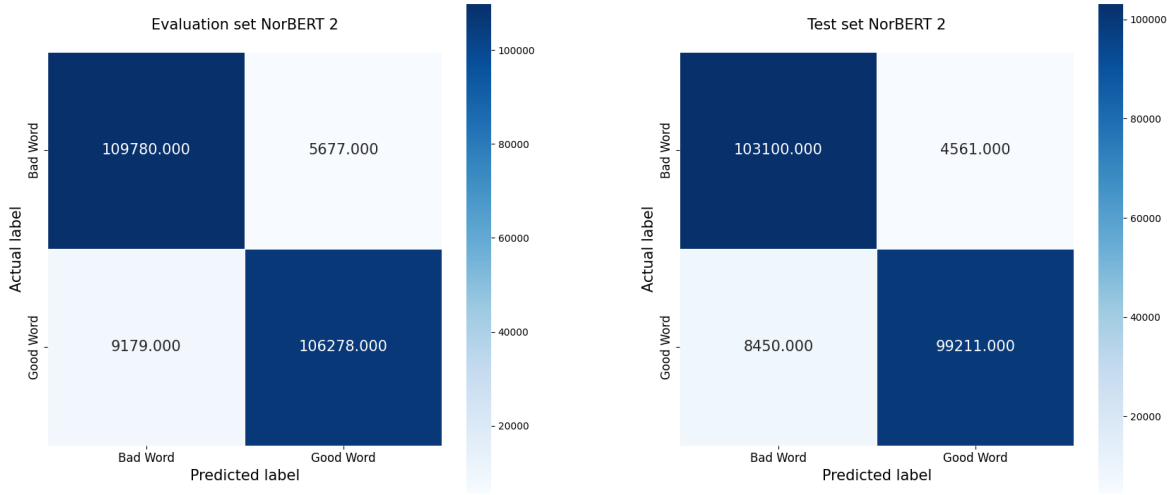


Figure 5.3: Heatmap showing classifications of bad and good words using NorBERT2 probabilities on the evaluation (a) and test (b) set. Feature: probability

Bad Words Labeled as Good (FP)	Good Words Labeled as Bad (FN)
hor (2213)	president (444)
dum (2044)	[UNK] (135)
dritt (417)	det (71)
svik (250)	i (70)
late (192)	og (45)

Table 5.5: Five most frequently misclassified words using NorBERT2 probabilities on the evaluation set. Amounts in parentheses. FP = False positives. FN = False negatives. Feature: probability

Bad Words Labeled as Good (FP)	Good Words Labeled as Bad (FN)
hor (1936)	president (452)
dum (1357)	[UNK] (103)
dritt (341)	det (69)
svik (252)	i (69)
late (155)	og (55)

Table 5.6: Five most frequently misclassified words using NorBERT2 probabilities on the test set. Amounts in parentheses. FP = False positives. FN = False negatives. Feature: probability

decreases by 1%. This slight decrease in scores is reflected in higher amounts of misclassifications as seen in Figure 5.4(a) compared to 5.4(b), where false negatives increase from 12925 to 14411, and false positives increase from 6837 to 8023.

Table 5.7 shows that the types of words remain unchanged for the false positives, and some changes for the false negatives compared to Table 5.8.

Results on the test set – The bigram model on the test set has an accuracy of 91%, for bad words recall is 94%, precision is 88%, and F1 score is 91%. For good words, recall is 88%, precision is 93%, and F1 is 90%, as Table 5.10 shows. It is better at finding bad words, while more precise when classifying good words. Figure 5.4(b) shows more false negatives (12925) than false positives (6837).

The misclassified words as seen in Table 5.8 is as we have seen with other models, mostly shorter bad words for the false positives, and shorter or political words for the false negatives.

Summary bigram – The bigram model, using probabilities as the only feature is, surprisingly, better than NB-BERT. A bigram model would most likely assign very low probabilities to bad words, which makes them easier to separate from the good words. We see this behaviour illustrated in figure A.4 showing probability distributions using the bigram model. This can explain the high recall it achieves for bad words (93% on evaluation set, 94% on test set). We can see that reflected in the high recall scores for bad words on both the test and evaluation set.

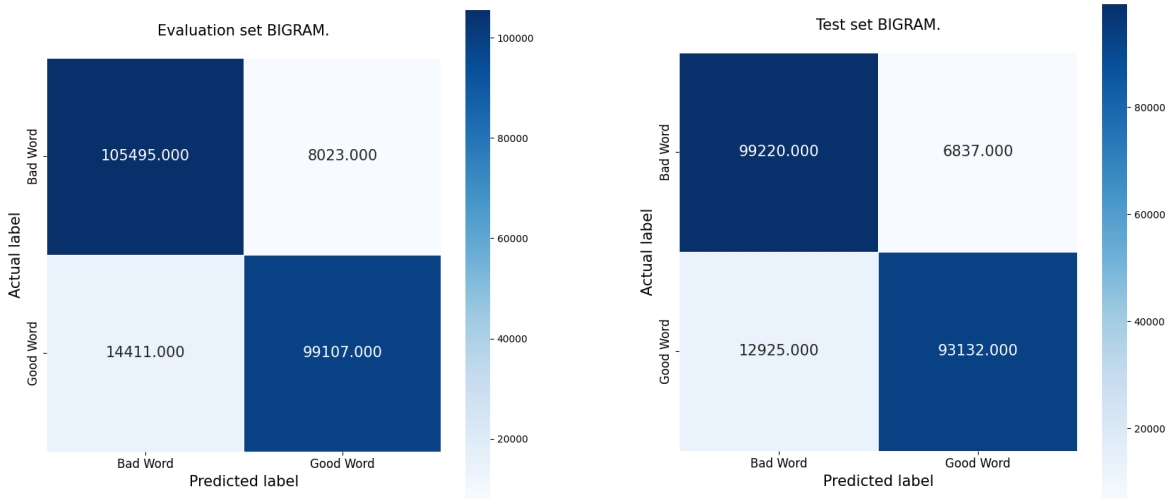


Figure 5.4: Heatmap showing classifications of bad and good words using the bigram probabilities on the evaluation (a) and test (b) set. Feature: probability

Summary of results using probabilities as the only feature – Using only the probabilities as features, the model that performs best overall is NorBERT1, with NB-BERT performing worse than the baseline. NorBERT2 have the highest recall for bad words on the test set, meaning that it found the most bad words. However, it has less precision than NorBERT1.

Bad Words Labeled as Good (FP)	Good Words Labeled as Bad (FN)
dum (4550)	president (502)
faen (1451)	men (203)
hore (593)	det (153)
gris (403)	og (129)
tulling (134)	så (99)

Table 5.7: Five most frequently misclassified words using bigram probabilities on the evaluation set. Amounts in parentheses. FP = False positives. FN = False negatives. Feature: probability

Bad Words Labeled as Good (FP)	Good Words Labeled as Bad (FN)
dum (3748)	president (495)
faen (1330)	men (246)
hore (487)	representanten (126)
gris (329)	takk (114)
tulling (123)	det (112)

Table 5.8: Five most frequently misclassified words using bigram probabilities on the test set. Amounts in parentheses. FP = False positives. FN = False negatives. Feature: probability

Class	Model	Precision	Recall	F1	Accuracy	Support
Bad Word	NB-BERT	0.82	0.86	0.84	0.83	115457
Good Word	NB-BERT	0.85	0.81	0.83		115457
Bad Word	NorBERT1	**0.95**	**0.95**	**0.95**	**0.95**	115457
Good Word	NorBERT1	**0.95**	**0.95**	**0.95**		115457
Bad Word	NorBERT2	0.92	**0.95**	0.94	0.94	115457
Good Word	NorBERT2	**0.95**	0.92	0.93		115457
Bad Word	Bigram	0.88	0.93	0.90	0.90	113518
Good Word	Bigram	0.93	0.87	0.90		113518

Table 5.9: Precision, recall, F1, and accuracy for the models on the evaluation set. Feature used: probability.

Class	Model	Precision	Recall	F1	Accuracy	Support
Bad Word	NB-BERT	0.85	0.86	0.86	0.85	107661
Good Word	NB-BERT	0.86	0.85	0.85		107661
Bad Word	NorBERT1	**0.95**	0.95	**0.95**	**0.95**	107661
Good Word	NorBERT1	0.95	**0.95**	**0.95**		107661
Bad Word	NorBERT2	0.92	**0.96**	0.94	0.94	107661
Good Word	NorBERT2	**0.96**	0.92	0.94		107661
Bad Word	Bigram	0.88	0.94	0.91	0.91	106057
Good Word	Bigram	0.93	0.88	0.90		106057

Table 5.10: Precision, recall, F1, and accuracy for the models on the test set. Feature used: probability.

5.2 Probabilities, mean, and variance as features

In addition to using only the word probabilities as features, we experimented with using the mean probability and the variance of the probabilities. We expected the models to be more accurate with these additional features. As earlier, we present the results from each model on each dataset. When presenting each models result on a dataset, we will compare these with previous results found using only probabilities.

5.2.1 NB-BERT

Results on the evaluation set – Using the evaluation set, Table 5.19 shows a decrease of 1% in accuracy from earlier results. For bad words, recall decreases 5% (from 86% to 81%), while precision increases 2% (from 82% to 84%). For good words, recall increases 3% (from 81% to 84%), while precision decreases with 4% (from 85% to 81%). F1 scores stay the same for good words while it decreases by 2% (from 84% to 82%) for bad words. This tells us that the classifier finds more good words, with less precision, and finds fewer bad words, but with higher precision. Figure 5.5(a) shows that false negatives decreased from 22123 to 18251, while false positives increased from 16194 to 22317 compared to figure 5.1(a). As stated earlier, we want to avoid this, as it means that more bad words are labeled as good, than vice versa.

As seen for previous results in Table 5.1, Table 5.11 shows that false positives are short in length, and false negatives are mostly political words. The word “dum” gets misclassified as bad the most often, which makes sense as it is less bad than “faen” or “hor” and has most likely been assigned higher probabilities in the documents.

Results on the test set – For NB-BERT using the test set with probabilities, mean, and variance as features, the accuracy increases slightly with 2%, from 85% to 87%, compared to earlier results, as Table 5.20 and Table 5.10 show. Both precision and recall go up 1% for both good word and bad word classification, while F1 scores go up 1% and 2% for bad words and good words respectively. These results show an improvement, and we can see from Figure 5.5(b) compared with Figure 5.1(b) that false negatives have decreased from 15694 to 14596 and false positives decreased from 15539 to 14443.

The types of words misclassified are almost identical to earlier in Table 5.2, where in the false positives column the word “late” has replaced “hore” in Table 5.12. The same pattern emerges as earlier, with shorter bad words being misclassified most frequently. Except for “faen” and “hor”, these bad words are less derogatory than many of those found in Hurtlex, and aren’t as severe of a misclassification, perhaps indicating that we should not have included them.

In the false negative case, the word “nettopp” replaces “trenger”. As before, we see words like “nettopp” and “òg”, as well as political terms such as “president” and “regjeringa” being most frequently misclassified.

Summary NB-BERT – Overall, NB-BERT showed improvement on the test set, but not on the evaluation set when using the combination of probabilities, mean, and variance as features. We expected the results to improve when using more features, but this

was only the case on the test set, which are arguably the most important results to focus on.

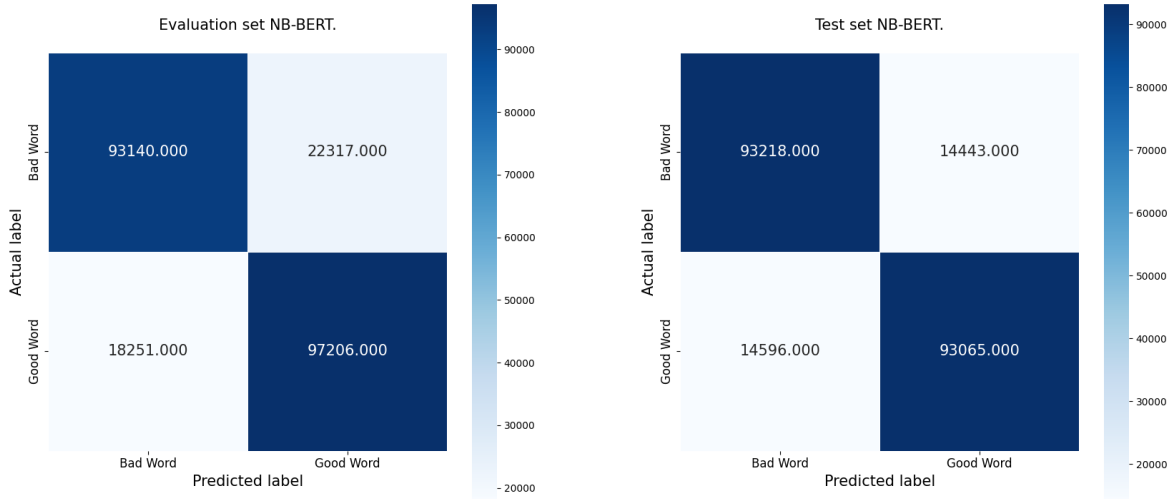


Figure 5.5: Heatmap showing classifications of bad and good words using NB-BERT probabilities on the evaluation (a) and test (b) set. Features: probability, mean, variance

Bad Words Labeled as Good (FP)	Good Words Labeled as Bad (FN)
dum (12195)	president (559)
hor (5809)	òg (152)
late (1002)	EØS (106)
faen (751)	komiteen (69)
hore (450)	regjeringa (64)

Table 5.11: Five most frequently misclassified words using NB-BERT probabilities on the evaluation set. Amounts in parentheses. FP = False positives. FN = False negatives. Features: probability, mean, variance

Bad Words Labeled as Good (FP)	Good Words Labeled as Bad (FN)
dum (5861)	president (626)
hor (4523)	òg (96)
faen (899)	regjeringa (53)
late (476)	trenger (38)
svik (305)	nettopp (37)

Table 5.12: Five most frequently misclassified words using NB-BERT probabilities on the test set. Amounts in parentheses. FP = False positives. FN = False negatives. Features: probability, mean, variance

5.2.2 NorBERT1

Results on the evaluation set – The results using the evaluation set are also largely unchanged from earlier. It got an accuracy of 95%, equal to earlier results. Precision, recall, and F1 scores for both good and bad words are also unchanged, as seen in Table 5.19 compared to Table 5.9.

Comparing Figure 5.6(a) with Figure 5.2(a), we can see that the types of classifications also remain largely unchanged with roughly the same ratios.

The types of bad words that get misclassified remains the same as when we used only probabilities as features as Tables 5.13 and 5.3 show. The false negative cases are, as when using only probabilities, words that are either short or political.

Results on the test set – Using NorBERT1 with the probabilities, mean, and variance, the results using the test set are also largely unchanged from the earlier results. Accuracy is the same, precision, recall, and F1 scores are also equal as seen when comparing Table 5.20 with Table 5.10. The ratio of false negatives to false positives is also largely unchanged as shown in Figure 5.6(b) compared to Figure 5.2(b).

The types of words that get misclassified have not changed for false negatives and false positives as Tables 5.14 and 5.4 show.

Summary NorBERT1 – There is no large changes between these results and earlier ones, which might not be too surprising given that NorBERT1 already had high scores on all metrics. In contrast, NB-BERT showed improvements using the test set, but the results using only probabilities was not that good to begin with. It could be that further increasing the amount of features, or using different features, will improve results. This we leave as future work.

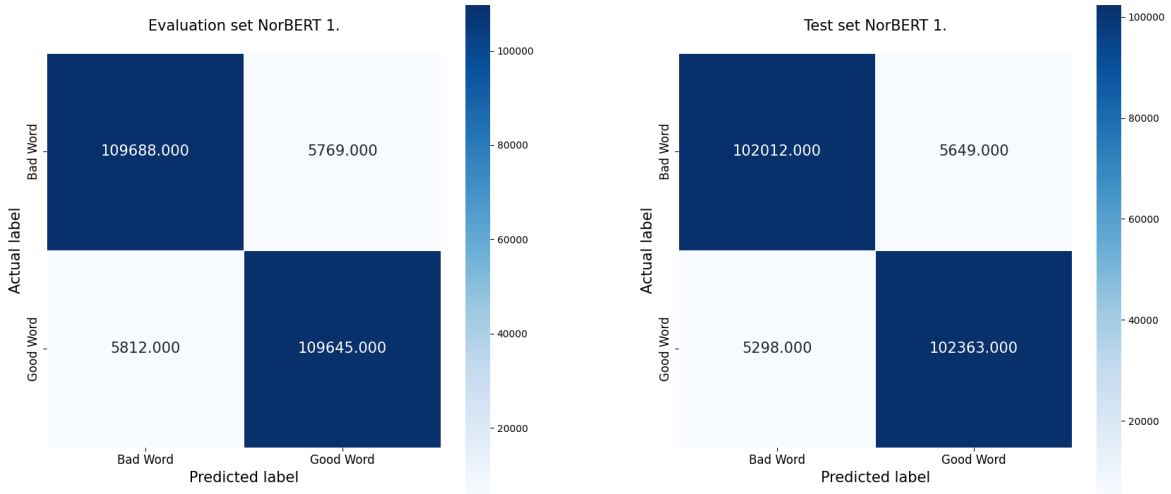


Figure 5.6: Heatmap showing classifications of bad and good words using NorBERT1 probabilities on the evaluation (a) and test (b) set. Features: probability, mean, variance

Bad Words Labeled as Good (FP)	Good Words Labeled as Bad (FN)
faen (1556)	president (741)
vås (544)	òg (106)
kjeltring (426)	takk (48)
stinker (274)	s (46)
avskyelig (262)	au (41)

Table 5.13: Five most frequently misclassified words using NorBERT1 probabilities on the evaluation set. Amounts in parentheses. FP = False positives. FN = False negatives. Features: probability, mean, variance

Bad Words Labeled as Good (FP)	Good Words Labeled as Bad (FN)
faen (1619)	president (699)
vås (467)	takk (69)
kjeltring (358)	òg (67)
avskyelig (228)	s (39)
stinker (220)	statsråden (37)

Table 5.14: Five most frequently misclassified words using NorBERT1 probabilities on the test set. Amounts in parentheses. FP = False positives. FN = False negatives. Features: probability, mean, variance

5.2.3 NorBERT2

Results on the evaluation set – Using the evaluation set also shows no significant changes. Comparing Table 5.19 and Table 5.9 shows that accuracy, precision, recall, and F1 scores are unchanged for both classes.

Comparing Figure 5.7(a) and Figure 5.3(a) show small differences in misclassifications, with 17 more false negatives, and 31 fewer false positives.

The types of misclassifications remains the same as when we only used probabilities as features, as can be seen when comparing Table 5.15 with Table 5.5.

Results on the test set – Using NorBERT2 on the test set with the combination of probabilities, mean, and variance shows no significant changes from earlier results. Comparing Table 5.20 and Table 5.10 we see that accuracy, precision, recall, and F1 scores are unchanged for both classes.

The amount of misclassifications, both false negatives and false positives are also mostly unchanged, with 19 more false negatives and 4 fewer false positives when comparing Figure 5.7(b) to Figure 5.3(b).

The top five misclassifications are also the same, with marginal differences in amounts for any word when comparing Table 5.16 and Table 5.6.

Summary NorBERT2 – As with NB-BERT and NorBERT1, we expected that adding the mean and variance as features would improve results, but surprisingly they have not.

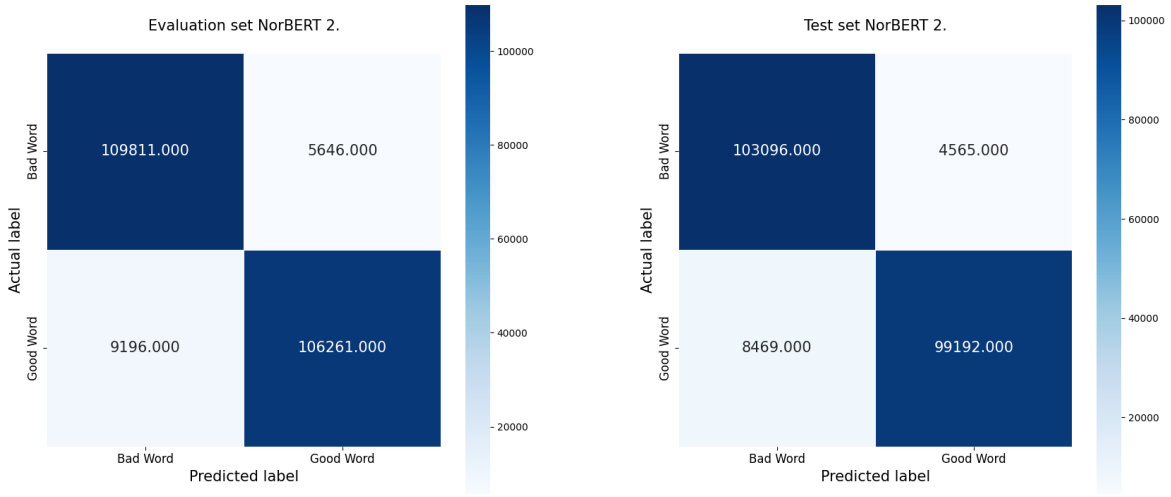


Figure 5.7: Heatmap showing classifications of bad and good words using NorBERT2 probabilities on the evaluation (a) and test (b) set. Features: probability, mean, variance

Bad Words Labeled as Good (FP)	**Good Words Labeled as Bad (FN)**
hor (2206)	president (449)
dum (1995)	[UNK] (137)
dritt (416)	det (76)
svik (242)	i (68)
late (191)	og (44)

Table 5.15: Five most frequently misclassified words using NorBERT2 probabilities on the evaluation set. Amounts in parentheses. FP = False positives. FN = False negatives. Features: probability, mean, variance

Bad Words Labeled as Good (FP)	**Good Words Labeled as Bad (FN)**
hor (1937)	president (453)
dum (1359)	[UNK] (105)
dritt (347)	det (73)
svik (248)	i (64)
late (151)	og (56)

Table 5.16: Five most frequently misclassified words using NorBERT2 probabilities on the test set. Amounts in parentheses. FP = False positives. FN = False negatives. Features: probability, mean, variance

5.2.4 Bigram

Results on the evaluation set – On the evaluation set, we see that accuracy, precision, recall, and F1 remain the same as before as Table 5.9 compared with Table 5.19 show. There are only 107 fewer false negatives, as Figure 5.4(a) and Figure 5.8(a) show.

As with the test set (see discussion below), the types of misclassifications, both false negatives and false positives are the same, with minor changes as can be seen in Table 5.7 and Table 5.17.

Results on the test set – Using the bigram on the test set with the combined probabilities, mean, and variance as features have show no difference from previous results.

Accuracy, precision, recall, and F1 are unchanged for both classes as Table 5.10 and Table 5.20 show.

Figure 5.4(b) and Figure 5.8(b) show that there are no differences in the distribution of types of classifications.

Table 5.18 and Table 5.8 show the same types being wrongly classified, with only small differences in amounts.

Summary Bigram – Using the combined probabilities, mean, and variance as features have close to no effect compared to only using probabilities when using the bigram model for both the evaluation and the test sets.

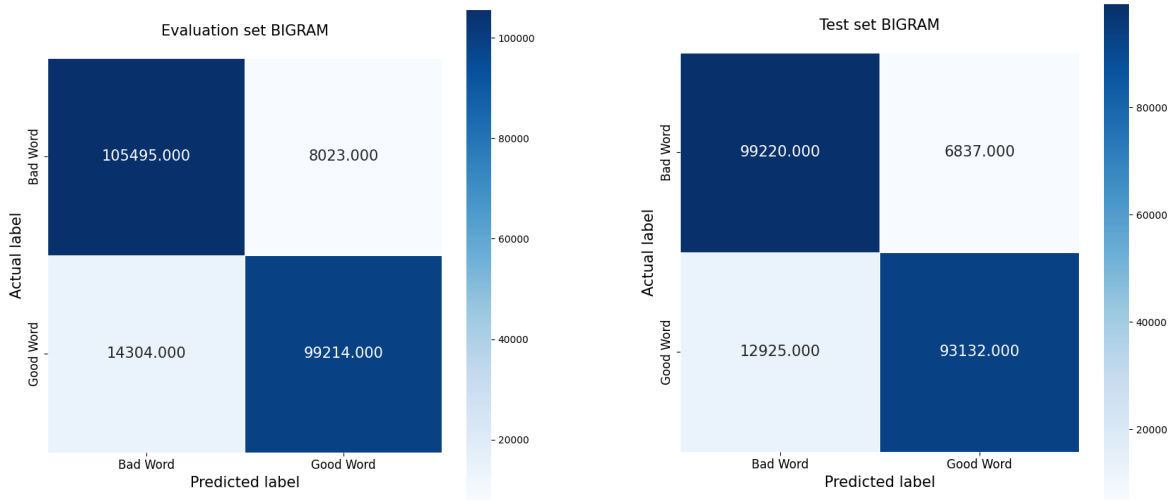


Figure 5.8: Heatmap showing classifications of bad and good words using bigram probabilities on the evaluation (a) and test (b) set. Features: probability, mean, variance

Summary of results – Overall, for all the models on the test and evaluation sets there were no significant improvements using the mean and variance as additional features. This was surprising as we expected that more features would improve the classifier. It could be that adding even more features would improve results, but this would probably only help models that already have relatively bad performance, such as NB-BERT, where we saw that an increase in number of features did improve its performance at

Bad Words Labeled as Good (FP)	Good Words Labeled as Bad (FN)
dum (4550)	president (504)
faen (1451)	men (217)
hore (593)	det (152)
gris (403)	og (128)
tulling (134)	så (100)

Table 5.17: Five most frequently misclassified words using bigram probabilities on the evaluation set. Amounts in parentheses. FP = False positives. FN = False negatives. Features: probability, mean, variance

Bad Words Labeled as Good (FP)	Good Words Labeled as Bad (FN)
dum (3748)	president (492)
faen (1330)	men (252)
hore (487)	representanten (129)
gris (329)	det (112)
tulling (123)	takk (108)

Table 5.18: Five most frequently misclassified words using bigram probabilities on the test set. Amounts in parentheses. FP = False positives. FN = False negatives. Features: probability, mean, variance

least on the test set. We can see that NorBERT1 and NorBERT2 do not gain much improvement, and in some cases have slightly more misclassifications.

With the highest scores on all metrics, NorBERT1 performs the best on both datasets with no little to no differences between the two types of feature sets. NorBERT2 performs well, but has lower scores on precision, making it less capable than NorBERT1. NB-BERT performs worse than a bigram model, which could be caused by it using a multilingual tokenizer with less coverage of the Norwegian language.

Class	Model	Precision	Recall	F1	Accuracy	Support
Bad Word	NB-BERT	0.84	0.81	0.82	0.82	115457
Good Word	NB-BERT	0.81	0.84	0.83		115457
Bad Word	NorBERT1	**0.95**	**0.95**	**0.95**	**0.95**	115457
Good Word	NorBERT1	**0.95**	**0.95**	**0.95**		115457
Bad Word	NorBERT2	0.92	**0.95**	0.94	0.94	115457
Good Word	NorBERT2	**0.95**	0.92	0.93		115457
Bad Word	Bigram	0.88	0.93	0.90	0.90	113518
Good Word	Bigram	0.93	0.87	0.90		113518

Table 5.19: Precision, recall, F1, and accuracy for the models on the evaluation set. Features used: probability. mean, variance.

Class	Model	Precision	Recall	F1	Accuracy	Support
Bad Word	NB-BERT	0.86	0.87	0.87	0.87	107661
Good Word	NB-BERT	0.87	0.86	0.87		107661
Bad Word	NorBERT1	**0.95**	0.95	**0.95**	**0.95**	107661
Good Word	NorBERT1	0.95	**0.95**	**0.95**		107661
Bad Word	NorBERT2	0.92	**0.96**	0.94	0.94	107661
Good Word	NorBERT2	**0.96**	0.92	0.94		107661
Bad Word	Bigram	0.88	0.94	0.91	0.91	106057
Good Word	Bigram	0.93	0.88	0.90		106057

Table 5.20: Precision, recall, F1, and accuracy for the models on the test set. Features used: probability, mean, variance.

5.3 Limitations

In this thesis, we treat the bad words from Hurtlex as being misplaced, or wrong, but a bad word isn't necessarily a wrong or misplaced word. In Hurtlex, there are a lot of swearwords, for example. These are words just like any other words, so we can't say categorically that the words should never be in a document, because a person could intend to use such a word when speaking.

When inserting bad words, we do so using minimum edit distance. If the NPSC has mostly shorter words, then there will mostly be shorter bad words replacing them from Hurtlex, creating an imbalance between short and long bad words, where longer words are less frequently introduced. We saw this pattern emerge when we presented the top five misclassifications for our models.

When finding probabilities for words using NorBERT2, we noticed that many words were, for some unknown reason, not recognized by the model. This results in the word being converted to the special token `[UNK]`. We saw that this was the case with accented letters such as "é" and "ü", making words such as "òg" not recognized. This might have affected the performance of the model as it simply doesn't see some of the words. This could have been a mistake from our side. We might have initialized the model wrongly from HuggingFace, but most likely it is just a limitation of the model and its vocabulary.

As our baseline, we used a bigram, which is very simple. We could have experimented with using a more advanced model as our baseline.

Chapter 6

Conclusion and Future Work

In this thesis, we have experimented with pre-trained language models trained on Norwegian text data, using them to test their ability to detect problematic words in speech-to-text transcripts, and to create a system to detect bad words using a logistic regression classifier together with the probabilities computed using the language models.

We used the NPSC corpus together with the Norwegian Hurtlex lexicon as underlying datasets. We augmented the NPSC by inserting bad words from Hurtlex into documents. Then, we computed probabilities of both words and bad words in each selected document. We used a logistic regression classifier together with the probabilities. We classified the words using two different sets of features: only the probabilities, and the probabilities along with the mean and variance of their values. We showed that increasing features gave better results for NB-BERT which was the model that did the worst to begin with, but gave near equal results for NorBERT1 and NorBERT2 with marginal differences.

We found that NorBERT1 and NorBERT2 are both capable of detecting bad words. These models tend to give such problematic words a lower probability than for the other (reference good) words in a sentences or a document. NorBERT1 performs slightly better, which is surprising as NorBERT2 is trained on more data and has a larger vocabulary. NB-BERT is less reliable, with lower scores than a bigram model which we believe can be attributed to the multilingual tokenizer of NB-BERT and the fact that the bigram gives very low scores to bad words.

Our work has some limitations. First, we could have experimented with using more features and various iterations for the classification with logistic regression. Second, we could have used other sources to find bad words to expand the overall total number of words. The NPSC that we used covers mostly political debates and speeches, which might not be as representative and contains too much domain-specific utterances. We saw that some of the models labeled many of the domain-specific political terms as bad words. Finally, when replacing a good word with a bad word we simply checked minimum edit distance between a good word and all bad words in Hurtlex, which is a somewhat crude way to insert bad words. We could therefore have experimented with other ways of selecting candidates from the Hurtlex lexicon.

The purpose of the thesis was to research the capabilities of three Norwegian pre-trained language models, and explore to what extent they are able to detect problematic words in texts created by automated speech recognition systems. As mentioned in Chapter 1, in this work we aimed at answering two interconnected research questions.

To remind the readers, these were:

- **Research Question 1** (**RQ 1**): To what extent can probabilities of words computed using pre-trained language models be leveraged to automatically detect severe errors in speech-to-text transcripts?

- **Research Question 2** (**RQ 2**): To what extent can we develop a system, using pre-trained language models, that can find severe errors in speech-to-text transcripts, using word probabilities computed using pre-trained models?

To answer research question 1, we showed in Section 4 and in Appendix A the results of computing probabilities for problematic and non-problematic words and we clearly show that NorBERT1, NorBERT2 are the most capable at separating problematic words from non-problematic words by giving problematic words lower probabilities and non-problematic words higher probabilities. NB-BERT proved to be less capable than a simple bigram baseline.

To answer research question 2, we showed in detail in Chapter 5 how using NorBERT1 and NorBERT2 together with a logistic regression classifier, we are able to show with high scores on relevant metrics that using probabilities of words it is possible to classify them as either problematic or non-problematic. We also show that using NB-BERT probabilities give worse results using the classifier, where we also highlighted that NB-BERT performs worse than a simple bigram model.

To summarise our main findings and contributions: first, we have shown that when computing probabilities of words using pre-trained Norwegian language models, the models generally assign lower values to problematic words synthetically inserted, and higher probabilities to non-problematic words. Second, we have both explored and reported that using a logistic regression classifier in combination with the probabilities computed using pre-trained Norwegian language models, have enabled us to show that it is possible to detect problematic words. Finally, we have discussed how there are differences in results based on which language model is used, such that NorBERT1 outperforms NorBERT2, while NB-BERT performs worse than a simple bigram model.

6.1 Future work

What we wish to attempt in the future is to use the logistic regression classifier that we trained on the probabilities, and use this on sentences from real world scenarios from TV 2. Bad words are often rare, and so it would be interesting to use the classifier to classify sentences where most of the words are good words, and see if it manages to find the few bad ones. For this, we especially want to explore the trade-off between false positives and false negatives, in order to avoid classifying good words as bad, and bad words as good.

We had intentions of not only detecting, but also correcting bad words, but due to time constraints we did not attempt this, and leave this endeavour for future work. A clear continuation of this work would be to correct the bad words after they have been detected. After being detected, a bad word can for example be replaced by asking a language model to mask the position of the bad word, and output a list of K-number of

possible candidates for correction. A document with a masked word can be fed into it, and a list of potential candidates with the highest probability will be given back. One could use a threshold that a candidate has to pass in order to serve as a good correction just like a word has to pass a threshold to be considered bad. Then one can find possible candidates that are above this threshold.

Appendix A

Appendix

The contents of this appendix shows the results we got when computing probabilities for good (non-problematic) and bad (problematic) words. All figures show distributions of probabilities of good and bad words using the models on the training set. We show the distribution both using regular probabilities and logarithmic probabilities.

The figures all show the models capabilities with regards to giving bad words low probabilities and good words high probabilities. These capabilities are easiest to see using logarithms.

A.1 NB-BERT

Computing probabilities using NB-BERT show that it manages to separate the good words from the bad words, but with a noticeable overlap, which means that many bad words are labeled as good and vice versa, as Figure A.1 shows.

A.2 NorBERT1

Computing probabilities using NorBERT1 shows improvements over NB-BERT. It manages to clearly separate the good words from the bad words, with little overlap as Figure A.2 shows.

A.3 NorBERT2

Computing probabilities using NorBERT2 shows similar capabilities as NorBERT1. It manages to clearly separate the good words from the bad words, with still more overlap than NorBERT1 as Figure A.3 shows. It assigns lower probabilities to many bad words, than NorBERT1.

A.4 Bigram

Computing probabilities using the bigram shows that the model assigns very low probabilities to most of the bad words. It also manages to separate the good words from the bad words to a lesser degree as Figure A.4 shows.

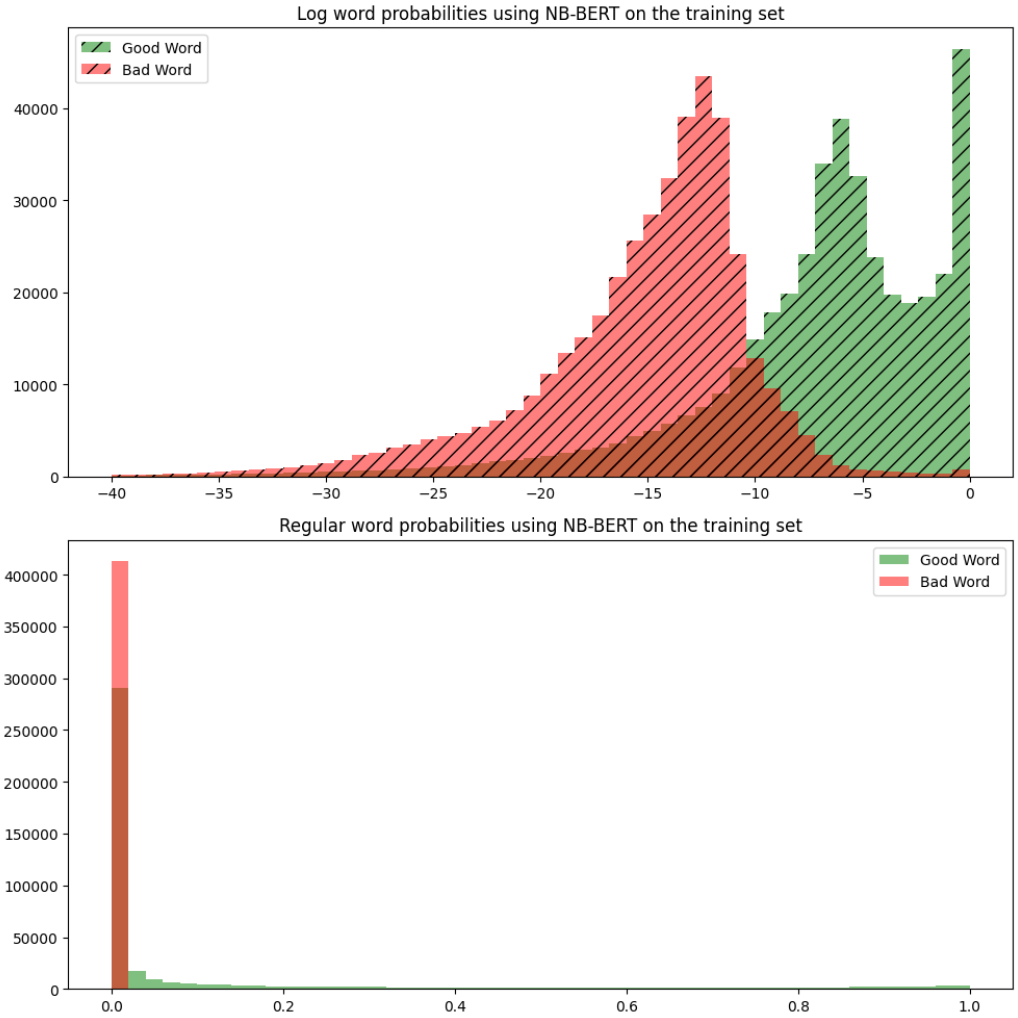


Figure A.1: Distribution of regular and logarithmic probabilities for good words and bad words using NB-BERT on the training set.

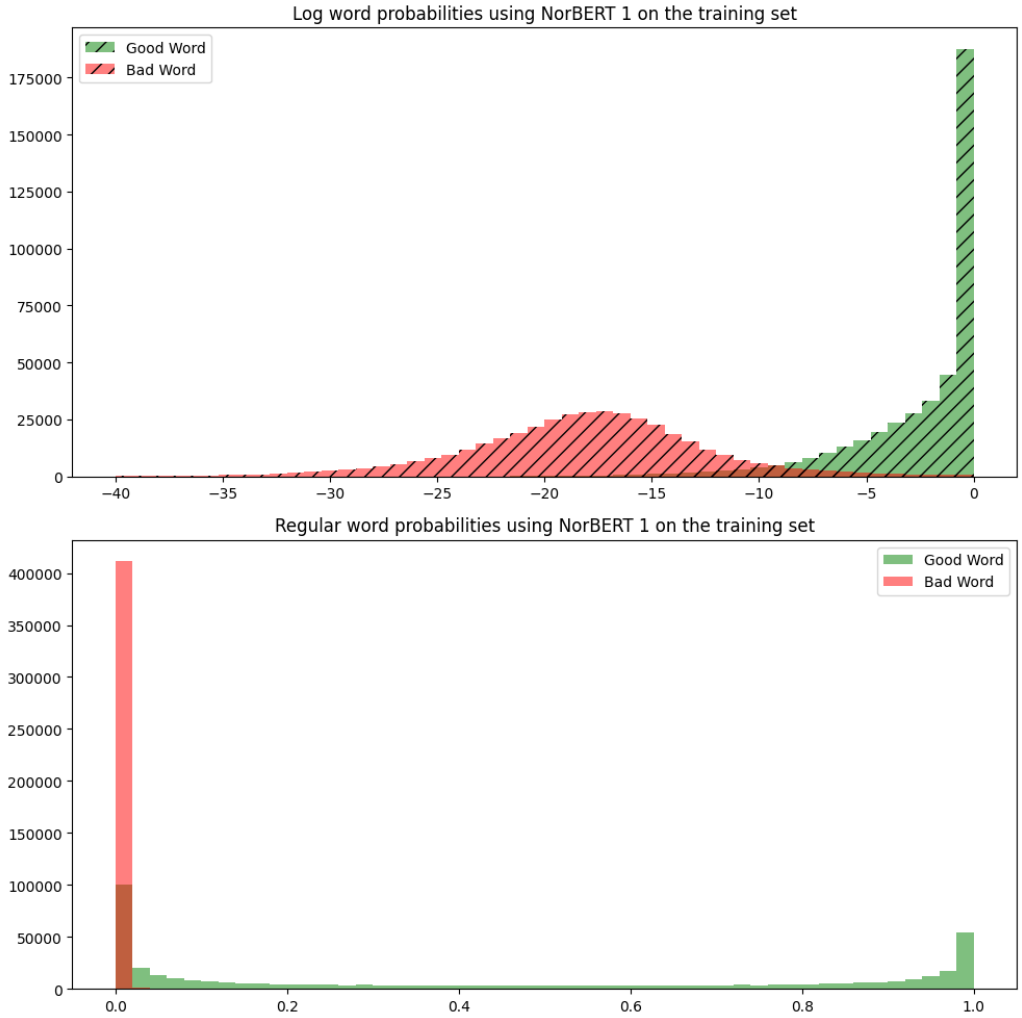


Figure A.2: Distribution of regular and logarithmic probabilities for good words and bad words using NorBERT1 on the training set.

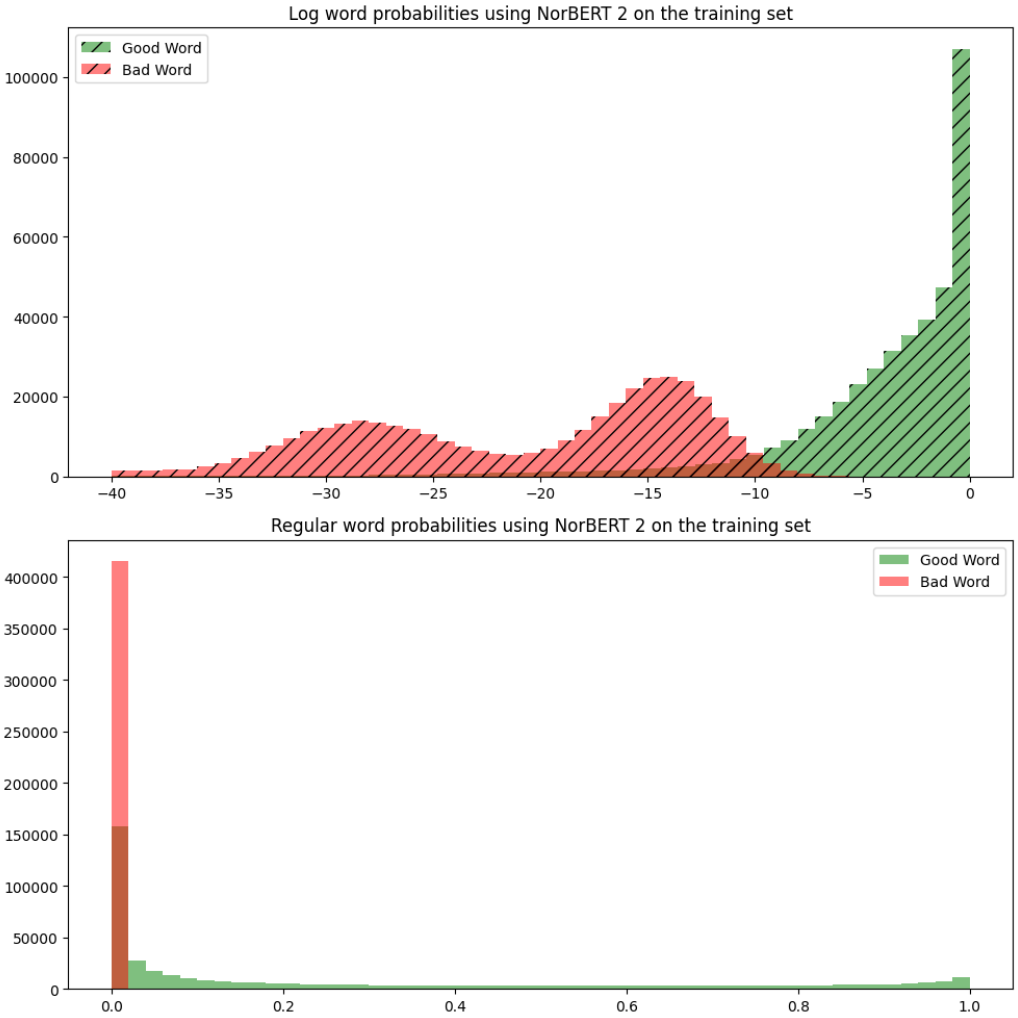


Figure A.3: Distribution of regular and logarithmic probabilities for good words and bad words using NorBERT2 on the training set.

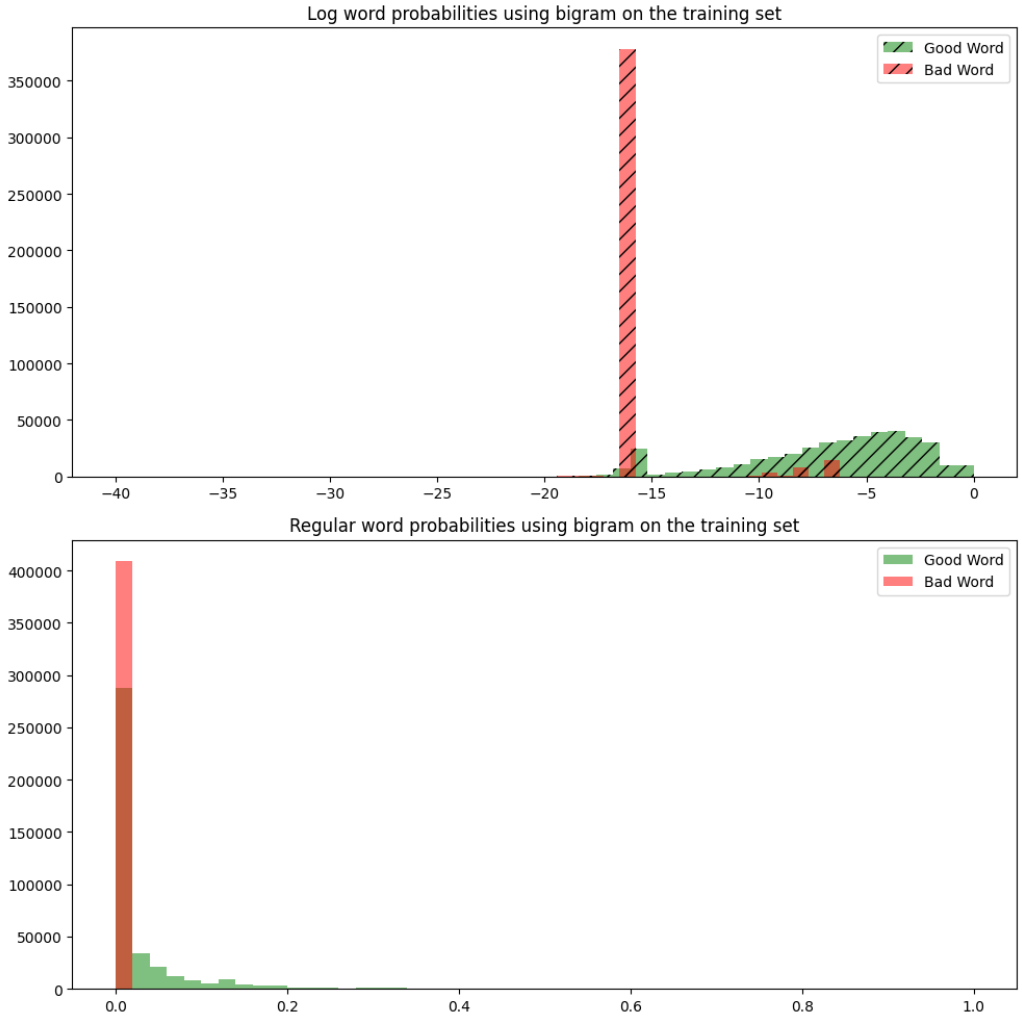


Figure A.4: Distribution of regular and logarithmic probabilities for good words and bad words using the bigram model on the training set.

Bibliography

Akbik, A., D. Blythe, and R. Vollgraf (2018), Contextual string embeddings for sequence labeling, in *Proceedings of the 27th International Conference on Computational Linguistics*, pp. 1638–1649, Association for Computational Linguistics, Santa Fe, New Mexico, USA. 2.3

Akbik, A., T. Bergmann, D. Blythe, K. Rasul, S. Schweter, and R. Vollgraf (2019), FLAIR: An easy-to-use framework for state-of-the-art NLP, in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, pp. 54–59, Association for Computational Linguistics, Minneapolis, Minnesota, doi:10.18653/v1/N19-4010. 2.3

Anantaram, C., A. Sangroya, M. Rawat, and A. Chhabra (2018), Repairing asr output by artificial development and ontology based learning, in *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, IJCAI'18, p. 57995801, AAAI Press. 2.3

Baevski, A., H. Zhou, A. Mohamed, and M. Auli (2020), wav2vec 2.0: A framework for self-supervised learning of speech representations. 2.3

Basile, V., C. Bosco, E. Fersini, D. Nozza, V. Patti, F. M. Rangel Pardo, P. Rosso, and M. Sanguinetti (2019), SemEval-2019 task 5: Multilingual detection of hate speech against immigrants and women in Twitter, in *Proceedings of the 13th International Workshop on Semantic Evaluation*, pp. 54–63, Association for Computational Linguistics, Minneapolis, Minnesota, USA, doi:10.18653/v1/S19-2007. 2.3

Bassignana, E., V. Basile, and V. Patti (2018), Hurtlex: A multilingual lexicon of words to hurt, in *Italian Conference on Computational Linguistics*. 3, 3.2, 4.3.1

Caselli, T., V. Basile, J. Mitrovic, and M. Granitzer (2020a), Hatebert: Retraining BERT for abusive language detection in english, *CoRR*, *abs/2010.12472*. 2.3

Caselli, T., V. Basile, J. Mitrović, I. Kartoziya, and M. Granitzer (2020b), I feel offended, don't be abusive! implicit/explicit messages in offensive and abusive language, in *Proceedings of the Twelfth Language Resources and Evaluation Conference*, pp. 6193–6202, European Language Resources Association, Marseille, France. 2.3

Cañete, J., G. Chaperon, R. Fuentes, J.-H. Ho, H. Kang, and J. Pérez (2020), Spanish pre-trained bert model and evaluation data, in *PML4DC at ICLR 2020*. 2.3

Chan, B., S. Schweter, and T. Möller (2020), German's next language model, in *Proceedings of the 28th International Conference on Computational Linguistics*, pp. 6788–6796, International Committee on Computational Linguistics, Barcelona, Spain (Online), doi:10.18653/v1/2020.coling-main.598. 2.3

Chan, W., D. Park, C. Lee, Y. Zhang, Q. Le, and M. Norouzi (2021), Speechstew: Simply mix all available speech recognition data to train one large neural network. 2.3

de Vries, W., A. van Cranenburgh, A. Bisazza, T. Caselli, G. van Noord, and M. Nissim (2019), Bertje: A dutch BERT model, *CoRR*, *abs/1912.09582*. 2.5

Devlin, J., M. Chang, K. Lee, and K. Toutanova (2018), BERT: pre-training of deep bidirectional transformers for language understanding, *CoRR*, *abs/1810.04805*. 2.4, 2.4.1, 2.5, 3.1.1, 4.1, 4.1

Du, J., S. Pu, Q. Dong, C. Jin, X. Qi, D. Gu, R. Wu, and H. Zhou (2022), Cross-modal asr post-processing system for error correction and utterance rejection, doi: 10.48550/ARXIV.2201.03313. 2.3

Dutta, S., S. Jain, A. Maheshwari, G. Ramakrishnan, and P. Jyothi (2022), Error correction in ASR using sequence-to-sequence models, *CoRR*, *abs/2202.01157*. 2.3

DHaro, L. F., and R. E. Banchs (2016), Automatic Correction of ASR Outputs by Using Machine Translation, in *Proc. Interspeech 2016*, pp. 3469–3473, doi: 10.21437/Interspeech.2016-299. 2.3

Enarvi, S., M. Amoia, M. Del-Agua Teba, B. Delaney, F. Diehl, S. Hahn, K. Harris, L. McGrath, Y. Pan, J. Pinto, L. Rubini, M. Ruiz, G. Singh, F. Stemmer, W. Sun, P. Vozila, T. Lin, and R. Ramamurthy (2020), Generating medical reports from patient-doctor conversations using sequence-to-sequence models, in *Proceedings of the First Workshop on Natural Language Processing for Medical Conversations*, pp. 22–30, Association for Computational Linguistics, Online, doi: 10.18653/v1/2020.nlpmc-1.4. 2.3

Errattahi, R., A. El Hannani, and H. Ouahmane (2018), Automatic speech recognition errors detection and correction: A review, *Procedia Computer Science*, *128*, 32–37, doi:https://doi.org/10.1016/j.procs.2018.03.005, 1st International Conference on Natural Language and Speech Processing. 2.2, 2.2.1

Farahani, M., M. Gharachorloo, M. Farahani, and M. Manthouri (2020), Parsbert: Transformer-based model for persian language understanding, *CoRR*, *abs/2005.12515*. 2.5

Hochreiter, S., and J. Schmidhuber (1997), Long short-term memory, *Neural computation*, *9*, 1735–80, doi:10.1162/neco.1997.9.8.1735. 2.3

Hrinchuk, O., M. Popova, and B. Ginsburg (2019), Correction of automatic speech recognition with transformer sequence-to-sequence model, *CoRR*, *abs/1910.10697*. 2.3

Hsu, W.-N., B. Bolte, Y.-H. H. Tsai, K. Lakhotia, R. Salakhutdinov, and A. Mohamed (2021), Hubert: Self-supervised speech representation learning by masked prediction of hidden units. 2.3

Husnjak, S., D. Perakovic, and I. Jovovic (2014), Possibilities of using speech recognition systems of smart terminal devices in traffic environment, *Procedia Engineering*, *69*, 778–787. 2.1

Jurafsky, D., and J. Martin (2021), *Speech and Language Processing An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. 2.1, 2.2.1, 3.1, 4.1, 4.2, 4.3, 4.3.1, 5

Koutsikakis, J., I. Chalkidis, P. Malakasiotis, and I. Androutsopoulos (2020), GREEK-BERT: the greeks visiting sesame street, *CoRR*, *abs/2008.12014*. 2.5

Kummervold, P. E., J. de la Rosa, F. Wetjen, and S. A. Brygfjeld (2021), Operationalizing a national digital library: The case for a norwegian transformer model, *CoRR*, *abs/2104.09617*. 1, 2.5, 4.3

Kutuzov, A., J. Barnes, E. Velldal, L. Øvrelid, and S. Oepen (2021), Large-scale contextualised language modelling for norwegian, *CoRR*, *abs/2104.06546*. 1, 2.5

Leng, Y., X. Tan, L. Zhu, J. Xu, R. Luo, L. Liu, T. Qin, X. Li, E. Lin, and T. Liu (2021a), Fastcorrect: Fast error correction with edit alignment for automatic speech recognition, *CoRR*, *abs/2105.03842*. 2.3

Leng, Y., X. Tan, R. Wang, L. Zhu, J. Xu, L. Liu, T. Qin, X. Li, E. Lin, and T. Liu (2021b), Fastcorrect 2: Fast error correction on multiple candidates for automatic speech recognition, *CoRR*, *abs/2109.14420*. 2.3

Leng, Y., X. Tan, W. Liu, K. Song, R. Wang, X.-Y. Li, T. Qin, E. Lin, and T.-Y. Liu (2022), Softcorrect: Error correction with soft detection for automatic speech recognition, *arXiv preprint arXiv:2212.01039*. 2.3

Lewis, M., Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer (2019), Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. 2.3

Li, J., V. Lavrukhin, B. Ginsburg, R. Leary, O. Kuchaiev, J. M. Cohen, H. Nguyen, and R. T. Gadde (2019), Jasper: An end-to-end convolutional neural acoustic model. 2.3

Liao, J., S. E. Eskimez, L. Lu, Y. Shi, M. Gong, L. Shou, H. Qu, and M. Zeng (2020), Improving readability for automatic speech recognition transcription, *CoRR*, *abs/2004.04438*. 2.3

Liu, Y., M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov (2019), Roberta: A robustly optimized bert pretraining approach. 2.3

Malmsten, M., L. Börjeson, and C. Haffenden (2020), Playing with words at the national library of sweden - making a swedish BERT, *CoRR*, *abs/2007.01658*. 2.5

Mani, A., S. Palaskar, N. V. Meripo, S. Konam, and F. Metze (2020), Asr error correction and domain adaptation using machine translation, *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 6344–6348. 2.3

Martin, L., B. Muller, P. J. O. Suá rez, Y. Dupont, L. Romary, É. de la Clergerie, D. Seddah, and B. Sagot (2020), CamemBERT: a tasty french language model, in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Association for Computational Linguistics, doi:10.18653/v1/2020.acl-main.645. 2.3

Mccowan, I., D. Moore, J. Dines, D. Gatica-Perez, M. Flynn, P. Wellner, and H. Bourlard (2004), On the use of information retrieval measures for speech recognition evaluation. 2.2.1

Meripo, N. V., and S. Konam (2022), Asr error detection via audio-transcript entailment, doi:10.48550/ARXIV.2207.10849. 2.3

Mokhtar, K., S. S. Bukhari, and A. R. Dengel (2018), Ocr error correction: State-of-the-art vs an nmt-based approach, *2018 13th IAPR International Workshop on Document Analysis Systems (DAS)*, pp. 429–434. 2.3

Morris, A., V. Maier, and P. Green (2004), From wer and ril to mer and wil: improved evaluation measures for connected speech recognition, doi:10.21437/Interspeech.2004-668. 2.2.1

Nanayakkara, G., N. Wiratunga, D. Corsar, K. Martin, and A. Wijekoon (2022a), Clinical dialogue transcription error correction using seq2seq models, doi:10.48550/ARXIV.2205.13572. 2.1

Nanayakkara, G., N. Wiratunga, D. Corsar, K. Martin, and A. Wijekoon (2022b), Clinical dialogue transcription error correction using seq2seq models, doi:10.48550/ARXIV.2205.13572. 2.3

Nangia, N., C. Vania, R. Bhalerao, and S. R. Bowman (2020), CrowS-pairs: A challenge dataset for measuring social biases in masked language models, in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1953–1967, Association for Computational Linguistics, Online, doi:10.18653/v1/2020.emnlp-main.154. 4.3

Narayanan, A., A. Misra, K. C. Sim, G. Pundak, A. Tripathi, M. Elfeky, P. Haghani, T. Strohman, and M. Bacchiani (2018), Toward domain-invariant speech recognition via large scale training. 2.3

Nguyen, D. Q., and A. T. Nguyen (2020), Phobert: Pre-trained language models for vietnamese, *CoRR*, *abs/2003.00744*. 2.5

Panayotov, V., G. Chen, D. Povey, and S. Khudanpur (2015), Librispeech: An asr corpus based on public domain audio books, in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5206–5210, doi:10.1109/ICASSP.2015.7178964. 2.3

Raffel, C., N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu (2019), Exploring the limits of transfer learning with a unified text-to-text transformer, doi:10.48550/ARXIV.1910.10683. 2.3

Sabry, S. S., T. Adewumi, N. Abid, G. Kovacs, F. Liwicki, and M. Liwicki (2022), Hat5: Hate language identification using text-to-text transfer transformer, doi:10.48550/ARXIV.2202.05690. 2.3

Salazar, J., D. Liang, T. Q. Nguyen, and K. Kirchhoff (2020), Masked language model scoring, in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Association for Computational Linguistics, doi:10.18653/v1/2020.acl-main.240. 4.3

Shen, K., Y. Leng, X. Tan, S. Tang, Y. Zhang, W. Liu, and E. Lin (2022), Mask the correct tokens: An embarrassingly simple approach for error correction, *arXiv preprint arXiv:2211.13252*. 2.3

Solberg, P. E., and P. Ortiz (2022), The norwegian parliamentary speech corpus, *CoRR*, *abs/2201.10881*. 1, 3, 3.1, 3.1.1

Song, K., X. Tan, and J. Lu (2020), Neural machine translation with error correction. 2.3

Taylor, W. L. (1953), cloze procedure: A new tool for measuring readability, *Journalism & Mass Communication Quarterly*, *30*, 415 – 433. 2.4.1

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin (2017), Attention is all you need, *CoRR*, *abs/1706.03762*. 2.3, 2.4, 2.4.1

Virtanen, A., J. Kanerva, R. Ilo, J. Luoma, J. Luotolahti, T. Salakoski, F. Ginter, and S. Pyysalo (2019), Multilingual is not enough: BERT for finnish, *CoRR*, *abs/1912.07076*. 2.5

Wang, A., A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman (2018), GLUE: A multi-task benchmark and analysis platform for natural language understanding, *CoRR*, *abs/1804.07461*. 2.4.1

Wu, Y., M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, L. Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean (2016), Google's neural machine translation system: Bridging the gap between human and machine translation, *CoRR*, *abs/1609.08144*. 2.5, 4.3

Xu, H.-D., Z. Li, Q. Zhou, C. Li, Z. Wang, Y. Cao, H. Huang, and X.-L. Mao (2021), Read, listen, and see: Leveraging multimodal information helps Chinese spell checking, in *Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021*, pp. 716–728, Association for Computational Linguistics, Online, doi:10.18653/v1/2021.findings-acl.64. 2.3

Zampieri, M., S. Malmasi, P. Nakov, S. Rosenthal, N. Farra, and R. Kumar (2019), SemEval-2019 task 6: Identifying and categorizing offensive language in social media (OffensEval), in *Proceedings of the 13th International Workshop on Semantic Evaluation*, pp. 75–86, Association for Computational Linguistics, Minneapolis, Minnesota, USA, doi:10.18653/v1/S19-2010. 2.3

Zitkiewicz, T. (2022), Tag and correct: high precision post-editing approach to correction of speech recognition errors, in *2022 17th Conference on Computer Science and Intelligence Systems (FedCSIS)*, pp. 939–942, doi:10.15439/2022F168. 2.3