Classification and visualisation of text documents using networks

Thembani Phaweni

Thesis presented for the degree of Master of Science in the Department of Statistical Sciences at the University of Cape Town.

Supervised by A/Prof Ian Durbach, Dr. Melvin Varughese, Prof. Bruce Bassett

June 2017

Acknowledgements

I would like to acknowledge the support of the African Institute of Mathematical Sciences.

Abstract

In both the areas of text classification and text visualisation graph/network theoretic methods can be applied effectively. For text classification we assessed the effectiveness of graph/network summary statistics to develop weighting schemes and features to improve test accuracy. For text visualisation we developed a framework using established visual cues from the graph visualisation literature to communicate information intuitively. The final output of the visualisation component of the dissertation was a tool that would allow members of the public to produce a visualisation from a text document.

We represented a text document as a graph/network. The words were nodes and the edges were created when a pair of words appeared within a pre-specified distance (window) of words from each other. The text document model is a matrix representation of a document collection such that it can be integrated into a machine or statistical learning algorithm. The entries of this matrix can be weighting according to various schemes. We used the graph/network representation of a text document to create features and weighting schemes that could be applied to the text document model. This approach was not well developed for text classification therefore we applied different edge weighting methods, window sizes, weighting schemes and features. We also applied three machine learning algorithms, naïve Bayes, neural networks and support vector machines. We compared our various graph/network approaches to the traditional document model with term frequency inverse-document-frequency. We were interested in establishing whether or not the use of graph weighting schemes and graph features could increase test accuracy for text classification tasks.

As far as we can tell from the literature, this is the first attempt to use graph features to weight bag-of-words features for text classification. These methods had been applied to information retrieval (Blanco & Lioma, 2012). It seemed they could also be applied to text classification. The text visualisation field seemed divorced from the text summarisation and information retrieval fields, in that text co-occurrence relationships were not treated with equal importance. Developments in the graph/network visualisation literature could be taken advantage of for the purposes of text visualisation.

We created a framework for text visualisation using the graph/network representation of a text document. We used force directed algorithms to visualise the document. We used established visual cues like, colour, size and proximity in space to convey information

through the visualisation. We also applied clustering and part-of-speech tagging to allow for filtering and isolating of specific information within the visualised document. We demonstrated this framework with four example texts.

We found that total degree, a graph weighting scheme, outperformed term frequency on average. The effect of graph features depended heavily on the machine learning method used: for the problems we considered graph features increased accuracy for SVM classifiers, had little effect for neural networks and decreased accuracy for naïve Bayes classifiers Therefore the impact on test accuracy of adding graph features to the document model is dependent on the machine learning algorithm used.

The visualisation of text graphs is able to convey meaningful information regarding the text at a glance through established visual cues. Related words are close together in visual space and often connected by thick edges. Large nodes often represent important words. Modularity clustering is able to extract thematically consistent clusters from text graphs. This allows for the clusters to be isolated and investigated individually to understand specific themes within a document. The use of part-of-speech tagging is effective in both reducing the number of words being displayed but also increasing the relevance of words being displayed. This was made clear through the use of part-of-speech tags applied to the Internal Resistance of Apartheid Wikipedia webpage. The webpage was reduced to its proper nouns which contained much of the important information in the text.

Training accuracy is important in text classification which is a task that can often be performed on vast amounts of documents. Much of the research in text classification is aimed at increasing classification accuracy either through feature engineering, or optimising machine learning methods. The finding that total degree outperformed term frequency on average provides an alternative avenue for achieving higher test accuracy. The finding that the addition of graph features can increase test accuracy when matched with the right machine learning algorithm suggests some new research should be conducted regarding the role that graph features can have in text classification.

Text visualisation is used as an exploratory tool and as a means of quickly and easily conveying text information. The framework we developed is able to create automated text visualisations that intuitively convey information for short and long text documents. This can greatly reduce the amount of time it takes to assess the content of a document which can increase general access to information.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Natural language processing is the intersection of machine learning, statistics and computational linguistics, and is concerned with allowing computers to interpret and make use of natural human language (Joachims, 1998, 1999; Yang and Pedersen, 1997; Pang, Lee and Vaithyanathan, 2002). This field has become increasingly important since the advent of the internet and the associated abundance of text based documents. This includes online news, social media, academic articles and other forms of text documents. Finding relevant information is often difficult. Furthermore filtering through the relevant information for the most pertinent information is crucial.

There are several tasks and sub tasks in natural language processing that collaborate to improve the experience of finding digital text data. Information retrieval is synonymous with web search. A query is submitted to an information retrieval system and the system tries to return relevant results. A step in determining if the result is relevant is determining if the topic of the result is the same as that of the query. For instance a query like *banana* is about fruit, so a returned result about the news might not be relevant. Text classification systems assign labels which represent topics to documents such that information retrieval systems can know which documents are relevant and to what extent.

A web search can return several million results many of which might be relevant to the query. For those results that are relevant to the query some might not be relevant to the person performing the query. An example is a query like *box* which can have two associated topics, sport and packaging. As such methods are needed to quickly determine the content of a document without the need to read the document. The area of text

visualisation has created methods of visually displaying and analysing documents.

This dissertation tackles two main areas in text analysis/mining: text classification, and text visualization. In both the area of text classification and text visualisation graph or network theoretic methods can be applied effectively. In general the terms graph and network are used interchangeably. For text classification graph/network theory could be used to improve the accuracy of algorithms. In text visualisation graph/network theory could be used to create intuitive and summarised visualisations. The primary goals of the dissertation are to build graph/network theoretic concepts into models for text classification and visualization, and to assess their usefulness.

1.1.1 Description of the problem - text classification

Text classification is concerned with automatically assigning a single, none or multiple labels to a given document. Text classification tasks often depend on existing collections of documents which experts have already labelled, these are supervised text classification tasks. There are other tasks where labels are assigned to documents based only on their content or the content of the document collection; these are unsupervised text classification tasks. The difficulty of the unsupervised tasks is that it is not clear how to evaluate whether an algorithm has been successful or not. Supervised tasks are evaluated on the accuracy of labels assigned to unseen documents.

For supervised tasks a dataset would include a collection of documents with associated expert provided labels. The document would then be pre-processed to transform it into a document model. The document model is a format of data that easily allows a document or collection of documents to be incorporated into a learning algorithm, in particular a machine learning algorithm. Many algorithms use matrices and a common document model is the term-document-matrix. It has words in the rows and documents in the columns and word counts in the entries. Not all the words are included in the document model: some are excluded because they provide relatively little information. These words are usually auxiliary parts-of-speech examples include *is*, *and*, *to*. Other words are transformed into a form more common among related words with similar meanings, *beauty* would become *beauti* and *beautiful* would become *beauti*. Punctuation would be removed along with cardinal numbers. The last two aspects of pre-processing include extracting specific words and taking their counts in the document model.

Traditionally single words were used in each row of the term-document-matrix. In recent times this has been extended to pairs of words, triplets or even longer phrases. The benefit

of these models is that they assume less independence between consecutive words in a document. The difficulty with these document models is that they are much larger than the original term document matrix. This has computational consequences. The entries of the document model are usually counts of the word or phrase. In this sense a document is a vector over the vector space of all the words or phrases in the document collection. Words with large counts will impact the direction of the document vector the most. Some words might generally appear more than others, in which case they would influence many documents. This might have adverse effects when assigning relevant labels to documents. Salton and Buckley (1988) introduced a method called inverse-document-frequency to address this. Inverse-document-frequency multiplies the word or phrase count with the log of the total number of documents divided by the number of documents the word appears in. This is always a positive number but the less common a word is in the document collection the more amplified the word's count will be. When the entry of a document model is no longer a count it is generally referred to as a weight. The method which is used to arrive at the weight is the weighting scheme.

In machine learning it is common to improve the accuracy of an algorithm by changing the input data in some way. For text classification this can be done by selecting the words or phrases in the document model, changing the document model itself or changing the entries of the document model. Selecting the words is generally referred to as feature selection. The motivation is to select a subset of words to improve the accuracy of the model or retain the accuracy while benefiting from improvements in computational efficiency. Nicolosi (2008) and Yang & Pedersen (1997) have reviewed these methods. It is not clear if there is any consensus regarding which methods should generally be applied. It is not common to add non-word features to the document model or replace words with them, as we do in this dissertation.

Much of the focus in recent times has been on specific algorithms and the benefits they can have on test accuracy from support vector machines (see Section 3.3) (Joachims, 1998, 1999) to logistic regression (Ng & Jordan, 2001) and naïve Bayes (see Section 3.1) (Nigam et al, 2000; McCallum & Nigam, 1998). Perhaps the most ubiquitous method in text classification is Naïve Bayes. Nigam et al (2000) and McCallum & Nigam (1998) are the few authors that dedicated entire papers to the method. They developed an unsupervised version of Naïve Bayes using the Expectation Maximisation method. They went on to develop Extended Naïve Bayes which could handle both discrete and continuous input data. Logistic regression was applied by Ng & Jordan (2001) and Pang et. al. (2002). While logistic regression can give good classification accuracy, neural networks tend to perform better in general. Support vector machines are also popular. Joachims (1998,

1999) found them to be robust against outliers and able to handle large feature sets with ease. For large datasets they can be computationally expensive which inspired Platt (1999) and Fan, Chen & Lin (2005) to develop less computationally expensive methods of finding solutions. Platt was able to reduce the quadratic optimisation problem (as discussed in Section 3.3.2) to smaller linear optimisation problems.

Graph representations of documents have been found useful in information retrieval which is a closely related field of text classification (Blanco & Lioma, 2012). Graph representations of documents offer two benefits, new features for the document model and new ways of representing the existing features.

The document model and methods of manipulating the entries in the document model can be used to increase the accuracy of the machine learning algorithm applied to them. This is done by representing text documents as graphs/networks. The words in the document are represented as nodes. When they appear within a pre-specified number of words from each other an edge is created to represent this co-occurrence. The edges carry a weight which either represents the edge itself or the distance between the words. When multiple edges are created between the same pair of words these edges are summed. The final graph is created for each document. There are summary statistics in the graph theory literature that can be used to summarise complex networks, including degrees of nodes, diameters of graphs and clustering coefficients of both graphs and nodes, among other statistics (Barabási et. al., 1999a, 1999b, 2002), (Watts and Strogatz, 1998).

The text classification portion of this dissertation will assess the effectiveness of graph/network summary statistics to develop weighting schemes and features for the document model to improve the accuracy of machine learning methods applied to text classification.

1.1.2 Description of the problem - text visualisation

Text visualisation is concerned with representing text documents or the results of text analysis in a visual and sometimes interactive manner. Text visualisations are applied either to a single document or document collections. They can incorporate data external to the documents like dictionaries or thesauruses. Ultimately text visualisation is concerned with conveying complex visual information in an aesthetic and intuitive manner. In general text visualisation methods for single documents are useful because they allow for insights to be extracted from documents without the need to read the documents. Some methods display many words which makes them as difficult to read as the documents they

were produced from. Examples include Docuburst (Collins, 2009) and TextArc (Paley, 2002). Other methods like Phrase Nets (Ham, Wattenberg and Viegas, 2009) create graph representations of documents with meaningful visualisations. Yet they prefer user specified relationships which are based on part-of-speech and other linguistic patterns. There are two issues here, first one must find suitable patterns, second, these patterns have not been found useful in the text summarisation field for identifying important words for the purposes of summarisation (Kim and Kan, 2009). A meaningful text visualisation of a large document will effectively be a visual text summary of that document. The text summarisation literature does agree with the co-occurrence relationships in graph representations of text documents (Michalea and Tarau, 2004; Liu et al. 2010).

If the purpose of a text visualisation is to explore or communicate the content of a document then it should be legible. For large documents this implies that it should summarise the content in some meaningful way. A text visualisation should use visual cues to communicate information. Gist Icons (DeCamp et. al., 2005) and ThemeRiver (Havre et. al., 2002) use colour, size and spatial location well but neither show the actual words in the text in an aesthetic and visual manner. Phrase Nets convey the same information with size and colour. Colour, size and spatial location are therefore not fully utilised by existing approaches in conveying several types of information.

A framework that can use colour, size and spatial location to convey information while providing a means of summarising is an improvement over previous approaches. The framework will lend from established fields like text summarisation, graph theory and natural language processing to inform the visualisation. The visualisation framework will use a graph/network as the basis of the visualisation. This is based on the text summarisation literature where graphs have been shown to be effective in determining the importance of key words or phrases (Hasan and Ng, 2014). The words are nodes and they are connected by an edge if they appear within a user specified number of words near each other. This is the same approach used in this dissertation to create graph representations of text. The edges are weighted typically by one and the edge weights are added if multiple edges are created between the same pair of words. In the graph/network visualisation literature methods called force-directed algorithms have been designed to automatically create aesthetic visualisations of (Bannister et al, 2012). These approaches represent graphs as physical systems where nodes carry repulsive forces and edges carry attractive forces. The overall goal of the algorithms is to minimise the energy of the physical system which typically leads to an aesthetically pleasing layout of the graphs. In addition clustering approaches have been developed for graphs which are able to group related nodes (Girvan-Newman, 2002; Newman, 2006). The grouping finds

subsets of nodes with dense edge connections within the clusters and relatively sparse edge connections between the clusters. Part-of-speech tagging assigns parts of speech to individual words within a text document with the state of the art hidden Markov model method achieving 97.24% accuracy (Toutanova et al., 2003) on individual words. Both clustering and part-of-speech tagging can be used to isolate portions of text which can be explored separately. In addition the framework will allow for filtering the graph by its graph properties. As such text can be summarised or reduced in several ways.

The framework will use established visual cues such as the size, colour and position of both nodes and edges to communicate information intuitively to users. Clustering and part-of-speech tagging can identify groups of words which can be isolated for further analysis. It is believed this will create an intuitive and useful framework for exploring text documents. The final output of this visualisation component of this dissertation is a tool that will allow users and members of the public to produce a visualisation from a text document.

1.2 Clarification of key terms

Graph/network: Graphs are mathematical objects that have been used to model complex networks. The networks contain objects (nodes) between which some kind of relationships (edges) exist. The nodes are the components and the connections are represented by the edges between the components. In general the terms graph and network are used interchangeably.

Document Model: The traditional document model is the term-document-matrix. It is a matrix that has words in the rows and documents in the columns and word counts in the entries. The transpose of the term-document-matrix is the document-term-matrix. This is a more common view of data as document (observations) are in the rows and words (features) are in the columns. The features in the document model are typically words, pairs of words, triples of words or in general word tuples. These word tuples have names, uni-grams (words), bigrams (pairs of words), trigrams (triples of words) and n-grams (n-tuples).

Weighting Scheme: The document model has a number in its entries to represent the feature. The feature representation methods are referred to as weighting schemes. Common weighting schemes include term-frequency and term-frequency inverse-document-frequency (TF-IDF). There is a separation between weighting schemes and document models in the sense that TF-IDF could be applied to a bigram document model.

1.3 Scope and limitations

This dissertation is interested in assessing the usefulness of graph/network theory when applied to text classification and text visualisation. It is typical in text classification to provide a model specification that outperforms others on the basis of test accuracy. For the purposes of this dissertation the graph features and weighting schemes that can lead to higher test accuracy are of interest. As such this dissertation will not suggest a particular model but will instead suggest how graph/network theory can be used in combination with a machine learning algorithm to achieve superior test accuracy. The text visualisation Section will produce a visualisation framework and tool. The tool is aimed at assisting users and members of the public to create similar visualisations to those described in the Chapter. The tool is not able to perform all the tasks described but is intended to be used with other network visualisation software like Gephi (gephi.com).

1.4 Dissertation outline

The dissertation is presented in two mostly self-contained parts. The first part focuses on text classification and the second part of text visualisation. Both parts share the use of a graph/network representation of documents which is presented in Chapter 2. Each part contains a review of literature, a discussion of the methods and a discussion of the results. Each part concludes by summarising the main findings and making suggestions for future study.

1.4.1 Text classification outline

Chapter 2 builds on the literature introduced in Chapter 1, the introductory chapter. Chapter 2 describes the methods of creating graph representations of documents. These include the word-word graphs, word-document graphs and document-document graphs. The two latter graphs were not used in the experiments as the word-word graph was the more intuitive choice, but are described here for completeness. Graph features are described along with methods of graph based weighting schemes. Some variations in the construction of the graphs are suggested as candidates for experiment.

Chapter 3 describes the machine learning methods that are applied in the text classification experiments. These methods include naïve Bayes, support vector machines and neural networks. The Section outlines various parameter settings for neural networks and support vector machines.

Chapter 4 provides an overview of the two datasets used in the experiments, then presents the experiments and their results. The NSF dataset has a single label associated with each document. The Reuters dataset has multiple labels associated with each dataset. Results are presented for both datasets. There are 1440 results in total which are summarised over various specifications described in the chapter. The chapter concludes with a summary of the results and suggestions for future research.

1.4.2 Text visualisation summary

Chapter 5 provides a broad introduction to text visualisation, reviewing the text visualisation and summarisation literature. A hidden Markov model for part-of-speech tagging is described in detail along with graph clustering by modularity and graph visualisation by a force directed algorithm called ForceAtlas2.

Chapter 6 describes a tool created to produce visualisation from text documents called TextNet, and provides examples of text visualisation based on the framework described. These examples are based on two children's, book Fantastic Mr Fox and Cinderella, a Wikipedia page for the Internal Resistance of Apartheid and the reviews of a website, Yelp.com. This is designed to show the versatility of the method for different lengths of document. The children's books are short, the Wikipedia page relatively long and the reviews are individually short but long when combined. Each document has a different approach applied to it as example of the kind of analysis a user could use the framework for. The chapter concludes with a brief summary of the work covered, and suggests some directions for future study.

Chapter 7 contains general conclusions and recommendations for both document classification and visualization.

1.5 Remarks on notation and usage

This section provides tables of notation for this dissertation. The first table (Table 1.1) describes the notation primarily used in the chapters 2. The second table (Table 1.2) continues to content in Chapter 4. The last table (Table 1.3) describes notation mostly used in the chapters 5 to 7.

Notation	Description
sets, vectors and matrices	are bold. When an element of a vector or matrix is referred to using subscripts then the matrix or vector will not be emboldened when the element is a scalar. For instance $\mathbf{V}$ is a set. yet V_2 is a scalar element of $\mathbf{V}$. Sets, vectors and matrices can be referenced by numbers, words or both depending on the context.
$\lvert\cdot\rvert$	$\lvert\mathbf{V}\rvert$ refers to the size of $\mathbf{V}$. When $\mathbf{V}$ is a set this is the number of elements in $\mathbf{V}$. If there are 20 elements in $\mathbf{V}$ then $\lvert\mathbf{V}\rvert = 20$.
$\mathbf{I}$	sets, vectors and matrices are assumed to have an index $\mathbf{I}$. The index has the elements that can be used to reference elements. The contents of the set, vector or matrix can be summed for each element in the index as opposed to some initial and final number. Further if the index contains words instead of numbers then number indices cannot be used. For instance $\sum_{i \in \mathbf{I}} X_i$ could be equivalent to $\sum_{i}^{n} X_i$ if $\mathbf{I} \subset \mathbb{R}$. $\mathbf{I}$ is used generically in Chapter 2 as the index for a document collection defined below as $\mathbf{C}$.
$\mathbf{V}$	represents the vocabulary. The vocabulary is the set of unique words in a document collection
$\mathbf{C}$	represents a document collection or corpus in Chapter 2. $\mathbf{C}$ is a set of sets. Thus $\mathbf{C}_n$ is a set of words or a document indexed by n. $m \in \mathbf{C}_n$ is a word.
$\mathbf{W}$	represents word vectors in Chapter 2. $\mathbf{W}$ is a square matrix with two types of indices, words for the rows and numbers for the columns. $\mathbf{W}_m$ is a vector. In Chapter 3 $\mathbf{W}$ also represents a parameter matrix for a neural network. The negative notation $\mathbf{W}_{-i}$ indicates that row i that is subtracted from the matrix. The rest of the matrix remains.
$\mathbf{D}$	is a document matrix. $\mathbf{D}_n$ is a document vector which is the sum of all the word vectors indexed by $\mathbf{C}_n$.
N_n	is the number of words in a document n.
l	l is the number of nodes in a region.
τ_j	is the number of documents in class j.
$\mathcal{T}$	is the total number of documents in the corpus.

Table 1.1: Table of notation: Text Classification

Notation	Description
y	For naïve Bayes $\mathbf{y}$ is a vector such that y_n is the class label for document $\mathbf{D}_n$. For neural networks $\mathbf{y}$ is a matrix such that $\mathbf{y}_n$ is a vector with the same length as the total number of unique classes. There $\mathbf{y}_n$ is a dummy indicator for the class label of document $\mathbf{D}_n$. For support vector machines y_n is the binary indicator for the class of vector $\mathbf{x}_n$.
c	is a variable which can take on the class values of documents.
k	represents the total number of classes in the response.
superscripts	are used to refer to layers in parameter matrices for neural networks in Chapter 3.
$\mathbf{u}$	refers to a parameter matrix for neural networks in Chapter 3. $\mathbf{u}^{(i)}$ is a parameter matrix which is for layer i and is different from $\mathbf{u}^{(j)}$ when $i \neq j$.
$\mathbf{a}$	is an activated layer for a neural network. $\mathbf{a}^{(i)}$ is associated with parameter matrix $\mathbf{u}^{(i-1)}$.
$\mathbf{z}$	is the product of the previous activated layer with the current parameter matrix $\mathbf{z}^{(l)}\mathbf{a}^{(l-1)} \cdot \mathbf{u}^{(l)}$.
H, Q	these are the rows (H) and columns (Q) of the $\mathbf{z}^{(i+1)}$ matrix.
$\hat{\cdot}$	$\hat{y}$ is the prediction for y.
δ	can be considered an error vector. This is true for the final layer of a neural network as $y - \hat{y} = \delta^{(L)}$ where L represents the final layer of the neural network. This is more abstract for intermediate neural network layers.
∇	is a gradient matrix. It can be used to update its corresponding weight matrix.
γ	is the learning rate for neural networks or a placeholder for $a_1 + sa_2 = \gamma$ in the SMO algorithm for optimising the support vector machine objective function. Finally γ is a parameter for the polynomial, sigmoid and radial basis kernels.
U, W	are parameter matrices in the neural network example like $\mathbf{u}$
ω	is the parameter matrix for the support vector machines.
$\lvert \cdot \rvert$	$\lvert x \rvert$ provides the size of x when x is a vector and when its a scalar this provides its absolute value.
$\lvert\lvert \cdot \rvert\rvert$	is the function $\left[\sum_{i=1}^{\lvert x \rvert} x_i^2\right]^{\frac{1}{2}}$, this provides the absolute value of x when x is a scalar.
a	is Lagrange multiplier or a kernel parameter.
$*$	c^* is the optimal value of c.
C	the misclassification parameter for the support vector machine.
ϕ	a kernel function. There are four kernels, the linear, sigmoid, radial basis and polynomial.
K_{ij}	is the short hand for $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j)$.

Table 1.2: Table of notation: Machine Learning Methods

Notation	Description
C_t	refers to the state variable at time t.
$\mathbf{C}^{t-1}$	refers to all the preceding state variables of C_t from 1 to $t-1$.
X_t	X_t represents the observation variable at time t.
$\mathbf{X}^{t-1}$	represents all the preceding observation variables from time 1 to $t-1$.
c_t	c_t represents the realisation of a state at time t.
x_t	x_t represents the realisation of a observation at time t.
$u_i(t)$	is a short hand for the probability of state i at time t $Pr(C_t = i)$.
$p_i(x)$	is the probability of an observation given a state $Pr(X_t = x, C_t = i)$. In a hidden Markov chain every observation is assumed to depend on at least one state.
$\mathbf{P}(x)$	is a matrix where each diagonal element is of the form $p_i(x)$, where i is varied over each state.
Γ	is a state process.
δ	the steady state vector for Γ, the state process.
L	is the likelihood function of a observation sequence. For force-directed algorithms L is the length of the visualisation window.
W	is the width of the visualisation window for force-directed algorithms.
V	is the set of vertices or nodes of a graph.
E	is the set of edges of a graph.
$\mathbf{e}$	is a vector of eigenvalues under graph clustering. For force-directed layout $\mathbf{e}$ is an edge represented by a pair i, j of node indices, such that $e_1 = i$.
$\mathbf{v}$	is a matrix of eigenvectors.
v_i	is a node. The nodes can be referenced through edges $v_{e_1} = v_i$. Nodes have two properties $v_i.pos$ a vector of x and y coordinates and $v_i.disp$ the vector of x and y displacement values.
$\pi(r, c)$	represents the maximum value of $L(x_r, c)$ for different values of c for the rth value of $\mathbf{x}$.
λ	a count function where $\lambda(x, y)$ is either the count of states x and y following each other or the count of state x preceding observation y. $\lambda(x, y)$ is not in general equal to $\lambda(y, x)$.
Δ	is the difference vector in the position between two nodes such that $\delta_{ij} = v_i.pos - v_j.pos$.
f_a	f_a is a function of attractive forces in a force directed algorithm.
f_r	f_r is a function of repulsive forces in a force directed algorithm.
$\|\|x\|\|_2$	is the function $\left[\sum_{i=1}^{\|x\|} x_i^2\right]^{\frac{1}{2}}$ when x is a difference vector between two points then this is the Euclidean distance between two points. When x is a vector of values then the result is the norm of the vector x.

Table 1.3: Table of notation: Chapters 5 to 7

Chapter 2

Constructing graph-based document features

Graphs are mathematical constructs that have been used to model complex networks. The graphs contain objects (nodes) between which some kind of relationships (edges) exist. The nodes are the components and the connections are represented by the edges between the components. unipartite graphs have nodes that are similar or comparable e.g. documents. Bipartite graphs have nodes of two different types e.g. terms and documents. The edges in the unipartite graph connect similar nodes whereas the edges in the bipartite graph only connect different types of nodes. Two unipartite graphs can be formed by projecting the bipartite graph into the space of either node set (words or documents only) making up the bipartite graph (Zhou, Ren et. al. 2007). Much of this Chapter is adapted from Newman (2010). In general, the terms graph and network will be used interchangeably.

This Chapter describes unipartite (Section2.1.1) and bipartite graphs (Section 2.1.2). It describes the text and document graphs (Section 2.2) before defining the term document matrix (Section 2.3). Finally features for the term document matrix are defined along with two types of weighting schemes (Section 2.4). A benchmark weighting scheme and some graph-based weighting schemes.

2.1 Graphs

Graph-based feature construction involves the following steps:

1. Selecting an appropriate way to view the collection of text documents as one or

more graphs

2. Selecting summary statistics with which to summarize the graph constructed in the previous step.

2.1.1 Unipartite text graph

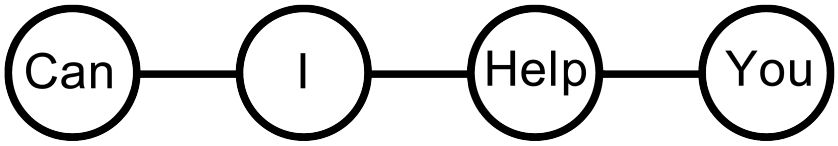

Figure 2.1: Unipartite text graph - "Can I Help You". A graph representation of the sentence with each word as a node and their immediate co-occurrence as an edge.

Can I help you?

The above sentence can be represented as a graph (Mihalcea and Tarau, 2004). Each word can become a node and edges can be created to indicate words that are next to each other in the sentence. The punctuation can be ignored. The resulting graph is shown in Figure 2.1. A graph can be reduced to an adjacency matrix. Each row and column of the adjacency matrix represents a word. Edges are considered to flow from the row to the column. Edges are indicated by positive values in the adjacency matrix. When every connection from a word to another is matched by the reverse connection then the graph is considered undirected. This also means that the adjacency matrix is symmetric along the diagonal.

$$A = \begin{array}{c} \\ Can \\ I \\ Help \\ You \end{array} \begin{array}{cccc} Can & I & Help & You \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{array}$$

2.1.2 Bipartite graphs

ID	DOCUMENT
6	Fine.
8	That book's really not good just in case, you know, browsing turned to buying. You'd be wasting your money.
9	Really?
10	Yes. This one though is... Very good.

Table 2.1: Table of documents. Each row is a document. The first column provides the column ID. The second column provides the content of the document.

A collection of four text documents is provided in Table 2.1. This can be represented as a bipartite graph (Zhou, Ren et. al. 2007), with one of the node types representing documents and the other representing words. None of the nodes of the same type are connected to each other. Figure 2.2 illustrates a portion of the Document-Text bipartite graph taken from the above text document collection. The associated adjacency matrix is provided below. Unlike the unipartite text graph this adjacency matrix is neither square nor symmetric in general (Zhou, Ren et. al. 2007).

$$A = \begin{array}{cccccc} Fine & Really & Not & Good & Very & \\ 1 & 0 & 0 & 0 & 0 & 6 \\ 0 & 1 & 1 & 1 & 0 & 8 \\ 0 & 1 & 0 & 0 & 0 & 9 \\ 0 & 0 & 0 & 1 & 1 & 10 \end{array}$$

2.1.3 Projections of the bipartite graph

The Document-Text bipartite graph can be projected into two unipartite graphs - one in which documents are nodes (Figure 2.3) and another in which words are nodes (Figure 2.4) (Newman, 2001). The document bipartite graph projection has edges between documents

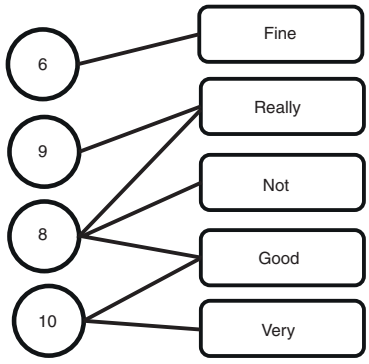

Figure 2.2: Bipartite graph of documents and words. Each circle is a document. Each rectangle is a word. The edges indicate that a word has been used within a document.

that share words. Documents that do not share words with any other document are not connected to other documents. The text bipartite graph projection has edges between words that share a document. Words that arise alone in a document are unconnected to other words.

$$A = \begin{array}{ccccc} 6 & 8 & 9 & 10 & \\ 0 & 0 & 0 & 0 & 6 \\ 0 & 0 & 1 & 1 & 8 \\ 0 & 1 & 0 & 0 & 9 \\ 0 & 1 & 0 & 0 & 10 \end{array}$$

$$A = \begin{array}{ccccc} Fine & Really & Not & Good & Very & \\ 0 & 0 & 0 & 0 & 0 & Fine \\ 0 & 0 & 1 & 1 & 0 & Really \\ 0 & 1 & 0 & 1 & 0 & Not \\ 0 & 1 & 1 & 0 & 1 & Good \\ 0 & 0 & 0 & 1 & 0 & Very \end{array}$$

2.2 Types of document graphs

There are two types of document representations that take advantage of graph representations. They are described further in the sections below.

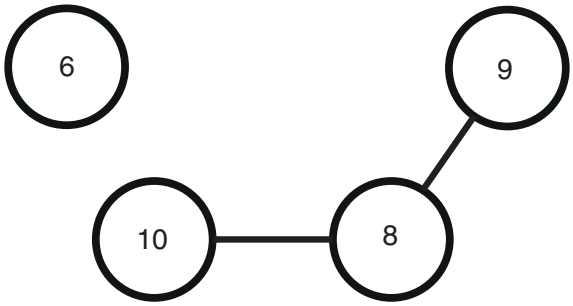

Figure 2.3: Document bipartite graph projection. When documents from Figure 2.2 share a word then they will share an edge in this projection.

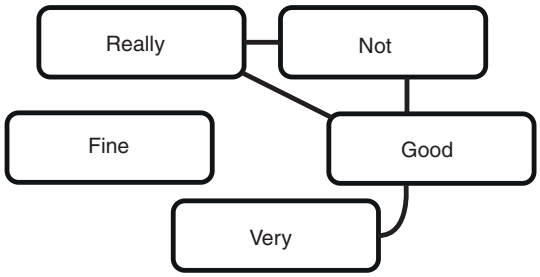

Figure 2.4: Text bipartite graph projection. When words from Figure 2.3 are used in the same document then they will share an edge in this projection.

2.2.1 Document graphs

The term-document matrix can be seen as a bipartite graph connecting terms to documents. A projection of this matrix can be taken to provide a unipartite graph of documents connected if they share terms. This graph can be analysed to provide insight on document connections in two ways.

First various characteristics of the nodes (i.e. documents) and their position in the graph could be measured. It might be expected that documents from similar classes have similar characteristics. This would suggest uniform characteristics within classes that are potentially different between classes. If this is the case, then these characteristics become features that might add more information to existing methods.

The second way is to use the graph to compute document distances. The distances would be based on path length (Floyd, 1962) as opposed to Euclidean distance. The path length from the first to the second node is the least umber of edges separating the two nodes. Documents that share terms would have small distances. Documents within similar domains might not share words but they might exist within communities that use similar vocabularies. As such they might have smaller distances than documents within very different domains. This was beyond the scope of the current dissertation.

2.2.2 Text graphs

An alternate approach to the document graphs in Section 2.2.1 is to construct a graph for each text document, in which nodes are words and an edge exists between words if they appear with fewer than a specified number of words separating them (Mihalcea & Tarau, 2004). Note that this graph is inherently unipartite, whereas the document graph in Section 2.2.1 is obtained as a document projection of the bipartite term-document graph.

In Section 2.1.1 the text graph was created from a document by connecting consecutive words. This can be rephrased as connecting words with a window of 1. This connection window can be varied to change the graphs that result (Mihalcea & Tarau, 2004).

2.2.3 Window based text graphs

Figure 2.5 and Figure 2.6 show the text graphs that result when the connection window is increased to 2 and 3 respectively. The corresponding adjacency matrices are provided below. A graph where all the nodes can be mutually reached by traveling over other nodes

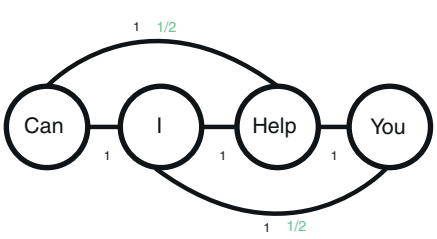

Figure 2.5: Window 2 text graph. All words that are separated by at most one word are joined by an edge. The green numbers represent distance weighting of the edges. The black numbers represent co-occurrence weighting when the edge connects words that are not adjacent. For adjacent words the black numbers represent both co-occurrence and distance edge weighting. Further details on the edge weighting can be found in Section 2.2.4.

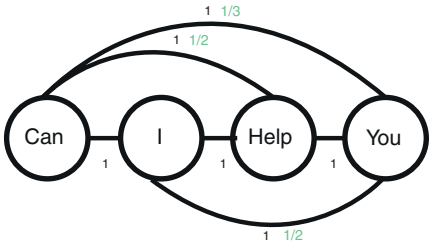

Figure 2.6: Window 3 text graph. All words that are separated by at most two word are joined by an edge. The green numbers represent distance weighting of the edges. The black numbers represent co-occurrence weighting when the edge connects words that are not adjacent. For adjacent words the black numbers represent both co-occurrence and distance edge weighting. Further details on the edge weighting can be found in Section 2.2.4

and on their edges is said to be connected (Watts & Strogatz, 1998). A graph where all the nodes are directly linked by an edge is fully connected. As the connection window is varied from 1 to 3 the text graph goes from connected to fully connected.

$$A = \begin{array}{c} \\ \\ \\ \\ \end{array} \begin{array}{cccc} Can & I & Help & You \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{array} \begin{array}{c} \\ Can \\ I \\ Help \\ You \end{array} \qquad A = \begin{array}{c} \\ \\ \\ \\ \end{array} \begin{array}{cccc} Can & I & Help & You \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{array} \begin{array}{c} \\ Can \\ I \\ Help \\ You \end{array}$$

The text graphs described in Section 2.1.2 and Section 2.2.1 allow for words in different documents to have different features. These differences could be used to discriminate between documents that might have the same words but in different configurations. Since the graph features can be specific to each document and term they can be used as feature weights.

The text graphs could have direction introduced (Mihalcea & Tarau, 2004). This can help to distinguish between terms that are used as prefixes or suffixes. This will also help to distinguish between terms that are mostly used at the beginning or end of sentences and phrases.

2.2.4 Distance based text graphs and weighted adjacency matrices

In Section 2.2.3 co-occurrence text graphs are presented with varying connection windows of two and three. We define *co-occurrence* to mean only taking into account that words appear in the same window, by providing an edge weight of one i.e. a binary indicator that does not measure the precise distance, in terms of the number of words, separating a word pair. A different approach is to use weights to represent how far away connected words are from each other. For example, the weighted adjacency matrix for the graph in Figure 2.6 with window size three is provided below. Similar approaches are followed by Mihalcea & Tarau (2004).

$$A = \begin{array}{ccccc} Can & I & Help & You & \\ 0 & 1 & \frac{1}{2} & \frac{1}{3} & Can \\ 1 & 0 & 1 & \frac{1}{2} & I \\ \frac{1}{2} & 1 & 0 & 1 & Help \\ \frac{1}{3} & \frac{1}{2} & 1 & 0 & You \end{array}$$

In the distance-weighted scheme, a pair of words separated by $w*$ intervening words obtains a weight of $\frac{1}{w*+1}$. Note that adjacent words (e.g. *can* and I) therefore obtain a weight of one. This provides two ways in which graphs can represent edges, through *co-occurence* or through *distance*.

2.3 The term document matrix

In this section some definitions are provided for the traditional document model, the term document matrix. These terms include tokenisation, vocabulary, word vector, document collection, document vector and finally term document matrix are defined (Mikolov et. al., 2010). These terms are used to describe the document models and various aspects of it. In the next section the term document matrix is extended for cases where words are represented by measures outside of their frequency (2.4). These measures are generally referred to as weights.

Tokenisation: When a text is reduced to features this process is called tokenisation (Webster & Kit, 1992). Each feature is then referred to as a token. The tokens used here will be lower case words without punctuation as demonstrated with the last sentence from Table 2.2. Tokens could be single words (unigram), pairs of words (bigram), phrases (ngrams) or sentences.

Yes. This one though is ... Very good ~ {yes, this, one, though, is, very, good}

Vocabulary: The vocabulary is a set of the unique words in a collection of documents. The vocabulary is signified by $\mathbf{V}$. For the documents from Section 2.1.2 the vocabulary would have 26 words.

$$\mathbf{V} = \{be, books, browsing, case, fine, good, ..., you, you'd, your\}$$

Word Vector: Under the matrix representation a word vector has size $|\mathbf{V}|$ with a positive number indicating the unique word the word vector represents (Joachims, 1998). The word vector is $\mathbf{W}_m$ and the location of the positive number is v such that

$$W_{m,v} > 0 \text{ and } W_{m,u} = 0 \ , \forall u \neq v$$

Using text from Section 2.1.2 *be* is the first word in the vocabulary thus its associated word vector has size 26. The first entry is a one and the remaining entries are zeroes.

$$\mathbf{W}_{be} = [1, 0, \ldots, 0], W_{be,1} = 1$$

Case is the 5th word in the vocabulary. Its word vector has an entry of one at position five and zero elsewhere.

$$\mathbf{W}_{case} = [0, 0, 0, 0, 1 \ldots, 0], W_{case,5} = 1$$

Document Collection: is a set of sets which contain the unique words from each document. Document collections are typically referred to as corpuses and corpora. An example of a two document, document collection is provided below:

$$\mathbf{C} = \{\{your, case, browsing\}, \{fine, good, food\}\}$$

A document would be a single instance of a document collection such as:

$$\mathbf{C}_1 = \{your, case, browsing\}$$

Document Vector: a document vector is a $|\mathbf{V}|$ sized vector with positive numbers at each of the indices representing the words it contains. Documents can be indexed by

Table 2.2: Term document matrix with term frequency weighting scheme. This matrix indicates the number of times each word (indicated in columns) appears in each document (indicated in rows).

fine	that	books	really	not	good	just	in	case	you	know	browsing	turned
1	0	0	0	0	0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	1	1	1	1	1	1
0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0	0	0

their location in the document collection through a subscript. A document vector in a document vector collection is indicated in the following way

$$\mathbf{D}_n = \sum_{i \in \mathbf{C}_n} \mathbf{W}_i$$

where i is the *ith* word in the document $\mathbf{C}_n$ and $\mathbf{W}_i$ is a word vector.

The fourth document (document 10) in Section 2.1.2 has the following words with associated vocabulary indices.

The vector representation of document 10 in Section 2.1.2 has 7 word vectors. At each of the above indices the word vectors have positive values.

$$\begin{aligned}\mathbf{D}_{10} &= \sum_{i \in \mathbf{C}_{10}} \mathbf{W}_i \\ &= \mathbf{W}_{yes} + \mathbf{W}_{this} + \mathbf{W}_{one} + \mathbf{W}_{though} + \mathbf{W}_{is} + \mathbf{W}_{very} + \mathbf{W}_{good} \\ &= [0,0,0,0,0,0,1,0,1,\ldots,1,0,0,0]\end{aligned}$$

Term Document Matrix: Each member of the document matrix is a $|\mathbf{V}|$ sized vector. The document matrix is $\mathbf{D}_{|I|\times|V|}$ with $|I|$ rows and $|V|$ columns for each document. Here I is the index of the document and $|I|$ is the number of documents. This matrix is the document term matrix. Its transpose is the term document matrix. A portion of the document term matrix for the documents in Section 2.1.2 is provided in Table 2.2

The definition of a word vector provided above states that the word vector has a positive number at the index representing the words position in the vocabulary. The number can be referred to as a weight. The way in which a weight is determined is a weighting scheme. Below in Section 2.4 some traditional and novel weighting schemes are provided.

2.4 Weighting schemes and graph features

The term document matrix is traditionally populated with term frequencies (Salton & Buckley, 1988; Joachims, 1998; Mikolov et. al., 2010). This can be extended to other methods of measuring the terms within the documents and these methods are generally referred to as weighting schemes. This dissertation is concerned with graph theoretic applications to improving test accuracy. Test accuracy can be increased through feature engineering. Weighting schemes fall within the category of feature engineering but so does the introduction of new features. In addition to the traditional or benchmark weighting schemes this section introduces graph-based weighting schemes as well as features.

The benchmark weighting schemes of term frequency and term frequency with inverse-document-frequency are described in Section 2.4.1. The rest of the section describes graph features and weighting schemes. First structure is considered, by decomposing a graph into its component parts features can be extracted (Section 2.4.2). Node count or the canonical size of the adjacency matrix are good measures of the extent of a graph (see Section 2.4.3). Another good measure of graph size is the diameter of the graph (Section 2.4.4). Both of these measures of size can be used as graph features. Measures of connectedness are common in network theory. Degree (see Section 2.4.5) and clustering coefficient have been used to measure the connectedness of graphs. Clustering coefficient is typically a measure of graph connectedness but through the use of neighbourhoods it can be applied to a node (Section 2.4.7). This section ends by describing how graph features are incorporated into the term document matrix.

2.4.1 Benchmark weighting scheme

The simplest weighting scheme is the term frequency scheme (Salton & Buckley, 1988). Each word receives a score of one such that $W_{mv} = 1$ and $W_{mu} = 0\ \forall u \neq v$. Thus if each component of the document vector is summed, the sum is equal to the number of words in the document.

$$\sum_{i \in \mathbf{V}} D_{ni} = N_n = |\mathbf{C}_n|$$

Arguably the most common weighting scheme is the Term-Frequency-Inverse-Document-Frequency weighting scheme introduced by Salton & Buckley (1988). Each word is provided a weight equal to the log of the corpus size divided by the total number of

Table 2.3: Term document matrix with inverse term frequency weighting scheme. This matrix indicates the number of times each word (indicated in columns) appears in each document (indicated in rows) multiplied by the inverse document frequency of the word.

fine	that	books	really	not	good	just	in	case	you	know	browsing	turned
2	0	0	0	0	0	0	0	0	0	0	0	0
0	2	2	1	2	1	2	2	2	2	2	2	2
0	0	0	1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0	0	0	0

documents containing the word multiplied by the frequency of the word. For a word vector in document n the weight is determined as follows:

$$W_{mv} = |m \in \mathbf{C}_n| \log_2 \left(\frac{|I|}{|m \in \mathbf{C}_n; \mathbf{C}_n \in \mathbf{C}|} \right) \ , \ W_{mu} = 0 \ \forall u \neq v$$

This weights every unique word in the document. Those words that are most common within the corpus receive lower weights associated with their frequency. Those words that appear infrequently in the corpus yet frequently in a document will have high weights. In general, this makes for a desirable weighting as words that are important to a document have large weights (Salton & Buckley, 1988), (Joachims, 1998).

The term frequency weighting scheme was implicitly applied when demonstrating the document-term matrix in the above section. Using the documents from Section 2.1.2 the term-frequency-inverse-document-frequency weighting scheme is demonstrated below. All words appear in either document 6 or 8. Their weights of 1 are multiplied by the inverse document frequency of the words.

$$\log_2 \frac{4}{1} = 2 \text{ and } \log_2 \frac{4}{2} = 1$$

Words like *really* and *good* that appear in two documents (see Table 2.2) and have lower weights than the other words that only appear in one document.

If *good* appeared in document eight twice then its inverse document frequency would remain the same but its weight in document eight would increase to two while it remains one in document ten.

2.4.2 Cycles

All graphs are combinations of trees and cycles (Arumugam, Hamid & Abraham 2013) where the combination can exclude either but not both. Trees are graphs such that there is a unique path between all nodes. Cycles are graphs where all nodes can be reached by moving from each node to its neighbour in a single direction. Trees cannot contain cycles otherwise there would be multiple paths between some of their nodes. When all else is equal a graph with a cycle will have a higher level of connectedness. The presence of a cycle can therefore give context to the level of connectedness.

2.4.3 Node count or canonical size of the adjacency matrix

For a text document the node count would correspond to the number of unique words. For a collection of documents, the node count would be the number of unique documents.

2.4.4 Diameter

There are numerous measures of graph size. The diameter is a measure used by Albert, Jeong & Barabasi (1999) node count of the largest connected component seems intuitive. The largest connected component is a subgraph of the graph of interest where all the nodes have a path length that is defined. All the nodes that are not connected to any node can be considered not to have a defined distance. The diameter is the largest path length between any pair of nodes (Floyd, 1962). The node count is the number of nodes in the largest connected component.

The diameter is the maximum shortest path length between any two nodes in the graph. In a tree where no cycles exist the diameter is $|\mathbf{C}_m| - 1$. The diameter of a graph with cycles is more complex. There are numerous algorithms for computing the diameter of a graph which seek to reduce the complexity of a search over all edges (e.g. Floyd, 1962).

The diameter can potentially describe style in text documents. A document that uses few words with high repetition would have a small diameter. Alternatively, a document with a large vocabulary and minimal repetition would have a large diameter. It is therefore possible that writers who produce documents that have similar diameter might have similar styles measured by the extent and use of their vocabularies.

2.4.5 Degree

The connectedness of nodes can be measured in numerous ways (Barabasi & Albert, 1999; Watts & Strogatz, 1998; Floyd, 1962). The size of the neighbourhood and density of edges in the neighbourhood of a node are intuitive measures. The neighbourhood of a node of interest can be defined as the nodes adjacent to it (Watts & Strogatz, 1998). The adjacent nodes are those connected by an edge. The number of the nodes in the neighbourhood is the degree of the node of interest. It is possible for a node to have a large degree and therefore a large neighbourhood but for the neighbourhood to consist of unrelated neighbours. This node could be considered as connecting disparate communities like a travelling sales man or a gregarious colleague. Other nodes exist in highly knit communities where neighbours are generally well connected. This node is likely very similar to its neighbours and easily identified by them. In the context of text classification measures of interconnectedness could help to uncover these characteristics of words that are not often captured by traditional feature extraction methods. Barabasi & Albert (1999) used degree and degree distributions to measure and compare the connectedness of graphs whereas Watts & Strogatz (1998) used the clustering coefficient. In this and the next two sections (2.4.7) both methods are investigated. The adjacency matrices below show a weighted (right) and unweighted (left) adjacency matrix.

$$A = \begin{array}{c|cccc} & Can & I & Help & You \\ Can & 0 & 1 & 1 & 0 \\ I & 1 & 0 & 1 & 1 \\ Help & 1 & 1 & 0 & 1 \\ You & 0 & 1 & 1 & 0 \end{array} \qquad A = \begin{array}{c|cccc} & Can & I & Help & You \\ Can & 0 & 1 & \frac{1}{2} & 0 \\ I & 1 & 0 & 1 & \frac{1}{2} \\ Help & \frac{1}{2} & 1 & 0 & 1 \\ You & 0 & \frac{1}{2} & 1 & 0 \end{array}$$

The degree of a word is the number of edges associated with it (Newman, 2001). In a traditional adjacency matrix this is the number sum along the column or the row of the word. In a weighed adjacency matrix, the same computation holds. The adjacency matrices for the window two text graphs are provided above for co-occurrence and distance edge weighting. The degrees of word are provided below (Newman, 2001).

$$1'A = [2, 3, 3, 2] \text{ and } 1'A = [1.5, 2.5, 2.5, 1.5]$$

Language has direction associated with words. For instance, the word *can* appears before *I* which appears after *help*. While co-occurrence graphs can capture the presence of words within a specified window and distance graphs the distance of words from each other directed graphs can capture their sequence. In a directed graph edges are created in a

single direction. In order for words to have a reciprocal edge they must be used in an exchanged direction. Directed graphs are not symmetric in general.

The directed versions of the above graphs are shown below. The assumed direction is forward. As the text is only moving forward and there are no repeated terms only the upper triangle is populated with edge weights.

$$A = \begin{array}{c|cccc} & Can & I & Help & You \\ Can & 0 & 1 & 1 & 0 \\ I & 0 & 0 & 1 & 1 \\ Help & 0 & 0 & 0 & 1 \\ You & 0 & 0 & 0 & 0 \end{array} \qquad A = \begin{array}{c|cccc} & Can & I & Help & You \\ Can & 0 & 1 & \frac{1}{2} & 0 \\ I & 0 & 0 & 1 & \frac{1}{2} \\ Help & 0 & 1 & 0 & 1 \\ You & 0 & 0 & 1 & 0 \end{array}$$

Degree in these graphs is computed the same way as it was above (Newman, 2001). The difference is that the resulting degree is out-degree, the number or weight of edges leaving a node.

$$1'A = [2, 2, 1, 0] \text{ and } 1'A = [1.5, 1.5, 1, 0]$$

Reversing the direction of the graph can be done by transposing the adjacency matrix. The in-degree, the number or weight of edges entering a node can be computed in the same way as above (Newman, 2001).

$$1'A = [0, 1, 2, 2] \text{ and } 1'A = [0, 1, 1.5, 1.5]$$

It is worth noting that the degree is just the sum of the in-degree and out-degree. In other words, an undirected graph is just a directed graph where the edges are made bidirectional.

2.4.6 Neighbourhoods

Adjacency matrices increase in size quadratically with respect to the number of nodes they contain. This can make visualising, analysing and recognising their substructures difficult. For this reason, graphs that relate to specific nodes can be considered in isolation. These graphs can generally be referred to as subgraphs. When a subgraph is formed by looking at a specific set of nodes and their connected nodes it is considered the neighbourhood of those nodes. Neighbourhoods that contain the nodes directly connected by an edge are discussed below.

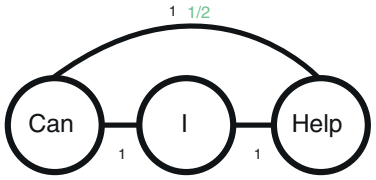

Figure 2.7: Text neighbourhood for 'Can' Undirected. All the words connected to 'Can' are in its neighbourhood. The black numbers represent co-occurrence weighting when the edge connects words that are not adjacent to 'Can'. For words adjacent 'Can' the black numbers represent both co-occurrence and distance edge weighting. The green number represent distance weighting. The edges are undirected.

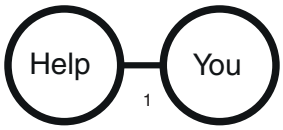

Figure 2.8: Text neighbourhood for 'Help' Directed forward. All the words proceeding 'Help' are in its forward directed neighbourhood. There is only one edge and its weight represents both distance and co-occurrence weighting.

In Section 2.2.3, co-occurrence text graphs were presented with connection windows two and three. Neighbourhoods of the graph with window two and three are provided below. The neighbourhoods are for the words *can* and *you* respectively.

The adjacency matrices for the above neighbourhoods are provided below. They are signified by S the symbol for a subgraph.

$$S = \begin{array}{cccc} Can & I & Help & \\ 0 & 1 & 1 & Can \\ 1 & 0 & 1 & I \\ 1 & 1 & 0 & Help \end{array} \qquad S = \begin{array}{ccc} Help & You & \\ 0 & 1 & Help \\ 1 & 0 & You \end{array}$$

It is worth noting that the degree of *can* in its neighbourhood is the same as it would be in the full graph. Once the region relating to each word is defined it can be used to extract features relating to that word.

2.4.7 Clustering coefficient

There are two types of clustering coefficients considered here. The clustering coefficient for the entire document and the clustering coefficient for individual words. The clustering

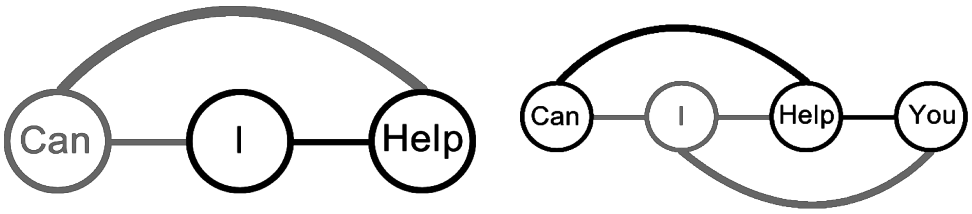

Figure 2.9: Clustering coefficient graphs 1. The left image shows the neighbourhood of 'Can'. Both 'Can' and the grey edges are excluded from the clustering coefficient computation. The right image shows the neighbourhood of 'I'. Both 'I' and the grey edges are excluded from the clustering coefficient computation.

coefficient is the number of actual edges divided by the possible number of edges among the neighbours of a node. When an entire graph is considered the average clustering coefficient of all the nodes can be used as the clustering coefficient of the entire graph (Watts & Strogatz, 1998).

The total possible number of edges is equal to the number of ways two nodes can be connected from the number of nodes in the neighbourhood $\binom{l}{2}$. Here l is the number of nodes.

Figure 2.9 and Figure 2.10 and illustrate the neighbourhoods of *can*, *I*, *help* and *you*. Only the edges among the neighbours of the respective edges are considered. For can there are two nodes and one edge. The total possible number of edges is one so the clustering coefficient is also one. For I there are three nodes with two edges. There are three ways to combine three nodes so the clustering coefficient is two-thirds. The other computations are similar.

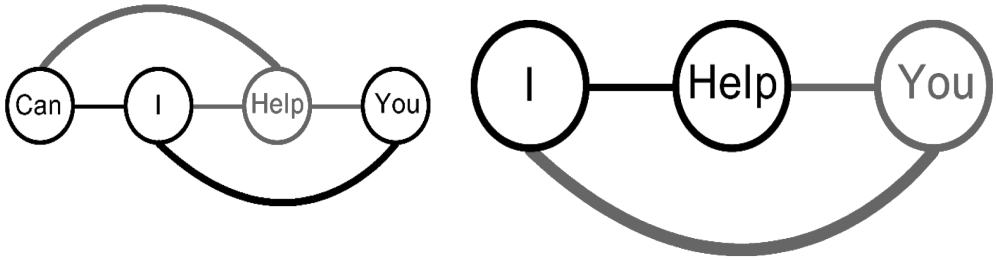

Figure 2.10: Clustering coefficient graphs 2. The left image shows the neighbourhood of 'Help'. Both 'Help' and the grey edges are excluded from the clustering coefficient computation. The right image shows the neighbourhood of 'You'. Both 'You' and the grey edges are excluded from the clustering coefficient computation.

$$
\begin{aligned}
cc(Can) &= \frac{1}{\binom{2}{2}} &&= 1 \\
cc(I) &= \frac{2}{\binom{3}{2}} &&= \frac{2}{3} \\
cc(Help) &= \frac{2}{\binom{3}{2}} &&= \frac{2}{3} \\
cc(You) &= \frac{1}{\binom{2}{2}} &&= 1
\end{aligned}
$$

The clustering coefficient function (cc) is used above. I provides the clustering coefficient of the node being referenced. the clustering coefficient of the entire graph is $\frac{1+\frac{2}{3}+\frac{2}{3}+1}{4} = \frac{\frac{10}{3}}{4} = \frac{5}{6}$

2.4.8 Incorporating the graph features into the term document matrix

There are two document models of interest: the traditional term document matrix and the term document matrix with graph features.

There were two approaches taken to including graph features in the term document matrix. The first was to add the features discussed above as if they were words i.e each graph feature would be appended to the end of the term document matrix. The alternative approach was to use the graph features directly as a weighting scheme for the words. The node count and diameter features could not be used on the word level as they refer to the entire graph. For this reason, only the degree and clustering coefficient could be used. The degree however could be divided into in and out-degree if direction is imposed on the graph. As such the final set of features used included degree, in-degree, out-degree and the clustering coefficient.

2.4.9 Remarks on weighting schemes

Term frequency measures the number of times a word appears in a document. Over a vector space of words this gives a document direction. Inverse document frequency is a measure specific to a corpus that highlights a word that is relatively frequency in a document.

In order for a graph to be used as a weighting scheme there should be a way to connect a specific word to a specific weight which is directly generated from the document. From the graph based features described above there are four weighting schemes: degree which is referred to as total degree subsequently, in degree, out degree and clustering coefficient.

In degree will assign higher weights to words that generally proceed other words whereas out degree will assign higher weights to words that precede other words. Total degree incorporates the strengths of both in and out degree. As a result the degree based methods are able to capture, perhaps in some naïve way, the structural information embedded in a document.

The clustering coefficient associated with a word captures the connectedness of the neighbourhood of the word. This is structural information not relating to the relative position of the words as with the degree measures nor the frequency of the word like term frequency.

2.5 Chapter summary

Three document models were described. A graph document model, the traditional document model with term features only and a modified document model which combines the two. Graph features were developed for the graph document model. In developing

the graph features some graph terminology was defined, including but not restricted to degree, diameter and clustering coefficient. The neighbourhood was defined which allows for graph features to be assigned to individual nodes. Further edge weighting methods like distance and co-occurrence were also defined. The graph features were used to create graph-based weighting schemes. The three document models and the associated weighting schemes or features can be integrated into machine learning algorithms as data matrices. The next chapter describes some machine learning algorithms from the literature.

Chapter 3

Review of classification methods

The classification methods or algorithms that were applied in the results chapter are described in this chapter. The classification methods include naïve Bayes, neural networks, and support vector machines.

Naïve Bayes is introduced first (Section 3.1). The method is briefly described and an example is provided on its workings. The neural network is introduced second (Section 3.2). The feed forward network is described along with backpropogation. An example follows in which the structure of the neural network is described. This is followed by a demonstration of the forward pass and backpropogation. The neural network section ends with a description of the hyper parameter setting process for the data used in Chapter 4. Finally the support vector machine is introduced (Section 3.3). This starts with an overview of the support vector classifier, a description of the optimisation problem and then the extension to both the kernel and soft-margin support vector machine. An overview of the solution to the support vector machines is then provided. Finally the section and chapter end with the hyper parameters for the support vector machines used in the experiments of Chapter 4.

3.1 Naïve Bayes

3.1.1 Description of naïve Bayes

Naïve Bayes is applied to the text classification problem by Nigam et al, (2000) and McCallum & Nigam, (1998).

The probability of a class given a document $P(c|\mathbf{D}_n)$, cannot be directly measured from the data, where c is the class and $\mathbf{D}_n$ is the document. The naïve Bayes classifier is an application of Bayes theorem that allows $P(c|\mathbf{D}_n)$ to be computed from probabilities that can be extracted from the collection of documents. These probabilities are, specifically: the relative frequency of observing each word in each class, and the overall marginal frequency of each word over the whole corpus. The goal is to use the probability of a document ($\mathbf{D}_n$) in a class (y_n) represented by ($P(c|\mathbf{D}_n)$) to classify documents into classes. The directly measurable probabilities are a word in the corpus $P(\mathbf{W}_m)$, a word in a given class $P(\mathbf{W}_m|c)$ and the document class prior $P(c)$ (Ng & Jordan, 2001). With the use of Bayes rule the posterior is defined in the following way

$$P(c|\mathbf{W}_m) = \frac{P(c) \cdot P(\mathbf{W}_m|c)}{P(\mathbf{W}_m)}$$

From the posterior the remaining challenge is to use the probability of a word in a class to arrive at the probability of a document in class. The Uni-gram, independence or bag-of-words assumption is that words in the text document are independent of each other (Lewis, 1998). Thus the joint probability of a document with respect to a class can be written as

$$P(c|\mathbf{D}_n) = \prod_{m \in \mathbf{C}_n} P(c|\mathbf{W}_m)$$

The posterior leaves only for the decision rule to be implemented. The class with the highest probability given the document is assigned (Lewis, 1998).

$$\hat{y}_n = \text{argmax}_c P(c|\mathbf{D}_n)$$

3.1.2 Discrete naïve Bayes example

The Iris dataset (Anderson, 1936 and Fisher, 1936) is available with the R software package. It has 150 observations of flowers divided into three classes. The flowers have

Table 3.1: Iris dataset. Variables values are replaced with their quartiles, with 1 being the 1st quartile (0-25 percent) and 4 being the 4th quartile (75-100 percent).

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
2	4	1	1	setosa
1	3	1	1	setosa
1	3	1	1	setosa
1	3	1	1	setosa
1	4	1	1	setosa
2	4	2	2	setosa

Table 3.2: Contingency table for sepal length with columns totals. Columns indicate quartiles, with 1 being the 1st quartile (0-25 percent) and 4 being the 4th quartile (75-100 percent).

Species	1	2	3	4
setosa	28	21	1	0
versicolor	3	18	18	11
virginica	1	2	16	31
totals	32	41	35	42

four variables Sepal Length, Sepal Width, Petal Length and Petal Width. The variables are numeric with precision to one decimal place. For the purposes of this example the numeric values have been divided into four classes each representing a quartile. This sets the stage for the example.

Sets of probabilities are needed. The probability of an observation in a class $P(c)$, the probability of a variable taking a value $P(x)$ and the probability of a variable taking a value within a specific class $P(x|c)$. This can be computed using a contingency table for each variable.

The contingency table in Table (3.2) shows the number of times Sepal Length took on specific values for all the classes of flowers and the total. There are 50 flowers of each type and 150 in total. As such each row can be divided by the total to arrive at the respective probabilities.

Such contingency tables exist for each variable which is enough to apply the naïve Bayes algorithm. The first row of the Iris dataset can be used to illustrate the computations. First the probabilities associated with each value for each variable can be extracted. The table (3.3) below shows these probabilities. Sepal length is in the second quartile for the first example which corresponds to the second column of the probability matrix for

Table 3.3: Probability matrix for sepal length with totals. Columns are quartiles. Probabilities are taken row-wise or over the quartiles.

Species	1	2	3	4
setosa	0.56	0.42	0.02	0.00
versicolor	0.06	0.36	0.36	0.22
virginica	0.02	0.04	0.32	0.62
totals	0.21	0.27	0.23	0.28

Table 3.4: Probabilities for each variable for the first observation. These probabilities represent the actual value taken on by each variable in the first observation for each class.

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
setosa	0.42	0.66	0.74	0.68
versicolor	0.36	0.04	0.00	0.00
virginica	0.04	0.16	0.00	0.00

each species. The probability matrix can be read as such: given a sepal length in the second quartile, the probability that the flower is setosa is 0.42. Similarly the probability it is versicolor is 0.36 and virginica 0.04. In a similar fashion the appropriate values are extracted from each contingency table for the remaining sepal and petal variables. The probability matrix (Table 3.4) is the result. This matrix corresponds to $P(x|c)$.

Each column of the $P(x|c)$ matrix for the first observation should be divided by $P(x)$. The Sepal Length of the first observation is two which corresponds to $P(x)$ at 0.27. Thus $0.42/0.27 \approx 1.54$ (see Table 3.5)

Subsequently each row is multiplied by the class prior of $50/150 = 1/3$ for each row. At this point each column represents a probability distribution for the variable over the classes. The columns equal one when summed. In theory one could use any of the columns to select the probability of the observation appearing in a specific class given its value for that variable. This would ignore the other variables. In order to take the other variables into

Table 3.5: Conditional probabilities for the first observation for each variable divided by marginal probabilities of the variables.

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
setosa	1.54	2.30	3	3
versicolor	1.32	0.14	0	0
virginica	0.15	0.56	0	0

Table 3.6: Probabilities for the first obvservation for each variable divided by value probabilities and multiplied by class prior

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
setosa	0.51	0.77	1	1
versicolor	0.44	0.05	0	0
virginica	0.05	0.19	0	0

account a row-wise product can be taken. This requires an assumption that the product of the individual variable probabilities is equal to the joint probability of the variables. This is the independence or Uni-gram assumption. Once computed the values are 0.39 for Setosa and 0 for the remaining flower classes. Normalised this is a 100% probability for the first class and zero for the remaining classes. This is a correct prediction as the observation has a class of Setosa.

When applied to the remaining observations this produces an accuracy rate of 97%. Only 5 observations are misclassified.

3.2 Neural networks

This section describes the feed forward neural network. This starts with a description of the feed forward process (Section 3.2.1) then the parameter training process using backpropogation (Section 3.2.2). This is followed by an example (Section 3.2.3) where the training and prediction processes are worked through. The section ends by describing the hyper parameter selection process for the neural network (Section A).

3.2.1 Feed forward networks

The neural network architecture described is the Feed Forward Neural Network (Rumelhart et al. 1988; LeCun, 1998). The Feed Forward Neural Network is a non-linear regression or classification algorithm. It was originally inspired by the functioning of the brain and organised into layers with neurons. The data starts from the first layer and moves through to the last layer. The neurons are passed through non-linear functions which behave like switches activating the neurons. The functions are therefore referred to as activation functions. The neurons are parameters and their parameter values are provided in a weight matrix $\mathbf{u}$. Each layer has a bias neuron which does not receive data from the previous layer. This is represented as an additional parameter in the weights matrix and a vector

of ones in the data from the previous layer. The back-propagation algorithm described below is used to train the parameter values using gradient descent.

The input layer is defined as $\mathbf{a}^{(1)}$. This input layer accepts the values from the data $\mathbf{D} \in \mathbb{R}^{n \times |V|}$ where $n = |I|$. The output layer is defined as $\mathbf{a}^{(L)}$ where there are L layers in total. The $\mathbf{a}$ signifies an activated layer. The layer index is i and the weight or parameter matrices for each layer are $\mathbf{u}^{(i)}, i \in [1, L]$. The dimensions of the weight matrices are such that there are as many rows in the matrix for the current layer as there are neurons in the previous layer. The columns in each matrix are the neurons. The neural network is therefore described by the equations below. The activation functions used are the sigmoid and softmax functions as defined below. The sigmoid function reduces the range of inputs from $(-\infty, \infty)$ to $(0, 1)$. The softmax function normalises over the inputs. The softmax function ensures that the final output values lie between 0 and 1 and sum to 1. As such they can be interpreted as probabilities for an observation lying within a given class (Bengio et. al., 2003).

Other authors, Bengio et. al. (2003), Mikolov et. al. (2010) pre-multiply feature vectors with parameter matrices. If the dataset is tall this will lead to wide outputs from the neural network. Post-multiplying by the parameter matrix leads to tall output from tall data. These authors also explicitly exclude the bias from the parameter matrices but this has little practical significance. As such the bias is included in the parameter matrices. When the parameter matrices are being post multiplied, then the bias is in the top row of the parameter matrix. This also assumes that the bias is added to the first column of the activated layer. This is explicitly shown in the example in Section 3.2.3.

The first layer accepts values from the data, which is a document matrix ($\mathbf{a}^{(1)} = \mathbf{D}$). Iteratively the data is adjusted by a weight matrix ($\mathbf{z}^{(i+1)} = \mathbf{a}^{(i)} \cdot \mathbf{u}^{(i+1)}$) and then activated $f(\mathbf{z}^{(i+1)})$ by some differentiable function f.The activation functions either highlight or suppress certain aspects of the previous layer (Le Cun et. al., 1998).

$$\mathbf{a}^{(i+1)} = f(\mathbf{a}^{(i)} \cdot \mathbf{u}^{(i+1)}) = f(\mathbf{z}^{(i+1)}) = \sum_{h=1}^{H} \sum_{q=1}^{Q} f(\mathbf{z}_{hq}^{(i+1)}) \text{ with } \mathbf{a}^{(1)} = \mathbf{D}$$

The data from the second to last layer $(L-1)$ is passed through a final activation function g which is typically different from those used in the hidden layers (Mikolov et. at., 2010).

$$\mathbf{a}^{(L)} = g(\mathbf{a}^{(L-1)} \cdot \mathbf{u}^{(L)}) = g(\mathbf{z}^{(L)}) = \sum_{i=1}^{n} \sum_{c=1}^{k} g_c(\mathbf{z}_i^{(L)})$$

The learning task requires a cost function, here the cross entropy function is shown.

$$Cost(\mathbf{a}^{(L)}, \mathbf{y}) = \frac{1}{n} \sum_{i=1}^{n} \sum_{c=1}^{k} \left[y_{ic} \ln(a_{ic}^{(L)}) + (1 - y_{ic}) \ln(1 - a_{ic}^{(L)}) \right]$$

A common neural network activation function is the sigmoid function (ref). The sigmoid function accepts a range of values anywhere in the real numbers and restricts them to lie between zero and one.

$$sigmoid(z) = f(z) = \frac{1}{1 + e^{-z}}$$

The softmax function accepts a vector of values and restricts each element of the vector a value between zero and one.

$$softmax_m(\mathbf{z}) = g_m(\mathbf{z}) = \frac{e^{z_m}}{\sum_{c=1}^{k} e^{z_c}}$$

This section continues to discuss the backpropagation algorithm used to train the neural networks. In addition, the parameter settings for the neural networks are described along with some fine tuning to reduce overfitting and prevent under and overflow errors.

3.2.2 Backpropagation

The learning of neural network parameters is through a method of gradient descent (LeCun, 1998) referred to as backpropagation (Rumelhart et al. 1988; Srivastava et al., 2014; Bengio et. al.,2003). This method minimises the cost function provided by propagating the error from the output layer through to the first hidden layer. In order to simplify the computations the gradients of the sigmoid and softmax functions are first expressed.

$$\begin{aligned} \frac{df(z)}{dz} &= -\frac{e^{-z}}{(1 + e^{-z})^2} \\ &= \frac{e^{-z}}{1 + e^{-z}} \left(\frac{-1}{1 + e^{-z}} \right) \\ &= f(z)\,(1 - f(z)) \end{aligned}$$

$$\begin{aligned}
\frac{\partial g_m(\mathbf{z})}{\partial z_m} &= \frac{e^{z_m}\sum_{c=1}^{k} e^{z_c} - e^{z_m}e^{z_m}}{(\sum_{c=1}^{k} e^{z_c})^2} \\
&= \frac{e^{z_m}}{\sum_{c=1}^{k} e^{z_c}} \left(\frac{\sum_{i=1}^{k} e^{z_k} - e^{z_m}}{\sum_{i=1}^{k} e^{z_c}} \right) \\
&= g(z_m)\left(1 - g(z_m)\right)
\end{aligned}$$

The traditional gradient descent method is to compute the gradient of parameters contributing to error and use this gradient to update those parameters. In the neural network framework this would correspond to computing the gradient of the weight matrices and updating them with respect to the error. Below the computation of the gradients are provided for the output layer and hidden layers (Rumelhart et al. 1988).

$$\begin{aligned}
\frac{\partial Cost}{\partial \mathbf{u}^{(L)}} &= \frac{\partial Cost}{\partial \mathbf{a}^{(L)}} \frac{\partial \mathbf{a}^{(L)}}{\partial \mathbf{u}^{(L)}} \\
&= \frac{1}{n}\sum_{i=1}^{n} \left[\mathbf{y}_i - \mathbf{a}_i^{(L)}\right] \frac{g'(\mathbf{a}^{(L-1)}\mathbf{u}^{(L)})}{\mathbf{a}^{(L)}(1-\mathbf{a}^{(L)})} \\
&= \frac{1}{n}\sum_{i=1}^{n} \left[\mathbf{y}_i - \mathbf{a}_i^{(L)}\right] \frac{g(\mathbf{a}^{(L-1)}\mathbf{u}^{(L)})(1 - g(\mathbf{a}^{(L-1)}\mathbf{u}^{(L)}))}{\mathbf{a}^{(L)}(1-\mathbf{a}^{(L)})} \mathbf{a}^{(L-1)} \\
&= \frac{1}{n}\sum_{i=1}^{n} \left[\mathbf{y}_i - \mathbf{a}_i^{(L)}\right] \mathbf{a}^{(L-1)}
\end{aligned}$$

The error is the difference between the observation and prediction (Mikolov et. al., 2010). This definition is sufficient for the final output but for the purposes of backpropagation it needs to be extended to include hidden layers and their output. This is because it is not immediately clear what the error in the hidden layers would be. Since gradient descent is used to minimise the cost the chain rule can be used to find and attribute error to specific parameters in any layer. In the final layer this error is expressed as $\delta^{(L)}$ which simplifies to the average error vector $\frac{1}{n}\sum_{i=1}^{n}\left[\mathbf{y}_i - \mathbf{a}_i^{(L)}\right]$. The gradient of the weight matrix for the final layer (L) is $\nabla^{(L)}$. In general $\mathbf{a}^{(l-1)}$ and $\delta^{(l)}$ are both vectors. Their outer product is taken to create the matrix $\nabla^{(l)}$ which is of the same dimension as its corresponding weight / parameter matrix $\mathbf{u}$ (Rumelhart et al. 1988).

$$\begin{aligned}\nabla^{(L)} &= \frac{\partial Cost}{\partial \mathbf{u}^{(L)}} \\ &= \frac{\partial Cost}{\partial \mathbf{a}^{(L)}} \frac{\partial \mathbf{a}^{(L)}}{\partial \mathbf{u}^{(L)}} \\ &= \delta^{(L)} \mathbf{a}^{T(L-1)}\end{aligned}$$

Similar expressions to those above can be created for the hidden layers using L as a superscript without loss of generality as it can be interchanged with any superscript without affecting the expression. It is worth noting that a general expression for $\delta^{(L-1)} := \mathbf{u}^{(L)} f'(\mathbf{a}^{(L-2)}\mathbf{u}^{(L-1)})$ is defined for all hidden layers up to the first hidden layer. The input layer has no weights to update therefore there is no error to be propagated to it.

$$\begin{aligned}\nabla^{(l-1)} &= \frac{\partial \mathbf{z}^{(l)}}{\partial \mathbf{u}^{(l-1)}} \\ &= \mathbf{u}^{(l)} \frac{\partial \mathbf{a}^{(l-1)}}{\partial \mathbf{z}^{(l-1)}} \frac{\partial \mathbf{z}^{(l-1)}}{\partial \mathbf{u}^{(l-1)}} \\ &= \mathbf{u}^{(l)} f' \left(\mathbf{a}^{(l-2)} \mathbf{u}^{(l-1)} \right) \mathbf{a}^{T(l-2)} \\ &= \delta^{(l-1)} \mathbf{a}^{T(l-2)}\end{aligned}$$

In the final step of backpropagation, the weight matrices are updated by using the gradient. Typically, a fraction of the gradient is applied which is referred to as the learning rate denoted below by γ.

$$\mathbf{u}^{(l)} := \mathbf{u}^{(l)} - \gamma \cdot \nabla^{(l)}$$

3.2.3 A neural network example

The following example starts by discussing the data and the task (Section 3.2.3.1), which in this case will be classification. Next the neural network architecture that is used in the example is described (Section 3.2.3.2). This architecture is arbitrarily selected. Determining the structure of a neural network is discussed in Section A. The forward pass is shown (Section 3.2.3.3), once the model is trained this is synonymous with prediction. Finally the training of the model is shown (Section 3.2.3.4) using the error from the forward pass to improve the accuracy of the next forward pass.

3.2.3.1 Data, response and task

For this example the Iris dataset (Anderson, 1936 and Fisher, 1936) is used. The features are separated into a matrix $\mathbf{x}$. The cross entropy cost function has been used effectively in classification tasks (Bengio et. al., 2003). The cost function is presented below. The summation is up to three as there are three species in the dataset and each one receives a prediction probability.

$$Cost(\mathbf{a}^{(3)}, \mathbf{y}) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{c=1}^{3}[y_{ic}\log(a_{ic}^{(3)}) + (1 - y_{ic})\log(1 - a_{ic}^{(3)})]$$

3.2.3.2 Structure of the neural network

For this example a single hidden layer with 5 neurons is used. The parameter matrix of the hidden layer is $\mathbf{U} \in \mathbb{R}^{5\times5}$. It accepts four parameters from $\mathbf{x} \in \mathbb{R}^{150\times4}$ plus a bias. It then produces five neurons that go to the output layer. The parameter matrix for the output layer is $\mathbf{W} \in \mathbb{R}^{6\times3}$. It accepts six parameters from the hidden layer, five from $\mathbf{U}$ with a bias. The output layer produces three values, one for each species. Both matrices are initialised randomly with a uniform distribution between 0 and 1.

3.2.3.3 Forward pass

The training of a neural network requires a forward pass of the data through the parameters and then a backward propagation of error to update the parameter matrices. In this section the forward pass is demonstrated. There are five neurons in the hidden layer. Before these values are activated they are represented by $\mathbf{z}^{(2)}$ which is the product of the input with a bias multiplied by the parameter matrix for the hidden layer U

$$\begin{aligned}
\mathbf{z}^{(2)} &= \mathbf{a}^{(1)} \cdot \mathbf{U} \\
&= [1\ \mathbf{x}_1] \cdot \mathbf{U} \\
&= \begin{bmatrix}1 & 5.1 & 3.5 & 1.4 & 0.2\end{bmatrix} \begin{bmatrix}0.96 & 0.94 & 0.24 & 0.26 & 0.39 \\ 0.34 & 0.45 & 0.29 & 0.45 & 0.81 \\ 0.61 & 0.36 & 0.77 & 0.05 & 0.59 \\ 0.20 & 0.63 & 0.40 & 0.29 & 0.64 \\ 0.64 & 0.99 & 0.93 & 0.49 & 0.57\end{bmatrix} \\
&= \begin{bmatrix}5.23 & 5.6 & 5.15 & 3.22 & 7.59\end{bmatrix}
\end{aligned}$$

The output of the hidden layer is passed through an activation function f which in this case is the sigmoid function. This creates $\mathbf{a}^{(2)}$.

$$\begin{aligned}\mathbf{a}^{(2)} &= f(\mathbf{z}^{(2)}) \\ &= \begin{bmatrix} f(\mathbf{z}_1^{(2)}) & f(\mathbf{z}_2^{(2)}) & f(\mathbf{z}_3^{(2)}) & f(\mathbf{z}_4^{(2)}) & f(\mathbf{z}_5^{(2)}) \end{bmatrix} \\ &= \begin{bmatrix} 0.99 & 1 & 0.99 & 0.96 & 1 \end{bmatrix}\end{aligned}$$

The five neurons that are activated are now passed through the output layer ($\mathbf{W}$) with a bias.

$$\begin{aligned}\mathbf{z}^{(3)} &= \begin{bmatrix} 1 & \mathbf{a}^{(2)} \end{bmatrix} \cdot \mathbf{W} \\ &= \begin{bmatrix} 1 & 0.99 & 1 & 0.99 & 0.96 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0.99 & 1 & 0.99 & 0.96 & 1 \end{bmatrix} \begin{bmatrix} 0.75 & 0.99 & 0.43 \\ 0.12 & 0.59 & 0.21 \\ 0.71 & 0.68 & 0.99 \\ 0.66 & 0.44 & 0.72 \\ 0.03 & 0.67 & 0.54 \\ 0.82 & 0.28 & 0.46 \end{bmatrix} \\ &= \begin{bmatrix} 3.09 & 3.62 & 3.32 \end{bmatrix}\end{aligned}$$

The output $\mathbf{z}^{(3)}$ is activated through g the softmax function to create $\mathbf{a}^{(3)}$. This output is a distribution over the possible classes. At this point this distribution would predict the class Versicolor yet the first data point is belongs to Setosa.

$$\begin{aligned}\mathbf{a}^{(3)} &= g(\mathbf{z}^{(3)}) \\ &= \begin{bmatrix} g_1(\mathbf{z}^{(3)}) & g_2(\mathbf{z}^{(3)}) & g_3(\mathbf{z}^{(3)}) \end{bmatrix} \\ &= \begin{bmatrix} 0.25 & 0.43 & 0.32 \end{bmatrix}\end{aligned}$$

3.2.3.4 An example of backpropogation

The purpose of back propagation is to use the errors to update the parameter matrices to improve the accuracy of the neural network (LeCun, 1998; Rumelhart et al. 1988). This requires extracting the derivatives of each layer and using them to update the weights of the parameter matrices. Starting at the back $\delta^{(3)} = \mathbf{y}_i - \mathbf{a}^{(3)}$ is computed. This allows for $\nabla^{(3)}$ the gradient of the output layer parameter matrix to be determined. A similar process follows for the hidden layer with slight changes to the computation of the $\delta^{(2)}$ vector. The objective is to minimise the cost which after the initial forward pass is 2.32.

$$
\begin{aligned}
\delta^{(3)} &= \mathbf{y}_1 - \mathbf{a}^{(3)} \\
&= \begin{bmatrix} 1 - 0.25 & 0 - 0.43 & 0 - 0.32 \end{bmatrix} \\
&= \begin{bmatrix} 0.75 & -0.43 & -0.32 \end{bmatrix}
\end{aligned}
$$

The first step is to compute the $\delta^{(3)}$ vector. Then $\delta^{(3)}$ is used to compute $\nabla^{(3)}$ the matrix of values that will be used to update the output layer parameter matrix.

$$
\begin{aligned}
\nabla^{(3)} &= \begin{bmatrix} 1 \\ \mathbf{a}^{(2)} \end{bmatrix} \cdot \delta^{(3)} \\
&= \begin{bmatrix} 1 \\ 0.99 \\ 1 \\ 0.99 \\ 0.96 \\ 1 \end{bmatrix} \begin{bmatrix} 0.75 & -0.43 & -0.32 \end{bmatrix} \\
&= \begin{bmatrix} -0.75 & 0.43 & 0.32 \\ -0.74 & 0.43 & 0.32 \\ -0.74 & 0.43 & 0.32 \\ -0.74 & 0.43 & 0.32 \\ -0.72 & 0.41 & 0.31 \\ -0.75 & 0.43 & 0.32 \end{bmatrix}
\end{aligned}
$$

A similar process follows for the hidden layer. There are parameters in the output layer that do not depend on the hidden layer. The derivatives of these parameters with respect to the hidden layer are therefore zero. For this reason these parameters are removed from the matrix. The row or column that is removed has a negative index. As such $\mathbf{W}_{-1}$ is the output layer matrix without the first row of weights. With this notation, the previously computed $\delta^{(3)}$ and the derivative of the sigmoid function f', $\delta^{(2)}$ can be computed.

$$
\begin{aligned}
\delta^{(2)} &= f'(\mathbf{z}^{(2)}) \circ (\delta^{(3)} \cdot \mathbf{W}_{-1}^T) \\
&= \begin{bmatrix} f'(\mathbf{z}_1^{(2)}) & f'(\mathbf{z}_2^{(2)}) & f'(\mathbf{z}_3^{(2)}) & f'(\mathbf{z}_4^{(2)}) & f'(\mathbf{z}_5^{(2)}) \end{bmatrix} \\
&\circ \\
&\begin{bmatrix} \begin{bmatrix} 0.75 & -0.43 & -0.32 \end{bmatrix} \begin{bmatrix} 0.12 & 0.71 & 0.66 & 0.03 & 0.82 \\ 0.59 & 0.68 & 0.44 & 0.67 & 0.28 \\ 0.21 & 0.99 & 0.72 & 0.54 & 0.46 \end{bmatrix} \end{bmatrix} \\
&= \begin{bmatrix} 0.005 & 0.004 & 0.006 & 0.037 & 0.001 \end{bmatrix} \begin{bmatrix} 0.23 & 0.08 & -0.08 & 0.44 & -0.35 \end{bmatrix} \\
&= \begin{bmatrix} 0.12 & 0.03 & -0.04 & 1.64 & -0.02 \end{bmatrix} \times 0.01
\end{aligned}
$$

Determining the gradient matrix $\nabla^{(2)}$ uses the same basic formulation as $\nabla^{(3)}$.

$$
\begin{aligned}
\nabla^{(2)} &= \begin{bmatrix} 1 \\ \mathbf{x}_1 \end{bmatrix} \cdot \delta^{(2)} \\
&= \begin{bmatrix} 1 \\ 5.1 \\ 3.5 \\ 1.4 \\ 0.2 \end{bmatrix} \begin{bmatrix} 0.12 & 0.03 & -0.04 & 1.64 & -0.02 \end{bmatrix} \\
&= \begin{bmatrix} 0.12 & 0.03 & -0.04 & 1.64 & -0.02 \\ 0.62 & 0.14 & -0.23 & 8.36 & -0.09 \\ 0.43 & 0.10 & -0.16 & 5.74 & -0.06 \\ 0.17 & 0.04 & -0.06 & 2.29 & -0.02 \\ 0.02 & 0.01 & -0.01 & 0.33 & 0.00 \end{bmatrix} \times 0.01
\end{aligned}
$$

The final step in the back propagation process is updating the parameter matrices. The ∇ matrices are subtracted from the parameter matrices after being adjusted by a learning rate γ which is 0.01 in this case. If the same data is passed through the new parameter matrices the cost comes down to 2.31. However only one observation has been used in a dataset of 150. When incorporating the remaining data there are three main approaches. Stochastic gradient descent, batch gradient descent or some mixture of the two. Stochastic gradient descent would repeat the above process for each observation individually. Batch gradient descent would pass all the data through the forward pass, compute the average error and use that to update the parameter matrices. This would be repeated for several iterations. A mixture of the two approaches would be to use batches smaller than the entire dataset. These batches are referred to as mini-batches. Stated in terms of mini-batch sizes, stochastic gradient descent has a mini-batch size of 1 while batch gradient descent has a mini-batch size of the entire dataset.

$$
\begin{aligned}
\mathbf{W} &= \mathbf{W} - \gamma \cdot \nabla^{(3)} \\
\mathbf{U} &= \mathbf{U} - \gamma \cdot \nabla^{(2)}
\end{aligned}
$$

Various parameters have been arbitrarily selected in this example. This includes the number of neurons in the hidden layer, the activation functions used in the hidden layer, the size of the mini-batches and the learning rate itself. The parameter selection process for the neural network implementation is discussed in Appendix A.

3.3 Support vector machines

3.3.1 Overview of support vector machines

Support vector machines separate observations into different classes by creating a decision boundary between them. They were originally developed by Vapnik (1982), Boser, Guyon and Vapnik (1992), Cortes and Vapnik (1995), with subsequent development from Platt (1999), Chang & Lin (2011) and Fan, Chen and Lin (2005). Figure 3.1 shows flowers by their petal width and height. The Setosa species is coloured in red and the remaining species in black. These species are clearly linearly separated by the width and height of petals. This is the Iris dataset (Anderson, 1936 and Fisher, 1936). A support vector machine can determine the equation of the line in blue that separates the classes. The solid blue points are support vectors. They are the closest points from each dataset to the decision boundary.

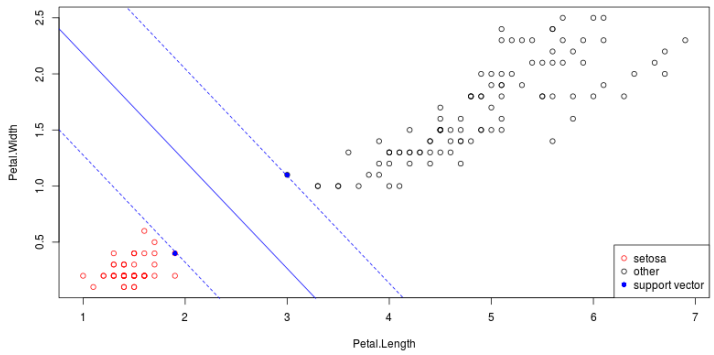

Figure 3.1: Linear kernel example. The Iris dataset with a linear SVM fit to separate Setosa from the other classes over the dimensions of petal length and petal width.

The support vectors are the only points that determine the equation of the decision boundary. This makes support vector machines robust against overfitting when there are many input features (Joachims, 1998, 1999).

The decision line is typically fit over more than two dimensions and generally referred to as a hyperplane (Boser, Guyon and Vapnik, 1992). The introduction of kernels allows the hyperplane to be non-linear, this is described in Section 3.3.2. The hyperplane cannot distinguish between multiple classes, it can only distinguish between two classes. Various strategies are applied to converting the binary problem to a multi-class problem. An example of such a strategy is *1 vs all* where the Setosa species in Figure 3.1 is compared to

all the others. This can be done in turn for each species. Each model can then be used to predict if a flower comes from the model's corresponding species or not. If multiple models predict that a flower belongs to their class then a heuristic can be applied to determine which prediction should hold.

This section starts by describing how the hyperplane is found as an optimisation problem. Extensions to the non-linear (Section 3.3.3) and imperfect separating boundary (Section 3.3.4) are presented. An outline of how the optimisation problem is solved follows (Section 3.3.5. Finally some hyper parameters (Section 3.3.6) for the models used in Chapter 4 are provided.

3.3.2 Introducing the optimisation problem

Given that a dataset has linearly separable classes there are infinitely many decision boundaries that can separate the classes. We chose the decision boundary to be the one that is furthest from each class (Boser, Guyon and Vapnik, 1992). This decision boundary has the least complexity and is therefore the least susceptible to overfitting. This requires that we find the decision boundary with no classification error that has the maximum distance to the closest points of either class. The equation of the boundary is given by $f(\mathbf{x}) = \omega^T\mathbf{x} + b = 0$ for $\mathbf{x}$ on the decision plane. The same line can be given by $g(\mathbf{x}) = c(\omega^T\mathbf{x} + b) = 0$, where c is a scalar value. Using the example in Figure 3.1 the blue line is the decision boundary. If there was another axis, that was orthogonal to the Petal.Width and Petal.Length dimensions then the decision plane $f(\mathbf{x}) = z = \omega^T\mathbf{x} + b$ would cut through the 3-dimensional space and create the decision boundary at the point where $z = 0$. The points in the Petal.Width, Petal.Length space are restricted to lie on the plane $z = 0$. As such the angle of the decision plane along Petal.Width x_1, or Petal.Length x_2, can change while the equation of the decision boundary remains the same (Cortes & Vapnik, 1995).

In other words ω can be chosen such that $\omega^T\mathbf{x}_m + b \geq 1$ for $\mathbf{x}_m$ above the decision boundary and $\omega^T\mathbf{x}_n + b \leq -1$ for $\mathbf{x}_n$ below the decision boundary. As the classification task is supervised and binary each observation is labelled and we can assume the negative label $y_n = -1$ and the positive label $y_m = 1$ (Boser, Guyon and Vapnik, 1992). Since the classes are perfectly separable the $f(\mathbf{x}) = \omega^T\mathbf{x} + b > 0$ for observations with $y_m = 1$ and $f(\mathbf{x}) < 0$ for observations with $y_n = -1$. ω is chosen such that the closest observations to the decision boundary are at a distance of 1. This results in the following constraints for all data points not on the decision boundary.

$$y(\omega^T \mathbf{x} + b) \geq 1$$

The distance from a point to the decision boundary is given by the projection of that point to a vector orthogonal to the decision plane. It is shown below that for an arbitrary point on the decision plane the inner product of that point with ω will be zero, which implies that ω is orthogonal to the decision plane.

$$\begin{aligned} \omega^T \mathbf{x}' + b &= 0 \\ \omega^T \mathbf{x}'' + b &= 0 \\ &\Rightarrow \\ \omega^T (\mathbf{x}' - \mathbf{x}'') + b &= 0 \quad \text{for } \mathbf{x}', \mathbf{x}'' \text{ on decision plane} \end{aligned}$$

The closest points to the decision boundary are said to lie on the margin. The margin is the distance from the decision boundary to the closest point of each class (Cortes & Vapnik, 1995). The points that lie on the margin have equality constraints such that $\omega^T \mathbf{x}_m^* = 1 - b$ and $\omega^T \mathbf{x}_n^* = -1 - b$. The margin can be computed using the difference vector from a point on the margin to a point on the decision boundary.

$$|margin| = \left| \left(\frac{\omega}{||\omega||} \right)^T \cdot (\mathbf{x}_m - \mathbf{x}) \right| = \left| \frac{1}{||\omega||} \cdot [1 - b - (0 - b)] \right| = \frac{1}{||\omega||}$$

Therefore the distance between the points that lie on opposite sides of the boundary can be computed by projecting the difference vector between $\mathbf{x}_m$ and $\mathbf{x}_n$ onto ω normalised.

$$\left| \left(\frac{\omega}{||\omega||} \right)^T \cdot (\mathbf{x}_m - \mathbf{x}_n) \right| = \left| \frac{1}{||\omega||} \cdot [1 - b - (-1 - b)] \right| = \frac{2}{||\omega||}$$

Either the margin or the distance between the closest points of each respective class can be maximised by minimising their inverse (Cortes & Vapnik, 1995). It is the parameters that maximise the margin that are of interest. Therefore $||\omega||$ can be squared making the problem quadratic and proportional to the boundary but without changing the parameters (Cortes & Vapnik, 1995). The optimisation problem is

$$\begin{aligned} &\min \frac{1}{2} ||\omega||^2 \\ \text{s.t } &y_i(\omega^T \mathbf{x}_i + b) - 1 \geq 0 \end{aligned}$$

The optimisation problem can be re-expressed as a single equation containing the inequality constraints. When the equation is optimised it obeys the inequality constraints. The equation is known as the Lagrange primal and is expressed below:

$$L(b,\omega) = \frac{1}{2}||\omega||^2 - \sum_{i=1}^{n} a_i \left[y_i \left(\omega^T \mathbf{x}_i + b \right) - 1 \right]$$
$$\text{s.t } 0 \leq a_i$$

The Lagrange primal is solved over b and ω. The derivatives of each of the variables defining the decision boundary can be computed and substituted back into the Lagrange primal to reduce the form to a function of the Lagrange multipliers (a_i).

$$\frac{\partial}{\partial \omega} L(b,\omega) = \omega - \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i = 0 \Rightarrow \omega = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$$
$$\frac{\partial}{\partial b} L(b,\omega) = -\sum_{i=1}^{n} \alpha_i y_i = 0 \Rightarrow 0 = \sum_{i=1}^{n} \alpha_i y_i$$

Substituting the partial derivatives into the Lagrange produces the dual which is optimised with respect to α (Boser, Guyon & Vapnik, 1992).

$$\begin{aligned} L(\alpha) &= \frac{1}{2}|| \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i ||^2 - \sum_{i=1}^{n} \alpha_i \left[y_i \left(\sum_{j=1}^{n} \alpha_j y_j \mathbf{x}_j \mathbf{x}_i + b \right) - 1 \right] \\ &= \sum_{i=1}^{n} a_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_j \mathbf{x}_i \end{aligned}$$

After optimising the dual what remains is to provide the equation for the decision boundary. The decision boundary requires b and ω. Specifically ω is the sum of all the points that have non-zero Lagrange multipliers with their class labels $\sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$ (Cortes & Vapnik, 1995). The points that have non-zero Lagrange multipliers sit on the boundary and are known as support vectors. These are the only points that determine the equation of the decision boundary.

$$\omega^* = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i, \text{ when } a_i > 0, \mathbf{x}_i \text{ is a support vector}$$

After computing ω^*, b can be solved for any support vector with the equation of the constraint

$$y_i(\omega^{T*}\mathbf{x}_i + b) = 1 \Rightarrow b^* = \frac{1}{y_i} - \omega^{T*}\mathbf{x}_i$$

The resulting prediction is

$$\hat{y}_i = \text{sign}(\omega^{T*}\mathbf{x}_i + b^*)$$

This explains how the decision boundary can be computed for perfectly linearly separable data. There are cases in practice when the data is either non-linearly separable or not perfectly separable at all. In these cases two extensions to the support vector machine are necessary. The first is the kernel support vector machine. This is a method of creating non-linear decision boundaries, it is discussed in Section 3.3.3. The second is the soft-margin support vector machine, this support vector machine allows for classes that cannot be perfectly separated by the decision boundary. In this scenario the training accuracy is less than 100%. The soft-margin support vector machine is discussed in Section 3.3.4.

3.3.3 The kernel SVM

Kernels play an important role for support vector machines. They allow for a support vector machine to fit a non-linear decision boundary (Boser, Guyon & Vapnik, 1992). To illustrate some data was generated and plotted in Figure 3.2. The green points were generated from a normal distribution with mean zero and variance two. Values were paired into points and all the points falling within the unit circle were excluded. The black points were generated from a normal distribution with mean one and variance one. The points were paired and only the points falling within the unit circle were kept.

green points $x_1 \sim$ N(0,2), $x_2 \sim$ N(0,2) where each point is (x_1, x_2) excluding points within $x_1 \leq \sqrt{1 - x_2^2}$

black points $x1 \sim$ N(1,1), $x_2 \sim$ N(1,1) where each point is (x_1, x_2) only including points within $x_1 \leq \sqrt{1 - x_2^2}$

In the original space it is clear that the data cannot be separated into its classes with a linear boundary. When the data is plotted in the kernel space (referred to as dual space by (Boser, Guyon & Vapnik, 1992) with radial basis kernel and $\sigma = 1$, it seems that a linear boundary could separate the data.

If a linear support vector machine is fit on the right hand image it becomes clear that a linear boundary can separate the data in the kernel space. What is more, the linear

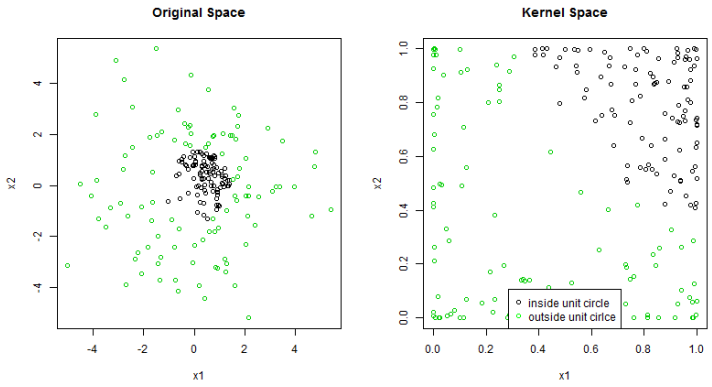

Figure 3.2: Radial kernel example. Both images show simulated data. The green points were generated from a normal distribution with mean zero and variance two. All the points falling within the unit circle were excluded. The black points were generated from a normal distribution with mean one and variance one. Only the points falling within the unit circle were kept. The right hand image passes this data through a radial basis kernel with $\sigma = 1$.

boundary can be drawn in the original data space as shown in the right hand image of Figure 3.3. The linear boundary in the kernel space becomes a circular boundary in the original data space.

It is not necessary to fit a linear support vector machine in the kernel space, this can be done directly by incorporating the kernel into the Lagrange dual problem.

$$L(\alpha) = \sum_{i=1}^{n} a_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \mathbf{x}_j \Rightarrow$$
$$L(\alpha) = \sum_{i=1}^{n} a_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

Here $K(\mathbf{x}_i, \mathbf{x}_j)$ is the kernel. Each point is passed through the kernel function $\phi(\mathbf{x})$ and the inner product of the result of the kernel functions is computed. In this way the support vector machine is computed in the kernel space.

$$\omega^* = \sum_{i=1}^{n} a_i y_i \phi(\mathbf{x}_i) \text{ when } a_i > 0 \text{ , } \phi(\mathbf{x}_i) \text{ is a support vector}$$

And

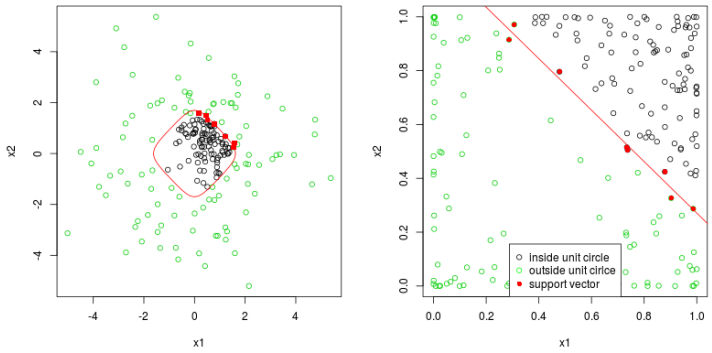

Figure 3.3: Radial basis example - fitted SVM. This image shows data from Figure 3.2. A linear support vector machine is fit in the kernel space (radial basis kernel with $\sigma = 1$) in the right hand image. The decision boundary is drawn in both images, demonstrating how a non-linear decision boundary is found.

$$b^* = \frac{1}{y_r} - \omega^{T*}\phi(\mathbf{x}_r)$$

Therefore the prediction rule is

$$\hat{y}_r = \text{sign}(\omega^{T*}\phi(\mathbf{x}_r) + b^*)$$

The kernel mapping is not necessary when computing the decision boundary (Cortes & Vapnik, 1995) it can be computed directly with the use of the kernel such that

$$\hat{y}_r = f(\mathbf{x}_r) = \text{sign}\left(\sum_{i=1}^{n} a_i y_i K(\mathbf{x}_i, \mathbf{x}_r) + b^*\right)$$

Where

$$b^* = y_m - \sum_{i=1}^{n} a_i y_i K(\mathbf{x}_i, \mathbf{x}_m) \text{ , } \mathbf{x}_i, \mathbf{x}_m \text{ are support vectors}$$

A further discussion on the kernels used continues in Section 3.3.6.

3.3.4 The soft-margin SVM

In practice it is possible that a dataset will not be perfectly separable and therefore the training accuracy will be less than 100% (Cortes & Vapnik, 1995). This requires an adjustment to the optimisation problem to allow for misclassified samples. The restriction $|\omega^T \mathbf{x}_i + b| \geq 1$ suggests that the closest points to the decision boundary are at $|\omega^T \mathbf{x}_i + b| = 1$. The soft margin support vector machine should allow for points even closer to the decision boundary such that they cross it. Therefore the adjustment ξ_i is added such that the restriction is adjusted for some points as follows $|\omega^T \mathbf{x}_i + b| \geq 1 - \xi_i$. This restriction is then included in the minimisation problem as stated:

$$\begin{aligned} \min \frac{1}{2}||\omega||^2 + C\sum_{i=1}^{n} \xi_i \\ \text{s.t } y_i(\omega^T \mathbf{x}_i + b) - 1 + \xi_i \geq 0 \\ \xi_i \geq 0 \end{aligned}$$

The Lagrange primal follows

$$\begin{aligned} L(b, \omega, \xi) = \frac{1}{2}||\omega||^2 + C\sum_{i=1}^{n} \xi_i - \sum_{i=1}^{n} a_i \left[y_i \left(\omega^T \mathbf{x}_i + b \right) - 1 + \xi_i \right] + \sum_{i=1}^{n} \beta_i \xi_i \\ \text{s.t } 0 \leq a_i, 0 \leq \beta_i \end{aligned}$$

When the derivatives of ω, b and ξ_i are:

$$\begin{aligned} \frac{\partial}{\partial \omega} L(b, \omega, \xi) = \omega - \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i = 0 \Rightarrow \omega = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \\ \frac{\partial}{\partial b} L(b, \omega, \xi) = -\sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i = 0 \Rightarrow 0 = \sum_{i=1}^{n} \alpha_i y_i \\ \frac{\partial}{\partial \xi_i} L(b, \omega, \xi) = C - a_i - \beta_i \Rightarrow a_i = C - \beta_i \end{aligned}$$

Both a_i and β_i are restricted to be above zero which suggests that a_i should remain between 0 and C in order to respect both restrictions. Therefore the Lagrange dual can be written:

$$L(\alpha) = \sum_{i=1}^{n} a_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_j \mathbf{x}_i \Rightarrow$$
$$L(\alpha) = \sum_{i=1}^{n} a_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_j, \mathbf{x}_i)$$
$$\text{s.t. } 0 \leq a_i \leq C$$

As the value of C tends to infinity the soft-margin support vector machine becomes more sensitive to misclassified points. At the point where it cannot admit misclassified samples it is effectively the same as the original hard-margin support vector machine. The soft-margin support vector machine can also supports kernel spaces. With the use of the soft-margin support vector machine and kernels it is possible to perform binary classification on datasets with classes that are neither perfectly nor linearly separable.

3.3.5 Solving for the Lagrange multipliers

$$L(\alpha) = \sum_{i=1}^{n} a_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

Maximising the Lagrange function of $L(\alpha)$ is a quadratic optimisation (Cortes & Vapnik, 1995) problem where $K(\mathbf{x}_i, \mathbf{x}_j)$ produces a symmetric positive semi-definite matrix and is restricted such that each value is above zero. This problem can be solved using traditional quadratic optimisation techniques however there are more computationally efficient ways of solving the optimisation problem for support vector machines. Platt (1999) introduced a method for solving the Support Vector Machine by solving for two Lagrange multipliers at a time. He called the method Sequential Minimal Optimisation. The algorithm has three components, select the Lagrange multipliers, update them and constrain them. Assuming the Lagrange multipliers have been selected the objective function is reduced to a function of two variables.

$$\begin{aligned} L(a_1, a_2) = \gamma - s a_2 + a_2 - \frac{1}{2} K_{11} (\gamma - s a_2)^2 \\ - \frac{1}{2} K_{22} a_2^2 - s K_{12} (\gamma - s a_2) a_2 \\ - y_1 (\gamma - s a_2) v_1 - y_2 a_2 v_2 + W_{constant} \end{aligned}$$

This function is optimised with respect to a_2, $s = y_1 y_2$, $y_1, y_2 \in \{-1, 1\}$ (Platt, 1999).

$$\begin{aligned}\frac{\partial}{\partial a_2}L(a_1,a_2) &= K_{11}(\gamma - sa_2) - s + 1 \\ &\quad - K_{22}a_2 + sK_{12}a_2 - sK_{12}(\gamma - sa_2) \\ &\quad + y_2v_1 - y_2v_2 + W_{constant} \\ &\quad \Rightarrow a_2^{new}(K_{11} + K_{22} - 2K_{12}) = s\gamma(K_{11} - K_{12}) + y_2(v_1 - v_2) - s + 1\end{aligned}$$

given

$$\begin{aligned}\gamma &= a_1^{old} + sa_2^{old} \\ v_1 &= f^{old}(\mathbf{x}_1) + b - y_1a_1^{old}K_{11} - y_2a_2^{old}K_{21} \\ v_2 &= f^{old}(\mathbf{x}_2) + b - y_1a_1^{old}K_{12} - y_2a_2^{old}K_{22}\end{aligned}$$

we can conclude

$$\begin{aligned}a_2^{new}(K_{11} + K_{22} - 2K12) &= a_2^{old}(K_{11} + K_{22} - 2K12) \\ &\quad + y_2(f^{old}(\mathbf{x}_1) - y_1 - f^{old}(\mathbf{x}_2) + y_2)\end{aligned}$$

The above derivation can be simplified by allowing $E_i = f(\mathbf{x}_i) - y_i$ and $\eta = K_{11} + K_{22} - 2K_{12}$ (Platt, 1999).

$$a_2^{new} = a_2^{old} - \frac{y_2(E_1 - E_2)}{\eta} = a_2^{old} - \frac{y_2(f^{old}(\mathbf{x}_1) - y_1 - (f^{old}(\mathbf{x}_2) - y_2))}{K(\mathbf{x}_1,\mathbf{x}_1) + K(\mathbf{x}_2,\mathbf{x}_2) - 2K(\mathbf{x}_1,\mathbf{x}_2)}$$

It is possible for this derivative to lead to infeasible updates with respect to the constraint that $0 \leq a_1 \leq C$ (Platt, 1999). To correct for this some boundaries are added. The line segment $a_1 = y_1(\xi - a_2y_2)$ is placed within a box restricted such that the above constraint is kept. The lower bound of the box is increased for a_1 to the line segment. The upper bound for a_2 is decreased to the line segment. The orientation of the line segment is determined by the class labels and as described below.

$$\begin{aligned}&\text{if } y_1 \neq y_2 \Rightarrow a_1 - a_2 = y_1\xi \quad \Rightarrow a_2 \in [\min(0, a_1 - a_2), \max(C, C + a_1 - a_2)] \\ &\text{if } y_1 = y_2 \Rightarrow a_1 - a_2 = y_1\xi \quad \Rightarrow a_2 \in [\min(0, a_1 + a_2 - C), \max(C, a_1 + a_2)]\end{aligned}$$

The new value of a_2^{new} of the target Lagrange multiplier is then restricted to the values described above.

$$a_2^{new,restricted} = LowerLimit \leq CurrentValue\ a_2^{new} \leq UpperLimit$$

Finally, the value of a_1 is updated using the new restricted value of a_2. This can be done without having to compute the value of the remaining Lagrange multipliers by adding the functions for the old and new multipliers (Platt, 1999).

$$a_1^{old} - y_1 y_2 a_2^{old} = y_1 \xi = a_1^{new} - y_1 y_2 a_2^{new,restricted}$$

$$\Rightarrow$$

$$a_1^{new} = a_1^{old} + y_1 y_2 \left(a_2^{old} - a_2^{new,restricted} \right)$$

The R package e1071 has a support vector machine implementation based on LIBSVM (Chang & Lin, 2011) which was used. LIBSVM implements a further modified version of SMO by Fan, Chen and Lin (2005) which further reduces the number of computations making it more efficient.

3.3.6 Choosing a kernel type

The support vector machines can be trained quickly for a variety of parameter settings. For this reason, the models were trained with various parameters in different settings. The parameters are listed below.

The kernels are the functions $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)\phi(\mathbf{x}_j)$ in the definition of the Support Vector Machine optimisation problem. The linear kernel is the linear product of two feature vectors. This cannot be parameterised.

$$\text{linear kernel} = \phi(\mathbf{x}) = \mathbf{x}$$

The radial basis kernel is based on the Gaussian distribution. It has a single parameter in $\gamma = \frac{1}{2\sigma^2}$. As shown in Figure 3.4 a larger σ value or smaller γ value creates a more rounded boundary (dashed purple line) in the original data space.

$$\text{radial basis kernel} = \phi(\mathbf{x}) = \exp\left(-\frac{||\mathbf{x}||^2}{2\sigma^2} \right) = \exp(-\gamma ||\mathbf{x}||^2)$$

The sigmoid kernel has two parameters of interest. The coefficient a and the intercept γ. A demonstration of the effect they have on the decision boundary is provided in Figure 3.5. Increasing a pushed the boundary to the bottom left and increasing γ made the curve steeper.

$$\text{sigmoid kernel} = \phi(\mathbf{x}) = -\frac{1}{1 + \exp(-(a\mathbf{x} + \gamma))}$$

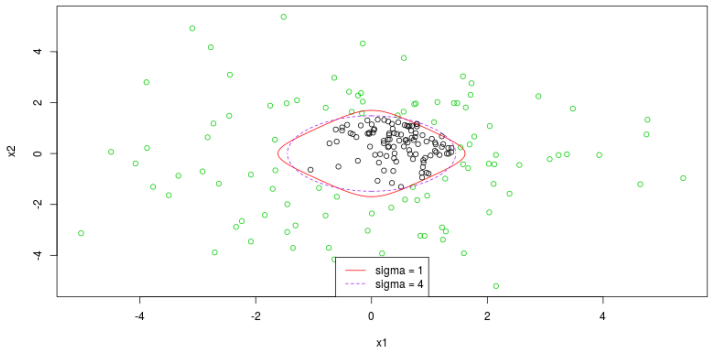

Figure 3.4: Radial basis parameters. The image shows the data from Figure 3.2. Decreasing the γ value creates a more rounded boundary indicated by the purple dotted line.

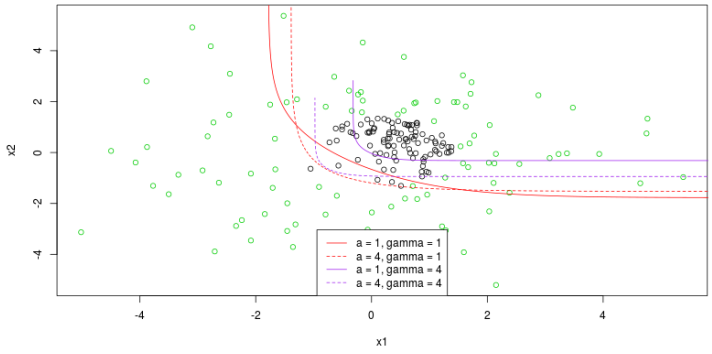

Figure 3.5: Sigmoid parameters. The image shows the data from Figure 3.2. Increasing a pushes the boundary to the bottom left as shown by the dotted lines. Increasing γ increases the steepness of the curve indicated by the purple lines.

The polynomial kernel has three parameters of interest. The coefficient, the intercept and the degree of the polynomial.

$$\text{polynomial kernel} = \phi(\mathbf{x}) = (a\mathbf{x} + \gamma)^d$$

The polynomial kernel is easier understood by looking at the effect its parameters have on the kernel space. For the simulated data first presented in Figure 3.2 a number of parameters were chosen and some plots were created in Figure 3.6. Each heading provides three numbers for γ, a and d respectively. Within the kernel space a clear linear separation is what is needed. What effects the ability to create this separation will vary between

Table 3.7: SVM parameters

Kernels	Linear	Radial	Sigmoid	Polynomial
Gamma		0.01, 0.1, 1	0.01, 0.1, 1	0.01, 0.1, 1
Coefficient			0, 0.1, 1	0, 0.1, 1
Degree				1, 2, 3

datasets. For the simulated data a quadratic kernel was an intuitive choice for data that can be separated by a circle. This was not sufficient, a small value for γ was also important in creating a kernel space with linear separation. The near linear separation of the kernel spaces with small γ and $d = 1$ demonstrate that there are often multiple ways of achieving the same effect with a polynomial kernel. It is not always possible to visualise data and parameters do not always have the intended effects. For this reason it is important to test different parameter settings.

For the data analysis the γ values were varied over the set of values $\{0.01, 0.1, 1\}$. The coefficients were varied over the set of values $\{0, 0.1, 1\}$. Finally, the degrees were varied from 1-3. Table 3.7 below shows each kernel and the parameters it was run over.

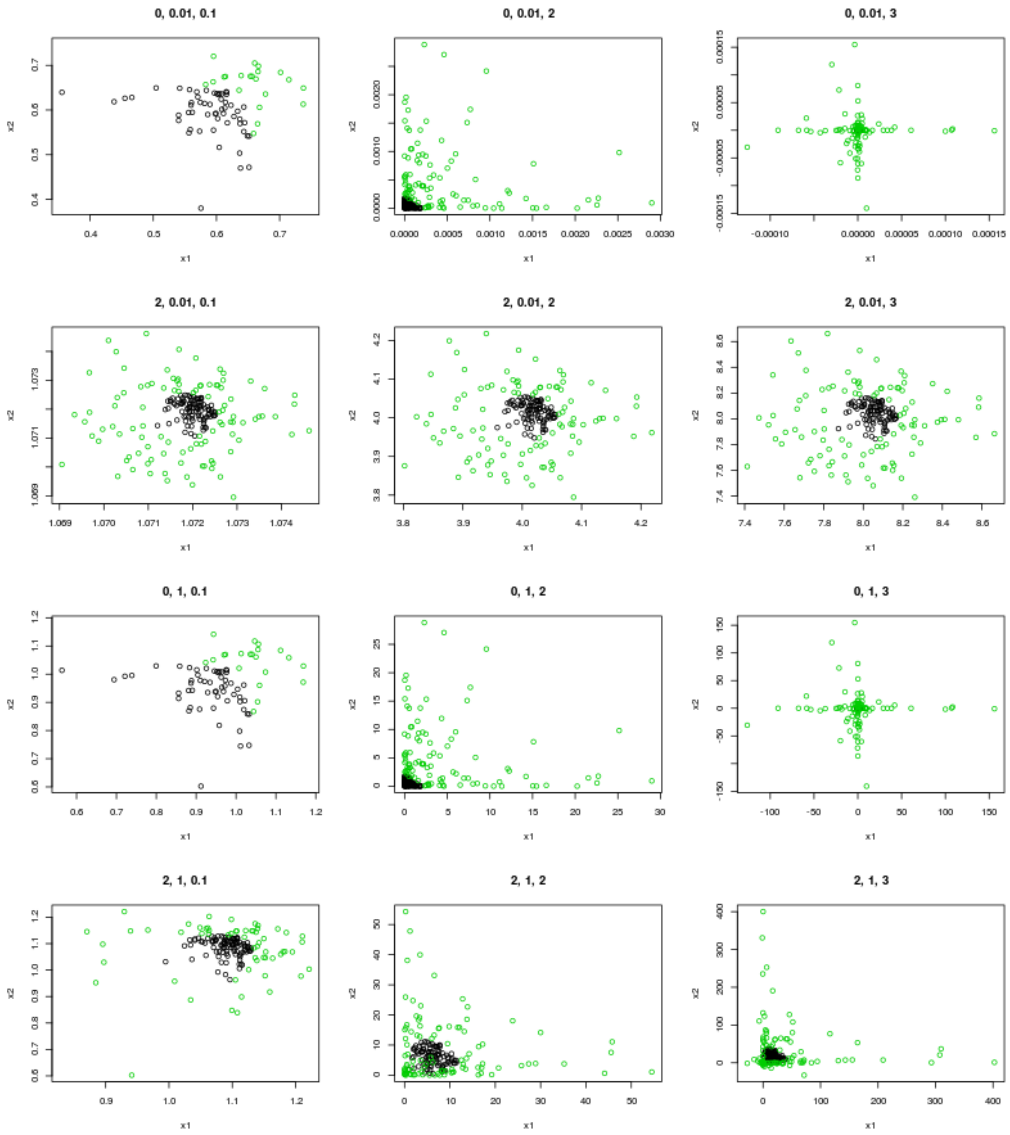

Figure 3.6: Polynomial kernel spaces. Each image shows the data from Figure 3.2 after being augmented by a polynomial kernel. The numbers in the subtitle of each image are γ, a and d. It can be seen from the first two images of the first and third rows that a linear boundary could separate the data.

Chapter 4

Testing document classification models with network features

This chapter describes and provides results for the experiments built to test the efficacy of graph based weighting schemes and features in text classification. These graph based weighting schemes and features are developed in Chapter 2. The methods used to perform the classification of text document are described in Chapter 3.

This chapter starts by describing the datasets. This starts with a description of how the datasets are pre-processed then follows into a more detailed look at each dataset. The first dataset is the National Science Foundation dataset (Section 4.1.2) the second dataset is the Reuters-21578 dataset (Section 4.1.3).

The section continues to develop the experiments, first by describing the metrics used to evaluate the methods (Section 4.2), then some preliminary analysis of the term features of both datasets. A description of the experiments follows (Section 4.4). Section 4.4.3 describes the hyper-parameter selection process for the neural network. The results are presented in two parts, first the main effects (Section 4.5) and then the interaction effects (Section 4.6). The section ends with conclusions (Section 7) from the results and some suggestions on future study (Section 4.7.2).

4.1 Description of dataset

4.1.1 Pre-processing the data

The data was pre-processed before exploratory analysis began. Pre-processing prepares a document for experiments. The methods illustrated below are filtering, pruning and stopword removal. The example documents from Section 2.1.2 and Table 2.2 are used. A few words are added to demonstrate the removal of numbers. The tm package (Feinerer, 2011) in R was used to perform the operations below. In order to illustrate the preprocessing operations, we made use of the following sentence:

> That book's really not good just in case, you know, browsing turned to buying. You'd be wasting your $80 of your money.

The punctuation, numbers and stopwords were removed. The tm package has a set of stopwords that it provides.

> That books really good just case know browsing turned buying Youd wasting money

The document was stemmed. This reduces words to their lexical forms that allows words that are largely similar to be merged as features (e.g. books/book or browsing/brows).

> That book realli good just case know brows turn buy Youd wast money

Spaces were removed and all the words were converted to lower case. This further combines words that are the same but might have different cases. It also facilitates the separation of documents into individual words as the words can be separated by spaces.

> that book realli good just case know brows turn buy youd wast money

This final version of the document was converted into a term document matrix. Words with fewer than two letters were excluded as features.

Table 4.1: Class distribution of NSF sample

ANI	AST	BES	EEC	EIA	HRD	IIS	Total
684	859	789	466	549	511	869	4727

4.1.2 National Science Foundation dataset

The National Science Foundation (NSF) was founded by the US congress in 1950. It provides funding to research projects. Researchers submit proposals to the NSF which determines whether or not to provide the researchers with a grant. An extract of the grant data was captured between 1990 and 1994. The research field and abstract were kept to form the dataset.

The research fields were distributed such that there were some research fields with very few abstracts and others with very many. It became clear during the preliminary experiments that this degree of skewness would make it extremely difficult for any of the proposed models to predict a large number of the (infrequently observed) classes. We therefore selected a subset of 4727 documents from the full dataset, consisting of seven classes, with the number of documents in each class of the same order of magnitude (smallest class 466; largest class 869; mean class size 675; see Table 4.1. This subsample was randomly split into four sets with three containing 1181 samples and the fourth 1182. Accuracy results are based on 4-fold cross-validation.

The class distribution of the sample is provided Table 4.1

The full names of each class are provided below.

- Advanced Networking Infrastructure (ANI)
- Astronomical Sciences (AST)
- Bioengineering and Environmental Systems (BES)
- Engineering Education and Centers (EEC)
- Experimental and Intergrative Activities (EIA)
- Human Resource Development (HRD)
- Division of Information & Intelligent Systems (IIS)

4.1.3 Reuters-21578 dataset

The Reuters-21578 data set was originally complied and labelled by staff from Carnegie Group and Reuters Ltd. The data set consists of 21578 Reuters Newswire articles

Table 4.2: Class distribution of Reuters sample

Earn	Acq	UK	Canada	Japan	Crude	Trade	Total
444	350	607	554	427	425	333	2364

that appeared in 1987. They were categorised into 256 labels. An important feature differentiating this dataset from the NSF dataset is that a single document can be assigned multiple labels. Much of the data relates to financial news and has three types of labels, topics, places and people. The data is available from David Lewis at daviddlewis.com.

The class distribution in the original dataset is highly skewed. We therefore extracted from the original data a relatively better balanced subset, with the relative frequency of each class show in Table 4.2. As multiple labels can be assigned to each document, the total sum of labels in the sample is 3140 while the total number of documents is 2364.

The full name of the Earn class is Earnings and Earnings Forecasts. The full name of the Acq class is Mergers and Acquisitions. Earnings refers to the quarterly or semi-annual reporting firms provide to their share and stockholder. Acquisitions refers to the purchase and sale of companies by other companies. The location based classes refer to activities taking place within those locations. Crude refers to crude oil. Trade refers to international trade by private companies and international trade dynamics between sovereign nations.

4.2 Accuracy metrics and statistical analysis

The main effects are presented with the maximum and average accuracy. For main effects both the testing and training accuracy are provided. For the interaction effects only the test accuracy is provided as the maximum accuracy is seen as supplementary and its inclusion clutters the presentation of results. The maximum accuracy is the highest accuracy achieved by a single model given the specifications being presented. The average accuracy is averaged over all the models within a specification.

The accuracy scores can provide insight into which specifications had a positive impact on test and training accuracy.

4.3 Preliminary analysis of term features

4.3.1 Quantitative summary of term features

The distribution of word counts per document can provide a more in-depth perspective on the datasets. The distributions for both datasets are skewed to the left (see Figure 4.1).

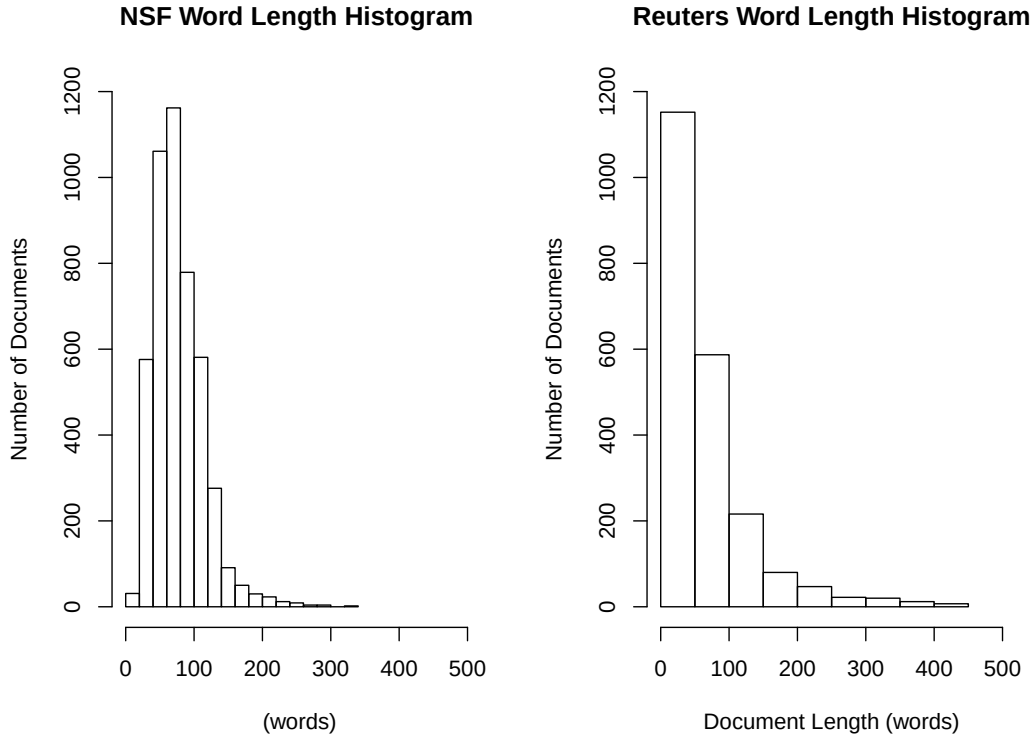

Figure 4.1: Histograms of document lengths for each dataset. The Reuters dataset has a shorter but wider histogram with a lower mode.

Boxplots of document length were plotted for each class in each dataset. This allows for a comparison within the datasets and across the classes. The NSF dataset has similarly sized classes mostly with median word counts between 50 and 100. The Reuters dataset has classes that are more varied in the document lengths. Earn and Canada are below 50 words on average. Japan, Crude and Trade have medians below 100 but the distributions are wider and all go over 100 words. There are also more outliers for the Reuters dataset than NSF dataset.

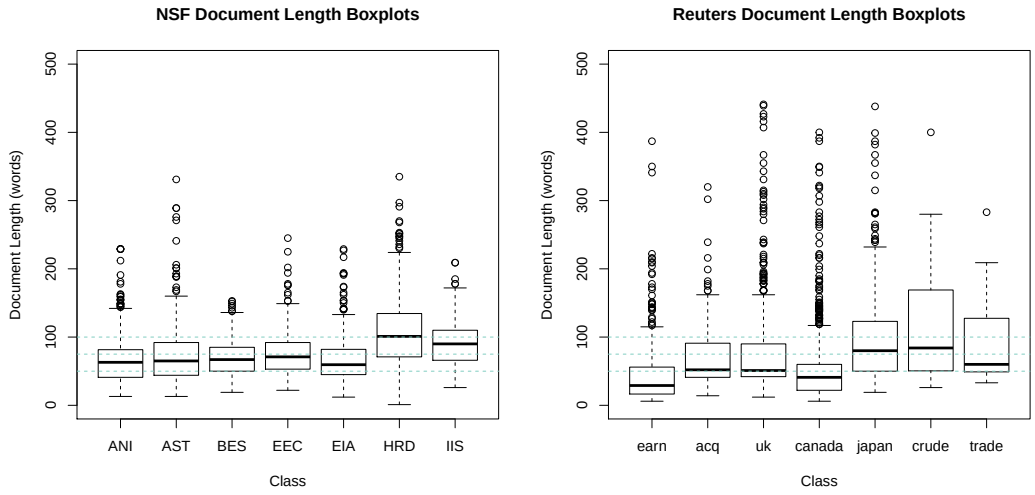

Figure 4.2: Boxplots of document lengths for each dataset. NSF on the left and Reuters on the right. The dotted lines are plotted at 50, 75 and 100 words. The NSF dataset has median between 50 and 100 words. The Reuters dataset mostly has class medians between 50 and 100 words. The box plots are more varied in size with the last three boxplots having an interquartile range larger than 50 words.

4.3.2 Shared Words

Document classes can be easily separated when they do not share words. By implication the fewer words are shared by document classes the easier it should be to separate them. The tables (Table 4.3, Table 4.4) show the number of words shared from a class in the row to the classes in the columns. ANI shares 45% of its words with AST.

NSF classes share between 30% and 50% of their words in general. EIA shares 60% of its words with HRD and IIS. EEC shares 62% of its words with HRD. These are the largest proportions of words being shared between classes.

The Reuters dataset has multiple classes per document. The percentage of shared words are higher for Reuters than NSF in general. There are 2 sources of shared words. The first is when classes have similar themes and therefore use similar words. This is similar to the NSF case. The second source of shared words is when classes are regularly assigned to the same document. In this case the words in the class are the same because some of the documents are the same.

UK and Canada have the most words shared with them from other categories. They have between 46% and 87% and 42% and 84% respectively with averages of 75% and 72%. Japan shares 91% of the words from Trade which is the single largest sharing of words.

Table 4.3: Shared word for NSF dataset. Percentages represent the amount of words shared between classes in rows and columns

	ANI	AST	BES	EEC	EIA	HRD	IIS
ANI	100%	45%	44%	43%	45%	52%	50%
AST	35%	100%	41%	38%	36%	47%	44%
BES	32%	38%	100%	37%	33%	47%	41%
EEC	45%	52%	54%	100%	49%	62%	54%
EIA	50%	51%	51%	51%	100%	60%	60%
HRD	33%	39%	42%	38%	35%	100%	41%
IIS	36%	41%	41%	37%	40%	46%	100%

Table 4.4: Shared words for Reuters dataset. Percentages represent the amount of words shared between classes in rows and columns

	earn	acq	uk	canada	japan	crude	trade
earn	100%	48%	76%	79%	55%	27%	21%
acq	40%	100%	73%	75%	50%	22%	19%
uk	34%	39%	100%	42%	44%	17%	14%
canada	39%	44%	46%	100%	39%	19%	14%
japan	34%	37%	60%	49%	100%	19%	20%
crude	60%	58%	85%	84%	67%	100%	31%
trade	57%	61%	87%	77%	91%	39%	100%

4.3.3 Most associated words

Lets say we have a word, *timber*, that appears twice in five classes and thirty times in the last class. If we have to decide where to put the word we might say there is a 30/40 chance it will appear in the last class and 2/40 chance to appear in the remaining classes. A prediction based on these probabilities would put *timber* it in the last class. Alternatively we might have a word, *business*, that appears once in the first five classes and twice in the last class. As such the chances, *business*, should be in the last class are 2/7 which is higher than 1/7 but absolutely low. If *timber* and *business* are ranked in the last class we would rank *timber* above *business*. This ranking indicates which words are more associated with the last class, which is to say *timber* is more associated with the last class than *business*.

The tables below (Table 4.5, Table 4.6) provided the 10 words most likely to indicate a particular class. This was used as an exploratory tool. It can summarise the classes to just 10 words that are either common or particular to a specific class. Ideally words should clearly indicate the content or theme of a class and not be generic. In order for a word to be in the top 10 it should ideally have a large probability associated with the corresponding class. The tables can also provide some insight into the style and content of the documents. It is not feasible to reproduce all the words in all the documents. This is a way to investigating some of the underlying data for insights in a manageable manner. In a sense, this speaks to the importance of text visualisation as described in Chapter 5. The words provided below have been stemmed as described at the beginning of Section 4.1.

For the NSF dataset the top words have thematic consistency and they differ between classes. ANI contains *NSFnet*, *SURAnet*, *PREPnet* and *internet* which are all related to telecommunications technologies. AST contains *star*, *interstellar*, *supernova* and *galaxy* these are all terms known to be associated with astronomy. EEC, EIA and HRD generally have three or four letter words associated with them. Some of these are consonant sequences, suggesting a greater use of acronyms.

For the Reuters dataset there is no word overlap between classes. The individual classes seemed well separated thematically. The Earn class contained 8 out of 10 words with less than 4 letters which are most likely acronyms. The Acq class mainly contained the names of companies like Royex Harcourt and Breakwater. These terms are specific to the time and place when these acquisitions took place. It is likely that a model built on this data would struggle to accurately predict news articles regarding acquisitions in a different time and place. Other classes have clear connections between the words and class labels, Crude and Trade do not. For instance, gilt and labour are banking and political references in the UK. Canada has cities named Alberta and Ontario. For Trade however the words seem

Table 4.5: Most associated words with NSF dataset per class. Each word has a distribution across all the classes. If a word appears in a class then it is in the top 10 words as sorted in descending order for probability within that class.

ANI	AST	BES	EEC	EIA	HRD	IIS
nsfnet	star	wastewat	industryunivers	cise	sem	queri
midlevel	interstellar	groundwat	iucrc	behalf	girl	discours
suranet	galact	aquif	nnf	elig	camp	deduct
bit	supernova	implant	steel	processor	math	transact
backbon	galaxi	sediment	fire	acm	rcms	semant
prepnet	dwarf	aqueous	erc	mention	grade	nonmonoton
internet	telescop	ferment	alfr	honor	ssc	tempor
oarnet	cloud	bioreactor	reu	cra	counsel	schema
connect	astronom	tissu	tribolog	gte	teacher	reason
mail	quasar	fate	tie	hisher	middl	knowledgebas

generic or unrelated specifically to the class. There are words like refocus, abrupt and chronic which could be used in any arbitrary context. Senior Japanese officials like Haruo Maekawa and Hiroshi Hirabayashi rank third and forth on the list of most influential words. These words and names are specific to Japanese trade and not trade in general. 91% of words in Trade can be found in Japan. Trade shares 69% of its words with other classes overall. It is likely that many of its most associated words are derived from other classes and therefore do not seem thematically unified.

4.4 Experiments

In Chapter 2 a number of modifications to the document model have been described. These include feature normalisation (inverse-document-frequency), graph based weighting schemes (total degree, in degree, out degree and clustering coefficient); other characteristics used in the construction of the network representing the document e.g. edge weighting, window size; and use of graph features of the entire document as features.

In this section the effects of these modifications on the accuracy of a number of document classification models are investigated. The results were analysed in general to understand the effects of the various model specifications on test and training accuracy. In particular, there are two questions that the results seek to answer.

- Do graph based weighting schemes offer performance improvements over traditional weighting schemes?

Table 4.6: Most associated words with Reuters dataset per class. Each word has a distribution across all the classes. If a word appears in a class then it is in the top 10 words as sorted in descending order for probability within that class.

earn	acq	uk	canada	japan	crude	trade
shr	norcro	stg	dome	ldp	prt	conabl
rev	avana	gilt	ontario	nakason	hydrocrack	hata
bodyshr	royex	lawson	rev	ministri	refineri	hirabayashi
debit	gelco	frn	noranda	yasuhiro	terra	maekawa
avg	harcourt	guin	wilson	nakasoneaposs	bpd	cif
mths	watanab	ordinari	writedown	yenaposs	grangemouth	refocus
bodyop	wick	guaranti	cts	miyazawa	graven	abrupt
div	blair	labour	domeaposs	prefectur	relief	chronic
shrs	breakwat	penc	avg	japanaposs	explos	customsclear
qtr	huski	lloydaposs	alberta	bodyjapanes	fieldaposs	donegan

- Does the inclusion of graph features to the term document matrix offer an increase in predictive accuracy?

The experiments are conducted over all the model specifications listed below. This produces 1440 results over both of the datasets. Each result has a test and training accuracy figure associated with it. The accuracy figures are produced with 4-fold cross validation. The test and training accuracy figures are the average of the four training and test figures that result from the cross validation process. Each of the datasets had a different kind of response, as such the modelling and accuracy was determined differently. This is explained for each dataset in turn, Section 4.4.1 for the NSF dataset and Section 4.4.2 for the Reuters dataset. The model specifications are provided here.

- 5 weighting schemes
- 6 window sizes
- 2 edge weightings
- 2 document models
- 2 feature normalisations
- 2 datasets
- 3 machine learning methods

4.4.1 NSF experiment

The sample of the NSF dataset used for experiments has 4727 documents each associated with one of 7 classes. Models are constructed using all 3 machine learning methods. Accuracy is determined by the proportion of correct predictions when comparing model predictions to the actual document labels.

4.4.2 Reuters experiment

The sample of the Reuters dataset used for experiments has 2364 documents. Each document is associated with at least 1 of 7 classes. This raises 2 questions. The first question is how predictions will be constructed i.e. how labels will be assigned to documents. The second is how accuracy will be measured.

There are several ways of assigning (possibly multiple) labels to a document. For a model predicting a single class for each document, typically the class with the highest probability is chosen. For the case of multiple classes some number of classes with the highest probability might be chosen instead. This creates the problem of deciding how many classes should be chosen. An alternative is to use a threshold probability and assign all labels with probability above the threshold. This would be an additional model parameter which would have to be tuned after training. It is possible that a document might have labels assigned to it with very low probabilities, which seems undesirable.

In our approach, each class was compared to the remaining classes using a binary response, 1 for this class and 0 for the remaining classes. This reduces the problem to multiple binomial experiments as opposed to a single multinomial experiment.

There were two approaches considered to computing accuracy when each document can belong to multiple classes. The first was to divide the total number of correct predictions made (i.e. over all classes) by the number of predictions made (i.e. seven). Thus if binary yes/no predictions for a particular document were correct for four classes and incorrect for three classes, then that document would have accuracy of $\frac{4}{7}$. Accuracies can then be averaged over documents. A problem with this approach is for all classes, there are substantially more documents that do not belong to a class (i.e. do not have that label) than do, therefore accuracy values might appear inflated. For example, if τ_j documents belong to the majority ('no') outcome for class j in $(1, \ldots, J)$, and there are $\mathcal{T}$ documents in total, then a model which simply predicts the majority outcome ('no')

$$\frac{\mathcal{T} - \sum_{j=1}^{J} \tau_j}{\mathcal{T}}$$

in each class will have an accuracy of , which for the Reuters dataset is 81%. While adjustments could be made to the interpretation of the test accuracy score, the superior approach seemed to be to impose a stricter criteria for a 'correctly classified' document. If a document is assigned all its observed labels, and only those i.e. if all seven binary classifications are correct, then the document is correctly classified. Thus, all the predicted labels of a document were compared to the true classes, and if a single label was incorrect then the document was said to be incorrectly labelled. The proportion of documents correctly labelled then become the measure of accuracy.

4.4.3 Choosing a neural network architecture

When selecting an appropriate neural network architecture, various features (or 'hyper-parameters') need to be determined: the number of hidden layers used, the numbers of nodes in each layer, functional forms for the activation function, data standardization, learning rates and associated learning features such as dropout and mini-batch size. These features can greatly influence both the accuracy and speed of the neural network (LeCun (1998); Bengio et. al. (2003); Mikolov et. al. (2010); and Huang et. al (2012)) but, as the number of possible parameter combinations is extremely large, it is not possible to search exhaustively for the best combination.

We selected hyper-parameter settings by conducting a preliminary investigation in which each parameter was varied independently of the others, in order to choose a suitable combination of parameters. All models use a neural network with two hidden layers, on the basis of initial tests finding these more effective than a single layer network. We investigated the following hyper-parameter values

- number of hidden layer nodes: 64 - 2048
- learning rates: 0.1, 1
- dropout rate: 0 - 0.75
- mini-batch size: 0.1 - 0.9 (as proportion of sample size)
- activation functions: sigmoid and hyperbolic tangent
- data normalisation: column-wise, none and row-wise

Our investigations identified the best architecture, relative to the number of parameters, as consisting of 256 and 512 in the first and second hidden layers respectively, a learning rate of 1, no dropout, and a mini-batch size of 0.75. We use this architecture to generate

the results reported in the following sections. Full details of these preliminary experiments are provided in Appendix A.

4.5 Results I: main effects

In this section the effects of the various model specifications are considered with respect to both training and test accuracy. The interaction effects are considered in the next section (4.6). The various specifications include weighting schemes, machine learning methods, edge weighting schemes, window sizes, feature normalisation and document models. The exact number of each specification is provided in Section 4.4.

The results are presented with the maximum and average accuracy. The main effects show both test and training accuracy while the interaction effects show test accuracy only. The maximum accuracy is the highest accuracy achieved by a single model given the specifications being presented. The average accuracy is taken over all the models within a specification.

The results begin with the weighting schemes which are compared individually and grouped into term frequency and graph based weighting schemes. This serves to address the first question the experiments aim to answer. This question is whether or not graph based weighting schemes offer increased accuracy over the benchmark methods. The benchmark methods are the term frequency methods. The section then continues to review the performance of the machine learning methods (naïve Bayes, neural networks and support vector machines). This is followed by a brief section on the edge weighting methods (co-occurrence and distance). The edge weights take into account the different ways in which edges can be weighted between a pair of words. Edge weights can either be assigned based on words appearing within the window ignoring how many words appear between them (co-occurrence). Alternatively edge weights can be assigned to words appearing within a given window and taking the distance, measured by intervening words, into account. The window sizes are next, these were 1, 2, 5, 10, 20, 35. Words that appear within a window have edges assigned to them. Feature normalisation which is the use of inverse-document-frequency not just for term frequency but also graph based weighting schemes is also reported on. Finally the document models themselves, the document model with and without graph features are discussed. While informative this section does not describe how the model specifications interact to determine overall accuracy. This is left for Section 4.6.

This section shows that when the graph based weighting schemes are averaged and

compared to term frequency the graph based weighting schemes have higher performance. When the graph based weighting schemes are compared individually total degree has higher accuracy than term frequency. Window sizes have a non-linear relationship with accuracy with intermediate window sizes performing best, while edge weighting seems to have little impact on accuracy. Finally the introduction of graph features seems to reduce accuracy. A closer look in Section 4.6 reveals that the introduction of graph based features has a more varied impact when reviewed by machine learning methods and weighting schemes.

4.5.1 Term frequency and graph weights

In the basic prediction problem, the input features are frequency counts of words (i.e. the term-document matrix). Weighting schemes refer to how the term-document matrix can be populated with weighted counts using either graph-based or other non-graph-based approaches or non-count entries. In this section we assess the effects of these approaches on model accuracy. The graph weighting schemes provide weights that are either degree, in degree, out degree and clustering coefficient. Other weighting schemes include tf-idf, which weighs counts by the relative frequency with which they occur in the corpus. Adjusting weights by relative frequency within the corpus can also be applied to graph-based weighting schemes. The graph-based term frequency weighting schemes are grouped and their average and maximum prediction accuracies presented, as described in Section 4.2. This provides an overall comparison between graph-based and non-graph-based weighting schemes.

Graph weighting schemes were slightly superior to Term Frequency on average with test accuracy of 67.58% compared to 66.18%. The maximum test scores however were the same. Graph weighting scheme training scores were higher which suggests that graph features were able to capture some information specific to the documents that Term Frequency could not. Higher test accuracy suggests that this information could be generalised to unseen cases.

		Mean	Maximum
Term Frequency	Test	66.18%	93.92 %
	Train	76.78%	100.00%
Graph-Based Weighting	Test	67.58%	93.92 %
	Train	79.44%	100.00%

To provide more clarity on the performance of individual weighting schemes the graph based weighting schemes are separated and presented.

Total Degree was the only graph feature to have superior test accuracy to Term Frequency on average with 69.51% to 66.18% (see Figure 4.3). In Degree had higher training accuracy than Term Frequency with 77.51% versus 76.78%. This suggests the In Degree has captured specific characteristics of the documents in the training set. As the test accuracy was lower than Term Frequency these characteristics could not be generalised by the models to unseen cases.

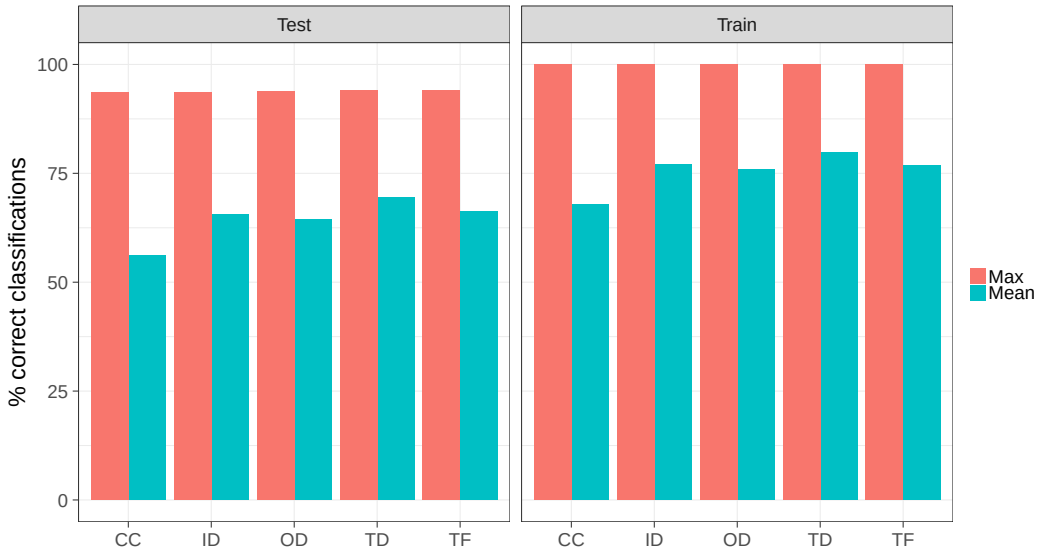

Figure 4.3: Accuracy graph for individual weighting schemes. Test accuracy scores to the left and training accuracy scores to the right. Each panel shows maximum and mean accuracy scores. The x-axis shows different weighting schemes indicated by CC (clustering coefficient), ID (in-degree), OD (out-degree), TD (total degree) and TF (term frequency).

4.5.2 Machine learning methods

From a machine learning perspective the term-document matrix is the data and the classes are the response. This lends itself to various machine learning methods. There are three machine learning methods described in Chapter 3, which include naïve Bayes (Section 3.1), neural networks (Section 3.2) and support vector machines (Section 3.3). For each method the average and maximum prediction accuracies are used to assess the effect machine learning methods have on predictive accuracy. Further analysis has shown that the predictive accuracy of machine learning methods is affected by feature normalisation and weighting schemes. Each effect is discussed in Sections 4.6.1 and 4.6.3 respectively.

Support Vector Machines had the highest mean test accuracy of 77.2% followed by Neural

Networks at 65.04% and finally Naïve Bayes at 50.84% (see Figure 4.4). The maximum test accuracy scores were the same for neural networks and support vector machines at 93.92%.

Support Vector Machines had large average training accuracy scores relative to other methods. The training accuracy was 99.93%. This training accuracy did not translate to as large an increase in test accuracy. This suggests some overfitting relative to the other machine learning methods.

Naïve Bayes combines probability values for a word within a document in a linear fashion. Words can be added to a test document in any combination to achieve the same prediction. This is the unigram assumption which is made explicitly in the computation of the posterior probabilities for the document. Support vector machines and neural networks can arrive at predictions through non-linear combinations of words. This is explicit in the support vector machine through the use of non-linear kernels like the radial basis, polynomial and sigmoid kernels. The neural network achieves non-linearity through the layers and activation functions. Since the words in a document are dependent on each other within phrases and paragraphs it speaks to reason that non-linear methods are more likely to accurately model their behaviour. This is shown in the relative superiority of support vector machines and neural networks over naïve Bayes.

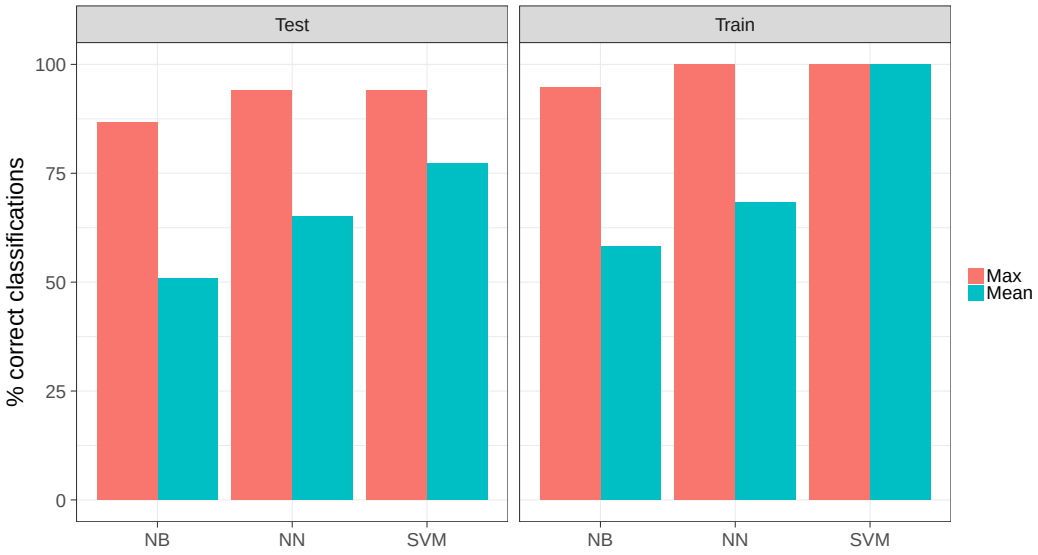

Figure 4.4: Accuracy graph for machine learning methods. Test accuracy scores to the left and training accuracy scores to the right. Each panel shows maximum and mean accuracy scores. The x-axis shows the machine learning methods NB (naive Bayes), NN (neural networks), SVM (support vector machines).

4.5.3 Edge weighting

Weighting schemes refer to how the term-document matrix can be populated with weighted counts using either graph-based or other non-graph-based approaches or non-count entries. When graph-based methods are used to populate the term-document matrix, a text graph is constructed and used for this purpose. The text-graph has edges that connect words based on appearing within a specified window of other words between them. The edges can be weighted to represent this relationship. There are two edge weighting methods considered, co-occurrence and distance-weighted. We define co-occurrence to mean only taking into account that words appear in the same window, by providing an edge weight of one i.e. a binary indicator that does not measure the precise distance, in terms of the number of words, separating a word pair. A different approach is to use weights to represent how far away connected words are from each other. Further details can be found in Section 2.2.4. The prediction accuracies for both types of edge weighting are presented below.

It is not surprising therefore that co-occurrence and distance edge weighting produced differences in test accuracy of 0.01% (see Figure 4.5). The maximum test accuracy scores did not differ. The mean training accuracy scores differed by 0.35%. The maximum training accuracy scores did not differ.

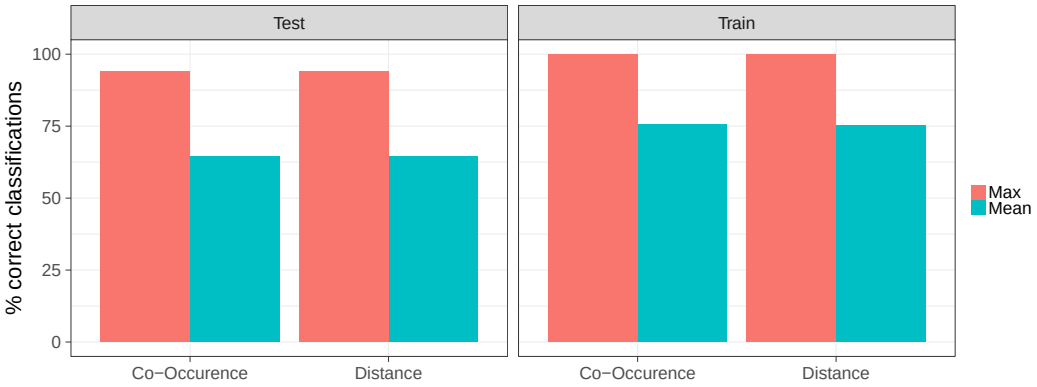

Figure 4.5: Accuracy graph for edge weighting methods. Test accuracy scores to the left and training accuracy scores to the right. Each panel shows maximum and mean accuracy scores. The x-axis shows co-occurrence and distance weighted edges.

4.5.4 Window sizes

When graph-based weighting schemes are used a text-graph is created (see Section 2.2.2) which creates an edge between words that appear within a specified window of words from each other. The windows can have different sizes. For each window size including 1, 2 5, 10, 20 and 35 the average and maximum prediction accuracy values are presented. Window sizes have an effect on prediction accuracy when applied over different weighting schemes, this effect is discussed in Section 4.6.2.

Maximum accuracy did not change for training or test sets at any window size (see Figure 4.6). The highest accuracy scores were recorded for window size 5 at 65.31% and window size 10 at 76.12% for testing and training sets respectively. Window sizes seemed to have an effect on test accuracy but the relationship was non-linear. Test accuracy rose up to window size 5 and dropped thereafter. It seemed to rise quickly to from 1 to 10 and drop slowly from 10 to 35. The last window size 35 had higher test accuracy than the first.

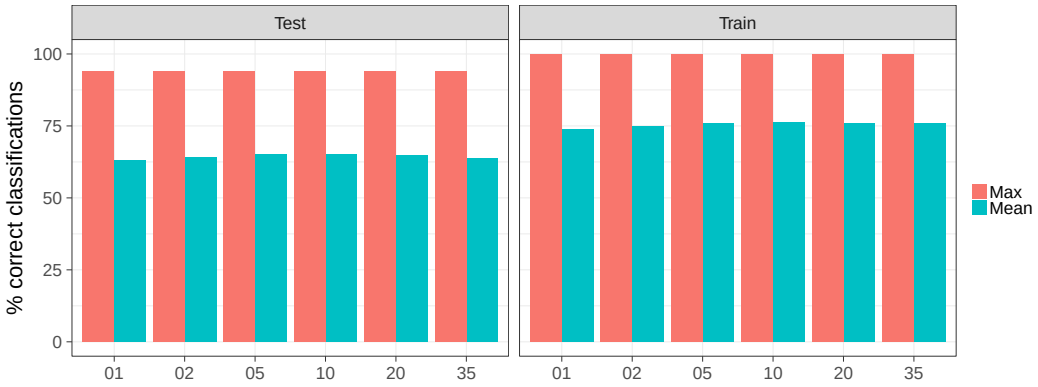

Figure 4.6: Accuracy graph of window sizes. Test accuracy scores to the left and training accuracy scores to the right. Each panel shows maximum and mean accuracy scores. The x-axis shows graph window sizes from 1,2,5, 10, 20 and 35.

4.5.5 Feature normalisation

Both graph-based and non-graph based weighting schemes can be adjusted by inverse document frequency. In this section the prediction accuracies of both approaches, with inverse document frequency and without are presented. There is an interaction effect between feature normalisation and machine learning methods which is discussed in Section 4.6.1.

Inverse document frequency had test accuracy of 68.15% relative to no feature normalisation

at 60.57% (see Figure 4.7). Inverse Document Frequency had higher maximum test accuracy at 93.92% relative to 93.65% without Inverse Document Frequency. The mean training accuracy scores were nearly 10% apart with Inverse Document Frequency at 80.04% and no feature normalisation at 70.93%. The maximum training scores were the same at 100%. Much of the increase in training accuracy was directly translated to test accuracy. This suggests that the use of Inverse Document Frequency highlights features that can improve accuracy in unseen cases.

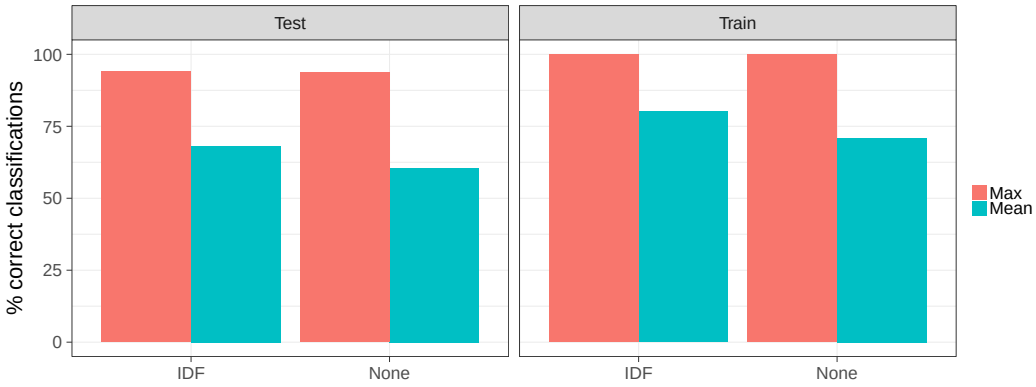

Figure 4.7: Accuracy graph of feature normalisation. Test accuracy scores to the left and training accuracy scores to the right. Each panel shows maximum and mean accuracy scores. The x-axis shows IDF (inverse document frequency) and None. Under IDF weighting schemes include inverse document frequency multipliers; under None they do not.

4.5.6 Document models

An augmentation to the term-document matrix would be to add graph-based features. The graph based features are extracted from the text graph for each document. The features are added to the term-document matrix by attaching additional feature rows to the term document matrix. The prediction accuracies in this section represent the term-document matrix and the term-document matrix with additional graph feature columns. There are interaction effects between document models and weighting schemes (Section 4.6.4) along with document models and machine learning methods (Section 4.6.5).

Adding graph features to the document model decreased average test accuracy from 67.15% to 61.57% (see Figure 4.8). The maximum test accuracy decreased from 93.92% to 93.78% in comparison. Training accuracy decreased from 79.96% to 71.01%. Maximum training accuracy showed no difference at 100% for both document models. In general graph features decreased test accuracy however Support Vector Machines had an increase in

test accuracy after graph features were added. This is discussed below in the interaction effects between the document models and machine learning methods.

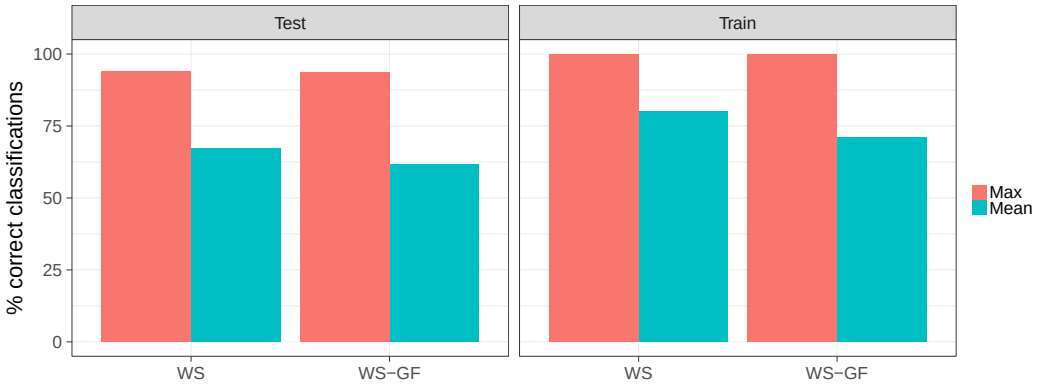

Figure 4.8: Accuracy graph for document models. Test accuracy scores to the left and training accuracy scores to the right. Each panel shows maximum and mean accuracy scores. The x-axis shows WS (weighting schemes) and WS-GF (weighting schemes with graph features).

4.6 Results II: interaction effects

The main effects could inform which specifications were important and how they affected accuracy. The main effects could not inform which combination of specifications were important and how they affected test accuracy together. The purpose of this section is to enrich the results from the main effects by understanding the interactions.

Many interaction effects were investigated however only five seems to be of note. These noteworthy interaction effects are between feature normalisation and machine learning (Section 4.6.1), window size and weighting schemes (Section 4.6.2), weighting schemes and machine learning methods (Section 4.6.3), weighting schemes and document models (Section 4.6.4) and finally machine learning methods and document models (Section 4.6.5).

In this section some context is added to feature normalisation, showing that while it generally increased average test accuracy naíve Bayes gained the more relative to other methods. Window sizes were non-linear in the effect they had on accuracy but these effects differed in shape (concave, convex or some mixture) and direction (increasing or decreases) for different weighting schemes. The graph based weighting schemes had higher average test accuracy for naíve Bayes than term frequency. Each of the weighting schemes were affected differently by the inclusion of graph features.

4.6.1 Feature normalisation and machine leaning methods

In this section inverse document frequency weighting is seen as a form of feature normalisation. Machine learning methods are shown to interact with feature normalisation to affect the predictive accuracy of the underlying task. There are three machine learning methods naïve Bayes (Section 3.1), neural networks (Section 3.2) and support vector machines (Section 3.3). The average prediction accuracies of machine learning methods are presented for the term-document matrix with and without inverse document frequency.

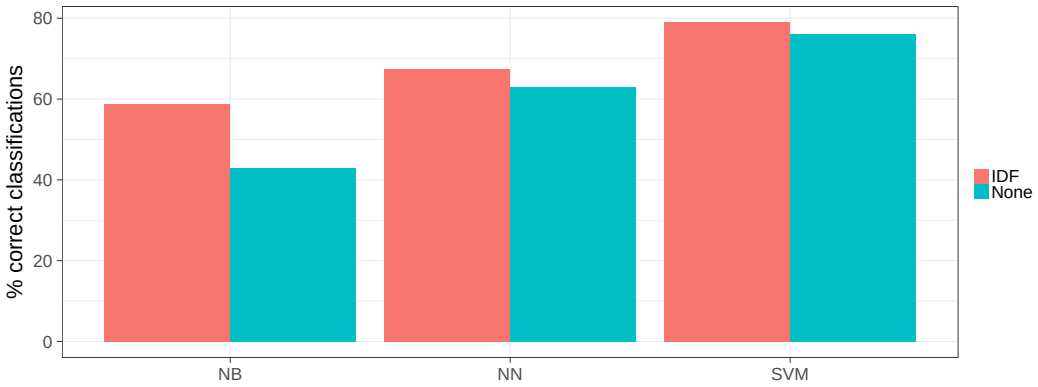

Figure 4.9: Accuracy graph for feature normalisation and machine leaning methods. This graph only shows test accuracy scores. IDF (inverse document frequency) in red and None in blue. The difference is whether or not the weighting schemes was adjusted with inverse document frequency. The x-axis shows NB (naive Bayes), NN (neural networks) and SVM (support vector machines).

All the machine learning methods benefited from feature normalisation (see Figure 4.9). Naïve Bayes benefited the most from feature normalisation with a 15.79% increase in test accuracy. This was less important for Neural Networks and Support Vector Machines with 4.3% and 2.95% increases in test accuracy respectively.

4.6.2 Window size and weighting schemes

Without an optimal process for selecting the window size of a word-word graph it was necessary to try different windows sizes (1, 2, 5, 10, 20 and 35). The term frequency weighting scheme would be indirectly affected by this through the graph-based features in the document model. The average prediction accuracies of the window sizes are presented for each weighting scheme in Figure 4.10.

Window sizes had varied effects on test accuracy depending on the weighting scheme (see Figure 4.10). Clustering coefficient and term frequency generally decreased as the window

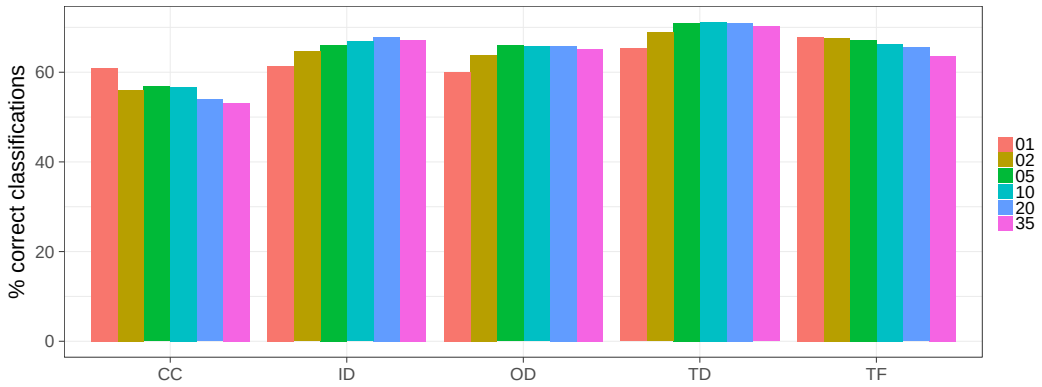

Figure 4.10: Accuracy graph of window size and weighting schemes. This graph only shows test accuracy scores. The colours indicate the different graph window sizes. The x-axis shows the weighting schemes CC (clustering coefficient), ID (in-degree), OD (out-degree), TD (total degree) and TF (term frequency).

size increased. Clustering coefficient decrease in a staggered way whereas term frequency decreased smoothly. Total degree, in degree and out degree all increased. Out degree seemed to plateau after window size 10 whereas total degree started to decrease after window size 10. In degree seemed to increased until window size 20.

From window size 2 total degree had higher accuracy than term frequency for each respective window size. At its highest at window size 20 in degree had higher test accuracy than term frequency at window size 1 which was its highest.

4.6.3 Weighting schemes and machine learning methods

There are five weighting schemes: term frequency, total degree, in degree, out degree and clustering coefficient. There are three machine learning methods naïve Bayes (Section 3.1), neural networks (Section 3.2) and support vector machines (Section 3.3). The average prediction accuracies of machine learning methods are presented for each weighting scheme in Figure 4.11.

Clustering coefficient and term frequency underperformed relative to other methods with 37.41% and 49.07% test accuracy respectively (see Figure 4.11). They were both the largest gainers in test accuracy from Naïve Bayes to Support Vector Machines with 36.7% and 29.11% increases respectively. The graph based methods gained similar amounts between Naïve Bayes and Neural Networks as they did from Neural Networks to Support Vector Machines. Term Frequency gained the most between Naïve Bayes and Neural

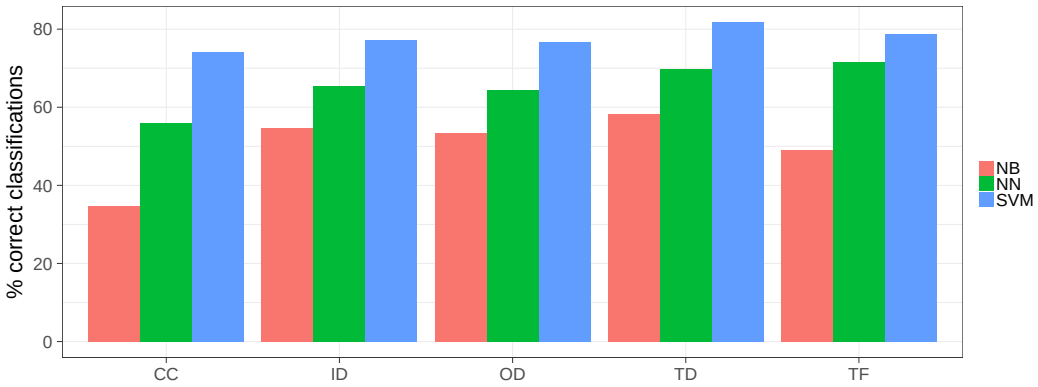

Figure 4.11: Accuracy Graphs of Weighting Scheme and machine learning methods. This graph only shows test accuracy scores. The colours indicate the machine learning methods: NB (naive Bayes) in red, NN (Neural Networks) in green and SVM (support vector machines) in blue. The x-axis shows the weighting schemes: CC (clustering coefficient), ID (in-degree), OD (out-degree), TD (total degree) and TF (term frequency).

Networks but it gained relatively little between Neural Networks and Support Vector Machines.

Naïve Bayes is where Total Degree made its greatest improvement over Term Frequency with over 9% increase in accuracy. For neural networks Term frequency had higher test accuracy by 0.8%. For SVM Total Degree had higher test accuracy by 1.6%.

4.6.4 Weighting schemes and document models

Weighting schemes populate the majority of the document model, however there is an additional component which is the graph features. These are non-word graph statistics taken over the entire word-word graph. These graph features are agnostic of the term features. This section presents the prediction accuracies of weighting schemes and the document models (weighting schemes and weighting schemes with graph features).

Most weighting schemes had a reduction in performance when graph features were added to the document model (see Figure 4.12). Clustering Coefficient, Term Frequency, Out Degree and In Degree had decreases in test accuracy of 11.96%, 11.52%, 5.44% and 0.54%. Total Degree was able to increase test accuracy by 4.94% when graph features were added to the document model. In Degree was robust to changes in the document model as its test accuracy remained relatively unchanged.

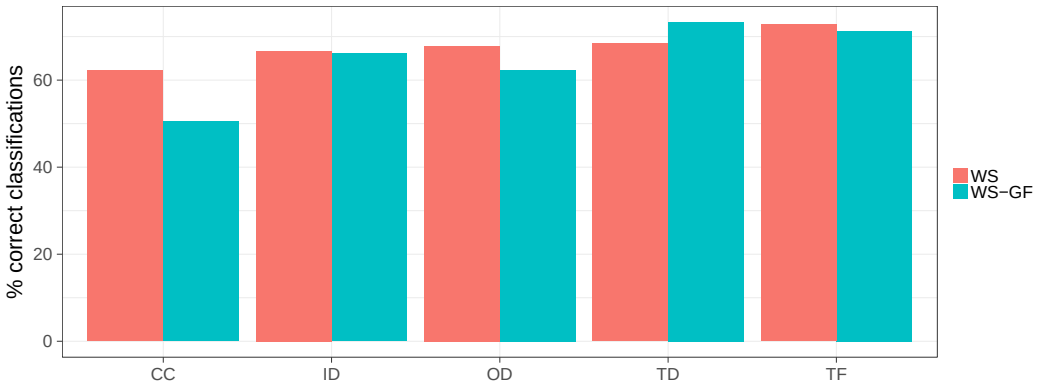

Figure 4.12: Accuracy graphs of weighting scheme and machine learning methods. This graph only shows test accuracy scores. The colours indicate the document models: WS (weighting schemes) in red and WS-GF (weighting schemes with graph features) in blue. The x-axis shows the different weighting schemes: CC (clustering coefficient), ID (in-degree), OD (out-degree), TD (total degree) and TF (term frequency).

4.6.5 Machine learning methods and document models

Graph features are numeric summaries of document graphs while term features represent specific words. Graph features are dense whereas term features are sparse. A machine learning method would ideally have to treat these features differently in order to discriminate between document classes. The prediction accuracies of document models and machine learning methods are presented in 4.13.

Each method responded differently to the addition of graph features to the document model (see Figure 4.13). Naïve Bayes had reduced performance by 22.31% while Neural Networks remained largely unchanged with a reduction in performance of 0.31%. Support Vector Machines alternatively saw an increase in performance from 74.94% to 80.23% in test accuracy.

4.7 Conclusions for Results

Some conclusions regarding the text classification experiments are presented along with some remarks regarding future study.

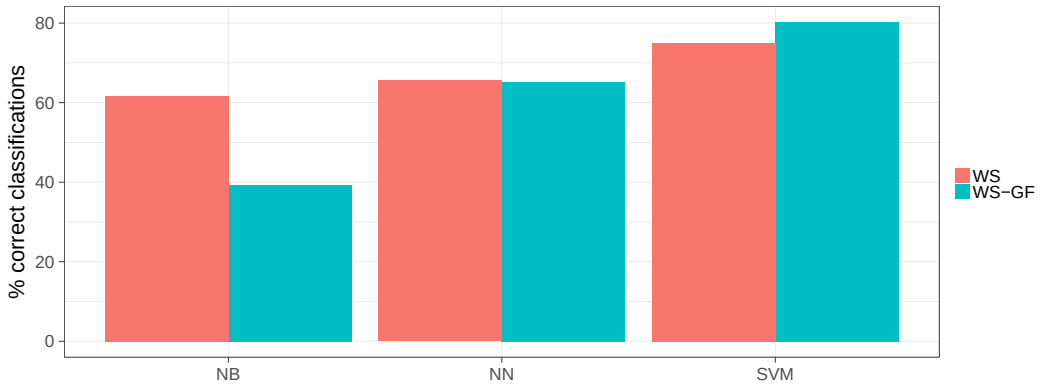

Figure 4.13: Accuracy graphs of weighting scheme and machine learning methods. This graph only shows test accuracy scores. The colours indicate the document models: WS (weighting schemes) in red and WS-GF (weighting schemes with graph features) in blue. The x-axis shows the machine learning methods: NB (naive Bayes), NN (Neural Networks) and SVM (support vector machines).

4.7.1 Discussion

This study was interested in whether graph theory could be exploited to improve test accuracy scores in text classification. A number of different approaches were possible including unipartite and bipartite graph representations of text documents and document collections. Unipartite text document representations were used and two graph approaches were tested

- Do graph weighting schemes increase test accuracy?
- Do graph document models with the term document matrix and graph weighting schemes increase test accuracy?

The graph weighting schemes had 1.4% higher test accuracy on average than Term Frequency. When looked at individually Total Degree was 3.33% higher than Term Frequency. The Support Vector Machines gained the most from graph based weighting schemes in test accuracy. When Total Degree and Term Frequency experienced their highest levels of accuracy under Support Vector Machines, Total Degree had test accuracy 3.6% higher than Term Frequency.

The addition of graph features to the document model decreased test accuracy from 67.15% to 61.57% on average. Naïve Bayes was a large contributor to the general decrease in test accuracy from adding graph features. Neural Networks had a minor decrease in test accuracy of 0.31% while Support Vector Machines benefited from the inclusion

of graph features. Most weighting schemes had decreased test accuracy when graph features were added to the document model. Clustering Coefficient and Term Frequency experienced near 11% decreases in accuracy. Total Degree was the only weighting scheme to experience an increase in test accuracy of nearly 5% when adding graph features. In Degree experienced the smallest decline of 0.56%. Naïve Bayes suffered from the inclusion of graph features while Support Vector Machines benefited.

Some important points are listed below.

- Graph weighting schemes outperformed Term Frequency on average.
- Total Degree outperformed Term Frequency on average by 3.33%
- Graph features decreased Test Accuracy on average. Much of this is attributed to Naïve Bayes.
- Support Vector Machines had higher accuracy with graph features.
- Total Degree both outperformed Term Frequency and had higher performance under graph features.

Overall graph weighting schemes did increase test accuracy on average. The addition of graph features increased test accuracy when Support Vector Machines were used but did not decrease it materially when Neural Networks were used.

4.7.2 Future study

There were several questions and limitation that arose as the research unfolded. These questions and limitations form the basis of future research. The most prominent questions were regarding the graph features. When extracting graph feature from the text documents few features where available to choose from. There was limited literature available on designing a graph based weighting scheme. Near the end of the research an area of graph feature extraction become known. This area is concerned with extracting subgraphs known as motifs. Motifs have been used to detect local structure in the graphs of E-coli (Shen-Orr, Milo, Mangan & Alon 2002), computer networks (Allan, Turkett & Fulp 2009) and human mobility networks (Schneider, CM, et al, 2013).

The local structure captured by motifs have been shown to carry meaningful information about the structure of graphs. In biological networks like E-coli the motifs encode specific interaction. It is possible that motifs could capture the local structure of documents. This structure could contribute to capturing differences in documents that are not traditionally captured by term features. There are challenges with extracting motifs. There are limited

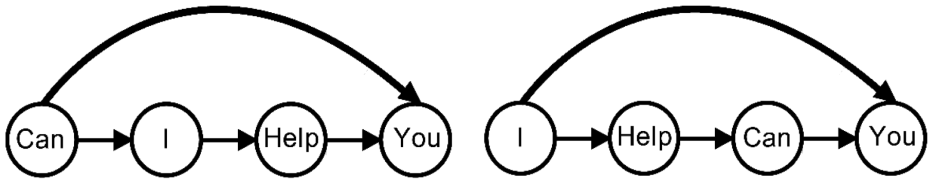

Figure 4.14: Comparison of feed forward Loops. Both loops have the same architecture but the nodes being connected within the architecture are different. This implies that the adjacency matrices will look different thus creating difficulty in identifying that the architectures are indeed the same.

libraries for extracting subgraphs larger than 3 nodes. In addition the extraction of subgraphs is made computationally expensive as the same motif can have many different adjacency matrices associated with it. These classes of adjacency matrices associated with the same subgraph must be mapped. The feed-forward-loop motif is a good example. This is a motif where the first node is connected to the last and the interim nodes are all connected to each other (see Figure 4.14). All the connections are directed forward. The adjacency matrices shown below are both feed-forward-loops. The first starts at *can* and the second at *I*.

$$A = \begin{array}{c|cccc} & Can & I & Help & You \\ Can & 0 & 1 & 0 & 1 \\ I & 0 & 0 & 1 & 0 \\ Help & 0 & 0 & 0 & 1 \\ You & 0 & 0 & 0 & 0 \end{array} \qquad A = \begin{array}{c|cccc} & Can & I & Help & You \\ Can & 0 & 0 & 0 & 1 \\ I & 0 & 0 & 1 & 1 \\ Help & 1 & 0 & 0 & 0 \\ You & 0 & 0 & 0 & 0 \end{array}$$

The extraction of motifs could have extended the number and the types of relationships being extracted. The motifs could be used to model specific linguistic patterns or stylistic traits that might improve or simplify the classification task. They also propose a language agnostic method of analysis which could be applied to other tasks like machine translation.

This exercise could be expanded to more datasets such as to make the conclusions more generally applicable.

Chapter 5

Review of text visualisation and text summarisation literature

This and the next chapter are concerned with the visualisation of text documents through the use of graphs. This chapter develops methods that are used in the framework used to visualise text documents. The focus of this chapter is to investigate how network techniques can be used to produce meaningful visual summaries of text. The methodology used is a combination of part-of-speech tagging (Section 5.4) and graph visualisation (Section 5.5). There are several ways to determine parts-of-speech from text, in this chapter the focus is on Hidden Markov Models (Section 5.4.1) and the Stanford Part-of-Speech tagger (Section 5.4.4) developed by Toutanova, Manning, Klein and Singer (2003). The graphs can be used for visualisation but also analysis. Specifically parts-of-speech can be used to identify words of interest but similarly graph clustering approaches can be used to extract clusters of interest. As such graph methods for visualisation are described (Section 5.5.3) along with graph clustering methods (Section 5.5.1). Both sections of this chapter are multifaceted. This chapter is not meant to review each area or develop it fully but merely introduce important concepts and methods as they relate to the visualisation framework.

The analysis of the framework which follows in Chapter 6 is qualitative in nature. The analysis focuses mostly on developing a framework for visualising text documents. An online tool is developed to illustrate some of the insights found and allow members of the general public to use the techniques.

5.1 Introduction

The internet has given rise to the large scale distribution of information. This includes text based information from textbooks, novels and tweets to journal and newspaper articles. While some of these text documents might come with their own summaries, others do not. This has created a new area of research known as text summarisation. Text summarisation is aimed at creating summaries from text documents using machine learning techniques. This usually falls under Natural Language Processing which is concerned with the interaction between natural text and computers (Mihalcea and Tarau, 2004; Hasan and Ng, 2014).

Text summaries are vital to providing insight to the contents of a document. They do not however provide a different means of engaging with the document. Visual summaries by comparison intuitively convey information and provide a different way of engaging with the document. In this chapter, a text visualisation method is proposed. The method leverages part-of-speech tagging (Toutanova et. al., 2003) and graph theory to produce graph, or network, representations of the text. Words are tagged to identify their part of speech. The document can then, if desired, be filtered on a particular subset of parts of speech e.g. nouns or nouns and verbs. The filtered document is then traversed sequentially to combine words that are within proximity of each other given a distance of a specified number of words. As described in Chapter 2, the words become nodes and their edges can accumulate weights commensurate to the frequency of proximity co-occurrence.

The resulting network can be visualised in two-dimensional space. Force-directed algorithms (Fruchterman and Reingold, 1991; Bannister et al, 2012; Jacomy et al, 2014) have become standard methods for visualising networks. They produce visualisations that are aesthetically pleasing and able to convey meaningful spatial information. Word nodes that are close to each other typically share more edges than nodes that are far apart. The words are nodes that can be sized according to their network statistics. For instance the size of a word node can be determined by the number of words it connects to this is known as degree. This is an intuitive representation of importance. When the networks are large they can become unwieldy, reducing their usefulness. Clustering allows large networks to be broken up into smaller components. These components are networks in their own right that can convey information.

5.2 Text summarisation literature

Text summarisation is concerned with reducing text documents to a collection of phrases or key words that summarise the text. This process can be extractive or abstractive. Abstractive text summarisation produces key words or phrases by augmenting the original text. This is ideally a few coherent sentences about the text. In a sense the abstractive approach is to understand and then explain a text. Extractive text summarisation detects important phrases or words in a document and produces them typically as a list (Hasan and Ng, 2014). This is not usually a coherent sentence. This is similar to key words near the abstract of an academic paper.

Extractive text summarisation can be a heuristic, supervised or unsupervised task. In all cases there are candidate key phrases or words. These are the potential key phrases or words that make up the document. Each task and approach therein tries to find ways of determining which candidates are the best for inclusion in a summary.

Graphs have been used in unsupervised extractive text summarisation for ranking candidate key phrases or words. These approaches treat words or phrases as nodes in a graph. The edge weights are determined by the co-occurrence of words or phrases within the document. A version of PageRank (Page and Brin, 1998), called TextRank (Mihalcea and Tarau, 2004) was adapted for this purpose. These approaches using word-word graphs, word-sentence graphs and sentence-sentence graphs have proved effective in determining key words or phrases (Hasan and Ng, 2014).

5.3 Text visualisation literature

There are myriad methods of text visualisation, Kucher and Kerren (2015) have created an extensive survey of these methods. A few methods are discussed below that focus either on words or phrases. Some methods use external tools like WordNet (Miller, 1995) which allows the methods to integrate synonyms and parts of speech. Others methods use spatial layout to convey information.

Word clouds (Feinberg., 2014) are text visualisations that use colour and size to position words in two-dimensional space. The size is typically indicative of the frequency of the word in the document. The spatial position is designed to allow the various words to fit on a page. The near-random layout of word clouds is a weakness in that it does not use spatial layout to convey information. The colours are also not always meaningful.

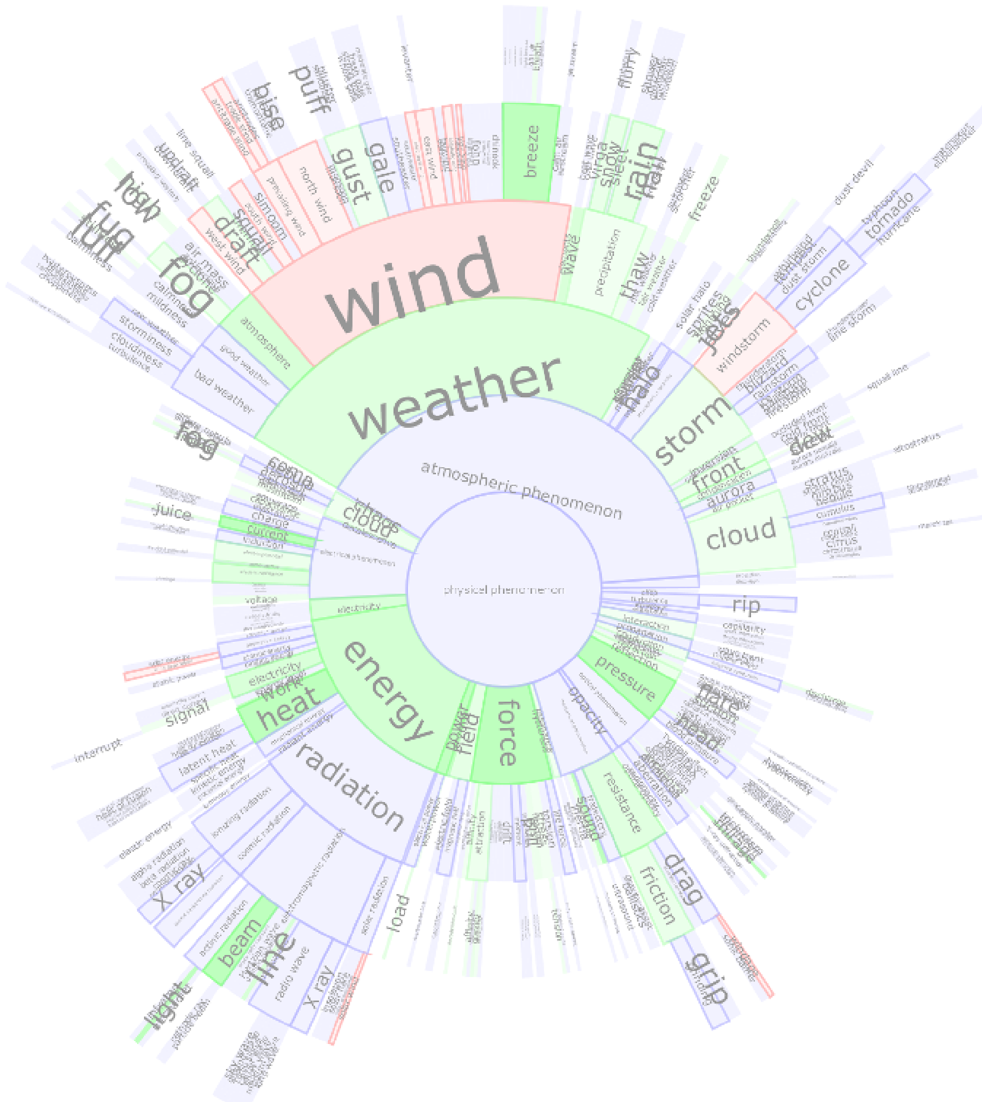

Figure 5.1: DocuBurst (Collins, 2009). Words are displayed in concentric circles with extending from the centre of the image to the to the outer edge. Each concentric circle is divided into a pie containing words that are related to pie in a smaller concentric circle which could be considered a parent. The words are associated through the WordNet hyponym relation (Miller, 1995).

Docuburst (Collins, 2009) (see Figure 5.1) uses the WordNet hyponym relationship to group words with similar meanings. The hyponym relation categorises words in a tree. The top of the tree has broad categories like furniture while the lower in the tree niche

categories like chair and then further down armchair. An example of two words that might be linked as through the hyponym relation are {lawyer, attorney}. Docuburst provides a radial display that can have a large number of elements and appear cluttered (Ham, Wattenberg and Viegas, 2009). ThemeRiver (Harve et. al., 2002) solves this problem by taking a temporal approach. The text is seen as slices each in a different time period. For each slice a few themes are highlighted.

Methods that convey information spatially include TextArc (Paley, 2002) (textarc.org) and graph approaches. TextArc maps co-occurrence information to two dimensional space. Some authors have used graphs which represent co-occurrence of words (Cho, 2009) to visualise text and convey information in space.

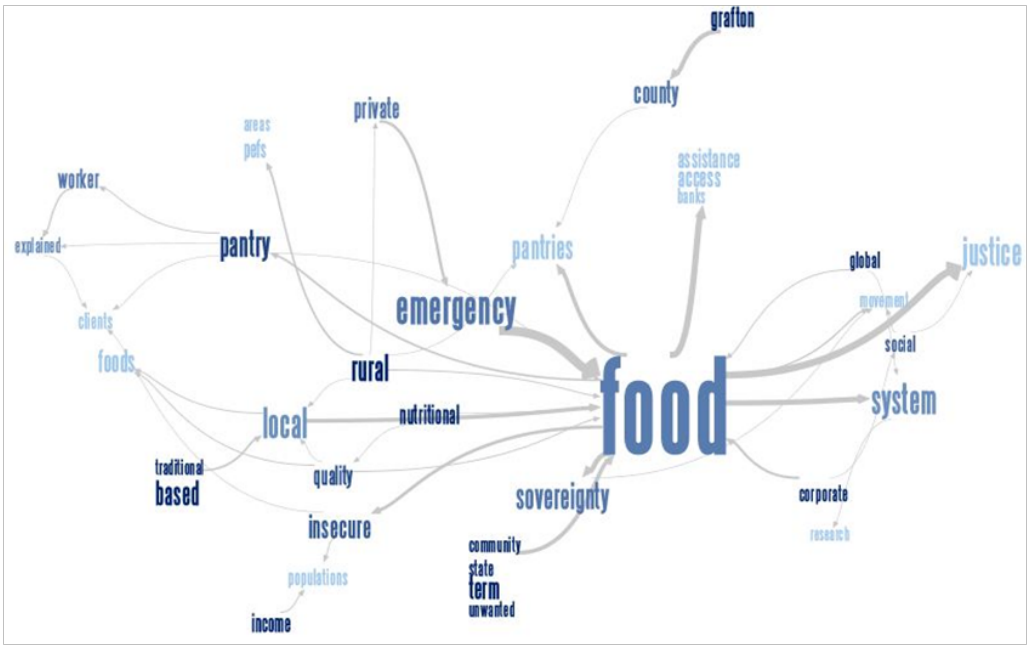

Figure 5.2: Phrase Net (Ham, Wattenberg and Viegas, 2009). Words are displayed as nodes with edges defined through co-occurrence, part-of-speech sequence or suffix sequences. The layout has been adjusted to increase legibility of the text.

Ham, Wattenberg and Viegas (2009) recognised two weaknesses of co-occurrence graph approaches. First, that their layouts can lead to jumbled text. Second that co-occurrence relations were only one 'generic' relation. Ham, Wattenberg and Viegas (2009) responded by developing Phrase Nets which use co-occurrence, part-of-speech (noun noun: mobile network) and suffix sequences (ion: abolition, ment: improvement), to create edges between words. The resulting graph is then visualised by an adjusted algorithm designed to increase

the legibility of text.

Jumbled text can easily be avoided with free graphing tools like Gephi that provide extensive control over graph layouts. If a method provides good insights but poor layout, then it is still worthwhile. In addition, an improvement to a visualisation tool is subjective. Further, the notion that co-occurrence is a 'generic' relation is not very meaningful.

Kim and Kan (2009) found syntactic features (part of speech sequences or suffix sequences) not to be useful in key phrase extraction. In contrast, co-occurrence methods like TextRank and Topical TextRank (Liu et al. 2010) have been shown to be effective in key phrase extraction. No consensus exists on how edges should be defined or what, if anything should be excluded from the visualisation. In this sense, the strength of Phrase Nets is their flexibility. The most informative approaches seem to convey the importance of words through size and relatedness through layout. None of these approaches seem to use colour to convey information. This is true even for methods like Docuburst that incorporate meta-information about the words.

The fields of text summarisation and visualisation both make use of graph methods. Visualisation methods are primarily extractive. Yet unlike key phrase extraction, they do not focus on reducing the amount of information, this is typically relegated to a practical afterthought or by-product, as demonstrated by word clouds, Docuburst and even Phrase Nets. A broader problem in text visualisation is that choices can be arbitrary. By aligning visualisation and summarisation, some arbitrariness can potentially be avoided. For instance, the basis of a visualisation can be a word-word graph where edge weights refer to co-occurrence. The remainder of the chapter investigates this integration of visualisation and summarisation.

The best methods for text summarisation seem to focus on graph weighting of phrases or words. The best methods for visualisation seem to be those that can easily convey the general content of a document, where size and colour are useful tools for conveying information. Our approach uses the text graphs (word-word graphs) that could be used to rank words and visualises them directly. After this step, the graph can be clustered, which produces mutually exclusive sets of words. The graph can convey importance through node size and cluster membership through colour.

5.4 Part-of-speech tagging

State-of-the-art part-of-speech taggers make use of Hidden Markov Models. These models accept words as inputs and produce sequences of tags as outputs. The tags (C_t) are

considered states generated by some unobserved process. The words are considered the result of random variables X_t that are dependent on the states. The Hidden Markov Models are introduced more formally below. A typical observation in a part-of-speech tagging dataset is a sentence, where each word has an associated part-of-speech tag. Consider the sentence below:

There wasn't a bit of trouble

Each word has an associated part-of-speech. This can be attached to each word by creating pairs which we define as word|part-of-speech. The parts of speech that each word in the above sentence are tagged with are listed below:

- EX existential there
- BEDZ was
- AT article (a, the, no)
- NN singular or mass noun
- IN preposition

The sentence below has the associated parts-of-speech attached to each word.

There|*ex* wasn't|*bedz* a|*at* bit|*nn* of|*in* trouble|*nn*

This can be viewed as a state process where each word is an observation generated by a part-of-speech process. As such the word only depends on the part-of-speech from which it emanate. The relationship between words and part-of-speech tags can be represented in terms of the below dependency relationship. Words depend on tags and tags depend on each other, through the markov property.

$$\begin{array}{ccccccccccc} ex & \rightarrow & bedz & \rightarrow & at & \rightarrow & nn & \rightarrow & in & \rightarrow & nn \\ \downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow & & \downarrow \\ There & & wasn't & & a & & bit & & of & & trouble \end{array}$$

5.4.1 Hidden Markov models

A Hidden Markov Model (HMM) has two components. The first is a state process (C_t) which obeys the Markov property. The second is a random variable X_t that follows a

distribution that is only dependent on the value of the state process at time t (Zucchini & MacDonald, 2009). Specifically, the following equations define the HMM.

$$Pr(C_t, \mathbf{C}^{t-1}) = Pr(C_t|C_1, \ldots, C_{t-1}) = Pr(C_t|C_{t-1})$$
$$Pr(X_t, \mathbf{X}^{t-1}, \mathbf{C}^t) = Pr(X_t, C_t)$$

The probability of the HMM for a specific value of x and state i is defined as $p_i(x) = Pr(X_t = x, C_t = i)$. The probability of a given state at a given time is $u_i(t) = Pr(C_t = i)$ Zucchini & MacDonald (2009).

This allows for the marginal distribution of to be evaluated over all the available states at a given time.

$$Pr(X_t = x) = \sum_i^p u_i(t) p_i(x)$$

This process can be rewritten using matrix notation. If $u_i(t)$ is a vector over the states and $p_i(x)$ the the diagonal element of a matrix $\mathbf{P}(x)$ where each column represent a different value of i, then the marginal distribution of x is

$$Pr(X_t = x) = \mathbf{u}(t)\mathbf{P}(x)1'$$

If the parameter process is characterised by a homogeneous Markov chain then $\mathbf{u}(t) = \mathbf{u}(1)\mathbf{\Gamma}^{t-1}$, where $\mathbf{\Gamma}$ the transition probability matrix. The likelihood of the data is the product of the marginal distributions of the data. The likelihood is then defined in this way

$$L(x_1, \ldots, x_n) = \mathbf{u}(1)\mathbf{P}(x_1)\mathbf{\Gamma}\mathbf{P}(x_2)\ldots\mathbf{\Gamma}\mathbf{P}(x_n) = \mathbf{u}(1)\mathbf{P}(x_1)\Pi_{i=2}^n\mathbf{\Gamma}\mathbf{P}(x_i) \tag{5.1}$$

An example from Zucchini & MacDonald (2009) is adapted below.

For an HMM with state process Γ and probability function $p_i(X_t = x)$ as defined below, we can compute the probability of the sequence of observations 111.

$$\Gamma = \begin{bmatrix} \frac{1}{4} & \frac{3}{4} \\ \frac{3}{4} & \frac{1}{4} \end{bmatrix}$$

$$Pr(X_t = x|C_t = 1) = p_1(0) = p_1(1) = \frac{1}{2}$$
$$Pr(X_t = 1|C_t = 2) = p_2(1) = 1$$

Table 5.1: Probabilities for all sequences of length three. Each sequence in the heading of the table has its associated probability below computed in the manner described above.

000	001	010	011	100	101	110	111
0.004	0.027	0.074	0.145	0.027	0.191	0.145	0.387

as such $P(1)$ is defined in the following way

$$P(1) = \begin{bmatrix} p_1(1) & 0 \\ 0 & p_2(1) \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & 1 \end{bmatrix}$$

The first step is to compute δ, the steady state vector for Γ. Γ can be decomposed into its eigenvalue $\mathbf{e}$, eigenvector $\mathbf{v}$ pairs which can then be used to compute δ. The diagonal elements of $\mathbf{e}$ are $e_1 = 1$ and $e_2 = \frac{1}{4}$ which implies $\lim_{n\to\infty} \mathbf{e}_2^n = 0$, thus $\delta = \{\frac{1}{3}, \frac{2}{3}\}$.

$$\begin{aligned}
\Gamma &= \mathbf{v}\mathbf{e}\mathbf{v}^{-1} \\
\lim_{n\to\infty} \Gamma^n &= \lim_{n\to\infty} \mathbf{v}\mathbf{e}^n\mathbf{v}^{-1} \\
\lim_{n\to\infty} \Gamma^n &= \lim_{n\to\infty} \begin{bmatrix} 0.7071068 & -0.7071068 \\ 0.7071068 & 0.7071068 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & -\frac{1}{2} \end{bmatrix}^n \begin{bmatrix} 0.7071068 & 0.7071068 \\ -0.7071068 & 0.7071068 \end{bmatrix} \\
&= \begin{bmatrix} 0.7071068 & -0.7071068 \\ 0.7071068 & 0.7071068 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0.7071068 & 0.7071068 \\ -0.7071068 & 0.7071068 \end{bmatrix} \\
&= \begin{bmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{bmatrix}
\end{aligned}$$

The probability of the sequence is expressed in the following way.

$$\begin{aligned}
L(1,1,1) &= \delta P(1)\Gamma P(1)\Gamma P(1)1' \\
L(1,1,1) &= \begin{bmatrix} \frac{1}{2} & \frac{1}{2} \end{bmatrix} \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{4} & \frac{3}{4} \\ \frac{3}{4} & \frac{1}{4} \end{bmatrix} \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{4} & \frac{3}{4} \\ \frac{3}{4} & \frac{1}{4} \end{bmatrix} \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} \\
&= 0.3867188
\end{aligned}$$

We can compute the probability of all the other sequences of length three in a similar way. The results are provided below.

5.4.2 Viterbi algorithm

The likelihood is useful for language modelling in the field of natural language processing. It can answer a question like, *What is the probability of these words appearing in a sentence*?

In part-of-speech tagging and named-entity-recognition the questions are typically, *What are the parts of speech, or named entities associated with these words*? For the latter tasks, the optimal sequence of tags is more important than the sequence themselves.

Determining the optimal sequence by testing every combination of tags could take $(\text{number of tags})^{\text{number of words}}$ operations. For long sentences and large numbers of tags this could be computationally costly. The Viterbi algorithm has been used to successfully reduce the search space such that it only depends linearly on the number of words in a sentence. The likelihood function above (see Equation (5.1)) can be rewritten in the following way (Zucchini & MacDonald, 2009):

$$L(x_1, \ldots, x_n, c_1, \ldots, c_n) = \mathbf{u}(1)\mathbf{P}(x_1)L(x_2, \ldots, x_n, c_2, \ldots, c_n)$$

The set of all states is given as $S = c_1, \ldots, c_n$. The maximum of this likelihood with respect to the states that produce the maximum is $\max_{c_1,\ldots,c_n \in \mathbf{S}} L(x_1, \ldots, x_n, c_1, \ldots, c_n) = \max_{c_1,\ldots,c_n \in \mathbf{S}} \{\mathbf{u}(1)\mathbf{P}(x_1)L(x_2, \ldots, x_n, c_2, \ldots, c_n)\}$ (Zucchini & MacDonald, 2009).

We can start by addressing

$$\begin{aligned} \max_{c_1 \in \mathbf{S}} L(x_1, c_1) &= \max_{c_1 \in \mathbf{S}} \mathbf{u}(1)\mathbf{P}(x_1) \\ &= \max_{c_1 \in \mathbf{S}} Pr(C_1|C_0)Pr(X_1|C_1) \\ &= Pr(c_1^*|c_0)Pr(X_1|c_1^*) \end{aligned}$$

Here C_0 is a state before the beginning of the sequence (Toutanova et. al., 2003). This is represented by a wild card ($*$) in the literature. The conditional probability of a state with respect to a wild card is the same as the marginal probability of the state. The above problem can be solved by computing the likelihood for each state and selecting the state that produces the largest likelihood. This provides a state that optimises the likelihood $L(x_1, c_1)$, c_1^*. We can then address the next part. It is important to note that because c_1^* has already been determined only c_2^* need be determined in the next step. This means that for each subsequent step only $|S|$ computations are needed to arrive at each subsequent optimal sequence.

$$\begin{aligned} \max_{c_2,c_1} L(x_1, x_2, c_1, c_2) &= \max_{c_2,c_1} \{Pr(C_1|C_0)Pr(X_1|C_1)Pr(C_2|C_1)Pr(X_2|C_2)\} \\ &= \max_{c_2} \{Pr(c_2|c_1^*)Pr(X_n|c_2)L(x_1, c_1^*)\} \end{aligned}$$

This is called the Viterbi algorithm. It allows for the computation of the optimal sequence

of states in $O(k|S|)$ computations where k is the length of a sequence and $|S|$ the total number of states (Zucchini & MacDonald, 2009).

$$\begin{aligned}\pi(1, c_1) &= \max_{c_1 \in \mathbf{S}} L(x_1, c_1) \\ \pi(2, c_2) &= \max_{c_2, c_1} L(x_1, x_2, c_1, c_2) \\ &= \max_{c_2 \in \mathbf{S}} \{Pr(C_2|c_1^*) Pr(X_2|C_2) \pi(1, c_1)\}\end{aligned}$$

Recursively the Viterbi algorithm is written as shown below.

$$\pi(k, c_k) = \max_{c_1, \ldots, c_k \in \mathbf{S}} \{Pr(C_k|c_{k-1}^*) Pr(X_k|C_k) \pi(k-1, c_{k-1}^*)\}$$

There is a final step which is to predict the probability that a sequence comes to an end:

$$\max_{c_1 \in \mathbf{S}} L(x_1, \ldots, x_n, c_1, \ldots, c_k, c_{k+1}) = \pi(k, c_k) Pr(C_{k+1}|C_k)$$

This requires computing $Pr(C_{k+1}|C_k)$ for each value that C_k can take in S. C_{k+1} is always the $STOP$ state. The computational time of determining all states that maximise the likelihood of a given set of observations at $O(k|S|^2)$. This is lower for large values of $|S|$ and k than the time it would otherwise take, $O(|S|^k)$ (Zucchini & MacDonald, 2009).

Example: When the likelihood for the observed sequence 101 is computed it comes to 0.1914062. Yet, there are 8 (2^3) permutations of states that can produce the observed sequence 101. The Viterbi algorithm can produce the probability associated with the most probable sequence of states. It can also produce the sequence of states. Next we demonstrate the Viterbi algorithm with a few computations. Given an initial steady state vector δ and the first element in the sequence $P(1)$ we can compute $\pi(1, c_1)$ the highest probability associated with the first observation 1 given the gamma matrix defined above. This yields a state of 2 and probability of $\pi(1, c_1) = 1/2$

$$\begin{aligned}\pi(1, c_1^*) &= \max(\delta P(1)) \\ &= \max\left(\begin{bmatrix}\frac{1}{2} & \frac{1}{2}\end{bmatrix}\begin{bmatrix}\frac{1}{2} & 0 \\ 0 & 1\end{bmatrix}\right) \\ &= \max\left(\begin{bmatrix}\frac{1}{4} & \frac{1}{2}\end{bmatrix}\right) \\ &= \frac{1}{2}\end{aligned}$$

We can repeat these computations for the second observation which is 0, by computing $\pi(2, c_1^* = 2, c_2) = 1/8$ with a state of 1.

$$\begin{aligned}\pi(2, c_1^* = 2, c_2) &= \max(\pi(1,2)\Gamma P(0)) \\ &= \max\left(\frac{1}{2}\begin{bmatrix}\frac{1}{2} & \frac{1}{2}\end{bmatrix}\begin{bmatrix}\frac{1}{2} & 0 \\ 0 & 0\end{bmatrix}\right) \\ &= \max\left(\begin{bmatrix}\frac{1}{8} & 0\end{bmatrix}\right) \\ &= \frac{1}{8}\end{aligned}$$

We can continue to the third observation also 1, $\pi(2, c_1^* = 2, c_2^* = 2, c_3) = 1/16$ also, with a state of 2.

$$\begin{aligned}\pi(3, c_1^* = 2, c_2^* = 1, c_3) &= \max(\pi(2,2)\Gamma P(1)) \\ &= \max\left(\frac{1}{8}\begin{bmatrix}\frac{1}{2} & \frac{1}{2}\end{bmatrix}\begin{bmatrix}\frac{1}{2} & 0 \\ 0 & 1\end{bmatrix}\right) \\ &= \max\left(\begin{bmatrix}\frac{1}{32} & \frac{1}{16}\end{bmatrix}\right) \\ &= \frac{1}{16}\end{aligned}$$

Two things emerge from the Viterbi algorithm. The first is that the probability associated with the most likely state sequence to produce 101 is $\frac{1}{16}$. The second is that in each successive round when the max is taken it is selected either from state 1 or state 2. As such the state sequence producing the probability can be deduced by changing max to argmax in the above expression. This state sequence is 212.

5.4.3 HMM's for text

An example of a HMM applied to par-of-speech tagging is presented below. This will demonstrate practical considerations for part-of-speech tagging, the format of the data and how predictions and tag sequences are retrieved from the model. In a Trigram Hidden Markov Model (HMM) the assumption is that the two previous states of the process effect the current state. This parameter process is generally referred to as a 2nd order Markov Chain. It no longer obeys the Markov property. This kind of model is more powerful than the first order Markov Chain in that it accepts greater levels of dependency in the data. This naively should improve the accuracy of a model with serial dependence like a Part-of-speech tagger.

The Trigram HMM is defined in the following way

$$\begin{aligned}Pr(C_t = i | C_{t-2}, C_{t-1}) &= u_i(t) \\ Pr(X_t = x | C_t = i) &= p_i(t)\end{aligned}$$

Both C_t and X_t are random variables the realisations of which are states and words respectively. There are two extra features added for convenience. They include the wild card (*) and $STOP$ which are added to the states. As such the set of states is $(STOP, *) \cup \mathbf{S} = \mathbf{S}*$.

The wild card is the value assigned to states C_0 and C_{-1}. The stop character is assigned to states C_{n+1}. This latter state is added to the end of all state sequences. Therefore the probability of a sequence of observations $x_1, \ldots, x_n$ is

$$L(x_1, \ldots, x_n, c_1, \ldots, c_{n+1}) = \prod_{i=1}^{n} Pr(x_i|c_i) \prod_{i=1}^{n+1} Pr(c_i|c_{i-2}, c_{i-1})$$

Model parameters can be estimated through maximum likelihood. For the tag sequence (c_{i-2}, c_{i-1}, c_i), the maximum likelihood estimates are

$$Pr(c_i|c_{i-2}, c_{i-1}) = \frac{c(c_i, c_{i-1}, c_{i-2})}{c(c_{i-1}, c_{i-2})}$$

and, for the words

$$Pr(x_i|c_i) = \frac{c(c_i \to x)}{c(c_i)}$$

where $c(a, b, c)$ denotes the number of times the tag sequence (a, b, c) has been observed, and $r \to a$ denotes the number of times that a state/tag r has been associated with an observation/word a. If the count of nouns is 327 ($c(N) = 327$) and count of vote is 4 ($c(vote) = 4$) then the count of vote and noun is less than or equal to 4. Let us say it is ($c(N \to vote) = 3$) then the conditional probability of generating the word vote with respect to the state noun is therefore $Pr(vote|noun) = \frac{3}{327}$.

The Viterbi algorithm defined above in Section 5.4.2 can be used to determine the optimal tag sequence for each observed sentence. The computational time would be $O(k|S|^3)$.

5.4.4 Best in class part-of-speech tagger

Toutanova, Manning, Klein and Singer (2003) created a tagger which at the time was the state of the art. It achieved 97.24% accuracy on the Penn State Treebank WSJ dataset and included various improvements over previous approaches. First it included both the forward and backward dependencies in modelling as opposed to just the forward dependency. This shifted the modelling approach from a Hidden Markov Model to a

Dependency Network. They also included more lexical information which had traditionally been used in language parsers but not part-of-speech taggers. Language parsers turn raw natural text into structured information. This could include extracting dates, names, objects and subjects from text.

The main difference between a Hidden Markov Model and the Bidirectional Dependency Network used by Toutanova et. al., (2003) is that the Bidirectional Dependency Network uses both previous and future information when modelling the current observations.

$$Pr(C_t = i|C_{t-1}, C_{t+1}) = u_i(t)$$
$$Pr(X_t = x|C_t = i) = p_i(t)$$

This makes it difficult to estimate $Pr(C_t)$ as its conditional distribution is based on $Pr(C_{t-1})$ which is also dependent on $Pr(C_t)$. These marginal probabilities can only be estimated through Gibbs Sampling.

While it is possible to use smoothed counts to determine the maximum likelihood estimates for the model, Toutanova et. al., (2003) instead chose to use multiple logistic regression models to determine the probability distribution of individual tags. This helps with underflow when there are many variables on which a tag is being conditioned on. Here λ is a more generalised version of c. It measures the counts of the states or observations that are associated with it using the same notation. Therefore $\lambda(c_t, c_{t-1}) = c(c_t, c_{t-1})$ and $\lambda(c_t, x_t) = c(c_t \rightarrow cx_t)$

$$u_i(t) = \frac{\exp(\lambda(c_t, c_{t-1}) + \lambda(c_t, x_t) + \lambda(c_t, c_{t+1}))}{\sum_{c'_t \in \mathbf{S}} \exp(\lambda(c'_t, c_{t-1}) + \lambda(c'_t, x_t) + \lambda(c'_t, c_{t+1}))}$$

With the probabilities for individual tags determined the remainder of the inferential process is the same as that of HMM's. The selected sequences are not selected on the basis of their probabilities but instead their scores.

$$score(\mathbf{x}) = \Pi_{i=1}^{|x|} Pr(x_i|Pa(x_i))$$

$Pa(x_i)$ denotes the parents of (x_i) or the variables, on which its probability depends. This is a score as opposed to a marginal distribution as described above. This produces two problems. The first is that highly probable sequences do not always get selected. The example below demonstrates how a highly probable sequence might not be selected. First the sequence is presented and then the score and probability values are compared. When

the largest score value is used as the criteria to select the sequence, sequences that are less probable are selected.

- this is a set of observed sequences: ({1,1},{1,1},{1,1},{1,2},{2,1},{3,3}). Each sequence is of the form $\{x_1x_2\}$ where x_1 is the parent of x_2. The parents of 1 are 1 ({1,1}) and 2 ({2,1}), the parent of 2 is 1 ({1,2}) and the parent of 3 is 3 ({3,3}).
- the score of (11) is given as the product of the conditional probabilities of each element of the sequence and with their parents: $score(11) = P(1|1) \times P(1|1) = \frac{3}{4} \times \frac{3}{4} = \frac{9}{16}$
- the score of (33) is given as in the manner described above for (11): $score(33) = P(3|3) \times P(3|3) = 1 \times 1 = 1$
- The score of (33) is higher than the score of (11). The probability of the sequences is given by the frequency of the sequence relative to the total number of sequences. Therefore the probability of P(11) = $\frac{3}{6}$ is higher than the probability of p(33) = $\frac{1}{6}$

The loss function is defined as the negative sum of the log elements of the score, where the elements of the score are $Pr(x_i|j)$.

$loss = -\Sigma_{j \in Pa(x_i)} \log(Pr(x_i|j))$

The second problem is that it is possible to have small or zero loss function while the training set error is still high.

- In a dataset of these observations: ({1,1},{2,2})
- The loss is always 0: $loss = -\Sigma_{j \in Pa(x_i)} \log(Pr(x_i|j)) = -log(Pr(11)) - log(Pr(22)) = -log(1) - log(1) = 0$
- The probability of either sequence in the data is 50%, $P(11) = P(22) = \frac{1}{2}$

The expected model accuracy is 50%. This accuracy cannot increase as the loss is zero and therefore cannot be used to train the model. Toutanova et. al., (2003) determined that these issues had limited impact on the practical application of the model. This was due to the features connecting tags and observations being effective even in the face of the theoretical concerns.

The state of the art model by Toutonova et al (2003) could be described as a Bidirectional Bigram Word and Bidirectional Trigram Tag model. Concretely the probability of a tag is described in the following way

$$P(C_t = i|C_{t-2}, C_{t-1}, C_{t+1}, C_{t+2}, X_t, X_{t-1}, X_{t+1})u_i(t)$$

with

$$u_i(t) = \frac{\exp\left(\sum_{b=-2}^{1} \lambda(c_t, c_{t+b}) + \lambda(c_t, x_{t-1}) + \lambda(c_t, x_t) + \lambda(c_t, x_{t+1})\right)}{\sum_{c'_t \in \mathbf{S}} \exp\left(\sum_{b=-2}^{1} \lambda(c'_t, c'_{t+b}) + \lambda(c'_t, x_{t-1}) + \lambda(c'_t, x_t) + \lambda(c'_t, x_{t+1})\right)}$$

5.5 Graphs for visualisation

Graph are used to visualise text data in Chapter 6. There are a few visual cues that make graphs particularly useful in this regard. First they convey space and size naturally through various automatic visualisation methods. Second they can easily be clustered, taking advantage of the spatial layout. This section describes how graphs can be constructed, clustered (Section 5.5.1) and finally visualised (Section 5.5.3).

The nodes in a text graph are words. The edges represent the co-occurrence of words within a pre-specified window of words from each other. The edge weights are derived from the number of times the words co-occur within the window. For example, if a window length of five is specified, and two words appear twelve times within this distance of each other then an edge will exist between the words with an edge weight of twelve. For further detail on constructing text graphs please refer to Chapter 2.

5.5.1 Modularity clustering

The detection of community structure in social, biological and computer networks is increasingly of value. Social networking websites are interested in detecting related groups of users. Search engines are interested in related groups of websites. In the biological sciences, groups of neurons and genes can be grouped to uncover meaningful behaviour. The goal of community detection is to uncover latent groups in data. This implies that good community detection methods should not admit a specified number of communities but they should admit when no community structure exists.

Fortunato (2010) provides an extensive summary of graph clustering methods. The focus here is on modularity clustering and the eignevector approach to optimising modularity (Newman, 2006). A good cluster is defined as one where there are more edges within the clusters than there are between the clusters. Modularity clustering maximises modularity which is a measure of the number of edges within clusters as opposed to between them.

The method devises an algorithm for splitting a graph into two clusters and then iteratively splitting each cluster until no appropriate split is possible.

For a graph with n nodes, a node i will be in the first cluster if its label is $s_i = 1$ and in the second cluster if its label is $s_i = -1$. For a pair of nodes the number of edges joining them would be represented by the entry A_{ij} in the adjacency matrix. These values would be non-negative integers. The degree of a node is given by k_i. As such the number of edges between a pair of nodes is given by $\frac{k_i k_j}{\sum_{l=1}^{n} k_l}$ (j a different node) when the edges are placed at random. Modularity is defined in the following way by Newman (2006).

$$Q = \frac{1}{4m} \sum_{i=1}^{n} \sum_{j=1}^{n} \left(\mathbf{A}_{ij} - \frac{k_i k_j}{2m} \right) (s_i s_j + 1) \text{ where } m = \frac{1}{2} \sum_{i=1}^{n} k_i \tag{5.2}$$

If the number of actual nodes is greater than the expected number of nodes then the contribution to modularity is positive. If two nodes are in the same group ($s_i s_j = 1$) then their contribution to modularity is positive. The challenge is then to find the node labels that optimise modularity (Newman, 2006).

$$\begin{aligned} Q &= \frac{1}{4m} \sum_{i=1}^{n} \sum_{j=1}^{n} \left(\mathbf{A}_{ij} - \frac{k_i k_j}{2m} \right) (s_i s_j + 1) \\ &\approx \frac{1}{4m} \sum_{i=1}^{n} \sum_{j=1}^{n} \left(\mathbf{A}_{ij} - \frac{k_i k_j}{2m} \right) s_i s_j \\ &= \frac{1}{4m} \mathbf{s}^T \mathbf{B} \mathbf{s} \text{ with } \mathbf{B}_{ij} = \mathbf{A}_{ij} - \frac{k_i k_j}{2m} \end{aligned} \tag{5.3}$$

By rewriting modularity in matrix notation (Equation (5.3)) the problem can be reduced to one of finding and maximising the largest eigenvector of the matrix $\mathbf{B}$ the modularity matrix (Newman, 2006). The modularity matrix can be decomposed as shown in Equation (5.4) such that each normalised eigenvector $\mathbf{u}_i$ is associated with an eigenvalue β_i. For the largest positive eigenvalue the challenge is to find values such that it is maximised.

$$\begin{aligned} Q &= \frac{1}{4m} \sum_{i=1}^{n} a_i \mathbf{u}_i^T \mathbf{B} \sum_{j=1}^{n} a_j \mathbf{u}_j \\ &= \frac{1}{4m} \sum_{i=1}^{n} (\mathbf{u}_i^T \mathbf{s})^2 \beta_i \text{ where } \mathbf{s} = \sum_{i=1}^{n} a_i \mathbf{u}_i \end{aligned} \tag{5.4}$$

Since the cluster labels are not provided the labels can be determined by finding the eigenvector, eigenvalue pairs of the modularity matrix. The signs of the largest eigenvector can then be used to separate the nodes into two classes. If all the nodes are the same sign then there is no split. From the first two clusters the sub-graphs can be individually clustered until no sub-graphs exist.

5.5.2 Modularity clustering example

The karate club graph developed by Zachary (1977) is a good graph to demonstrate the application of modularity clustering. The graph has 34 nodes and 156 edges. The adjacency matrix is too large to display as such the graph is plotted below.

We illustrate the calculation of entries in the B matrix by considering two nodes, say v_1 and v_2, with degree 16 and 9 respectively, and with an edge between them. The expected number of edges between v_1 and v_2 is given by $16 * 9/156 = 0.923$. As the actual number of edges is 1, this gives $\mathbf{B}_{12} = 1 - 0.923 = 0.077$. Once the whole of $\mathbf{B}$ is computed in similar fashion, its eigen decomposition can be used to cluster the nodes. Specifically all the nodes corresponding to the eigenvector values for the largest eigenvalue with the same sign will be clustered together.

This provides a binary split of the graph. To form subsequent clusters this process can be repeated on the subgraphs as demonstrated below. The process comes to an end when the subgraphs can not be subdivided.

5.5.3 Force-directed algorithms for graph visualisation

Force-directed algorithms visualise graphs in two- or three-dimensional space. The objective is to produce aesthetic visualisations by reducing the graph to a physical system. The algorithms that solve these systems are Force-Directed Algorithms (Bannister et al, 2012). This sections discusses the Fructherman Reingold algorithm by Fruchterman and Reingold (1991). This is an older algorithm which is successful in creating aesthetic layouts for graphs. The next section discusses Force Atlas 2 a newer force directed algorithm which provides some additional functionality, specifically scaling and the prevention of node overlap.

They start with graph defined as a pair $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ of nodes $\mathbf{V}$ and edges $\mathbf{E}$. Their algorithm treats every pair of nodes $v_i, v_j \in \mathbf{V}$ as possessing repulsive forces $f_r(z) = \frac{k^2}{z}$. Those nodes that are connected by an edge have attractive forces $f_a(z) = \frac{z^2}{k}$. The

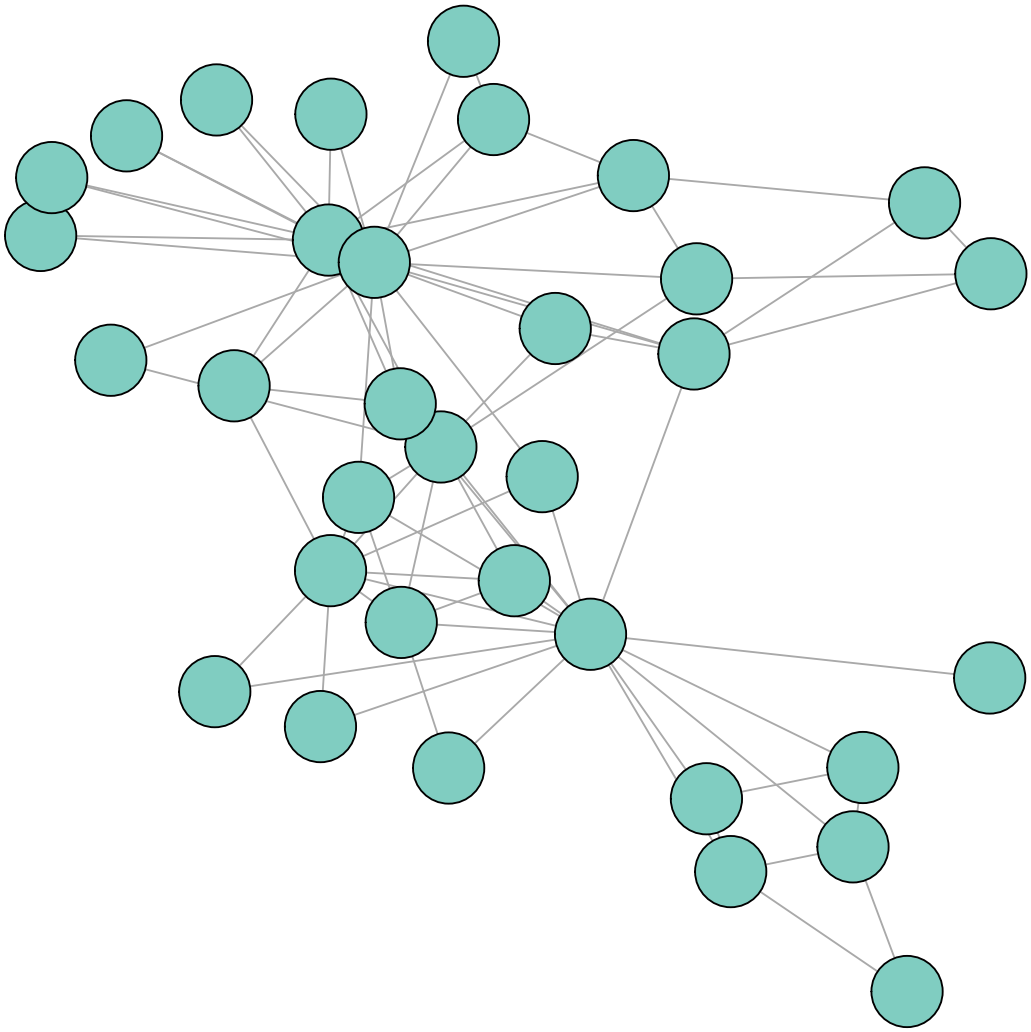

Figure 5.3: The Karate Club graph. The nodes are not yet clustered but the visualisation method shows areas of density and separation. A good visualisation should create clusters in these segments.

Table 5.2: Eigen values for each node in the karate example. Each row provides the ID of two nodes along with their eigenvalues and associated cluster. Nodes with negative eigenvalues are assigned to cluster one whereas nodes with positive eigenvalues are assigned to cluster two.

Node	Eigenvalue	Cluster	Node	Eigenvalue	Cluster
1	-0.388	1	18	-0.132	1
2	-0.270	1	19	0.139	2
3	-0.132	1	20	-0.058	1
4	-0.253	1	21	0.139	2
5	-0.134	1	22	-0.132	1
6	-0.146	1	23	0.139	2
7	-0.146	1	24	0.217	2
8	-0.209	1	25	0.056	2
9	0.054	2	26	0.075	2
10	0.048	2	27	0.116	2
11	-0.134	1	28	0.103	2
12	-0.078	1	29	0.068	2
13	-0.129	1	30	0.206	2
14	-0.135	1	31	0.096	2
15	0.139	2	32	0.102	2
16	0.139	2	33	0.324	2
17	-0.059	1	34	0.370	2

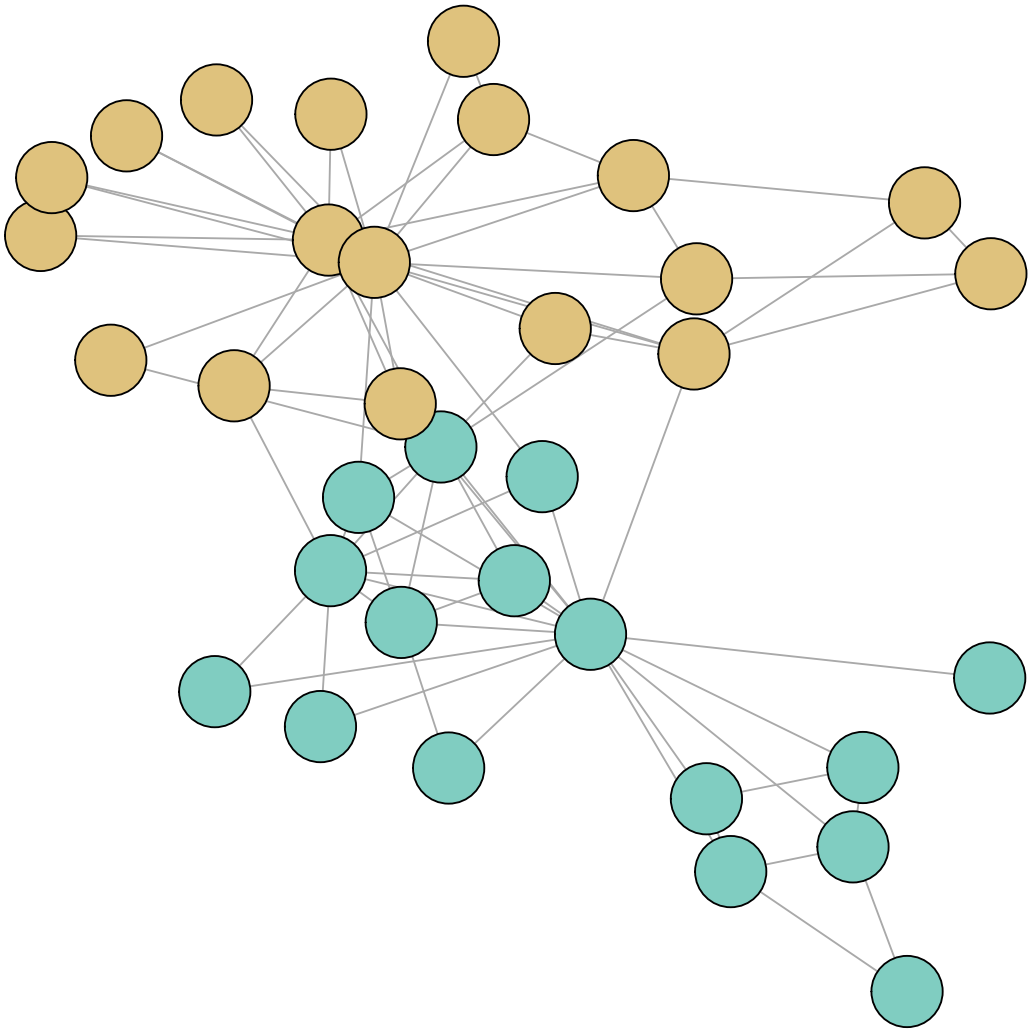

Figure 5.4: Karate Club dataset first clustering. The data is separated into two halves where the edges seem more clustered at in the top and bottom halves but not between them.

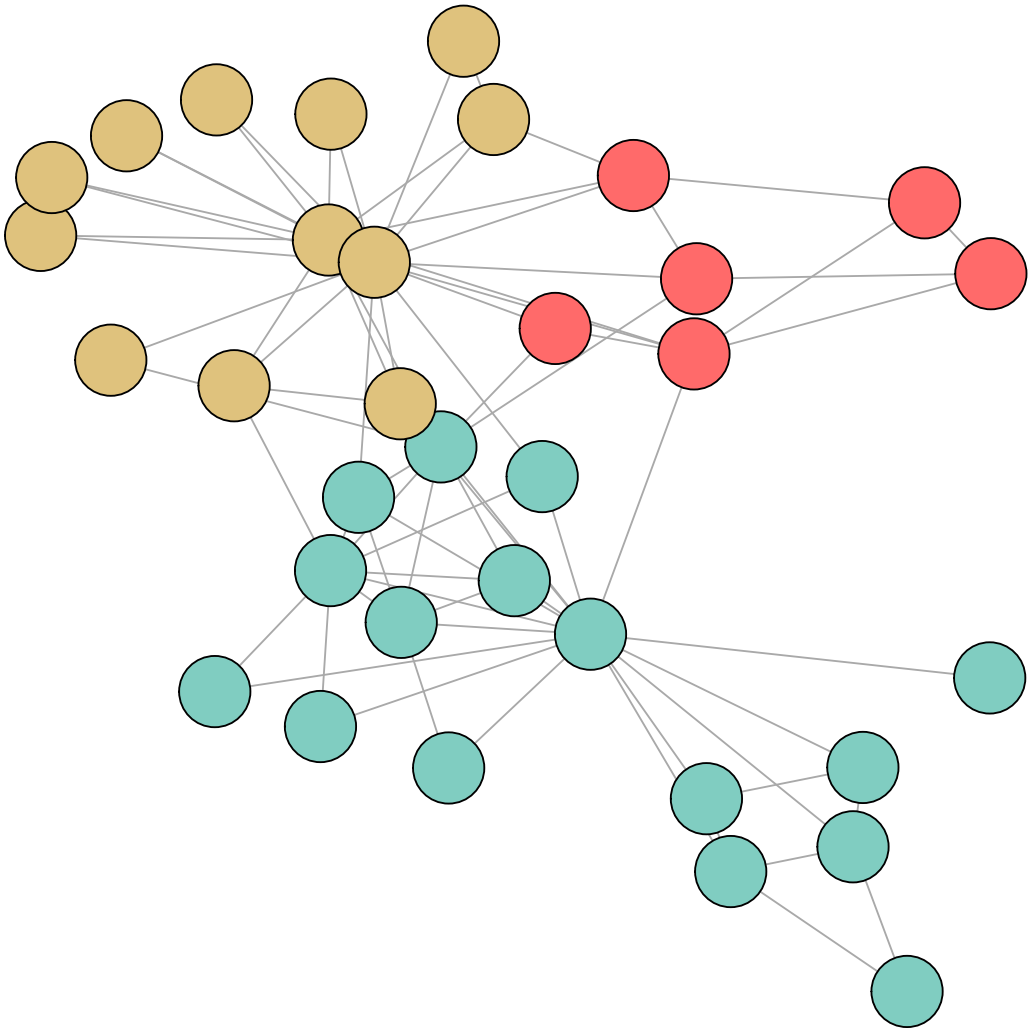

Figure 5.5: Karate Club second clustering. The first cluster is sub-divided into two clusters with many nodes in separate clusters not interacting.

attractive forces increase as the distance between two connected nodes increases. The repulsive forces decrease as two nodes become further apart. The nodes have two properties, position ($v_i.pos$) and displacement ($v_i.disp$). The algorithm is iterative, updating the position of the nodes by less and less as the layout improves. When these forces are in balance the algorithm is in equilibrium. The distance measure used is Euclidean.

$$\Delta_{ij} = v_i.pos - v_j.pos$$

An example is provided for a four node, four edge graph. The first, second and third nodes form a fully connected graph. The fourth node is only connected to the third. The adjacency matrix is provided below:

$$A = \begin{array}{ccccc} 1 & 2 & 3 & 4 & \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 2 \\ 1 & 1 & 0 & 1 & 3 \\ 0 & 1 & 1 & 0 & 4 \end{array}$$

As such an aesthetic layout would be one that puts the first three nodes in an equilateral triangle and leaves the fourth node closest to the third but equally far from both the first and second nodes. The Frutherman Reingold algorithm can achieve this layout as follows.

First the frame is set with width (W) and height (L) 20000 pixels. The constant k for the forces is computed as $k = \sqrt{W * L/n}$ where n is the number of nodes. The positions of the nodes is randomised on a stage with 20000 pixels across and 20000 pixels high. Figure 5.6 shows the layout of the random initialisation. The nodes are numbered such that in subsequent iterations of the algorithm it is clear which node moved to which position. The black lines are the edges connecting the nodes.

The algorithm, iteratively computes the repulsive forces between all pairs of nodes, and then computes the attractive forces between the connected nodes. Finally these forces are combined in the displacement of the vertices which move the nodes. The displacement is reduced as the number of iterations increases.

Next the layout algorithm is described. In the first portion of the algorithm, every node applies a repulsive force on every other node. The repulsive force is scaled by the difference between the nodes divided by their Euclidean distance. The Euclidean distance of x is given by $||x||_2$ where as the magnitude of the repulsive force is given by $f_r(z)$ as defined above.

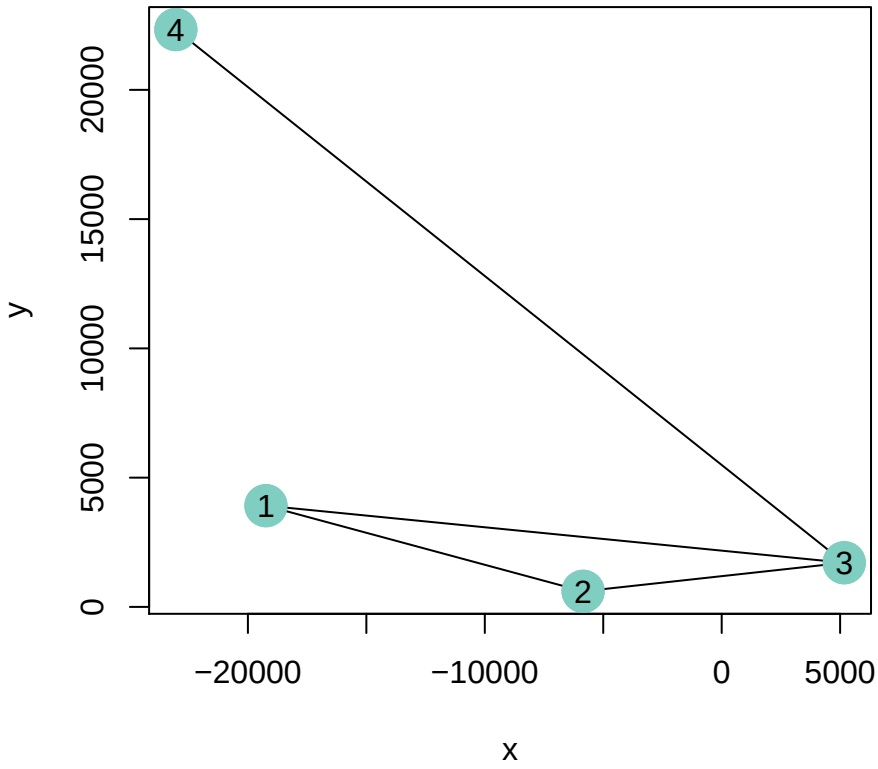

Figure 5.6: Random initialisation of the Fruchterman Reingold layout in a 20000 x 20000 grid.

Algorithm 1 Compute repulsive forces

```
1:  procedure COMPUTE FORCES ON NODES
2:      for i in 1:|V| do
3:          v_i.disp = 0
4:          for j in 1:|V| do
5:              if i ≠ j then
6:                  v_i.disp = Δ_ij/||Δ_ij||_2 × f_r(||Δ_ij||_2)
7:              end if
8:          end for
9:      end for
10: end procedure
```

The edges are pairs $e = i, j$ where i and j are vertices. Thus $v_{e_1} = v_i$. All nodes linked by an edge exert attractive forces on each other. The attractive force defined as $f_a(z)$ is scaled by the difference of the nodes (Δ_{e_1,e_2}) divided by the Euclidean distance of the nodes. This is also added to the total displacement of the nodes.

Algorithm 2 Compute attractive forces

1: **procedure** COMPUTE FORCES ON ALONG EDGES
2: **for** e in E **do**
3: $\Delta_{e_1,e_2} = v_{e_1}.pos - v_{e_2}.pos$
4: $v_{e_1}.disp = v_{e_1}.disp - \Delta_{e_1,e_2}/||\Delta_{e_1,e_2}||_2 \times f_a(||\Delta_{e_1,e_2}||_2)$
5: $v_{e_2}.disp = v_{e_2}.disp + \Delta_{e_1,e_2}/||\Delta_{e_1,e_2}||_2 \times f_a(||\Delta_{e_1,e_2}||_2)$
6: **end for**
7: **end procedure**

In Figure 5.7 the forces are demonstrated. The first image shows the random initialisation of the nodes. The second image (top right) shows the effect of the attractive forces. The attractive force $f_a(z) = \frac{z^2}{k}$ becomes stronger as the nodes are further apart. As the attractive force exerts a powerful force on node 3 in the direction of node 4. Nodes 1 and 2 are pushed in the direction of node 3. The third image (bottom left) shows the effect of the repulsive forces. Node 4 is pushed in the direction of 3. The repulsive force $f_r(z) = \frac{k^2}{z}$ becomes weaker as the nodes are further apart. As a result all the nodes are pushed away from the centre but with relatively less force than the attraction. Finally the forces are combined and the final displacement of the first iteration is shown the final image of Figure 5.7 (bottom right). The image is more indicative of the attractive forces as the force of the attraction was much greater than that of the repulsion.

The last step of the algorithm is to change the position of each node by its total displacement $v_i.disp$. The algorithm has a cooling parameter t which is reduced with each iteration. This cooling parameter is designed to reduce the change in displacement of node. The displacement is not added directly to the position of the node, instead the minimum between the displacement and cooling parameter are added to the position of the node. This is also scaled by the displacement of the node divided by the norm ($||v_i.disp||_2$) of the displacement. The nodes are restricted to lie within the boundary of the box of length L and width W by restricting the size of the x and y coordinates of the nodes.

(Fruchterman and Reingold, 1991)

Figure 5.8 shows the progression of the algorithm as nodes are displaced by the forces in each iteration. The algorithm is oscillating between the layout of iteration 5 and iteration 10 but eventually settles on the layout similar to that of iteration 5. This is because in each

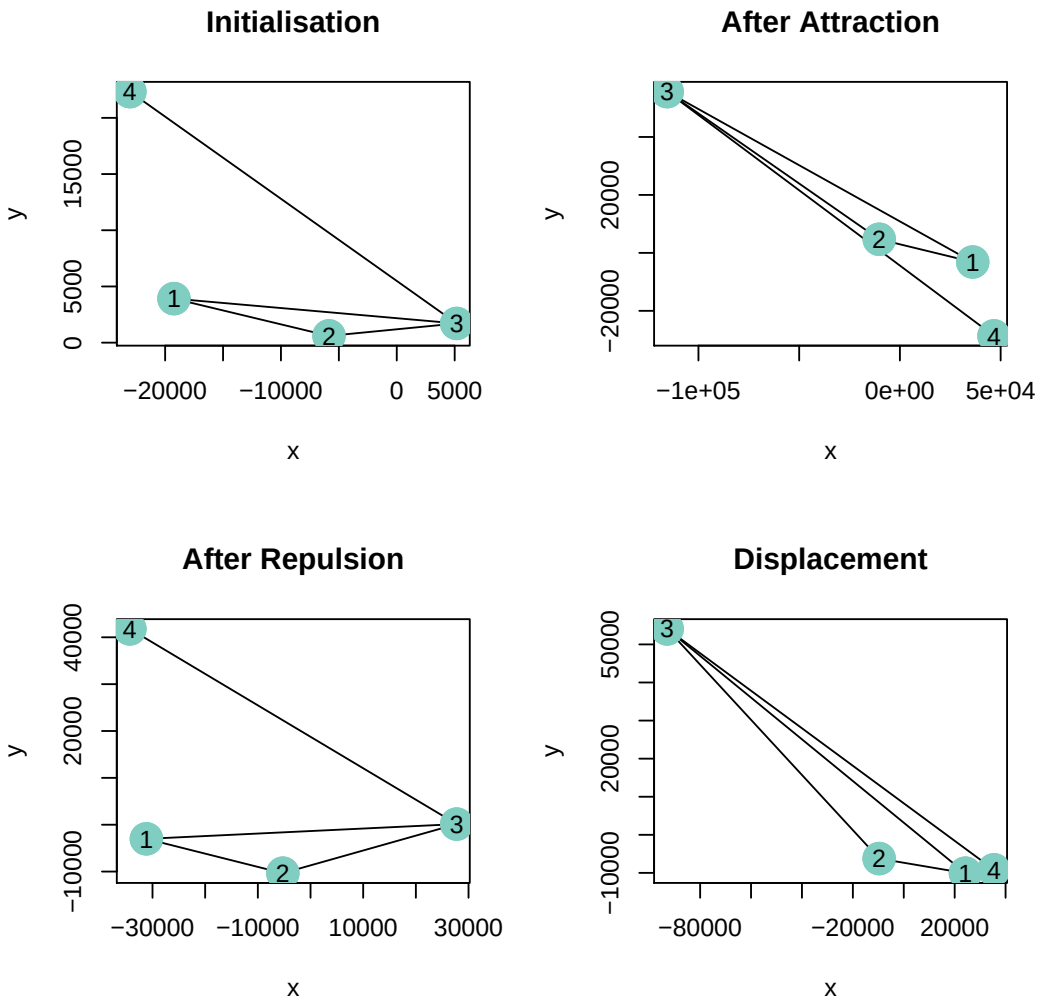

Figure 5.7: Demonstration of Fructherman Riengold attractive and repulsive forces and how they contribute to the displacement of vertices.

Algorithm 3 Update position of nodes

procedure UPDATE POSITION OF NODES
  for i in 1:$|V|$ **do**
    $v_i.pos := v_i.pos + (v_i.disp/||v_i.disp||_2) \times \min(v.disp, t)$
    $v_i.pos.x := \min(W/2, \max(-W/2, v_i.pos.x))$
    $v_i.pos.y := \min(L/2, \max(-L/2, v_i.pos.y))$
  end for
  reduce the temperature as the layout approaches a better configuration
  t := cool(t)
end procedure

iteration either the attractive or repulsive forces are relatively strong. If the attractive forces are strong the nodes move closer together which leads to strong repulsive forces and vice versa. The temperature of the algorithm controls the maximum displacement of nodes in each iteration. Reducing the temperature can help to reduce the effect of the oscillating layout bringing the visualisation to a final state. For this example the final state is the 25th iteration.

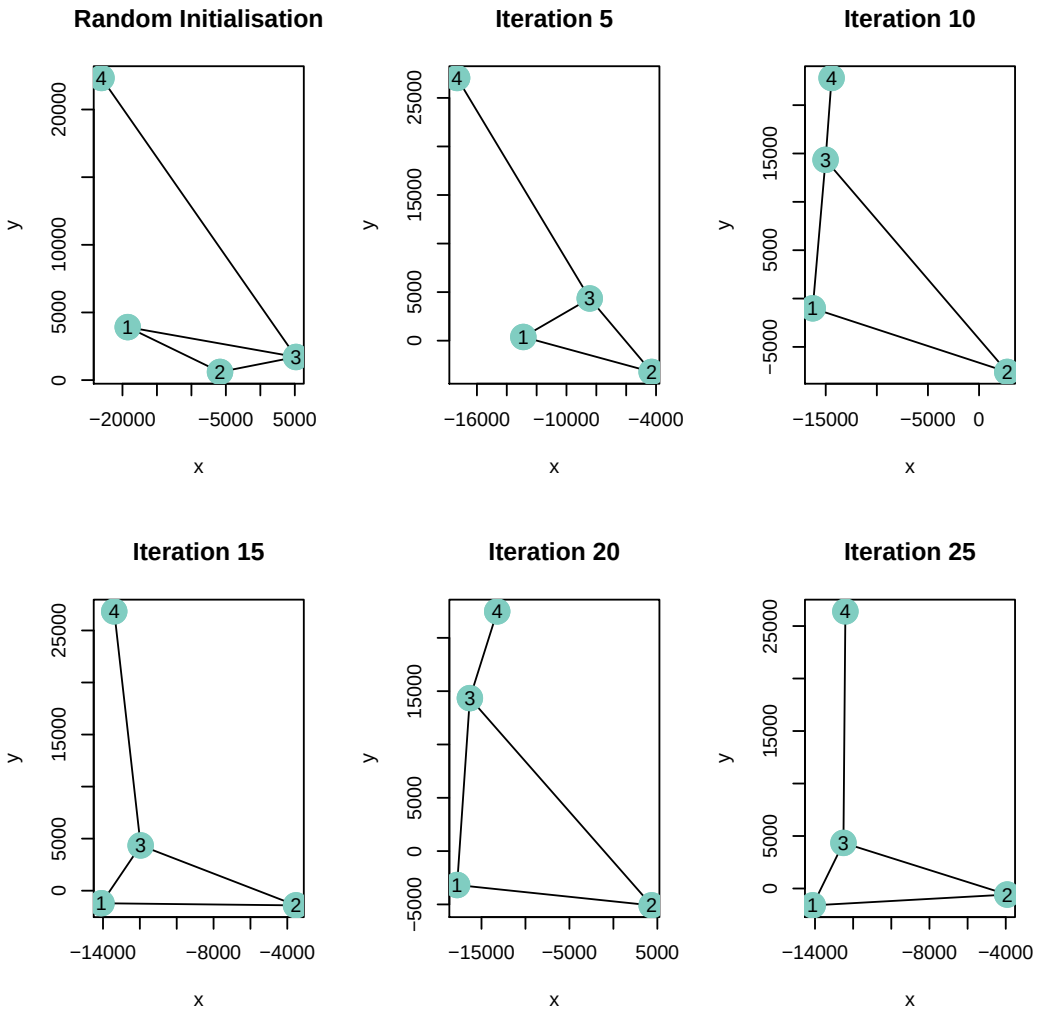

Figure 5.8: Still frames of Fruchterman Reingold algorithm from the random initialisation to equilibrium. The graph with four nodes and four edges tends towards a visualisation where the first three nodes increasingly form an equlateral triangle while the four node is balanced at the end of the triangle.

One example of a force-directed algorithm is ForceAtlas2 (Jacomy et al, 2014), used by

the well-known Gephi network visualisation software. It provides some useful features that enhance the visualisation, including scaling and the prevention of node overlap. Scaling allows for the space on which the graph is drawn to be increased. This was useful for graphs containing many nodes and labels with long names. By increasing the space on which the graph is drawn the distance between nodes is increased and the scale of the nodes is adjusted downwards. The scale of nodes is also adjusted such that nodes are closer in size. This also adjusts the scale and distance of labels making them more readable.

The forces for the ForceAtlas2 algorithm are defined as given below. The attractive force $f_a(v_i, v_j)$ is equal to the euclidean distance between two points $||v_i.pos - v_j.pos||_2$. The attractive force is represented by the product of the degrees with 1 added to each degree $(deg(v_i)+1)(deg(v_j)+1)$ divided by the euclidean distance of the nodes. This is multiplied by a free parameter k_r which the practitioner can set.

$$f_a(v_i, v_j) = ||v_i.pos - v_j.pos||_2$$
$$f_r(v_i, v_j) = k_r \frac{(deg(v_i)+1)(deg(v_j)+1)}{||v_i.pos - v_j.pos||_2}$$

An advantage of ForceAtlas2 is its flexibility. An option it provides is to adjust the layout such that nodes which can have different sizes can be prevented from overlapping. This requires adjustments to the forces described above. First Euclidean the distance is adjusted such that it takes the size of the node into account. If the centres of the nodes are far apart then their Euclidean distance is large. However if the combined size of the nodes is larger than their distance then these nodes are overlapping.

$$d'(v_i, v_j) = ||v_i.pos - v_j.pos||_2 - size(v_i) - size(v_j)$$

The forces are adjusted for three cases. The first when the nodes are overlapping, second when the nodes are not overlapping and finally when the nodes are just touching.

Case 1: $d'(v_i, v_j) > 0$ nodes are not overlapping

The attractive force is equal to the adjusted distance measure which takes the size of the nodes into account. The repulsive force is scaled by the adjusted distance instead of the Euclidean distance.

$$f_a(v_i, v_j) = d'(v_i, v_j)$$
$$f_r(v_i, v_j) = k_r(deg(v_i)+1)(deg(v_j)+1)/d'(v_i, v_j)$$

Case 2: $d'(v_i, v_j) < 0$ nodes are overlapping

The attractive forces are not applied. The repulsive forces are not scaled.

$$f_a(v_i, v_j) = 0$$

$$f_r(v_i, v_j) = k_r(deg(v_i) + 1)(deg(v_j) + 1)$$

Case 3: $d'(v_i, v_j) = 0$, nodes are touching

No forces are applied to the nodes

In this way the ForceAtlas2 algorithm prevents nodes from overlapping.

Chapter 6

TextNet - A software tool for creating and visualising graphs

An online tool was created for constructing graphs from PDF documents and plain text. The tool is called TextNet. TextNet extracts text from PDF documents. Once the document is in text form it performs part-of-speech tagging. It provides the option to filter the text by proper nouns or return the entire tagged text. TextNet then constructs a graph and visualises it. The visualisation method used is Force Atlas 2. The Stanford English Part-of-speech Tagger an implementation of the model by Toutanova, Manning, Klein and Singer (2003) is used for part-of-speech tagging. Sigma.js is a JavaScript library for graph visualisation, it is used for visualising the graphs. The tool is constructed using PHP which allows it to be easily hosted online for public use.

TextNet allows for filtering by part-of-speech. It also allows for filtering by node degree. The graphs can be moved around within the visualisation. It allows for zooming in and out by scrolling the mouse wheel up and down. It highlights neighbourhoods when nodes are clicked. Finally it allows rotation of the graph when two fingers are used on a touch display.

6.1 Description of interface

In this section the user interface of TextNet is described. It covers the two main pages, the home page (Section 6.1.1), the visualisation page (Section 6.1.2), some limitations of TextNet (Section 6.1.3) and finally some practical considerations for reconciling different words that can take on different parts of speech (Section 6.1.4).

6.1.1 Home page

This brief section describes the TextNet home page which starts with an introduction followed by a number of fields where users can input their preferences.

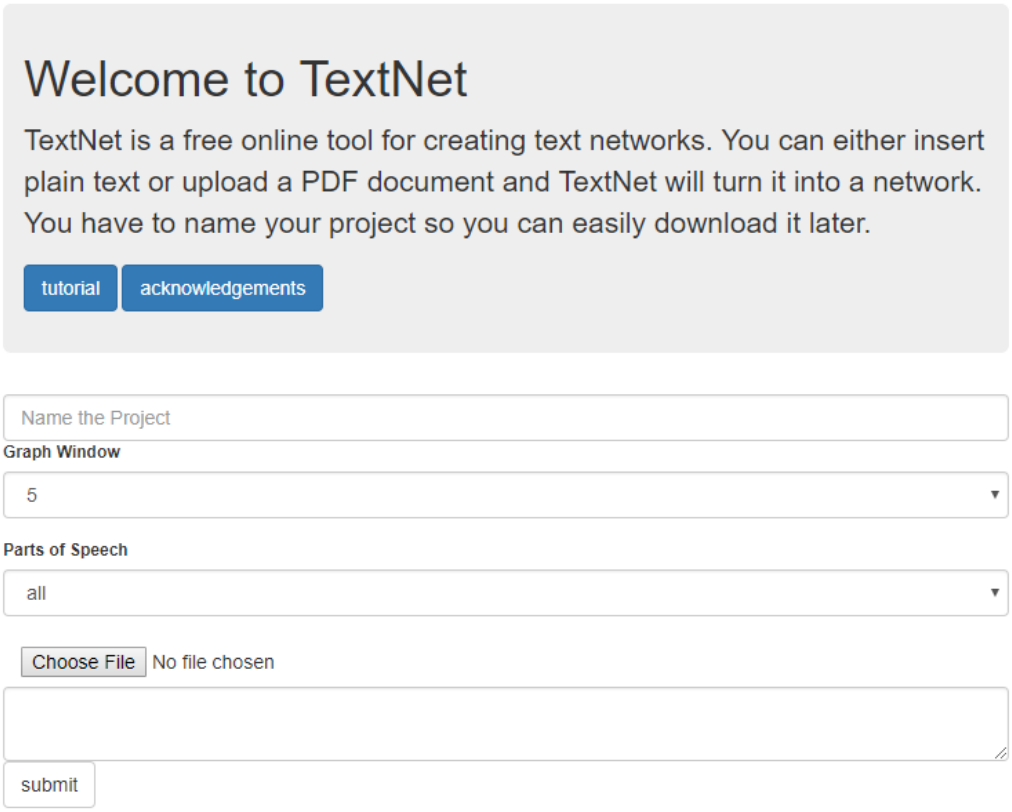

Figure 6.1: TextNet home page. Users can enter the name of the project in the first field. The second field is a drop down menu with co-occurrence windows from 5 to 100 increasing by 5. The third field allows users to select between a graph that has all the parts-of-speech or a graph with proper nouns only. Finally users can either upload a document or paste text into the text field.

The TextNet home page has five input fields.

1. The first is the Project Name field. Users can name their project using this field. This creates a web address associated with their project.

2. The second part is the graph window which allows them to choose between 5 and 100 in increments of five.
3. The third part is the part-of-speech selector. This allows users to choose whether to include all parts of speech or just the proper nouns.
4. The fourth part is the file upload for PDF documents. Users can select any PDF document and upload it using this feature. Not all PDF documents are supported by the PDF parser which is created by PDFParser.org. Users are advised to use an alternative PDF converter if this should fail or take too long.
5. The large text field is for plain text documents. Users can copy the text of a Wikipedia Page, website or plain text document and paste it into this field.
6. Finally users can submit the form and wait to be taken to the next page

The homepage also provides a link to the acknowledgements page which lists the various software libraries used in creating TextNet. Owing to some difficulties with the GEXF file format and PDF conversion libraries which lead to errors or failure to display graphs, it has become necessary to add a link to a tutorial on the home page. The tutorial takes a PLOS One journal article and shows how to overcome some common issues to eventually produce a visualisation. There are six visualisations provided, three full graphs and three proper noun graphs as described in Section 6.2.1.

6.1.2 Visualisation page

The visualisation page of TextNet has various aspects. This section describes them in turn using visual aids. The descriptions follow the sequence of how a practitioner might use TextNet. First the graph is initially visualised. The practitioner can then use the zooming functionality, the minimum degree filter, the part of speech filter and finally view the node information panel. All of these aspects provide functionality and flexibility for the practitioner to analyse a text graph.

When users arrive at the visualisation page they will be confronted by a large colourful graph. The colours represent the top 17 parts of speech by number of words. The remaining parts of speech will have the same colour as the 18th largest part-of-speech. The edges are coloured based on the nodes they emanate from.

Users can zoom in and out of the visualisation by scrolling up or down with the mousewheel. Depending on the configuration of the computer this will either increase or decrease the scale of the graph as demonstrated in Figure 6.2 as compared to Figure 6.3.

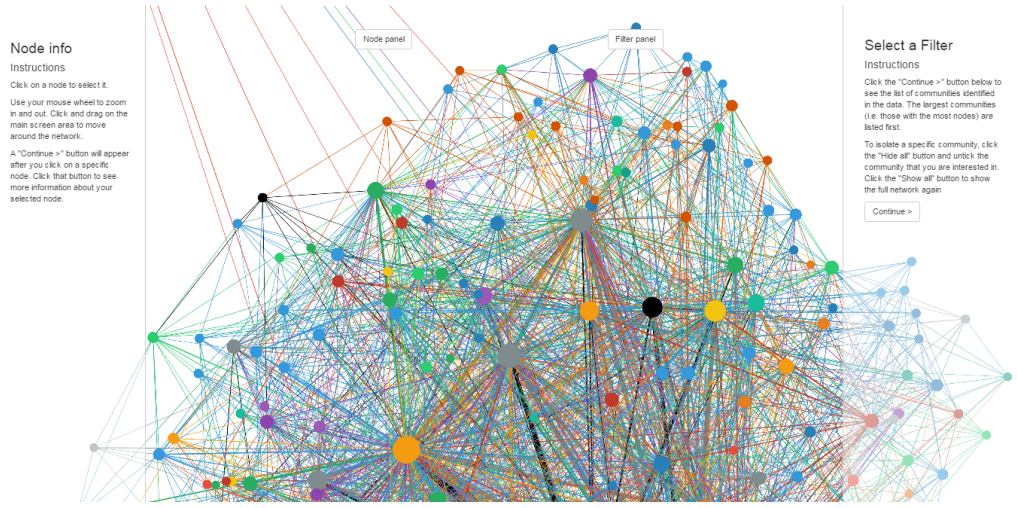

Figure 6.2: TextNet visualisation page - full graph. This is the page that a user is directed to after configuring the graph on the home page. It starts with a random initialisation and applies a layout algorithm.

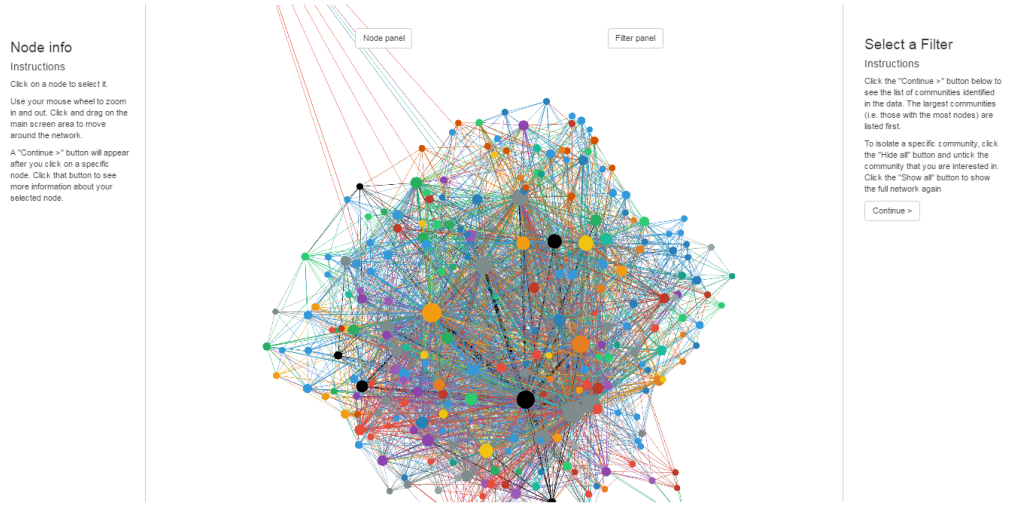

Figure 6.3: TextNet visualisation page - zoom. Users can zoom in and out of the graph using the mouse wheel or the track pad when directly hovering over the graph.

The right panel allows users to filter by degree or parts of speech. The degree filter provides a range from zero to the highest node degree of the graph. As the minimum degree is increased, the graph is filtered such that only the nodes with higher degree remain in the visualisation. In Figure 6.4 above the degree has been set to 18 which has reduced the number of remaining nodes in the visualisation.

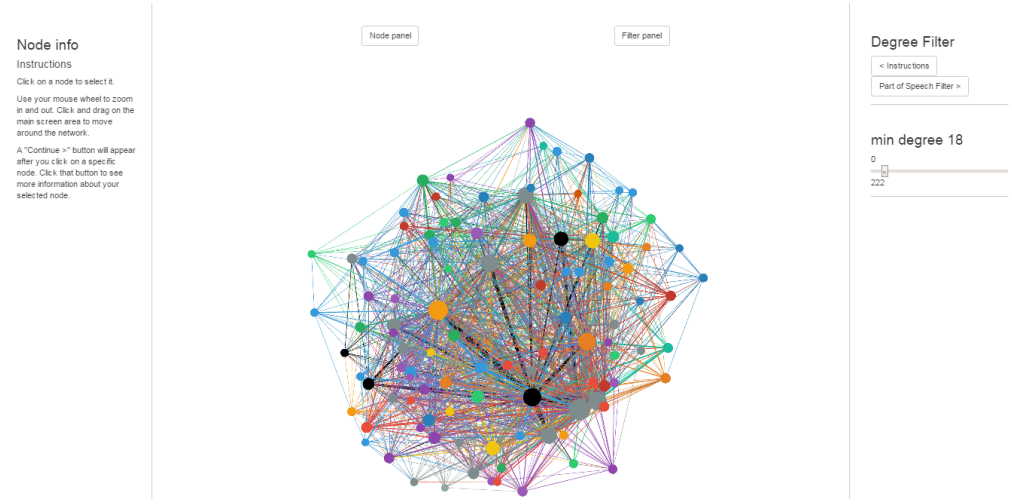

Figure 6.4: TextNet visualisation page - degree filter. There is a filter to the right of the graph which hides nodes with a degree less than that indicated by the position of the filter.

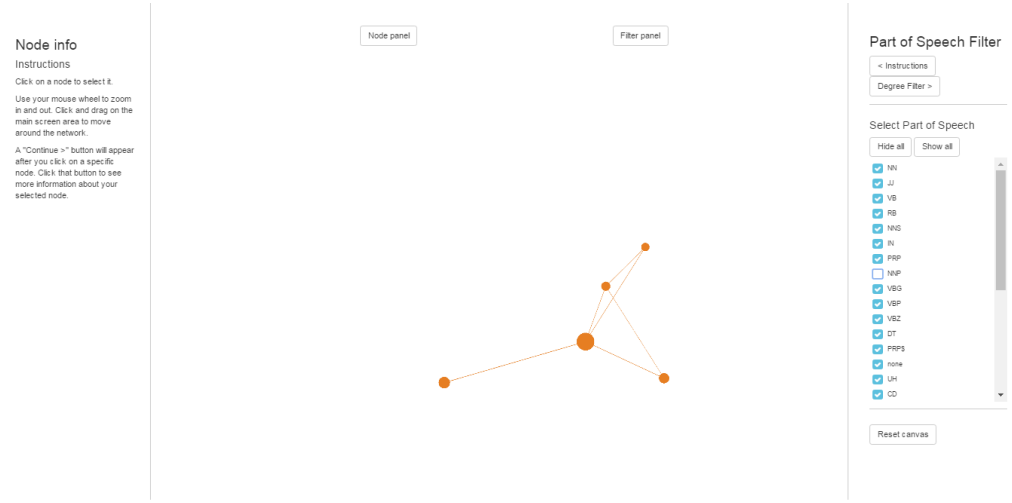

Figure 6.5: TextNet visualisation page - part-of-speech filter. There is a part of speech filter to the right of the graph. When a part-of-speech is selected it is hidden. In order to filter to a specific part-of-speech the 'hide all' button can be used to select all part-of-speech and that part-part-speech which is of interest can be deselected.

When a part-of-speech is selected the nodes within that part-of-speech disappear. In order to filter by a single part-of-speech a user can select all the remaining parts of speech. This can be tedious when there are many parts of speech. The *Hide all* button selects all parts of speech and allows users to deselect the part-of-speech they would like to view. In Figure

6.5, NNP which represents proper nouns has been deselected to isolate the proper nouns.

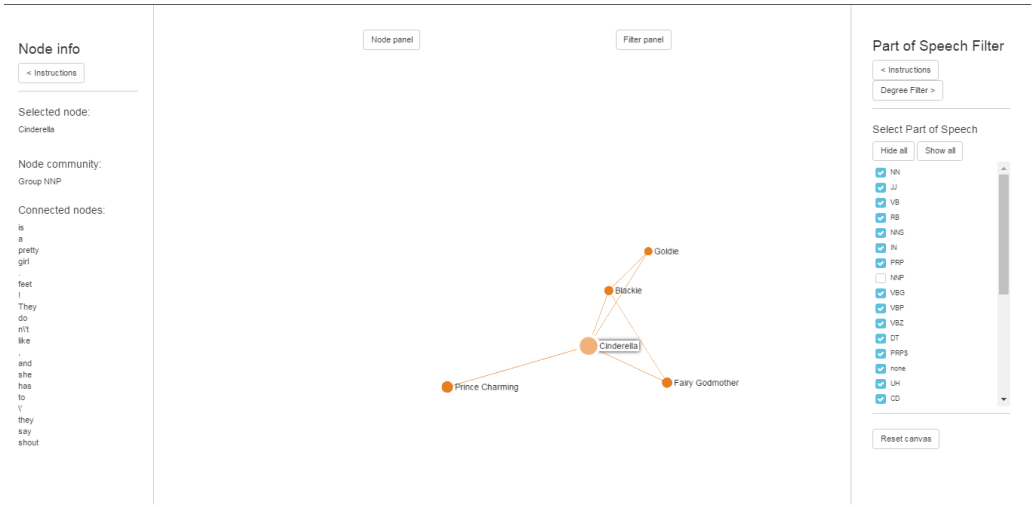

Figure 6.6: TextNet visualisation page - node info. There is a pane to the left of the graph which shows the information relating to a specific node when it is clicked. this information includes the name of the node, its part-of-speech and some neighbours.

The left panel provides additional information regarding nodes. It provides the name of the node, its part-of-speech and the top nodes associated with it. This information is updated when a node is clicked on. The other nodes and edges become grey in order to highlights the selected node and its immediate neighbours.

6.1.3 Limitations

TextNet was used for creating graphs in the illustrative examples below. The software is not capable of performing modularity clustering. The Gephi Software package was used for clustering and filtering the graph. TextNet also uses libraries to extract text from PDF documents and display interactive graphs (Sigmajs.org). These libraries have their own limitations which will change over time. These limitations are best tracked through the respective websites. TextNet itself has a limitation regarding the GEXF file format which is used to display graphs. Special characters with accents like é, ë and various others are not supported by TextNet which leads improperly formatted GEXF files.

6.1.4 Post-processing

Words can often take on different parts of speech. Yet nodes in a graph are unique, and therefore nodes with different parts of speech need to be reconciled. In our implementation, parts of speech are assigned based on the final instance of a word. This was done purely for convenience. While in theory this can lead to words having unexpected or less dominant parts of speech, we found this to have little practical consequence.

In addition various nodes might refer to the same entity. For instance *Smith* and *Mr Smith* might both be proper nouns yet they both refer to *John Smith*. These are important considerations for the presentation of a text network in a formal setting. For the purposes of preliminary text visualisation however, they serve to highlight characteristics of the data that might be exploited for classification or other purposes.

6.2 Illustrative examples

In this section we illustrate the use of TextNet using examples drawn from a variety of text documents. In all cases the visualization approach starts with a text document, classifies the parts of speech in the document, converts the words in the network into a graph, filters the graph by part-of-speech, and finally clusters nodes in the graph i.e. words, using modularity-based clustering.

6.2.1 Filtering Fantastic Mr Fox

Fantastic Mr. Fox is a children's novel written by Roald Dahl and first published in 1970. Mr. Fox lives underground with his wife and four children. He has three neighbours who are farmers: Boggis, Bunce and Bean. Mr. Fox regularly outsmarts them in order to steal livestock from each farmer. In frustration, the farmers plot to catch Mr. Fox. The farmers dig up the Fox's home but the Fox's respond by digging deeper. The farmers decide to stage a stake-out at the entrance to the Fox's home. The Fox's along with the other underground creatures are starved as they cannot escape to look for food. This prompts Mr. Fox to embark on an ambitious mission to find food for his family and friends. He manages to dig another exit and steal from each farmer in turn. The story concludes with the farmers still stalking the foxhole while the Fox's and their friends create a secret underground community. (Wikipedia.org/wiki/Fantastic_Mr_Fox, 2017)

The children's book Fantastic Mr Fox was used to demonstrate the effectiveness of proper noun filtering. Words within 60 words of each other were joined by edges. When the same

edge was created multiple times its size or strength was increased. The window size could easily cover a short paragraph. This connects words that might have a tenuous connection in the story. Nevertheless, the edge weights compensate for this by allowing the thickness of edges between highly connected words to be larger. This keeps the relationship between highly connected words clear. When window sizes are small more disconnected components will appear. Force directed algorithms push disconnected components apart in visual space. In the resulting image (Figure 6.8) the distance between disconnected graphs is scaled relative to the node and edge sizes which makes them smaller and harder to see. This is why it sometimes preferable to use larger window sizes which produce a single connected graph.

The original graph shown in Figure 6.7 was large and difficult to use. Each word is associated with a part-of-speech. Each part-of-speech is associated with a colour. Together they form a convoluted image which is difficult to extract meaning from.

The graph in Figure 6.8 was filtered to only show the proper nouns. Nodes with degree lower than 189 were removed. The resulting image shows the main characters and some venues. The node sizes convey the importance of specific proper nouns. The edge weights convey the strength of relationships between those proper nouns with thicker edges having higher edge weights. Nodes are close together when they share many edges, in this way groups are formed by proximity in space. The edge weight also contributes to proximity, if the weight is higher the nodes are closer. All of these attributes convey information with an intuitive interpretation. Little expertise is required to interpret the image.

6.2.2 Ego networks for Cinderella with descriptive parts of speech

Cinderella by Ruth Hobart is a version of the famous story of a young women living with her sisters. Cinderella sleeps in the attic and does much of the house work at the behest of her sisters. Her pets are her friends, including several mice, a cat named Blackie and a goldfish named Goldie. Her sisters receive an invitation for a ball at the palace. Cinderella is interested in attending but her sisters discourage her. On the day of the ball Cinderella's Fairy Godmother arrives to help her attend the ball. She transforms a pumpkin into a coach, the mice into horses, Goldie into a footman and Blackie into a coachman. Cinderella attends the ball and dances with Prince Charming. The Fairy Godmother's spell wears off at midnight and Cinderella has to leave in a rush when she realises the hour was fast approaching midnight. She leaves her shoe behind and the prince uses it to find her. They live happily ever after.

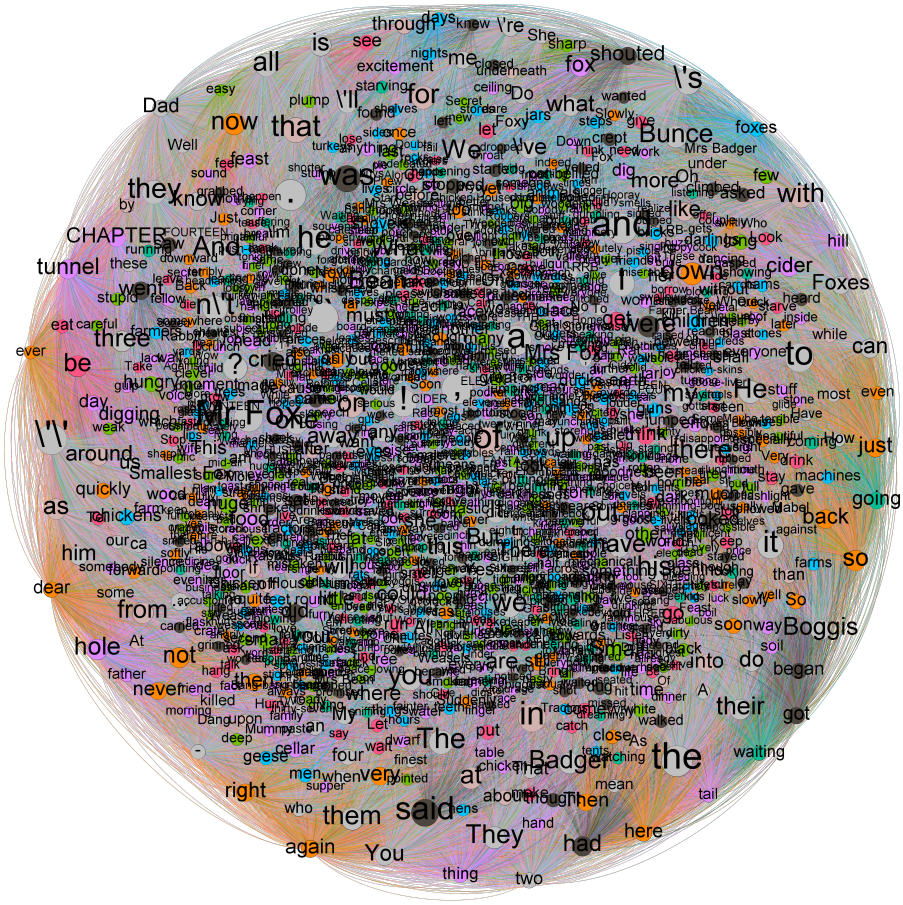

Figure 6.7: Full Text graph for Fantastic Mr. Fox. This graph represents the entire text of Fantastic Mr. Fox, all the words and parts of speech.

An ego network is a network filtered on a specific node. Only the neighbours of that node are included. In this sense it can also be considered the neighbourhood of a specific node. The Cinderella story was used to demonstrate the usefulness of ego networks and combinations of parts of speech. A graph was created by connecting words within a distance of five words from each other. The original graph was filtered to include proper nouns, nouns, adjectives and verbs.

What motivated this collection of parts of speech is their important role in the English language of conveying information. The other parts of speech are mostly concerned with

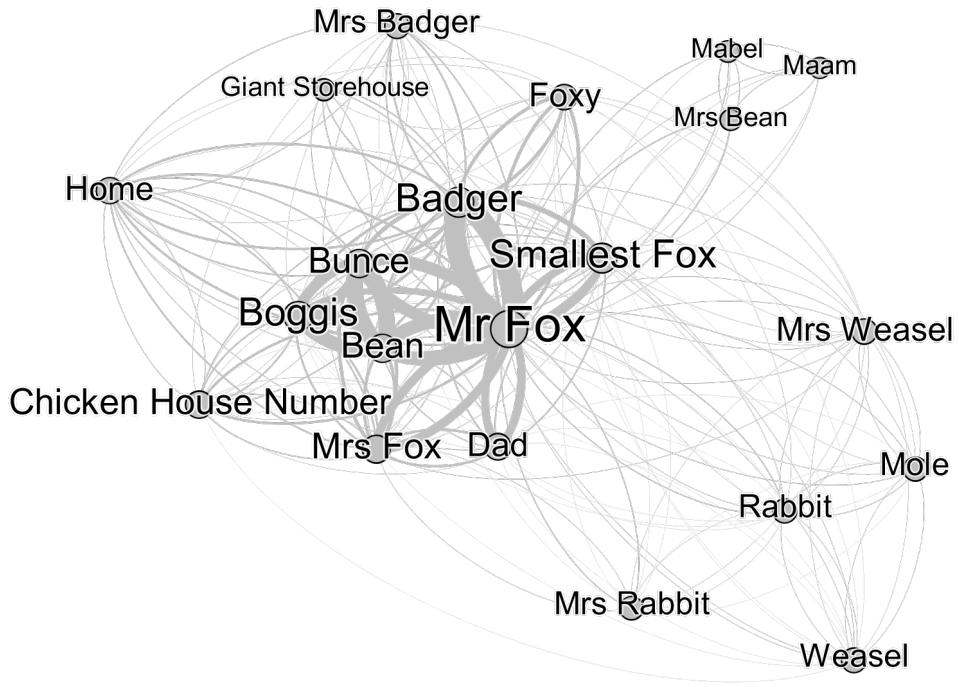

Figure 6.8: Proper noun graph for Fantastic Mr Fox. The other parts-of-speech have been removed and only the proper nouns remain. This graph shows how they are connected with thick edges indicating stronger connections.

facilitating the collaboration of these parts of speech.

Originally, the Cinderella network was built with a window size of 60 words. This produced a highly connected graph. The ego network for the Cinderella node was connected to many of the high degree adjectives and verbs. This reduced the value of isolating it for richer insights. Subsequently the window size was reduced down to five. This created a more sparsely connected graph. Ego networks for Cinderella, Blackie and Goldie were produced to further demonstrate the effectiveness of ego networks.

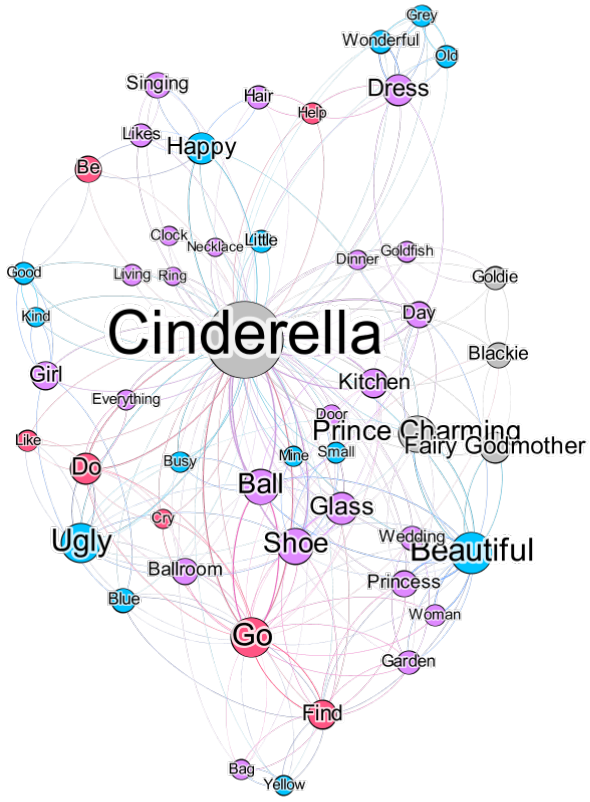

Figure 6.9: Cinderella ego network with nouns, proper nouns, adjectives and verbs. Colours convey parts-of-speech. Verbs in red, Nouns in purple, adjectives in blue and proper nouns in grey.

The large nodes in the graph (Figure 6.9) are other characters like *Prince Charming* and the *Fairy Godmother*. Nouns like, *ball*, *shoe*, *glass* and *dress*; adjectives like *happy*, *ugly* and *beautiful*; verbs like *go*, *do*, *help* and *find* all provide some context around the story. The network around the main character has much information but relatively little context. The ego networks for her goldfish and cat below show the effectiveness of an ego network when applied in a more limited context.

The ego networks also proved to be useful for smaller characters. Figure 6.10 conveyed

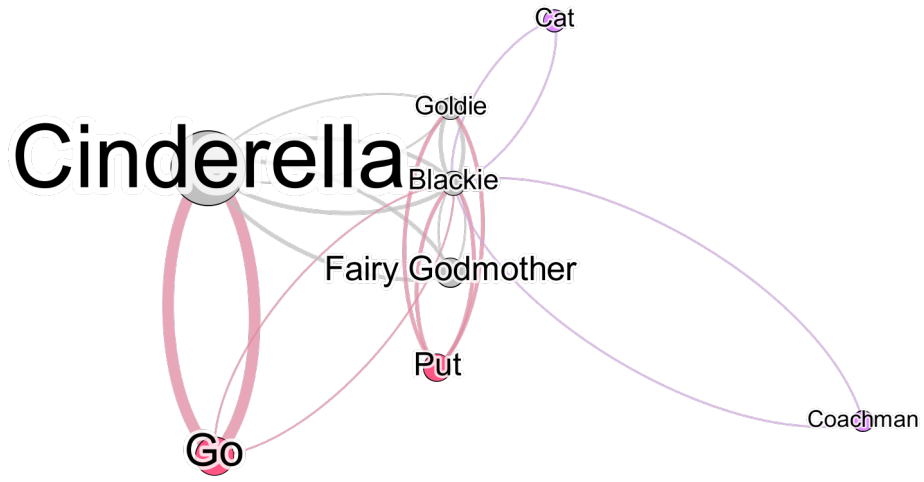

Figure 6.10: Blackie ego network with nouns, proper nouns and verbs. There are no adjectives connected to Blackie.

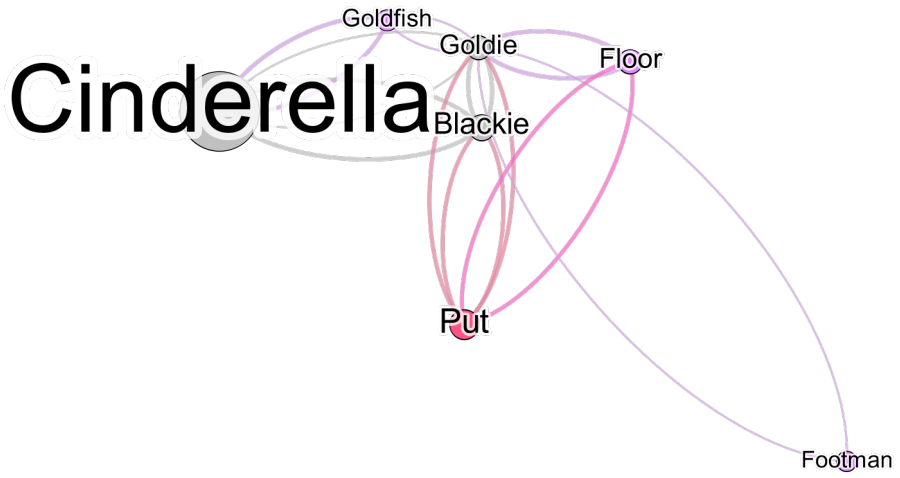

Figure 6.11: Goldie ego network with nouns, proper nouns and verbs. There are no adjectives connected to Goldie.

that *Blackie* was connected to both *cat* and *coachman*. Similarly, Figure 6.11 conveyed that *Goldie* was connected to both *goldfish* and *footman*. Both creatures were converted

to their respective human roles to allow Cinderella to attend the ball.

6.2.3 Filtering and clustering one large document

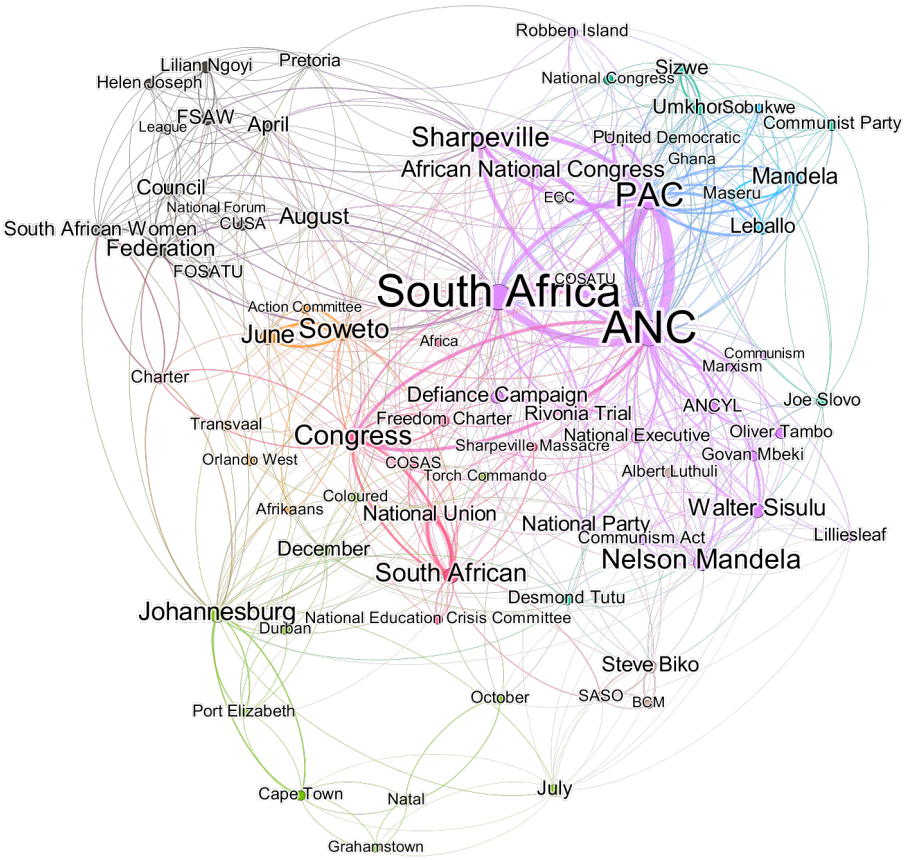

Figure 6.12: Internal resistance to Apartheid Wikipedia page clustered proper noun graph. Each cluster is coloured.

The Wikipedia page for the Internal Resistance to Apartheid was converted to a text graph as shown in Figure 6.12. The resulting graph was large and unwieldy. The nodes were filtered to proper nouns. This reduced the graph to the names of individuals, organisations, events, movements, locations and dates. The final proper noun graph was still large and

difficult to analyse. The graph was then clustered using modularity clustering. This produced a number of clusters which were each coloured. The colours indicate the relative size and proximity of clusters. The orange cluster is adjacent to the red cluster and not adjacent to the blue cluster. In a force directed layout space conveys meaning as such inferences can be draw regarding the proximity of clusters by observing their adjacency. The purple cluster has more nodes than the orange cluster, this indicates the purple cluster is larger than the orange cluster. The main visualisation excluded smaller nodes as they would be difficult to see. This is easier to visualise but some information is lost. There are two clusters that were investigated individually with fewer nodes excluded. A good cluster should group nodes that are related to each other in addition to conveying the information expected form a graph visualisation. Node sizes, edge thickness and proximity in space should all convey information consistent with the facts contained in the text.

6.2.3.1 Political Entities of the Apartheid Era

The purple cluster in Figure 6.13 contained important names, themes, ideas and individuals who were important to the liberation struggle in South Africa. They also related to two important events, the Rivonia Trials and the Sharpeville Massacre.

The thickest edge weights existed between the *ANC*, *PAC* and *South Africa*. The *PAC* was strongly connected to *Poqo* their armed wing and *Sharpeville*, the scene of a massacre. The *ANC* was connected to its leaders *Walter Sisulu*, *Nelson Mandela* and *Goven Mbeki*. It was also connected to *Sharpeville*, the *ANCYL* (its youth chapter) along with concepts like *Marxism* and *communism*.

6.2.3.2 Soweto Uprising

When the orange cluster (Figure 6.14) was isolated and smaller nodes included it became clear that this cluster contained nodes related to the Soweto Uprising. Shortly after the Afrikaans medium decree, several schools around Orlando West in Soweto organised protests. The protests turned bloody when the South African Defense Force opened fire on the student protesters.

6.2.4 Clustering multiple small documents

Yelp is a reviews and discovery website for businesses. It allows users to make reservations and order food through its subsidiary Eat24. Yelp supports businesses in responding to and dealing with reviews that they receive on the service.

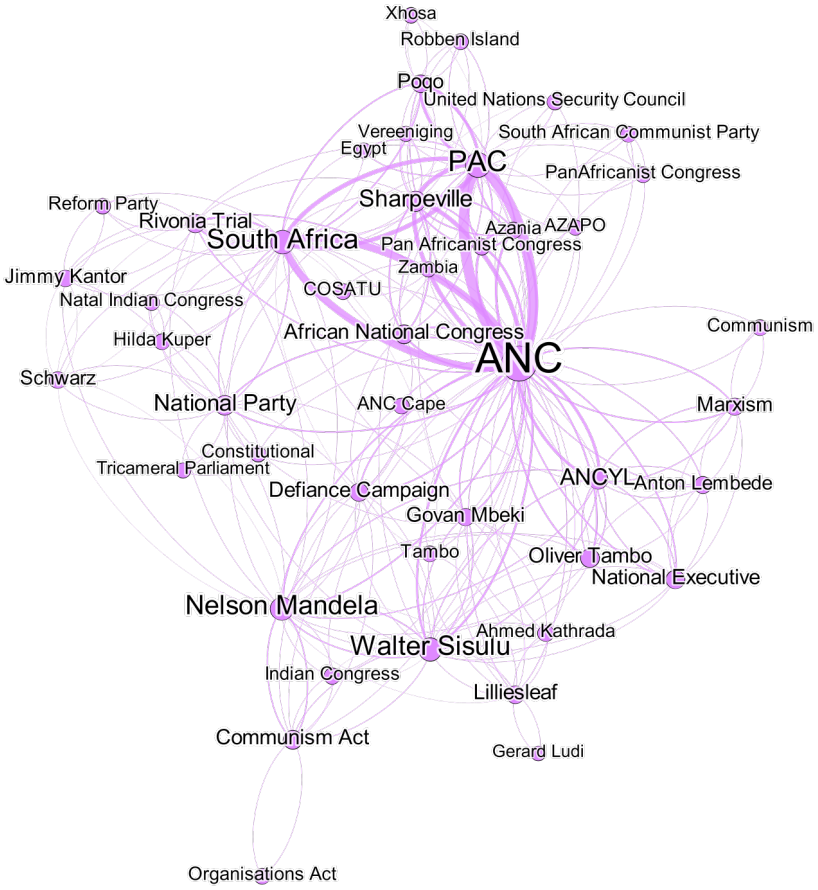

Figure 6.13: Internal resistance to Apartheid - ANC cluster graph. This is a subgraph of the within the Internal Resistance to Apartheid graph which only shows nodes automatically clustered together in what is referred to as the ANC cluster.

Yelp recently started releasing some of their data for academic purposes. The data contained several files relating to reviews, tips, users and businesses among other data. Yelp reviews can be analysed for insights regarding user sentiment towards specific brands and locations. The reviews are generally short documents. We combined 1000 reviews into a single document, and analyzed the resulting document using TextNet. The last paragraph of the previous review would be followed by the first paragraph of its subsequent review.

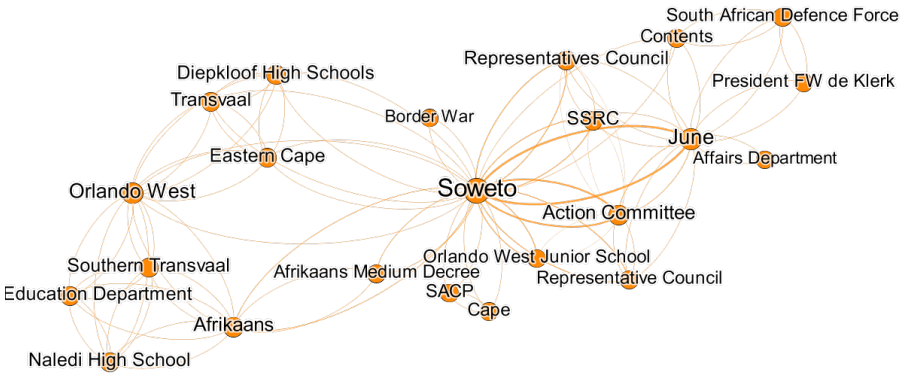

Figure 6.14: Internal resistance to Apartheid - movement cluster graph. This is a subgraph of the within the Internal Resistance to Apartheid graph which only shows nodes automatically clustered together in what is referred to as the Movement cluster. This cluster seems to relate to specific aspects of the anti-Apartheid movement in a general sense.

The final combined document was converted into a graph and visualised. Words that were adjacent were connected. The resulting visualisation was clustered using modularity clustering. The clusters seemed to extract themes from the data. Two clusters are presented to illustrate this.

The first cluster (Figure 6.15) shows ingredients. *Chicken*, *cheese*, *got* and *ordered* are prominent nodes. Chicken is connected to nodes like *fried*, *pizza* and *salad*. *Bacon*, *eggs* and *potatoes* are closely connected with thick edges. Almost all the nodes are either an ingredient one might expect on a menu or are directly related to an ingredient.

The second cluster (Figure 6.16) shows words referring to the overall experience at a location. The prominent nodes are *really*, *nice*, *staff*, *friendly*, *didn't* and *us*. The thick edges exist between word pairs like:

- *really* and *enjoy*, *really* and *nice*
- *nice* and *staff*, *friendly* and *staff*, *friendly* and *waitress*, *friendly* and *server*.

Most of the words refer to the relationship between the customers and the staff. Overall, the relationship seems to be positive. Yelp reviews convey user defined star ratings. This

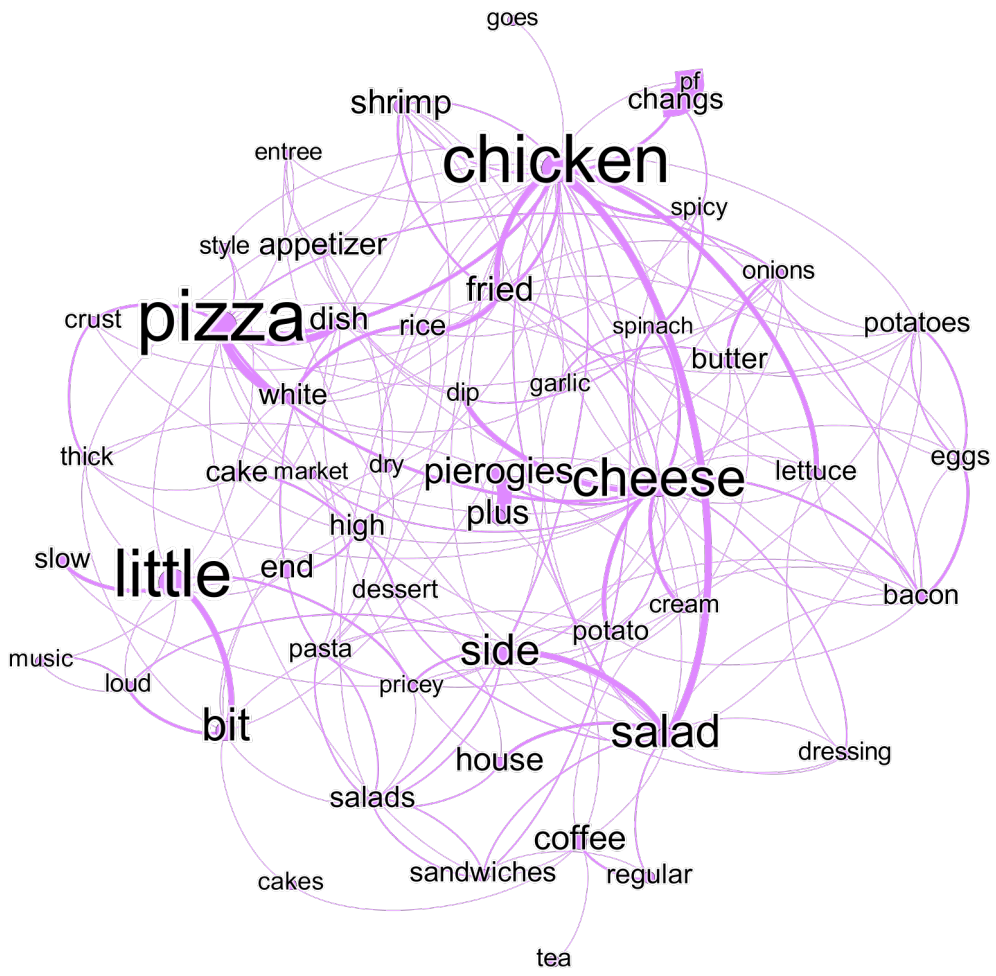

Figure 6.15: Yelp ingredients cluster. The Yelp dataset was clustered and this cluster prominently includes ingredients.

makes them a suitable dataset for sentiment analysis. There are interesting cases where word pairs that carry a certain sentiment can be part of word triplets that carry a different or heightened sentiment. Thick edges exist between *bad* and *experience*. Thick edges exist between *wasn't* and *bad* and *really* and *bad*. This creates two different alternatives in terms of sentiment. The first *wasn't bad experience* and the second *really bad experience*. This visualisation has important implications for sentiment analysis. It might be possible to use the modularity method to cluster text data and isolate groups of words that carry sentiment information.

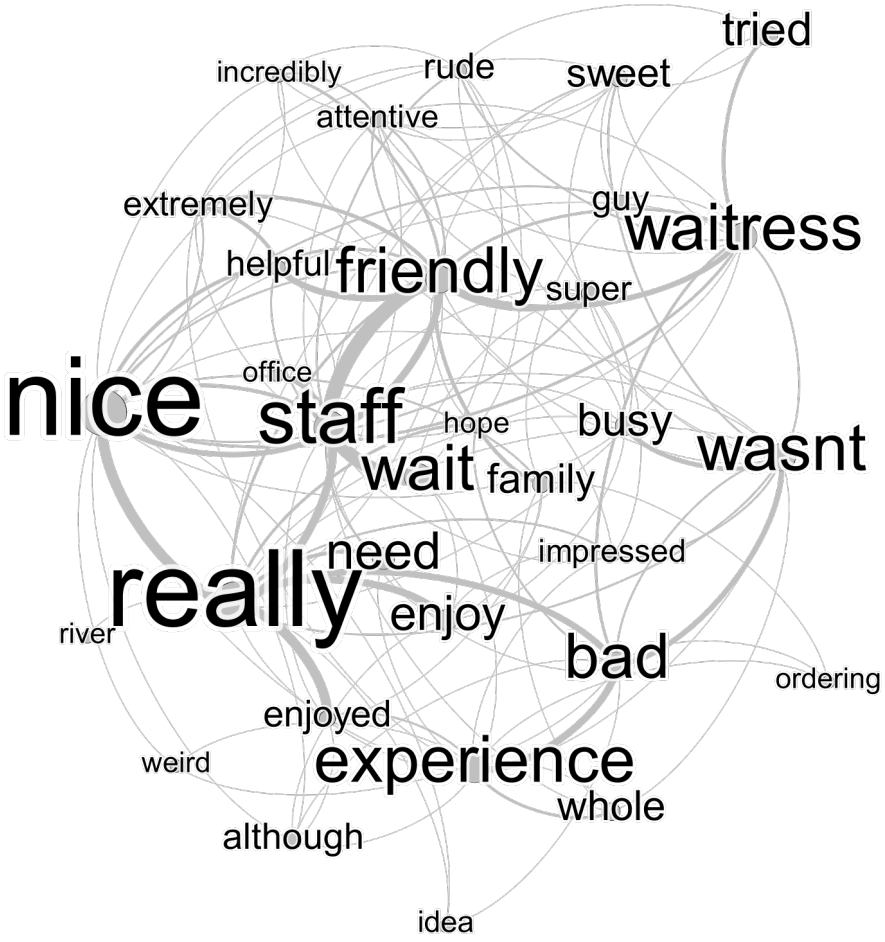

Figure 6.16: Yelp - experience cluster. After the Yelp dataset was clustered, this cluster mostly seemed to capture elements of the customer experience.

6.3 Text Visualisation Conclusions

This section has two subsection, first some remarks to conclude the previous two sections (Section 6.3.1). This is followed by some thoughts on future work (Section 6.3.2).

6.3.1 Concluding remarks

A framework for text visualisation has been presented. The framework converts text documents or collections to graphs. The resulting visualisation can often become large and therefore difficult to use. Part-of-speech tags and Modularity clustering or some combination of the two methods can be used to overcome this.

In general graph visualisations have other useful properties. Colour is used to convey information about clusters or attributes. Edge weights are able to visually demonstrate the strength of relationships between node pairs.

Filtering and clustering are useful for text visualisation. Clusters seem to extract themes. This can happen even in separated documents which are combined after the fact, like in the case of the Yelp reviews dataset (see section 6.2.4). This happens whether or not parts of speech have been filtered, as was shown for the Internal Resistance to Apartheid Wikipedia page (see section 6.2.3). Filters can be applied to attributes like parts of speech, nodes like the ego network, cluster membership or some combination of the above. Each filter reduces the amount of information being reviewed which makes the visualisation easier to navigate. The framework is a powerful means of visualising text data across different sizes of documents. Free software for creating text graphs is available at textnet.co.za.

6.3.2 Further work

In a graph, nodes should be unique. Therefore a limitation of the framework is multiple nodes that refer to the same entity. This appeared in the *Fantastic Mr Fox* example. The automatic combination of nodes that refer to the same entity could be a useful addition to the framework. This would require disambiguating the context of specific words. When the words associated with an entity are *Farmer Boggis* and *Boggis* then it is clear these nodes should be connected. However when the words are *Trump*, *The Donald* and *Mr President* then the task becomes more complex.

Word Sense Disambiguation is the task of labeling words in a sequence with the correct meaning (Mihalcea, 2005). A word like *bank* might have meanings like *edge of river* or *financial institution*. The task is to use the context within which the word *bank* arises to assign the correct meaning. The task of combining nodes that refer to the same entity is similar. Word Sense Disambiguation determines the correct meaning of a word. The task suggested above determines the meaning of words and combines them into a single node if they are the same. In both cases, context is important. Graph theoretic approaches have been applied to Word Sense Disambiguation by Mihalcea (2005) and Sinha & Mihalcea

(2007). It might be possible to advance these methods for the automatic combination of nodes with similar meanings.

Chapter 7

Conclusions

The purpose of this dissertation was to assess the effectiveness of graph/network theory when applied to text classification and text visualisation. Text classification is concerned with assigning a single or multiple labels to a text document. Text visualisation is concerned with creating visual representations of text documents or the analysis of text documents. The conclusions for each area are presented in turn

7.1 Conclusions - text classification

There were two text classification tasks. The first tasks assigned a single label to each document using the NSF dataset. The second task assigned at least one labels to each document of the Reuters dataset. These tasks were aimed at assessing the effectiveness of graph/network weighting schemes and features. The weighting schemes were based on summary statistics of graphs like Total Degree, In-Degree, Out-Degree and Clustering Coefficient. These weighting schemes were applied to each word which was represented by a node in a word-word graph. The edges of the graphs were created when a pair of words appeared within a specified window of each other. Several windows were provided from 1, 2, 5, 10, 20 and 35. The edges were weighted and so each time an edge was created its weight was added to the existing edge weight. There were two methods suggested for creating edge weights co-occurrence and distance. The co-occurrence edge weight was always one. As such the total edge weight for a pair of words was the number of times the pair of words appeared within the specified window of each other. The distance edge weight was the reciprocal of the number of words that separated the pair of words (one/number of separating words). In addition there were some features added to the

document model, these included the diameter and clustering coefficient of the word-word document graph among others.

The purpose of the experiments was to apply three machine learning algorithms to the two text classification tasks over the various graph/network specifications to determine which would positively affect test accuracy. These graph/network specifications included the weighting scheme, of which there were four, the window size, of which there were six and the edge weighting of which there were two. An addition consideration was the use of inverse-document-frequency for the graph weighting schemes. The graph features themselves were also of interest. All of this was compared to the traditional approach of using term frequency with the traditional term-document-matrix document model.

The results were presented for main effects and interaction effects. The measure of superiority was test accuracy on the text classification task. The results are presented here together as the interaction effects provide clarity regarding the main effects.

- Total degree out performed term frequency on the basis of test accuracy when averaged over all the specifications. It was the only graph weighting scheme to do so. Total degree was superior to test accuracy specifically for naïve Bayes and support vector machines. Term frequency was slightly better on neural networks. Total degree was nearly 10% better than term frequency for naïve Bayes.
- Support vector machines were the best machine learning method followed by neural networks and naïve Bayes.
- There was no difference between distance and co-occurrence edge weighting.
- The window sizes were largely similar but window size five had higher test accuracy than the other window sizes. The accuracy increased from one to five and then decreased consistently after five to 35. Total degree had higher test accuracy for 4 of the 5 window sizes than term frequency. In addition term frequency had decreasing test accuracy with respect to window size.
- The use of inverse document frequency increased test accuracy even for graph/network based weighting schemes.
- The inclusion of graph/network features to the document model generally decreased test accuracy. When the document model with graph/network features had the total degree weighting scheme applied to it, test accuracy increased. In addition when support vector machines were applied to the document model with graph/network features test accuracy increased.

The use of graph/network based weighting schemes does have a positive affect on test

accuracy. This is clear as total degree is generally superior to term frequency in the experiments presented. In addition the inclusion of graph/network features to the document model can have positive affects on test accuracy when complemented with either the total degree weighting scheme or support vector machine algorithm.

7.2 Conclusions - text visualisation

A text visualisation framework was created based on text graphs. The text graphs were visualised using a force directed algorithm called ForceAtlas2. The force directed algorithm grouped nodes in space if they shared many edges. The size of nodes was based on the number of edges leaving or entering a node. The edges were created when a pair of words were separated by less than a pre-specified distance. Edges were made thicker when their edge weight was larger. Colour was used to indicate group membership either to a modularity cluster or part-of-speech group. The framework was used in four settings to demonstrate the usefulness of the framework as an exploratory tool. Two short stories, a long Wikipedia page and the first 1000 reviews of from the Yelp.com academic dataset were used.

- A children's book, Fantastic Mr. Fox was used to demonstrate the effectiveness of filtering by proper nouns. The proper noun filter revealed many of the main characters and the edges revealed the strength of their relationships. Closely related characters had large edges and unrelated characters had small or no edges.
- The story of Cinderella demonstrated the usefulness of filtering by multiple parts of speech and filtering by a neighbourhood of a node, known in the network theory literature as an ego network. Three graphs were used for three of the characters in the story. Each graph was able to convey information regarding that particular character and other characters they interacted with.
- A Wikipedia page on the Internal Resistance to Apartheid helped to show how large documents could be analysed. First the document was filtered for parts of speech. Proper nouns seemed to contain much of the important information as the Wikipedia page was mostly about events that were carried out by people at specific locations. The resulting proper noun graph was still large and illegible. This graph was then clustering using modularity clustering. The resulting clusters were able to group related events, persons and locations.
- The Yelp dataset was an illustration of the versatility of both the text graph framework and clustering method. The Yelp dataset is effectively a document

collection where each document is relatively short. Visualising each document in turn would only produce anecdotal insights whereas what is required is insight across the entire dataset. The first 1000 reviews were grouped and a single text graph was created for them. The graph was clustered using modularity clustering which captured themes within the collection of reviews.

The framework was effective for different document lengths. Modularity clustering proved effective at extracting meaningful clusters with or without part-of-speech filtering. Filtering by node sizes, parts-of-speech particularly proper nouns, nouns and adjectives along with modularity clusters were able to produce meaningful summaries.

A shortcoming of the approach is that nodes referring to the same named entity can be separate. An example is *Farmer Boggis* and *Boggis*, who are the same character from Fantastic Mr Fox. Size, colour, position in space and edges are used to interpret the visualisation. When nodes representing the same entity are separate it is possible for them to occupy different clusters, positions in space and sizes. This suggests that a user would have to analyse each node separately and combine the conclusions. If the user is unaware of which nodes are the same then the analysis will be incomplete. This motivates the need to find an automatic system of combining words that refer to the same named entities.

7.3 Final remarks

Graph features proved effective in both areas of natural language processing where they were applied, text classification and text visualisation. Improved text classification makes it easier to find relevant documents while improved text visualisation makes it possible to review the documents for exact content without reading them. Improved methods of finding relevant information are increasingly important when there is abundant information available over the internet and other digital sources. When documents are large and dense a visualisation tool can make them easier to reach and extract meaning from. This has important implications for large and sometimes dense documents produced by government, scientific and parastatal agencies for public consumption.

References

[1] R. Albert, H. Jeong and A. L. Barabasi. "The diameter of the world wide web". In: *Nature* 401 (1999), pp. 130-131.

[2] E. G. Allan Jr., W. H. Turkett Jr. and E. W. Fulp. "Using Network Motifs to Identify Application Protocols". In: *Proceedings of the 28th IEEE Conference on Global Telecommunications*. GLOBECOM'09 (2009), pp. 4266-4272.

[3] E. Anderson. "The Species Problem in Iris". In: *Annals of the Missouri Botanical Garden* 23.3 (1936), pp. 457-509.

[4] S. Arumugam, I. Sahul Hamid and A. V.M. "Decomposition of Graphs into Paths and Cycles". In: *Journal of Discrete Mathematics* 2013 (2013).

[5] M. J. Bannister, D. Eppstein, M. T. Goodrich and T. Lowell. "Force-Directed Graph Drawing Using Social Gravity and Scaling". In: *Graph Drawing: 20th International Symposium, GD 2012, Redmond, WA, USA, September 19-21, 2012, Revised Selected Papers*. Ed. by W. Didimo and M. Patrignani. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 414-425.

[6] A. Barabasi and R. Albert. "Emergence of Scaling in Random Networks". In: *Science* 286.5439 (1999), pp. 509-512.

[7] A. Barabási, H. Jeong, Z. Néda, E. Ravasz, A. Schubert and T. Vicsek. "Evolution of the social network of scientific collaborations". In: *Physica A: Statistical Mechanics and its Applications* 311.3 (2002), pp. 590-614.

[8] Y. Bengio, R. Ducharme, P. Vincent and C. Jauvin. "A Neural Probabilistic Language Model". In: *Journal of Machine Learning Research* 3 (2003), pp. 1137-1155.

[9] R. Blanco and C. Lioma. "Graph-based Term Weighting for Information Retrieval". In: *Information Retrieval* 15.1 (2012), pp. 54-92.

[10] B. E. Boser, I. M. Guyon and V. N. Vapnik. “A Training Algorithm for Optimal Margin Classifiers”. In: *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*. COLT ’92. New York, NY, USA: ACM, (1992), pp. 144-152.

[11] S. Brin and L. Page. “The Anatomy of a Large-scale Hypertextual Web Search Engine”. In: *Computer Networks and ISDN Systems* 30.1-7 (1998), pp. 107-117.

[12] K. Byrne. “Nested named entity recognition in historical archive text”. In: *ICSC 2007 International Conference on Semantic Computing*. ICSC ’07 (2007), pp. 589-596.

[13] C. Chang and C. Lin. “LIBSVM: A Library for Support Vector Machines”. In: *ACM Transactions on Intelligent Systems and Technology* 2.3 (2011), pp. 27:1-27:27.

[14] P. Cho. *News Story Maps*. 2009. <URL: http://typotopo.com/projects.php?id=mappingnews> (visited on 12/05/2016).

[15] C. Collins, S. Carpendale and G. Penn. “Docuburst: Visualizing Document Content Using Language Structure”. In: *Proceedings of the 11th Eurographics / IEEE - VGTC Conference on Visualization*. EuroVis’09 (2009), pp. 1039-1046.

[16] C. Cortes and V. Vapnik. “Support-Vector Networks”. In: *Machine Learning* 20.3 (1995), pp. 273-297.

[17] A. Dasgupta, P. Drineas, B. Harb, V. Josifovski and M. W. Mahoney. “Feature Selection Methods for Text Classification”. In: *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. KDD ’07 (2007), pp. 230-239.

[18] R. D. DeCamp P, Frid-Jimenez A, Guiness J. “Gist icons: Seeing meaning in large bodies of literature”. In: *Proceedings of the IEEE Symposium on Information Visualization (InfoVis ’05)*. (2005).

[19] R. Fan, P. Chen and C. Lin. “Working Set Selection Using Second Order Information for Training Support Vector Machines”. In: *Journal of Machine Learning Research* 6 (2005), pp. 1889-1918.

[20] J. Feinberg. *Wordle*. 2014. (visited on 2016).

[21] I. Feinerer, K. Hornik and D. Meyer. “Text Mining Infrastructure in R”. In: *Journal of Statistical Software* 25.5 (2008), pp. 1-54.

[22] R. A. Fisher. “The Use of Multiple Measurements in Taxonomic Problems”. In: *Annals of Eugenics* 7.7 (1936), pp. 179-188.

[23] R. W. Floyd. “Algorithm 97: Shortest Path”. In: *Communications of the ACM* 5.6 (1962), p. 345.

[24] S. Fortunato. “Community detection in graphs”. In: *Physics Reports* 486.3-5 (2010), pp. 75-174.

[25] T. M. J. Fruchterman and E. M. Reingold. “Graph Drawing by Force-directed Placement”. In: *Software: Practice and Experience* 21.11 (Nov. 1991), pp. 1129-1164.

[26] M. Girvan and M. E. J. Newman. “Community structure in social and biological networks”. In: *Proceedings of the National Academy of Sciences of the United States of America* 99.12 (2002). Ed. by L. A. Shepp, pp. 7821-7826.

[27] F. van Ham, M. Wattenberg and F. B. Viegas. “Mapping Text with Phrase Nets”. In: *IEEE Transactions on Visualization and Computer Graphics* 15.6 (2009), pp. 1169-1176.

[28] K. S. Hasan and V. Ng. “Automatic Keyphrase Extraction: A Survey of the State of the Art”. In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (2014), pp. 1262-1273.

[29] S. Havre, E. Hetzler, P. Whitney and L. Nowell. “ThemeRiver: Visualizing Thematic Changes in Large Document Collections”. In: *IEEE Transactions on Visualization and Computer Graphics* 8.1 (2002), pp. 9-20.

[30] E. H. Huang, R. Socher, C. D. Manning and A. Y. Ng. “Improving Word Representations via Global Context and Multiple Word Prototypes”. In: *Annual Meeting of the Association for Computational Linguistics (ACL)*. (2012).

[31] T. Joachims. “Making large-Scale SVM Learning Practical”. In: *Advances in Kernel Methods - Support Vector Learning*. Ed. by B. Schölkopf, C. Burges and A. Smola. Cambridge, MA: MIT Press, (1999). Chap. 11, pp. 169-184.

[32] T. Joachims. “Text Categorization with Suport Vector Machines: Learning with Many Relevant Features”. In: *Proceedings of the 10th European Conference on Machine Learning*. ECML ’98 (1998), pp. 137-142.

[33] S. N. Kim and M. Kan. “Re-examining Automatic Keyphrase Extraction Approaches in Scientific Articles”. In: *Proceedings of the Workshop on Multiword Expressions: Identification, Interpretation, Disambiguation and Applications*. MWE ’09. Stroudsburg, PA, USA: Association for Computational Linguistics, (2009), pp. 9-16.

[34] K. Kucher and A. Kerren. “Text visualization techniques: Taxonomy, visual survey, and community insights”. In: *Visualization Symposium (PacificVis), 2015 IEEE Pacific*. Hangzhou, China: IEEE, (2015).

[35] Y. LeCun, L. Bottou, G. B. Orr and K. B. Muller. "Efficient BackProp". In: *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*. London, UK, UK: Springer-Verlag, (1998), pp. 9-50.

[36] D. D. Lewis. "Naive (Bayes) at Forty: The Independence Assumption in Information Retrieval". In: *Proceedings of the 10th European Conference on Machine Learning*. ECML '98. London, UK, UK: Springer-Verlag, (1998), pp. 4-15.

[37] Z. Liu, W. Huang, Y. Zheng and M. Sun. "Automatic Keyphrase Extraction via Topic Decomposition". In: *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*. EMNLP '10 (2010), pp. 366-376.

[38] R. Mihalcea. "Unsupervised Large-vocabulary Word Sense Disambiguation with Graph-based Algorithms for Sequence Data Labeling". In: *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing*. HLT '05 (2005), pp. 411-418.

[39] R. Mihalcea and P. Tarau. "TextRank: Bringing Order into Texts". In: *Proceedings of EMNLP-04and the 2004 Conference on Empirical Methods in Natural Language Processing* (2004).

[40] T. Mikolov, M. Karafiát, L. Burget, J. Cernocky and S. Khudanpur. "Recurrent neural network based language model.". In: *INTERSPEECH*. Ed. by T. Kobayashi, K. Hirose and S. Nakamura. ISCA, (2010), pp. 1045-1048.

[41] G. A. Miller. "WordNet: A Lexical Database for English". In: *Communications of the ACM ACM* 38.11 (1995), pp. 39-41.

[42] M. Newman. *Networks: An Introduction*. New York, NY, USA: Oxford University Press, Inc., (2010).

[43] M. E. Newman. "Modularity and community structure in networks". In: *Proceedings of the National Academy of Sciences of the United States of America* 103.23 (2006), pp. 8577-8582.

[44] A. Y. Ng and M. I. Jordan. "On Discriminative vs. Generative Classifiers: A Comparison of Logistic Regression and Naive Bayes". In: *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*. NIPS'01 (2001), pp. 841-848.

[45] N. Nicolosi. "Feature Selection Methods for Text Classification". PhD thesis. (2008).

[46] K. Nigam, A. K. McCallum, S. Thrun and T. Mitchell. "Text Classification from Labeled and Unlabeled Documents Using EM". In: *Machine Learning* 39.2-3 (2000),

pp. 103-134.

[47] K. Nigam, A. K. McCallum, S. Thrun and T. Mitchell. "Learning to Classify Text from Labeled and Unlabeled Documents". In: *Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence*. AAAI '98/IAAI '98 (1998), pp. 792-799.

[48] W. Paley. "TextArc: Showing Word Frequency and Distribution in Text". In: *Proceedings of IEEE Symposium on Information Visualization* (2002).

[49] B. Pang, L. Lee and S. Vaithyanathan. "Thumbs Up?: Sentiment Classification Using Machine Learning Techniques". In: *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10*. EMNLP '02 (2002), pp. 79-86.

[50] J. Platt. "Fast Training of Support Vector Machines using Sequential Minimal Optimization". In: *Advances in Kernel Methods: Support Vector Learning*. Ed. by B. Schölkopf, C. Burges and A. Smola. MIT Press, (1998).

[51] D. E. Rumelhart, G. E. Hinton and R. J. Williams. "Neurocomputing: Foundations of Research". In: Ed. by J. A. Anderson and E. Rosenfeld. Cambridge, MA, USA: MIT Press, (1988). Chap. Learning R, pp. 696-699.

[52] G. Salton and C. Buckley. "Term-weighting Approaches in Automatic Text Retrieval". In: *Inf. Process. Manage.* 24.5 (1988), pp. 513-523.

[53] C. M. Schneider, V. Belik, T. Couronné, Z. Smoreda and M. C. Gonzalez. "Unravelling daily human mobility motifs". In: *Journal of The Royal Society Interface* 10.84 (2013), p. 20130246.

[54] S. S. Shen-Orr, R. Milo, S. Mangan and U. Alon. "Network motifs in the transcriptional regulation network of Escherichia coli". In: *Nature Genetics* 31 (2002), pp. 64 - 68.

[55] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.1 (2014)

[56] K. Toutanova, D. Klein, C. D. Manning and Y. Singer. "Feature-rich Part-of-speech Tagging with a Cyclic Dependency Network". In: *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*. NAACL '03 (2003), pp. 173-180.

[57] V. Vapnik. *Estimation of Dependences Based on Empirical Data: Springer Series in Statistics (Springer Series in Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1982.

[58] D. J. Watts and S. H. Strogatz. “Collective dynamics of /‘small-world/’ networks”. In: *Nature* 393.6684 (1998), pp. 440-442.

[59] J. J. Webster and C. Kit. “Tokenization As the Initial Phase in NLP”. In: *Proceedings of the 14th Conference on Computational Linguistics - Volume 4*. COLING ’92. Stroudsburg, PA, USA: Association for Computational Linguistics, 1992, pp. 1106-1110.

[60] J. Xue and D. M. Titterington. “Comment on”On Discriminative vs. Generative Classifiers: A Comparison of Logistic Regression and Naive Bayes“”. In: *Neural Processing Letters* 28.3 (2008), pp. 169-187.

[61] Y. Yang and J. O. Pedersen. “A Comparative Study on Feature Selection in Text Categorization”. In: *Proceedings of the Fourteenth International Conference on Machine Learning*. ICML ’97 (1997), pp. 412-420.

[62] T. Zhou, J. Ren, M. Medo and Y. C. Zhang. “Bipartite network projection and personal recommendation.”. In: *Physical review. E, Statistical, nonlinear, and soft matter physics* 76 4 Pt 2 (2007), p. 46115.

[63] W. Zucchini and I. L. MacDonald. *Hidden Markov Models for Time Series: An Introduction Using R*. 1st. Chapman and Hall/CRC, 2009.

Appendix A

Parameter settings for neural networks

A.1 Structure of neural networks

There are various features of a neural network that need to be determined: the number of hidden layers used, numbers of nodes in each layer, functional forms for the activation function, data standardization, learning rates and associated *learning* features such as dropout and mini-batch size (discussed below). These features have been used in various neural network implementations by LeCun (1998), Bengio et. al. (2003), Mikolov et. al. (2010) and Huang et. al (2012). These features generally determine the accuracy and speed of the neural network. As the number of possible parameter combinations is extremely large, it is not possible to search exhaustively for the best combination. We therefore conducted a preliminary investigation in which each parameter was varied independently of the others, in order to choose a suitable combination of parameters. All models use a neural network with two hidden layers, on the basis of initial tests finding these more effective than a single layer network.

Various first and second hidden layer sizes were tested between 64 neurons and 2048 neurons. The default sigmoid activation function was used along with batch gradient descent and two learning rates 0.1 and 1. Figure A.1 shows two heatmaps representing predictive accuracy for a given number of neurons in the first and second layers of a neural network, along the columns and rows respectively. The heatmap on the left shows the neural networks trained with a learning rate of 0.1 and the heatmap on the right shows the neural networks trained with a learning rate of 1. The darker colours indicate lower

predictive accuracy and the brighter colours indicate higher predictive accuracy. The best model, measured by test accuracy relative to the number of neurons, had a first hidden layer of 256 and a best second hidden layer of 512.

Dropout is a method developed by Srivastava et al. (2014) that seeks to improve the robustness of a neural network. Dropout achieves this by randomly removing some neurons while the network is training. These neurons do not contribute to certain training iterations. This forces the remaining neurons to learn to create the correct prediction. This ensures that no single neuron is overpowering the rest when predictions are made.

Stochastic gradient descent can converge faster and to better solutions that batch gradient descent (LeCun, 1998). This is because fewer examples are used in a training iteration. In addition when the data is randomised the neural network cannot learn the order of the data. The advantage of using subsets of the data is that the neural network is trained quicker.

For each method a parameter must be selected: for dropout (Srivastava et al., 2014) the number of neurons to drop in each training iteration; for stochastic gradient descent the size of the subset or mini-batch that must be trained in each iteration. The dropout value was varied from 0 to 0.75. At 0 no dropout was applied. At 0.75, 75% of neurons were being excluded from training between the first and second layers. Dropout above 0.75 was not tested as results were increasingly becoming poorer. The stochastic gradient descent mini batch sizes were varied from 0.1 to 0.9. Zero was not tested as it implies training no data. Batch gradient descent is equivalent to a mini batch size of 1 which has been used when selecting the layer sizes. Mini batch sizes below 0.1 were not tested as test accuracy was decreasing with mini batch size at that level. Figure A.2 has two images, the left image shows the results of the hyper parameter search for dropout and mini batch size. The darker colours indicate lower predictive accuracy and the brighter colours indicate higher predictive accuracy. Dropout of 0 with mini batch size of 0.75 produced the best results, by test accuracy.

The activation function was the last feature of the neural network that had to be selected. The traditional activation function borrowed from logistic regression is the sigmoid function. The more recent activation function is the hyperbolic tangent. The sigmoid function produces values between 0 and 1 while the hyperbolic tangent produces values between -1 and 1. The hyperbolic tangent has larger derivatives than the sigmoid as it has double the height. This is especially true when evaluated close to zero. This can help the neural network train faster as larger gradients allow gradient descent to reach the optimal parameter in fewer steps (LeCun, 1998). In order to take advantage of the larger gradients

the data should be normalised so it can lie close to zero.

There are two ways to normalise data. Variable wise (column wise) or document wise (row wise). Both normalisation divide the data by its standard deviation thus most of the elements will be close to the range between 1 and -1. Figure A.2 has two images, the one on the right shows the predictive accuracies of the hyper parameter search for activation functions and data normalisation. The darker colours indicate lower predictive accuracy and the brighter colours indicate higher predictive accuracy. The results did not favour the hyperbolic tangent nor did they favour normalisation. The best model used the sigmoid function without standardisation. The final model, which had high test accuracy but relatively few parameters, had first hidden layer of 256 and second hidden layer of 512. No drop out was used with mini batch sizes of 0.75. No normalisation was used with the sigmoid activation function.

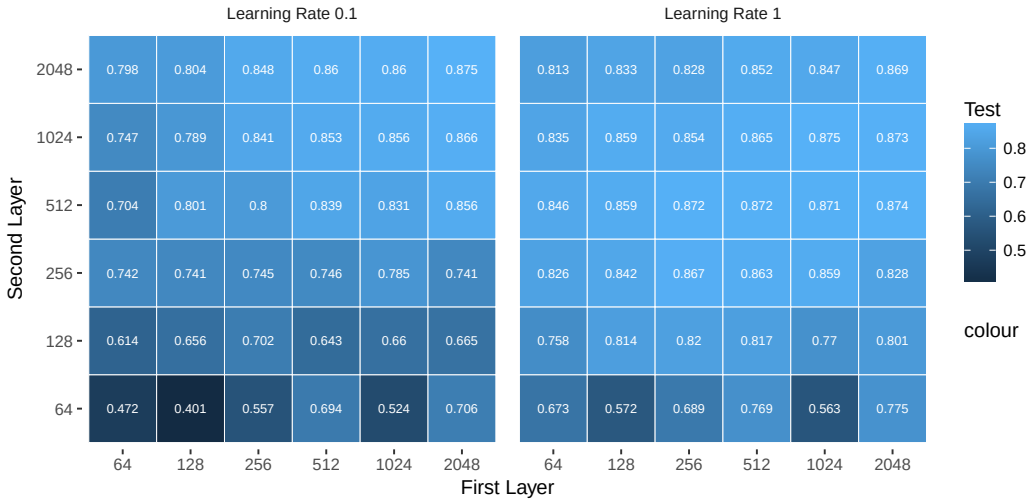

Figure A.1: Parameter setting for neural networks: learning rates 0.1 and 1, several layer size for each layer. Lighter colour corresponds to higher test accuracy.

A.2 Enhancing performance of neural networks

When implementing methods computational underflow can become a concern. Underflow is when a numerical value becomes so small that it can only be stored in memory as a zero. Zeroes and ones create undefined values within the cross entropy cost function as they both require the evaluation of the log(0). For this reason, the square root of the output layer was evaluated in the cross entropy cost function. Stated differently the final

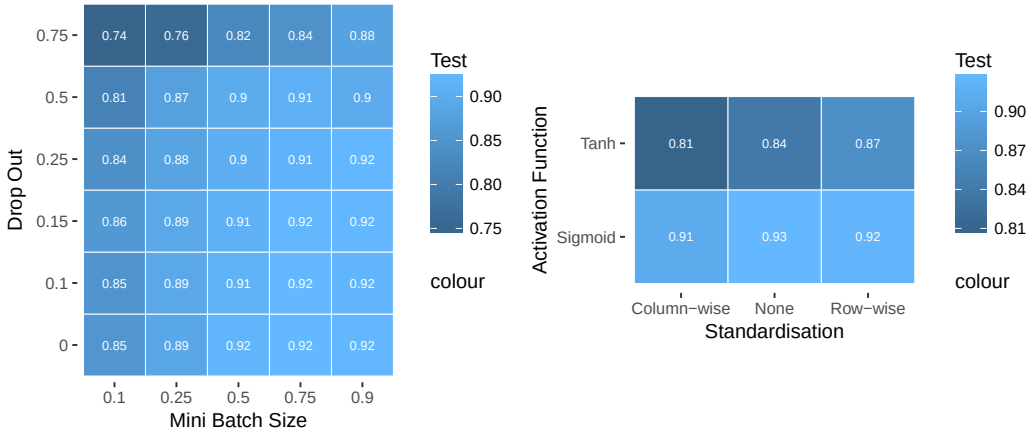

Figure A.2: Parameter settings for neural networks. Drop out, stochastic gradient descent mini batch sizes, activation functions and standardisation. Lighter colour corresponds to higher test accuracy.

activation function was the square root of the softmax.