Classifying recognised speech with deep neural networks

RA Strydom

orcid.org 0000-0002-4364-2148

Dissertation accepted in fulfilment of the requirements for the degree Master of Engineering in Computer and Electronic Engineering at the North-West University

Supervisor: Prof E Barnard

Graduation: June 2021

Student number: 33487642

Declaration

I, Rhyno Strydom hereby declare that the dissertation entitled "Classifying recognised speech with deep neural networks" is my own original work and has not already been submitted to any other university or institution for examination.

R.A. Strydom

Student number: 33487642

Signed on the 25th day of November 2020 at Potchefstroom.

Acknowledgements

We thank the Novus Group for providing the data needed for this investigation. Also a special thanks must be given to Dr Charl van Heerden at Saigen for facilitating this arrangement and producing the speech-recognition results.

Writing acknowledgements is both a privilege and a fear. I am grateful for the chance to thank the many people who helped me to complete my master's degree but I am also scared that I'm going to miss out someone important. So, to pre-empt that, thank you to ALL my dear family and friends for having been there for me as I went through this slightly mad process of researching and writing my dissertation.

I believe that people don't come to me — they are sent to me. One such an extraordinary person sent to me is Prof Etienne Barnard to whom I offer my eternal gratitude. Thank you Prof for having kept me grounded while I waded through unknown territory. Your guidance, advice and admonishments helped me to stay on course. Thank you for suffering my endless queries and concerns.

Special thanks to Wian Snyman who put up with my constant whining of Why? How? Show me again. Thank you for sharing your knowledge and expertise in a calm and controlled manner.

To undertake a master's degree one needs a formidable person to steer you through a maze of administrative detail. Such a person is Ulrike Janke. Thank you for your generosity, time and patience, which helped me to achieve my goals.

I am also deeply indebted to Prof Marelie Davel for her assistance in helping me to understand the dynamics of a study group. Prof, I value your commitment.

I am grateful to Tian Theunissen. He was my friend, ally and pillar of strength during many hours of darkness — whether late at night or early mornings. Thank you for your patience and honesty.

To Nuette Heyns — my partner in crime — thank you for your friendship and support. You remained positive even during lock-down.

Last but not least — my family — they are the people to whom I owe the greatest debt. These amazing, unique and one-of-a-kind humans have helped me to discover an answer where there did not always seem to be one, discover that it is not what happens that matters, it is what you do with it. Thank you for unceasing moral support and loving comfort.

Abstract

We investigate whether word embeddings using deep neural networks can assist in the analysis of text produced by a speech-recognition system. In particular, we develop algorithms to identify which words are incorrectly detected by a speech-recognition system in broadcast news. The multilingual corpus used in this investigation contains speech from the eleven official South African languages, as well as Hindi. Popular word embedding algorithms such as word2vec and fastText are investigated and compared with context-specific embedding representations such as doc2vec and non-context specific statistical sentence embedding methods such as term frequency-inverse document frequency (TF-IDF), which is used as our baseline method. These various embedding methods are then used as fixed length input representations for a logistic regression and feedforward neural network classifier. The output is used as an additional categorical input feature to a CatBoost classifier to determine whether the words were correctly recognised. Other methods are also investigated, including a method that uses the word embedding itself and cosine similarity between specific keywords to identify whether a specific keyword was correctly detected. When relying only on the speech-text data, the best result was obtained using the TF-IDF document embeddings as input features to a feedforward neural network. Adding the output from the feedforward neural network as an additional feature to the CatBoost classifier did not enhance the classifier's performance compared to using the non-textual information provided, although adding the output from a weaker classifier was somewhat beneficial.

Keywords: *word embeddings, word2vec, fastText, doc2vec, TF-IDF, Deep Neural Networks, CatBoost*

Contents

List of Figures

List of Tables

List of Acronyms

ASR	automatic speech recognition system
AUC	area under the curve
Adam	adaptive moment estimation
BERT	bidirectional encoder representations from transformers
BoW	bag-of-words
CBoW	continuous bag-of-words
CNN	convolutional neural networks
DNN	deep neural networks
ELMo	embedding from language models
FN	false negative
FP	false positive
FPR	false positive rate
GloVe	global vectors for word representation
GPT-2	generative pretrained transformer-2
LSTM	long short-term memory
MLP	multi-layered perceptron
MLM	masked language model
MuST	Multilingual Speech Technologies
NLP	natural language processing
PCA	principal component analysis
PV-DBoW	distributed bag-of-words of paragraph vectors
PV-DM	distributed memory model of paragraph vectors

RNN	recurrent neural networks
ROC	receiver operating characteristic
ReLU	rectified linear unit
SG	skip-gram
SHAP	Shaply additive explanation
SVM	support vector machine
TF-IDF	term frequency-inverse document frequency
TF-WIDF	term frequency-weighted inverse document frequency
TF-MIDF	term frequency-modified inverse document frequency
TN	true negative
TNR	true negative rate
TP	true positive
TPR	true positive rate
t-SNE	t-distributed stochastic neighbourhood embedding

Chapter 1

Introduction

This chapter introduces the objectives of the study why it is relevant and interesting. The scope is discussed, as well as the research questions and objectives.

1.1 Background

Modern developments in natural language processing (NLP) have produced promising performance, and the widespread adoption of deep neural networks (DNNs) has played an important role in achieving this. In the past, statistical language modelling methods such as n-grams, dominated the field of NLP [1]. With larger corpora being made available on the internet in recent years, these methods have drastically improved, but have reached a saturation point resulting in only minor advances. The most significant limitation of these types of methods is that they are purely statistical and fail to capture regularities such as the linguistic structure within languages.

More recent improvements in the field include the development of machine learning-based approaches [2] to solve NLP problems. This, however, resulted in shallow mod-

els, trained on handcrafted, high-dimensional sparse feature vectors. These vectors are time-consuming to construct in many cases and are incomplete representations of the data. Over the last few years, however, DNNs based on dense feature vector representations have produced superior results in NLP tasks, mainly owing to the success of word embeddings and deep learning methods.

Mikolov's word2vec [3] demonstrated how representing words as continuous vectors (word embeddings) outperformed traditional statistical language modelling techniques and was able to capture the semantic relationship between words. An illustration of this can be seen in Figure 1.1 below. Other techniques, such as the "one-hot encoding of words", suffer from being extremely sparse, resulting in high-dimensional feature vectors, that are unable to fully capture the semantic relationships between words. Word embeddings are an attempt to remedy this shortcoming by representing words as dense, low-dimensional feature vectors.

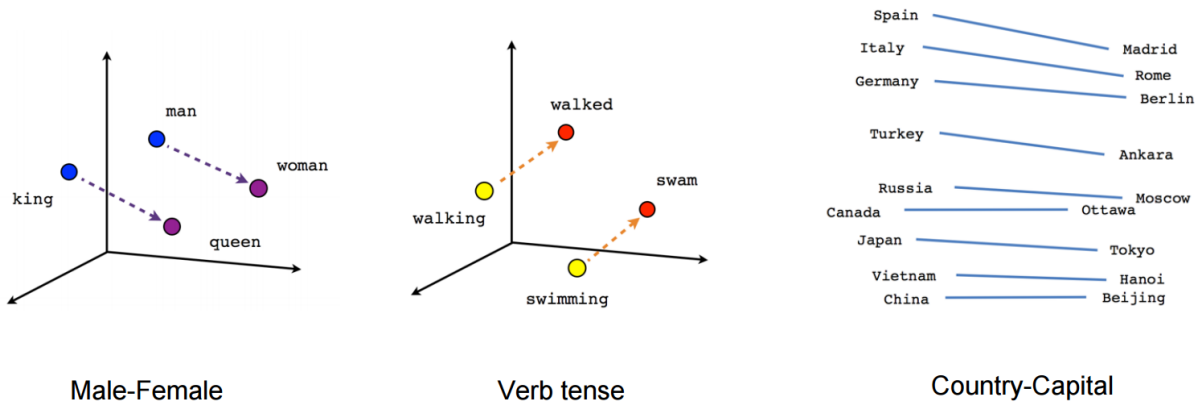


Figure 1.1: Word vectors represented in Euclidean space are able to capture semantic relationships between other word vectors in the same space [4].

Numerous other techniques for representing words as vectors ensued such as global vectors for word representation (GloVe) [5] where the word embeddings were trained based on their co-occurrence with other words in the corpus. Facebook's fastText [6], learning each character in a word's orthographic representation, helps to generate better embeddings for out-of-corpus words, given that orthographic n-grams are able to capture some key structural components of words. These different methods are, how-

ever, all related in that they depend on the same distributional hypothesis [7], which state: "words that appear in the same context share semantic relationships with each other."

In certain NLP tasks such as sentiment analysis, just understanding the relationship between different words alone is not enough to capture the true meaning from text. The methods used to train simple word vectors ignore local word ordering [8]. If for example we have the sentences "the food was not unpleasant, it was fairly good" and "the food was not good, it was fairly unpleasant", the local ordering of the words is important for capturing the separate meaning of each phrase (the word "not" comes before the word "unpleasant"). A possible solution to this problem is to use DNN architectures such as convolutional neural networks (CNNs) [9] and recurrent neural networks (RNNs), to retain local word order in sentences. These DNN architectures are able to capture both low- and high-level features of language, allowing deeper semantic meaning to be extracted.

Expanding on word embeddings, Mikolov demonstrated that it is possible to train vector representations for variable-length text fragments such as paragraphs and documents [10]. These embeddings could then be used for text classification tasks such as sentiment analysis. Where traditional methods, such as representing a sentence as a bag-of-words (BoW), do not indicate the semantic relationship between words in a sentence or word order being lost in sentences that use similar words, Mikolov shows that it is possible to train document or paragraph embeddings to predict neighbouring words in context of the sampled paragraph.

In this research, embeddings are used to analyse text that is produced by an automatic speech-recognition (ASR) system, in order to develop algorithms to improve the identification of words in broadcast news, this is made more clear in Appendix A. The application of deep learning methods to speech data is particularly relevant, as words that correlate strongly with related entities in the news can be identified better. The commercial application motivating this focus helps advertising companies ensure that the advertisements they paid for were broadcast during the appropriate time of day.

1.2 Motivation

Applying textual analysis to radio broadcast data can improve false positive (FP) identification of certain entities mentioned in such broadcasts, for example, helping advertising agencies know if their advertisements were aired at the right time. In recent developments in the field of NLP there has been a move away from methods such as BoW or bag-of-n-grams models, towards the use of embeddings and deep learning based techniques. We therefore propose to investigate the use of deep learning methods to process recognised speech in order to improve the identification of FPs.

Given this problem, the initial scope of the research project is limited to data obtained from an affiliate of the Multilingual Speech Technologies (MuST) research group. Saigen[1], a commercial vendor of speech-recognition technology and affiliate of the MuST research group, has supplied broadcast media (i.e. radio and television) audio data and the corresponding speech-recogniser output, which will be used to analyse the various techniques. The focus of this investigation is limited to word2vec, fastText and doc2vec using one-layer DNNs with varying widths, a logistic regression and a CatBoost classifier.

1.3 Research Questions

With the aim of improving FP identification in recogniser outputs with DNNs. We formulate the following research questions:

- Is it possible to use word embeddings to identify FP words in recognised speech?
- Is there a way to capture context with embeddings to help identify FP words more accurately?
- How can we use these trained word embeddings alongside a DNN classifier to

[1]https://www.saigen.co.za/

determine whether the words were correctly identified?

1.4 Objectives Of The Study

Considering the research questions, the objectives of the study include the following:

- Determine suitable word embedding techniques to identify FP words in speech data.
- Investigate the effectiveness of word and document embeddings with regards to identifying false positive words in speech data.
- Find appropriate hyperparameters for DNN architectures to compare the performance of using different types of word embedding techniques, with particular attention to the limited amount of data available.

1.5 Research methodology

This study consists of applied investigative research based on previous DNN NLP architectures used in common NLP tasks. The research is done through empirical experiments using different architectures and techniques, applied to the output of an existing speech-recognition system. The following steps will be performed as part of the research process:

- **Literature review:** Background reading relating to different types of DNN NLP architectures used for text classification, will be done. We will investigate how to use various types of word embedding techniques on domain-specific datasets to capture semantic relationships.
- **Code base development:** A code base needs to be developed for preprocessing, training and evaluation of the recogniser outputs.

- **Experimental development:** This step involves determining appropriate training hyperparameters for DNN architectures to compare results of different types of embeddings for FP detection of recogniser outputs, as well as analysing results and comparing results and compare different NLP approaches, to understand FP detection in such data.
- **Assess and discuss the experimental findings:** Different DNN architectures alongside their embeddings will be evaluated and discussed.

1.6 Dissertation overview

The remainder of this dissertation consists of the following chapters:

- **Chapter 2: Related Work**. We review background literature and detail the necessary information regarding word embeddings and natural language processing techniques.
- **Chapter 3: Data Analysis and Exploration**. We investigate the recognised speech-text data and the relevant preprocessing techniques used before these are used to train word embeddings and do text classification.
- **Chapter 4: Experimental Procedure**. We explain how the word embeddings are trained and used as input to the various classifiers. Additionally, we explain how these different word embeddings are evaluated alongside these classifiers.
- **Chapter 5: Experimental Results**. We compare and discuss the results of using the different types of embeddings with the various classifiers.
- **Chapter 6: Conclusion and Recommendations**. We discuss how objectives were met and summarise key findings. We also make some suggestions for future work.

1.7 Publications

A paper named "Classifying recognised speech with deep neural networks", was accepted for presentation at the *2020 Southern African Conference for Artificial Intelligence Research*. The paper submitted condenses some of the key topics discussed in this document.

Chapter 2

Related Work

In this chapter, we detail background information regarding the study and introduce relevant literature on word embeddings and natural language processing in general.

2.1 Introduction

This chapter gives a summary of the different types of word and document embeddings examined throughout the study and how they work. We show different methods used to train and evaluate word and document embeddings, explaining how embeddings can be used with various machine learning classifiers.

2.2 Statistical embedding methods for documents

2.2.1 Bag-of-words

One of the earliest and most common methods used to represent documents as fixed length vector representations is BoW [2]. This method represents text based on the frequency of occurrence of words in a document, neglecting any word order, hence the name. The method tries to emphasise the idea that similar documents would have similar terms, and from these similar terms some meaning between documents can be derived. An example of such a document representation can be seen in Table 2.1, where "doc 1" depicts the sentence: "the fox jumped over the dog" and "doc 2" shows: "the fox jumped". Methods relating to how these vectors can be used to classify text are expanded upon in Section 2.6.

One of the caveats related to this method is that it does not contain any semantic information regarding words in a document. Using this method also neglects ordering of words and since each word is equally weighted, words are seen as equally important even though certain words (e.g. the word "bad" in a review) are more indicative of the central idea being conveyed in the document.

Table 2.1: BoW example, showing the frequency of each word in the document.

	the	quick	brown	fox	jumped	over	lazy	dog
doc 1	2	0	0	1	1	1	0	1
doc 2	1	0	0	1	1	0	0	0

2.2.2 Term frequency-inverse document frequency

One of the main problems of the BoW method is that it gives equal weighting to each word in the document, which can cause numerous noisy, purposeless terms to be captured. Term frequency-inverse document frequency (TF-IDF) [11], a modified method

related to BoW, is used in an attempt to remedy this shortcoming. TF-IDF is a way to represent the importance of a word in a document when examining the corpus as a whole; the statistic is calculated as:

$$w_{i,j} = tf_{i,j} \times \log(\frac{N}{df_i}) \tag{2.1}$$

where: $w_{i,j}$ = TF-IDF of word i in document j
$tf_{i,j}$ = term frequency of word i in document j
N = the number of documents
df_i = number of times word i occurs in all documents.

Term frequency refers to the number of times a specific word appears in a document divided by the number of words in a document, while inverse document frequency indicates the importance of the word and is calculated by taking the number of documents in the corpus and dividing it by the number of times a specific word appears in the corpus. This has the effect of rarer words having a larger numerical value assigned to them, the intuition being that rarer words carry more meaning in a document and should therefore be assigned a larger numerical value than words that appear more frequently and carry no meaning.

Another common technique is weighted inverse document frequency (WIDF) [12], obtained by dividing the term frequency of a word in a document with the sum of the total number of times that term appears in all documents as shown by:

$$w_{i,j} = tf_{i,j} \times \frac{1}{\sum_{j=1}^{D} tf_{i,j}} \tag{2.2}$$

where: $w_{i,j}$ = TF-WIDF of word i in document j
$tf_{i,j}$ = term frequency of word i in document j
D = the number of documents

The authors claim the method to be superior in text categorisation tasks, as IDF only indicates whether or not a specific word is present in document in relation to the number of documents the word appeared in (e.g. count one if the word was present in a

document). The value is not indicative of the frequency of a specific word within a given document considering the frequency of the word in all documents.

One of the main disadvantages related to WIDF is that as the number of documents present in the corpus increases, words that have a similar frequency of occurrence have almost equal weight. To repress this effect over larger corpora with more documents IDF and WIDF are combined - a method known as modified inverse document frequency (MIDF) [13]. The authors claim this has a normalizing effect of the term frequency over the corpus as whole. Improving classification results on tasks such as text categorisation, the TF-MIDF value is calculated using:

$$w_{i,j} = tf_{i,j} \times \frac{df_i}{\sum_{j=1}^{D} tf_{i,j}} \tag{2.3}$$

where: $w_{i,j}$ = TF-MIDF of word i in document j
$tf_{i,j}$ = term frequency of word i in document j
df_i = number of times word i occurs in all documents.
D = the number of documents

We limit our study to using TF-IDF as our baseline method, as it is commonly used in literature [14]. Both TF-IDF and the BoW method can be optimised by constraining their dictionaries to a limited number of words based on their frequency of occurrence. Restricting the vocabulary size allows the model to disregard extremely rare words that might be present in the corpus adding additional unwanted dimensions to these document vectors. Another common way to improve the performance of these methods is using a bag of n-grams, such as “good morning” (bigram), “yes we can” (trigram) and “can you please stop” (four-gram). This allows some context to be captured between documents, using words that commonly appear together. Using either the BoW or TF-IDF method, however, still results in a sparse document representation neglecting word order in a sentence.

2.3 Word embeddings

Unlike statistical methods such as BoW or TF-IDF, word embeddings are able to represent words as low-dimensional representations and are able to capture the semantic relationship between words. This section explores two methods, namely fastText and word2vec.

2.3.1 Word2vec

Word2vec [3] is used to represent words as continuous vector representations. The algorithm has two variants, namely continuous bag-of-words (CBoW) and skip-gram (SG). CBoW maximises the probability of the target word by looking at the context words, thus given a set of words at its input it aims to predict a specific word at its output. SG follows the inverse rule to this. To give intuition concerning the mechanics behind these methods, SG is used as an example.

The technique is designed to maximise the average log probability by predicting context words denoted by w_{t+j}, given a specific target word denoted as w_t, as shown by:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0}\log p\left(w_{t+j} \mid w_t\right) \tag{2.4}$$

where: T = the number of training words in the sequence of words around the context window
w_t = the centre word
w_{t+j} = the words left and right of the centre word
c = the size of the training context window.

The probability of a target word predicting a context word shown as $p(w_{t+j} \mid w_t)$ can be defined using the softmax function as

$$p\left(w_O \mid w_I\right) = \frac{\exp\left({v'_{w_O}}^{\top} v_{w_I}\right)}{\sum_{w=1}^{W}\exp\left({v'_w}^{\top} v_{w_I}\right)} \tag{2.5}$$

where: v_w = refers to the input vector representation of the word
v'_w = refers to the output vector representation of the word.
W = refers to all words to be predicted at the output.

SG typically works better for rare words and smaller amounts of data. Each word in a sentence is used as a pairwise training example, where CBoW averages the context words to predict a word. For larger datasets CBoW obtains higher accuracy for frequent words. It also trains faster than the SG method. Mikolov [3] claims that CBoW obtains higher accuracy on syntactic type tasks, whereas SG performs slightly worse on syntactic tasks but performs better on semantic tasks. A visual illustration of how each method works is shown in Figure 2.1.

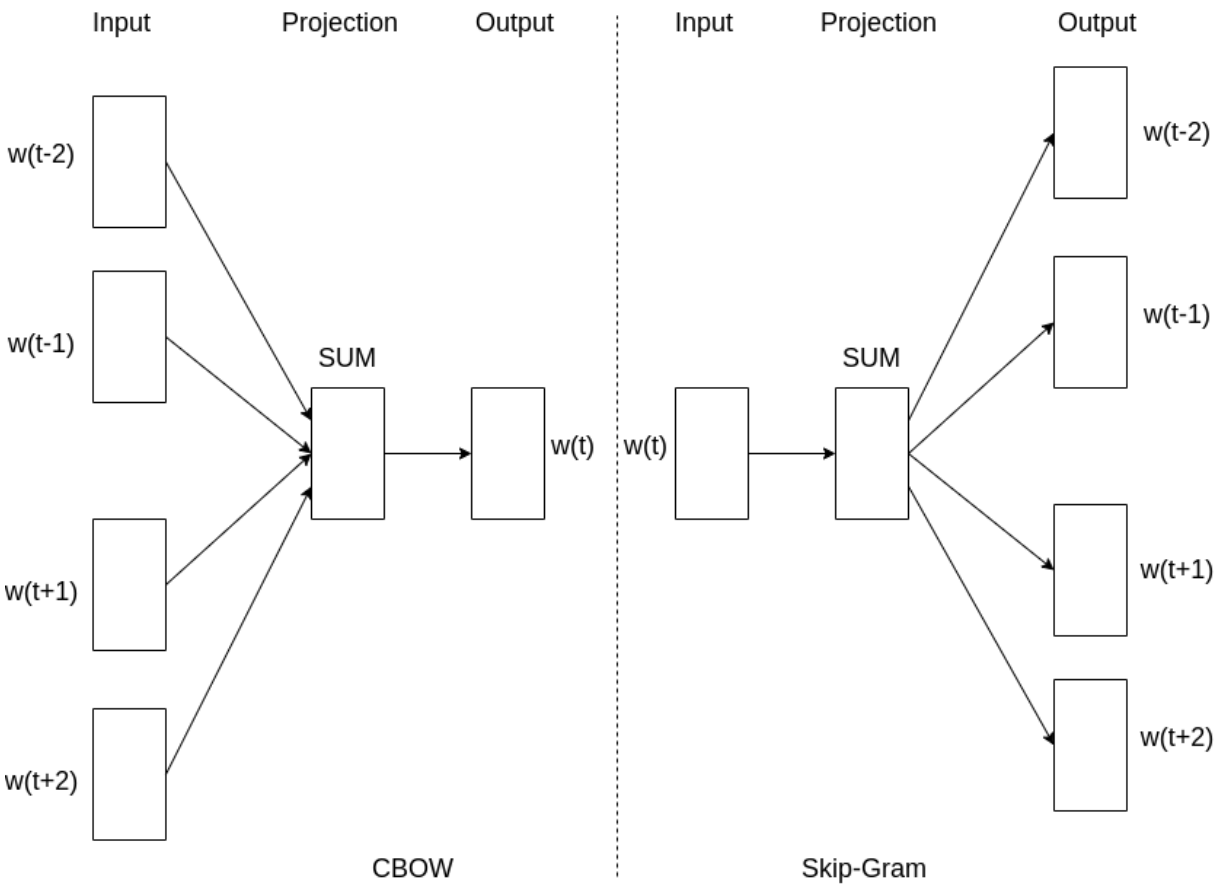


Figure 2.1: Diagram illustrating how to train CBoW and SG word embeddings. Note that for CBoW the learned weight matrix at the output is used to represent an embedding vector for a word, whereas SG uses the learned weight matrix at the input.

The main benefit of projecting words as n-dimensional embeddings is that a large number of words from large corpora of text can be represented efficiently as low-dimensional feature vectors. These embeddings can then be used as input features for machine learning algorithms in text classification tasks. It should be noted that to accurately capture useful word embeddings the embedding models needs to be trained on large corpora of text, typically in the range of millions or billions of words.

2.3.2 fastText

fastText [6] is a technique related to word2vec, in that it follows the same distributional hypothesis and can also be trained using either the CBoW or SG method. fastText claims to be able to generate better word embeddings for out-of-corpus words and morphologically rich languages, as it takes into account each character in a word's orthographic representation. This enriches the word embedding, given that orthographic n-grams are able to capture some key structural components of words. An example of such an n-gram for the word "dizzy" and n=3 is illustrated by the character n-grams:

$$< \text{di}, \text{diz}, \text{izz}, \text{zzy}, \text{zy} >$$

The brackets $<$ and $>$, placed at the boundaries of the word, are used to distinguish words from other words on which the model is trained. In practice n-grams are usually set to either greater than or equal to 3 or smaller than or equal to 6. These values tend to work well and we use them throughout our study.

2.3.3 Refining word vectors

Sub-sampling

Both word2vec and fastText use a probabilistic approach called sub-sampling to limit the number of frequent noisy words (e.g. stop words) presented during training. An

excess of such words would produce poor word vectors, since the frequent words would dominate the representation. Each word in the text has a probability associated with its frequency of occurrence where the word is more likely to be removed if it is very frequent, as illustrated by:

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}} \tag{2.6}$$

where: $P(w_i)$ = probability that a specific word is kept
$f(w_i)$ = refers to the frequency of occurrence of a specific word
t = refers to the chosen threshold.

A common threshold value "t" used in practice is 1e-5. The value is empirically shown by Mikolov [15] to work well. It ensures that words whose frequency is greater than this threshold value are forcefully sub-sampled. Having a minimum word count, thus limiting words to some minimum number of times they should appear in the corpus, also enhances the word embedding quality. Doing this, disregards very infrequent words that could be present owing to mispronunciations or corrupted speech-text detection examples.

Negative sampling

Training word embeddings typically require large datasets, with a rich vocabulary to train meaningful word vectors. This results in a large number of weight updates. If, for example, you have a vocabulary of 100 000 words and a 300-node neural network (i.e. the embedding dimension is 300), trying to map embeddings for those words would require about 30 000 000 weight updates during each epoch, which is very expensive to compute. Negative sampling [15] is an attempt to combat this by only updating a small number of negative words (i.e. words that are not in the context window and we would like to output 0). Negative sampling words that are selected to be updated are selected based on a unigram distribution, shown by Equation 2.7, where more frequent words are sampled more often than less frequent ones during each epoch. The "α" term is a common heuristic used in practice to smooth out the unigram distribution. This value

is used in an effort to combat the imbalance between common and rare words. A value of 0.75 is usually used for the term. Typically on large datasets negative sampling on two to five words is sufficient where smaller datasets require a larger number of negative samples, typically between five and twenty words.

$$P\left(w_i\right) = \frac{f\left(w_i\right)^{\alpha}}{\sum_{j=1}^{k} f\left(w_j\right)^{\alpha}} \tag{2.7}$$

where: $P\left(w_i\right)$ = probability that a specific word is chosen
$f(w_i)$ = refers to the frequency of occurrence of a specific word in the corpus
$f(w_j)$ = refers to the frequency of occurrence of all words in the corpus.
α = smoothing factor.

Pretrained embeddings

As training word embeddings typically requires large amounts of text data to get good vector representations for words, pretrained word embeddings are typically employed as a starting point for many NLP tasks. Google's word2vec model[1] is a popular choice. It is trained on the Google News dataset, using a 100 billion word corpus. Another good choice is using Facebook's fastText models[2] trained on wiki-news data, consisting of a million learned word vectors trained on 16 billion words, or their 2 million word vector common crawl dataset consisting of 600 billion words.

2.3.4 Evaluating word embeddings

Training word embeddings is an unsupervised learning task, requiring a clearly identified objective in order to evaluate how successful word embeddings are at classifying text (e.g. identifying FPs). Moreover, alterations must be made to the unsupervised

[1]https://code.google.com/archive/p/word2vec/
[2]https://fasttext.cc/docs/en/english-vectors.html

learning algorithm based on how well it performs on the task at hand, which is known as extrinsic evaluation [16]. Intrinsic evaluation typically occurs where the word embedding is evaluated on human judgment regarding word relations. A typical example of this would be a word analogy task such as:

$$\vec{\omega}_{king} - \vec{\omega}_{man} + \vec{\omega}_{woman} \approx \vec{\omega}_{queen} \tag{2.8}$$

In semantic similarity type tasks an assessor is typically given a set of words and probed to determine the level of similarity between pairs of words. Throughout the study extrinsic evaluation is considered to determine the quality of the word embeddings.

2.4 Document embeddings

Doc2vec [10] shows it is possible to represent documents of variable lengths as fixed length paragraph vectors. These document embeddings can then be used for text classification tasks such as sentiment analysis or predicting falsely detected words given a certain context. Document embedding also comes in two variants, namely distributed bag-of-words of paragraph vectors (PV-DBoW) and distributed memory model of paragraph vectors (PV-DM). PV-DM averages or concatenates the word and paragraph vectors to predict the next word in context, and the paragraph vector that is learned, acts as a memory that remembers what is missing from the current context. PV-DBoW, on the other hand, forces the model to predict sampled words randomly from the paragraph at its output; this method thus ignores word ordering at its output. An illustration of these two methods is shown in Figure 2.2. Both these methods are explored and training is done in a semi-supervised fashion, where each document of interest is tagged as either a true or false detection regarding a specific keyword of interest in the news. The tagged true or false keyword is then used as paragraph ID to summarise the context of either occurrence.

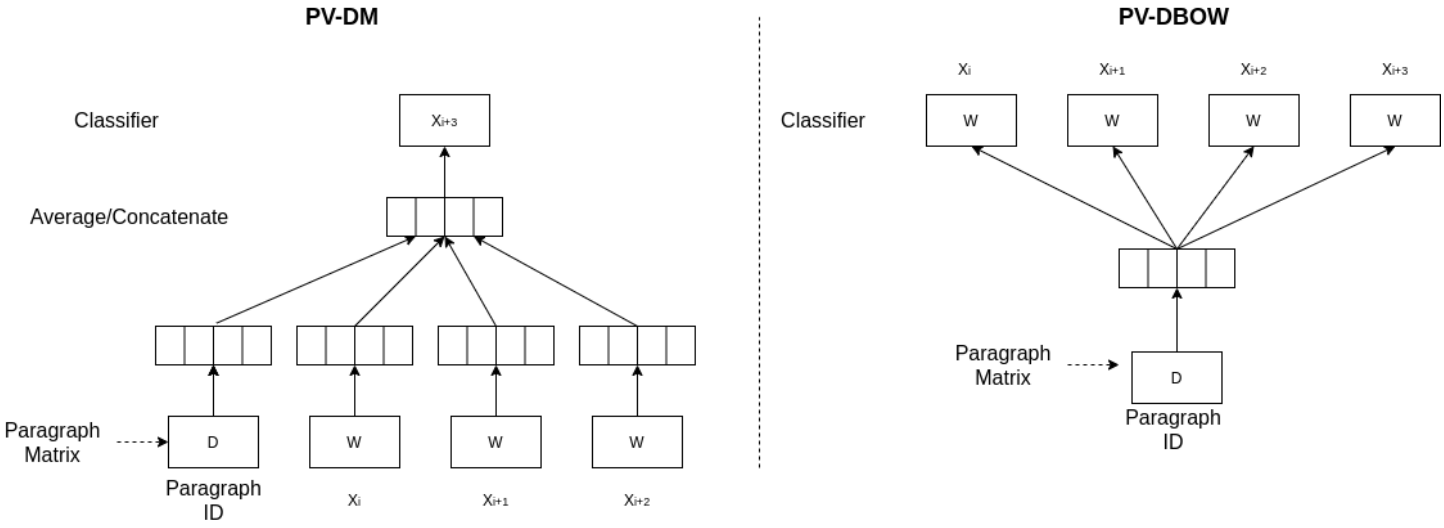


Figure 2.2: Diagram illustrating PV-DBoW and PV-DM.

2.5 Context-aware embeddings

The problems with embeddings such as word2vec, GLoVe and fastText are that they create context-insensitive embeddings. Words such as “apple”, the fruit, and “apple”, the computer company, share the same embedding, despite the word being semantically different in each case. Context-aware embeddings, model the word in question by augmenting it to fit the context in which it appears, thus an entire sentence is used to determine the context of a word. Some of the most important advancements in recent years include embedding from language models (ELMo) [17] and transformer based architectures such as bidirectional encoder representations from transformers (BERT) [18].

2.5.1 ELMo

ELMo, a deep learning architecture used to create deep contextualised word representations is one of the most notable advances in this direction. ELMo uses a bidirectional RNN architecture with multiple layers to create contextualised task-specific embeddings. It specifically uses a mechanism known as attention [19], to weight each hidden

layer before layers are combined to forms a contextual representation of a word. The attention layer forms an important part of determining which word in a sentence carry more meaning concerning the overall context of the sentence. This is done by representing each word at its input as a linear combination of the RNN's internal layer representations (i.e. both hidden states are concatenated), as depicted in Figure 2.3.

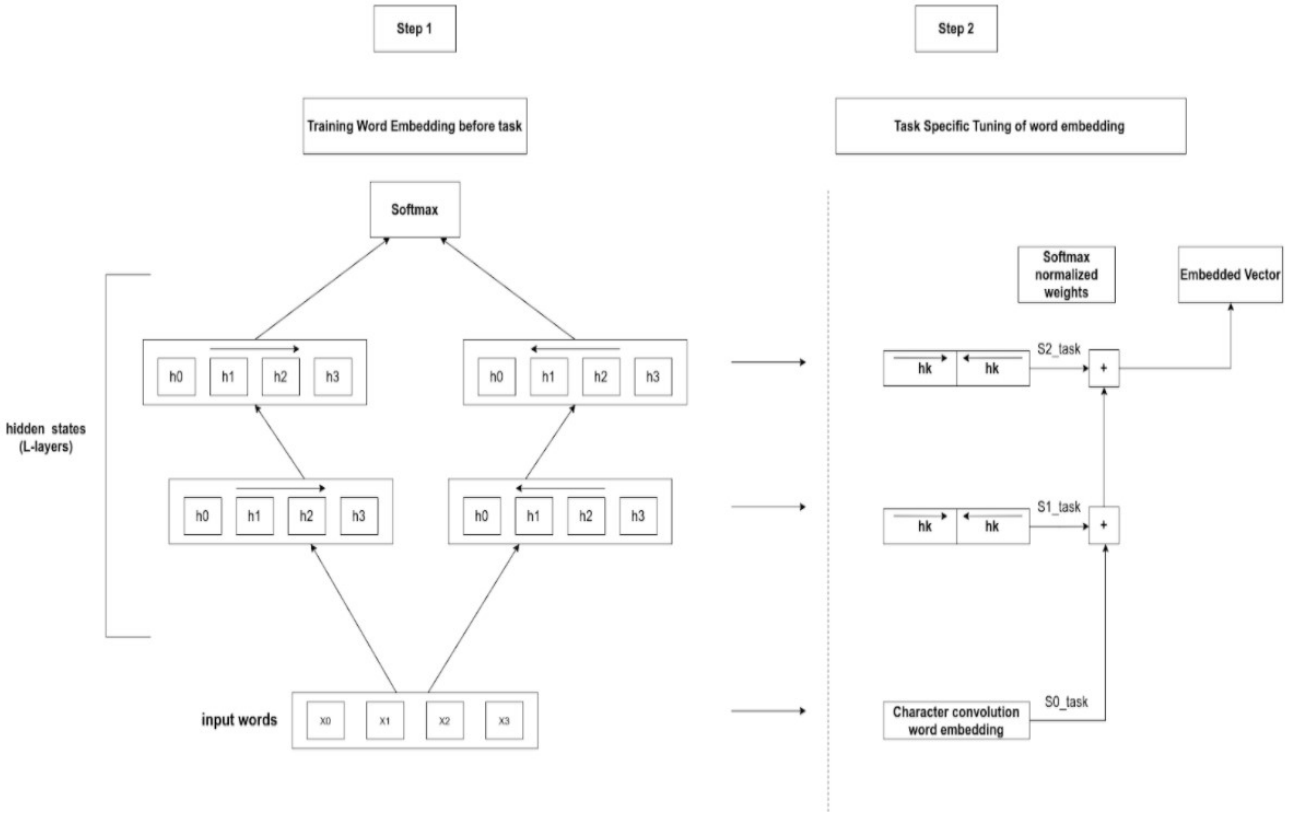


Figure 2.3: Diagram illustrating how task specific embedding can be created using ELMo.

2.5.2 BERT

One of the problems using RNN architectures is that longer sequences (i.e. sentences in our case) can cause problems such as the vanishing gradient [20] problem, where the algorithm essentially stops learning. Another is that due to RNN's inherent sequential nature they struggle to make use of parallelization and take a long time to train. Vaswani [21] builds on this idea of using the attention mechanism and shows

how exploiting transformer-based architecture with an attention heads, a type of feed-forward neural network architecture that can consider an entire sequence at once is able to train faster than previous sequential RNN or CNN based models. OpenAI expands on this using generative pretrained transformer-2 (GPT-2) [22] which uses the same transformer-based architecture but does this unidirectional when doing language modelling tasks. Both ELMo and GPT-2, however, only consider words left to right, where every token (i.e. word) can only attend to the previous token in the self-attention layers. BERT an advancement on this idea on the other hand focuses on both left and right words being said in a sentence, to sum up, its context as a whole in the sentence - it uses a ”masked language model”(MLM). This randomly masks some of the input words to predict the masked token using the context gathered of the rest of the sentence. BERT, however, is usually trained on extremely large corpora of text to get meaningful contextual representations. It also takes extremely long to train from scratch as it consists of 340 million trainable parameters. Usually, for both BERT and ELMo pre-trained models are used and layers added to train embeddings for task-specific purposes.

2.6 Text classification

Text classification is the task of classifying a document into a predefined category. This can be done using either a statistical or machine learning approach. We consider the latter case. The key goal of the machine learning algorithm is to be able to separate documents (i.e. either paragraphs or sentences) based on different contexts. Documents with similar contexts are seen as part of the same set of documents and made distinguishable from documents of different contexts. Examples of tasks such as this include sentiment analysis, document categorisation and language detection. Linear classifiers such as logistic regression are typically used as baseline methods for NLP problems. These classifiers might be very simplistic but can sometimes yield strong baselines for comparing different text classification methods, as shown by Joachims [14] and Fan [23]. Neural networks and support vector machines (SVMs), on the other hand are

non-linear classifiers that can distinguish between more complex patterns present in data. Malhar [24] shows how twitter data can be categorised using a support vector machine (SVM), outperforming other machine learning methods. Manevitz et al. [25] show how they are able to obtain superior results using a simple feedforward neural network to classify documents on the standard Reuters database. Comparing this method to various traditional machine learning techniques such as nearest neighbour, naive-Bayes and one-class SVM's. Typically in automatic speech-recognition systems (ASRs) token identification is used as a means of identifying specific words uttered [26]. Throughout this study, we apply NLP techniques to plain text as a means to recognise text after it has been identified by an ASR system.

2.6.1 Logistic regression classifier

Logistic regression [27] is a linear classification method, used to assign observations to discrete sets of classes. It relies on transforming its output value using a sigmoid or logistic function, returning a probability value given an input. Each input represented as "$\mathbf{x}$" is multiplied by a specific weight "W" and added with a bias "$\mathbf{b}$", as shown using:

$$\mathbf{z} = W^T\mathbf{x} + \mathbf{b}. \tag{2.9}$$

The sum of these values, shown as "$\mathbf{z}$", is then used as input to a sigmoid function, which is used to obtain a probability for a specific input feature between [0,1], as shown by:

$$\mathbf{p} = \frac{1}{1 + e^{-\mathbf{z}}}. \tag{2.10}$$

The weights and biases are learned by measuring the classification error with a loss function and using the back propagation algorithm to fine-tune these values. A standard example of such a loss function is shown in Equation 2.11. This specific loss function is known as binary cross-entropy loss.

$$\log - \text{loss} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log p_i + (1 - y_i) \log (1 - p_i)] \tag{2.11}$$

where: $p_i =$ the predicted probability associated with a specific input sample
$y_i =$ the actual ground truth probability value
$N =$ the number of samples needed to be classified.

2.6.2 Deep neural network classifier

Deep neural networks [28] or multi-layered perceptrons (MLPs), in their most basic form consist of an input layer, a hidden layer and an output layer that are fully connected with one another, as depicted in Figure 2.4. Each perceptron consist of weights and a bias that are passed through a non-linear activation function, typically a sigmoid or rectified linear unit (ReLU). The way in which perceptrons are connected ensures that complex non-linear functions can be approximated in multi-dimensional space. The weights in the network are tweaked in a similar fashion to those of a logistic regression classifier, where the weights and biases are fine-tuned, measuring the classification error using a loss function and then optimised using the backpropagation algorithm.

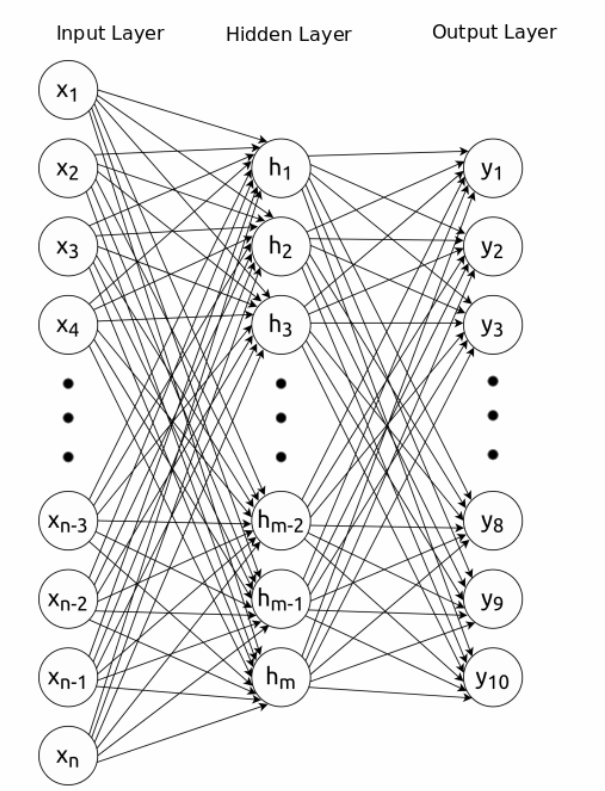


Figure 2.4: Diagram of single hidden layer MLP.

2.6.3 Overfitting

The main criterion for a standard machine learning algorithm is its ability to generalise well over unseen data [28]: we would like the error obtained on the test set, also known as a generalisation error, to be as low as possible. If one assumes that the data the model is trained and tested upon are obtained haphazardly, we are constrained in terms of how well we can close the error gap, also known as generalisation gap, between our training and test sets. If we assume the data are from the same data-generating distribution and samples are independent from one another, we are able to describe our training and test error mathematically.

We try to reduce the generalisation error of our algorithm by limiting the amount the learning algorithm overfits or underfits the available training data. Overfitting occurs when the generalisation gap becomes too large. When this occurs, the model produces

predicted values that have "high variance". This means the model is overly sensitive to small perturbations in the training set, which is due to the algorithm learning random noise in the training data. The second scenario, known as underfitting, occurs when the machine learning algorithm is unable to acquire an adequately low error on the training set. In these circumstances the model is said to have a high bias, meaning the algorithm is missing important correlations between the training features and the target values. This is a direct result of the machine learning algorithm making erroneous assumptions about the underlying data distribution.

By changing the model's capacity we are able to control how likely it is to either overfit or underfit. The model's capacity can be described as the means through which a learning algorithm can fit a wide variety of functions. An over-parametrised model can easily overfit on the training data by memorising it, causing the model to have a higher generalisation error, whereas models with insufficient capacity are unable to represent the training set and underfit on the available data.

One way to limit underfitting is having sufficient capacity within the model to represent the data more accurately. Conversely, having too much capacity increases the model's chances of having high variance and overfitting on the training data. One way to solve both these problems is using a technique known as early stopping. Early stopping can be used to measure the generalisation gap between the training and test error by using a subset of the training data, known as a validation set. When the difference between the training and the validation error is adequately small, it causes preset criteria to stop training and thus avoids overfitting on the data. Another commonly used method, namely dropout, randomly drops a subset of neurons during training in each iteration, resulting in an ensemble of subnetworks that are formed inside the base neural network. This reduces the network's ability to overfit on specific features as it cannot rely on redundant details of specific examples in the training set.

2.6.4 Representing words in a sentence as document vectors

Standard machine learning algorithms require that their input be represented as a fixed length feature vector. Word2vec and fastText, however, only represent the vectors of individual words. Document vectors are therefore created by taking the average of each word vector in a sentence, as shown in Equation 2.12. This method is regularly used to incorporate the semantic meaning of words in a single document vector [29].

$$\frac{1}{n}\sum_{i=0}^{n} w_i \tag{2.12}$$

where: w_i = the word vector of a specific word i in a document
n = the number of words contained in a document.

2.6.5 CatBoost classifier

Byron [30] compared feedforward neural networks and boosted decision trees in a particle physics application and found that boosting was easy to use and was superior when compared to the feedforward neural network. Boosted trees have also recently shown excellent results in various machine learning applications, such as ad-click through rate predictions [31]. In recent years, these techniques have also had many successful implementations on the machine learning competition site Kaggle[3].

Boosting is a method commonly used by machine learning practitioners to reduce both bias and the variance of the machine learning algorithm by combining multiple weak learners (i.e. decision trees) to make one strong classifier. Each subsequent tree is trained on the mistakes made by the previous tree in an attempt to reduce the objective function. The algorithm weights each individual decision tree's prediction, which is based on the classifier's ability to classify examples. If a weak classifier incorrectly classifies instances, less weight is added to the individual classifier's decision. It is

[3]https://www.kaggle.com/competitions

typically used when one has a problem consisting of a diverse set of features. CatBoost [32] is a gradient-boosting algorithm developed by Yandex[4], which is advantageous, as it is easy to use and does not require the categorical data to be preprocessed to numerical values like many other gradient-boosting decision tree algorithms such as XGBoost and LightGBM. It generally outperforms many of these algorithms, as it uses symmetrical trees and a different strategy for calculating leaf values when selecting the tree structure. This is claimed to help with overfitting.

2.7 Conclusion

In this chapter, various embedding techniques are discussed, including how these embeddings are used and trained. The different ways in which word embeddings are evaluated are discussed, as well as how they are applied to different types of classifiers and how the various classifiers work.

[4]https://yandex.com/

Chapter 3

Data Analysis and Exploration

In this chapter we investigate the various preprocessing techniques used and also analyse the data used throughout the study.

3.1 Introduction

This chapter focuses on the preprocessing techniques used throughout the study, giving some useful insight into the available data used to train the various machine learning classifiers. Recognised speech data from an ASR system is investigated, specifically focusing our study on recognised speech data obtained from local broadcasters.

3.2 The speech recognition text data set

The speech-text data were acquired from local South African broadcasting stations and are an amalgamation of radio news and television channel data. That was obtained from the output of a single ASR system able to recognise all 12 South-African lan-

guages. The data obtained from the system were manually labelled to indicate FPs by people working for the Novus Group.

The text in the dataset is predominantly English; there are almost 10 times more English sentences in comparison with the second most frequently occurring language in the dataset, Afrikaans, as shown in Figure 3.1. We view our corpus as an English corpus, with code-switched intrusions from other languages. The languages recognised by the ASR system are specific to the South-African context where code-switching occurs. There are 887 431 speech-text examples of which many contain more than one keyword (i.e. specific words of interest to the Novus Group). Of these examples 676 129 TPs and 211 302 are TNs (FPs from the speech-text classifier). This means that 76.19% of the data consist of TPs and 23.81% of TNs.

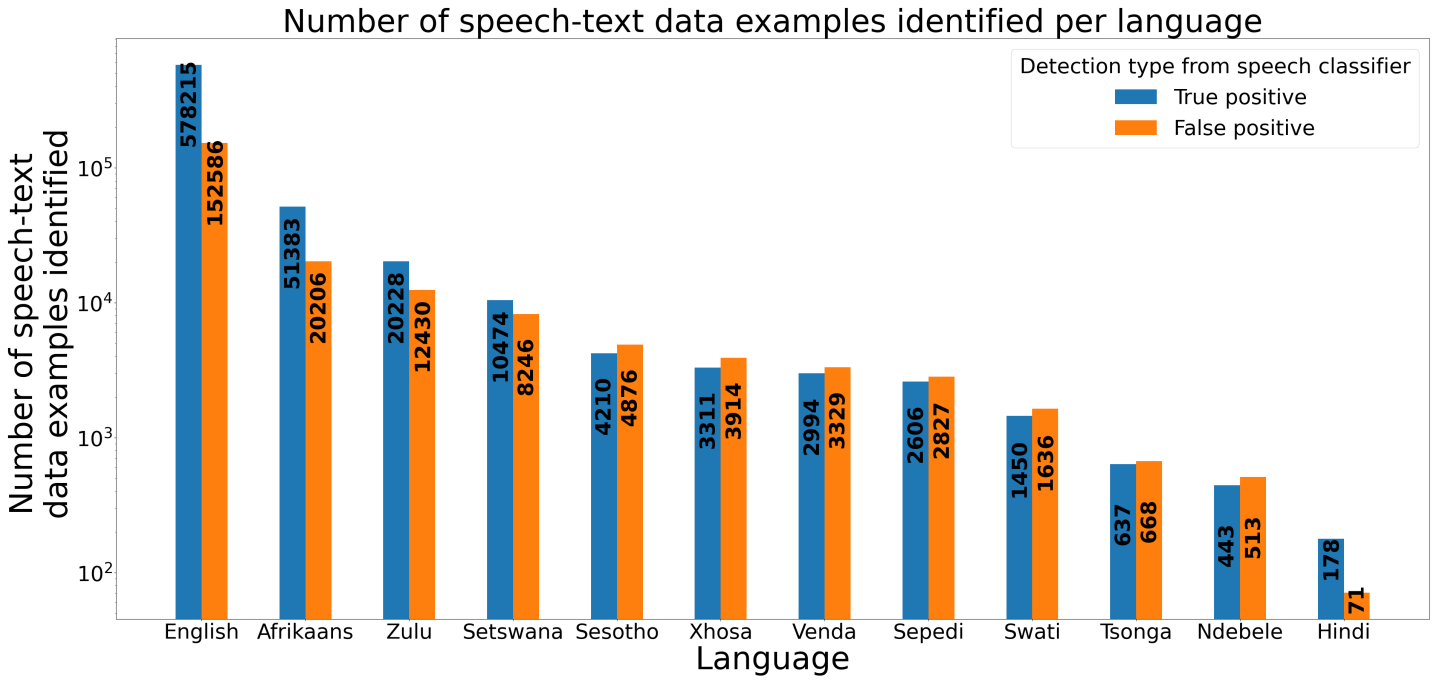


Figure 3.1: Frequency of occurrence for different languages in the corpus. Note, the legend title "Detection type from speech classifier" refers to the data from the ASR system that was labeled by people.

There are 106 broadcasting stations in the corpus. The most frequent of these, as shown

in Figure 3.2, is Ignition, a television channel about cars on DStv, followed by Business-DayTV, a television channel dedicated to business news. The rest of the broadcasters are mostly radio stations and business news television channels.

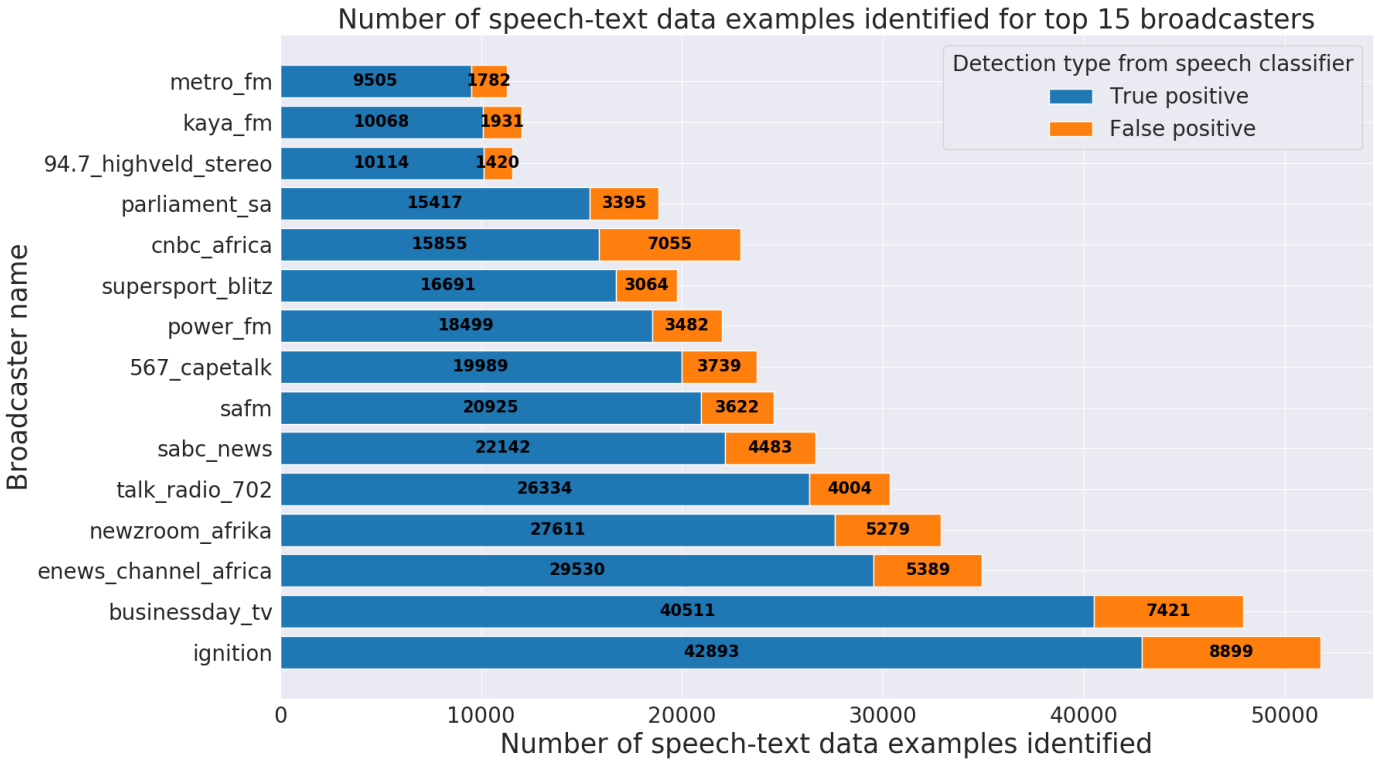


Figure 3.2: Top 15 broadcast stations found in the corpus, out of 106 different broadcast stations (i.e. obtained for all 12 languages).

Speech-text examples obtained from the ASR system and labeled by the people working at the Novus Group are shown in Table 3.1. Unrelated from these examples other interesting keywords obtained from different languages include: aartappels, snotkop (i.e. Afrikaans examples), mzansi_super_league, zweli_mkhize (i.e. Zulu examples), dj_maphorisa and shekhinah (i.e. Xhosa examples).

Table 3.1: Window size around different keywords example.

Keyword of interest	**Speech-text example**	**Type of example**
suid afrikaanse lugdiens	ingeskakel by pretoria fm tyd nou vir die nuus hooftrekke vir die nuushooftrekke die **suid afrikaanse lugdiens** het vroeer vandag bevestig dat hy sy vyf duisend een honderd vier en sestig werknemers in kennis gestel het van n herstruktureringsproses wat tot werksverliese	True Positive
suid afrikaanse lugdiens	persone vermis of dood is nie video's van die tornado het gister soos 'n veldbrand op sosiale media versprei die **suid afrikaanse lugdiens** waarsku dat n staking by sal al sy operasionele bedrywighede tot stilstand kan dwing dit kom nadat vakbonde gedreig	False Positive
disney	story of all time when i sars about aston i go about the surplus being shana screening on smart these weaker cartier catching right to a **disney** there is no there is no cuts knows there's no cuts and there is no sweets	False Positive
disney	trying to do is to come out with a bang ride and remember there are also often **disney** plus as part of the bundle with woolworths yes bn for twelve ninety_nine so that is even more come compelling proposition neymar come compelling proposition when you think about all the different kinds of can not and available that blacks yesterday raised prices try netflix is twelve ninety_nine which is almost double what	True Positive

3.3 Preprocessing

Before words are encoded to be used as input features to a machine learning algorithm, it is typical to clean the available text first. This is done so that meaningful data can

be extracted from the corpus. Specifically, we focus on using lower-casing, symbol removal, stop word removal, lemmatisation and phrase detection [33].

3.3.1 Lower-casing text, symbol and stop word removal

A standard form of text preprocessing is lower-casing words in text. This ensures uniformity in the provided corpus and consequently that stronger word vectors can be learned for different variations of the same word (e.g "fox", "Fox" and "FOx"), instead of training a different vector for each variation. Unnecessary symbols are also removed (e.g. "_", "*", "+","(" and ")"), as they have no real use except contributing to the syntax of a sentence. Stop word removal involves removing words from the corpus that do not contribute to the overall meaning conveyed by sentences in a corpus. Typical examples of English stop words are 'a', 'the', 'is' and 'are'.

3.3.2 Lemmatisation

In many NLP problems it is beneficial to convert certain words to their original base form or 'lemma'. Considering words such as 'kick' and 'kicked', it is useful to map these words to the same root word 'kick', as words such as these are just different forms of the same verb and carry the same meaning. This process is known as text normalisation. Examples of such words include "geese" becoming "goose" and "caring" becoming "care". This ensures consistency in the corpus, meaning better feature representations can be made with the available text. This was done using the python spaCy library [34], typically used for advanced NLP research.

3.3.3 Phrase detection

Phrases have a specific meaning that is not always conveyed by the composition of the individual words of which the phrase is made up. Phrases can be learned by identi-

fying words that are frequently found together in a specific context and infrequently found in other contexts. A data-driven approach [15], as shown by Equation 3.1, can be used to find phrases in a corpus of text where bigrams, with a score (w_i, w_j) higher than a specific threshold, are used to create a phrase from those words.

$$\text{score}\left(w_i, w_j\right) = \frac{\text{count}\left(w_i w_j\right) - \delta}{\text{count}\left(w_i\right) \times \text{count}\left(w_j\right)} \tag{3.1}$$

where: $\text{count}(w_i w_j)$ = bigram count of words in the corpus
$\text{count}\left(w_j\right)$ = unigram count of words in the corpus
δ = discounting coefficient used to prevent too many phrases consisting of infrequent words to be generated.

Identifying common phrases in text, such as "New York" or "Net income", without having a numerical representation for each word separately, would effectively increase the vocabulary size of the available words that can be learnt, leading to a richer vocabulary of embedded representations of words used by the machine learning algorithm to distinguish more clearly between concepts.

3.4 Preprocessing pipeline

Before the text is converted to a numerical representation for the various machine learning algorithms, preprocessing is done on the corpus, shown in Figure 3.3.
The preprocessing steps are listed as follows:

1. Extract speech-text data from JSON files — more information can be found in Appendix B.1.
2. Clear the data of any unnecessary symbols and change all the words in the corpus to lower-case.
3. Remove English stop words.

4. Lemmatise the words in text, converting all English words to their base form.

5. Identify common phrases in text.

6. Extract data with different window sizes around individual keywords.

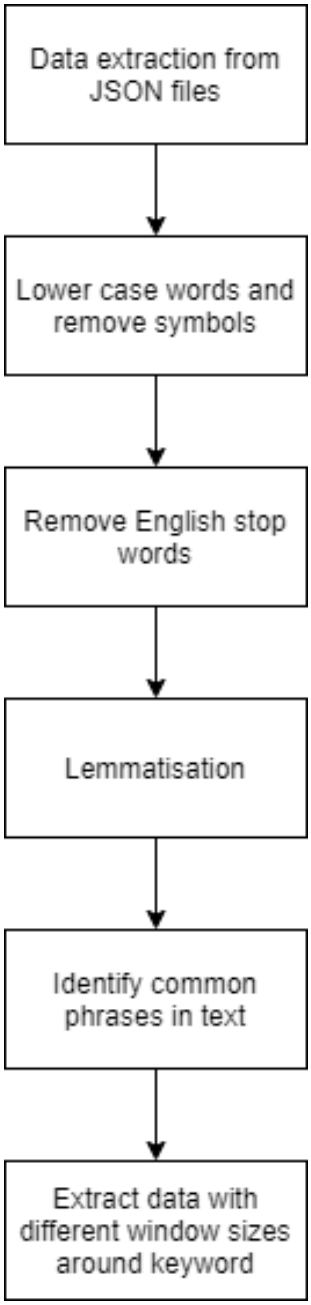


Figure 3.3: Flow diagram detailing data extraction and preprocessing pipeline.

3.5 General corpus information

The most frequent words and word pairs found in the corpus, shown in Figure 3.4 are "eskom", "nedbank", state entities, private companies, the names of important people and many different car brands, which is to be expected, as Ignition and BusinessDayTV are among the most frequent broadcasting channels.

The raw corpus consists of 248 069 944 words with a vocabulary size of 69 052. After preprocessing, the word count is significantly reduced to 104 565 491 and the vocabulary size increased to 237 548. It is important to note, phrase detection to be the main reason for the increase in vocabulary size. There are 4 186 unique keywords, of which 2 729 keywords have examples of where they appear in text as both true and false detection examples. Of the examples, 1457 were labeled to belong exclusively to either true or false.

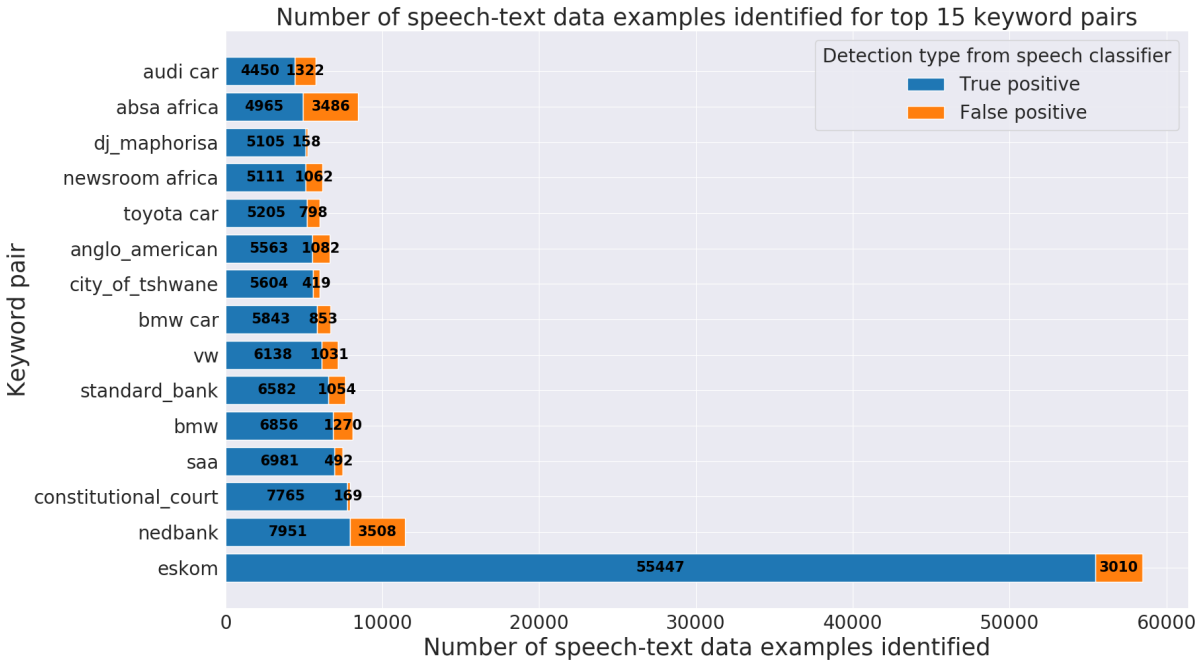


Figure 3.4: Top 15 words and word pairs found in the corpus.

Keyword pairs are broken into individual keywords with different word window sizes around specific keywords of interest (window size refers to the words to the left and

right of the keyword). An example of this can be seen in Table 3.2 for windows size 5 and 11 respectively.

Table 3.2: Window size around different keywords example.

Window size	**Sentence with keyword in the middle**
window 5	quick brown **fox** jumped over
window 11	the hungry vigilant quick brown **fox** jumped over the lazy dog

A new corpus is thus created where each keyword has its own pseudo-sentence. This is done to investigate how many context words around specific keywords are needed to achieve good classification. Information on this newly generated corpus can be seen in Table 3.3. It is clear from this table that as the window size increases, so does the vocabulary and word count, which is to be expected.

Table 3.3: Corpus information, showing window sizes, word count and vocabulary size.

Window size around keyword	**Number of words in corpus**	**Vocabulary size**
window 5	7 562 980	**105 685**
window 11	15 521 514	**150 951**
window 21	27 244 705	**184 033**
window 41	46 840 613	**208 273**
window 61	63 174 235	**217 431**
window 121	99 466 572	**226 408**

After the data was captured, the new corpus (i.e. the clean pseudo sentence corpus) is split into a training, validation and test set for each window size, as shown in Table 3.4. Note that the number of examples decreases as the window size increases, since keywords occurring near one another (i.e. within the bounds of the specific window size) are combined to form a single example. For example, for a window size of 11: “the hungry vigilant quick brown **fox** jumped over the lazy dog” and ”the hungry vigilant

quick brown **fox** jumped over **fox** the lazy dog" are treated as single pseudo sentences for both cases. This is done to prevent examples with the same basic context, being split into separate pseudo sentences, to prevent information leaking into the validation or test set when the data is partitioned.

Table 3.4: Corpus information, showing how data is split into training, validation and test set for each window size.

Window size around keyword	**Training set**		**Validation set**		**Test set**	
	Word count	Number of examples	Word count	Number of examples	Word count	Number of examples
5	6 067 935	1 262 568	739 063	153 763	755 982	157 304
11	12 449 657	1 200 563	1 520 160	146 706	1 551 697	149 648
21	21 849 359	1 141 341	2 672 410	139 619	2 722 936	142 322
41	37 572 891	1 064 076	4 585 077	130 023	4 682 645	132 665
61	50 676 774	1 016 754	6 182 061	124 128	6 315 400	126 718
121	79 805 073	945 361	9 723 801	115 197	9 937 698	117 759

Figure 3.5 shows the most frequent keywords found in a corpus with a window size of 121. Splitting the data in this way resulted in some common nouns appearing more frequently, due to the way the data was captured. These common nouns do not contribute much to the identification of specific words and are only a result of how the data was split.

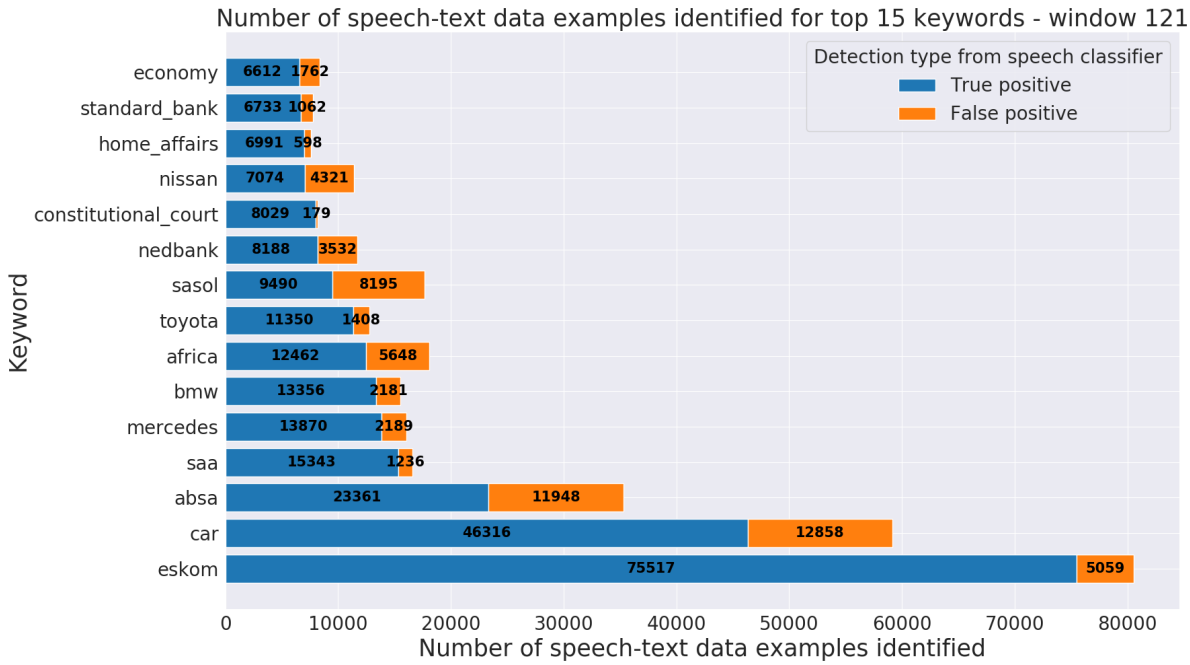


Figure 3.5: Top 15 words found in the corpus, with window size of 121.

3.6 Conclusion

In this chapter, we describe the various preprocessing methods used throughout the study and why these are useful. We analyse the available recognised speech data and introduce a way of creating pseudo-sentences around specific keywords of interest, which helps investigate the number of context words necessary to obtain desirable classification results.

Chapter 4

Experimental Procedure

In this chapter, we discuss the experimental approach that was used for data selection and classifier training.

4.1 Introduction

The current chapter highlights some of the key design choices and evaluation metrics used throughout the study. We elaborate on the different classification methods used and how those classifiers are compared with one another and evaluated. We consider a logistic regression classifier as our baseline text classification method and compare its performance to using the average cosine similarity method elaborated on later in Section 4.4.3 and a DNN classifier with varying widths. In contrast with these methods, which are used to classify text, we also introduce a CatBoost classifier to train on the meta-data associated with the text-data.

We propose a strategy for dealing with the unbalanced data, and a technique to di-

vide the data into a training, test and validation set, relative to the number of times specific keywords appear in the corpus. Moreover, we discuss the particular choice of hyperparameters used to train word and document vectors.

With regard to classification, we also introduce a method that tags keywords with flags, explaining how text classification can be performed using only the word vectors and these keywords. We then modify this approach to work for DNN classifiers as well. Further, we explain ways in which word embeddings can be visualised in two-dimensional space and the training methodology used to ensure the models are sufficiently trained.

4.2 Training, validation and test split

The cleaned corpus data is split into a training, validation and test set by randomly splitting the data in accordance with how frequently a specific keyword appears in the corpus — 10% of the data is used for the test and validation set respectively, thus 80% of the data remains for training. Therefore, if "sasol" detected as true appeared 20 times and "sasol" detected as false appeared 10 times in the corpus, a total of four and two randomly selected sentences are taken from the "sasol" true and false categories respectively to form the validation and test set. We report the accuracy achieved on the test set and do not attempt to perform statistical significance testing since our goal is to give an indication of the relative performance for various choices of algorithms and parameters, rather than a definite ranking.

The data obtained from the clean corpora (i.e. the six corpora with different window sizes) used the same validation and test data. Therefore, a corpus with a window size of 5 around keywords would have the same train, validation and test set as a corpus with a window size of 121. The only difference between the train, validation and test set would be the number of words surrounding keywords. The same sentence example is used in each case.

4.3 Representing text with word and document vectors

This section investigates the various hyperparameters that were used to create the document and word embeddings. Both word and document embeddings are trained using the gensim API [35]. Tuning the hyperparameters more systematically might lead to slightly better results, but based on our empirical exploration we do not expect such improvements to be large.

4.3.1 Word2vec and fastText

In preliminary investigations, the following hyperparameters for fastText and word2vec, shown in Table 4.1, exhibited good overall performance in tasks with extrinsic evaluation. These values were empirically selected and correspond with some of the practical values suggested by Miklolov [15]. The window size was changed from 5 to 10, as it showed slightly better results and minimum word count was set to 2. This is done to omit words being removed from South African languages with fewer detection examples.

It is important to note that the window size referred to in the context of training word and document embeddings refers to the hyperparameter used to train these embeddings. This does not refer to the window size of the new corpora created using the pseudo sentences discussed in Section 3.5. Another important note: the embedding for each corpus (referring to the 6 corpora discussed in Section 3.5) is trained using the training data for words available in that corpus. This means that six different word2vec word embedding models were trained. This is done for all 12 languages in each corpus, thus a single embedding model is trained for a specific corpus for all 12 languages pooled together.

It was decided not to use pretrained embeddings owing to the size of the available corpus and the domain-specific nature of the data, being speech-text data for several languages. Pretrained embeddings are also not available for all South African languages.

For some languages such as Afrikaans, pretrained embeddings can be found on the fastText website[1]. Studies such as that of Altszyler [36] suggest that good word embeddings can be obtained by training on relatively large amounts of domain-specific data. Their study showed that training with a medium-size dataset of ($\pm$ 10 millions of words) achieved good results.

Table 4.1: Word2vec and fastText hyperparameters.

Parameter	**Value**
Embedding dimension	300
Window size	10
Minimum word count	2
Negative sampling rate	5
Sub-sampling rate	1e-5

4.3.2 Doc2vec

Hyperparameters suggested by Lau [37] are used. The minimum word count set to 2 for the same reasons as the word embeddings to obtain the values shown in Table 4.2. The document vectors are trained using a smaller embedding dimension than the word vectors, as no significant performance differences were measured when increasing its size beyond 100 dimensions.

[1]https://fasttext.cc/docs/en/crawl-vectors.html

Table 4.2: Doc2vec hyperparameters.

Parameter	Value
Embedding dimension	100
Window size	15
Minimum word count	2
Negative sampling rate	5
Sub-sampling rate	1e-5

4.3.3 TF-IDF

The only hyperparameter that was tuned to create the TF-IDF document vectors is the dictionary size. Dictionary sizes of 60 000, 80 000 and 100 000 (i.e. for all 12 languages) are considered in this investigation, choosing the dictionary size based on how well TF-IDF was able to classify text using the logistic regression classifier (i.e. extrinsic evaluation).

4.4 Classification

A Flow diagram illustrating the experimental procedure is shown in Figure 4.1. All the approaches and configurations we considered are listed below:

1. TF-IDF with LR.
2. Word2vec with LR.
3. fastText with LR.
4. Doc2vec with LR.
5. Word2vec with average cosine similarity method.

6. fastText with average cosine similarity method.
7. TF-IDF with DNN.
8. Word2vec with DNN.
9. fastText with DNN.
10. Doc2vec with DNN.
11. Word2vec with FFNN with true and false flags.
12. fastText with FFNN with true and false flags.
13. CatBoost using 5 features.
14. CatBoost using 5 features and TF-IDF with DNN.
15. CatBoost using 5 features and Word2vec with DNN.
16. CatBoost using 5 features and fastText with DNN.

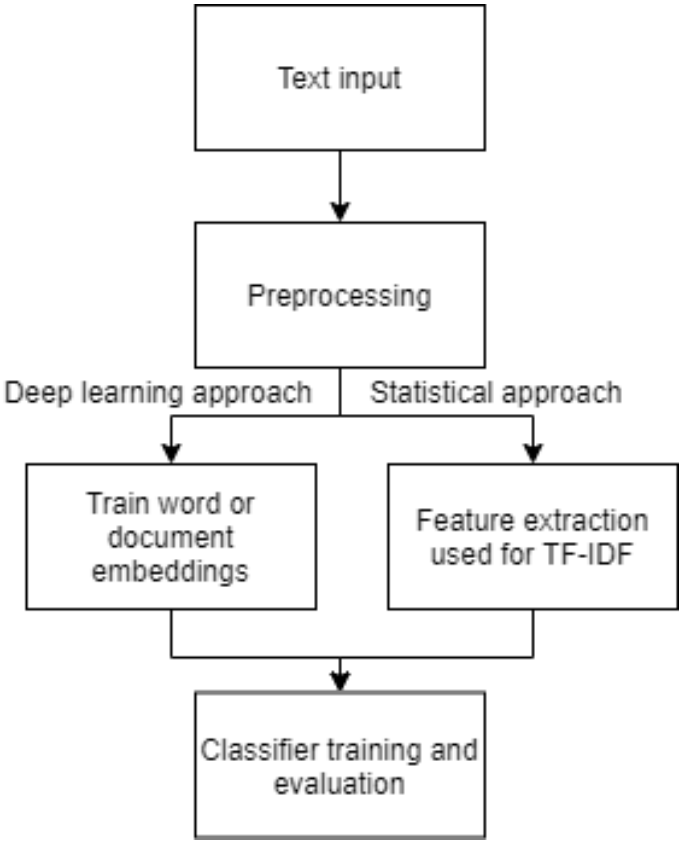


Figure 4.1: General process flow diagram of classification and evaluation.

4.4.1 Logistic regression and DNNs using textual data

Doc2vec, word2vec, fastText and TF-IDF are investigated using a logistic regression (linear classifier) and a one layer feedforward neural network classifier (non-linear classifier) with varying widths of 100, 200 and 400 hidden nodes. This is done to see what effect capacity has on the classification performance of the DNN. In addition, class-weighted logistic regressors and feedforward neural networks are used, to mitigate the effect of having an unbalanced dataset. Each class is weighted in inverse proportion to its frequency of occurrence:

$$\text{class-weight} = \frac{\text{total number of samples}}{\text{number classes} \times \text{number of samples of specific class}}. \tag{4.1}$$

This weight value is multiplied by each classifier's loss function, yielding:

$$\log - \text{loss} = -\frac{1}{N}\sum_{i=1}^{N} \text{class-weight} \times \left[y_i \log p_i + (1 - y_i) \log (1 - p_i)\right]. \quad (4.2)$$

This has the effect that larger weight updates occur for less frequently seen classes in the dataset than for frequent ones.

4.4.2 CatBoost classifier using textual and non-textual data

A CatBoost classifier is trained using the corresponding metadata associated with the textual features. Features including the target word, broadcast station name, the total duration of the recorded clip in minutes, station language and day of the week are used to train the CatBoost classifier. This is used as a baseline method and to see whether adding additional features, such as the output of the DNN trained on text data, results in better classification.

4.4.3 Average cosine similarity method

Inspired by word2vec's distributional hypothesis, the average cosine similarity method uses the trained word vectors for classification. The premise is that keywords that are identified as FPs will be surrounded by similarly incorrect words that have nothing to do with the actual keyword. This essentially means that occurrences of words such as "Disney_t" (labelled with a flag "t" for true detection) will be surrounded by completely different words than those of "Disney_f" (labelled with a flag "f" for false detection), as illustrated in Figure 4.2. During inference one would identify the keyword, using a dictionary of all the keywords of interest, and compare the cosine similarity, shown by Equation 4.3, for each word in the sentence for both its true and false keyword examples by adding this flag. The average cosine similarity for both these cases could then be compared with each other, where if the average similarity for "Disney_t" is larger than "Disney_f", the detection is most likely a true detection, and vice versa.

The output to this decision is thus binary (therefore a binary classifier). The investigation is done for different window sizes, comparing both fastText and word2vec, and investigating both SG and CBoW.

$$\cos\theta = \frac{a \cdot b}{\|a\|\|b\|} \tag{4.3}$$

where: $\cos\theta$ = cosine similarity between the keyword a and arbitrary word b
a = the keyword of interest for either its true or false examples
b = an arbitrary word in the sentence.

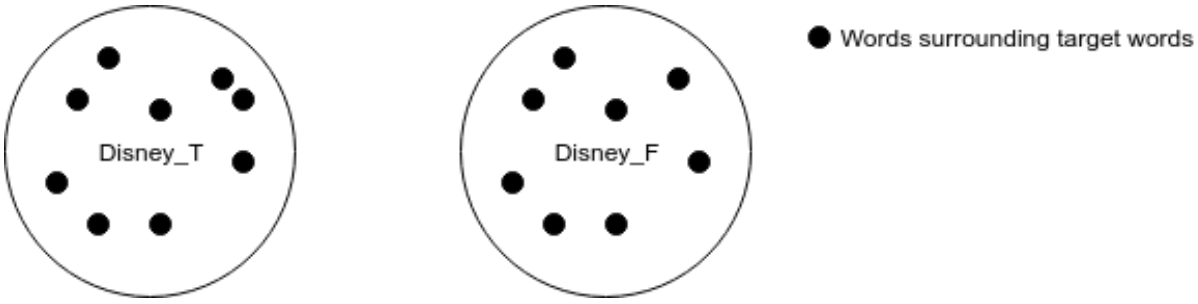


Figure 4.2: True detection word embedding cluster for the word Disney (shown left) and FP word embedding cluster for the word Disney (shown right).

4.4.4 Flag method with DNNs

The method described in Section 4.4.3 is extended so that it can be implemented with a feedforward neural network. The newly proposed method aims to take the average sentence vector comparing both detected examples, as shown in Figure 4.3. This augments the data, increasing its size, forcing the model to focus more on the context words around the keywords of interest. We hypothesise that this may enhance the performance of the network since the network is forced to learn more about the context around a specific keyword to classify text correctly.

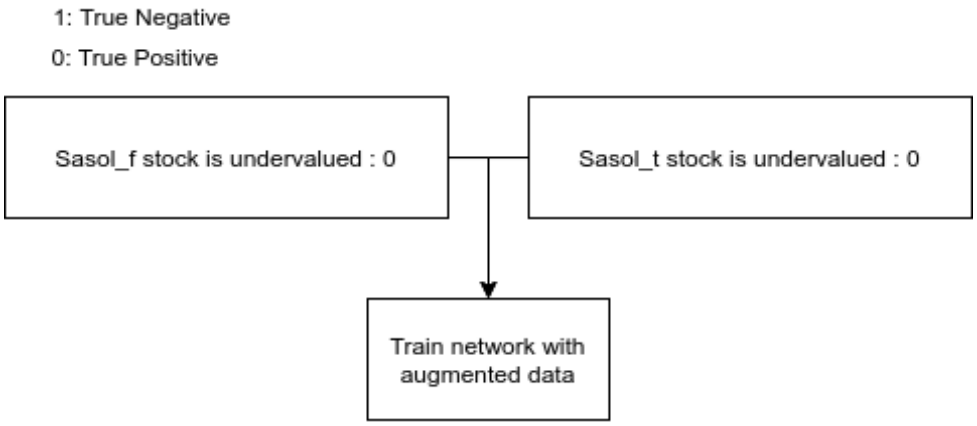


Figure 4.3: Diagram illustrating how data is augmented.

During inference, one would successively augment the keyword detected by adding a "_t" (i.e. indicating true) and "_f" (i.e. indicating false) flag to the word, as shown in Figure 4.4. After this, we take the average sentence vector for both cases (i.e. true and false keyword) and inspect which example convinces the model the most. The probability of which example classifier is more certain can be determined, using Equation 4.4, since the events are independent from each other. One would compare the probability of each event being either false or true, denoted as $P(\text{False})$ and $P(\text{True})$ respectively in each case. If the probability of $P(\text{True})$ is higher than $P(\text{False})$ it is most likely a true detection.

$$P(\text{False}) = P_1(\text{False}) \times P_2(\text{False}) \qquad \text{and} \qquad P(\text{True}) = P_1(\text{True}) \times P_2(\text{True}). \quad (4.4)$$

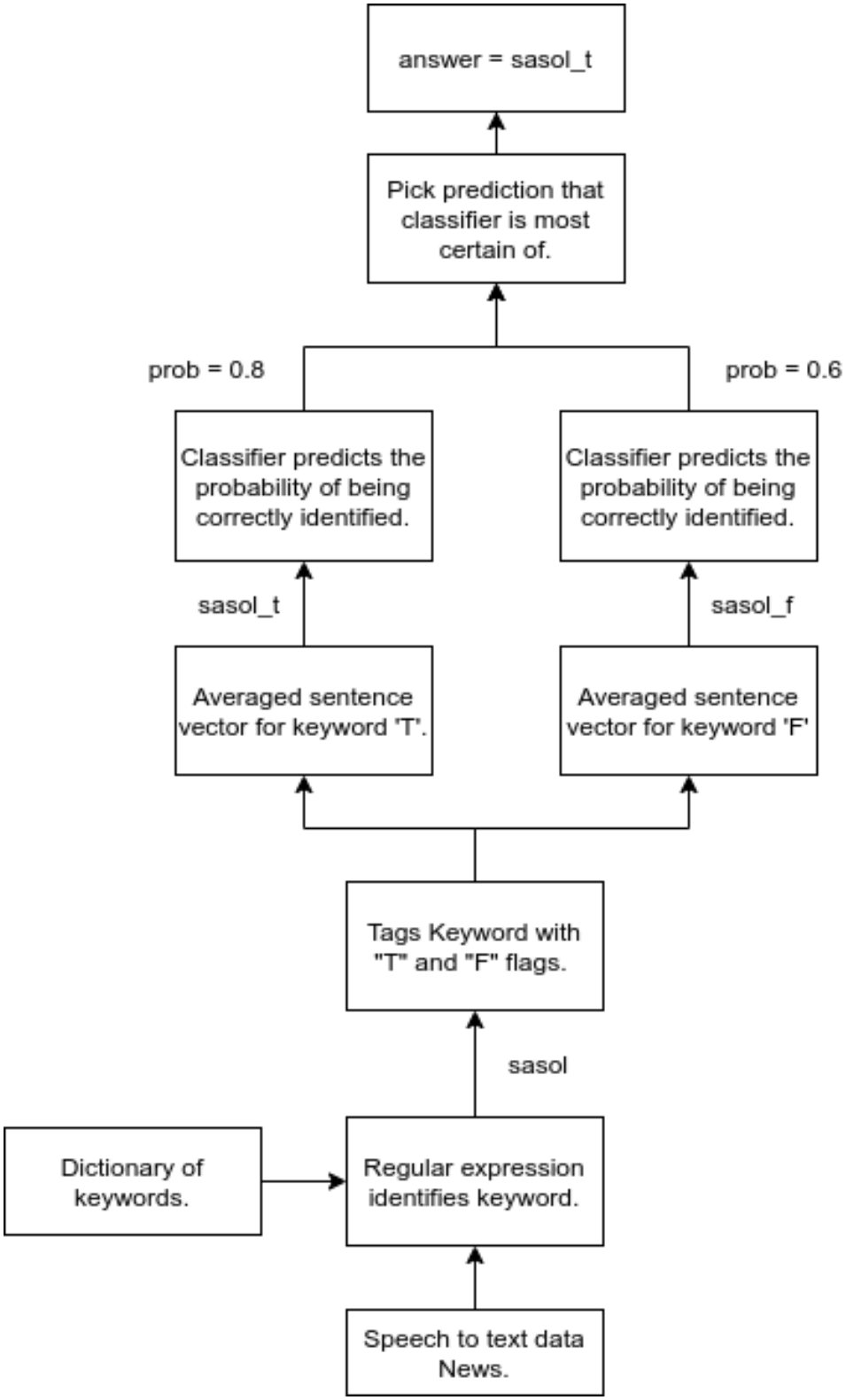


Figure 4.4: Diagram illustrating how inference would work for flag method with average sentence embedding.

4.5 Evaluation metrics

A receiver operating characteristic (ROC) [38] curve is used alongside its corresponding area under the curve (AUC) values to help determine the various machine learning algorithms' effectiveness. A curve is created by plotting true positive rate (TPR) against false positive rate (FPR) at various probability thresholds. From this curve the AUC value is determined. This value helps indicate how well the classifier is at differentiating between different classes and can be seen as a measure of the degree of separability between classes. The optimum threshold used for evaluation is determined using the Youden statistic [39]. This statistic is calculated by subtracting the classifier's TPR from the true negative rate (TNR) to denote the threshold value where these values are at a minimum.

In various classification problems, it is standard to use a confusion matrix to express the classification performance of a machine learning algorithm. For a two-class classification problem, a confusion matrix table can be constructed, as shown by Table 4.3. The table consists of four terms namely, True Positive (TP), False Positive (FP), False Negative (FN) and True Negative (TN) detection examples. The positive class coincides with speech-text examples that were correctly detected by the speech-recognition system, whereas the negative class indicates FP detection by this system. These values are used as our classifier TN examples and are indicated as such.

Table 4.3: Confusion matrix table.

		True class	
		Positive	Negative
Predicted class	Positive	True Positive (TP)	False Positive (FP)
	Negative	False Negative (FN)	True Negative (TN)

where: TP = data labelled as correct (i.e. positive class) that are predicted as correct
TN = data labelled as incorrect (i.e. negative class) that are predicted as incorrect
FN = data labelled as correct that are predicted as incorrect
FP = data labelled as incorrect that are predicted as correct.

4.5.1 Cohen kappa metric

The various classifiers are evaluated based on the Cohen kappa [40] statistic. It is used to compensate for the effect of the class imbalance present in the data, and measures the inter-rater agreement between categorical items.

The observed agreement, depicted by p_o, is identical to the accuracy of the model, calculated using:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.5}$$

$$p_o = \text{Accuracy}. \tag{4.6}$$

The expected agreement (i.e. the probability of the agreement between the models predicted and actual class values as if happening by chance), depicted as p_e, is then calculated using:

$$p_{\text{true}} = \frac{TP + FP}{TP + TN + FP + FN} \cdot \frac{TP + FN}{TP + TN + FP + FN} \tag{4.7}$$

$$p_{\text{false}} = \frac{FN + TN}{TP + TN + FP + FN} \cdot \frac{FN + FP}{TP + TN + FP + FN} \tag{4.8}$$

$$p_e = p_{\text{true}} + p_{\text{false}}. \tag{4.9}$$

Using the calculated p_e and p_o value, the Cohen kappa statistic can then be obtained

using:

$$\kappa = \frac{p_o - p_e}{1 - p_e} = 1 - \frac{1 - p_o}{1 - p_e}. \tag{4.10}$$

The calculated statistic can then be interpreted using Table 4.4, where a κ lower than 0 indicates a less than chance agreement and a kappa equal to 1 indicates perfect agreement amongst raters.

Table 4.4: Cohen kappa metric.

Cohen kappa	Agreement
<0.00	Less than chance agreement
0.01-0.20	Slight agreement
0.21-0.40	Fair agreement
0.41-0.60	Moderate agreement
0.61-0.80	Substantial agreement
0.81-0.99	Almost perfect agreement

4.5.2 Shapley additive explanations values

Assessing why machine learning models make certain predictions can be a complex task. To remedy this, SHapley Additive exPlanations (SHAP) values [41] can be used to assign each feature an importance value based on a specific prediction. To evaluate feature importance in the CatBoost classifier, mean absolute SHAP values are used. The SHAP values are calculated by comparing what the model predicts with and without a specific categorical feature, and this is then used to determine which features are relevant and which are not.

4.5.3 Visualising word embeddings

It is often useful to visualise high-dimensional word embeddings in a two or three-dimensional map. For this investigation a technique known as t-distributed stochastic neighbourhood embeddings (t-SNE) [42] is used as a dimensionality reduction technique to view the relationships of different words in two-dimensional space and reveal global structures (clusters of words and topics) within the data. Older techniques such as principal component analysis (PCA) [43] are linear methods that keep low-dimensional representations of contrasting data-points distant from one another. Stochastic non-linear techniques such as t-SNE, preserve the high-dimensional local structure of the data. This ensures that points that are nearer to one another in higher dimensional space are still near one another in a low-dimensional representation. This is typically unachievable using linear techniques.

4.6 Deep neural network and CatBoost classifier optimisation details

Both the CatBoost and DNN classifier use early stopping based on the AUC value to counter overfitting. The model stops training if there is no further improvement greater than 0.1% on the validation set AUC value after waiting for 10 epochs. The DNN uses the adaptive moment estimation (Adam) [44] optimiser with a batch size of 1 024 and a learning rate of 0.001, including a ReLU function before the softmax layer. The logistic regression classifier used default values and a liblinear solver; the CatBoost classifier used the default values and a learning rate of 0.1. Better performance might be achieved through a systematic hyperparameter tuning. This is an exploratory study, and such tuning was not performed since our informal explorations indicated that only limited benefits could be achieved in this way.

4.7 Conclusion

In this chapter, we paid particular attention to partitioning the corpus into a training, validation and test sets which ensure that the underlying data distributions between the sets remained similar. We consider different methods of doing classification using the textual and non-textual features (i.e. categorical features associated with the text data) present in the data, also introducing a way of dealing with the class imbalance, using weighted cross-entropy. Considering various hyperparameters we motivate our specific choices for training word and document embedding. Finally, we explain how the machine learning models are evaluated and trained.

Chapter 5

Experimental Results

In this chapter, we discuss the results obtained when identifying false positive keywords in the text using different embedding methods and classifiers, using both textual and non-textual features associated with the ASR data.

5.1 Introduction

In this chapter, we show the results obtained using the experimental procedure discussed in Chapter 4. We compare the different classification methods discussed using fastText, word2vec, doc2vec and TF-IDF as embedding methods. Additionally we also show the results obtained using the available text data and the flag method, discussed in Section 4.4.3 and 4.4.4.

It should be noted that initial investigations found that word2vec and fastText models trained using the SG method performed better than models trained using the CBoW method. Similarly, doc2vec methods trained using the PV-DBoW method performed significantly better compared to PV-DM. The results of these methods are therefore not

presented in the following Tables. They are, however, present for the average cosine-similarity method. Note that test set results are not present in Section 5.2 and Section 5.3, given the weaker performance of these classifiers compared to other methods and the small difference noticed between validation and test set performance.

5.2 Logistic regression classifier

This section investigates the most meaningful results obtained using different window sizes and embedding methods with a logistic regression classifier, used as a baseline method throughout the study. From Table 5.1, it is clear that TF-IDF outperforms all other methods. Comparing different dictionary sizes, a dictionary size of 100 000 proved to give the best overall results. As shown in Appendix C.1.1, the difference in performance between dictionary sizes is, however, marginal. During subsequent investigations, we only consider a dictionary size of 100 000. Using doc2vec, word2vec and fastText as embedding methods did not yield better results than using our baseline TF-IDF method. There were no significant performance differences noticed using either word2vec or fastText: both these methods preformed slightly better than using doc2vec. In all cases using a window size larger than 5 showed beneficial results, doc2vec was the most sensitive to smaller context windows and showed severe performance degradation using a window size smaller than 11.

Table 5.1: Results obtained on the validation set using the logistic regression classifier.

Embedding method	**Window**	**Cohen kappa validation set**
TF-IDF (Dictionary = 100 000)	41	**0.437**
word2vec SG	61	0.324
fastText SG	61	0.325
doc2vec DBoW	121	0.306

Figure 5.1 depicts how doc2vec is able to capture related concepts. Concepts such as cars depicted by the keywords of car names are close to one another, as well as keywords related to real estate and business.

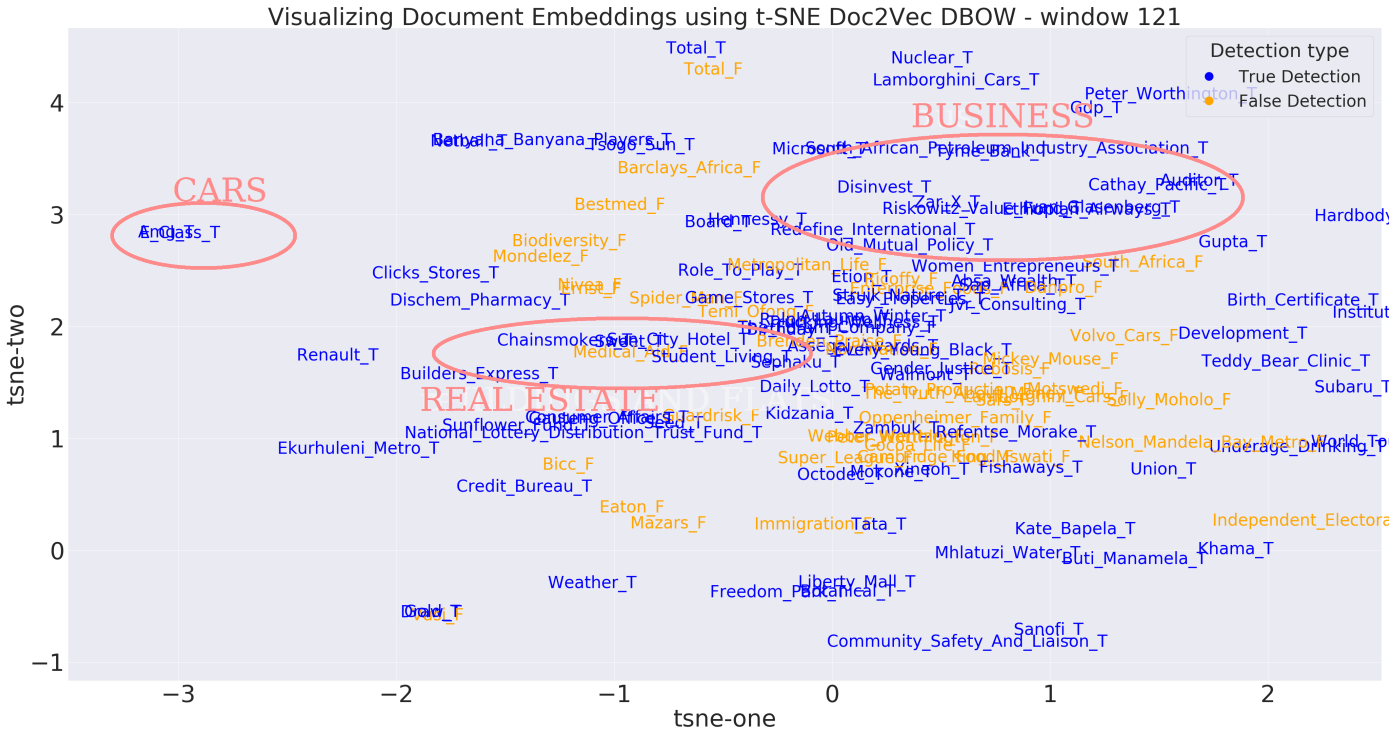


Figure 5.1: Illustration using t-SNE of document vectors that were trained using window size of 121.

Using fastText, a t-SNE plot is created using the top 20 words with the highest cosine similarity with the specific keywords shown in Figure 5.2. This plot shows different word clusters forming with different keywords in the news. A word such as “jaguar” surrounds itself with different model names of the “Jaguar” brand, and names such as “land rover”. This is relevant, as these are both British car makers owned by the same holding company Tata Motors. They were merged in 2013 to form Jaguar Land Rover Limited. These words are separated from common words associated with keywords such as “eskom” and “city of cape town”, which are surrounded by completely different words.

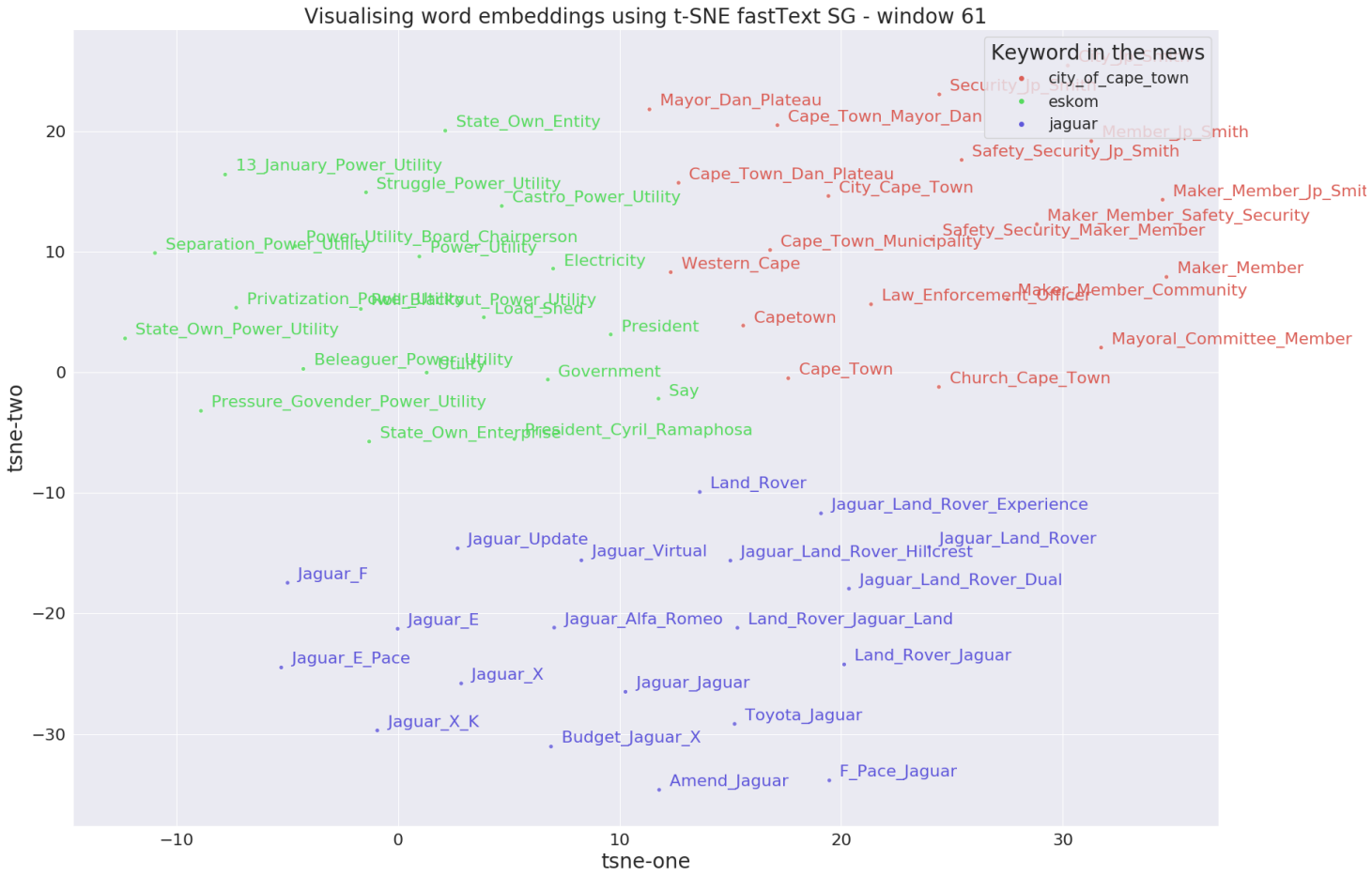


Figure 5.2: A depiction using t-SNE of word vectors trained using fastText.

5.3 Average cosine distance method

The average cosine similarity method used to classify words based on true and false flags (i.e. by tagging specific keywords) achieved better performance than the logistic regression classifier. Considering fastText, the CBoW method did significantly worse than using the SG method. The best result was obtained using the same window size. Word2vec also showed the best performance using SG. Using the CBoW method the performance differences were not as drastic as with fastText. Using slightly larger win-

dow sizes also seemed beneficial. It is interesting to note that using a vocabulary size of approximately 155 000 proved useful in most cases. Note that the vocabulary size is slightly smaller than that reported in Table 3.3, as words that appear only once are discarded for meaningful embeddings to be trained (see Appendix C.1.2 for additional information).

Table 5.2: Results obtained on the validation set using the average cosine distance method.

Embedding method	**Window**	**Cohen kappa validation set**	**Vocabulary size**
word2vec SG	61	0.330	143 190
word2vec CBoW	121	0.307	154 873
fastText SG	21	**0.364**	155 421
fastText CBoW	21	0.250	155 421

Figure 5.3 shows a t-SNE plot for the top 50 words that were most similar to each case of the keyword "eskom", denoting its true and false detection type. This is based on cosine similarity between all words in the fastText model's dictionary and those two specific keywords. It is interesting to note that there is a definite separation between these cases, although the keywords themselves seem rather close to one another. This might be due to the effect of common words shared by both keywords, which during training implicitly cause these words vectors to approach one another.

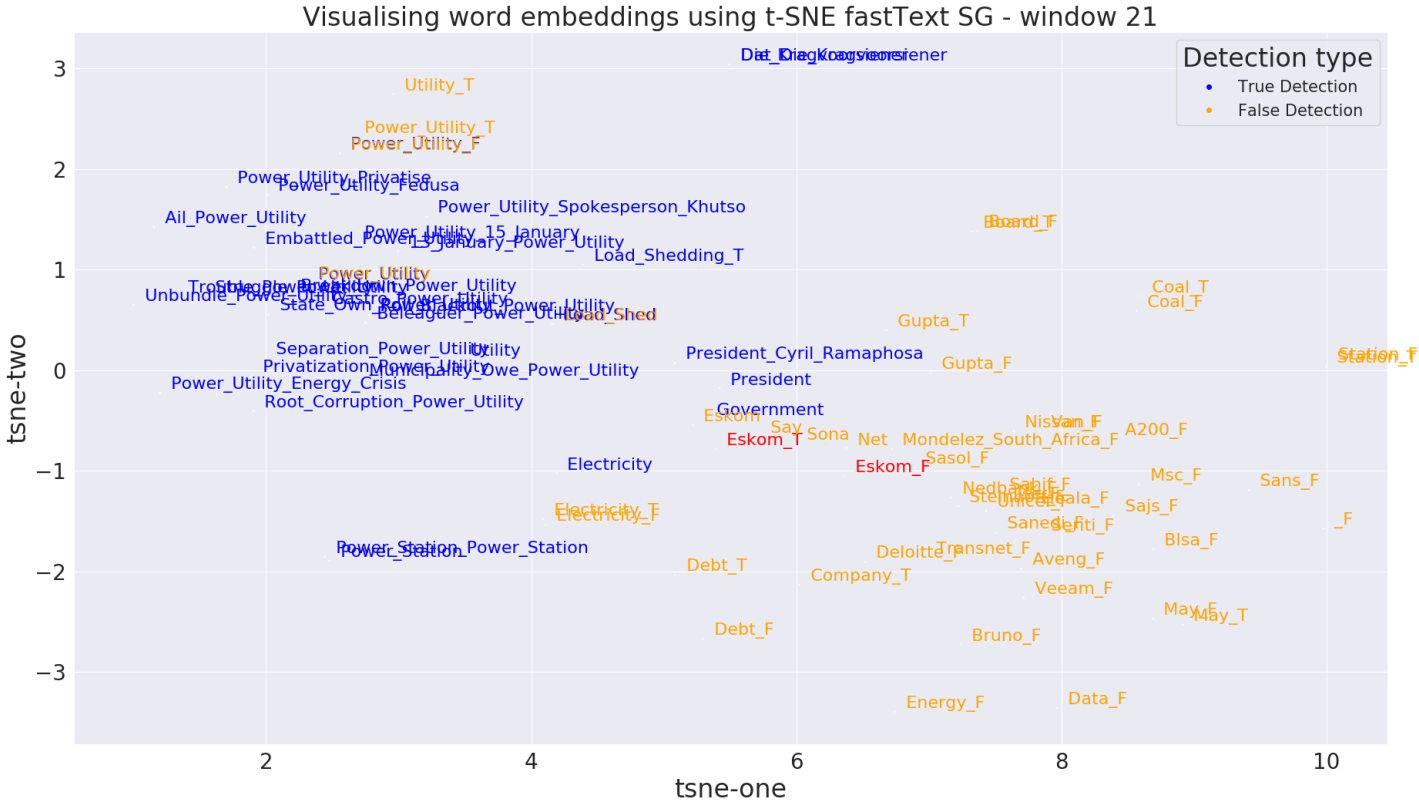


Figure 5.3: Figure illustrating how keywords are separated from each other using flags, the keyword "eskom" is used as an example.

5.4 Feedforward neural network classifier

In this section, we investigate how a feedforward neural network classifier is used with various embedding methods to classify text. Comparing the results obtained in Table 5.3, with the result obtained by the logistic regression function in Table 5.1 a significant difference in performance can be noticed with regard to the word and document embedding techniques.

Table 5.3: Results obtained on the validation and test set using a feedforward neural network classifier with varying widths.

Embedding method	**Window**	**Cohen kappa DNN-100 validation set**	**Cohen kappa DNN-200 validation set**	**Cohen kappa DNN-400 validation set**	**Cohen kappa DNN-400 test set**
TF-IDF (Dictionary = 100 000)	121	**0.461**	**0.466**	**0.467**	**0.465**
word2vec SG	21	0.383	0.393	0.398	0.393
fastText SG	11	0.386	0.391	0.399	0.393
doc2vec DBoW	121	0.405	0.408	0.416	0.415

This difference is made more clear looking at the ROC curve shown in Figure 5.4, indicating a stark difference in performance between the two classifiers.

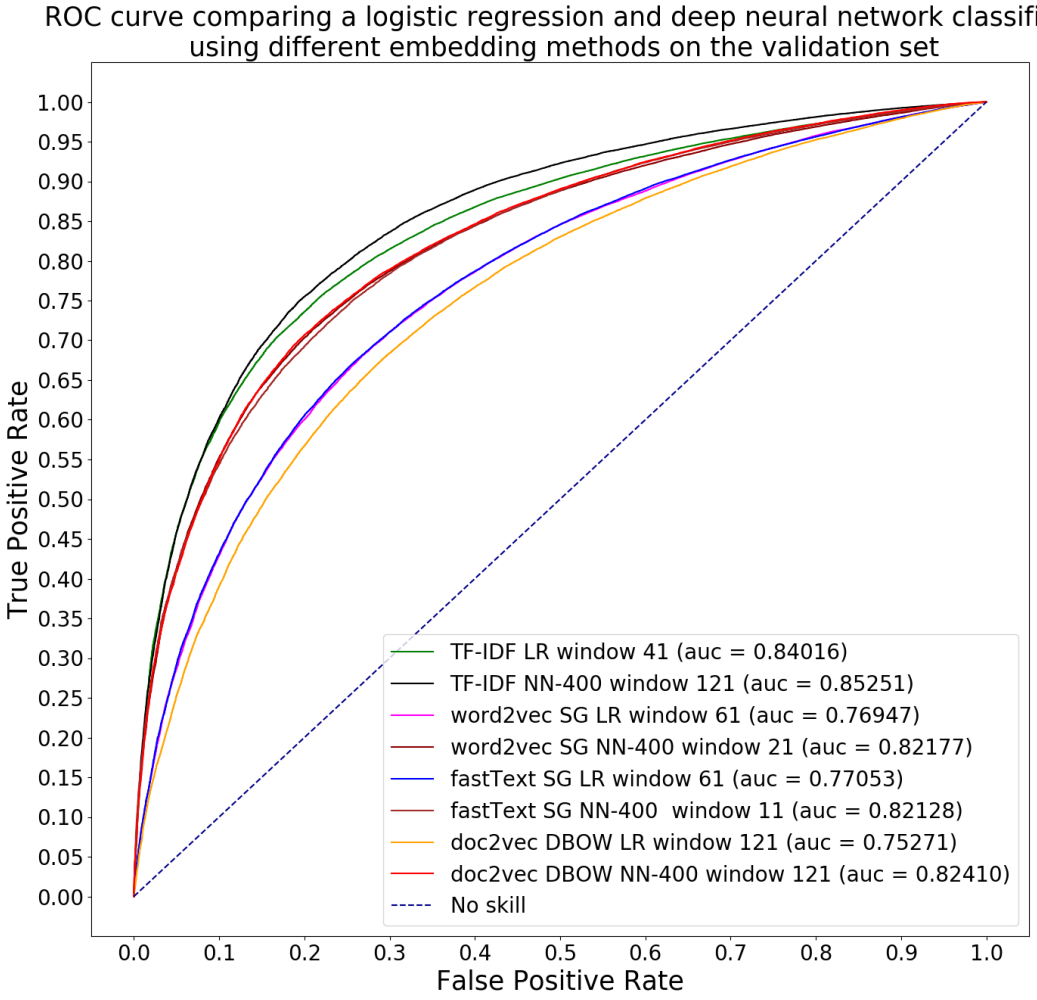


Figure 5.4: ROC plot for DNN-400 and LR classifier evaluated on the validation set using different embedding technique.

TF-IDF did not reveal such a notable improvement compared to the other embedding methods used. This distinction is due to word2vec, fastText and doc2vec embeddings being non-linear feature representations. Logistic regression is a linear classifier and is unable to distinguish between these non-linear features effectively. In contrast, the neural network is a non-linear classifier and is better equipped to capture the non-linearities present in these features. The ROC curve for the final results are shown in Figure 5.5 (additional information can be found in Appendix C.1.3).

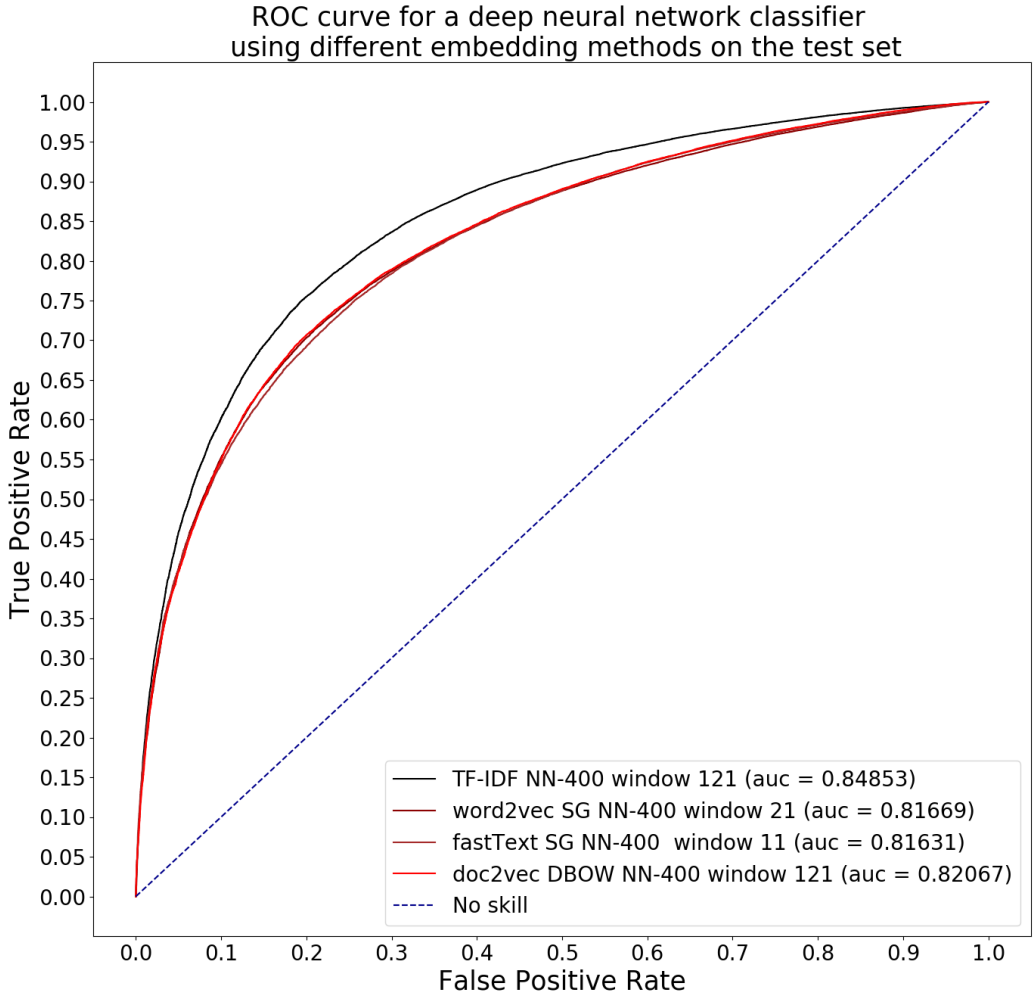


Figure 5.5: ROC plot for DNN-400 classifier evaluated on the test set using different embedding technique.

Another interesting observation with regard to TF-IDF and doc2vec seems to be that larger window sizes typically yield better results than smaller windows. Doc2vec was also the best-performing method among the embedding techniques. Although the Cohen kappa metric for the word2vec and fastText embeddings were not far behind. One caveat associated with using doc2vec is that inferring document vectors typically tends to take significantly longer than inferring an average sentence vector using word2vec or fastText. This might be a problem for real time application. Increasing the capac-

ity of the feedforward neural network with more hidden nodes increases the Cohen kappa metric for all embedding methods although the performance starts to plateau after increasing the neural network size beyond 200 hidden nodes. We consider the DNN with 400 hidden nodes, as it obtained the best results.

It is interesting to note that smaller window sizes tend to perform better for both fastText and word2vec embeddings. A possible reason for this might be that averaging word vectors over larger window sizes results in some information loss with regard to important words in the context window, which might indicate a FP or its absence. In light of the t-SNE plots in Figure 5.6—5.9, it is intriguing to notice that for both fastText and word2vec embeddings. Denser clusters are formed for smaller window sizes and more spread-out clusters for larger window sizes. This strengthens the intuition that the meaning of certain words using smaller window sizes is more clearly conveyed in the words used to create an average sentence vector, which might lead to better classification.

Having a larger vocabulary of trained word embeddings might lead to more common words sharing their meaning with other words in the embedding dictionary, leading to more spread-out clusters.

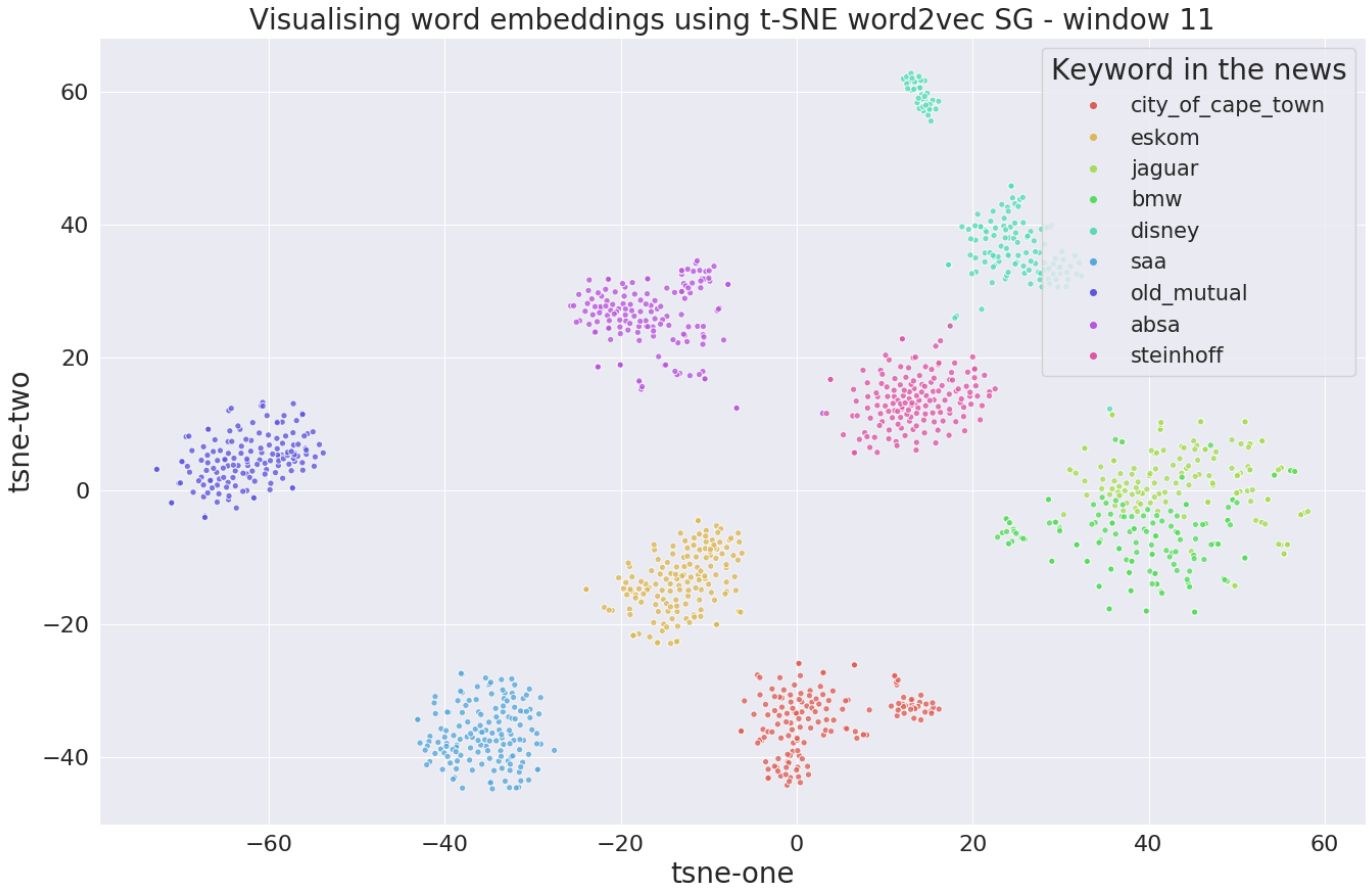


Figure 5.6: t-SNE scatter plot for word2vec embeddings using a window size of 11, with unique keywords. Words with the highest-ranking cosine similarity with each keyword are shown.

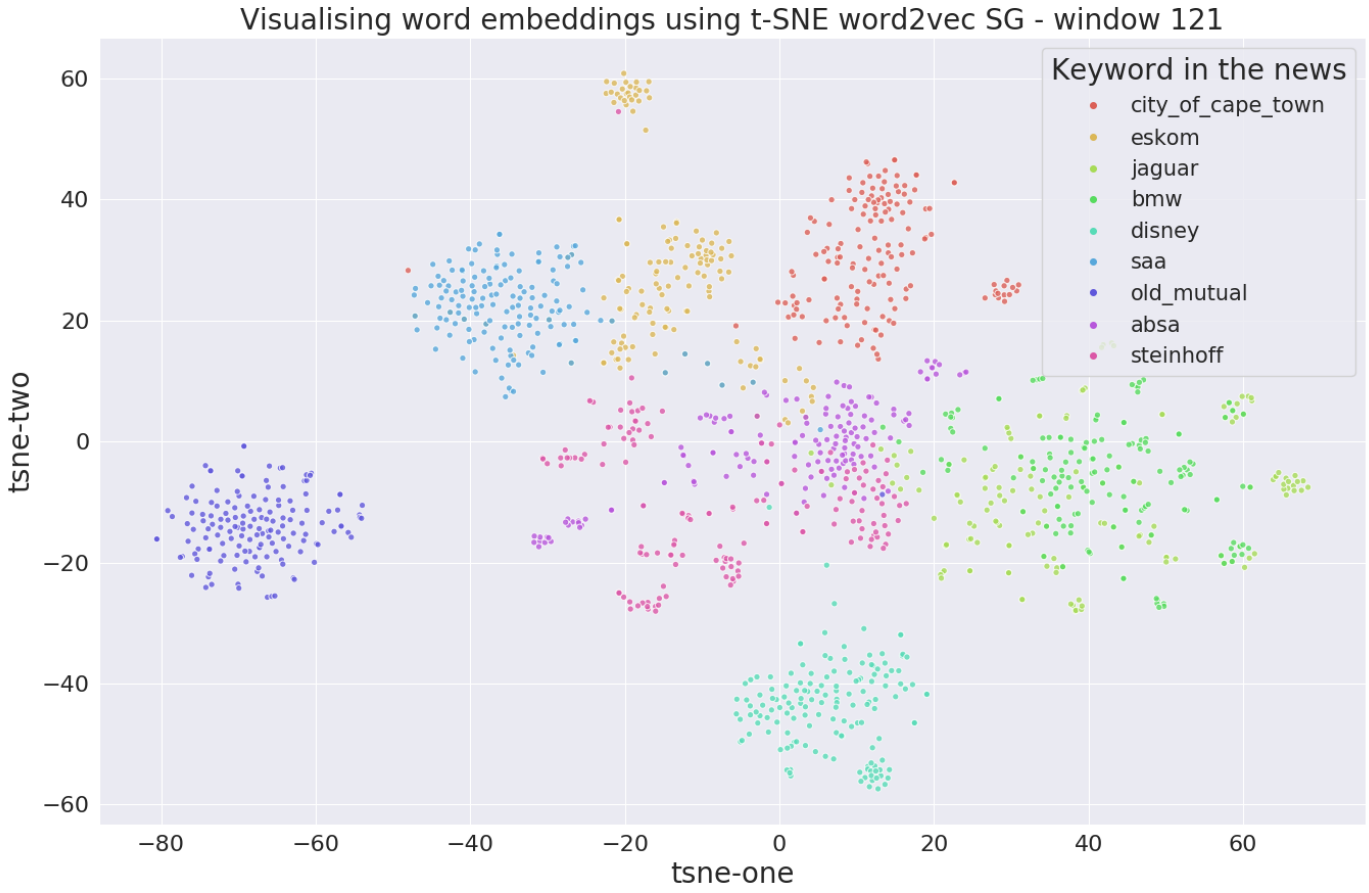


Figure 5.7: t-SNE scatter plot for word2vec embeddings using a window size 121, with unique keywords. Words with the highest-ranking cosine similarity with each keyword are shown.

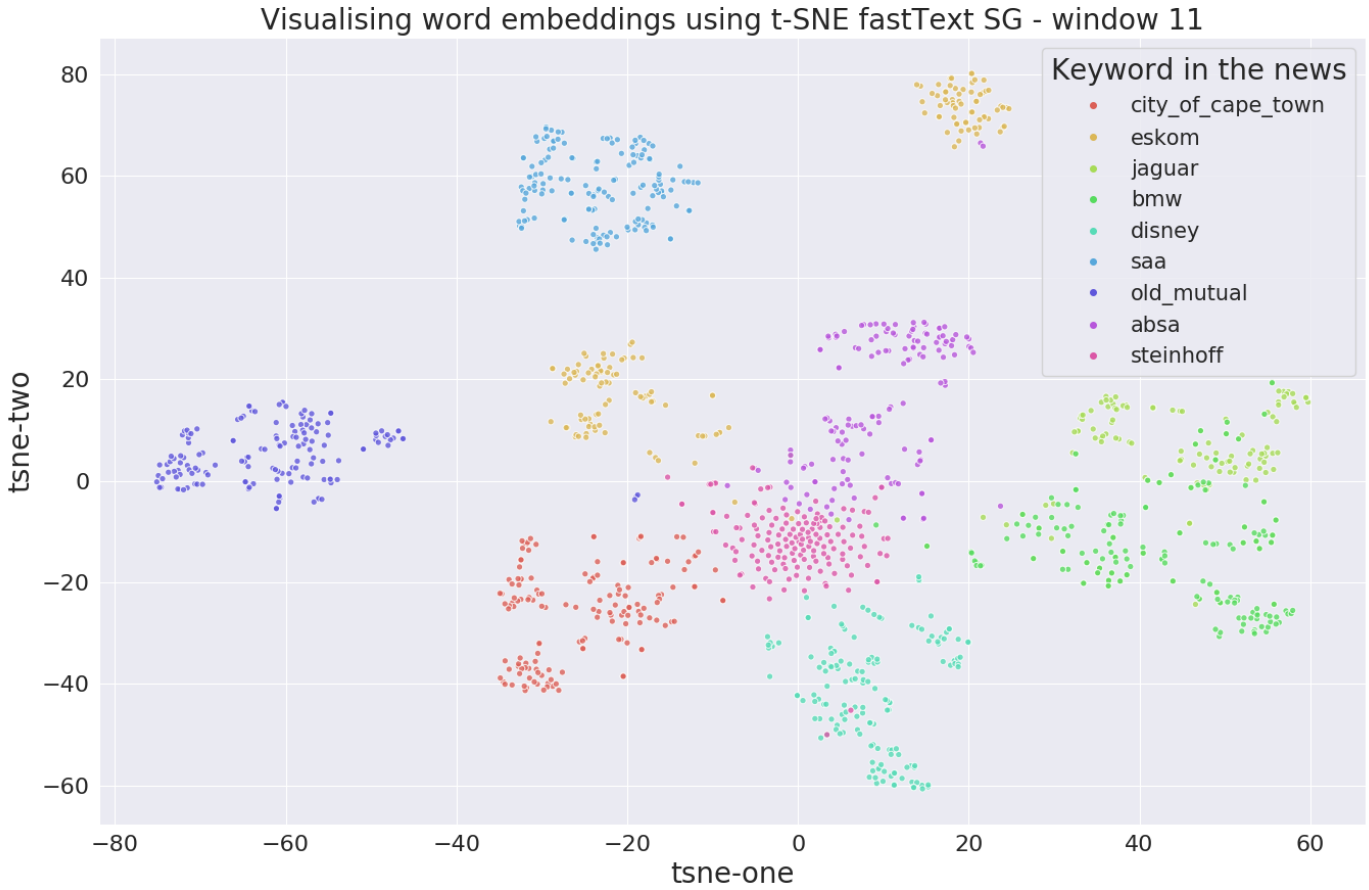


Figure 5.8: t-SNE scatter plot for fastText embeddings using a window size 11, with unique keywords. Words with the highest-ranking cosine similarity with each keyword are shown.

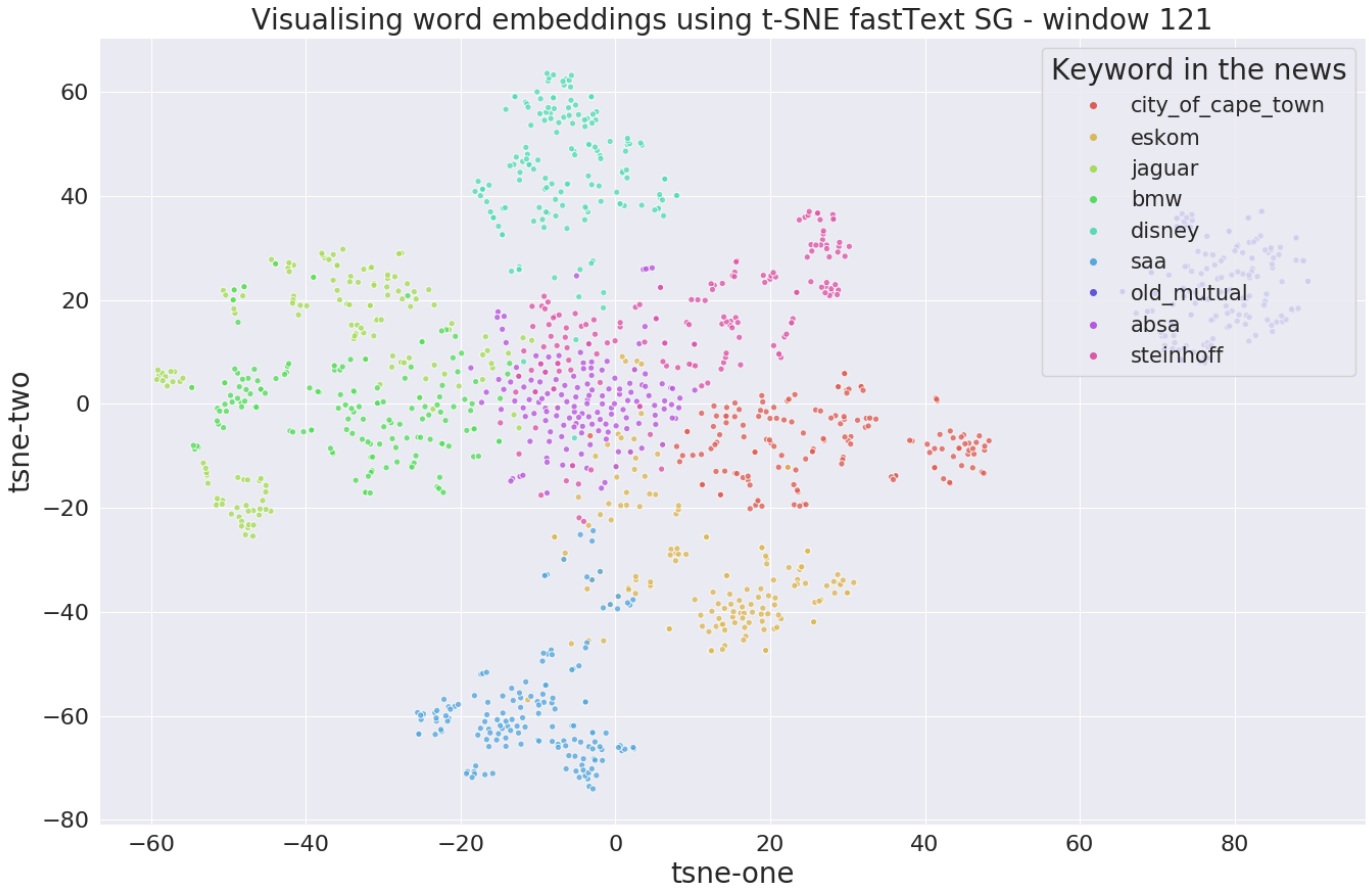


Figure 5.9: t-SNE scatter plot for fastText embeddings using a window size 121, with unique keywords. Words with the highest-ranking cosine similarity with each keyword are shown.

5.5 Adding flags to keywords using DNNs

With regard to the method described in Section 4.4.4, the results shown in Table 5.4 were obtained for a feedforward neural network with a single hidden layer of 400 nodes. Using smaller window sizes yielded a moderately better result than larger windows for both fastText and word2vec embeddings. Applying fastText embeddings

resulted in a slightly higher Cohen kappa score than using word2vec, although nothing substantially different from the scores obtained without adding flags. This might be due to stronger word embeddings, trained on more words, being needed to give a clearer indication of whether or not a falsely mentioned word was encountered. The method did, however, perform better than using average cosine distance between words and keywords alone, as described in Section 5.3, refer to Appendix C.1.4 for additional information.

Table 5.4: Results obtained on the validation and test set using a 400 node feedforward neural network classifier using the flag method.

Embedding method	**Window**	**Cohen kappa validation set**	**Cohen kappa test set**
word2vec SG	21	0.385	0.381
fastText SG	21	**0.407**	**0.404**

5.6 CatBoost classifier

In this section, we investigate the performance of a CatBoost classifier using the available metadata associated with the speech-text data. The results from Table 5.5 show that the CatBoost classifier performs the same over all window sizes, which is to be expected as no information is added or removed by changing the window size with regard to the categorical features. The small discrepancies between the different window sizes is a result of slight differences in corpus size for the validation and test set resulting from the grouping effect described in Section 3.5.

Table 5.5: Results obtained using the CatBoost classifier, only using metadata associated with text.

Window	**Cohen kappa validation set**	**Cohen kappa test set**
5	0.499	0.496
11	0.500	0.494
21	0.501	0.495
41	0.499	0.493
61	0.497	0.492
121	0.495	0.494

SHAP values were used to determine the importance of each feature, as shown in Figure 5.10—5.13. The most important features for either small or large window sizes seems to be the target word identified and the broadcast station's name. The broadcast station's language, day of the week and total duration of the recorded clip (i.e. "total_minute") do not have much effect on the classification result, these features can thus be removed.

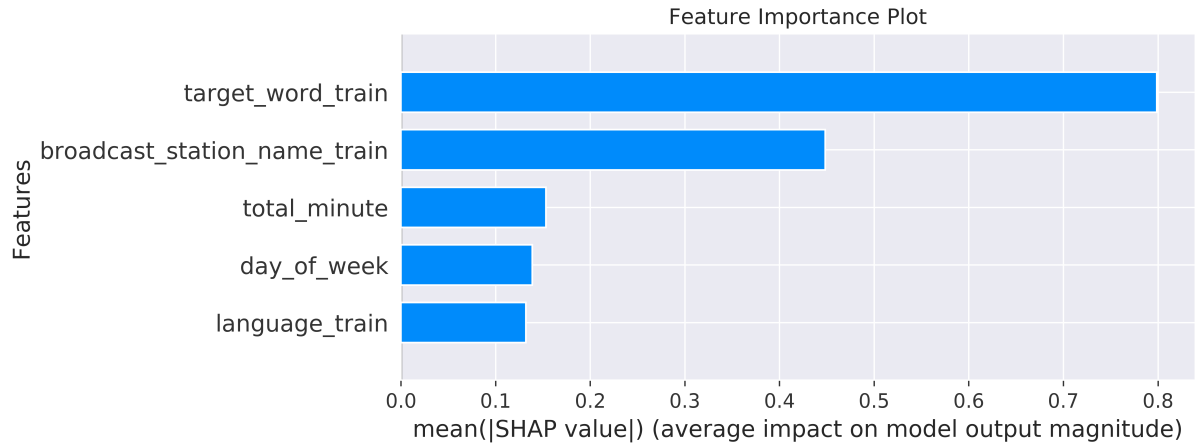


Figure 5.10: Mean absolute SHAP values of baseline CatBoost classifier for window size of 5, evaluated on validation set.

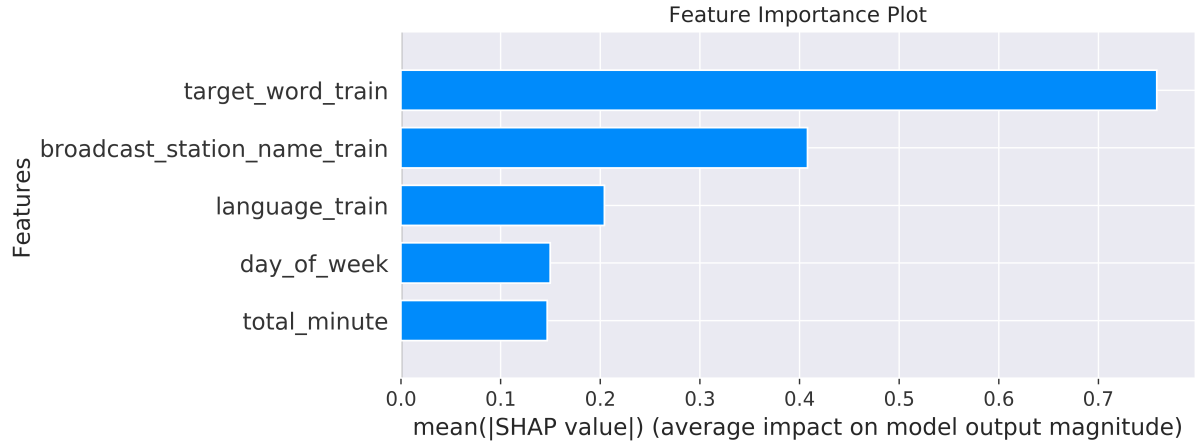


Figure 5.11: Mean absolute SHAP values of baseline CatBoost classifier for window size of 121, evaluated on validation set.

The baseline CatBoost classifier achieves good results by relying on the target word and broadcast station's name as its most important input features. Adding the output from the 400 hidden node feedforward neural network using the TF-IDF method, however, degrades the model's performance, shown in Table 5.6. The model also becomes less reliant on the broadcast station's name and target word as important input features, and sees the neural network's output as the most important indicator of whether a FP or true detection was encountered, shown in Figure 5.12 —5.14, overfitting the CatBoost classifier to that specific feature. The five meta-data features are retained for all

CatBoost classifiers trained, also when the output of the DNN is added as an additional input feature. Using the output of the 400 hidden node neural network with word2vec and fastText (i.e. weaker models), however, slightly improves the CatBoost classifier performance. Similar to the baseline CatBoost classifier adding the broadcast station's language, day of the week and total duration of the recorded clip do not have much effect on the classification result, and can thus be removed.

Table 5.6: Results obtained using the CatBoost classifier and output of 400-node DNN.

Embedding method	**Window**	**Cohen kappa DNN-400 and CatBoost validation set**	**Cohen kappa DNN-400 and CatBoost test set**
TF-IDF (Dictionary = 100 000)	121	0.476	0.476
word2vec SG	21	**0.512**	**0.508**
fastText SG	11	0.510	**0.508**

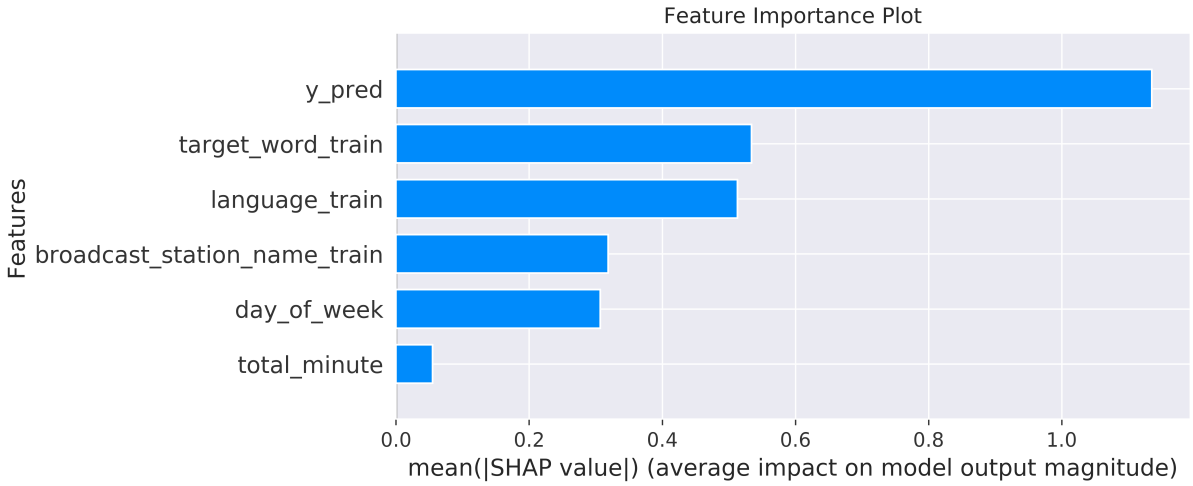


Figure 5.12: Mean absolute SHAP values of CatBoost classifier adding output of the 400 node neural network using TF-IDF, evaluated on validation set.

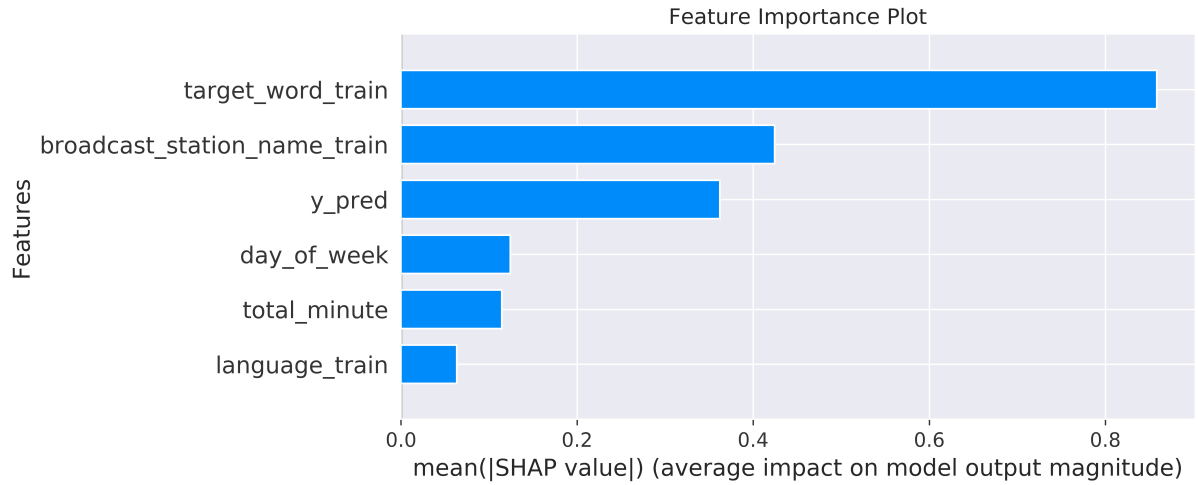


Figure 5.13: Mean absolute SHAP values of CatBoost classifier adding output of the 400 node neural network using word2vec SG, evaluated on validation set.

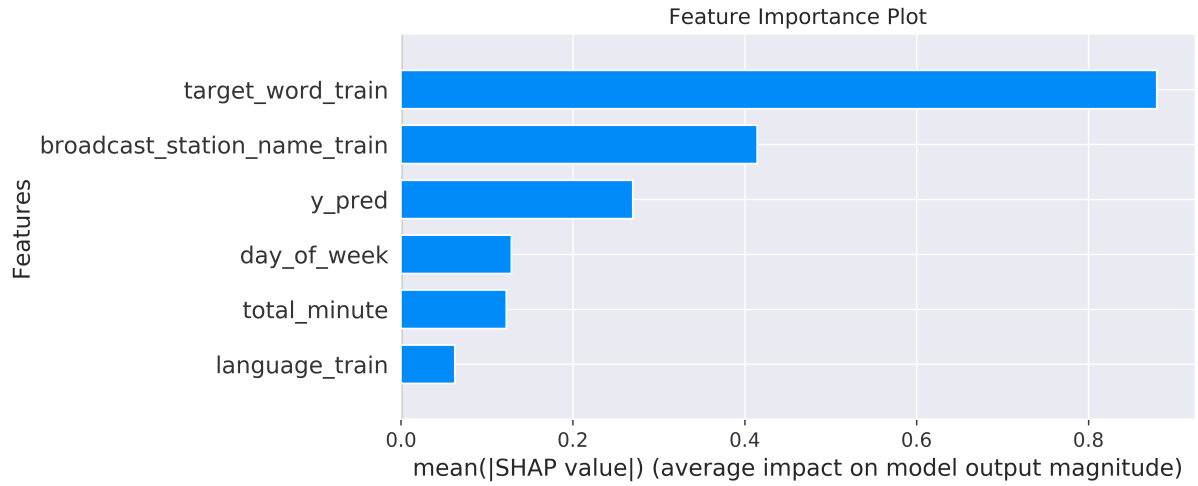


Figure 5.14: Mean absolute SHAP values of CatBoost classifier adding output of the 400 node neural network using fastText SG, evaluated on validation set.

5.7 Hardware used

Training of the various classifiers and embeddings was performed on a server with an Intel Xeon Processor E5-2695 with 14 cores and 28 threads clocked at 2.3GHz, with 128GB RAM memory. Training embeddings on this hardware required approximately

20min to 40 minutes, depending on the exact configuration.

5.8 Conclusion

In this chapter we investigated the use of different classifiers using both the textual and non-textual data obtained from the ASR system to classify whether or not a false positive word occurred in broadcast news. We considered the use of four embedding methods and found using our baseline TF-IDF method with a 400-node one layer neural network gave the best overall performance when only considering the textual features to do classification. Using the non-textual categorical features associated with text proved to give better results than using the textual features alone. The CatBoost classifier is more clearly able to distinguish whether or not a false positive word occurred using the broadcast station name and target word (i.e. keyword). Adding the output from feed-forward neural network to help with classification only yielded slightly better results than using weaker embedding methods to classify text such as word2vec or fastText. Using TF-IDF caused the CatBoost model to overfit on the output of the neural network resulting in slightly worse performance than obtained discarding this additional feature. Adding flags to keywords did not yield any significant performance gains in comparison with adding no flags at all. More data (i.e. examples of false positive and true positives for specific keyword from the ASR system) might be necessary to train better embeddings to more clearly distinguish between certain words.

Chapter 6

Conclusion and Recommendations

In this chapter, we indicate the main conclusion derived from the study and make recommendations for future work.

6.1 Introduction

As initially discussed in Chapter 1, the aim of the dissertation was to improve FP identification in a speech recogniser output using DNNs. This chapter discusses the degree to which the initial objectives of the study were met. We examine some of the important findings that were made and discuss possible avenues for future work.

6.2 Observations and findings

In this study, different word embedding techniques were investigated employing a DNN and logistic regression classifier, adding the best textual representation output

of the DNN classifier as an additional input feature to a CatBoost classifier. The research question that was considered was: *is it possible to use word embeddings to identify FP words in recognised speech?* Using word embeddings showed it was possible to obtain a reasonably good result for both (i.e. word2vec and fastText) classifiers (i.e. fair agreement concerning the Cohen kappa metric); however, using TF-IDF as a textual representation with a feedforward neural network outperformed all other embedding-based methods. TF-IDF is able to capture important features clearly in text that the machine learning algorithm can use to discriminate between an FP and TP word. This is due to the limited amount of data and the domain-specific nature (i.e. broadcast speech-text data) of the task. Training word2vec, doc2vec and fastText embeddings on a larger corpus would result in better word embeddings, increasing the models' chances of detecting a falsely mentioned word.

In light of the research question (is there a way to capture context with embeddings to help identify FP words more accurately?), we found that using context-specific embedding methods such as doc2vec yielded slightly better results than using word2vec and fastText, especially for larger window sizes (i.e. larger context windows), but still lagged behind statistical non-context-specific document embedding methods such as TF-IDF. Moreover, using flags to split the context in which particular keywords appeared, using word2vec and fastText, did not yield any significant performance increases either when applied as input features to a DNN or when applying the average cosine similarity method. This is due to the limited number of FP examples used to train embeddings for these words; for a clearer separation between classes, more examples of FP keywords are needed. As there are 4 186 unique keywords, with differing amounts of TP and FP occurrences, a substantial number of these examples have too few FP examples. Considering, for example, the word "constitutional_court" in Figure 3.5, which only has 179 examples of being identified as an FP, this can have a significant impact on performance, as there are not enough examples of words such as this to incorporate the context accurately.

On the question of how these trained word embeddings can be used alongside a DNN classifier to determine whether the words were correctly identified, we found that us-

ing a DNN network classifier showed significant performance differences compared to our baseline logistic regression classifier. This is not surprising; what is, however, surprising is the performance difference noticed between using other embedding methods and TF-IDF. The performance difference in using a DNN with word2vec, fastText and doc2vec was more drastic than the TF-IDF method. This is due to the non-linearity of these embedding methods when compared to TF-IDF. DNNs are better able to distinguish between these non-linear embeddings than linear classification methods such as logistic regression.

We also found that using the output of the feedforward neural network using TF-IDF as an additional feature to the CatBoost classifier degraded its performance. The CatBoost classifier overfitted on this feature, neglecting additional information provided, such as the broadcaster's name and keyword of interest. Considering the output of the DNN using a weaker textual representation method such as word2vec or fastText slightly enhanced the models' performance compared to the baseline CatBoost classifier relying on the metadata associated with the speech-text data. Using the metadata associated with the textual features consistently outperformed any method that used only the textual features to do classification. More interesting, the target word of interest and the broadcast station name were among the strongest features used to determine whether or not an FP word had occurred. Using these features to do classification, however, might be problematic in the long run. For example, the keywords associated with a specific broadcaster are likely to change over time, producing a significant degradation in performance, as the CatBoost model is heavily reliant on these features to do good classification. A strong argument can thus be made for primarily considering the textual features associated with the data as these are more robust to these changes.

In preliminary investigations, using the SG over CBoW based method for word2vec and fastText yielded slightly better results. This is attributed to SG paying more attention to rarer words in comparison with CBoW, which attempts to predict the word given the context. SG furthermore forces the model to learn the context of a specific word. CBoW will thus perform better on a larger training set [45] compared to SG,

which does not average out peculiarities of a specific word, given a context window. The same observations are made using PV-DM with doc2vec. It should also be noted that some labelling errors were found, especially with regard to car names, such as "BMW", where the data were labelled as false, but given the context and video clip, it was a TP. These types of examples can influence the performance of classifiers trained on such material, reducing the identification accuracy for some keywords of interest.

In conclusion, the contributions of this work are twofold: on the one hand, we showed that text obtained from a speech-recognition system provides an interesting and important use case for NLP techniques. On the other, we demonstrated the relative capabilities of various NLP solutions, including embeddings and more traditional methods on such a task in the multilingual South African context.

6.3 Suggestions for future work

The following observations were made with regard to future work; concerning the classification of falsely mentioned words using the textual data from the ASR system.

- The use of a bi-directional long short-term memory (LSTM) [46] network could be investigated. A bi-directional LSTM is commonly used for sequential modeling problems and retains past and future context information without destroying word order. A brief exploration was done using word2vec and fastText with different window sizes, as shown in Appendix C.2, without any rigorous optimisation. It showed almost similar performance to that of the baseline TF-IDF method using a 400-node DNN.

- Better word vectors could be trained by procuring more ASR text data; word embeddings are typically trained on large corpora in excess of a billion words. Procuring more text data in all languages would improve classification results using any machine-learning method, as more meaningful word embeddings could be trained.

- Context-aware embeddings such as ELMo [17] or BERT [18] could be trained and fine-tuned for a specific text classification task.
- Another intriguing option might be to apply a character-level CNN to classify text. Zhang [47] shows that character-level convolution does not require any semantic or syntactic understanding of language to classify text, as text is treated as a raw signal of characters. They claim that these networks could be beneficial in developing a single system that works for different languages. The only caveat with this method is that it requires large amounts of training data.
- Further examining FP detection that transpires from specific broadcasters or languages, specifically focusing on FP that are unique between languages or unique to a specific broadcaster.
- Investigating how specific languages such as Zulu or Xhosa, written in a conjunctive manner, influences results, especially with regard to the high number of word types that these languages have and how it influences word embeddings in a multi-lingual environment.

Bibliography

[1] R. Rosenfeld, "Two decades of statistical language modeling: Where do we go from here?" *Proceedings of the IEEE*, vol. 88, no. 8, pp. 1270–1278, 2000. DOI: 10.1109/5.880083.

[2] T. Young, D. Hazarika, S. Poria, and E. Cambria, "Recent trends in deep learning based natural language processing [Review Article]," *IEEE Computational Intelligence Magazine*, vol. 13, no. 3, pp. 55–75, 2018, ISSN: 15566048. DOI: 10.1109/MCI.2018.2840738. arXiv: arXiv:1708.02709v8.

[3] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *Proceedings of Workshop at ICLR*, vol. 2013, pp. 1–12, Jan. 2013.

[4] *Word2vec: Tensorflow vector representation of words*, https://data-flair.training/blogs/tensorflow-word2vec/, Accessed: 2020-10-20.

[5] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543. [Online]. Available: http://www.aclweb.org/anthology/D14-1162.

[6] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching Word Vectors with Subword Information," *Transactions of the Association for Computational Linguistics*, vol. 5, pp. 135–146, 2017. DOI: 10.1162/tacl_a_00051. arXiv: arXiv:1607.04606v2.

[7] Z. S. Harris, "Distributional structure," *WORD*, vol. 10, no. 2-3, pp. 146–162, 1954. DOI: 10.1080/00437956.1954.11659520.

[8] Y. Goldberg, "A primer on neural network models for natural language processing," *J. Artif. Int. Res.*, vol. 57, no. 1, pp. 345–420, Sep. 2016, ISSN: 1076–9757.

[9] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989. DOI: 10.1162/neco.1989.1.4.541.

[10] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," *31st International Conference on Machine Learning, ICML 2014*, vol. 4, May 2014.

[11] K. S. Jones, "A statistical interpretation of term specificity and its application in retrieval," *Journal of Documentation*, vol. 28, pp. 11–21, 1972.

[12] T. Tokunaga and I. Makoto, "Text categorization based on weighted inverse document frequency," in *Special Interest Groups and Information Process Society of Japan (SIG-IPSJ*, 1994, pp. 33–39.

[13] C. Chelliah, G. M, B. Subramanian, K. S.M.A, and N. Ramaraj, "A novel term weighting scheme midf for text categorization," *Journal of Engineering Science and Technology*, vol. 5, Mar. 2010.

[14] T. Joachims, "Text categorization with support vector machines: Learning with many relevant features," in *Proceedings of the 10th European Conference on Machine Learning*, ser. ECML'98, Chemnitz, Germany: Springer-Verlag, 1998, pp. 137–142, ISBN: 3540644172. DOI: 10.1007/BFb0026683. [Online]. Available: https://doi.org/10.1007/BFb0026683.

[15] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds., Curran Associates, Inc., 2013, pp. 3111–3119. [Online]. Available: http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf.

[16] T. Schnabel, I. Labutov, D. Mimno, and T. Joachims, "Evaluation methods for unsupervised word embeddings," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, Lisbon, Portugal: Association for Computational Linguistics, Sep. 2015, pp. 298–307. DOI: 10.18653/v1/D15-1036. [Online]. Available: https://www.aclweb.org/anthology/D15-1036.

[17] M. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, New Orleans: Association for Computational Linguistics, Jun. 2018, pp. 2227–2237. DOI: 10.18653/v1/N18-1202. [Online]. Available: https://www.aclweb.org/anthology/N18-1202.

[18] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423. [Online]. Available: https://www.aclweb.org/anthology/N19-1423.

[19] D. Bahdanau, K. Cho, and Y. Bengio, *Neural machine translation by jointly learning to align and translate*, cite arxiv:1409.0473Comment: Accepted at ICLR 2015 as oral presentation, 2014. [Online]. Available: http://arxiv.org/abs/1409.0473.

[20] S. Hochreiter, Y. Bengio, P. Frasconi, and J. Schmidhuber, "Gradient flow in recurrent nets: The difficulty of learning long-term dependencies," in *A Field Guide to Dynamical Recurrent Neural Networks*, S. C. Kremer and J. F. Kolen, Eds., IEEE Press, 2001.

[21] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *CoRR*, vol. abs/1706.03762, 2017. arXiv: 1706.03762. [Online]. Available: http://arxiv.org/abs/1706.03762.

[22] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2019.

[23] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "Liblinear: A library for large linear classification," *Journal of Machine Learning Research*, vol. 9, pp. 1871–1874, Aug. 2008. DOI: 10.1145/1390681.1442794.

[24] M. Anjaria and R. M. R. Guddeti, "Influence factor based opinion mining of twitter data using supervised learning," *2014 Sixth International Conference on Communication Systems and Networks (COMSNETS)*, pp. 1–8, 2014.

[25] L. Manevitz and M. Yousef, "One-class document classification via neural networks," *Neurocomputing*, vol. 70, pp. 1466–1481, Mar. 2007. DOI: 10.1016/j.neucom.2006.05.013.

[26] D. Ram, A. Asaei, and H. Bourlard, "Sparse subspace modeling for query by example spoken term detection," *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 26, no. 6, pp. 1130–1143, 2018. DOI: 10.1109/TASLP.2018.2815780.

[27] K. Kowsari, K. J. Meimandi, M. Heidarysafa, S. Mendu, L. E. Barnes, and D. E. Brown, "Text classification algorithms: A survey," *CoRR*, vol. abs/1904.08067, 2019. arXiv: 1904.08067. [Online]. Available: http://arxiv.org/abs/1904.08067.

[28] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, pp. 139, 145, 167–170, 192–194, http://www.deeplearningbook.org.

[29] C. De Boom, S. Van Canneyt, T. Demeester, and B. Dhoedt, "Representation learning for very short texts using weighted word embedding aggregation," *Pattern Recogn. Lett.*, vol. 80, no. C, pp. 150–156, Sep. 2016, ISSN: 0167-8655. DOI: 10.1016/j.patrec.2016.06.012. [Online]. Available: https://doi.org/10.1016/j.patrec.2016.06.012.

[30] B. P. Roe, H.-J. Yang, J. Zhu, Y. Liu, I. Stancu, and G. McGregor, "Boosted decision trees, an alternative to artificial neural networks," *Nucl. Instrum. Meth. A*,

vol. 543, no. 2-3, pp. 577–584, 2005. DOI: 10.1016/j.nima.2004.12.018. arXiv: physics/0408124.

[31] X. He, J. Pan, O. Jin, T. Xu, B. Liu, T. Xu, Y. Shi, A. Atallah, R. Herbrich, S. Bowers, and J. Q. Candela, "Practical lessons from predicting clicks on ads at facebook," in *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising*, ser. ADKDD'14, New York, NY, USA: Association for Computing Machinery, 2014, pp. 1–9, ISBN: 9781450329996. DOI: 10.1145/2648584.2648589. [Online]. Available: https://doi.org/10.1145/2648584.2648589.

[32] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin, "Catboost: Unbiased boosting with categorical features," in *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., Curran Associates, Inc., 2018, pp. 6638–6648. [Online]. Available: http://papers.nips.cc/paper/7898-catboost-unbiased-boosting-with-categorical-features.pdf.

[33] K. Kowsari, K. J. Meimandi, M. Heidarysafa, S. Mendu, L. Barnes, and D. Brown, "Text classification algorithms: A survey," *Information*, vol. 10, no. 4, p. 150, 2019. DOI: 10.3390/info10040150. [Online]. Available: https://app.dimensions.ai/details/publication/pub.1113697716%20and%20https://www.mdpi.com/2078-2489/10/4/150/pdf.

[34] M. Honnibal, I. Montani, S. Van Landeghem, and A. Boyd, *spaCy: Industrial-strength Natural Language Processing in Python*, 2020. DOI: 10.5281/zenodo.1212303. [Online]. Available: https://doi.org/10.5281/zenodo.1212303.

[35] R. Řehůřek and P. Sojka, "Software Framework for Topic Modelling with Large Corpora," English, in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, http://is.muni.cz/publication/884893/en, Valletta, Malta: ELRA, May 2010, pp. 45–50.

[36] E. Altszyler, S. Ribeiro, M. Sigman, and D. Fernandez Slezak, "The interpretation of dream meaning: Resolving ambiguity using latent semantic analysis in a small corpus of text," *Consciousness and Cognition*, vol. 56, pp. 178–187, Nov. 2017, ISSN:

1053-8100. DOI: 10.1016/j.concog.2017.09.004. [Online]. Available: http://dx.doi.org/10.1016/j.concog.2017.09.004.

[37] J. H. Lau and T. Baldwin, "An empirical evaluation of doc2vec with practical insights into document embedding generation," in *Proceedings of the 1st Workshop on Representation Learning for NLP*, Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 78–86. DOI: 10.18653/v1/W16-1609. [Online]. Available: https://www.aclweb.org/anthology/W16-1609.

[38] T. Fawcett, "An introduction to roc analysis.," *Pattern Recognit. Lett.*, vol. 27, no. 8, pp. 861–874, 2006.

[39] W. J. Youden, "Index for rating diagnostic tests," *Cancer*, vol. 3, no. 1, pp. 32–35, 1950, ISSN: 10970142.

[40] J. Cohen, "A coefficient of agreement for nominal scales," *Educational and Psychological Measurement*, vol. 20, pp. 37–46, 1960.

[41] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., Curran Associates, Inc., 2017, pp. 4765–4774. [Online]. Available: https://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf.

[42] L. Van Der Maaten and G. Hinton, "Visualizing data using t-SNE," *Journal of Machine Learning Research*, vol. 9, pp. 2579–2625, 2008, ISSN: 15324435.

[43] H. Hottelling, "Analysis of a complex of statistical variables with principal components," *J. Educ. Psy.*, vol. 24, pp. 498–520, 1933. [Online]. Available: https://ci.nii.ac.jp/naid/10022591676/en/.

[44] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, Dec. 2014.

[45] L. Jin and W. Schuler, "A comparison of word similarity performance using explanatory and non-explanatory texts," in *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Denver, Colorado: Association for Computational Linguistics, May 2015, pp. 990–994. DOI: 10.3115/v1/N15-1101. [Online]. Available: https://www.aclweb.org/anthology/N15-1101.

[46] B. Jang, M. Kim, G. Harerimana, S. U. Kang, and J. W. Kim, "Bi-LSTM model to increase accuracy in text classification: Combining word2vec CNN and attention mechanism," *Applied Sciences (Switzerland)*, vol. 10, no. 17, 2020, ISSN: 20763417. DOI: 10.3390/app10175841.

[47] X. Zhang, J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28, Curran Associates, Inc., 2015, pp. 649–657. [Online]. Available: https://proceedings.neurips.cc/paper/2015/file/250cf8b51c773f3f8dc8b4be867a9a02-Paper.pdf.

Appendix A

ASR System and Classifying Speech-Text Data

Figure A.1 illustrates how the data obtained from the ASR system was labeled as false positive by the people working at the Novus group. Also shown is the speech-text data used as input to the various NLP algorithms investigated to classify whether a false positive had occurred. The specific example shows how the ASR system mistakenly classifies the word ”Disney” the company of with the Afrikaans word ”dis nie”. This is due to both word sounding the same but meaning different things. Other examples of FP that occur include words such as the ”JSE” (i.e. Johannesburg stock exchange) being mistaken for words in songs like ”Jessy” (i.e. someone’s name).

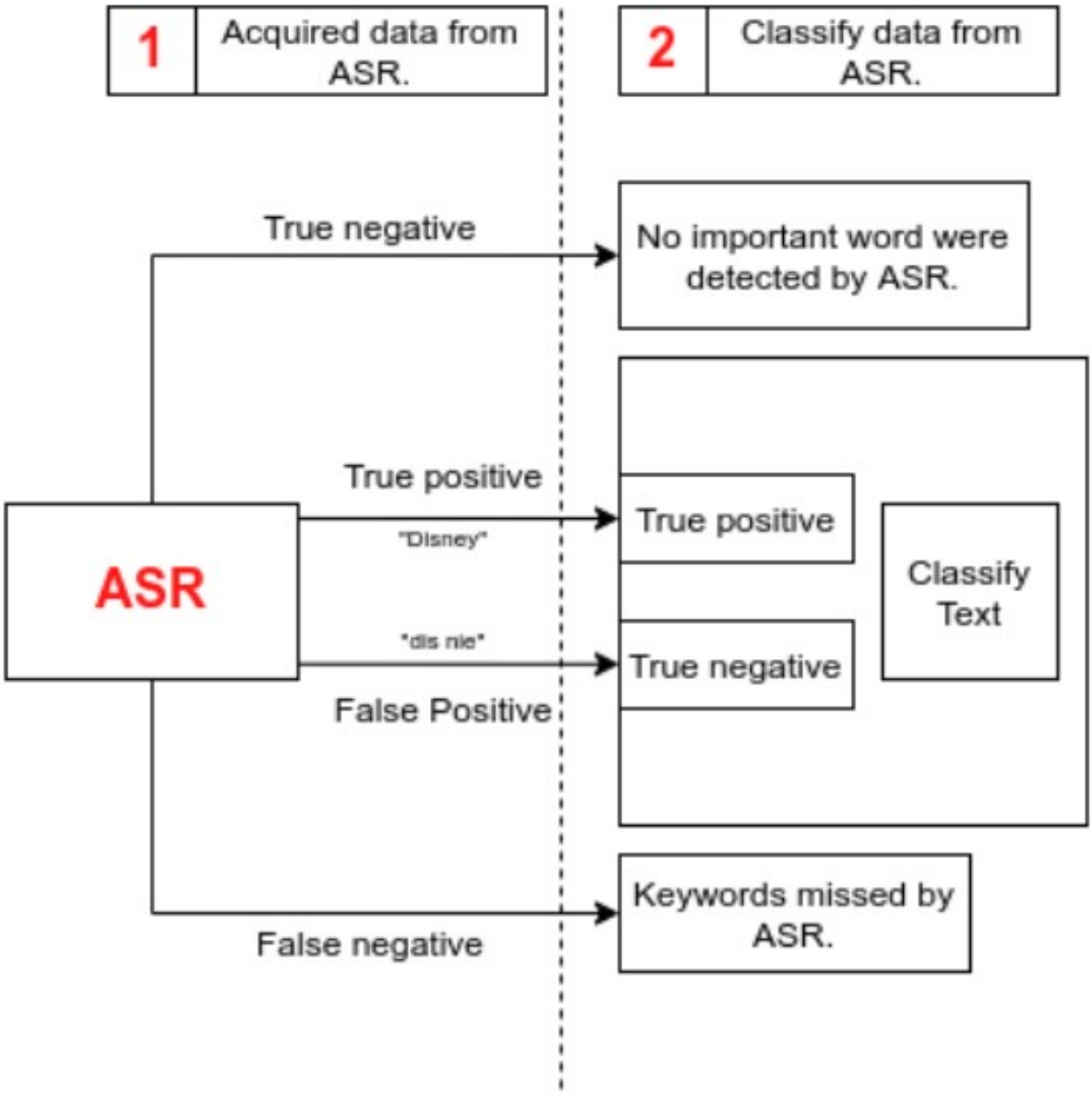


Figure A.1: Flow diagram detailing how the data obtained from the ASR system and classified using speech-text data. Note, the FP that were acquired from the ASR system were labeled by the people working for the Novus Group.

Appendix B

JSON file description

The data is captured in a JSON format, shown in listing B.1, containing the speech-text data (i.e. "word" in list of "words" JSON object) as well as the metadata associated with it — the metadata includes: the language of the speech-text data, date and time the data was captured (specified as broadcast-date), keywords of interest identified in the speech-text data, the broadcast station name, a link to the mp3/mp4 file the data was captured in (specified as the feed-path) and whether the data represents the occurrence of a false positive word or not (reject reason - "0" means true detection, "1" means false detection). The JSON files are processed and restructured as a single CSV file.

B.1 Appendix: Chapter 3

```
{
    "keyword": "+\"Disney\" +\"Thursday\"",
    "details": {
        "keyword": "+\"Disney\" +\"Thursday\"",
        "reject_reason": 0,
        "broadcast_date": "2019-09-26 15:46:00",
        "words": [
        {
                "time": "01:47:00",
                "actual_time": 106.8,
                "word": "disney",
                "duration": 0.39,
                "word_score": 0.83
```

```
            },
            {
                "time": "01:48:00",
                "actual_time": 107.78,
                "word": "include",
                "duration": 0.25,
                "word_score": 0.89
            }
            ],
            "station_detail": {
            "feed_path": "http://vpn.clientA.co.za:10080/mnt/
               broadcast/indexed/20190926/5FM/5FM.2019-09-26
               _15-46-00.mp3",
            "language": "English",
            "station_name": "5FM"
        }

}
}
```

Listing B.1: Example of what the JSON object looks like

Appendix C

Results for Different Classification and Embedding Methods

C.1 Appendix: Chapter 5

C.1.1 Logistic regression classifier

Result obtained using a logistic regression classifier with different embedding methods are shown in Tables B.1-C.4. Note the threshold column indicates the probability threshold at which the various measurements were made on the validation set.

Table C.1: Logistic regression classifier using TF-IDF.

Window	Cohen kappa	AUC	F1-Score	Precision	Recall	Threshold
Dictionary size 60 000						
5	0.403	0.818	0.759	0.801	0.742	0.481
11	0.416	0.832	0.774	0.820	0.757	0.484
21	0.423	0.838	0.780	0.826	0.762	0.487
41	0.433	0.838	0.781	0.825	0.765	0.486
61	0.429	0.834	0.777	0.820	0.761	0.489

121	0.426	0.827	0.771	0.812	0.755	0.490
Dictionary size 80 000						
5	0.403	0.818	0.759	0.802	0.742	0.483
11	0.418	0.833	0.775	0.820	0.757	0.486
21	0.425	0.839	0.781	0.826	0.763	0.489
41	0.436	0.839	0.783	0.825	0.766	0.487
61	0.432	0.836	0.779	0.821	0.762	0.490
121	0.429	0.829	0.772	0.813	0.757	0.492
Dictionary size 100 000						
5	0.403	0.819	0.759	0.802	0.742	0.484
11	0.419	0.834	0.775	0.821	0.758	0.487
21	0.426	0.840	0.782	0.827	0.764	0.490
41	0.437	0.840	0.783	0.826	0.767	0.4889
61	0.434	0.837	0.780	0.821	0.763	0.491
121	0.432	0.830	0.774	0.814	0.758	0.492

Table C.2: Logistic regression classifier using doc2vec DBoW.

Window	Cohen kappa	AUC	F1-Score	Precision	Recall	Threshold
5	0.205	0.686	0.659	0.731	0.632	0.522
11	0.248	0.726	0.692	0.764	0.664	0.517
21	0.276	0.745	0.709	0.779	0.682	0.517
41	0.295	0.753	0.716	0.780	0.691	0.514
61	0.301	0.755	0.716	0.777	0.692	0.511
121	0.306	0.753	0.714	0.771	0.692	0.503

Table C.3: Logistic regression classifier using word2vec SG.

Window	Cohen kappa	AUC	F1-Score	Precision	Recall	Threshold
5	0.317	0.764	0.717	0.772	0.696	0.490
11	0.316	0.769	0.727	0.788	0.703	0.480
21	0.322	0.773	0.733	0.794	0.708	0.482
41	0.322	0.771	0.729	0.789	0.706	0.483
61	0.324	0.769	0.728	0.785	0.705	0.485
121	0.313	0.757	0.717	0.774	0.695	0.489

Table C.4: Logistic regression classifier using fastText SG.

Window	Cohen kappa	AUC	F1-Score	Precision	Recall	Threshold
5	0.316	0.762	0.717	0.772	0.696	0.483
11	0.317	0.770	0.727	0.788	0.703	0.481
21	0.322	0.774	0.733	0.794	0.708	0.482
41	0.325	0.771	0.731	0.790	0.707	0.483
61	0.325	0.771	0.728	0.785	0.705	0.486
121	0.314	0.756	0.717	0.774	0.696	0.490

C.1.2 Average cosine similarity method

Tables C.5-C.8 show the results obtained using the average cosine similarity method with various window sizes. Evaluated on the validation set.

Table C.5: Average cosine similarity method using word2vec SG.

Window	Cohen kappa	AUC	F1-Score	Precision	Recall	Vocabulary size
5	-0.026	0.478	0.298	0.617	0.313	**48 703**
11	0.191	0.587	0.537	0.722	0.504	**78 667**
21	0.257	0.612	0.629	0.727	0.596	**106 148**
41	0.318	0.620	0.676	0.729	0.651	**131 623**
61	0.330	0.641	0.709	0.742	0.690	**143 190**
121	0.322	0.673	0.759	0.767	0.753	**154 873**

Table C.6: Average cosine similarity method using fastText SG.

Window	Cohen kappa	AUC	F1-Score	Precision	Recall	Vocabulary size
5	0.177	0.630	0.604	0.742	0.571	**72 620**
11	0.328	0.702	0.723	0.778	0.708	**118 093**
21	0.364	0.719	0.747	0.789	0.729	**155 421**
41	0.335	0.686	0.750	0.769	0.739	**186 519**
61	0.286	0.660	0.730	0.752	0.717	**198 764**
121	0.247	0.648	0.703	0.742	0.683	**210 024**

Table C.7: Average cosine similarity method using word2vec CBoW.

Window	Cohen kappa	AUC	F1-Score	Precision	Recall	Vocabulary size
5	-0.023	0.478	0.298	0.617	0.313	**48 703**
11	0.109	0.577	0.525	0.715	0.492	**78 667**
21	0.163	0.612	0.629	0.727	0.596	**106 148**
41	0.194	0.620	0.676	0.729	0.651	**131 623**
61	0.241	0.625	0.695	0.732	0.675	**143 190**
121	0.328	0.673	0.751	0.760	0.744	**154 873**

Table C.8: Average cosine similarity method using fastText CBoW.

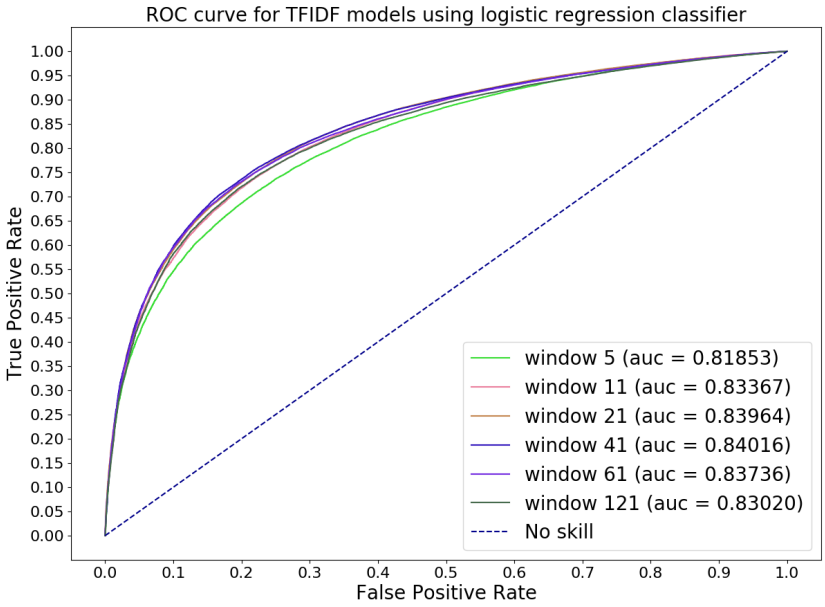
Window	Cohen kappa	AUC	F1-Score	Precision	Recall	Vocabulary size
5	0.102	0.590	0.452	0.746	0.441	**72 620**
11	0.162	0.632	0.545	0.762	0.518	**118 093**
21	0.250	0.633	0.725	0.736	0.716	**155 421**
41	0.229	0.604	0.738	0.730	0.752	**186 519**
61	0.219	0.597	0.737	0.729	0.756	**198 764**
121	0.227	0.597	0.743	0.736	0.768	**210 024**

C.1.3 DNN classifier

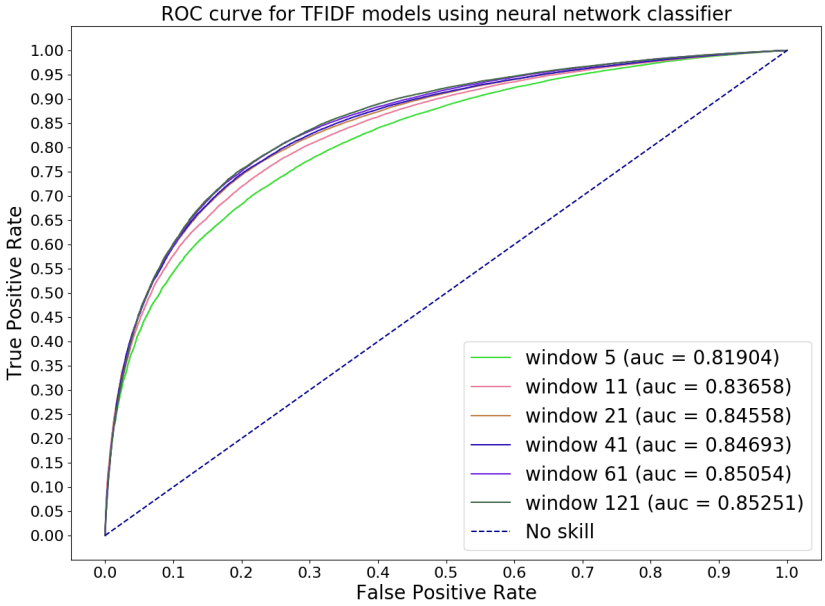
The ROC curves shown in Figure C.1-C.12 were obtained for a DNN with 400 hidden nodes (i.e. DNN-400) and a logistic regression classifier. Note Tables C.9-C.12 indicate results obtained for a 100 node DNN. Evaluated on the validation set.

Table C.9: TF-IDF Feedforward neural network 100 hidden nodes.

Window	Cohen kappa	AUC	F1-Score	Precision	Recall	Threshold
5	0.396	0.816	0.756	0.800	0.739	0.905
11	0.410	0.831	0.772	0.818	0.754	0.918
21	0.426	0.841	0.782	0.827	0.764	0.918
41	0.439	0.843	0.784	0.826	0.768	0.920
61	0.448	0.846	0.786	0.826	0.770	0.915
121	0.457	0.846	0.785	0.822	0.771	0.921



(a) Logistic regression classifier and TF-IDF.

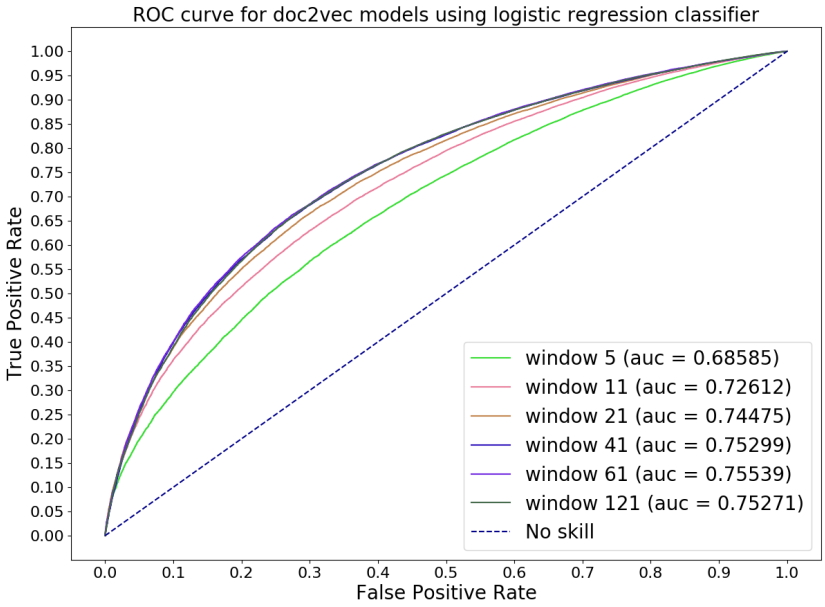


(b) DNN-400 classifier and TF-IDF.

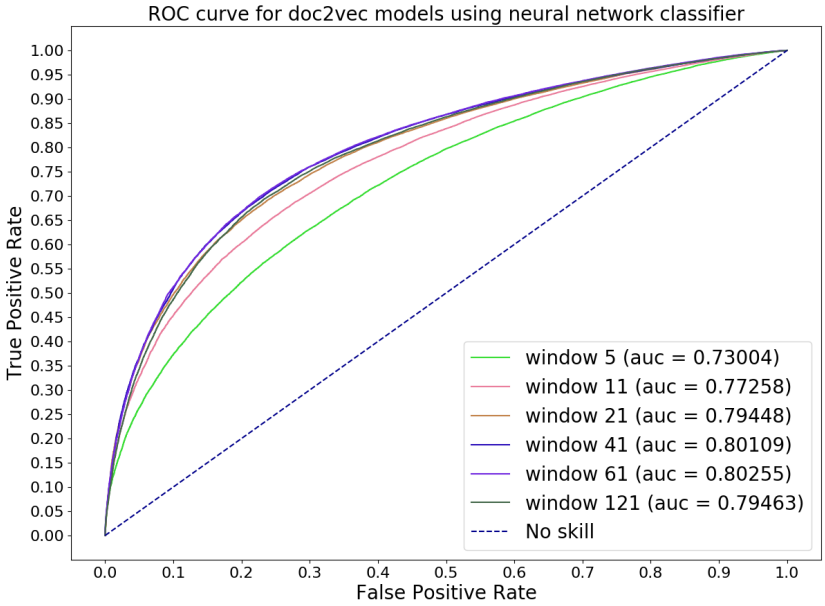
Figure C.1: ROC curves for different classifiers and window sizes using TF-IDF with a dictionary size of 100 000, evaluated on the validation set.

Table C.10: Feedforward neural network classifier with 100 hidden nodes using doc2vec DBoW.

Window	Cohen kappa	AUC	F1-Score	Precision	Recall	Threshold
5	0.287	0.748	0.702	0.761	0.679	0.911
11	0.338	0.789	0.738	0.795	0.715	0.923
21	0.372	0.809	0.757	0.810	0.736	0.929
41	0.392	0.815	0.763	0.812	0.744	0.916
61	0.397	0.817	0.762	0.809	0.744	0.909
121	0.405	0.817	0.761	0.805	0.745	0.891



(a) Logistic regression classifier using doc2vec DBoW.

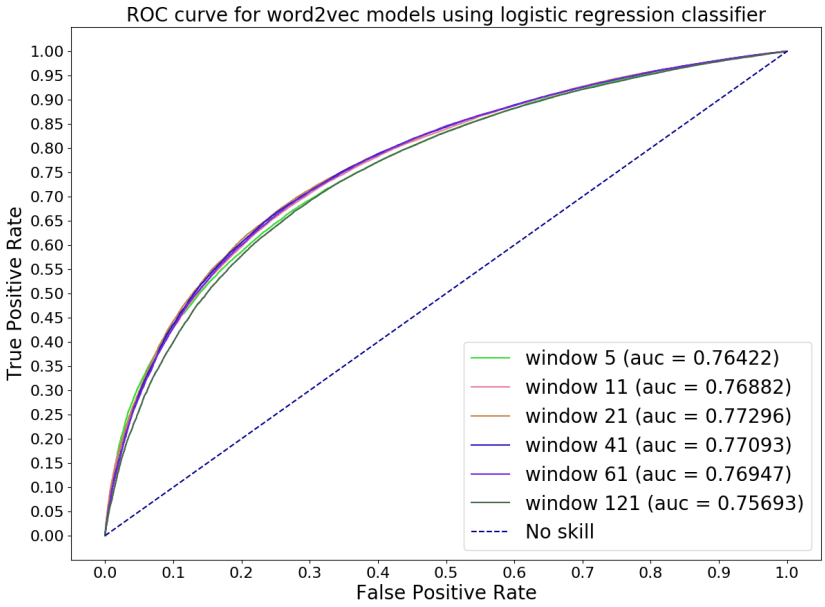


(b) DNN-400 classifier using doc2vec DBoW.

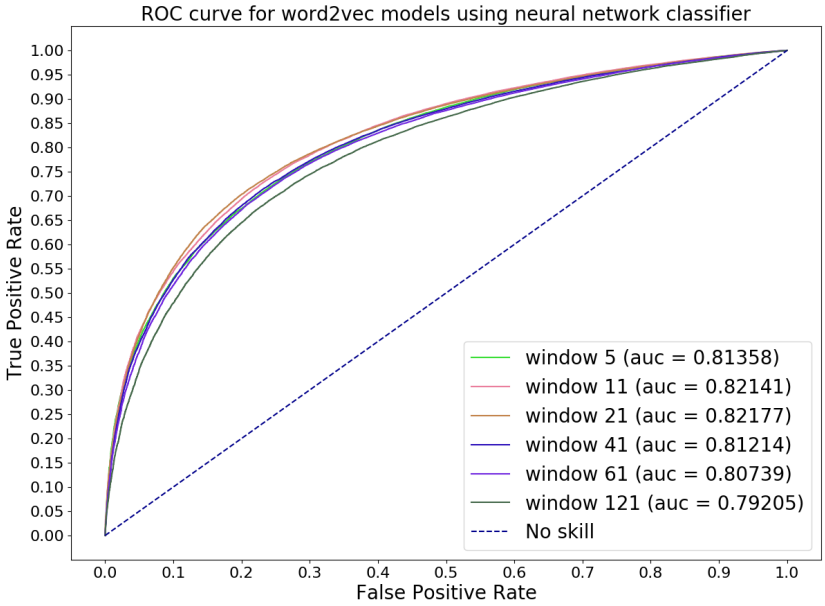
Figure C.2: ROC curves for different classifiers and window sizes using doc2vec, evaluated on the validation set.

Table C.11: Feedforward neural network classifier with 100 hidden nodes using word2vec SG.

Window	Cohen kappa	AUC	F1-Score	Precision	Recall	Threshold
5	0.383	0.808	0.749	0.795	0.732	0.897
11	0.382	0.813	0.759	0.809	0.739	0.922
21	0.383	0.814	0.762	0.813	0.741	0.914
41	0.369	0.802	0.752	0.804	0.732	0.894
61	0.375	0.802	0.752	0.802	0.732	0.912
121	0.351	0.783	0.736	0.787	0.716	0.908



(a) Logistic regression classifier using word2vec SG.

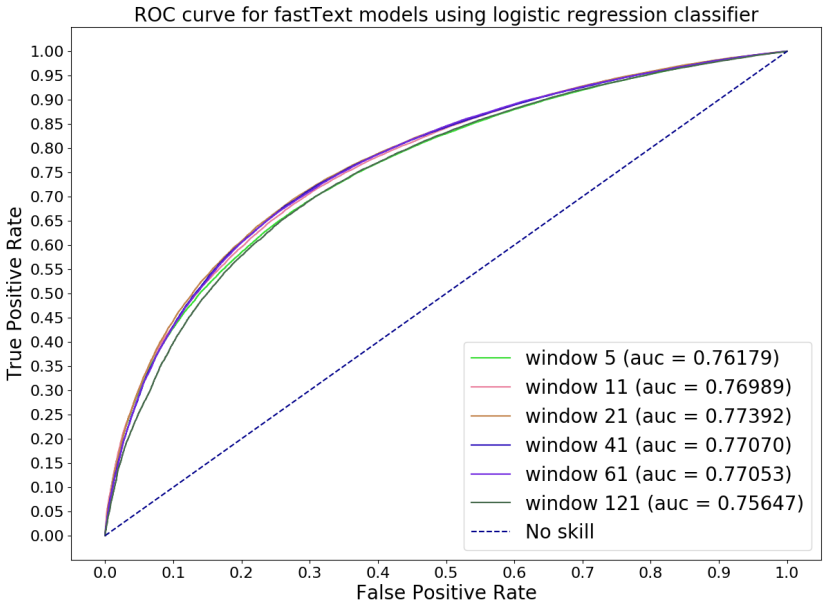


(b) DNN-400 classifier using word2vec SG.

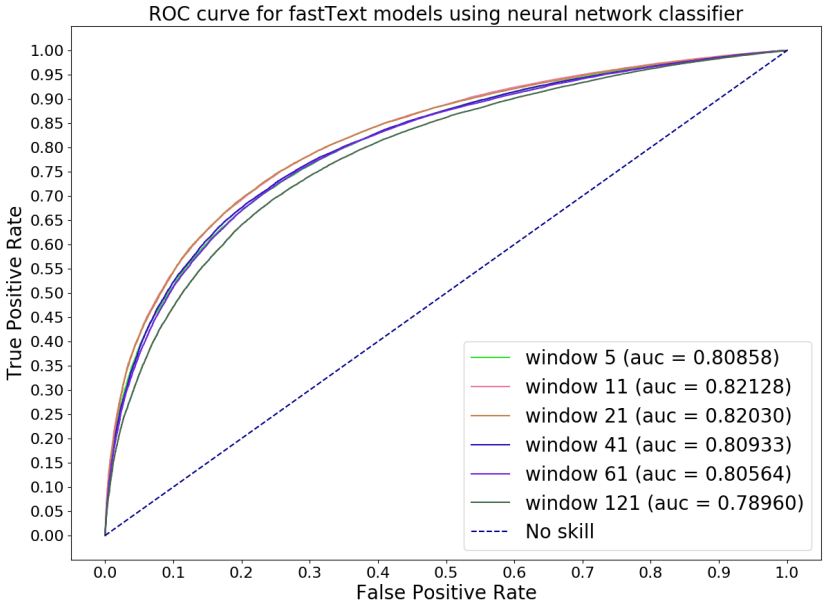
Figure C.3: ROC curves for different classifiers and window sizes using word2vec, evaluated on the validation set.

Table C.12: Feedforward neural network classifier with 100 hidden nodes using fastText SG.

Window	Cohen kappa	AUC	F1-Score	Precision	Recall	Threshold
5	0.384	0.805	0.750	0.795	0.733	0.900
11	0.386	0.816	0.760	0.810	0.741	0.930
21	0.382	0.813	0.761	0.813	0.741	0.911
41	0.376	0.805	0.755	0.807	0.735	0.906
61	0.374	0.800	0.751	0.802	0.732	0.920
121	0.350	0.782	0.735	0.786	0.716	0.917



(a) Logistic regression classifier using fastText SG.



(b) DNN-400 classifier using fastText SG.

Figure C.4: ROC curves for different classifiers and window sizes using fastText, evaluated on the validation set.

C.1.4 DNN classifier with flags

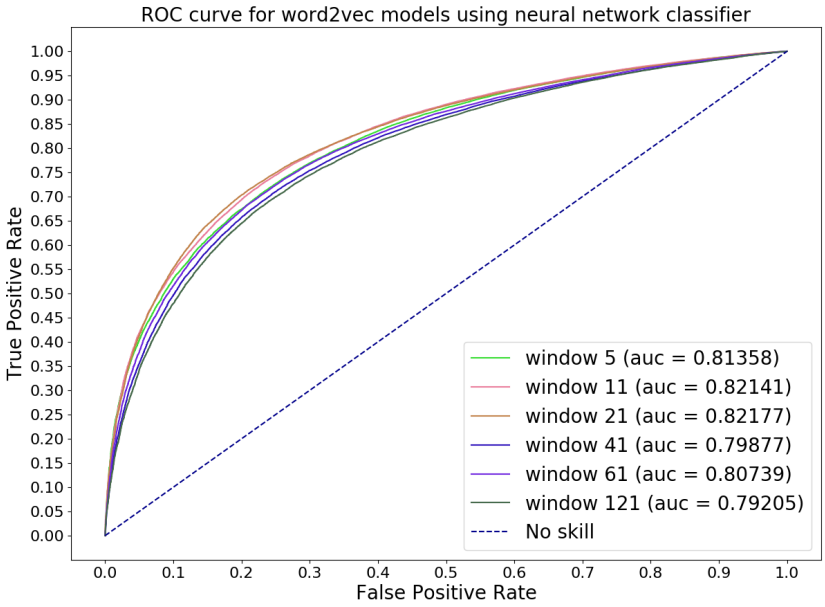
The following Tables C.13-C.14 show the results obtained using a 400 node DNN with and without flags.

Table C.13: Feedforward neural network classifier with 400 hidden nodes using word2vec SG.

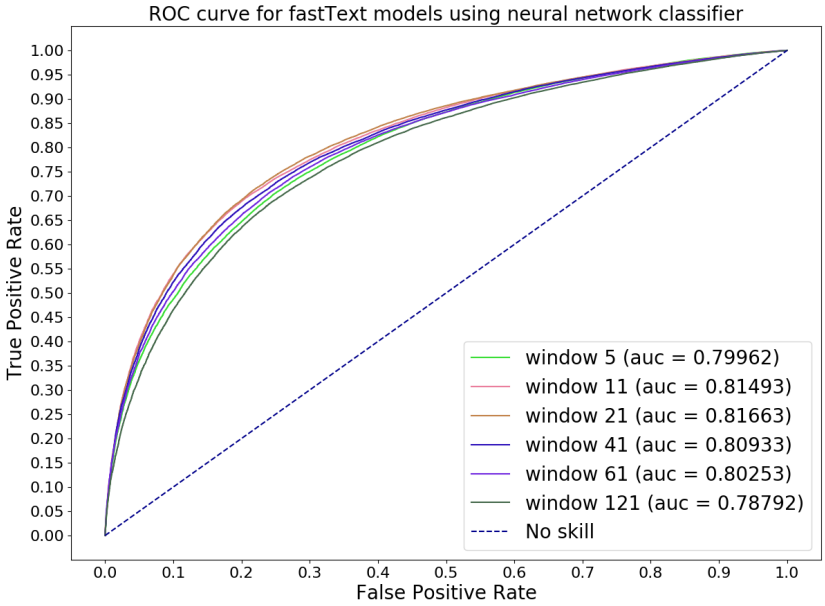
	With flags		**Without flags**	
Window	**Cohen kappa val**	**Cohen kappa test**	**Cohen kappa val**	**Cohen kappa test**
5	0.344	0.340	0.394	0.396
11	0.373	0.374	0.395	0.393
21	**0.385**	**0.381**	**0.398**	**0.394**
41	0.375	0.379	0.382	0.384
61	0.369	0.373	0.382	0.376
121	0.353	0.359	0.364	0.361

Table C.14: Feedforward neural network classifier with 400 hidden nodes using fastText SG.

	With flags		**Without flags**	
Window	**Cohen kappa val**	**Cohen kappa test**	**Cohen kappa val**	**Cohen kappa test**
5	0.372	0.372	0.390	0.383
11	0.400	0.402	**0.399**	**0.396**
21	**0.407**	**0.404**	0.395	0.392
41	0.388	0.388	0.383	0.383
61	0.380	0.382	0.382	0.373
121	0.358	0.363	0.362	0.361



(a) DNN-400 classifier and word2vec SG.



(b) DNN-400 classifier and fastText SG.

Figure C.5: ROC curves for different classifiers and window sizes using word2vec and fastText with flags, evaluated on the validation set.

C.2 Appendix: Chapter 6

The following results were obtained using a bi-directional LSTM using various sequence lengths based on window size. The hyperparmeters shown in Table C.15 were used to see if reserving the sequence (i.e. both left and right sequence in text) of words in a sentence could better classify falsely mentioned words. The results obtained, shown in Table C.16-C.17 and Figure C.6, indicates similar performance to that of 400 node feedforward neural network using TF-IDF, which reserves no word ordering. Considering this fact using TF-IDF with a LSTM was not investigated.

Table C.15: Bi-LSTM network hyperparameters.

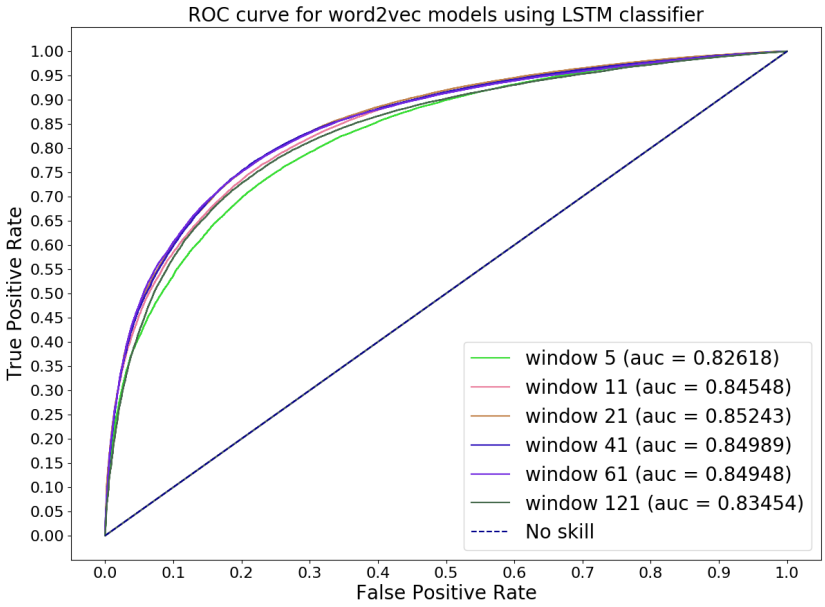
Parameter	**Value**
Embedding dimension	300
Hidden layers	128
Depth of network	2
Fully connected layer	128
Batch size	128
Learning rate	0.001
Dropout	0.2
Epochs trained	5
Weighted cross-entropy	True

Table C.16: Bi-directional LSTM classifier using word2vec SG.

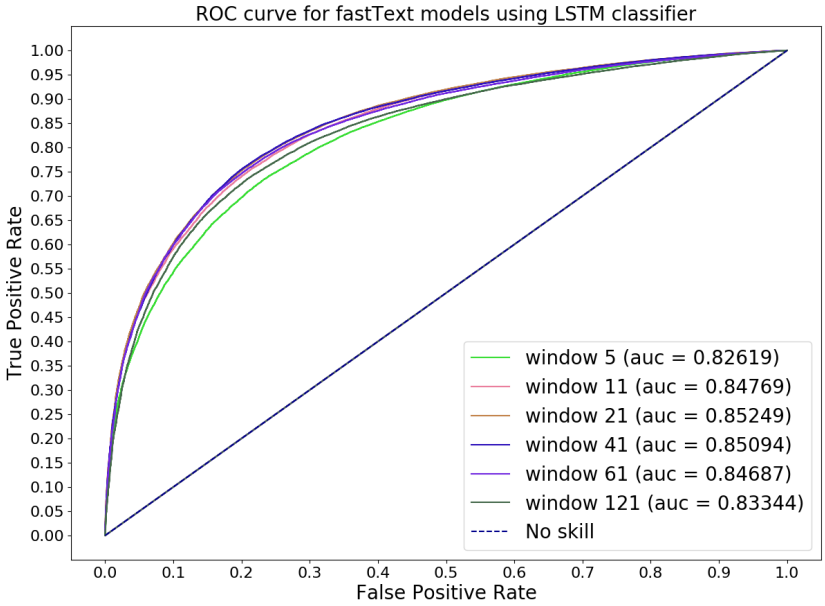
Window	**Cohen kappa validation set**	**Cohen kappa test set**
5	0.418	0.415
11	0.439	0.436
21	0.450	0.447
41	0.456	**0.453**
61	**0.458**	0.450
121	0.436	0.432

Table C.17: Bi-directional LSTM classifier using fastText SG.

Window	**Cohen kappa validation set**	**Cohen kappa test set**
5	0.416	0.415
11	0.442	0.437
21	0.450	0.449
41	**0.457**	**0.454**
61	0.453	0.447
121	0.437	0.437



(a) DNN-400 classifier and word2vec



(b) ROC curves evaluated on test set.

Figure C.6: ROC curves for an LSTM classifier using different window sizes considering word2vec and fastText embeddings, evaluated on the validation set.