*Article Recommendation with Artificial Intelligence*

**Thomas Rogers**

The University of Derby

Abstract

This paper intends to explore the effectiveness of a variety of artificially intelligence methodologies, such as artificial neural networks, to create a user-based recommendation engine. The project will leverage multiple attributes in an article, such as sentiment, length, content, and subject, to recommend articles based on their users previously liked articles. Users will be required to submit their articles to a front end interface, and these, along with scraping of a set of predefined websites will be the data-set by which we create recommendations.

Traditional recommendation engines often rely on a collaborative filtering approach, where user preferences are based on other users’ data. In our case, we’d look at two users’ articles, and found that User 1 liked the articles{P,Q,R,S} and User 2 liked {P,Q,R} we might reasonably be able to assume that User 2 would enjoy article S. This ignores a lot of the minutiae and features of the article, such as length, tone or even writing style that a content-based system allows for. This project will focus on a Content-Based approach.

Acknowledgements

Dave Voorhis: Supervisor.

Table of Contents

[1. Introduction 8](#_Toc474165264)

[1.1. Problem Background 8](#_Toc474165265)

[1.2. Project Aim and Objectives 9](#_Toc474165266)

[2. Literature Review 10](#_Toc474165267)

[2.1. Introduction 10](#_Toc474165268)

[2.2. Recommendation Systems: State of the art. 11](#_Toc474165269)

[2.2.1. Collaborative filtering 11](#_Toc474165270)

[2.2.2. Content-Based Recommendations 13](#_Toc474165271)

[2.2.1. Recommendation System Conclusions 15](#_Toc474165272)

[2.3. Neural Networks 16](#_Toc474165273)

[2.3.1. Multilayer Classifiers: 16](#_Toc474165274)

[2.4. Key Concept 3 17](#_Toc474165275)

[2.5. Conclusions 17](#_Toc474165276)

[2.5.1. Key Issues 19](#_Toc474165277)

[2.5.2. Refined Research Questions 19](#_Toc474165278)

[3. Research Methodology 20](#_Toc474165279)

[3.1. Introduction 20](#_Toc474165280)

[3.2. Corpus 20](#_Toc474165281)

[3.3. Corpus Feature Extraction 21](#_Toc474165282)

[3.3.1. Feature 1: Length 21](#_Toc474165283)

[3.3.2. Feature 2: Sentiment 21](#_Toc474165284)

[3.3.3. Feature 3: Flesch–Kincaid readability score 22](#_Toc474165285)

[3.4. Neural Network Architecture 23](#_Toc474165286)

[3.5. Limitations 23](#_Toc474165287)

[3.6. Conclusions 23](#_Toc474165288)

[4. Findings and Analysis 24](#_Toc474165289)

[4.1. Introduction 24](#_Toc474165290)

[4.2. Analysis 24](#_Toc474165291)

[4.3. Conclusions 24](#_Toc474165292)

[5. Discussion 25](#_Toc474165293)

[5.1. Introduction 25](#_Toc474165294)

[5.2. Issue 1 25](#_Toc474165295)

[5.3. Issue 2 25](#_Toc474165296)

[5.4. Conclusion 25](#_Toc474165297)

[6. Conclusions and Recommendations 26](#_Toc474165298)

[6.1. Conclusions 26](#_Toc474165299)

[6.1.1. Conclusion 1 26](#_Toc474165300)

[6.1.2. Conclusion 2 26](#_Toc474165301)

[6.2. Recommendations 26](#_Toc474165302)

[6.2.1. Recommendation 1 26](#_Toc474165303)

[6.2.2. Recommendation 2 26](#_Toc474165304)

[7. Bibliography 27](#_Toc474165305)

[8. Appendices 28](#_Toc474165306)

# Introduction

## Problem Background

We live in the age of information, where it is possible for anyone to relay and distribute information at the touch of a button. With this, however, comes the problem of information overload. Information overload describes the difficulty comprehending or decision making when presented with a wide array of information (Toffler, 1971). In an age where content creators such as the New York Times can publish between 200-500 articles a day (Meyer, 2016) the act of disseminating and classifying this information becomes vitally important.

Recommendation Systems aim to filter and categorize this information to make it more applicable to an individual. They aim to expose preferences of users, or to classify items into a pool of ratings, such as a 1-5 scale. Recommendation systems are used in a range of domains, including but not limited to: Product Recommendation, movie recommendation, online-dating, and search queries. These systems help users find meaningful content amongst the breadth of content that is posted daily online.

These systems traditionally work in one of two ways, collaborative filtering and content-based filtering.

Collaborative based filtering can be seen as an automated extension to how people traditionally are recommended things; relying on others to form meaningful recommendations. Users preferences are stored in a utility matrix storing their rating for an item. From this matrix we can compare between users likes and dislikes and form predictions in the sparse areas of a user’s matrix. This assumes that user’s interests and disinterests are similar, and that one user’s interests can be derived from analysis of other user’s interests.

A collaborative filtering approach arguably, ignores important language and context features that may be important for users. Additionally, a collaborative approach to filtering also poses the risk of recommending users items outside of their preferred area of interest; it is easier to find similarity between specific items than it is to find two users that share exactly the same areas of interest. In fact, collaborative based filtering runs the risk of increasing homogeneity, as it is suggested that users with the most activity tend to influence scoring towards more popular products (A. Kent et al, 1970).

The content-based approach, conversely, attempts to predict recommendations based on a set of descriptors and a user profile. This user profile is built by analysing a collection of user defined likes and dislikes or, alternatively, through a user created profile listing their likes and dislikes. Content-based systems provide a more individually oriented set of recommendations and can be weighted towards specific features users enjoy.

## Project Aim and Objectives

The aim of this project is to develop a content-based recommendation system, using a variety of attributes extracted from an article to form a user-profile. This will be a Feed-Forward Neural network, which will take as its input a variety of normalized inputs. Our intention here is to monitor the efficacy on a simple neural network, and to conclude whether further testing into more complicated models are required.

Our first objective is to perform a literature review, analyse the current state of the art, and to gain understanding of the current models to ensure that we can utilize and expand upon them.

One objective is then to create a model, and assess the efficacy, of a neural network as the content-based recommendation system. This will also involve analysis of existing models, data-analysis of articles and extracting features to analyse. These features focus on several natural language problems, such as sentiment analysis and sentence classification. These will form the basis of the recommender, which will take characteristics of an article and use them to update the user profile weights.

We will then evaluate whether this model will be appropriate for a front-facing user application, and gve recommendations on how this should be implemented.

# Literature Review

## Introduction

The intent of this literature review is to assess research documents, and the current state of the art to form a rationale for the belief that neural networks provide a promising foundation for content-based recommendation.

Firstly we look towards the state of the art recommendation systems, by evaluating existing models, including collaborative filtering and content-based filtering. This will provide invaluable insight into the rationale of other recommendation systems, expose some existing algorithms and introduce some of the ways in which textual information is processed and disseminated.

The section will then then branch into discussing neural networks and their uses across a wide range of domains. The impetus of this research is to gain understanding of the justification behind using neural networks in different domains, and to expose some common links and ideas that can be built upon. This research will also tackle some of the existing neural network models at a high-level. This will help in selecting an appropriate model for the task of creating a content based recommendation system.

Additional to this any relevant citations will be followed to find related works. These may not strictly be related to neural networks, but will hopefully provide historical context or provide a non-computational motivation for the work being done.

This will form the basis of a coherent argument as to why content-based recommendation is beneficial, and why a neural-network based approach is sensible.

## Recommendation Systems: State of the art.

Recommendation systems have become increasingly necessary in a word dense with information. They aim to filter out information not relevant to the user, and aid in reducing information overload.

Information overload describes the difficulty comprehending or decision making when presented with a wide array of information (Toffler, 1971). Reducing this is imperative in an age where content creators such as the New York Times can publish between 200-500 articles a day (Meyer, 2016).

Recommendation systems are usually categorized by their filtering type. The two most popular are ‘collaborative’ and ‘content-based’ filtering.

### Collaborative filtering

Collaborative Filtering is the original approach to filtering content based on user preference (Schafer, Frankowski, Herlocker and Sen, 2007). It asserts that you can source new recommendations from other users who have similar tastes. It is considered to be the most popular and implemented recommendation system.

There are many ways in which Collaborative filtering recommendation systems are implemented. One of the most common is the Nearest neighbours approach (Ricci, 2011) These approaches, also known as memory-based approaches, are simple and efficient.

Data in collaborative filtering systems are represented as scalar (such as ratings), binary (liked or disliked), or unary (capturing tertiary information, without gaging actual appreciation). All three of these can provide invaluable insight into a user’s interactions, although scalar information, and some forms of binary information presents more explicit information, which can be much more beneficial than the implicit information gained from unary data.

In collaborative based filtering we approach the problem by presenting a multiclass classification, or regression, problem. We attempt to learn a function that predicts new ratings for items yet unrated. Or *f*(u,i) where u is a user, and I is an item. We use a matrix of Users (U) and Items (I) and a set of Recommendation data (R). We then perform a matrix operation to maximise the similarity between Iu and Iv where u and v represent users. The following formula is then used.

*Pu,i* = vu + k ∑j in U *w(u,j)(vji – v\*j)*

Where w is the weight of the user, and v\* is the mean rating for user u and vji is the rating of item i given by user j. K is a normalization factor (Konstan et al., 1997).

Accuracy is then established using a validation and training set of values. We repeat this process, reducing the error as we go.

This approach has inherent problems with data sparsity. Data sparsity is where user matrices share few or no similarities, making it difficult to establish similarities and thus create recommendations for users (Grčar, 2005). This data sparsity becomes an issue in an environment where users are very homogeneous (ie. There is no common traits between users) or indeed when there are few users.

Grcar also touches upon cosine Similarity for establishing similarity between two vector sets. This could be useful when approaching the issue of finding similarity between documents for our neural network. We can utilize this for a bag-of-words approach, or create ‘word vectors’. We will touch upon this more in the subsequent sections. In a collaborative method it is used to establish the weights between two user/item vectors.

A nearest neighbour approach is not the only approach to collaborative filtering. We can consider collaborative filtering as a classification or regression task. These employ a variety of techniques, one of which is the use of Bayesian models (Meng and Chen, 2009)

These methods help us establish ratings for items in a sparse matrix, therefore allowing us to predict ratings over large datasets with incomplete data. In other words, It handles the data sparsity issue of memory based approaches.

In Chen and George’s Bayesian model users are modelled into subgroups based on their ‘rating probabilities.’ We can then use these to approximate preference groups for ratings, and source our rating predictions based on the user and what group they fall into. It uses some established (or supervised) data vectors; items (I), ratings (R) and Judges (J), and one unestablished (or unsupervised) data vector of groups (G). The probability of all judges in group G is then established as the product of multinomial probabilities. The product of all judges in J, over the sum over all groups of a multinomial probability based on whether the judge rated the item.

Difficulties in this approach stem from searching for an effective group vector. An incredibly large data set is used, and finding groups from this can be a difficult task. A hybrid search approach is used, and only a small selection of the dataset is utilized, which may provide inaccurate grouping results.

Other collaborative based approaches use Support Vector Machines (SVM), Decision trees and most importantly Neural Network implementations. I will research neural network based approaches further on in this research section.

While Collaborative based approaches provide a simple and intuitive approach for sourcing recommendations they do have some significant limitations. One such limitation is the increased homogeneity of content recommendations. It is suggested that users with the most activity tend to influence scoring towards more popular products (A. Kent et al, 1970). This ultimately means that popular items will be weighted more heavily. A more nuanced approach, such as accounting for item frequency, will also need to be considered. This increases the complexity of the model.

Additionally, there is a trade-off to be made between accuracy and computational complexity Lee. Sun and Lebanon (Lee, 2012) conclude that whilst methods that utilize matrix factorization provide the most accurate results they require a lot of computational power to create. This, in a consumer product, is not a positive.

The computational power trade-off is one that needs to be considered for a Neural network based approach. Neural networks are traditionally quite slow to train, and some considerations will need to be made between the accuracy and complexity of the model and the computational requirements needed to use it in a user facing application. This will be discussed more in the methodology sections, as well as in the research section of neural networks.

From the above analysis, collaborative based filtering presents issues due to a perhaps fundamental misunderstanding; users with similar preferences do not necessarily share the same interests. Content-based recommendation systems circumvent this issue, by correlating features and users, rather than users and users.

### Content-Based Recommendations

Another approach for Recommendation systems is content-based filtering. Recommendations here are based on a user’s interactions with the system, or from a ‘user profile.’ (Brusilovsky, Kobsa and Nejdl, 2007). This profile is formed of preferences towards relevant properties and fields that define the item being recommended. Content-based recommendations therefore focus more upon the individual user and their preferences, rather than trying to establish links between groups of users.

User profiles play a pivotal role in a content-based recommendation system. The impetus is to create a model of user preference, by utilizing representations of item features, whether these be derived features, such as length, or user-provided features such as ratings. User profiles may also store a history of user-interaction. This model can then be used as the input of a function that can be used as the input of a function that retrieves a list of items to recommend.

Feature extraction is critical when creating a content based recommendation system. For instance, Oord, Dieleman and Shrauwen, when forming their content-based music recommendation system cited difficulty with extracting features from a piece of audio. (Oord et al, 2013). They cited a semantic gap with music – that high level features are difficult to extract. They cited a number of music information retrieval techniques that enabled them extract these high level features, which enabled them to utilize it as their input vectors.

Similarly, there is a semantic gap issue in feature extraction on articles. As previously mentioned methods like TF.IDF provide a good basis for retrieving a vector spaced representation of the key words in the article, but finding an appropriate way to extract features such as sentiment, identifying related concepts and the weighting that each topic should have are non-trivial. These features can be hard, but not impossible to extract, as many of the features of an article will be unrestricted text. This will be discussed in the research subsection regarding natural language processing.

When extracting features for a user profile there are different types of information that can be extracted. Explicit information, provided by the user, ie: ratings, such as those provided in Ali K’s TiVo recommendation system (Ali and Stam, 2004), or implicit recommendations, that can be inferred from a user’s interactions with a system, such as the articles read, time spent on an article (weighted in terms of length) or classifying articles into K-Clusters.

Some Content-Based recommendation systems utilize a decision tree and rule induction approach to learning information. One such example is the ID3 Decision Tree outlined by Quinlan J, (Quinlan, 1986). Here the decision trees are given individual words or phrases as criterion to classify information and categorize articles.

However, decision trees, whilst providing good foundations for structure data, are not suitable for unstructured tasks such as those found in natural language processing. Ultimately, it is argued that decision trees, whilst promising, provide poor accuracy in text classification (Pazzani and Billsus, 1997).

Despite this, Kim, J et al have developed a system using decision trees that create personalized advertisements. While not exactly in the same domain, it does show that decision trees can be used to identify and create content based on individual users. (REFERENCE RESULTS PAGE OF ARTICLE) (Kim et al, 2006).

The decision tree learning approach do not provide an adequate way to disseminate and identify user trends, instead, they provide simple classifiers. This may be beneficial to categorize articles into categories, which could then be used to provide a more accurate representation of users tastes. However we cannot provide a good overview of user taste across multiple domains, as few users enjoy articles from one domain, and it is natural to assume that multi-domain articles will not share content similarities. (Reference).

A K-Neighbours approach may be suitable to ensure that articles are classified into the correct domain. This then ensures that any comparative analyses between articles is done against articles that are thematically similar. One way to accomplish this is to vectorise the article, using the TF.IDF method described earlier. It is then possible to cluster articles based on their cosine similarity, or their distance from a centroid, or point of interest.

Other, more advanced methods, use distributed representations of words, or ‘word vectors.’ One prominent example of this is Word2Vec (Milkov et al, 2013). It’s roots are in the skip-gram model, that states samplings of frequently clustered words can be represented as vectors, and that these vectors can be compositionally expressive. In other words, semantic similarity can be matched using simple mathematical operations. Word Vectors provide a promising inputs to many machine learning tasks.

Clustering provides a foundation for Content-based filtering, but may lack the specificity and accuracy needed for filtering based tasks.

Bayesian Networks, and more probabilistic networks, have been utilized by researchers when attempting to optimize the content-based filtering problem. One such approach attempts to utilize Naïve Bayes (Mcallum, 1998). Naïve bayes asks the question using a posteriori knowledge (pre-learned) can we classify document D into class C. This then uses Bayes theorem, as demonstrated below.

P(c|d) = P(c)P(d|c) / P(d)

This process is applied to a vector of classes and documents, (TODO : LATER)aaaa

One notable issue with this approach was that they perform poorly on variable length data, due to them needing to estimate parameters. They also have poor accuracy when data is sparse, such as the case of rare-categories. In the above model, a vocabulary vector was the only considered parameter, which does not provide us with sufficient information to accurately model user preference. Since content-based filtering can use a variety of document features to classify its articles, a linear learning based approach might be more suitable.

It is worth noting that content-based filtering systems do suffer from some significant shortcomings.

The most noticeable, and perhaps damning of which is the predilection for these systems to over-specialize. Indeed, it is true that content-based systems are weighted heavily towards items already rated. This is known as the serendipity problem; users may have trouble finding new and interesting content that does not fit into the categories they have already prescribed.

Indeed, this poses an issue – However, there are several solutions to this. In a system such as a news site, or article site, providing a fresh list of new articles for users to read may, regardless of their current user profile may help them expand their user profile, diversifying their tastes**. \*\*Important considerations to mention at end**

Additionally, it is important to note that collaborative and content based approaches are not mutually exclusive. Luis M de Campos et al demonstrated this by combining Bayesian networks that utilize user-oriented content filtering and collaborative filtering.

As such, a content-based filtering could provide a binary decision as to whether recommend an article to users based on several article features, such as sentiment, length or category.  **\*\*Important considerations to mention at end**

### Recommendation System Conclusions

It is clear from the evaluation into recommendation systems that the majority of work has been focused on collaborative based filtering. While collaborative based filtering does help provide heterogeneous results for filtering, helping avoid issues of specializing and content discovery, it makes assumptions about the similarities between people and the content they might enjoy. Additionally, it does not consider more intricate details of an article.

Content-recommendation, meanwhile, has had a focus on more detailed information about the content being recommended. It enables for a more personal approach utilizing user-profiles. There are concerns with this approach about the homogeny of content it can produce.

However, despite this there are plenty of reasons to utilize content-based filtering. It is important to remember that content-based filtering does not have to run in isolation, and can be combined into a hybrid-based filtering system.

Content-based filtering can provide a valuable insight into users reading habits, and can be used as a step between collaborative filtering to decide based on the articles content whether a user would enjoy it based on a ‘readers-profile.’

## Neural Networks

Neural networks owe their name to the biological concept of neurology; in the human brain each neuron is connected by a series of axons, these work in unison to form human cognition. Similarly, artificial neural networks present an interconnected network, that alter weightings between neurons that allow them to learn functions (Opening the Black Box, Artificial Neural Networks).

Neural networks are composed of neural units. These singular neural units, or perceptron’s were designed by Widrow and Hoff (Widrow, 1986). They could solve simple linear problems, but lacked the capability to solve more complicated, non-linear problems. This was expanded by Werbos (REFERENCE!!) to include multiple layers, and learn weightings between layers using Backpropagation. These multilayer perceptrons (MLP) could solve much more complicated, non-linear function approximation.

Since the advent of this in 1974 MLPs have been applied to a variety of interesting domains. This section aims to discuss some of the domains and the applications of neural networks with the aim of extracting any features and traits that may be useful in creating our neural networks.

### Multilayer Classifiers:

Classification tasks are perhaps one of the most active research areas currently for the use of Neural Networks, and have been utilized across a wide series of domains including banking and bankruptcy prediction, credit scoring, and perhaps the most iconic handwriting recognition.

In fact, the effectiveness of neural network architectures has an empirical basis; used and studied frequently in a variety of sectors. One such example of this is a comparative study performed Gian Carlo and Franco Varetto (Altman, Marco and Varetto, 1994). This study shows a feed forward neural network architecture, utilizing back propagation for weight updates, and a sigmoidal activation function. The figure below demonstrates a singular neural component that forms the basis of a MultiLayer Perceptron.

Input Vector X

Activation Function

Output

x1

x2

x3

Y(∑wixi)

w1

w2

w3

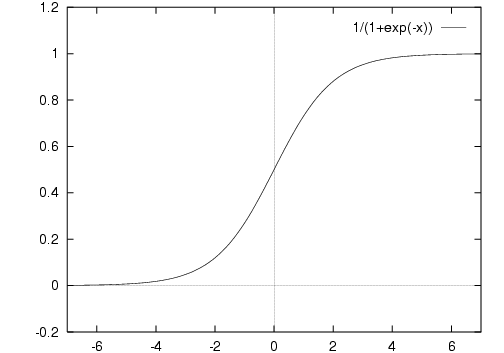
X1

## Key Concept 3

## Conclusions

***Figure 1. Typical Perceptron topology.***

Where X is the input vector, W is the weight vectors, and the activation in the case of this study is a sigmoid function. We apply this activation function to the sum of the weights\*input layer.



***Figure 2. Graph of the Sigmoid function.***

The Sigmoid function outputs a figure from range 0-1, and in this case the activation node, where the inputs and their weights are summed, will correctly trigger between 0.5 and 1.

This neural network took inputs related to financial information about a company, and performed a supervised learning task. That the training data used was separated into a testing, and validation set, where they could train their network and update weights, using backpropagation (figure 3.) The output was a number between 0 and 1, due to the sigmoidal nature of the hidden nodes. We then can use this model to approximate a classification based on the inputs.

### Convolution Neural Networks

An extension of the feed forward neural network model is a convolution neural network. Convolution neural networks advance the state of the art by utilizing the image processing technique of convolution.

Convolution neural networks are “deep” learning networks, which mean they incorporate a variety of layers. In a convolution, neural network these layers are responsible for different tasks.

One such layer is the convolving layers used to extract local differing features, from various regions of an image. The convolution layer of a CNN use a filter to extract features. A filter, or kernel searches an input for features shared across an image; they convolve with a learned kernel. In laymens, convolution can be understood in terms of a sliding window applying to a subsection of a matrix.

Convolving Kernel

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 |

***Figure 3. Simplistic visualisation of a convolving kernel.***

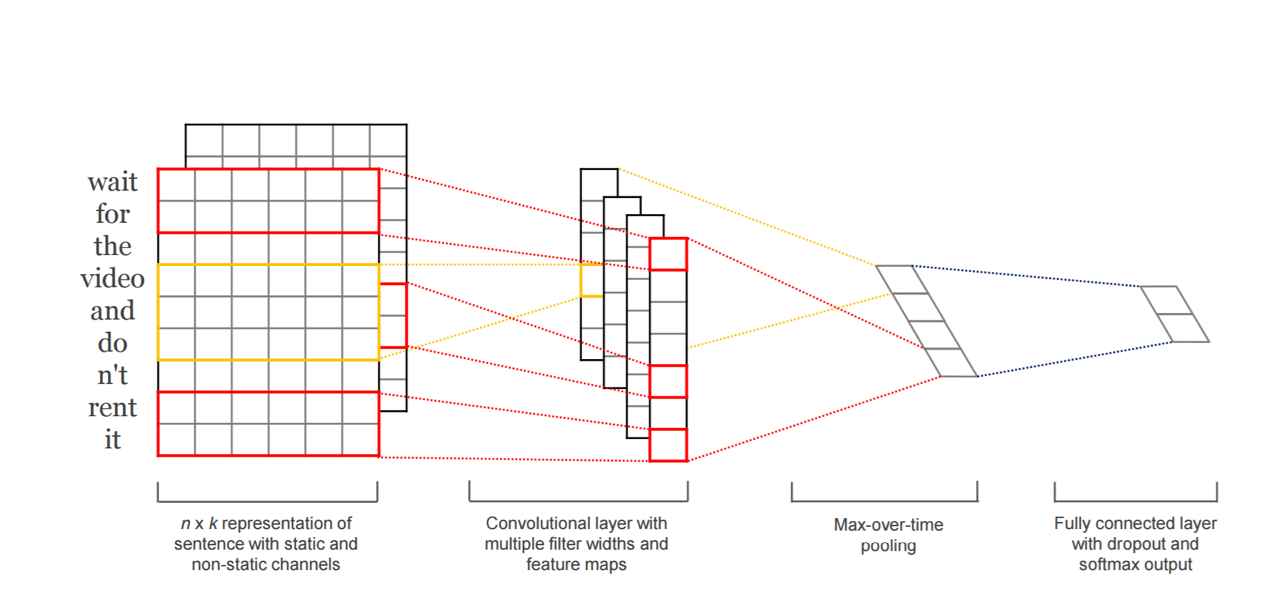
In the context of a neural network, these convolutions apply non-linear activation functions to slowly learn a value based on supervised input. We can then later use these filters to unlabelled examples.

Pooling layers then allows for subsampling of inputs. It allows us to manipulate information gained from the previous input to better contextualise it. It also allows for trimming of extremely large filters into a more manageable, standardised output.

Typically, these models are used for image processing tasks, but they are also used in natural language processing tasks. One such example of this is the sentiment analysis task performed by Yoon Kim, (Kim, 2014).

Yoon Kim’s work uses pretrained word vectors, in this case Google’s word2vec trained on a corpus from Google news. The goal was to classify sentences from film reviews into two categories, positive and negative. In other terms, the CNN in Yoon Kim’s work performed simplistic sentiment analysis.

Yoon Kim’s model applies filters that are used to extract a single feature from each filter. These filters are applied with multiple windows (as shown in figure 3), passed to a penultimate layer which pools the previous layers and goes through a SoftMax layer to output probabilities.



***Figure 4.Yoon Kim’s CNN Model (REFERENCE)***

CNNs may not be the best model for natural language processing tasks, other models, such as the recurrent neural networks we will discuss next can provide better accuracy, however, they are easily parallelizable and as shown above, some models can provide reasonable accuracy. They gap the bridge between computational effectiveness, and accuracy.

### Spotify and Deep Neural Networks

### Key Issues

### Refined Research Questions

# Research Methodology

## Introduction

This intent of this section is to document the parameters used, the rationale of these parameters, and the way in which they were extracted.

The first sub-section will discuss the corpus, and the rationale around why it was selected, and the features it contains. It will attempt to provide a critical look at the corpus, and suggest areas in which it could be improved to better suite our purpose. We will also provide a brief overview of how this corpus is pre-processed.

The second sub-section will focus on feature extraction over the corpus. It will look at the features that were extracted, provide an overview of the method used to extract the features, and finally discuss the reason as to why these features were extracted. This will attempt to justify the use of each feature.

The third sub-section will discuss the neural network architecture, including the hyper-parameters, the input, the number of hidden layers, the activation function and the ways in which we will update the weights.

Overall, this section will attempt to provide both methodology to repeat this study, but also attempts to elucidate the reader as to why we decided on these features. Hopefully this will allow the reader to question, and improve upon our techniques.

## Corpus

The corpus selected is a BBC news corpus provided for use as for machine learning problems. This data is compiled by the Insight centre for data analytics (Insight - BBC Datasets, 2006). The corpus consists of 2225 documents extracted from the BBC News websites from the years 2004-2005. Each individual article contains a title and article. All other meta-data has been extracted, including social media tags, hyperlinks and images.

The corpus is separated into 5 different sections, (business, entertainment, politics, sport and tech). These labels are encoded into the ‘Article’ class in the java code, and are used in the pre-processing.

This corpus was selected as it provides important, cogent features relevant to our study. Its single source means that there’s a lot less deviation in writing style and content. It also separates articles into a variety of topics, providing heterogeneity. The corpus features a formal-tone to the language that will reduce issues of colloquialism and allow for easier extraction of information such as ‘Part Of Speech Tags’ and word lemmatization.

All the articles in the corpus go through a pre-process step that lemmatizes all the language, based on their ‘part of speech’ tags. This is so that we can correctly cluster words such as adjectives, plurals nouns etc. This step is performed using the Stanford Natural Language Processing library (McClosky et al, 2014).

In future studies we would like to test the efficacy of our model on general web-content, not constrained to a single corpus. However, we feel the provided corpus is a good starting area, and allows for easy replicability.

## Corpus Feature Extraction

To provide input to our neural network we are required to extract a series of features, and normalize them. In the following section, we intend to outline the features to be extracted, and provide the rationale as to why these inputs will be used.

Sub-corpuses are created based on some predicates. Each article from this sub-corpus are analysed by a series of “article-extractors”. The outputs of these converters are used to form a labelled csv. This CSV file will be used as the input for our neural network, where each comma represents a new feature to analyse, and each new line represents a new article. Each of these articles will be labelled with a ‘1’ if the user ‘likes’ it, and a ‘0’ if they do not.

The sub-corpuses will be discussed further at the end of this section.

### Feature 1: Length

The first feature we consider is the length of the article. Article length is directly correlated with the amount of time an article takes to read. Studies have shown that there’s a link between length, and time taken to read, and reader engagement and retention. Studies by Medium show that 7 minutes is the ideal reading time for most users, which correlates with around 1600 words (Sall, 2013).

The above view considers the average reader of a medium article. This data may not be representative of individuals. Instead, since we are attempting to filter based on user profile it would be better to use prior information. By using the length of previously read articles.

We normalize this information so that 200 or fewer words are considered as a 0.0, and 1600 or more words are considered 1.0 (so that 800 words would be considered 0.5, etc). This normalization process is essential in ensuring that all the inputs to our neural network are from the same range.

### Feature 2: Sentiment

We also consider the sentiment score an important feature of the article, and the reader’s profile. We posit that frequently reading negatively charged articles, or positively charged articles may indicate a preference. Further to this, thanks to the encoding of topic, and writing styles, we are hoping to encode more complicated ideas such as “likes positive articles about technology, dislikes positive articles about politics.”

Initially, we used Yoon Kim (Kim, 2014) model to perform sentiment analysis on our corpus. However, we found the accuracy of this model lacking, finding that the accuracy of the model is around 70-75%, but optimized and trained for much smaller texts. We therefore posited that we instead would be better using a more mature service. With that in mind, we used IBM’s AlchemyAPI (IBM Alchemy, 2017)

IBMs Watson, and their alchemy API allowed us to send a query with a text-payload. It will then analyse this against their machine learning model, and return a -1.0–1.0 score where 1.0 is a strongly positive sentiment, and -1.0 is a strongly negative sentiment.

IBM Watson uses modern machine learning techniques to extract sentiment, avoiding the use of vocab features. It uses a recurrent neural network, which means it can alter it’s state, and therefore begins to understand sentence dependencies. For example, the IBM Watson model would be able to distinguish between “This move is not great” and “This move is great.”

IBM Watson provides a high degree of accuracy, and is constantly updates with new examples. As such, we cannot provide a deterministic way to provide these scores. However, there is a text file included in the source code project with the scores as of November 2016.

We normalize these scores to a 0.0-1.0 score, as to provide a standard input to the system.

### Feature 3: Flesch–Kincaid readability score

Another feature we feel is important is the readability score. The Flesch-Kincaid readability score determines the level of difficulty required to read an article. Developed by the U.S Navy in 1975 to determine the difficulty of technical documents, however, it has become a standard over a wide range of applications, including Microsoft word.

The Flesh-Kincaid readability formula is as follows where TW = Total Words, TS = Total Sentences and TSY = Total Syllables in the document.

*206.835 – 1.015(TW/TS) – 84.6(TSS/TW)*

***Figure 5. Flesch-Kincaid Readability score (Flesch, Rudolf. "How to Write Plain English". University of Canterbury. Archived from the original on July 12, 2016. Retrieved 12 July 2016.)***

The score factors heavily the amount of polysyllabic words. It purports that, if a lot of polysyllabic words are used in a document, that it will be more difficult to read.

We use this score, normalized so that a 75+ score is considered 0.0 and a lower score around 35 and lower is considered 1.0 as input values. We believe that these are important as recommending articles that are beyond the reading ability of the user may prove irritating, or worse, may dissuade a user from using our service. Our hope with this input is that it will filter articles that are either too trivial, or too difficult for a user to digest.

Additionally, as mentioned before, we hope that this will be synthesised into a more complicated relationship. A user may like more ‘difficult’ articles about technology, such as technical documents, but perhaps more trivial articles about entertainment.

### Feature 4: WordVector Cosine Similarity and TF-IDF

We also perform a text frequency, inverse document frequency (TF-IDF) analysis on the documents to find the important words in each document. We hope that with this analysis we can encode, more accurately, similarity between this article and other articles in a dataset.

Firstly, we perform a TF-IDF analysis on the document, against the corpus. This should help us terms that are important in a document, as it returns a weighting score for how pertinent the information is. We store this in a text document, and later place then into a hash map for later retrieval.

The TF-IDF score is retrieved using a TF-IDF class. It performs the following operations.

*TF.IDFi,j = TFi,j  X log(N/DFi)****Figure 6. The Formula for TF.IDF, where TFi,j is the frequency of word i in j, DFi is the occurrences of i in the document set, and N is the number of documents.***

We then use a Word2Vec model, which provides vectorised word-embeddings, these vectors embed high-dimensional information about the word being used based on the context it was used in; a word used in similar contexts should share semantic information. This means we can use standard vector operations to model similarity.

Therefore, we can perform a dot-product operation to retrieve the cosine similarity between two or more words, or more accurately, the concepts those words convey. We perform the following operation.

***Figure 7. The Average of the cosine similarity of the Word2Vec Vectors where n is the number of articles in the “liked” article set.***

### Feature 5: Topic Category Hot Vector.

Encoding the topic of the document into the neural network is also an important feature that must be considered.

The topic information is already encoded into the Article class via the repository. Therefore, all we need to do is encode this into a hot vector. A One-Hot vector encodes multiple choices as a 1xN bitmap vector, for instance:

|  |  |
| --- | --- |
| Business | [0,0,0,0,1] |
| Entertainment | [0,0,0,1,0] |
| Politics | [0,0,1,0,0] |
| Sport | [0,1,0,0,0] |
| Tech | [1,0,0,0,0] |

***Figure 8. How Topics are encoded into a hot-vector.***

This information is intended to help the neural network encode complicated relationships, and discern preference for multiple features.

### Feature 6: K-Means Cluster with ‘Part-of-Speech’ tags.

We additionally try to discern more complicated features, such as styles of writing, using an unsupervised clustering method. We use K-Means clustering to cluster based on the ‘Part-of-Speech’ tags present within the document.

To do so we first pre-process the documents, lemmatizing the contents to ensure a standardised language. From these Lemmatized features we encode them into an array of their part of speech tags. These are selected from the list below.

|  |  |  |
| --- | --- | --- |
| Number | Tag | Description |
| 1. | CC | Coordinating conjunction |
| 2. | CD | Cardinal number |
| 3. | DT | Determiner |
| 4. | EX | Existential *there* |
| 5. | FW | Foreign word |
| 6. | IN | Preposition or subordinating conjunction |
| 7. | JJ | Adjective |
| 8. | JJR | Adjective, comparative |
| 9. | JJS | Adjective, superlative |
| 10. | LS | List item marker |
| 11. | MD | Modal |
| 12. | NN | Noun, singular or mass |
| 13. | NNS | Noun, plural |
| 14. | NNP | Proper noun, singular |
| 15. | NNPS | Proper noun, plural |
| 16. | PDT | Predeterminer |
| 17. | POS | Possessive ending |
| 18. | PRP | Personal pronoun |
| 19. | PRP$ | Possessive pronoun |
| 20. | RB | Adverb |
| 21. | RBR | Adverb, comparative |
| 22. | RBS | Adverb, superlative |
| 23. | RP | Particle |
| 24. | SYM | Symbol |
| 25. | TO | *to* |
| 26. | UH | Interjection |
| 27. | VB | Verb, base form |
| 28. | VBD | Verb, past tense |
| 29. | VBG | Verb, gerund or present participle |
| 30. | VBN | Verb, past participle |
| 31. | VBP | Verb, non-3rd person singular present |
| 32. | VBZ | Verb, 3rd person singular present |
| 33. | WDT | Wh-determiner |
| 34. | WP | Wh-pronoun |
| 35. | WP$ | Possessive wh-pronoun |
| 36. | WRB | Wh-adverb |

***Figure 6. Pen Treebank project Part of Speech Tags (Anon, 2014)***

We vectorise these part of speech tags, and present them and perform a K-Means analysis on them using K=3. Our hopes here is that we are able to decipher, based on the number of part-of-speech tags, specific speech styles that may help us understand a user.

We posit the following hypothesis: A descriptive piece of writing should have more adjectives relative to other parts-of-speech, similarly personal or opinion pieces should contain more personal-pronouns. With this, we can separate and cluster styles of writing based on these assumptions.

K-Means clustering works on clustering information based on the sum of distances between a centroid and an individual point. Data points are assigned based on the min of the sum of distance.

s

***Figure 9. K-Means Formula, where K is the number of centroids and x is the number of features present in the dataset.***

Similarly to the Topics, we then encode this in a one-hot vector of 1-K, where K = 3.

## Neural Network Architecture

Once we have extracted the features from each article, we can use them as inputs for our neural network. We will proceed to discuss our neural networks architecture, it’s activation functions and the weight update function.

We then create an iterator out of our previously extracted features from our corpus. This means our first layer consists of 12 input layers. 1 layer for the length, normalized from 0-1, another layer for the sentiment score between 0-1, the flesch-readability score normalized from 0-1, the cosine similarity of word vectors, the hot vector of topic, which takes up 5 inputs, and the cluster for which the article belongs, which takes up 3. This information, along with a label indicating if the article is liked or disliked forms the training and evaluation data fed to the neural network. The evaluation data is a random sample of around 10% taken from the training data. We use this to assess the efficacy of the neural network.

We then needed to determine how many hidden layers and neurons were necessary. We used recommendations from Jeff Heaton, author of “Introduction to Neural Networks for Java” (Heaton, 2009). This recommendation stated that the number of hidden layers should be equal to one, and the number of neurons required in that layer should be the mean of the neurons in the input and output layer. In other words, we have 1 layers of hidden neurons with ((12+2)/2 = 7) neurons.

There are two output layers, one activating if the neural network believes that a person ‘likes’ this article (or has a ‘1’ label) and the other for not-liking (or a 0). It is important to note that “Not liking” an article does not necessarily mean disliking, for this we would have to model more output responses.

As such, our neural network has the following topology.



## Limitations

## Conclusions

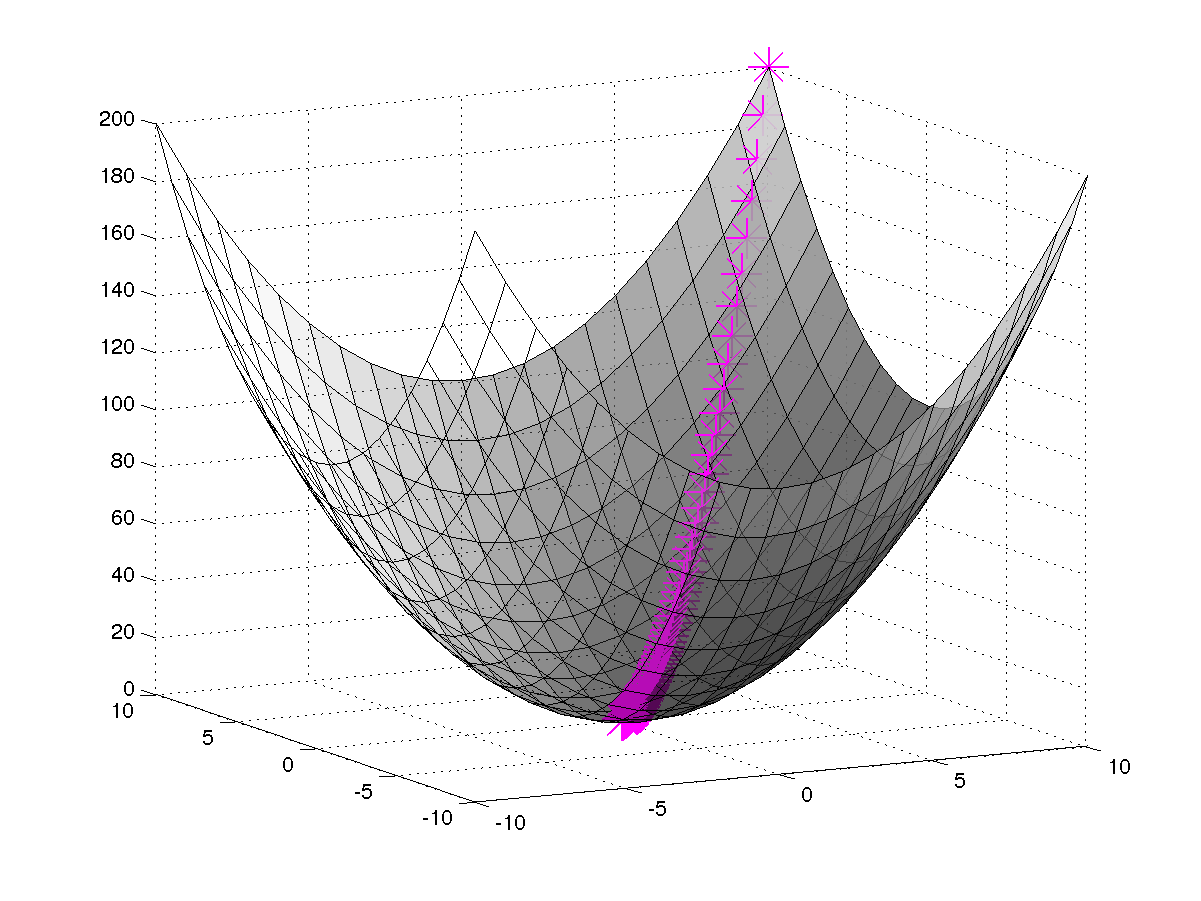
**Figure 10. Topology of our feed-forward neural network, where in is the input, hn is the hidden laeyer and 0n is the output.**

Our neural network uses a rectified linear unit, or “ReLu” for it’s activation function between the input layer and the hidden layers. The ReLu function computes the function f(x) = max(0,x), meaning that the function is thresholded at 0. We decided upon this activation function as it allows for greatly accelerated convergence of stochastic gradient descent over other activation functions (Krizhevsky et al, 2012) We also use a Gradient Descent approach for updating our weights. We decided upon Gradient Descent over Stochastic Gradient descent due to the desire for our application to be deterministic. For our example, of around 2000 training examples, this is acceptable. However, if we were to deploy this application for use in, say, a web-application we would have to be more cognizant of speed. In that case, a stochastic approach would provide better speed at the cost of accuracy.

Gradient Descent attempts to find a local minimum of an error function by iterating through the negative of a gradient. We repeat this for the number of epochs, or, alternatively until we have converged. We use the partial derivative of the error function to calculate the newly updated weights, and iterate on this until we reach a local optima. We thus update the weights as follows.

**Figure 11. Updating the weights using Gradient Descent.**

This allows us to iterate down a function and find a local optima, as the example shows below.



**Figure 12. An example of Gradient Descent (MatLabCentral, 2017)**

The activation function between the hidden layer and the output layer uses a softmax function. This follows standard practice for neural networks that deal with classification problems. The softmax calculation is used to determine the probability of example X belonging with class W. In our example we use this figure as an activation function, and so retrieve either a 1 if t belongs to the ‘liked’ class, or ‘0’ otherwise.  
 We tune the hyperparameters of our neural network to be the following. Learning Rate is 0.01, We run 12 iterations through a batch of 50 using 30 epochs.

# Findings and Analysis

## Introduction

This intent of this section is to discuss the results of the neural network and against the selected corpus. It will be structured as follows.

Each new sub-section of this section will consist of a brief explanation of the features being tested, the rationale for testing this feature, the results including the accuracy, precision, recall and F1 score.

The accuracy of the model is merely the percentage of results accurately predicted, if we predicted that the model accurately predicted 211/222 results correctly in the evaluation set we would say we have an accuracy of (211/222) = 0.9505.

The precision of a model is defined as the is the fraction of retrieved instances that are relevant. In the case where we have labelled class 1 as 1 33 times, labelled class 1 as 0 11 and class 0 as 0 178 times our precision score would be ((33 / (33+11)) / 2) + (178/(178+0))/2), or 87.5%.

The recall score is the percentage of the document that has been successfully retrieved. For instance, in the case where we have labelled class 1 as 1 33 times, labelled class 1 as 0 11 and class 0 as 0 178 times our precision score would be ((33 / (33+0)) / 2) + (178/(178+11))/2), or 97.09%

The F1 Score is the Harmonic average of the precision and recall scores. It provides us with a good idea of how precise, and sensitive a model is.

The Training data will use 80% of the corpus, where the evaluation data will use 20% of the corpus. This means that we have approximately 400 examples in our evaluation data. We randomize the inputs to ensure a good spread.

## Conclusions

# Discussion

## Introduction

## Issue 1

## Issue 2

## Conclusion

# Conclusions and Recommendations

## Conclusions

### Conclusion 1

### Conclusion 2

## Recommendations

### Recommendation 1

### Recommendation 2

# Bibliography

# Appendices

# Bibliograpghy

Ali. K and Stam. W (2004) TiVo: making show recommendations using a distributed collaborative filtering architecture*, Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining pp.394-401*

Collaborative Filtering Recommender Systems, Schafer, Frankowsi, Herlocker and Sen in *The adaptive web*. 1st ed. Berlin: Springer (2007).

Grčar, M., Mladenič, D., Fortuna, B. and Grobelnik, M. (2006). Data Sparsity Issues in the Collaborative Filtering Framework. *Advances in Web Mining and Web Usage Analysis*, pp.58-76.

IBM Alchemy. (2017). IBM: IBM.

Insight - BBC Datasets. (2006). [online] Mlg.ucd.ie. Available at: http://mlg.ucd.ie/datasets/bbc.html [Accessed 21 Feb. 2017].

Kent, Allen & University of Pittsburgh (1979). Use of library materials : the University of Pittsburgh study. M. Dekker, New York

Kim, J., Lee, K., Shaw, M., Chang, H., Nelson, M. and Easley, R. (2006). A preference scoring technique for personalized advertisements on Internet storefronts. *Mathematical and Computer Modelling*, 44(1-2), pp.3-15.

Kim, Y (2014) Convolutional Neural Networks for Sentence Classification.

Konstan, J., Miller, B., Maltz, D., Herlocker, J., Gordon, L. and Riedl, J. (1997). GroupLens: applying collaborative filtering to Usenet news. *Communications of the ACM*, 40(3), pp.77-87.

Lee. J, Sun. M and Lebanon. G (2012) *A Comparative Study of Collaborative Filtering Algorithms.*

Manning, Christopher D., Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J.

Bethard, and David McClosky. 2014. [The Stanford CoreNLP Natural Language Processing Toolkit](http://nlp.stanford.edu/pubs/StanfordCoreNlp2014.pdf) In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pp. 55-60.

McCallum. A and Nigam, K. (1998) A Comparison of Event Models for Naive Bayes Text Classification.  
  
Widrow. B, Lehr. M, Beaufays. F, Wan. E, Bilello. M Adaptive signal processing. (1986). *Signal Processing*, 10(2), p.208.  
  
Altman, E., Marco, G. and Varetto, F. (1994). Corporate distress diagnosis: Comparisons using linear discriminant analysis and neural networks (the Italian experience). *Journal of Banking & Finance*, 18(3), pp.505-529.

MENG, X. and CHEN, L. (2009). Collaborative filtering recommendation algorithm based on Bayesian theory. *Journal of Computer Applications*, 29(10), pp.2733-2735.

Meyer, R. (2016). *How Many Stories Do Newspapers Publish Per Day?*. [online] The Atlantic. Available at: https://www.theatlantic.com/technology/archive/2016/05/how-many-stories-do-newspapers-publish-per-day/483845/ [Accessed 21 Feb. 2017].

Mikolov. T, Sutsekever, I, Chen. K, Corrado. G, and Dean. (2013) Distributed representations of words and phrases and their compositionality*, Advances in neural information processing systems, pp.3111-3119*

Oord. A, Dieleman. S and Schrauwen. B (2013) Deep content-based music recommendation,

Pazzani, M. and Billsus, D. (1997). *Machine Learning*, 27(3), pp.313-331. *Proceedings of the 26th International Conference on Neural Information Processing Systems pp*.2643-2651.

Quinlan, J. (1986). Induction of decision trees. *Machine Learning*, 1(1), pp.81-106.

Ricci, F. (2011). *Recommender Systems handbook*. 1st ed. New York [etc.]: Springer.

Sall. M, The Optimal Post is 7 Minutes – Data Lab. (2013). [online] Medium. Available at: https://medium.com/data-lab/the-optimal-post-is-7-minutes-74b9f41509b#.bmdjwb77u [Accessed 21 Feb. 2017].

Stanford – Penn treebank project. (2014). [online] Available at: https://www.ling.upenn.edu/courses/Fall\_2003/ling001/penn\_treebank\_pos.html [Accessed 21 Feb. 2017].

Toffler, A. (1971). *Future shock*. 1st ed. New York: Bantam Books.

Heaton, J. (2009). *Introduction to neural networks with Java*. 1st ed. Chesterfield (MO, USA): Heaton Research.

Krizhevsky. A, Sutskever. I and Hinton. G (2012). *Imagenet classification with deep convolutional neural network. Advances in Neural Information Processing Systems Pg. 2012*

Matlab Central (2017). Gradient Descent Visualization - File Exchange - MATLAB Central. [online] Uk.mathworks.com. Available at: https://uk.mathworks.com/matlabcentral/fileexchange/35389-gradient-descent-visualization?requestedDomain=www.mathworks.com [Accessed 6 Mar. 2017].