Recognising and Disambiguating Place Names in Text Documents

SUPERVISOR – CHRIS JONES

Ellie Robins | Individual Project | June 2020

Abstract

Named Entity Recognition is required for recognising and disambiguating place names in a text document. Named Entity Recognition is a branch of Natural Language Processing which labels the known entities in a piece of text. There are many tools already available that can achieve NER.

In this project, I have used Named Entity Recognition tools to achieve the task of recognising place names and evaluated them by calculating their precision, recall and F1 values. I have trained baseline classifier on specific training data to allow them to perform NER. The NER tools and Classifiers are compared to see which the best options for Named Entity Recognition are.

Acknowledgments

I would like to thank my supervisor Chris Jones for his continued support and guidance throughout the project.

Table of Contents

Table of Figures

1.0 Introduction

1.1 Project overview

Natural Language Processing (NLP) is a field of linguistics and machine learning which allows computers to analyse, read and manipulate human written data and text. Named Entity Recognition (NER) is a branch of NLP which takes the individual entities of a text document and assigns a tag to that entity. It can be an essential task for social media applications such as Facebook for promotional advertisement, applications such as google maps to give the most suitable options for searches and other applications that could use customer interactions to improve the experience of the user. Most branches of NLP are challenging due to the richness and ambiguity of natural language.

My project is an information retrieval project that is concerned with developing machine learning methods to perform this task. Geo-references are a common place task in GIS that involves associating information with same location in physical space which can take different forms of postal addresses, place names, post codes and coordinates. This location should only refer to one place which is typically is the case with given reference. In a document identifying references to location helps understand its geographical context. Geoparsing will include identifying geo-references which will use named entity recognition which is the process of assigning words or groups of words to a set of predefined classes or categories such as locations, person, organisation etc. Most named entity recognition algorithms exploit lists of known locations. These lists are called gazetteers. The task can be challenging because of the difficulty of distinguishing genuine place names from other terms, such as the names of people and organisations, and because some place names (such as Newport) are ambiguous due to different places having the same name. There are various approaches to geoparsing which include simple list lookup which is simple fast and language independent, knowledge or rule-based methods which uses surrounding context to capture geographical context. Machine learning approaches to geoparsing employ types of evidence that a name is a place name based on whether it occurs in a gazetteer or if it is preceded by spatial relations such as 'near' or 'within' and whether it is a particular instance of a vernacular place name such as 'Big Apple' is associated to New York City, USA.

Looking at current NER's being used will give a good idea of how the algorithms work and how they perform. This project will see the efficient working classifiers and where they have opportunities to improve as well as using machine learning libraries such as Scikit-Learn to develop a classifier and how that can be done and how to improve a custom written NER.

1.2 Aims and objectives

The first aim of the project is the use the current standards for NER to detect a place name in a text document. Objectives of this aim will include research and understanding machine learning algorithms as having a greater understanding of how machine learning work and how the algorithms are trained will help use the NER algorithms to their best ability and further on train a classifier from scratch.

Objectives:

- Research machine learning on a whole to ensure that there is a clear understanding while starting the project
- Understand and download the relevant tools and libraries required while working with machine learning
- Research the standard NER algorithms used currently
- Apply them and use them to detect entities in text documents

Secondly, aiming to evaluate current NER tools by getting accuracy measurements of overall accuracy and the furthering on to precision, recall and F1 value.

Objectives:

- Evaluate the NER tools for accuracy using overall accuracy, precision, recall and F1 values.
- Evaluate the NER tools for speed

The next aim is to use a machine learning library to create a classifier that predicts if the word is a location or not in a corpus. The same accuracy measurements that are used to evaluate the current standards of NER can be used to evaluate the created classifier.

Objectives:

- Research machine learning libraries that can be used to create classifier (e.g. Scikit Learn) and how to create a classifier using those libraries
- Research pre-trained embeddings downloads and how they are used
- Create a classifier using pre-trained embeddings
- Evaluate the classifier using different embeddings, different dimensions of embeddings, different training data and different classifiers

2.0 Background

This section focuses on the background research of the tools that will be used to detect place names in a text document. Machine learning explains how supervised and unsupervised algorithms are trained. The following background research of NLP and classifiers will give a helpful explanation of how the software tools in named entity recognition are used and how classifiers can be trained as a way of improving performance of the standard tools.

2.1 Machine Learning

Machine learning is an application of artificial intelligence which gives systems the ability to learn and improve from experience without being explicitly programmed. It focuses on developing computer programs that can access data and use it to learn. The aim of machine learning is to allow computers to learn automatically without assistance of human intervention. [Expert System Team, 2017]

There are 6 stages of machine learning which include data collection, data storing, data analysis, algorithm development, checking algorithm generated and use of the algorithm to further conclusions. To look for patterns various algorithms are used which are divided into unsupervised and supervised learning. Unsupervised learning is where the machine receives only a set of input data, there after the machine is set up to determine the relationship between entered data and any other hypothetical data. Unsupervised machine learning implies that the computer itself will find any relationships and patterns between data sets. Supervised machine learning implies computer ability to recognise elements based on samples provided to the machine. The computer studies elements and develops the ability to recognise new data based on the previous data, examples of supervised machine learning algorithms include decision trees and linear regression. [Kgarkovyna, 2019]

Machine learning often works with high dimensional data which is best represented by matrices. Therefore, mathematical analysis derivatives and gradients will be needed for optimisation problems. The main data analysis libraries are NumPy, Pandas, Scikit, SciPy, iPython and matplotlib. [Kgarkovyna, 2019]

2.2 Natural Language Processing

Natural language processing is a specialized area of study of linguistics, computer science and artificial intelligence. It is concerned with interactions between computers and human languages especially how to program machines to process and analyse large amounts of human language data. The study of natural language processing started around the 1950s although work has been found from earlier periods. [Bates, M (1995)]

Natural language processing facilitates conversations between machines and humans, it holds all the systems that allows a machine to handle interactions with a human's language. [Goyal, Kumar, Gupta- October 2017] There are various applications of named entity recognition which include question answering systems which try to find exact answers to the natural language questions in a large document. A system with speech tagging, dependency parsing and NER system can improve the speed and accuracy of getting correct answers. Information extraction systems can include NER systems as recognising named entities in any documents can help retrieve the correct information requested by the user. NER systems can be used in opinion mining which is identifying and extracting subjective information in source materials, therefore used in social media for marketing. [IGI-Global] Other applications that refer to the location are how social media advertise local businesses or for hotels in a location that is spoken about. A new tool developed called GeoTxt, the app uses NER techniques which finds proper nouns and associates them with given places or objects. [Karimzadeh, M., Pezanowski, S., MacEachren, A. M., & Wallgrün, J. O. (2019)]

2.3 Speech Tagging

In NLP Speech tagging is used to identify a word in a corpus to a corresponding part of speech established on context and its definition. Part of speech tagging cannot have a generic mapping as the same word can have several different part of speech tags in different sentences based on different context. Speech tagging is not necessarily useful in solving specific NLP tasks however it proves helpful in simplifying other problems. [Malhotra, Godayal, 2018] When words have speech tags it could help NER algorithms to determine its entity, for example if a word is identified as an adjective it would be easier for the NER to realise the word cannot be a location and could help determine the word surrounding the adjective e.g. 'Melbourne is a beautiful city'. The adjective 'beautiful' is used to describe the location 'Melbourne' a couple of words before it.

2.4 Dependency Parsing

Dependency parsing is the task of assigning a syntactic structure to a sentence. The syntactic structure represents its grammatical structure and defines the relationship between head words and the other words. [Kadam, 2019] Dependency parsing can provide the structure and relationship between the words as well as the subjects and objects of a verb.

2.5 Named Entity Recognition

Named entity recognition, referred to as NER, takes unstructured texts extracts information that label named entity in the texts. The texts are labelled from pre-defined categories such as location, person, organisation, percentage etc. [Technopedia, 2015] Many of the NER systems that have been developed take an unannotated text block and produces annotations on them of the tags that have been identified and located. Named entity recognition can also be referred to as entity identification or extraction. An example of this would look like this:

Taking an unannotated block of text:

Simon Cowell bought a house in New York City in 2020. He operates his businesses under Sony.

And producing annotation such as this:

Simon Cowell [person] bought a house in New York City [location] in 2020 [time]. He operates his businesses under Sony [organisation]

Named entity recognition has a key role in many natural language processing applications such as machine translation and question answering. The most popular and recognised named entity recognition platforms, systems or algorithms are Stanford NER, NLTK and SpaCy.

Stanford NER is a Java implementation of a named entity recogniser which comes with a well-engineered feature extractor for NER and includes different options to define features of the extractor. It is known as a CRF Classifier which is a conditional random field classifier. The software gives a general implementation of a linear chain CRF sequence model. The code from Stanford NER can be used to build a sequence model for NER by training your own models labelled data. [Jenny Rose Finkel, Trond Gregnagr and Christopher Manning. 2005]

NLTK, natural language tool kit, is a platform for building python programs to work with human language data. NLTK has a module that is used for interfacing with Stanford tags. It uses the download of the tagger models and then uses the StanfordNERTagger class to write python program using the Stanford NER. NLTK has been described as 'a wonderful tool for teaching, and working in, computational linguistics using Python'. [2001-2019 NLTK Project. Nitin MAdnani, Rami Al-Rfou]

SpaCy is a free open source library for natural language processing in python. It focuses on providing software for production usage and features name entity recognition. [Honnibal, 2015]

Firstly, Stanford named entity recognition can be implemented as a way of detecting named entities in text documents.

2.6 Software Tools

2.6.1 Stanford Natural Language Processing Group

The group is a group of employees at the Stanford University, researchers studying a post-graduate, hired programmers and students studying at the university who work together to create, develop and improve algorithms that allow machines to process and understand human language. Their work can include a range between research in computational linguistics and key applications in human language technology. The work of natural language can be training the machine for sentence understanding, automatic question answering, translation between human and machine language, syntactic parsing and tagging and models of text and visual scenes. Syntactic parsing and tagging are where Stanford named entity recogniser would fall under. The Stanford NLP Group have a recognisable feature which is their effective combination of sophisticated and depth linguistic modelling and data analysis with machine learning and deep learning approaches to natural language processing. Known technologies developed by the group are coreference resolution system. The NLP Group at Stanford includes members of the Linguistics and Computer Science departments and is part of the Stanford AI Lab.

2.6.2 Stanford Named Entity Recognition

Stanford NER is a developed named entity recognition which is a Java implementation. It labels words in text that are names of organisations, persons and locations. Stanford NER has different classes class 3, 4 and 7 which labels sequences of a words in a text which are for class 3 names of organisations, persons and locations, for class 4 locations, persons, organisations and miscellaneous entities and for class 7 locations, organisations, persons, dates, times, percentages and money. Stanford NER makes other models available for different languages and circumstances which includes models trained on the CoNLL 2003 English training data. Stanford NER is known as a CRF Classifier. CRF represents conditional random field and a CRF classifier are a class of statistical modelling applied in machine learning and pattern recognition. It can consider context while labelling samples. Stanford NER software provides general implementation of a linear chain CRF sequence models by users training their own models on labelled data. The software for Stanford NER is available for download and licensed under the GNU General Public License. The software packable includes components for command line use, running it as a server and a Java API. [Jenny Rose Finkel, Trond Gregnagr and Christopher Manning. 2005]

When implementing Stanford NER, the NLTK Stanford interface can be implemented to be able to use Stanford NER while writing the program in python. The 'nltk.tag.stanford' which is an interface to the Stanford Part-of-Speech and Named-Entity Taggers can be used as long as certain pre-conditions are met. These pre-conditions are ensuring Stanford tagger models are downloaded, which can be downloaded from the Stanford NER website, and that the Stanford model environment variables are set.

The Stanford NLTK Tagger can be implemented which is a class for the named entity tagging task using the Stanford. When referencing the Stanford tagger models the class of the model is required which will change what named entities are recognised, the most common class is class 3 which only identifies people, organisation and locations. The Stanford NLTK Tagger is imported from the natural language tool kit Stanford module, it includes the Stanford POS Tagger as well and consists of the tag function. The tag function applies the most appropriate tag to the given token and returns a list of the corresponding tagged tokens. A Python function open() can be used to read a text document and used the Stanford NLTK tagger to tag the tokens in the text document which returns the list of the tagged tokens from the text documents.

2.6.3 NLTK

Natural language tool kit is an important platform for building Python programs to work with human language data. It provides interfaces to large structured texts and language resources as well as text processing libraries for classification, tokenisation's, tagging, parsing and semantic reasoning. NLTK has a guide introducing programming fundamentals along with topics in computational linguistics. NLTK provides an API documentation making it suitable for people of all programming levels. NLTK can be used to tag text, identify entities in text and display a parse tree which is a rooted tree that represents the syntactic structure of strings in the text. [2001-2019 NLTK Project. Nitin MAdnani, Rami Al-Rfou]

NLTK can use POS tag to tag tokens in text. The POS tagger processes a sequence of words and tags an appropriate tag to the word token.

2.6.4 SpaCy

SpaCy is a python written open-source software library for advanced natural language processing. It is an industrial strength natural language processing tool in python, which unlike NLTK, is used more for production uses not just teaching and research. It is designed for production use and will assist the programmer to build applications that process and understand volumes of text. SpaCy can be used to build information extraction as well as natural language understanding systems which can include machine and human translation. It provides various linguistic annotations to understand texts grammatical structure for example how words are related to each other and word types. [Honnibal, 2015]

SpaCy can tokenize text during processing, which splits the text into words, punctuation etc. which is done by applying rules specific to each language. Although punctuation rules are usually generalised, but a tokenizer exception depends strongly on specifics of an individual language so each available language on SpaCy has its own subclass that includes lists of hard-codes data and exception rules. Once tokenization is completed during processing, SpaCy can parse and tag a given text document. SpaCy can then make a prediction of which tag or label most likely applies using a statistical model. The statistical model is produced by showing a system enough examples for it to be able to make a prediction of which label applies in this context. SpaCy takes on the job of named entity recognition by being able to recognise various types of named entities in a document by asking the model for a prediction. Like many other NER's models work on the examples they are trained on so does not work perfectly. [2020 Explosion AI]

2.6.5 SpaCy Model and Language

SpaCy has models which can be installed as Python packages which allows them to be a component of an application like any other module. SpaCy provides supports for many languages. There is a language class which is created when spacy.load() is called which includes a shared vocabulary and language data.

2.6.7 SpaCy Named Entity Recogniser

The natural language processor library is assigned to loading the language model using the SpaCy load option. The document is then assigned to nlp library which runs the named entity recogniser on the text. Named entities are available as the 'ents' of a 'doc' in SpaCy. The entities have text - which is the entity itself, the entities label which could be:

Type of Entity	Description
PERSON	People (including fictional)
NORP	Nationalities, Religions or Political Groups
FAC	Buildings, Airports, Highways
ORG	Companies, Agencies, Institutions
GPE	Countries, Cities, States, Counties
LOC	Non-GPE locations – bodies of water, mountains
PRODUCT	Objects, vehicles, foods

Figure 1: Entity Labels for SpaCy

The above entities are the relevant labels that will be focused on during this project. The full list can be found at https://spacy.io/api/annotation#named-entities.

The output of SpaCy's named entity recogniser differs from Stanford and NLTK.

The following text:

New Zealand is a sovereign island country in the southwestern Pacific Ocean. The country has two main landmasses the North Island, and the South Island and around 6OO smaller islands. It has a total land area of 268,OOO square kilometres. New Zealand is about 2,OOO kilometres east of Australia across the Tasman Sea and 1,OOO kilometres south of the Pacific island areas of New Caledonia, Fiji, and Tonga. Because of its remoteness, it was the last large habitable landmass to be settled by humans. During its long period of isolation, New Zealand developed a distinct biodiversity of animal, fungal, and plant life. The country's varied topography and its sharp mountain peaks, such as the Southern Alps, owe much to the tectonic uplift of land and volcanic eruptions. New Zealand's capital city is Wellington, and its most populous city is Auckland.

Gives the following output when running the code for SpaCy's NER on the command line:

```
two CARDINAL

the South LOC

6OO CARDINAL
```

268,OOO CARDINAL

Australia GPE

1,OOO QUANTITY

Fiji GPE

Tonga PERSON

Many entities in the text documents were not identified therefore training on the model will be required.

2.6.8 Comparison of NLTK, Stanford NLP and SpaCy

The three natural language processing technologies NLTK, Stanford NLP and SpaCy are all well known for their natural language processing tools. NLTK and SpaCy are written in Python and Stanford NLP is written in Java but there are various tools written to use Stanford NLP in Python. Neural network model's functionality is offered by SpaCy and Stanford NLP but not NLTK. Integrated word vectors and entity linking are provided only by SpaCy and not NLTK or Stanford NLP. Integrated word vectors, multi-language support, tokenization, part-of-speech tagging, sentence segmentation and entity recognition functionalities are provided by all three. Dependency parsing are not provided by NLTK. Coreference resolution is only provided by Stanford NLP. [Explosion AI (2), 2020] Using these examples as functionalities SpaCy offers the most in comparison to Stanford NLP and NLTK.

2.7 Classifiers

Sci-kit learn have various amounts of classifiers that can be used for an NER. Many linear models that include classifiers that can be used are Logistic regression, LinearSVC, SGD and SVM. SVM's are support vector machines which are a set of supervised learning methods for classification, SVM includes SVC, NuSVC and LinearSVC. SVM is effective in high dimensional space which will be beneficial when using pretrained embeddings to get the embeddings of a word when the dimension of the embeddings can go up to 300. Like other classifiers the SVM classifiers take in two arrays one of shape [n_samples, n_features] featuring the training samples and the other array is of shape [n_samples] which include the class labels.

LinearSVC is a class capable of performing multi-class classification on a dataset, it is like SVC with the kernel parameter set to linear. It is implemented in terms of liblinear (open source machine learning library) which gives it more flexibility in choice of penalties and loss functions, and it should scale better to larger number of samples. SGD Classifier can optimise the same cost functions as LinearSVC by adjusting the penalty and loss parameters. SGD also requires less memory to LinearSVC. All classifiers from sci-kit learn have a fit function that fit the classifier with the given training data and returns an instance of the estimator. After the classifier has been fitted the instance of the estimator can be used to predict test data which in this case would be word embeddings. [2019, Scikit learn]

2.8 Scikit Learn

Scikit Learn is a software machine learning library written in Python. It has efficient tools for predictive data analysis and is built on matplotlib, Numpy and SciPy. It features various classification, regression and clustering algorithms including scale vector machines. Scikit learn provides numerous tutorials to working with text classification and other machine learning problems. [2011, Pedregosa et al]

2.9 Embeddings

A word embedding in machine learning is a taught representation for a text where the words have the same meaning will have a similar representation naturally capturing their meaning. One of the key breakthroughs of deep learning on natural language processing issues is the approach to representing words and documents. Word embeddings are a class of techniques which represent individual words as real-valued vectors in a predefined vector space. Each word is mapped to one vector and the values of the vector are learned in a way that resembles a neural network. The idea of using a dense distributed representation for each word is key to the approach. The individual words can be represented as high dimensional data and reduced to low dimensions of the embeddings. Thousands or millions of dimensions would be required for sparse word representations. The distributed representation is learned based on the usage of words. Meaning words used in a similar way will have a similar representation and naturally capture their meaning.

Word embedding methods learn a real-valued vector representation for a predefined fixed sized vocabulary from large text. The learning process is either joint with the neural network model on some task or it is an unsupervised process using the documents statistics.

Using word embeddings start with learning embeddings before using them. For each individual problem, case or project a word embedding can be learned which requires large amount of text data to ensure useful embeddings are taught. The large text data can be millions or billions of words. When training the word embeddings, it can be taught standalone where the model is trained to learn the embedding and used as a part of another model. It can also be learned as part of a large task specific model. Researchers make pre-trained word embeddings available such as word2vec and GloVe which can be used instead of training your own embeddings from scratch. To use pre-trained embeddings the options are static where the embeddings are kept as static and used as a component of the model. Or the updated option which is pre-trained embeddings to use the increase the model, but the embedding is updated while training the model. [Jason

Brownlee, 2017] The embeddings can be used to extract an embedding of a word from training data and then using it to be used as features for a classifier.

2.9.1 Taking N Number of Words Either Side of Word to be Classified

When a classifier is trying to classify a word in a NER it could be beneficial to understand the rest of the sentence to ensure the classification is correct. For example if a sentence was "I am ____ big apple" the blank word could be 'eating' or 'in the' which will help a human and a classifier understand what the bigram 'big apple' is referring too. This demonstrates how getting the embeddings of the words either side of the word to be classifier can improve the performance of a classifier.

2.10 Bag of words

Bag of words is a method of extracting features from text documents which can be used for a machine learning classifier. It creates a vocabulary of the unique words that occur in the text document in the training data. It creates the vocabulary first by cleaning the text, tokenizing, building the vocabulary and generating the vectors. Cleaning and tokenizing text can be task specific, but the usual steps can include removing punctuation and removing stop words. Stop words tend to be words with less than a certain several characters e.g. 'the', 'a', 'and'. Some limitations to the bag of words method is that it does not consider the meaning of a word in the document and ignores the context in which it is used. Also, the vector size can be large in terms of time and computation for large text documents which may require the need of ignoring the words irrelevant to the case. [Free Code Camp, 2018]

2.11 Accuracy, Precision, Recall and F1 Values

A classifier could possibly produce the following confusion matrix:

		Predicted/Classified	
		Negative	Positive
Actual	Negative	998	0
	Positive	1	1

Figure 2: Confusion Matrix

Below demonstrates where true and false positives and negatives occur in a confusion matrix:

		Predicted/Classified	
		Negative	Positive
Actual	Negative	True Negative	False Positive
	Positive	False Negative	True Positive

Figure 3: Confusion Matrix showing True/False Positives and Negatives

Accuracy refers to how close something is to a known value. Accuracy is total number of correctly identified tags divided by the total number of tags in the data:

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

Figure 4: Accuracy Equation

NLTK metrics scores that is used to import accuracy is a module from NLTK metrics package which imports different types of functions which can give different measurements and values that are required.[NLTK Metrics 2019] The accuracy function passes a list of reference values and a corresponding list of the values predicted by the classifier as parameters and returns the fraction of which values are equal therefore which have been correctly predicted. The accuracy takes in this fraction was multiplied by 100 to become a percentage rather than the original fraction produced by the accuracy function.

From the table of a possible classifier the results would imply the classifier has an accuracy of 99.9% which would seem remarkable. Unfortunately, the mis-classified actual positive which was predicted as a negative could in some cases be a fraud case classified as a genuine case or a terrorist predicted as a non-terrorist. Thankfully in the case of detecting place names the consequences would not be as dire, but it demonstrates that the accuracy of a model is not the be all and end all metric to measure a classifier.

Instead precision and recall measurements would be better in getting a clearer idea of how accurate a classifier is performing. Precision is calculated by dividing the true positives by the total predicted positives, showing you how accurate a classifier out of the predicted positives.

$$Precision = \frac{True\ Positives}{False\ Positives + True\ Positives}$$

Figure 5: Precision Equation

This is a preferred measurement when there are high costs associated with false positives such as spam emails as an important email predicted as a positive for spam could be missed.

Recall is the calculation of dividing the true positives by the total actual positives therefore, including the true positives and false negatives.

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

Figure 6: Recall Equation

Recall would be a better measurement to use if the cost of false negatives were high should as detecting fraudulent in a bank as if something is wrongly predicted as a negative it could result in theft.

Finally, F1 value is a function of both precision and recall. It is calculated as shown in the below equation.

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Figure 7: F1 Value Equation

As accuracy can be largely contributed by the number of true negatives it can give a false sense of reliability when the important positives wrongly predicted are overlooked. F1 is helpful as it provides a balance between precision and recall and is a better measurement when there is a large uneven class distribution such as many actual negatives. [Shung, 2018]

2.12 Data Sources

As the project focus is on detecting location names in a text document, I chose to use plenty of data sources that come from Wikipedia pages of countries or regions of a country. These Wikipedia contents are on the following countries:

- New Zealand
- South East England
- Cwmbran
- United States
- Australia
- Iceland
- Finland
- South Wales
- Southern France
- Fiji
- Philippines

The text documents of the extracted contents include various locations and organisations and are approximately 200 words.

Kaggle is an online community of data scientists and machine learning practitioners where datasets are available to use and complete tasks with. A dataset can be downloaded specifically for an NER task which includes the entity label and POS tag. The dataset is large, so it was trimmed down to 5000 words. Kaggle's dataset includes various tags such as ‘art’ which represents an artifact, so the dataset was edited to only include other, location, person, and organisation.

The New Zealand locality data is around 1,100 lines of words. The locality data comes from Landcare New Zealand (which can be found at https://www.landcareresearch.co.nz/). It contains descriptions of the position or size of sites in New Zealand which can include addresses, highways, roads, schools, and farms for example. The sites are where biological samples are found.

Every word, punctuation and digits are labelled in all datasets. If the word is a location a tag ‘I-LOC’ will be attached after the word, if the word is a person ‘I-PER’ tag will be attached, ‘I-ORG’ for organisation and then the ‘O’ label represents ‘other’ meaning it has no named entity.

3.0 Approach

This section includes how I will approach the issue of detecting a place name in a text document. Using the current NER tools to detect a place name in a document and then comparing their accuracy measurements. After evaluating their measurements SpaCy can be trained to improve the model’s performance. This section will also see the difference in using the Wikipedia page contents of locations to human written notes of the New Zealand locality data. The comparisons can be compared with speed to get a full performance measurement of the NER’s. Finally, this section will discuss moving on from using current NER tools to creating classifiers and how to optimise them.

3.1 Accuracy of the NER Tools

If the NER’s are not very accurate compared the human annotated data then it would need to be trained by using annotated data to be able to understand and learn from examples of what are locations, place names and organisations to use in future cases of different text.

3.1.1 Testing Stanford NER for Accuracy

It can be clear to see that Stanford NER works on text documents as it labels the text, but it can often predict the incorrect label. To evaluate the NER it is useful to get a calculation of accuracy which can be used compare the NER to others. If the accuracy value is incredibly low, then it is suggested the NER predicts many labels incorrectly and may require more training. If the accuracy value is relatively high but slow it may not be useful to use therefore it is beneficial to get the value to use in conjunction with other performance measurements such as speed to see which is the best NER.

3.1.2 Measuring NLTK for Accuracy

NLTK was evaluated for accuracy to see its performance in comparison to Stanford NER. Similarly, to measuring Stanford NER for accuracy the same annotated data was used for fairness to measure NLTK for accuracy this can be compared to the accuracy results from NLTK and Stanford NER. [NLTK metrics, 2019]

3.1.3 Precision, Recall and F1 values for Stanford NER and NLTK

In the previous measurement for accuracy on Stanford NER and NLTK the annotations on the data include many 'O' labels which would mean there is a large number of true negatives predicting the 'O' labels increasing the accuracy percentage when there may be many false positives or false negative going unnoticed which could be critical for the use of the classifier.

The recall value will be more important than the precision function, false negatives have a much higher cost with determining locations in texts. If locations are not identified when they should be it can prevent a different classifier from disambiguating the correct place name. This could be an issue for advertising companies who want to release the correct adverts in the correct locations or for map applications it can give the wrong directions or suggestions. Calculating the precision and F1 value is also useful even if may not be as important as the recall value they provide more accuracy results which can be used to compare the NER's.

3.1.4 Accuracy Measurements for SpaCy

To make the accuracy measurements of SpaCy named entity recogniser the input data and annotations need to be in the same form as training data. The training data include the sentence to be tagged in quotation marks and then the entities in curly brackets. The entities contain two numbers and the entities name in quotation marks, the numbers represent the start character index and the end character index of the entity so SpaCy understands where the entity is in the sentence.

Training Data Example:

TRAIN_DATA = [

("c150m SW of old factory, Vaughan Rd", {"entities": [(24,34, "FAC")]}),

("550mN from Bush rd-Gladfied rd intersect", {"entities": [(11,18, "FAC")]}),

("P.N. - Himatangi Main Highway 0.75 miles East of junction with Jackytown Rd.", {"entities": [(7,29, "FAC"), (64, 76, "FAC")]})]

As it requires the index of the start character and end character of the entity and then the label it will take numerous number of hours to annotate every line in the data sources especially as the New Zealand locality data is 1,500 lines.

3.1.5 Training SpaCy

SpaCy has models which are statistical and every decision they make is a prediction. These decisions can be deciding if a word is a named entity. The predictions are based on previous examples the model has been exposed to during training. Training a model requires training data which can be examples of text and labels that the model needs to predict. The training data could be named entity and any other information.

The model will then evaluate the unlabelled text and make a prediction from the previous training data. As humans understand if the model has given the wrong label, the model can receive feedback in form of an error gradient of the loss function which then calculates the difference between the training example text and the expected output. If the difference between the test output and the expected output is greater it means the updates to the model will be greater. [Explosion AI, 2016-2020]

[https://spacy.io/usage/spacy-101]

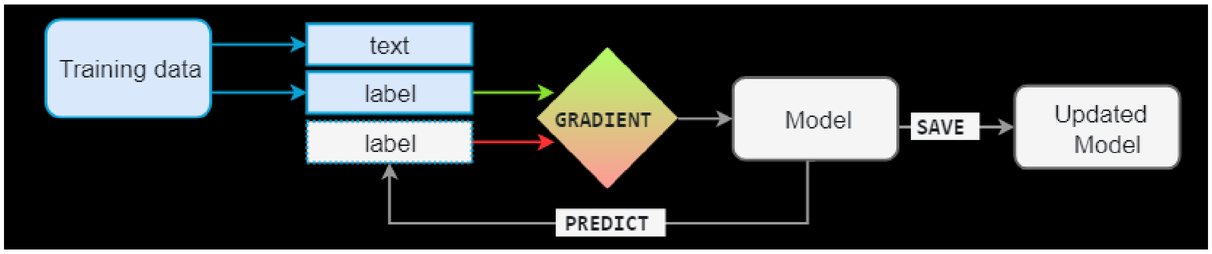

Figure 8: Training SpaCy Diagram

When training a model, it is important that the model does not want to just memorise the examples but come up with a theory that can be used across other examples. The model should not learn one instance of a word, such as 'Apple' which in some texts can be a fruit or others can be the company. Which is why the training data should always be representative of the types of data that would be processed such as if a model was trained on Wikipedia where sentences are not in first person would perform badly on Twitter where most text is from the first person.

Evaluation data is also required as well as training data in order to know how the model is performing and if it is learning the correct information from the training data. With only the training data there will be no way of knowing how well it is generalising. To update an existing model decent results can be produced with few examples if they are representative.

To update SpaCy's named entity recogniser with examples for a specific application, starting with an existing pretrained model will require example texts and the character offsets and then labels each of the entities which are contained in the texts. Firstly, the model required can be loaded or an empty model can be created with spacy.blank() with the identification of the language being used. With a blank model the entity recogniser needs to be added to the pipeline. An existing model will require all other pipeline components to be disabled during training using nlp.disable.pipes to ensure only the entity recogniser is being trained. Secondly the examples need to be shuffled and looped over. Update the model which can be achieved by calling nlp.update which steps through the words of the input. The model makes a prediction for every word which is then compared to the annotations with the correct labels to see if the prediction is correct. If the prediction is incorrect the model will adjust its weights so that the correct result will score higher next time. The trained model can be saved using nlp.to_disk and then finally the model should be tested to make sure the entities in the training data are recognised correctly. [Explosion AI, 2016-2020 (4)]

Using the same text documents that contain short information on various locations to train the model, using examples of text documents with numerous locations will be beneficial to training the model for detecting locations to achieve the project aim. It is important not to repeatedly iterate over the same examples as the model cannot get used to evaluating any other examples.

Before model training the SpaCy named entity recogniser the following entities were identified:

Australia GPE

Commonwealth ORG

Australia GPE

Australian NORP

Oceania GPE

sixth ORDINAL

26 million CARDINAL

Australia GPE

Adelaide PRODUCT

Australia GPE

7,617,930 square kilometres QUANTITY

the north-east LOC

the south-east LOC

2.8 CARDINAL

Australia GPE

After model training the SpaCy named entity recogniser the following entities were identified:

Australia GPE

Commonwealth ORG

Australia GPE

Australian NORP

Tasmania GPE

Oceania GPE

sixth ORDINAL

26 million CARDINAL

Australia GPE

Canberra GPE

Sydney GPE

Melbourne GPE

Brisbane GPE

Perth GPE

Adelaide GPE

Australia GPE

7,617,930 square kilometres QUANTITY

the north-east LOC

the south-east LOC

2.8 CARDINAL

Australia GPE

3.2 Using New Zealand Data

To measure the accuracy and speed of the named entity recognisers and have an overall evaluation of their performance locality data for New Zealand is used. The New Zealand data is written in note form with many words shortened so it would be beneficial to gather performance measurements of the NER's on this data to see if they work as well without the text being eloquently written like in the other datasets.

3.3 Speed of the NER Tools

To gain an overall understanding of Stanford NER evaluating its speed is a convenient way to see its benefits, drawbacks and ultimately if the accuracy of the named entity recogniser outweighs any issues with speed.

3.4 Moving on from Current NER Tools

3.4.1 Developing a Classifier

As an alternative to current NER tools a classifier can be developed using machine learning libraries. I will be writing a new classifier to try and improve performance and the accuracy measurements.

Firstly, to begin training an NER classifier training and test data will be necessary, as in all machine learning classifiers. Training data will allow the classifier to learn a function by taking an input that is mapped to an output, in the case of detecting locations the input would an array of word embeddings, and the mapped output would be the location or other tags. If the data being used for training is a text document, such as the locality New Zealand data discussed above, where data is annotated as below:

Ruatangata I-LOC West I-LOC , O on O roadside O , O 0.7 O km O along O Worsnops I-LOC Road I-LOC from O Ruatangata I-LOC , O east O side O of O road O .

The text document can be read through using any programming language and then the words and tags can be separated into different lists. Although the words need to be changed to embeddings either using pretrained embeddings or vectorization the tags will remain in plain English tags but will need to transform to an array. Sci-kit learn module model selection have a test train split class which takes in the embeddings and the array of tags and split it into train and test data under the same conditions that will be able to be fitted and predicted by the chosen classifier.

3.5 Bag of Words vs Embeddings

This section will discuss the ways of getting pre-trained embeddings for features when training a classifier as well as how to get the features using the bag of words method. It will also discuss the difference between the both and which would be better for a NER classifier.

3.5.1 GloVe

A word embeddings algorithm called GloVe (The Global Vectors for Word Representation) is an unsupervised learning algorithm to obtain vector representations of words developed by Pennington, et al. at Stanford. The algorithm constructs an explicit word context or word co-occurrence matrix using statistics across the whole large text document. The result of the word co-occurrence matrix is a learned model which generally results in better embeddings. [Jason Brownlee, 2017] Text documents can be downloaded from GloVe that have 50, 100, 200 and 300-dimension pre-trained embeddings. The embeddings used were trained under: Wikipedia 2014 + Gigaword 5 (6B tokens, 400K vocab, uncased, 50d, 100d, 200d, & 300d vectors, 822 MB download): glove.6B.zip.

3.5.2 Vectorizing

If pretrained embeddings are not available, the classifier can be trained from scratch using word vectors which are created by classes such as Count Vectorizer. Scikit Learn library has a feature extraction class which is implemented to extract features to be used in a classifier. Feature extraction has a text section which includes Count Vectorizer that focuses on extracting features for a classifier from text. Count Vectorizer takes a collection of text documents and coverts them to word vectors which are represented using a compressed sparse row matrix. Count Vectorizer includes text pre-processing, tokenization and filters stop words and then creates a dictionary of features from the text document and transforms the features to vectors. This demonstrates a bag of words representation where the frequency of a word is used as a feature in classifier training. These vectors are the word vectors that can be used to train a classifier.

3.5.3 Comparison of Bag of Words vs Embeddings

Both embeddings and bag of words extract features from text documents to use them in training a classifier. Pre-trained embeddings have many benefits which include reduced training time and need for pre-processing text data. It can be trained on large datasets which the user of these pre-trained embeddings may not have access to and therefore help improve the performance of a classifier. Using a vectorizer like Count Vectorizer from Scikit Learn will transform words to word vectors and use them to train the classifier and allow it to predict from other test data. As the words are only transformed to word vectors and do not carry the benefits of pre-trained embeddings trained on large datasets with numerous dimensions it would be expected that training a classifier using pre-trained embeddings would perform better. [2020 – Pai] However as using Scikit-Learn only requires a small amount of code to create the model therefore a baseline model may benefit from a bag of words representation instead of pre-trained embeddings. If datasets are small context is most likely to be domain specific therefore the corresponding vector from pre-trained embeddings cannot be found. [2018, Ma]

The higher the dimension the longer the vector is therefore it will give more data for the machine learning classifier to take in and learn from. As the longer the vector and higher dimension of embeddings the classifier has a better idea of how to predict the word embedding. It would be expected for the higher the dimension of the pre-trained embeddings used to train the classifier then the better the classifier will perform.

3.5.4 Improving a Baseline Classifier

To get the best results of a classifier and to know how to use them to the best of their ability it will be beneficial to get results of various variables. These variables will include the type of classifier, the training data, the dimension of the embeddings and the n number of words either side of the word to be classified. While comparing the only factor to change will be the variable that is being evaluated and all other information taken into the classifier will remain constant. When comparing the training data or dimensions of embeddings, the input of the classifier will be New Zealand locality data and GloVe's pretrained embeddings of 200 dimensions.

While using classifiers SGD and Linear SVC are remarkably similar minus the cost of extra memory therefore it would be expected that the classifiers would perform similarly. Although smaller datasets are not recommended for supervised machine learning classifiers they can produce decent results on accuracy by the fact the small input dataset are split using Scikit Learn's train test split class which means the test data will be even smaller than the original annotated dataset. In this case the text file 'Fiji.txt', as used previous in measuring accuracy of Stanford NER and NLTK, will be used as an example of a small dataset to train the classifier. The text document has many re-occurring words which may impact the accuracy results as it is trying to predict a word it already has training on. Other datasets have been used to compare the small 'Fiji.txt' dataset with a larger dataset and then another even greater dataset. The middle sized dataset 'NER_dataset.txt.' is around 6000 words from the Kaggle NER dataset [which can be found at https://www.kaggle.com/abhinavwalia95/entity-annotated-corpus] and the largest dataset is the New Zealand locality data which is around 24,000 words. Larger datasets would be expected to perform better when loaded and used for models predicting unseen data as the more examples a machine learning classifier can get the more it can learn from the examples and have a better chance of predicted entities correctly.

3.5.5 Averaging Embeddings

When n number of embeddings are taken from the left and right as well as the embedding of the word to be classified it will increase the number of data values that will be used for training. A way of avoiding using such a high number of data values is the embeddings could be averaging the embeddings so there is only one vector of the same size as the dimensions of embeddings being used for training.

As there will now be a smaller number of data values used for training, this could impact the accuracy measurements. The classifier will see less data values when using the average in comparison to use all embeddings of the word and the surrounding words. Averaging could essentially be hiding some embeddings that would determine if a word is a location or not i.e. if there is a word 'near' left to the word to be classified it could show it is a location but with the average the embedding for 'near' would not be specifically shown.

When the classifier uses just the embedding of the word, the number of data values will be the same as when using this average. Although vectors are the same size, the average should improve the classifier's performance as the average does take into account the surround words.

4.0 Implementation

This section will include how NER was implemented and how the classifiers can be created.

4.1 NER Tools

To begin implementation of the NER algorithms relevant data sources need to be obtained to be able to test them. Stanford, NLTK and SpaCy can all be implemented in Python.

The prerequisites that are required for Stanford, NLTK and SpaCy to be used are:

- Stanford classification model and jar files need to be downloaded from the Stanford Natural Language Processing Group
- NLTK needs to be installed via Python
- SpaCy needs to be installed via Python

For Stanford NER to work in Python, a Stanford NER Tagger needs to be imported using NLTK. The Stanford NER Tagger has the classification model path and the Stanford tagger jar file path passed through as parameters. It has a tag function which will attach the relevant label to the word in the text file.

NLTK can be used to tag words in a text file, firstly the NLTK POS-tag is used to give part of speech tags to the text and then ne_chunk function passes through the words (with POS-tags attached to them) to identify named entities.

SpaCy loads models and then uses the model to tag the text which can accessed by printing the text and the labels of the entities.

4.1.1 Annotating Data for Stanford and NLTK

To estimate if Stanford NER and NLTK NER is accurate, data were annotated with whether they fell under the category for organisations, locations or persons which are the three entities that class 3 of the Stanford NER recognise. After each token there is a space and then the label for the entity ‘I-LOC’ for location, ‘I-PER’ for person and ‘I-ORG’ for an organisation and any of the tokens that do not fall under this category are labelled as O.

An example of the annotated data in a text document is below, the data is taken from the Wikipedia page for the United States of America, that can be found at: https://en.wikipedia.org/wiki/United_States. Only the first two paragraphs from the Wikipedia page was annotated, as expected the ‘O’ tag was the most commonly occurring tag with 137 of tags, the location tag occurred 21 times, person tag with 2 occurrences and finally the organisation tag occurred 16 many times. How the data is annotated is demonstrated below:

The O United I-LOC States I-LOC of I-LOC America I-LOC , O commonly O known O as O the O United I-LOC States I-LOC or O America I-LOC , O is O a O country O consisting O of O 50 O states O , O a O federal O district O , O five O major O self-governing O territories O , O and O various O possessions O . O At O 3.8 O million O square O miles O , O it O is O the O world's O third O or O fourth O - O largest O country O by O total O area O and O is O slightly O smaller O than O the O entire O continent O of O Europe I-LOC . O Most O of O the O country O is O located O in O central I-LOC North I-LOC America I-LOC between O Canada I-LOC and O Mexico I-LOC . O With O an O estimated O population O of O over O 328 O million O , O the O U.S. I-LOC is O the O third O most O populous O country O in O the O world O . O The O capital O is O Washington I-LOC , O D.C. I-LOC , O and O the O most O populous O city O is O New I-LOC York I-LOC City I-LOC . O

The O United I-LOC States I-LOC is O a O federal O republic O and O a O representative O democracy O . O It O is O a O founding O member O of O the O United I-ORG Nations I-ORG , O World I-ORG Bank I-ORG , O International I-ORG Monetary I-ORG Fund I-ORG , O Organization I-ORG of I-ORG American I-ORG States I-ORG , O NATO I-ORG , O and O other O international O organizations O . O It O is O a O permanent O member O of O the O United I-ORG Nations I-ORG Security I-ORG Council I-ORG . O Its O President O is O Donald I-PER Trump I-PER

Once the file is read and split the resulting list includes every word and label as values. It can be indexed using every other item starting from the first word to get the words and every other item starting for the first label to get the labels. The list of words would be returned as:

‘The', 'United', 'States', 'of', 'America', ',', 'commonly', 'known', 'as', 'the', 'United', 'States', 'or', 'America', ',', 'is', 'a', 'country', 'consisting', 'of', '50', 'states', ',', 'a', 'federal', 'district', ',', 'five', 'major', 'self-governing', 'territories', ',', 'and', 'various', 'possessions', '.', 'At', '3.8', 'million', 'square', 'miles', ',', 'it', 'is', 'the', "world's", 'third', 'or', 'fourth', '-', 'largest', 'country', 'by', 'total', 'area', 'and', 'is', 'slightly', 'smaller', 'than', 'the', 'entire', 'continent', 'of', 'Europe', '.', 'Most', 'of', 'the', 'country', 'is', 'located', 'in', 'central', 'North', 'America', 'between', 'Canada', 'and', 'Mexico', '.', 'With', 'an', 'estimated', 'population', 'of', 'over', '328', 'million', ',', 'the',

'U.S.', 'is', 'the', 'third', 'most', 'populous', 'country', 'in', 'the', 'world', '.', 'The', 'capital', 'is', 'Washington', ',', 'D.C.', ',', 'and', 'the', 'most', 'populous', 'city', 'is', 'New', 'York', 'City', '.', 'The', 'United', 'States', 'is', 'a', 'federal', 'republic', 'and', 'a', 'representative', 'democracy', '.', 'It', 'is', 'a', 'founding', 'member', 'of', 'the', 'United', 'Nations', ',', 'World', 'Bank', ',', 'International', 'Monetary', 'Fund', ',', 'Organization', 'of', 'American', 'States', ',', 'NATO', ',', 'and', 'other', 'international', 'organizations', '.', 'It', 'is', 'a', 'permanent', 'member', 'of', 'the', 'United', 'Nations', 'Security', 'Council', '.', "It's", 'President', 'is', 'Donald', 'Trump'

Data was annotated as it is easy for a human to recognise if a word is a name, an organisation or a place name which can be used to test if the machine is working correctly in being able to detect the correct place names or not.

4.1.2 Accuracy of Stanford and NLTK

The implementation of extracting an overall accuracy value for Stanford and NLTK can be demonstrated in the flow chart. The flow chart below demonstrates the stages of the Python script written to calculate the accuracy value.

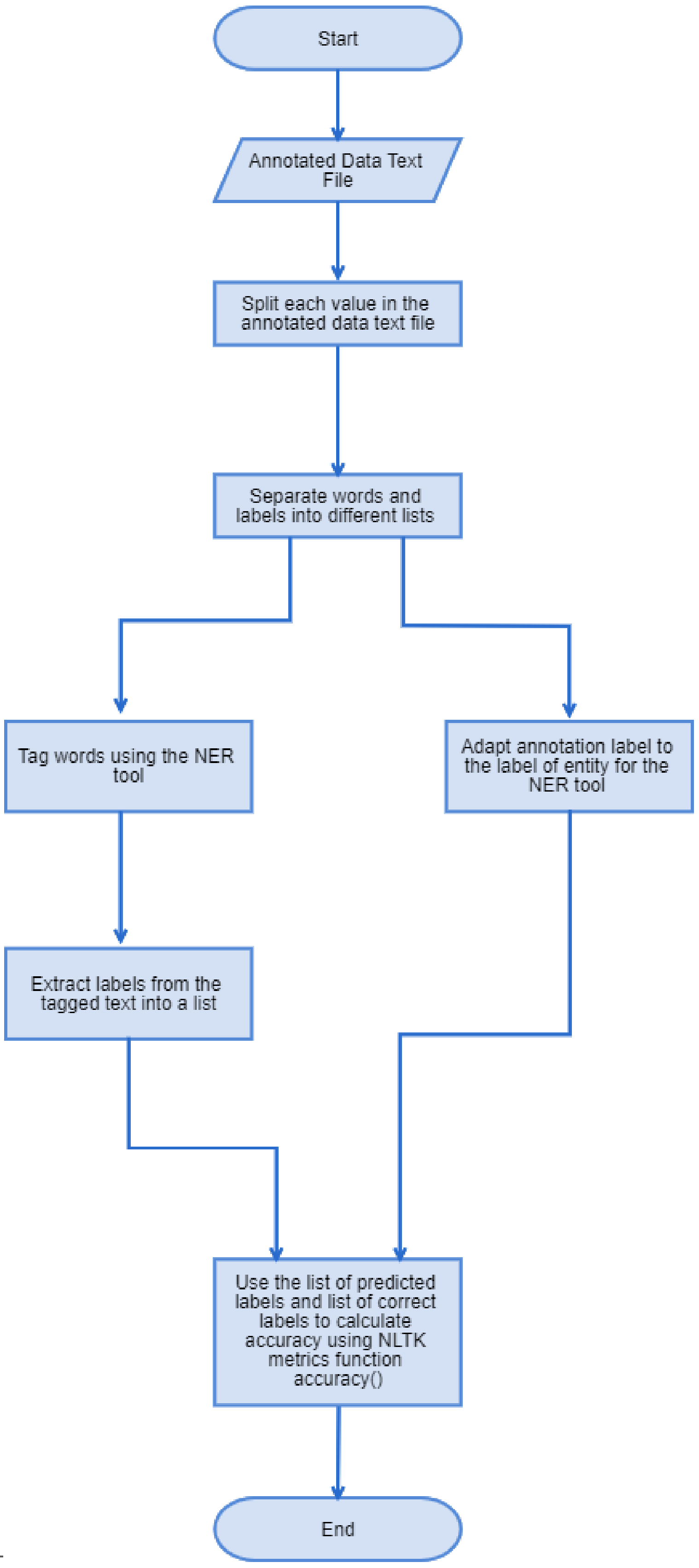
Start
Annotated Data Text File
Split each value in the annotated data text file
Separate words and labels into different lists
Tag words using the NER tool
Adapt annotation label to the label of entity for the NER tool
Extract labels from the tagged text into a list
Use the list of predicted labels and list of correct labels to calculate accuracy using NLTK metrics function accuracy()
End

Figure 9: Flow Chart for Accuracy

To be able to run the Python script written to calculate the accuracy value the following pre-requisites need to be imported:

- NLTK
- Stanford NER Tagger from NLTK library module Tag
- Accuracy from NLTK library module Metrics

The python program script takes in an annotated text document which is used to test the NER tool. The text document is inputted using python's open function and reads the file using read function. All values in the file are split using the split function and then two new lists are created by extracting the words and labels by indexing the resulting list of the split function.

The list of words is then labelled using Stanford NER tagger or NLTK. The Stanford NER tagger must be created and have its models, jar files and encoding established. The Stanford NER uses its tag function to label the text. The text is iterated through to extract the predicted labels into a resulting list. NLTK will use its pos_tag function to assign part of speech tags to the text, then use the chunk function to group into noun phrases. The NLTK tree2conllstr function is used to convert predictions to multiline strings and then to lists. The part of speech tags is then removed through indexing of the lists and the NLTK labels are then transformed to mirror the labels from the input file. The list is iterated through to extract the predicted labels into a list.

The list of labels extracted from the input data is iterated so that the name of the labels are changed to correspond the labels of the Stanford NER tool.

The NLTK accuracy function from the metrics class is used to calculate the accuracy of the NER tools. The lists of predicted labels and correct labels are compared in the function to gather a fraction of how many predicted labels are correct. After the accuracy function result, the fraction is multiplied by 100 to get a percentage.

4.1.3 Precision, Recall and F1 values of NLTK, Stanford and SpaCy

Precision, Recall and F1 values were calculated differently for Stanford and NER than they were for SpaCy.

Precision, Recall and F1 values can be calculated for Stanford and NLTK using Scikit Learn's classification report function. The flow chart below demonstrates how the classification report is calculated:

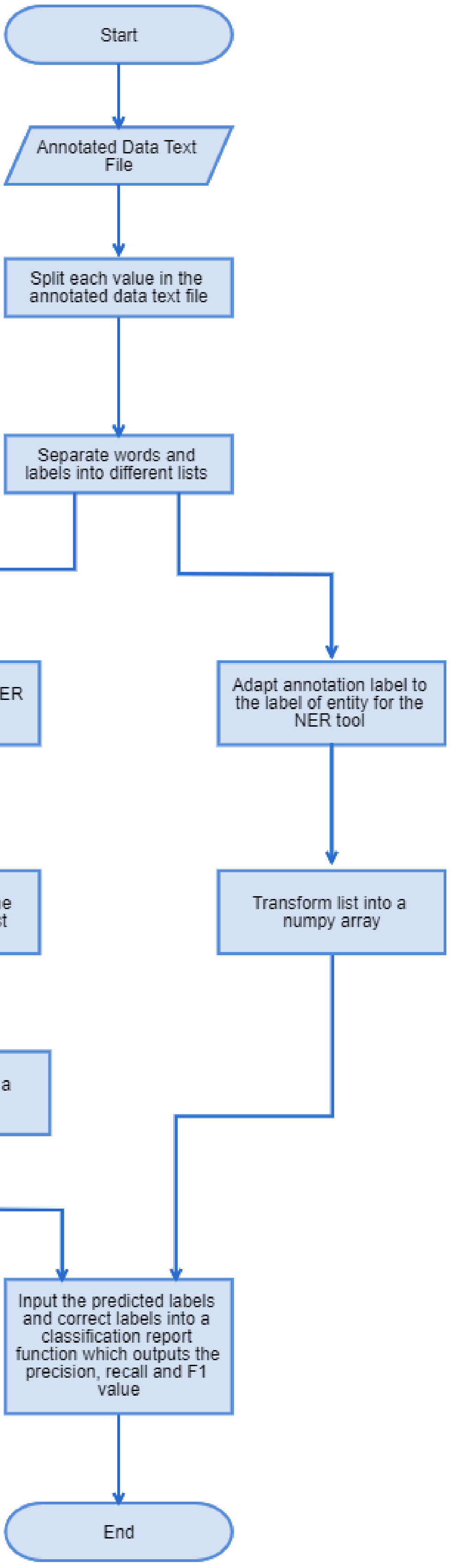
Start
Annotated Data Text File
Split each value in the annotated data text file
Separate words and labels into different lists
Tag words using the NER tool
Adapt annotation label to the label of entity for the NER tool
Extract labels from the tagged text into a list
Transform list into a numpy array
Transform list into a numpy array
Input the predicted labels and correct labels into a classification report function which outputs the precision, recall and F1 value
End

Figure 10: Flow Chart for Precision, Recall and F1 for Stanford and NLTK

For the Python script to run to be able to compute the classification report the following pre-requisites are required to be imported:

- Classification Report from Sklearn library Metrics module
- NLTK
- Stanford NER Tagger from NLTK library Tag module
- NumPy

The same process for opening the annotated text file and producing 2 lists of the words and labels as explained in 4.1.2 is carried out.

In the same way as when calculating accuracy, the Stanford NER Tagger and NLTK (part-of-speech tagger, chunk function and tree2conllstr functions) are used to tag the words. The predicted labels are transformed to match the correct labels which will be used for comparison. The labelled text is iterated through and the predicted labels are extracted into a list. This list is then converted to a NumPy array using the array function from NumPy library.

The list of labels from the annotated text are transformed to match the predicted labels and then converted to a NumPy array using the array function from the NumPy library.

The classification report requires an input of arrays. The predicted labels and the correct labels array are passed through the classification report function to output the report. The report includes precision, recall and F1 values.

SpaCy requires a different method to extract precision, recall and F1 values. The flow chart below demonstrates how SpaCy extracts these values:

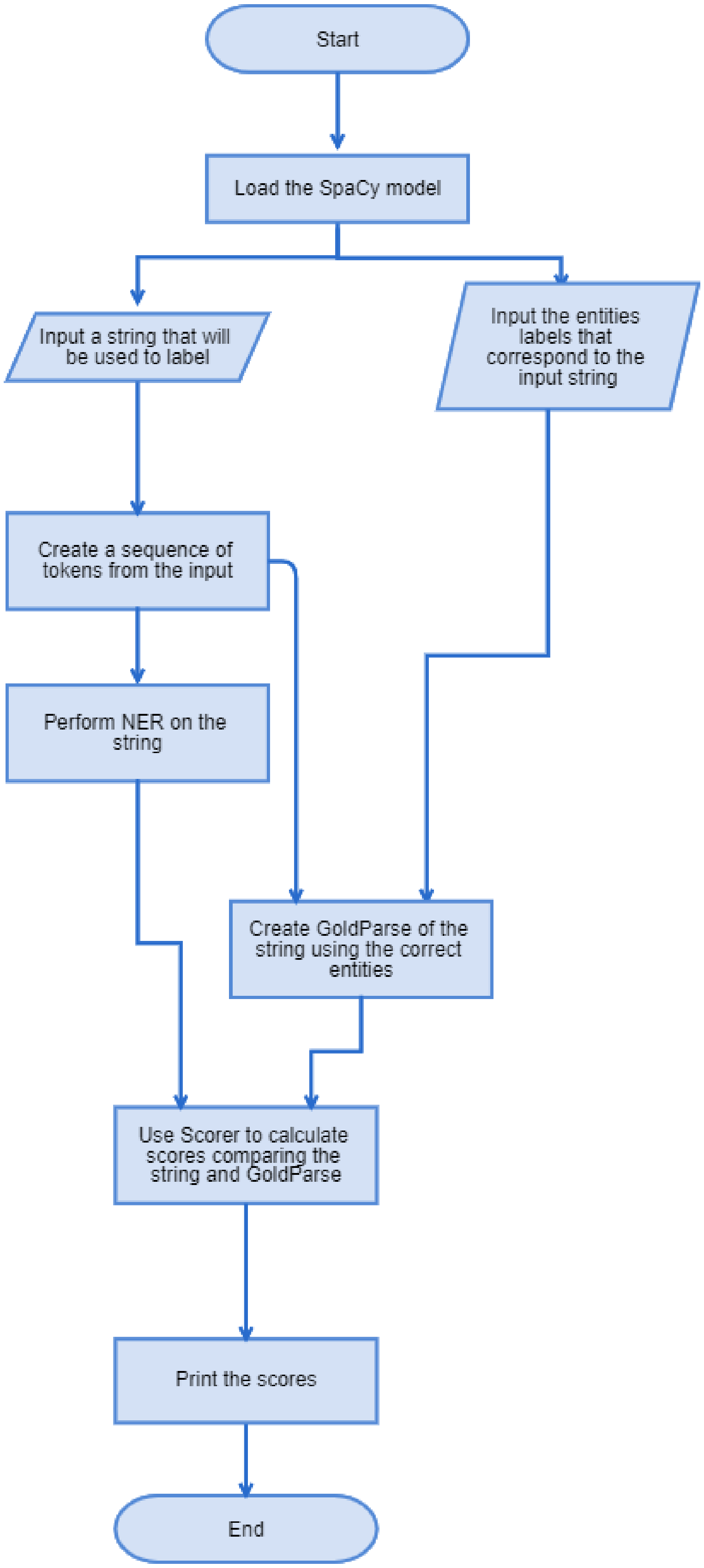
Start
Load the SpaCy model
Input a string that will be used to label
Input the entities labels that correspond to the input string
Create a sequence of tokens from the input
Perform NER on the string
Create GoldParse of the string using the correct entities
Use Scorer to calculate scores comparing the string and GoldParse
Print the scores
End

Figure 11: Flow Chart for Precision, Recall and F1 values for SpaCy

Before the Python script for SpaCy's precision, recall and F1 value calculations is run the following pre-requisites need to be imported:

- SpaCy
- GoldParse from SpaCy library Gold module
- Scorer from SpaCy library Scorer module

SpaCy has a Scorer class which computes evaluation scores of SpaCy's NER classifier. Firstly, the SpaCy model needs to be established, then python script inputs a string that will be used to label. The entities of the string need to be established in the form (number, number, label), these entities give any correct labels in the inputted string.

Scorer has a score method which takes in two parameters: doc and gold. Doc is a container for accessing linguistics features and gold is a collection for training annotations. Doc is the predicted annotations from SpaCy's input which is created using the SpaCy model to label the string. Gold are the correct annotations which is created using GoldParse which take in the sequence of tokens from the input (created using the make_doc function and the entities).

The score method from the scorer class produces the following outputs: token_Acc for token accuracy, tags_acc for part-of-speech tag accuracy, uas for unlabelled dependency score, las for the labelled dependency score, p for the named entity accuracy precision, r for named entity accuracy score for recall and f for named entity accuracy for f1 score. [Explosion AI (3), 2020]

4.1.4 Speed of Stanford, NLTK, SpaCy

Speed of the NER tools can be measured using the time library in Python. The flowchart below demonstrates how the speed is collected:

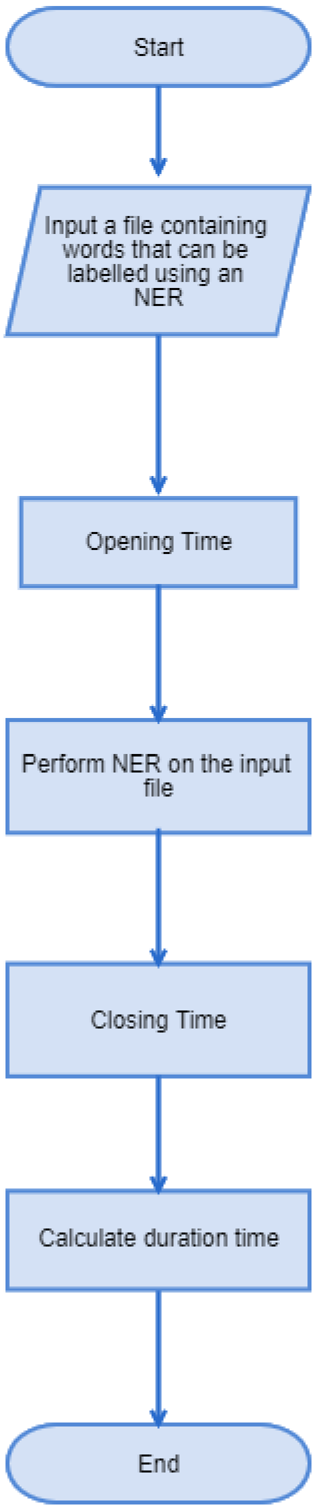
Start
Input a file containing words that can be labelled using an NER
Opening Time
Perform NER on the input file
Closing Time
Calculate duration time
End

Figure 12: Flow Chart for Measuring Speed

Before the speed can be calculated for the NER tools the following pre-requisites need to be imported:

- NLTK
- Stanford NER Tagger from NLTK library
- SpaCy
- Time

The Python script must first input a file that an NER can perform on. The time library has a time function that gives the current time when the function is called. The function is first called under ‘opening’ before the NER tool begins.

Then the NER tool performs NER on the input file. Stanford and NLTK perform NER is the same way described in 4.1.2. SpaCy’s model is loaded and then used on the text to perform NER.

After NER is performed the time function is used again to get a ‘closing’ time on the process.

The duration of the NER performance in calculated by subtracting the opening time from the closing time. This duration can then be printed for evaluation.

4.1.5 Using the NER Tools on New Zealand Locality Data

The same method is used as before to measure the overall accuracy of Stanford NER and NLTK which is explained in 4.1.2.

Many roads that described in the data have had their names shortened such as “Gladfield Rd” which results in the NER’s not recognising the road as a location. Due to the results being so poor, implementation was modified to change any abbreviations of road to the full-length of the word to see the impact of the performance.

As discussed, previous, accuracy is not always the best measurement for evaluating the classifier. Precision, recall and F1 values are proven to be better measurements as they take in consideration for false positives and false negatives which can have high costs. They help where there is a large class distribution which is common especially in this case as most words do not have a label and therefore the measurement will be how well it is not identifying entity labels and not how well it is identifying place names. The same method is used to gather the precision, recall and F1 values as demonstrated in 4.1.3.

As with the other implementation of calculating SpaCy’s accuracy measurements the full dataset would be impossible within the time frame to annotate as the data needs to be in the same format as the training data. Therefore, several lines will be taken from the dataset to calculate the measurements. As we are looking to get measurements to compare the dataset including the original abbreviations of the word road and the full-length word, the lines used for measurements will include roads before and after the abbreviations are changed.

4.2 Classifiers

4.2.1 Count Vectorizer

Count Vectorizer is imported from Scikit Learn feature extraction class for text. Below shows a flow chart which demonstrates how Count Vectorizer is used to extract features to train a classifier:

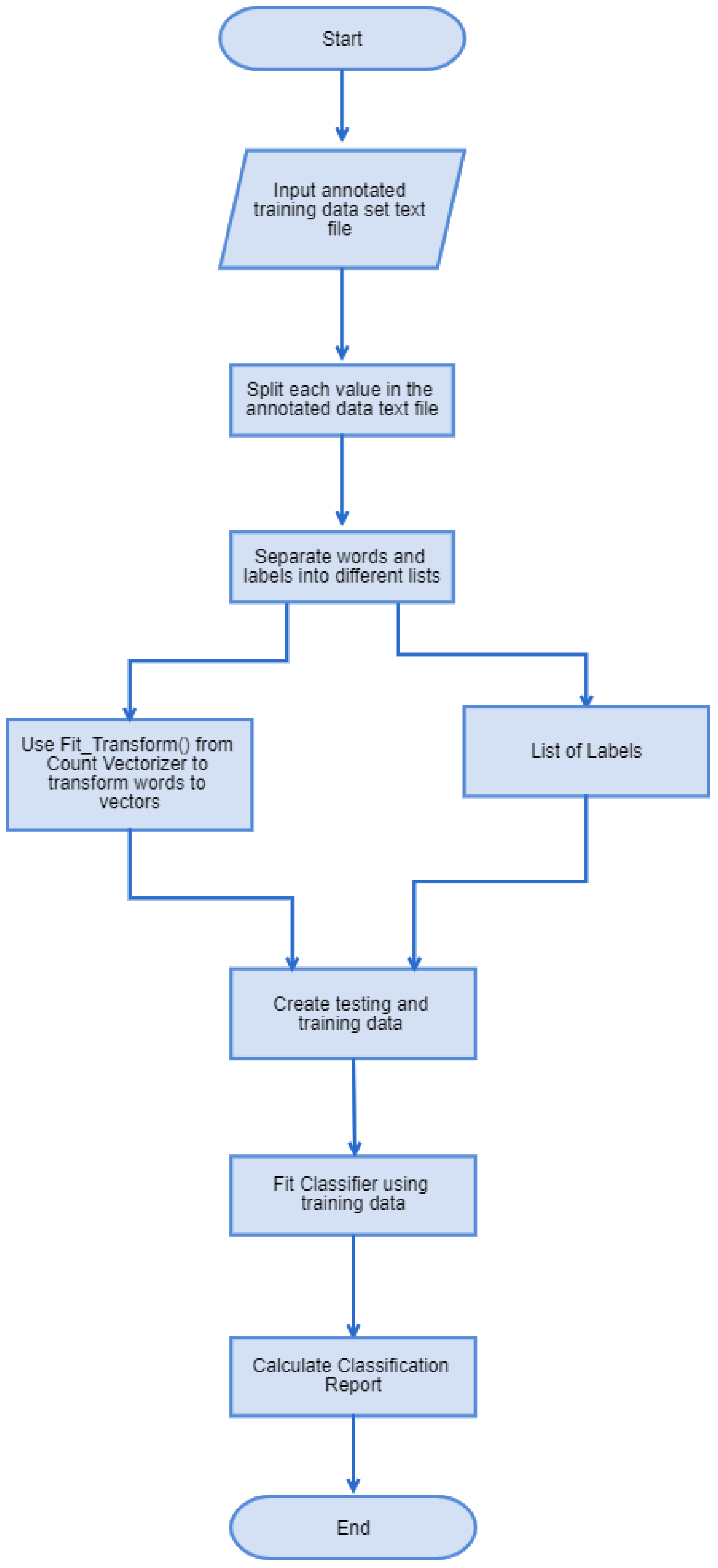
Start
Input annotated training data set text file
Split each value in the annotated data text file
Separate words and labels into different lists
Use Fit_Transform() from Count Vectorizer to transform words to vectors
List of Labels
Create testing and training data
Fit Classifier using training data
Calculate Classification Report
End

Figure 13: Flow Chart for Count Vectorizer Implementation

For the Python script to train a classifier with features extracted using Count Vectorizer the following pre-requisites needs to be imported:

- NumPy
- SGD Classifier from linear model or Linear SVC from SVM module of Sklearn library
- Train Test Split function from model selection module of Sklearn library
- Classification Report from metrics module of Sklearn library

The input file is the training dataset that include words and their relevant labels. The input datafile values are split using the split function and the resulting list can be used in creating 2 new lists of just the words and labels.

The implementation of Count Vectorizer converts a sequence of items of strings to a matrix of token counts. The count vectorizer has a method for fit transform which learns the vocabulary dictionary and returns a document-term matrix. The document-term matrix is an array of shape (n_samples, n_features). The fit transform function takes in the words that have extracted from the input training file.

To obtain train and test data from the original dataset, the Scikit Learn train test split function is used which splits the train and test data 0.25 proportion to the original dataset. The function takes the result of the count vectorizer fit and transform function and the list of labels and results in training data and testing data. The training and testing data includes the words vectors and corresponding labels.

When a classifier is trained using its fit function the parameters are the training data of a sparse matrix in shape (n_samples, n_features) and target values of a ndarray of shape (n_features).

The newly trained classifier can use it's predict function to get predicted labels of the testing data for the words. These predicted labels are then compared to the testing data for labels to output the classification report function.

4.2.2 Using Pre-trained Embeddings

Pre-trained embeddings can be used to gather features to train a classifier on. The following flow chart demonstrates the Python script written to achieve this:

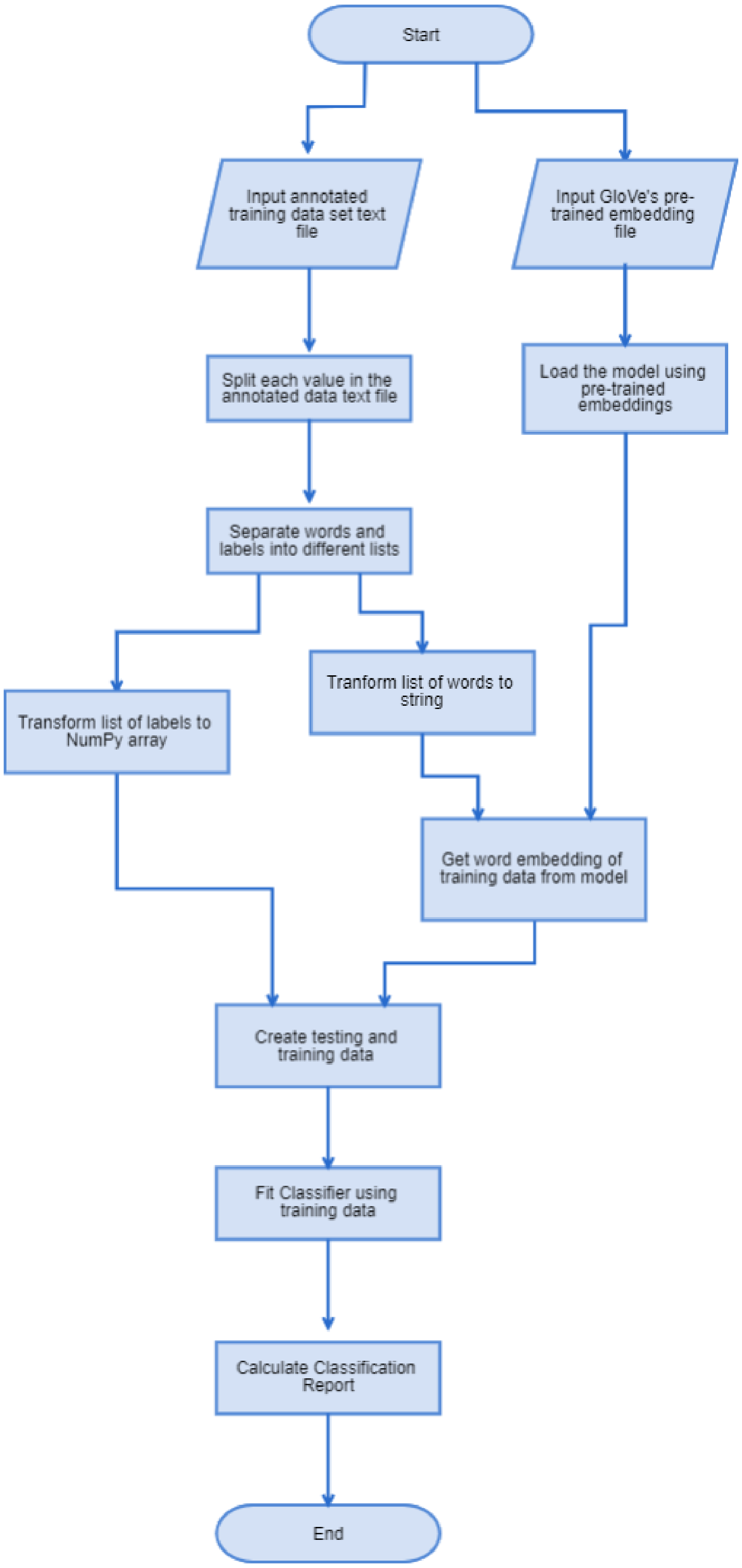
Start
Input annotated training data set text file
Input GloVe's pre-trained embedding file
Split each value in the annotated data text file
Load the model using pre-trained embeddings
Separate words and labels into different lists
Tranform list of words to string
Transform list of labels to NumPy array
Get word embedding of training data from model
Create testing and training data
Fit Classifier using training data
Calculate Classification Report
End

Figure 14: Flow Chart for Using Pre-trained Embeddings

The relevant modules and libraries that are required to be imported for training an SGD classifier with pre-trained embeddings are:

- SGD Classifier from linear model or Linear SVC from SVM module of Sklearn library
- NumPy
- Train Test and Split function from model selection module in Sklearn library
- Classification Report from the metrics model in Sklearn library

There are two input files required when training a classifier with pre-trained embeddings, the annotated training dataset file, and the pre-trained embeddings file. The annotated training dataset file is opened using the open function and the values are split using the split function. The resulting list can be used to create two new lists of just the words and labels by indexing the list of the contents from the input file.

The pre-trained embeddings are downloaded from https://nlp.stanford.edu/projects/glove/ the zip files contains text documents with the word embeddings to be loaded. The different text documents contain the different dimensions of embeddings which need to be in the same directory as the script calling them. The file is read by the Python script using the read function and the encoding is set to 'utf-8'. Every line in the file is split into a list. The word is taken from the list by taking the 0 index. The rest of the line would be the embedding and can be converted to a Numpy array. Finally, the word and its embedding are used to create a dictionary with the word as the key and the embeddings as the value. The dictionary is used as a model to get embeddings.

The list of words is first converted to a string using the join function so the string of all the words from the training input file can be put through the word to vector function. The word to vector function takes in the string and then for every word in the string the get function is used to receive the embedding from the model. The embedding will be the dimension of the same dimensions used in the model.

The list of labels is transformed to a NumPy array so it can be used to train the classifier.

The NumPy array of vectors and the NumPy array of labels are used in train test split function which produces train and test data 0.25 proportion to the input data. The result is training and testing data for the words and its corresponding labels.

Like when using word vectors created from count vectorizer the training data for embeddings and the labels are used to fit the classifier. The classifier can then be used to predict the testing data of the embeddings. The testing labels are the correct labels that can be used to compare the predicted ones.

The classification report takes the predicted labels that were predicted by the newly trained classifier and compares them against the correct labels. From this it produces the precision, recall and F1 values for the classifier.

4.2.4 Different Scikit Learn Classifiers

Different types of classifiers are used such as SGD and LinearSVC. To use Linear SVC or SGD classifier, it would need to be imported from the SVM or Linear Model module in Sklearn library as demonstrated throughout 4.2.

The only change in implementation is using LinearSVC or SGD to fit the training data and then using it to predict.

LinearSVC and SGD classifiers both have inputs of a sparse matrix with shape of (n_samples, n_features) with the word vectors or embeddings and array with shape of (n_features) with the labels corresponding the words. Once the classifier has an instance of itself which is created with using the fit function the classifier can then predict labels for words.

4.2.5 Using Surrounding Words to Classify a Word

To get a certain number of embeddings around a word to be classified the list of words from the training data needs to be processed to get these words. The following flow chart demonstrates how this is obtained:

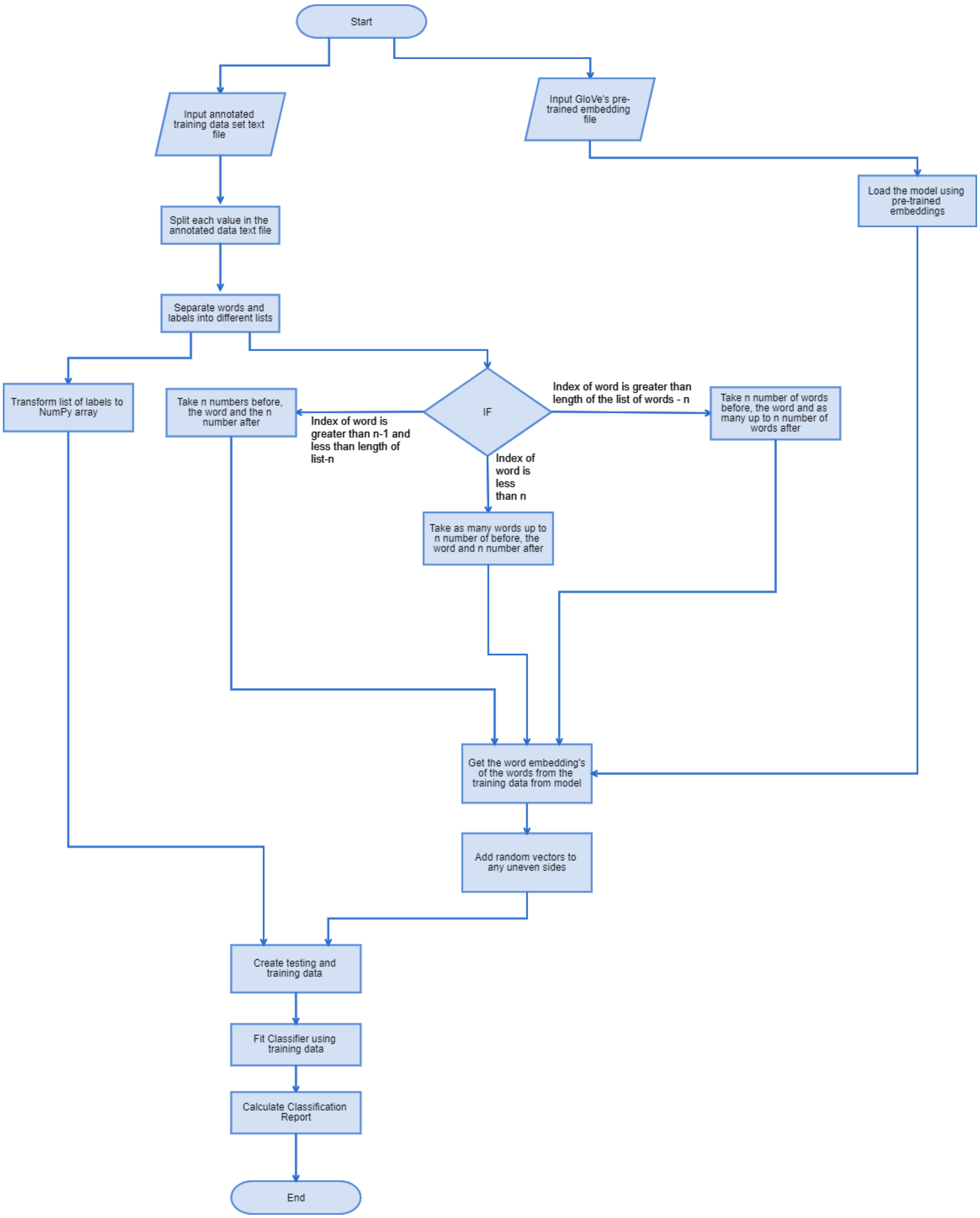
Start
Input annotated training data set text file
Input GloVe's pre-trained embedding file
Load the model using pre-trained embeddings
Split each value in the annotated data text file
Separate words and labels into different lists
Transform list of labels to NumPy array
Take n numbers before, the word and the n number after
IF
Index of word is greater than n-1 and less than length of list-n
Index of word is greater than length of the list of words - n
Take n number of words before, the word and as many up to n number of words after
Index of word is less than n
Take as many words up to n number of before, the word and n number after
Get the word embedding's of the words from the training data from model
Add random vectors to any uneven sides
Create testing and training data
Fit Classifier using training data
Calculate Classification Report
End

Figure 15: Flow Chart for Taking n Embeddings Around Word to be Classified for Training a Classifier

The imports necessary to train a classifier using pre-trained embeddings and taking n number of words before and after the word to be classified include:

- Random
- NumPy
- SGD Classifier from linear model or Linear SVC from SVM module of Sklearn library
- Train Test Split function from Sklearn's model selection
- Classification Report from Sklearn library module metrics

As with just using pre-trained embeddings to extract features for a classifier an annotated training dataset input in required as well as the GloVe's pre-trained embedding file. The annotated file is opened using the open function and read using the read function. The values in the file are split into a list. To create two new lists of just the words and labels the resulting list of splitting the file values can be indexed to get them. The GloVe's pre-trained embeddings file which is used to create a model is also opened using the open function.

To create the model the same steps are taking as in 4.2.2. This model is a dictionary with the word as a key and the value being their embedding. The model can be used to get embeddings of words in the training data.

The words can be iterated through and then the embeddings are taken from either side of the word. If there are 3 embeddings either side of a word being taken and the embeddings have dimensions of 100 the feature would now have 700 data values instead of just 100 values as it includes the embedding of the word to be classified as well as 3 embeddings to the left and right.

If 3 embeddings are being taken either side of the word to be classified, the first and the last 3 words in the training data will not have three embeddings both sides of the word. To overcome this issue the embeddings either side will be taken until they reach the end of the list. Where there are less than three embeddings, a random array of the same dimensions will be added until it matches the length of the 3 embeddings to the left, the word and the 3 embeddings to the right.

The list of labels from the input file are converted to a NumPy array so it can be used to train the classifier.

The NumPy array of the embeddings and the NumPy array of the labels are used to get testing and training data using Sklearn train test split function. The training and testing data for word and labels are 0.25 in proportion to the dataset.

The training data for the words and labels are used to fit a classifier which then results in an instance of itself which can perform the predict function. The classifier can try predicting labels for the testing data of words.

The classification report takes in the predicted labels of the words and the correct labels and outputs the accuracy measurements.

4.2.6 Averaging Embeddings

When n number of embeddings around the word to be classified are taking as features for training they can then be calculated to create one embedding that is the average of the n number of words before, the word and the n number of words after the word to be classified. The following flow chart demonstrates this, it being like taking n number of embeddings around the word to be classified with averaging function being included before training:

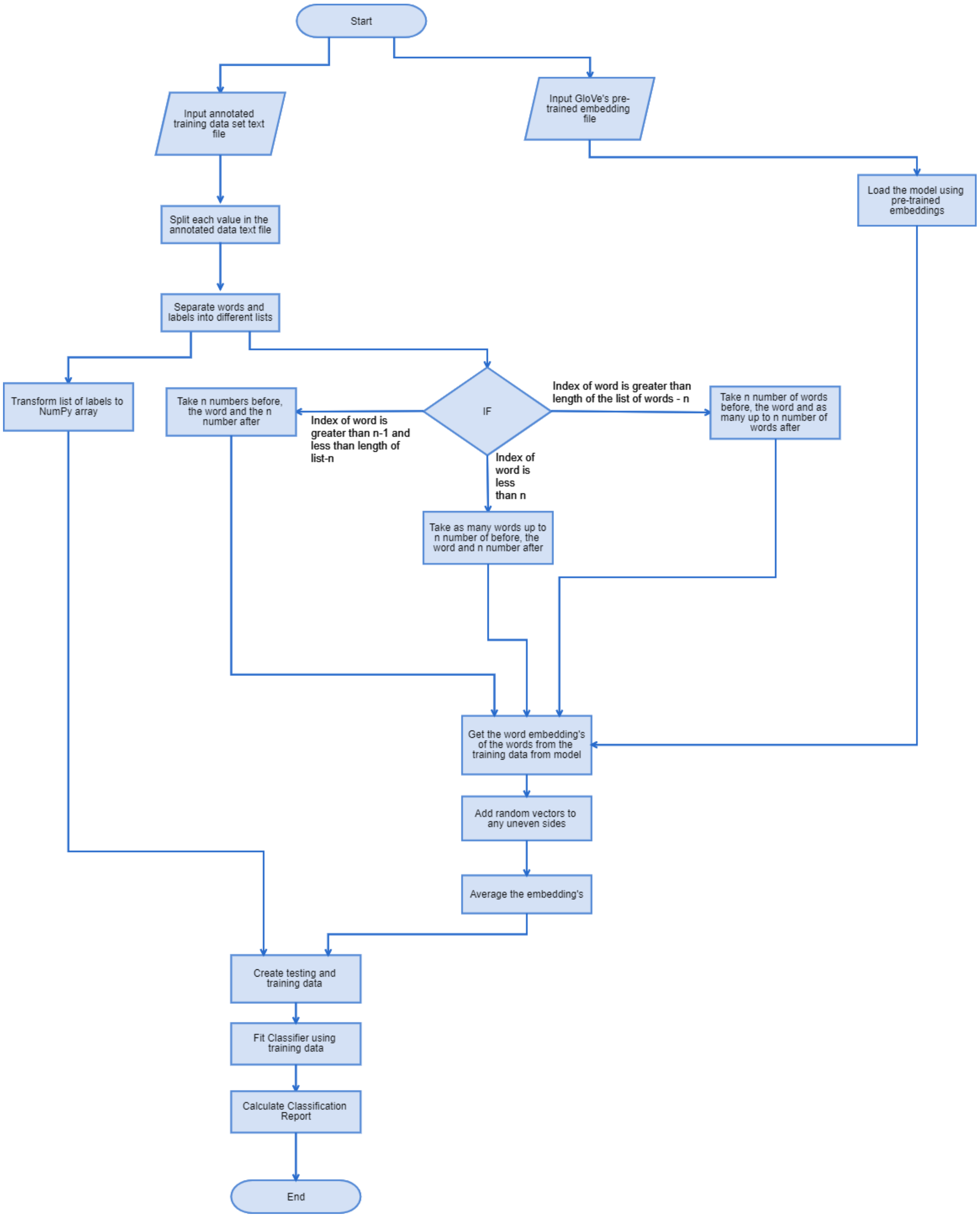
Start
Input annotated training data set text file
Input GloVe's pre-trained embedding file
Load the model using pre-trained embeddings
Split each value in the annotated data text file
Separate words and labels into different lists
Transform list of labels to NumPy array
Take n numbers before, the word and the n number after
IF
Index of word is greater than n-1 and less than length of list-n
Index of word is greater than length of the list of words - n
Take n number of words before, the word and as many up to n number of words after
Index of word is less than n
Take as many words up to n number of before, the word and n number after
Get the word embedding's of the words from the training data from model
Add random vectors to any uneven sides
Average the embedding's
Create testing and training data
Fit Classifier using training data
Calculate Classification Report
End

Figure 16: Flow Chart for Averaging n Embeddings around Word to be Classified for Training a Classifier

The imports required for this implementation are the exact same as when taking n number of embeddings around the word as well as the word to be classified.

The only difference to 4.2.5. implementation is after there is n embeddings before, the word to be classified and n embeddings after there is n*2+1 embeddings that can be used as features for the classifier training. Instead of using the n*2+1 embeddings they are averaged to create 1 embedding.

The NumPy average function is used and the values are averaged on the zero axis which will be n*2+1. The average is then taking as the input for train test learn. Finally, training the classifier with the average embedding and labels and then producing a classification report.

5.0 Results and Evaluation

The following section includes results of the implementation of the NER tools and personally created classifiers. The first results presented include performance measurements of the NER tools SpaCy, NLTK and Stanford. The performance measurements are overall accuracy and then the precision, recall and F1 values. Secondly the speed of the NER tools is presented. Then the accuracy results are presented for the NER tools using the New Zealand locality data.

The second half of the result presentation in this section include the results of newly created classifiers. Firstly, the results show the comparisons of using Count Vectorizer instead of pre-trained embeddings then the next results use the GloVe pre-trained embeddings. Followed by the results of using the same pre-trained embeddings with taking 3 words to the left and right of the word to be classified and then taking 5 words instead of 3. Next shows the results of using LinearSVC classifier instead of the SGD classifier which was used in the previous set of results. In the final section uses graphs to demonstrate the comparison of the variables when training the classifiers.

5.1 NER Tools

5.1.1 Comparison of NLTK and Stanford NER Accuracy Results

I chose to run 11 different text documents that were annotated by myself to be able to gather a set of results of overall accuracy. The accuracy result is the total number of true positives and true negatives divided by the total predicted values (true positives + true negatives + false negatives + false positives). The accuracy is measured by the identification of all entities including 'O' for those with no entities, location, person, and organisations. All text documents contain a short paragraph of information of a certain area, region or country, all data from the text documents come from the Wikipedia page of each location. The data was chosen to include locations so that the data would be able to compare many location tags and entities as that is the main area of focus for this project to be able to take the detected place names and disambiguate them. All files are similar in length.

The range of Stanford NER accuracy percentage range from 85.7% to 99.4% which is a range of 13.7%. The NLTK has a bigger range from 79.8% to 97.2% which is a range of 17.4%. The average score for accuracy of Stanford NER is 94.8% which gives evidence that the Stanford NER is highly accurate but there is room for improvement. Some accuracy issues can occur from human error when annotating data. The NLTK accuracy average score is 88.9% which is still a high score but less than Stanford NER so the NLTK is less reliable and accurate named entity recogniser than the Stanford NER. In every file the NLTK is always less accurate that the Stanford NER.

Text file of annotated data	Stanford NER	NLTK
New Zealand	95.6%	91.1%
South East England	85.7%	81.4%
Cwmbran	97.9%	89.4%
America	99.4%	97.2%
Australia	96.8%	94.7%
Iceland	97.2%	94.3%
Finland	94.0%	79.8%
South Wales	92.2%	95.2%
Southern France	92.0%	87.5%
Fiji	93.5%	84.6%
Philippines	97.9%	82.4%
Average	94.8%	88.9%

Figure 17: Accuracy Measurements for Stanford and NLTK

5.1.2 Precision, Recall, F1 Values

Stanford NER Precision, Recall and F1 Values

Below demonstrates the precision, recall and f1 values for the same files that were measured for accuracy in table 1. Unlike the accuracy results the precision and recall values are calculated on the identification of location entities and not the other, person or organisation entities. This makes it easier to see if the NER tools are better for location identification.

File	Precision Value	Recall Value	F1 Value
New Zealand	1.0	0.8	0.889

South East England	0.857	0.6	0.706
Cwmbran	1.0	0.875	0.933
America	1.0	0.952	0.976
Australia	1.0	0.6	0.749
Iceland	1.0	0.733	0.846
Finland	1.0	0.844	0.915
South Wales	1.0	0.727	0.842
Southern France	0.926	0.595	0.725
Fiji	0.967	0.696	0.810
Philippines	1.0	0.919	0.958
Average	0.977	0.758	0.849

Figure 18: Precision, Recall and F1 for Stanford

Recall is the most important value in the case of detecting place names in a text document. False negatives have a higher cost than false positives when detecting place names as the locations that go unmissed could result in incorrect disambiguation of the locations found. For example, if 'Manhattan' were falsely identified as a negative it could result in the words 'York' being mis-identified as the English City not the American state. The average of precision value for Stanford for these files is 0.977 which is remarkably high and demonstrates the Stanford NER works well at not producing false positives. The average recall value for Stanford NER calculated using these files is 0.758. Although it is not as high as the precision value it is still relatively high therefore Stanford is better at not producing false positives than false negatives. F1 value is 0.849 on average, when compared to the accuracy average value calculated by NLTK metrics class accuracy (shown above in table 1) which was 94.8%. Although the accuracy average value is higher than the F1 value it can be a misconception as the accuracy value takes in the identification of all entities not just the location entity like F1 does. It is easier to identify a true negative so the result would be higher when taking these into account, as F1 value does not take these into consideration the accuracy value will be higher than the F1.

NLTK Precision, Recall and F1 Values

Below shows the precision, recall and f1 values for NLTK for the text files that were used to previously calculate the overall average for the NER.

Text File	Precision Value	Recall Value	F1 Value
New Zealand	1.0	0.467	0.636
South East England	0.889	0.4	0.552
Cwmbran	1.0	0.375	0.545
America	0.864	0.792	0.826
Australia	1.0	0.5	0.667
Iceland	1.0	0.467	0.636
Finland	0.941	0.5	0.653
South Wales	0.941	0.516	0.667
Southern France	0.778	0.5	0.609

Fiji	0.875	0.304	0.452
Philippines	1.0	0.324	0.489
Average	0.935	0.478	0.612

Figure 19: Precision, Recall and F1 for NLTK

The average precision value for NLTK for these text files is 0.935, which is significantly high but lower than the precision value for Stanford NER using the same files therefore confirming that the Stanford NER is a better classifier than NLTK. The recall value average for NLTK is 0.478 which is significantly lower than Stanford average recall value of 0.758. The F1 value average for the NLTK classifier is 0.612 which is lower than Stanford NER by 0.237 which is a big difference. This confirms and supports the argument the Stanford NER is a better evaluator than NLTK.

SpaCy Precision, Recall and F1 Values

Below gives the precision, recall and F1 value for several lines of each data file.

Text File	Precision Value	Recall Value	F1 Value
New Zealand	1.0	1.0	1.0
South East England	1.0	0.50	0.66
Cwmbran	1.0	0.66	0.80
America	1.0	0.50	0.66
Australia	1.0	0.50	0.66
Iceland	1.0	0.50	0.66
Finland	0.5	0.66	0.57
South Wales	1.0	0.75	0.85
Southern France	0.6	0.43	0.50
Fiji	0.5	0.40	0.44
Philippines	0.5	0.33	0.40
Average	0.83	0.57	0.65

Figure 20: Precision, Recall and F1 for SpaCy

The average precision value for SpaCy is 0.83 which is a relatively high score showing SpaCy is good at identifying true positives. The average recall value is 0.57 which shows SpaCy performs poorly with regards to identifying false negatives in comparison to identifying true positives. The average F1 value is 0.65 which demonstrates that SpaCy performs relatively well regarding accuracy.

5.2.3 Comparing Speed of the NER Tools

The same 11 text files on areas, regions and countries that were used to test the named entity recognisers for accuracy are used for testing NLTK and Stanford NER for speed. If speed were to become an issue, then it may be a better option to use a less accurate NER if it is going to work faster. The results are measured in seconds. The range of speed for Stanford NER is from 4.79 seconds to 3.46 seconds which is a range of 1.3 seconds. The range of speed for NLTK is from 0.50 seconds to 0.28 seconds which is a range of 0.22

seconds which is considerably lower than the range for Stanford NER. The average speed for Stanford NER is 4.34 seconds. The average speed for NLTK is 0.37 seconds which is also a considerably faster measurement than the speed for Stanford NER. NLTK is a lot slower than Stanford NER but Stanford NER is more accurate than NLTK, as all text files used for this comparison have been quite a small text file with all similar lengths it may be interesting to use different file sizes to see if the greater time taken for Stanford NER to work is worth using it for its better accuracy in comparison to NLTK. SpaCy has an astonishing faster named entity recogniser with beating NLTK by 0.14 seconds in its slowest time. SpaCy has an average of 0.12 seconds compared to Stanford's 4.34 seconds average and NLTK's 0.37 seconds. SpaCy only ranges from 0.24 to 0.06 seconds with a range of 0.18 seconds. If speed were the only factor to influence a developer's choice of named entity recogniser then SpaCy would be the best choice.

Text file of data	Stanford NER	NLTK	SpaCy
New Zealand	4.49 seconds	0.38 seconds	0.24 seconds
South East England	4.79 seconds	0.33 seconds	0.07 seconds
Cwmbran	4.33 seconds	0.28 seconds	0.06 seconds
America	4.61 seconds	0.39 seconds	0.17 seconds
Australia	4.54 seconds	0.40 seconds	0.09 seconds
Iceland	4.29 seconds	0.35 seconds	0.11 seconds
Finland	4.53 seconds	0.32 seconds	0.08 seconds
South Wales	3.46 seconds	0.35 seconds	0.11 seconds
Southern France	3.62 seconds	0.42 seconds	0.15 seconds
Fiji	4.70 seconds	0.50 seconds	0.17 seconds
Philippines	4.42 seconds	0.37 seconds	0.11 seconds
Average	4.34 seconds	0.37 seconds	0.12 seconds

Figure 21: Speed Measurements

5.2.4 Using New Zealand Locality Data for NER Tools

The accuracy of Stanford NER, it has an accuracy percentage of 79.59%. When all cases of "rd" are placed with the full word "road" the accuracy percentage increases to 80.30%. This demonstrates the NER's ability to recognise the full name of a location but struggles to recognise human slang or written notes of an area.

NLTK gives an accuracy percentage of 68.58% for the locality data which again shows that NLTK is the least accurate NER in comparison to Stanford NER. When changing all "rd" occurrences to "road" just like with Stanford the percentage increases to 69.44%.

Below shows the precision, recall and f1 value for the Stanford NER for the locality data on New Zealand. It also shows a comparison of the values after the abbreviation of 'road' are changed to the full-length word.

Text File	Precision Value	Recall value	F1 value
Locality Data (original)	0.822	0.198	0.319
Locality Data (full-length of any abbreviations of road)	0.732	0.228	0.347

Figure 22: Precision, Recall and F1 value for Stanford using Locality Data

The recall value for the locality data is significantly low, which can demonstrate some issues of Stanford NER. Many entities were not identified where they should have been. Where the abbreviations for road are changed to the full word it clearly shows how the NER can then identify roads better rather than when they were not being identified when written as human short notes. This could be a potential area of focus for Stanford as it could be trained on using shortened human notes or slang to identify locations that may not be written as correctly as they should.

Below shows the precision, recall and f1 value for NLTK for the same locality data:

Text File	Precision Value	Recall Value	F1 Value
Locality Data (original)	0.085	0.038	0.053
Locality Data (full-length of any abbreviations of road)	0.108	0.044	0.063

Figure 23: Precision, Recall and F1 values for NLTK using Locality Data

NLTK have incredibly low precision, recall and f1 values compared the Stanford NER. Having the abbreviations of road changed to the full word does help increase the values meaning NLTK must also struggle to identify roads if the word is not in full, this could also be a potential area to be improved for NLTK, like it is in Stanford NER.

Below shows the precision, recall and F1 values for SpaCy NER for the New Zealand locality data. It will compare the original data and the data with any abbreviations of road written as the full length of the word.

Text File	Precision	Recall	F1 Value
Locality Data (original)	1.0	0.20	0.33
Locality Data (full-length of any	1.0	0.25	0.40

abbreviations of road)			

Figure 24: Precision, Recall and F1 for SpaCy using Locality Data

Using SpaCy' s named entity recogniser for the English language it produces the named entity labels along with their texts. When SpaCy NER labels the texts, it uses more labels which are explained in the background of SpaCy NER. than Stanford NER so many areas that could be considered as a location such as "Castlepoint Road" or "Lake Orbell" are labelled as 'FAC'. As the text in the locality data file is written as notes many descriptions of roads are written as "Mangaone Rd" using the shortened "Rd" for road. SpaCy does not recognise these as roads which would give the "FAC" label, they produce ORG labels which indicates SpaCy NER is tagging the text "Mangaone Rd" as an organisation. Other locations were labelled as organisations for example "Paroa", "Thorton Main", "Whakatone", "Tauranga", "Morton Mains" and "Maitland Vale". Sentences within the locality data include directions such examples as "1km along Goatleys Road" which was labelled by SpaCy as a Quantity. The "1km along" could indication why SpaCy NER thought it was a quantity by with the word "road" having a word before it beginning with a capital letter it should of distinguished the difference and recognised "Goatleys Road" as a separate entity which would be associated with a "FAC" or a location label. SpaCy does not recognise many roads where in the sentence the word road does not start with a capital letter, such as if "Goatleys road" was written this way rather than the previous "Goatleys Road". As the word 'Road' does not have a capital letter SpaCy does not label it as a place. The NER recognised "Omahuta forest (compartment 8)" as an organisation where it should have recognised "Omahuta forest" as a location. The NER also recognises many places as PERSON for example "Elie Bay", "Ahuriri Lagoon" and "Te Kaw". Some other areas that impacted SpaCy's ability to label text correctly was where notes were written to demonstrate a measurement or road between two places such as "Hawera-New Plymouth main highway". A human can read this and understand it is the main Highway between the places Hawera and New Plymouth but SpaCy only recognises and labels "Plymouth" as a place which may be the Plymouth in England. Ignoring the "Hawera" and the "Main Highway" which could be used to disambiguate that the place is in New Zealand not England, UK.

5.2.5 Evaluation of NER Tools

Below shows the average precision, recall and F1 values for the NER tools using the small datasets from Wikipedia on different countries, counties, and places.

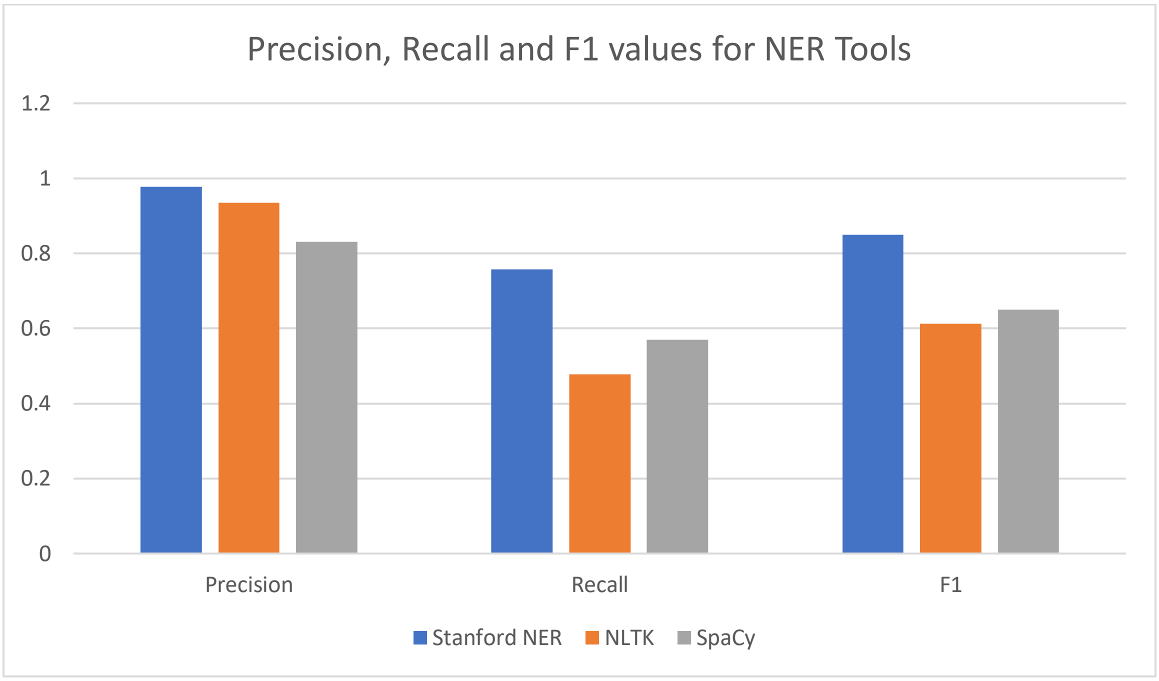

Figure 25: Evaluation of Precision, Recall and F1 for NER Tools

In terms of precision (identifying true positives) Stanford NER performs better than NLTK which performs better than SpaCy. Recall values shows Stanford NER performing the best, SpaCy next and NLTK performing the worst. The F1 value gives a better overall accuracy value using the precision and recall values which shows Stanford NER performing the best with SpaCy in second and NLTK performing the worst out of the three. As Stanford only has the 3 values: person, organisation and location for its entities, it may mean entities such as roads and bodies of water are being classed as locations in Stanford but not in SpaCy where there are different labels for these such as 'LOC' for a body of water (Oceans etc) and 'FAC' for roads/highways. This may impact the accuracy results as roads annotated as a location may not come up as the location label for SpaCy. This may result in future work needed to group the different location-related labels in SpaCy into one label to see if SpaCy's accuracy measurements need improving.

Next will include the evaluation of the speed of the NER tools. This will suggest if the speed will have an impact on using a standard even if it has high accuracy.

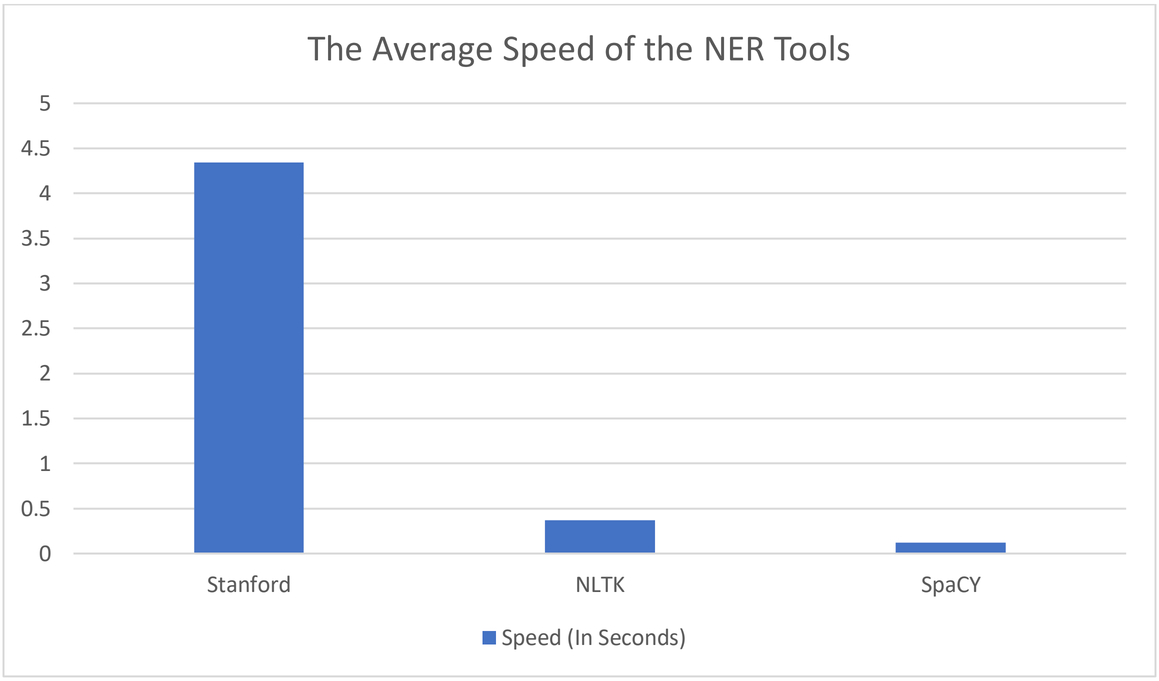

Figure 26: Average Speed Evaluation

SpaCy is the quickest NER on average and Stanford is the slowest. Although Stanford has the highest accuracy measurements if it is taking an average of over 4 seconds to perform NER on such small datasets. So, it may be useful to consider using SpaCy who also has relatively high accuracy measurements and has a very quick average of 0.12 seconds for the same datasets.

It is useful to compare the NER tools performance on a larger datasets. Using the New Zealand locality data, it can give an idea of how the NER's performance changes using the larger dataset.

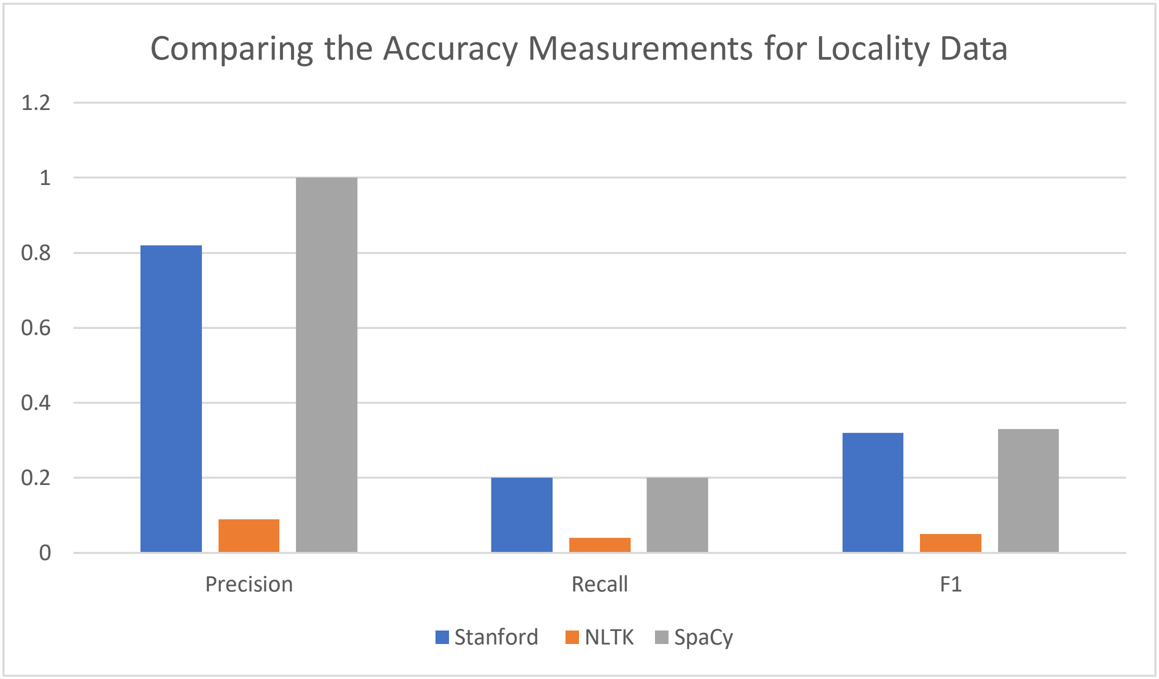

Figure 27: Comparing the NER tools using the Locality Data

Although Stanford and SpaCy have relatively high precision values all NER tools worked poorly on the locality data. As previously discussed, the locality data is human written notes and it may be the cause of the poor results from all three NER tools. This is a motivation for creating my own classifier as it can be trained on human written notes whereas the NER tools used will not be trained on this type of format.

5.2 Classifiers

5.2.1 Count Vectorizer

The results below are of an SGD classifier that is trained using Count Vectorizer to get word vectors of the training data. The different training data is used to compare the results of different sized dataset. The training and test data is split using Scikit Learn's train test split function, the training and test data is automatically set to 0.25 proportion to the dataset. The only features used in the classifier are the word vectors of the word to be classified.

Training Data	Precision	Recall	F1
Fiji	0.6	0.43	0.5
NER dataset (5000 words)	0.69	0.57	0.62
New Zealand Locality	0.77	0.59	0.67
Average	0.69	0.53	0.60

Figure 28: Using Count Vectorizer for training an SGD Classifier

5.2.2 Pre-trained Embeddings

Below include the results of an SGD classifier that is trained using word embeddings obtained by GloVe's pretrained embeddings. The results include the different dimensions and each different training datasets. The training and test data is also split using the train test split function from Scikit-Learn with the training and test data both set to 0.25 proportion to the dataset. The only features used in the classifier is the word embedding of the word to be classified.

Training Data	Embedding Dimension	Precision	Recall	F1
Fiji	50	0.57	0.67	0.62
Fiji	100	0.92	0.85	0.88
Fiji	200	0.86	0.86	0.86
Fiji	300	0.69	0.90	0.78
Ner dataset (5000)	50	0.20	0.50	0.29
Ner dataset (5000)	100	0.35	0.45	0.39
Ner dataset (5000)	200	0.37	1.0	0.54
Ner dataset (5000)	300	0.41	1.0	0.58
NZ locality	50	0.77	0.30	0.43
NZ locality	100	0.65	0.30	0.41
NZ locality	200	0.59	0.30	0.40
NZ locality	300	0.61	0.30	0.40
Average		0.58	0.62	0.55

Figure 29: Using GloVe's pretrained embeddings to train an SGD Classifier

The next results shown below are of an SGD classifier that is trained using the word embeddings obtained by the GloVe's pretrained embeddings. The results include the different dimensions and each different training datasets. The training and test data is split using the train test split function from Scikit-Learn with the training and test data both set to 0.25 proportion set to 0.25 proportion to the dataset. The features used in the classifier are the word embeddings of the word to be classified along with the 3 words before and the 3 words after.

Training Data	Embedding Dimension	Precision	Recall	F1
Fiji	50	0.88	0.58	0.70
Fiji	100	0.6	0.75	0.67
Fiji	200	0.40	1.0	0.57
Fiji	300	0.64	0.75	0.69
Ner dataset (5000)	50	0.49	0.69	0.57

Ner dataset (5000)	100	0.52	0.61	0.52
Ner dataset (5000)	200	0.51	0.49	0.50
Ner dataset (5000)	300	0.52	0.64	0.57
NZ locality	50	0.66	0.52	0.59
NZ locality	100	0.66	0.53	0.59
NZ locality	200	0.64	0.52	0.58
NZ locality	300	0.65	0.58	0.61
Average		0.60	0.64	0.60

Figure 30: Using Glove's pre-trained embeddings and 3 words either side of word to be classified to train an SGD Classifier

The next results shown below are of an SGD classifier that is trained using the word embeddings obtained by the GloVe's pretrained embeddings. The results include the different dimensions and each different training datasets. The training and test data is split using the train test split function from Scikit-Learn with the training and test data both set to 0.25 proportion set to 0.25 proportion to the dataset. The features used in the classifier are the word embeddings of the word to be classified along with the 5 words before and the 5 words after.

Training Data	Embedding Dimension	Precision	Recall	F1
Fiji	50	0.60	0.46	0.52
Fiji	100	0.83	0.71	0.77
Fiji	200	0.44	0.80	0.57
Fiji	300	0.91	0.71	0.80
Ner dataset (5000)	50	0.41	0.49	0.45
Ner dataset (5000)	100	0.61	0.68	0.64
Ner dataset (5000)	200	0.65	0.53	0.59
Ner dataset (5000)	300	0.59	0.46	0.51
NZ locality	50	0.66	0.55	0.60
NZ locality	100	0.65	0.54	0.64
NZ locality	200	0.64	0.56	0.60
NZ locality	300	0.62	0.52	0.57
Average		0.63	0.58	0.61

Figure 31: Using GloVe's pre-trained embeddings and 5 embeddings either side of word to be classified to train an SGD Classifier

5.2.3 Averaging Embeddings

The next results shown below are of an SGD classifier that is trained using the word embeddings obtained by the GloVe's pretrained embeddings. The results include the different dimensions and each different training datasets. The training and test data is split using the train test split function from Scikit-Learn with the training and test data both set to 0.25 proportion set to 0.25 proportion to the dataset. The features used in the classifier are the average of the word embeddings of the word to be classified along with the 3 words before and the 3 words after.

Training Data	Dimensions	Precision	Recall	F1
Fiji	50	0.47	0.69	0.56
Fiji	100	0.40	0.25	0.31
Fiji	200	0.55	0.50	0.52
Fiji	300	0.33	0.42	0.37
Ner Dataset (5000)	50	0.50	0.54	0.52
Ner Dataset (5000)	100	0.65	0.62	0.63
Ner Dataset (5000)	200	0.66	0.50	0.57
Ner Dataset (5000)	300	0.58	0.70	0.63
NZ Locality	50	0.50	0.42	0.46
NZ Locality	100	0.57	0.43	0.49
NZ Locality	200	0.57	0.42	0.48
NZ Locality	300	0.54	0.50	0.52
Average		0.53	0.50	0.51

*Figure 32: SGD Classifier trained with **Average** of Embeddings 3 to left and 3 to right and word to be classified*

The next results shown below are of an SGD classifier that is trained using the word embeddings obtained by the GloVe's pretrained embeddings. The results include the different dimensions and each different training datasets. The training and test data is split using the train test split function from Scikit-Learn with the training and test data both set to 0.25 proportion set to 0.25 proportion to the dataset. The features used in the classifier are the average of the word embeddings of the word to be classified along with the 5 words before and the 5 words after.

Training Data	Dimensions	Precision	Recall	F1
Fiji	50	0.2	0.24	0.12
Fiji	100	1.0	0.20	0.33
Fiji	200	0.33	0.20	0.25
Fiji	300	0.40	1.0	0.57
Ner Dataset (5000)	50	0.25	0.30	0.27

Ner Dataset (5000)	100	0.30	0.57	0.57
Ner Dataset (5000)	200	0.36	0.44	0.40
Ner Dataset (5000)	300	0.32	0.39	0.35
NZ Locality	50	0.20	0.42	0.27
NZ Locality	100	0.35	0.41	0.38
NZ Locality	200	0.36	0.50	0.42
NZ Locality	300	0.30	0.41	0.35

Figure 33: SGD Classifier Trained Using Average of Embeddings 5 to left, 5 to right and the word to be classified

5.2.4 Using Different Classifiers

The next results shown below are of a LinearSVC classifier that is trained using the word embeddings obtained by the GloVe's pretrained embeddings. The results include the different dimensions and each different training datasets. The training and test data is split using the train test split function from Scikit-Learn with the training and test data both set to 0.25 proportion set to 0.25 proportion to the dataset. The features used in the classifier is the word embeddings of the word to be classified.

Training Data	Embedding Dimension	Precision	Recall	F1
Fiji	50	0.67	0.91	0.77
Fiji	100	0.87	1.0	0.93
Fiji	200	0.75	0.75	0.75
Fiji	300	0.79	1.0	0.88
Ner dataset (5000)	50	0.36	1.0	0.53
Ner dataset (5000)	100	0.36	1.0	0.53
Ner dataset (5000)	200	0.37	0.99	0.53
Ner dataset (5000)	300	0.30	1.0	0.46
NZ locality	50	0.66	0.30	0.42
NZ locality	100	0.48	0.20	0.28
NZ locality	200	0.59	0.30	0.39
NZ locality	300	0.51	0.40	0.45

Figure 34: Using GloVe's pretrained embeddings to train a Linear SVC classifier

The next results shown below are of a LinearSVC classifier that is trained using the word embeddings obtained by the GloVe's pretrained embeddings. The results include the different dimensions and each different training datasets. The training and test data is split using the train test split function from Scikit-Learn with the training and test data both set to 0.25 proportion set to 0.25 proportion to the dataset. The features used in the classifier are the word embeddings of the word to be classified as well as the 3 words before and 3 words after.

Training Data	Embedding Dimension	Precision	Recall	F1
Fiji	50	0.62	0.71	0.67
Fiji	100	0.80	0.73	0.76
Fiji	200	0.62	0.50	0.56
Fiji	300	0.60	0.82	0.69
Ner dataset (5000)	50	0.60	0.60	0.60
Ner dataset (5000)	100	0.60	0.56	0.58
Ner dataset (5000)	200	0.45	0.52	0.48
Ner dataset (5000)	300	0.55	0.58	0.56
NZ locality	50	0.65	0.54	0.59
NZ locality	100	0.66	0.57	0.61
NZ locality	200	0.63	0.55	0.59
NZ locality	300	0.63	0.57	0.60

Figure 35: Using GloVe's pretrained embeddings and taking 3 embeddings either side of word to be classified to train a Linear SVC Classifier

The next results shown below are of a LinearSVC classifier that is trained using the word embeddings obtained by the GloVe's pretrained embeddings. The results include the different dimensions and each different training datasets. The training and test data is split using the train test split function from Scikit-Learn with the training and test data both set to 0.25 proportion set to 0.25 proportion to the dataset. The features used in the classifier are the word embeddings of the word to be classified as well as the 5 words before and 5 words after.

Training Data	Embedding Dimension	Precision	Recall	F1
Fiji	50	0.29	0.29	0.29
Fiji	100	0.73	0.92	0.81
Fiji	200	0.71	0.62	0.67
Fiji	300	0.89	0.57	0.70
Ner dataset (5000)	50	0.47	0.60	0.53
Ner dataset (5000)	100	0.52	0.63	0.57
Ner dataset (5000)	200	0.52	0.65	0.58

Ner dataset (5000)	300	0.62	0.52	0.57
NZ locality	50	0.66	0.54	0.59
NZ locality	100	0.64	0.53	0.58
NZ locality	200	0.57	0.54	0.56
NZ locality	300	0.59	0.53	0.56

Figure 36: Using GloVe's pretrained embeddings and taking 5 embeddings either side of the word to be classified to train a Linear SVC Classifier

5.2.4 Evaluation of Classifier Results

Comparing the main changing variables of the classifier being the type of classifier, using pretrained embeddings or using Count Vectorizer, increasing the dimensions of embeddings and taking n number of words either side of the word to be classified many pattern can be found and evaluated.

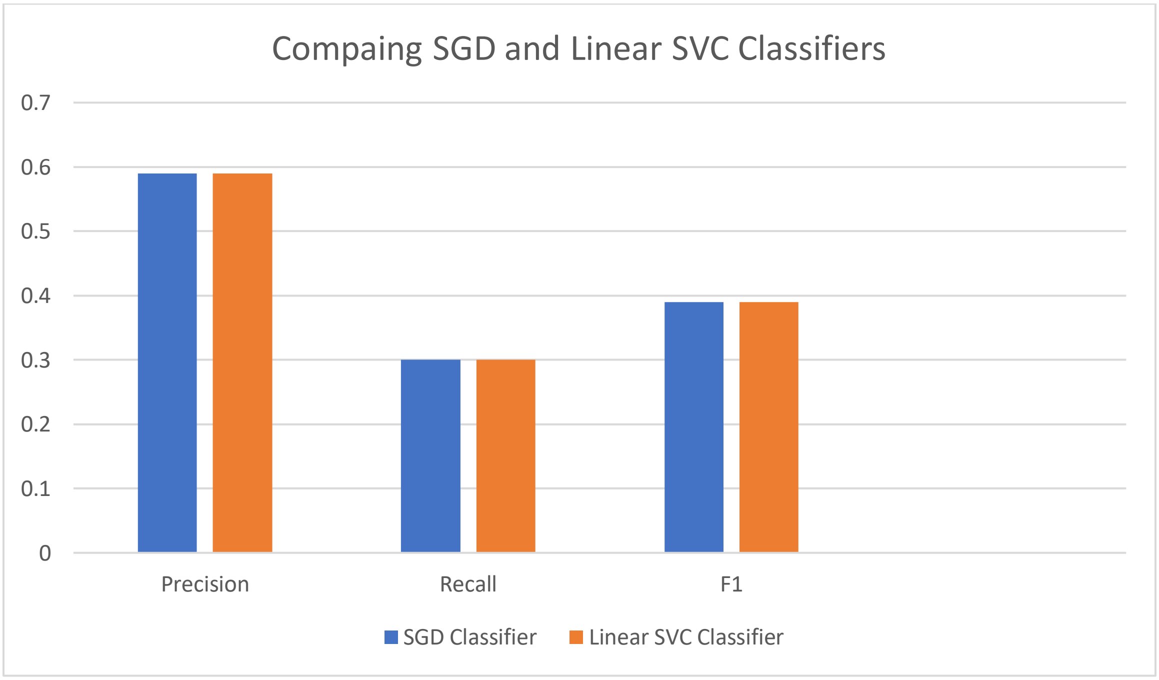

Figure 37: Using NZ data with word embeddings obtained from GloVe's pretrained embeddings with dimension of 200. The classifier uses features of embeddings of the word to be classified.

The use of different classifiers SGD and Linear SVC made little to no difference in the accuracy measurements, as seen in figure 26 there is no change to precision, recall and F1 values. Therefore, SGD or Linear SVM would both be suitable classifiers to use to compare other training data variables.

The training data of a classifier can have a big impact on its performance as small dataset may make it easier but the lack of examples the classifier can learn from can hinder its performance when using new data not from the same dataset. The three training datasets that are used to compare are a Fiji text document, which contain a small number of lines where the word 'Fiji' is also repeated numerous times, 6000 words from a large dataset designed for training an NER classifier is used and finally a 24000 word dataset of locality data for New Zealand which like Fiji has been used previously to evaluate the Stanford, SpaCy and NLTK.

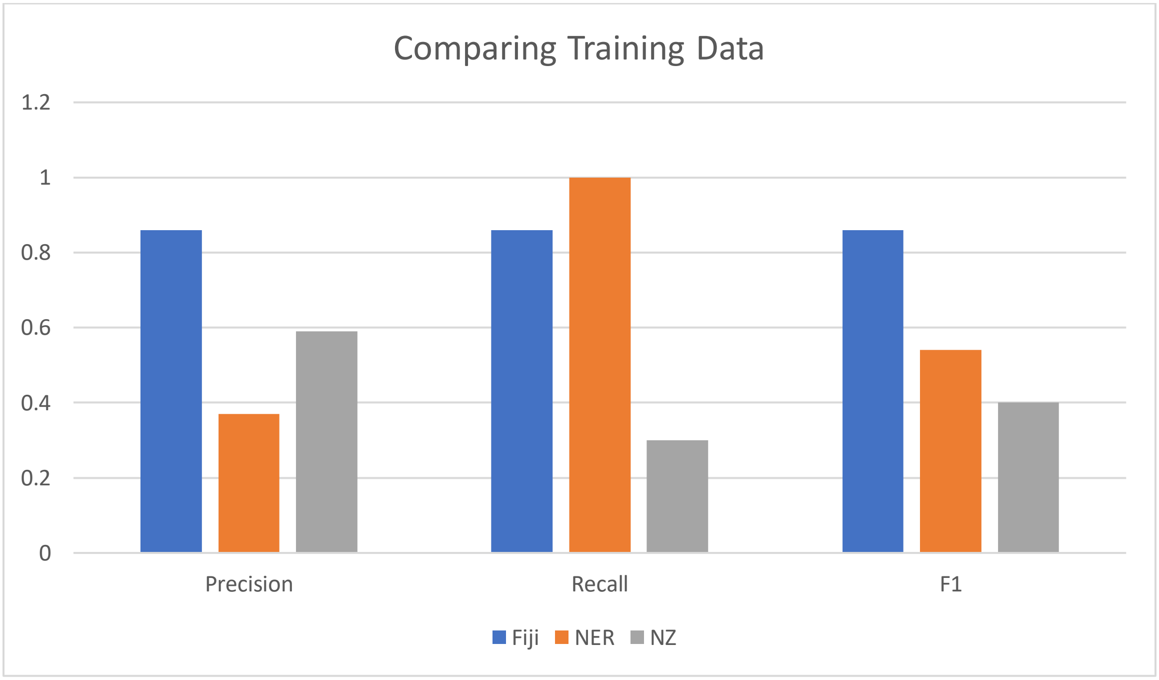

Figure 38: Comparison of using Different Training Data Using Dimensions of 200 GloVe's pretrained embeddings trained on an SGD Classifier

The precision values were better on the smaller and the largest datasets for training, as all datasets were split using Scikit Learn's train test split class where the train and test data is set to 0.25 in proportion to the input dataset it may suggest the Fiji text document is so small that the result are better as there are not many new words in the test data compared to the training data the classifier has already learned on. Recall values for NER dataset have the highest value. F1 values decrease as the size of the datasets used to train the classifier decrease.

Moving forward the New Zealand data is being used to compare the following variables to ensure it reflects the changing variable effecting results. Instead of using pre-trained embeddings to transform words to embeddings scikit learn have a count vectorizer that can be used to transform the words to word vectors and therefore train the classifier.

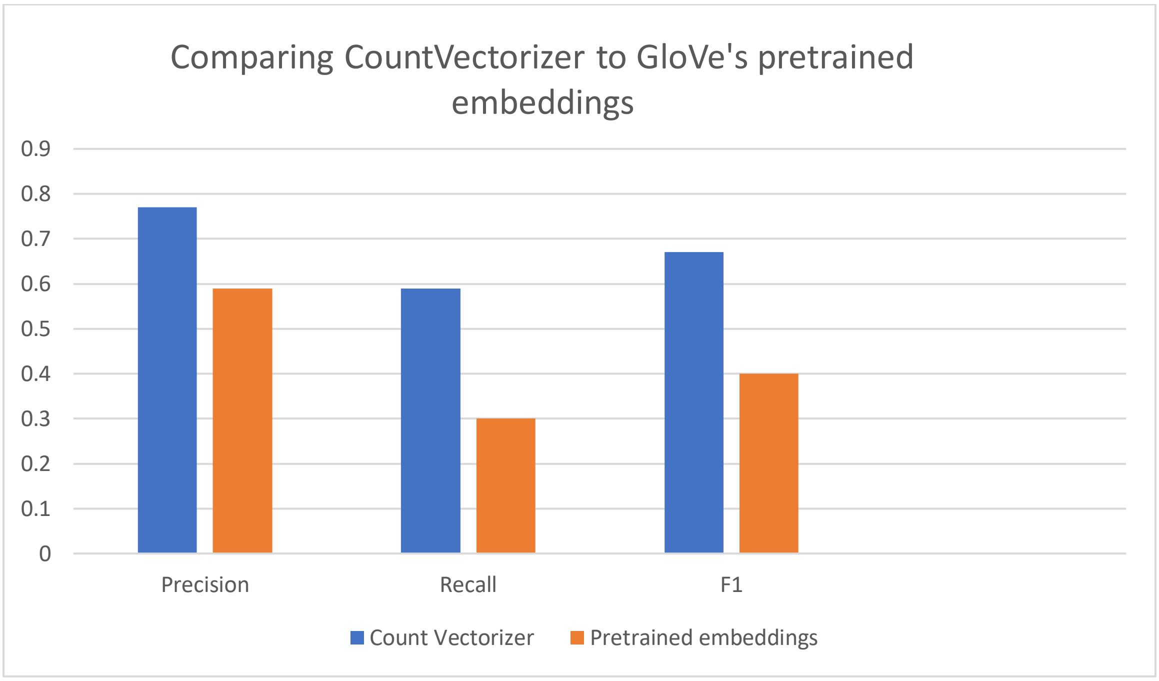

Figure 39: Comparing Count Vectorizer vs GloVe's pretrained embeddings to train an SGD Classifier both using New Zealand data for training

The training datasets appear to be not large enough that the embeddings do not perform better than count vectorizer. The New Zealand locality data has a considerable repetition of words therefore it may be easier for the classifier to learn associations between entities and context words. With this the bag of words representation could be better suited and is the reason why it performs better than pre-trained embeddings.

In future pre-trained embeddings would still better suit for classifier writing as they take time away for text pre-processing and are trained on large datasets that the programmer would not have access to. Although count vectorizers may have more dimensions than embeddings the pre-trained embeddings may be better at encapsulating the meaning within their dimensions. Another positive of using pre-trained embeddings are that the embedding feature of a word to be classified can have 200 non-zero values (for embeddings of 200 dimensions), where for the count vectorizer of the same word to be classified although there will be more dimensions there will be more zero values.

Increasing the dimensions of the embeddings can change the performance of the classifier:

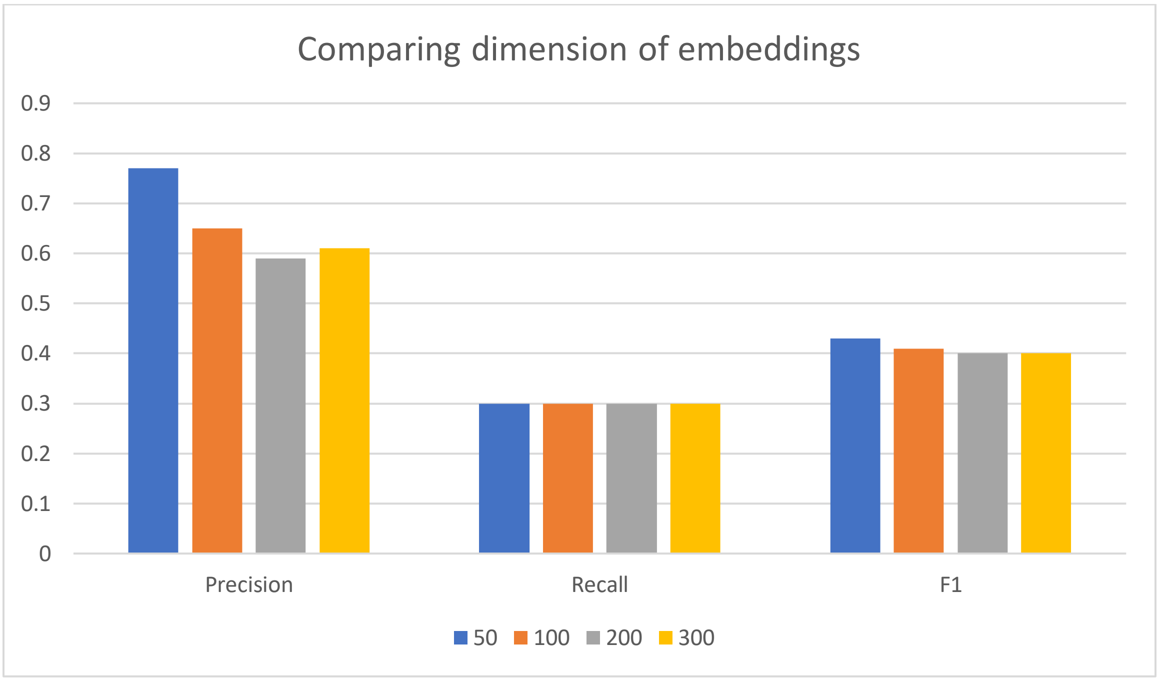

Figure 40: Comparing the different dimensions of GloVe's embeddings to train an SGD Classifier using New Zealand Data for training

Surprisingly, the results of the classifiers with increasing dimensions of embeddings are not what was expected. Precision decreases with the increasing dimensions but increase from 200 to 300 dimensions. Recall is not impacted by the increasing of the dimensions and F1 also decreases but only very slightly.

The training data is small and therefore would probably be better suited using a bag of words representation. As pre-trained embeddings may have been the poor choice for this classifier the results of changing the dimensions of the embeddings could be impacted and not produce the results that would be expected. Therefore, it may be beneficial to test the changing embeddings on larger datasets to see if the results are common or if these are anomalies as the training data is small.

Classifiers can perform better if they can capture context of a sentence, taking n number of words from either side of the word to be classified can improve the classifiers performance.

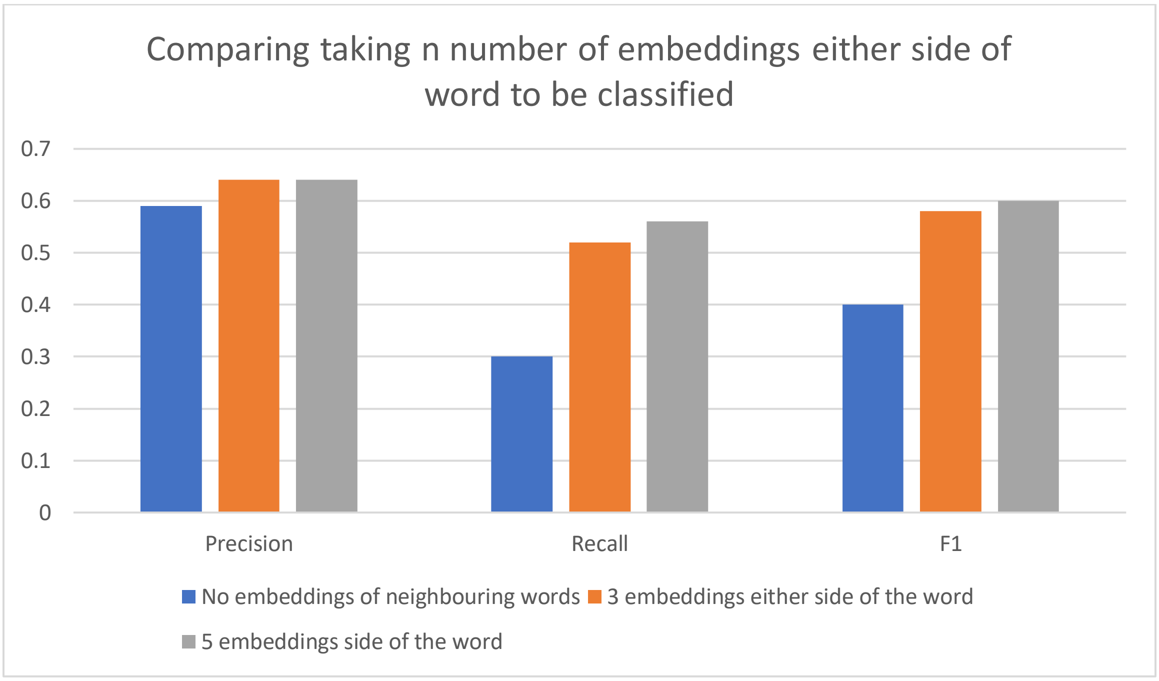

Figure 41: Comparing window sizes of embeddings around the word to be classified using GloVe's pretrained embeddings to train an SGD Classifier using New Zealand data for training

As expected, increasing the number of words taken left and right of the word to be classified increasing its accuracy performance. All three values precision, recall and f1 increased with the increasing number of words taken either side, although precision stayed the same when increased from 3 to 5 words. As this experiment would not be able to go underway without using pre-trained embeddings it demonstrates another benefit of using pre-trained embeddings for classifier training.

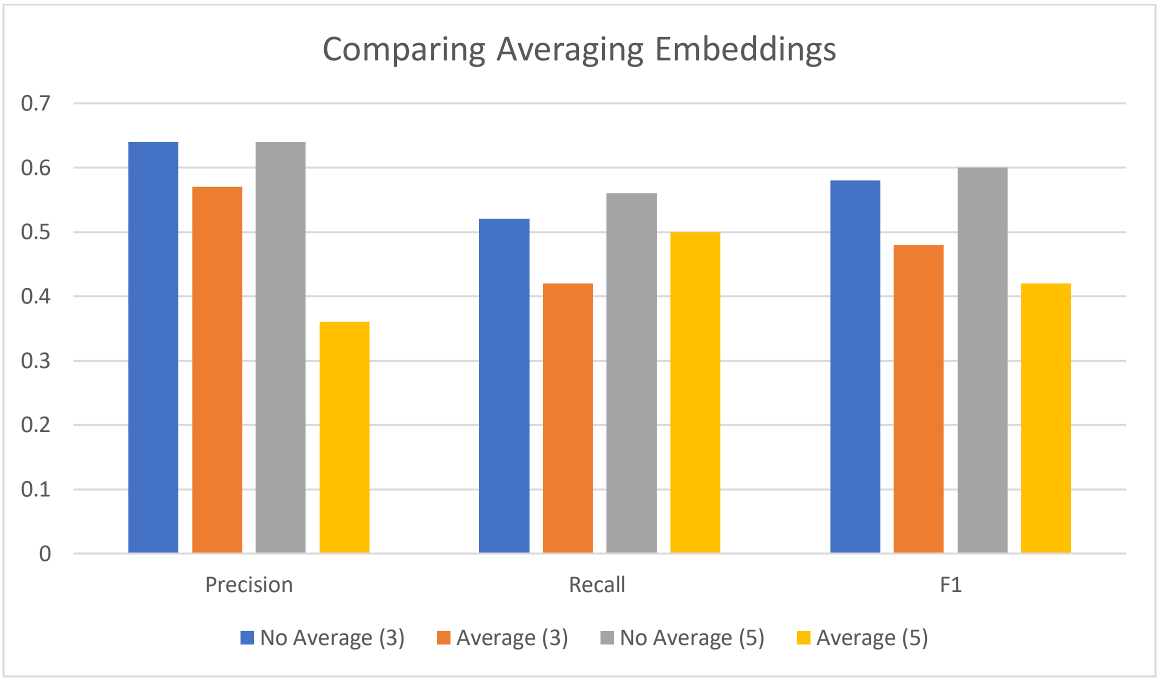

Figure 42: Comparing Averaging Embeddings vs Not-Averaging Embeddings with window size of 3 and 5 using New Zealand data for training. The Number in the bracket contains the window sized used.

Using 3 embeddings left and right of the word to be classified and the embedding of the word as an example will result in 700 data values being used for each word when training with pre-trained embeddings of dimensions of 100. When taking the average, the training data for the classifier will go from 700 data values to 100. With the less data values in training the accuracy measurements decrease. As predicted, all precision, recall and F1 values decrease when taking the average for training instead of all the values.

6.0 Future Work

The goal of this project is to 'recognise and disambiguate place names in a text document', throughout the project recognising place names has been successful and evaluating the techniques that do so. This section includes what shortcomings in the implementation have occurred and what future work could be done to further the completion of this project goal.

6.1 Training of NER Tools

To improve the technique of recognising place names in a text document further training on NER tools could be required. SpaCy gives clear tutorial on training the NER tool and update the model to perform better.

SpaCy model could be trained to improve NER on human annotated data like the locality data. Using New Zealand locality data for training which has sentences including "Whareroa Road" which can be used in training to allow the model to learn that a word with "rd", "RD" or "road" after is a location. It can use that information and the training example to remember next time it comes across a sentence such as "Mangaone Rd" to realise it is a location or a "FAC", not an organisation which SpaCy previously labelled it as. Previously, SpaCy has labelled a sentence such as "3.52 km south of Mangatoetoenui Station" as a quantity due to the presence of the 3.52, further training of the SpaCy model could result in the model recognising the location separately from the quantity. SpaCy model could also be trained to recognise locations based on the presence of spatial relations such as 'near' just like used in the example in New Zealand locality data: "NEAR ROTORUA AIRPORT". SpaCy identified "Alexandra" in the locality data as a location but also identifies it as a name, training could help by improving the SpaCy model to see that the same word did have the same entity of being a location.

To ensure the accuracy results of SpaCy are easier to compute and compare the Stanford and NLTK future work could include grouping some of SpaCy's entity labels together and creating a program to input SpaCy's training data in an easier way. As many locations such as oceans or highways are given the label 'LOC' or 'FAC' where countries and cities are given the label 'GPE' it could be argued that SpaCy is correctly identifying them as locations but in Stanford and NLTK they will all be labelled 'LOCATION'. To achieve a better comparison of the three NER tools a function could be included to group the labels that all account for a location in SpaCy to one location label. This will help compare the three tools as they will all have 'location' as a label for the same things. To calculate the accuracy measurements of SpaCy they must be in the training data form showing in 3.1.4. To input each line and entities separately takes an enormous amount of human time so an idea to make this process more elegant and faster could be to input a file of annotated data and convert it to a json file in the correct format.

6.2 Using Different Pre-Trained Embeddings

Future work on creating NER classifiers could be using different pre-trained embeddings other than GloVe used in the project examples include FastText and Word2Vec. SpaCy also have a build in embeddings that could be used to train a classifier. It would be beneficial to see how the different pre-trained embedding sources affected the accuracy performance.

6.3 Disambiguating

The task of disambiguating place names was not implemented in this project. Further work could include implementing tools to do this. The Edinburgh GeoParser is a system that can detect and disambiguate a place name with respect to a gazetteer [2010, Grover, Tobin, Bryne, Woollard, Reid, Dunn, Ball]. The geoparser can be used with several gazetteers and can be used to process a variety of input text processes. The Edinburgh Geoparser can be downloaded via The Language Technology Group website (https://www.ltg.ed.ac.uk/software/) and is described as a toolkit for georeferencing text. Like the NER tools that have been evaluated within the project The Edinburgh Geoparser could be evaluated to get accuracy measurements for its performance.

6.3 Training Classifiers using Different Features

An aim that was not implemented was to create a classifier that can detect and geo-reference a place name by the presence of biological specimens described in text. The New Zealand locality data used in the project comes from "Landcare New Zealand" which describes biological specimens that occur in places. The places are described by region, map series, map number, east coordinates, north coordinates, locality, and altitude. All this information of the biological specimens can be as features in a classifier so when the classifier sees a certain biological specimen in a text document it can recognise and geo-reference a place name from the specimens would occur.

7.0 Conclusions

The goal of the project is to 'recognise and disambiguate place names in a text document', as disambiguating was not implemented the conclusions will include the work of detecting place names. This section will conclude which of the NER tools used are the best option and whether they are better in comparison to the classifiers written.

Using the precision, recall and F1 values to compare the NER tools the conclusion is Stanford NER gives the best performance results. Stanford gives the best precision value of 0.98 in comparison of NLTK's value of 0.94 and SpaCy's value of 0.83. Stanford also achieves the best recall of 0.76 in comparison to SpaCy's 0.57 value and NLTK's 0.48 recall value. It subsequently has the best F1 value of 0.85 with SpaCy in second with 0.65 and NLTK with the worst score of 0.61.

Stanford performs NER very slowly in comparison to the other NER tools with an average of 4.34 seconds which is significantly higher than NLTK's average speed of 0.37 seconds and SpaCy's average speed of 0.12 seconds. Therefore, it is concluded Stanford would not be the best choice going forward especially as it could be dealing with exceptionally large documents in practice.

SpaCy gives high precision, recall and F1 values, allows you to train the model and performs faster than Stanford so the conclusion I have achieved is SpaCy is the best NER tool to use in the future. As it also gives a greater range of labels for the entities it could be further adapted to improve disambiguating place names in future work as roads and lakes will not be classed as a location but can be used to help decide which place name the text is discussing i.e. if the Hudson River is labelled 'LOC' by SpaCy it would be clear that the word 'York' in the same text would be referring to 'New York City' and not the English city.

Using the average of precision, recall and F1 values of the SGD Classifier trained using pre-trained embeddings and the NER Tools we can compare their performance in terms of accuracy.

When using pre-trained embeddings to train an SGD Classifier the average precision value is 0.58 which is lower than when using SpaCy with 0.83, NLTK with 0.94 and Stanford with 0.98 therefore if false positives had a very high cost then using the NER Tools would be more beneficial.

SGD Classifier trained with pre-trained embeddings gave a higher recall value of 0.62 compared to SpaCy with 0.57 and NLTK with 0.48 but did have a lower recall value than Stanford with 0.76. As recall has a higher cost in this project as false negatives can lead to the incorrect disambiguation of a place name then it would be suggested to use Stanford or the SGD classifier trained with pre-trained embeddings to reduce the amount of false negatives identified.

F1 gives a value than can be used as a substitute for the overall accuracy. The SGD classifier trained with pre-trained embeddings gave a F1 value of 0.55 which is lower than NLTK's value of 0.61, SpaCy's F1 value of 0.57, and Stanford's value of 0.85. Overall, using the F1 values gives the conclusion that the NER Tools would be preferred than the SGD Classifier trained with pre-trained embeddings.

In conclusion taking words either side of the word to be classified as input for training increases performance measurements of a classifier. The window size refers to the number of words either side of the word to be classified taken as features for training. With a window size of 0 the precision value is 0.58, increasing the window size to 3 the precision value increases to 0.60 and increases again to 0.63 when the window size increases to 5. The recall does not follow the exact same pattern as precision as with a window size of 0 the recall value is 0.62, as the window size increases to 3 the recall value increases to 0.64 but as the window size increases to 5 the recall actually decreases to 0.58. As F1 value gives a better overall value of accuracy increasing the window size does increase the F1 value. Window size of 0 gives 0.55 F1 value, then the window size of 3 gives F1 value of 0.60 and the window size of 5 gives a 0.61 F1 value. Overall, the larger the window size the greater the accuracy performance but as recall has high costs in this project it would be better to use a window size of 3 as it gives the best recall value.

In summary, on formally written text documents Stanford and SpaCy work extremely well to recognising location entity labels correctly and SpaCy works amazingly fast which gives it its edge of being the better NER. The NER tools already published work better than the current classifiers written for this project.

8.0 Reflection on Learning

Completing this final year project has been rewarding in furthering my skills in programming and project management. This project has broadened my areas of research as machine learning and natural language processing is an area of computer science, I had not previously had experience in.

8.1 Work Done

Starting this project, I had no experience in working with machine learning algorithms or natural language processing. I have managed to learn how algorithms work, how classifiers can be trained and how they are evaluated. I feel that my programming skills and understanding how to optimise libraries and imported modules have improved within this project. Although disambiguating place names was not implemented the techniques to recognise a place name was implemented successfully and code was written to evaluate them. To create a classifier using Scikit Learn library to perform NER is a massive achievement for me and a symbol of how far I have progressed in this area.

Reflecting on my progress on the project I would have started following more practical tutorials of machine learning and NLP during research to get a better idea of how the tools worked and how classifiers are trained. This could have optimised my time during classifier creation and could have possibly allowed me to gain the time to move on to disambiguating place names.

8.2 Time Management

The initial plan was ambitious in hindsight due to amount of research required. Developing a new skill in machine learning is not something that can be developed quickly but rather developed constantly over a long period of time. From the beginning of the project to now I have gained a greater understanding of machine learning classifiers and natural language processing, it was unrealistic to expect this knowledge to be gained within the time frame proposed which resulted in some aspects like disambiguating not being implemented.

The initial plan was not followed. The initial plan outlined completing the task of NER, then disambiguation and finally evaluating and creating classifiers to improve performance of the tasks. The plan changed to completing NER, evaluating NER tools and then creating classifiers to try and increase the performance measurements of NER before moving on to disambiguation. I believe it was better to change the plan as I gained valuable skills in the process of evaluating NER tools and creating classifiers for NER which would be needed when the task of disambiguation was carried out. When carrying out the first tasks in the initial plan it came apparent that further work on NER and classifiers would be required to move on in the project. I believe I am now in a position that moving forward I have the skills required if I were to carry out the rest of the project.

Skype meetings were set up regularly between myself and my supervisor to discuss project progress and what could be done next. These calls also gave a great hand in explaining machine learning processes and how NER's, embeddings and classifier training worked. These meetings allowed to be to continue to keep focus on the tasks in hand and guided me to change the plan when necessary.

9.0 Appendices

9.1 Text Documents

Below give the text documents used as data sources described in 2.12.

New Zealand:

New Zealand is a sovereign island country in the southwestern Pacific Ocean. The country has two main landmasses the North Island, and the South Island and around 6OO smaller islands. It has a total land area of 268,OOO square kilometres. New Zealand is about 2,OOO kilometres east of Australia across the Tasman Sea and 1,OOO kilometres south of the Pacific island areas of New Caledonia, Fiji, and Tonga. Because of its remoteness, it was the last large habitable landmass to be settled by humans. During its long period of isolation, New Zealand developed a distinct biodiversity of animal, fungal, and plant life. The country's varied topography and its sharp mountain peaks, such as the Southern Alps, owe much to the tectonic uplift of land and volcanic eruptions. New Zealand's capital city is Wellington, and its most populous city is Auckland.

South East England:

South East England is the most populous of the nine official regions of England at the first level of NUTS for statistical purposes. It consists of Berkshire, Buckinghamshire, East Sussex, Hampshire, the Isle of Wight, Kent, Oxfordshire, Surrey, and West Sussex. As with the other regions of England, apart from Greater London, the south east has no elected government.

Cwmbran:

Cwmbrân is a new town in Wales lying within the historic boundaries of Monmouthshire it forms part of the county borough of Torfaen Cwmbran was designated as a new town in 1949 to provide new employment opportunities in the south eastern portion of the South Wales Coalfield.

America:

The United States of America, commonly known as the United States or America, is a country consisting of 50 states, a federal district, five major self-governing territories, and various possessions O. At 3.8 million square miles, it is the world's third or fourth - largest country by total area and is slightly smaller than the entire continent of Europe I-LOC. Most of the country is located in central North America between Canada and Mexico I-LOC. With an estimated population of over 328 million, the U.S. is the third most populous country in the world O. The capital is Washington, D.C., and the most populous city is New York City.

The United States is a federal republic and a representative democracy. It is a founding member of the United Nations, World Bank, International Monetary Fund, Organization of American States, NATO, and other international organizations. It is a permanent member of the United Nations Security Council. Its President is Donald Trump.

Australia:

Australia, officially the Commonwealth of Australia, is a sovereign country comprising the mainland of the Australian continent, the island of Tasmania, and numerous smaller islands. It is the largest country in Oceania and the world's sixth-largest country by total area. The population of 26 million is highly urbanised and heavily concentrated on the eastern seaboard. Australia's capital is Canberra, and its largest city is Sydney. The country's other major metropolitan areas are Melbourne, Brisbane, Perth, and Adelaide.

Australia is the oldest flattest, and driest inhabited continent, with the least fertile soils. It has a landmass of 7,617,930 square kilometres A megadiverse country, its size gives it a wide variety of landscapes, with deserts in the centre, tropical rainforests in the north-east, and mountain ranges in the south-east. Its population density, 2.8 inhabitants per square kilometre, remains among the lowest in the world. Australia generates its income from various sources including mining-related exports, telecommunications, banking, manufacturing, and international education.

Iceland:

Iceland is a Nordic island country in the North Atlantic, with a population of 360,390 and an area of 103,000 km2, making it the most sparsely populated country in Europe. The capital and largest city is Reykjavík. Reykjavik and the surrounding areas in the southwest of the country are home to over two-thirds of the population. Iceland is volcanically and geologically active. The interior consists of a plateau characterised by sand and lava fields, mountains, and glaciers, and many glacial rivers flow to the sea through the lowlands. Iceland is warmed by the Gulf Stream and has a temperate climate, despite a high latitude just outside the Arctic Circle. Its high latitude and marine influence keep summers chilly, with most of the archipelago having a tundra climate.

Finland:

Finland Swedish : Finland, Finland, officially the Republic of Finland is a Nordic country in Northern Europe bordering the Baltic Sea, Gulf of Bothnia, and Gulf of Finland, between Sweden to the west, Russia to the east, Estonia to the south, and north-eastern Norway to the north. The capital and largest city is Helsinki. Other major cities are Espoo, Tampere, Vantaa, Oulu, Turku, Jyväskylä, Lahti and Kuopio.

South Wales:

South Wales is a loosely defined region of Wales bordered by England and the Bristol Channel to the east and south. It has a population of around 2.2 million, almost three-quarters of the whole of Wales, including 400,000 in Cardiff, 250,000 in Swansea and 150,000 in Newport. Generally considered to include the historic counties of Glamorgan and Monmouthshire, South Wales extends westwards to include Carmarthenshire and Pembrokeshire. In the western extent, from Swansea westwards, local people would probably recognise that they lived in both south Wales and west Wales. The Brecon Beacons National Park covers about a third of South Wales, containing Pen y Fan, the highest British mountain south of Cadair Idris in Snowdonia.

Southern France:

Southern France, also known as the South of France or colloquially in French as le Midi, is a defined geographical area consisting of the regions of France that border the Atlantic Ocean south of the Marais Poitevin, Spain, the Mediterranean Sea and Italy. It includes: Nouvelle-Aquitaine in the west, Occitanie in the centre, the southern parts of Auvergne-Rhône-Alpes in the northeast, Provence-Alpes-Côte d'Azur in the southeast, as well as the island of Corsica in the southeast. Monaco and Andorra are sometimes included in definitions of Southern France although they are principalities.

The term Midi derives from mi and di in Old French, comparable to the term Mezzogiorno from the Southern Italy. The time of midday was synonymous with the direction of south because in France, as in all of the Northern Hemisphere north of the Tropic of Cancer, the sun is in the south at noon. The synonymy existed in Middle French as well, where meridien can refer to both midday and south. The Midi is considered to start at Valence, hence the saying "à Valence le Midi commence".

Fiji:

Fiji, officially the Republic of Fiji is an island country in Melanesia, part of Oceania in the South Pacific Ocean about 1,100 nautical miles northeast of New Zealand's North Island. Its closest neighbours are Vanuatu to the west, New Caledonia to the southwest, New Zealand's Kermadec Islands to the southeast, Tonga to the east, the Samoas and France's Wallis and Futuna to the northeast, and Tuvalu to the north. Fiji consists of an archipelago of more than 330 islands of which 110 are permanently inhabited and more than 500 is lets, amounting to a total land area of about 18,300 square kilometres. The most outlying island is Ono i Lau. The two major islands, Viti Levu and Vanua Levu, account for 87% of the total population of 883,483. The capital, Suva, on Viti Levu, serves as the country's principal cruise ship port. About three-quarters of Fijians live on Viti Levu's coasts, either in Suva or in smaller urban centres such as Nadi where tourism is the major local industry or Lautoka, where the sugar cane industry is paramount. Due to its terrain, the interior of Viti Levu is sparsely inhabited.

Philippines:

The Philippines, officially the Republic of the Philippines, is an archipelagic country in Southeast Asia. Situated in the western Pacific Ocean, it consists of about 7,641 islands that are broadly categorized under three main geographical divisions from north to south: Luzon, Visayas and Mindanao. The capital city of the Philippines is Manila and the most populous city is Quezon City, both part of Metro Manila. Bounded by the South China Sea on the west, the Philippine Sea on the east and the Celebes Sea on the southwest, the Philippines shares maritime borders with Taiwan to the north, Japan to the northeast, Palau to the east, Indonesia to the south, Malaysia and Brunei to the southwest, Vietnam to the west, and China to the northwest.

References

2001-2019 NLTK Project. Nitin MAdnani, Rami Al-Rfou http://nltk.org

2007-2019 Scikit-learn developers https://scikit-learn.org/stable/modules/svm.html

2016 -2020 Explosion AI (3) - https://spacy.io/api/scorer

2016-2020 Explosion AI (2) https://spacy.io/usage/facts-figures

2016-2020 Explosion AI(4) - https://spacy.io/usage/training

2016-2020 Explosion AI, https://spacy.io/usage/spacy-101

ARAVIND PAI, MARCH 16, 2020 – "An Essential Guide to Pretrained Word Embeddings for NLP Practitioners" https://www.analyticsvidhya.com/blog/2020/03/pretrained-word-embeddings-nlp/

August 20, 2019 https://www.nltk.org/api/nltk.metrics.html

Bates, M (1995). "Models of natural language understanding". Proceedings of the National Academy of Sciences of the United States of America.

Bird, Steven, Edward Loper and Ewan Klein (2009), Natural Language Processing with Python. O'Reilly Media Inc.

Bold360 by LogMeIn - https://www.bold360.com/features/conversational-ai/natural-language-processing?gclid=CjwKCAiA7t3yBRADEiwA4GFlIxU_Hkfp7CfQrIVM9uHPb4sf60q93jxdfdfLoUKEjT8HX01sJ48jrhoCWxUQAvD_BwE&gclsrc=aw.ds

Claire Grover, Richard Tobin, Kate Byrne, Matthew Woollard, James Reid, Stuart Dunn, and Julian Ball. 2010b. Use of the Edinburgh Geoparser for georeferencing digitised historical collections. Philosophical Transactions of the Royal Society A, 368(1925):3875-3889

Edward Ma Jul 22, 2018 – "3 basic approaches in Bag of Words which are better than Word Embeddings" https://towardsdatascience.com/3-basic-approaches-in-bag-of-words-which-are-better-than-word-embeddings-c2cbc7398016

Europa Technologies - https://www.europa.uk.com/what-is-a-gazetteer/

Expert System Team, Blog March 2017 - https://expertsystem.com/machine-learning-definition/

Free Code Camp: 18 DECEMBER 2018 "An introduction to Bag of Words and how to code it in Python for NLP"

Goyal, Kumar, Gupta- October 2017 – International Journal of Advance Research in Science and Engineering Volume No.06, Issue No.10

IGI-Global - https://www.igi-global.com/dictionary/using-the-flipped-classroom-to-improve-knowledge-creation-of-masters-level-students-in-engineering/21327

Jason Brownlee on October 11, 2017 in Deep Learning for Natural Language Processing - What Are Word Embeddings for Text?

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation.

Jenny Rose Finkel, Trond Gregnagr and Christopher Manning. 2005 – Incorporating Non-local information into Information Extraction Systems by Gibbs Samling. Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL 2005), pp 363-370 - http://nlp.stanford.edu/~manning/papers/gibbscrf3.pdf

Karimzadeh, M., Pezanowski, S., MacEachren, A. M., & Wallgrün, J. O. (2019). GeoTxt: A scalable geoparsing system for unstructured text geolocation. Transactions in GIS, 23(1), 118–136. https://doi.org/10.1111/tgis.12510.
Shirish Kadam Mar 31, 2019 - https://medium.com/@5hirish/dependency-parsing-in-nlp-d7ade014186

Kaustumbh Jaiswal, April 18, 2019 – Custom Named Entity Recognition Using SpaCy - https://towardsdatascience.com/custom-named-entity-recognition-using-spacy-7140ebbb3718

Koo Ping Shung, March 15 2018 - https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9

Lafferty, J., McCallum, A., Pereira, F. (2001). "Conditional random fields: Probabilistic models for segmenting and labelling sequence data". Proc. 18th International Conf. on Machine Learning. Morgan Kaufmann. pp. 282–289.

March 06, 2020 Python Software Foundation https://docs.python.org/3/library/time.html

Mathew Honnibal, February 19, 2015 https://explosion.ai/blog/introducing-spacy

Oleksii Kgarkovyna, April 9, 2019 - https://towardsdatascience.com/beginners-guide-to-machine-learning-with-python-b9ff35bc9c51

Sachin Malhotra and Divya Godayal, June 2018 - https://www.freecodecamp.org/news/an-introduction-to-part-of-speech-tagging-and-the-hidden-markov-model-953d45338f24/

Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

Simone Magnolini, Valerio Piccioni, Vevake Balaraman, Marco Guerini, Bernardo Magini, 2019 – How to Use Gazetteers for Entity Recognition with Neural Models

T. Mikolov, E. Grave, P. Bojanowski, C. Puhrsch, A. Joulin. Advances in Pre-Training Distributed Word Representations

Technopedia - August 11, 2015 - https://www.techopedia.com/definition/13825/named-entity-recognition-ner