Web Embedded Topic Modelling System

A Dissertation
Presented to
The Academic Faculty

By

Ahmed Elmi

In Partial Fulfilment
Of the Requirements for the Degree
MSc Computer Science

Swansea University
Prifysgol Abertawe

Swansea University
Department of Computer Science
Swansea, Wales

2019

ABSTRACT

Qualitative data analysing methods aren't nearly as available as quantitative ones. A great deal of knowledge can be gained by studying the hidden heuristic structures in text. It has always been the case that to gain knowledge from a book, one must read it. What if there was a way to gain that knowledge without having to read it? Topic Modelling is an automated qualitative analysis of large documents to abstractly identify key topics. Many professions could benefit from such a tool for example psychologists could analyse the transcripts of schizophrenic patients and discover a pattern of thought. Others include sociologists and journalists who need to analyse many documents and articles. Topic modelling isn't a readily accessible tool. In this project I explore and implement a way to increase the availability of topic modelling to the larger community.

ACKNOWLEDGEMENTS

Firstly, I would like to thank my supervisor Dr Matthew Roach for his support and guidance throughout this project. The advice, feedback and guidance I have received greatly helped complete this project.

I would also like to thank my second marker Mr Stewart Powell for his feedback to my project proposal document.

I would like to thank my parents for keeping me on the right track and for the continuous support they have given me.

Also, thank you to my friends who remind me that we all need to take a break sometimes.

Finally, I would like to thank anybody who has participated in this project whether it was just testing/reviewing or simply giving guidance at any stage.

Project Outline

Introduction

Here I will give a brief introduction on what my project is about and the key points that I will discuss. I will state the targets that I wish to reach and my motivation for this project.

Related Work

In depth analysis of work that others have done to contribute to the field. Compare and contrast to my project.

Approach

This section completely describes the steps that I have taken to implement the project and my reasons for the methods I have taken.

Testing

Here we shall discuss the different methods that I have used to test the quality of my approach.

Project Management

Here I explain the way that I have structured my project and why I did things the way I did.

Evaluation

Looking back at the approach; what thing went well? What things didn't?

Conclusion

A summary of the project, the things I could have done differently and any future developments that could be made.

TABLE OF CONTENTS

Abstract ii

ACKNOWLEDGEMENTS iii

LIST OF FIGURES vi

CHAPTER 1. Introduction 1
1.1 Scenario 1
1.2 Project Aims/Specification 2

CHAPTER 2. Related Work 3
2.1 Latent Semantic Analysis (LSA) 3
2.2 Latent Dirichlet Allocation (LDA) 5
2.3 LDA2Vec 7
2.3.1 Inner Mathematics 9

CHAPTER 3. Approach 10
3.1 Back End 10
3.1.1 Pipelining 10
3.1.2 Algorithm 11
3.2 Front End 12
3.2.1 pyLDAvis 12
3.2.2 Web Development 13

CHAPTER 4. Testing 18
4.1 Unit Testing 18

CHAPTER 5. Project Management 20
5.1 Software Lifecycle Method 20
5.2 Schedule 21
5.3 Risk Assessment 22
5.4 Coding Convention 23

CHAPTER 6. Evaluation 24
6.1 Limitations 24
6.2 Overview 25

CHAPTER 7. Conclusion 25
7.1 Project Outline 25
7.2 Further Development 26

Bibliography 28

LIST OF FIGURES

Figure 1 – Document vs term Matrix (Joshi, 2018) 4

Figure 2 – Singular Value Decomposition (Joshi, 2018) 4

Figure 3 – Word prediction using pivot word (Introducing our Hybrid lda2vec Algorithm, 2016) 7

Figure 4 – Corpus Bag of Words vs Skip-gram (Mikolov, Chen, Corrado, & Dean, 2013) 8

Figure 5 – pyLDAvis (Qi, 2018) 12

Figure 6 – Implemented File Upload ModelForm. 14

Figure 7 – Uploaded File List with Download and Delete buttons. 16

Figure 8 – Table of results from the questionnaire 19

Figure 9 – Waterfall SDLC (Sami, 2012).....20

Figure 10 – Gantt Chart 21

Figure 11 – Risk assesment table 22

CHAPTER 1. INTRODUCTION

In this day and age, the growth of data increases exponentially. The conventional methods of information extraction from data are being depreciated. Nevertheless, as technology advances; so do the strategies of reconnoitring massive amounts of data. Topic modelling is an example of a machine learning technique that discovers concealed themes and structures within large text datasets.. In this project I will show how data can be mined to extract information in a more efficient way.

1.1 Scenario

A bookstore owner needs to sort a large number of books by genre. The genre is not so easily identified on the books. It would not be efficient to read the books and manually sort them. Running a topic model on an electronic collection of the books could automatically identify the genre of the books.

I intend to create a web embedded topic modelling system so that it can be more accessible to people who may require it.

Topic modelling could be majorly beneficial to scientists that are researching DNA genome sequencing. Performing topic modelling on a set of documents that describes genetic disorders and the DNA patterns observed could allow a better insight into how genetic disorders work and their interactions with each other.

1.2 Project Aims/Specification

By identifying a set of aims that this project should meet, I can ensure that my work stays honest and that there is a clear working order. This will make it easier for me to track progress and separate the main goal of the project into smaller sub targets. The specifications that this project should achieve are:

- Files can be uploaded
- Files can be deleted from the server
- Files can be downloaded from the server
- Topic modelling algorithm runs on uploaded files
- User has the ability to choose the number of topics the algorithm should model

CHAPTER 2. RELATED WORK

Every topic model follows the same understanding which is that:

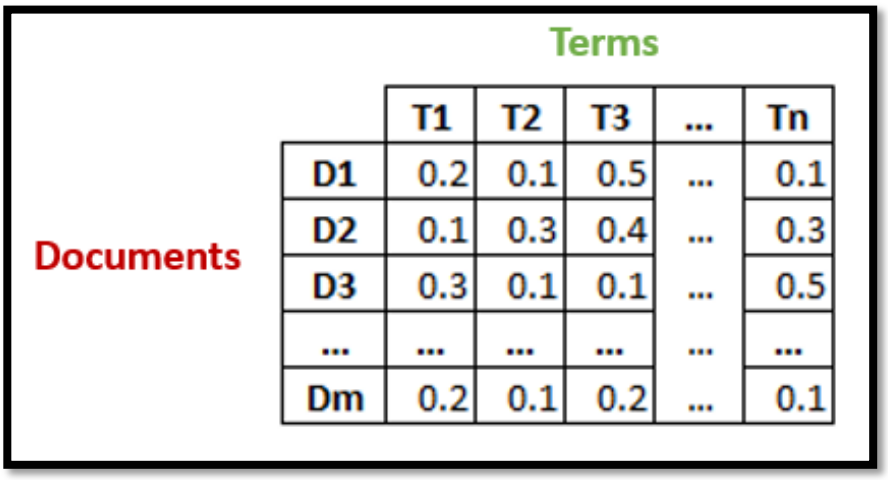
- All documents are a collection of topics.
- All topics are a collection of words.

In topic modelling there are many algorithms that can be used to explore the data. In this chapter I will go over a few of these algorithms and explain how each one works.

2.1 Latent Semantic Analysis (LSA)

Latent Semantic Analysis is one of the most foundational techniques in topic modelling. LSA assumes that the latent topic of a word is based upon its surrounding words.

LSA starts by creating a large rectangular matrix comprised of documents and distinct words. If there are m number of text documents and n number of distinct terms/words then a matrix of m x n will be created to compute k number of topics which is specified beforehand. The matrix is filled with term frequency-inverse document frequency (tf-idf) values. A tf-idf value is a statistic that describes a words weighting within a document (Anand Rajaraman, 2011).



	T1	T2	T3	...	Tn
D1	0.2	0.1	0.5	...	0.1
D2	0.1	0.3	0.4	...	0.3
D3	0.3	0.1	0.1	...	0.5
...	...	...	...	...	...
Dm	0.2	0.1	0.2	...	0.1

Figure 1 – Document vs term Matrix (Joshi, 2018)

Naturally, the document-term matrix contains a lot of noise and redundancy therefore a mathematical concept called Singular Value Decomposition (SVD) is used to reduce the dimensionality of the matrix to k dimensions (Deerwester, 1990).

$$\mathbf{A = USV^T}$$

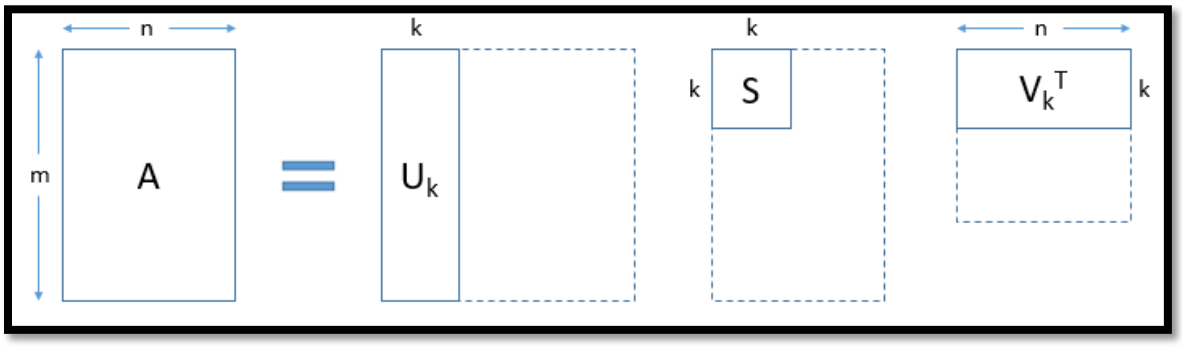


Figure 2 – Singular Value Decomposition (Joshi, 2018)

The above formula shows how SVD breaks down a large matrix A into 3 smaller matrices U, S and VT. In matrix U, the rows represent the document vectors in relation to topics; the rows in matrix V are term vectors in relation to topics. These vectors give us the opportunity to apply a cosine similarity calculation to measure (Mihalcea, Corley, & Strapparava, 2006):

- How similar different document are to each other.
- How similar different words are to each other.
- How similar terms and documents are to each other.

This information means one can analyse the hidden latent patterns within a document and cluster all the words into k number of topics. For example two vectors with an angle difference of 0° will have a cosine similarity of 1 because cos(0°) = 1.

2.2 Latent Dirichlet Allocation (LDA)

LDA is a more probabilistic method compared to that of LSA. It uses two distribution matrices; a word-topic matrix filled with the probability of a word being selected within a specific topic and a topic-document matrix filled with the probability of a specific topic belonging to a specific document. LDA is generally known as a "distribution over distributions" algorithm (Xu, 2018).

Initially when the collection of documents is parsed, each word is randomly assigned to one of the k topics. Through every iteration, each word is reanalysed and reconfigured to

suit a better topic cluster based on the probability that topic t generated word w. After many iterations, a steady state of word to topic allocation will be reached.

LDA uses two statistical concepts which make it a Bayesian version of probabilistic LSA. These are dirichlet distribution and the multinomial distribution.

The dirichlet distribution takes in a parameter α which is a number between 0 and 1. This value controls the variance from the mean; in context of topic modelling, it controls how easy it is for a word to belong to a mixed set of topics. The closer α (the normalisation constant) gets to 0, the stronger the variance within the distribution. Words will be more inclined to be part of only one topic. With a higher α value, the general variance of words to topics is low therefore words will be more inclined to belong to more topics (Geiger & Heckerman, 2011).

Multinomial distribution calculates the probability of certain events happening that happening that have more than two possible outcomes. Otherwise it would be classified as binomial distribution. In topic modelling, we use multinomial distribution to predict which topics a word belongs to. LDA assumes that a document is made up of a set of topics and that every word has been chosen to belong to these topics. Therefore working backwards, multinomial distribution calculates the probability that a word was generated from a specific topic based on the other words currently in that specific topic. As the number of iterations increases the multinomial distribution fine tunes the precision of the topics and the accuracy of word allocation is observed (Blei & Lafferty, 2006).

2.3 LDA2Vec

LDA2Vec is an evolved data mining method that combines LDA with a deep learning tool called word2vec. It is considered a much more complex natural language processing method because unlike the other algorithms, LDA2Vec can make sense of meaning from text.

Word2vec is a technique used to create word embedding by mapping words to vectors in a vector space using neural networks. It was created by a team of researchers at Google led by Tomas Mikolov (Levy & Goldberg, 2014). The fascinating feature about this vector space is word vectors that share a similar context meaning are positioned closer together. Word2vec follows the assumption that a word is characterised by the company it keeps i.e. its surrounding words.

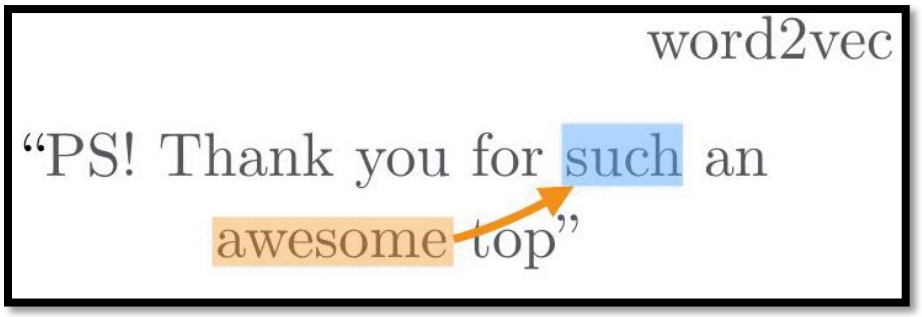


Figure 3 – Word prediction using pivot word (Introducing our Hybrid lda2vec Algorithm, 2016)

Word2vec selects a word in the text (the pivot word) and parses through all the words that occur around the pivot word. These words become context words. Each context word forms a word-context pair with the pivot word. There are two architectures of word2vec; the continuous bag of words (CBOW) and the skip-gram architecture. In the CBOW

model, the pivot word is determined given the surrounding context words. In contrast, the skip-gram model predicts the surrounding context words given the pivot word.

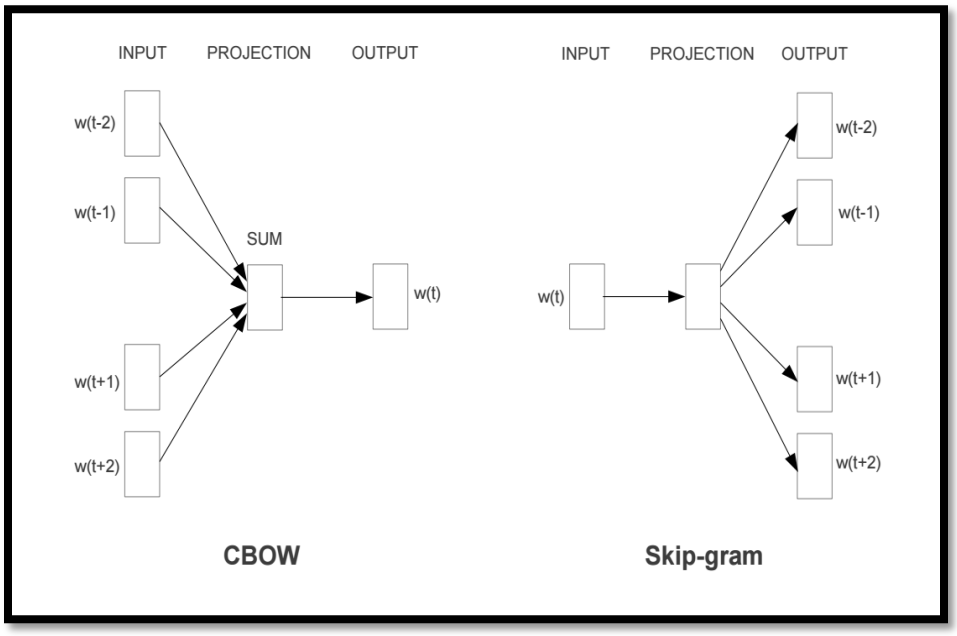


Figure 4 – Continuous Bag of Words vs Skip-gram (Mikolov, Chen, Corrado, & Dean, 2013)

Since the word embedding produce fixed word vectors, it is possible to do vector arithmetic. For example:

King – Man + Woman = Queen

Completing the above calculation will lead from the King word vector to the Queen word vector.

In LDA2Vec, the skip-gram model is used with the pivot word vector and a document vector (sum of all the word vectors in the document) to produce a context vector. This context vector allows LDA2Vec to produce more interpretable topic allocations.

2.3.1 Inner Mathematics

Words that appear in the same topics must share a semantic meaning. In the vector space these words would be embedded closer together. LDA2Vec combines this knowledge with the capabilities of topic modelling whether it is a count-based approach such as LSA or a predictive approach such as LDA. LDA2Vec can be train and optimised using the maximum likelihood method. This is a procedure that estimates a parameter based on given observations. In the case of predictive topic modelling algorithms it is desirable to maximise the probability that an estimated word belongs to a chosen topic.

Using a softmax function we can map a categorical distribution of words over K (topics) different outcomes with respect to its probability belonging.

$$P(w_t|h) = \text{softmax}(\text{score}(w_t, h)) = \frac{\exp\{\text{score}(w_t, h)\}}{\sum_{\text{Word w' ir Vocab}} \exp\{\text{score}(w', h)\}}$$

The above equation calculates the probability of a target word w_t being compatible with a context h and recursively compares it with the compatibility that previous words have. (Mikolov, Sutskever, Chen, Corrado, & Dean, 2013)

CHAPTER 3. APPROACH

3.1 Back End

3.1.1 *Pipelining*

Raw text data cannot just be injected into a topic modelling algorithm. It must first be transformed. The documents are processed by trimming off certain words such as pronouns as they don't affect the allocation of a topic. Such words are called 'stop-words'. A list of well determined stop words can be used to automatically remove them from a document set. One such dictionary can be found with the natural language processing toolkit library of python. Other strings can be added to the dictionary such as punctuation and words with a character count of less than 3.

Stemming and lemmatisation are natural language processing methods in which words are reduced to their base form. For example, 'troubling' would be reduced to 'troubl' and 'running' would be lemmatised to 'run'. The difference between stemming and lemmatisation is that when stemming the root word (stem) may not actually be a real word whereas with lemmatisation the root word (lemma) is a real word.

Each unique word carries significance in assigning a topic therefore each word must be treated as its own entity. This is known as tokenisation. The tokens will be the input for the topic modelling algorithm.

3.1.2 Algorithm

As soon as the data has been pre-processed, the tokens can be fed into the algorithm. The implemented algorithm that I have chosen use is the Latent Dirichlet Allocation algorithm because it is typically the most effective algorithm and has a large community backing. Alternative algorithms such as LSA call for a massive document set to guarantee accurate results. Moreover, the issue regarding polysemy (different meanings of a word) is best handled with LDA.

Initially, I did the coding for the back end in a Jupyter Notebook. I wanted to keep the back end code separate from the front end code so that I could make specific optimisations easier. Within the Jupyter Notebook I could manually upload files into a list which I used to test if my LDA algorithm was working. The LDA model would only accept files of specific types such as .txt or .csv. I made sure that there was a system in place that would validate whether the uploaded files were of the accepted type before running the topic model. In the developing stage I could manually enter the number of topics that I desired for the model to run however this would be much more difficult for a general user to do therefore I created a variable ‘ntopics’ which would store an integer value from user input into a form which I shall further discus in the front end implementation.

Furthermore, since the topic modelling algorithm is intended to run on files that the user uploads; it would retrieve files from a path location where these files are stored on the server. As I will be locally hosting the web application, the files will be stored on my

laptop. The files are retrieved from a specified path and validated before being read into a list ready to be modelled.

3.2 Front End

3.2.1 pyLDAvis

The conventional way of displaying the results of topic modelling is an array that contains bags of words and each bag being an abstract topic. In essence, there is a two dimensional array and each inner array is an abstract topic. Within these topics are a list of words accompanied by a decimal value which represents that words weighting within that topic. The implemented LDA model produced an output of the array format. I believed this to less interpretable especially for non-technical users. PyLDAvis is a python library that offers a more interactive output visualisation for topic models. This will greatly help with the usability and representation of the abstract topics that are modelled. Using pyLDAvis, I can directly save the output as a HTML file which will then be rendered and returned to the user when the algorithm is complete.

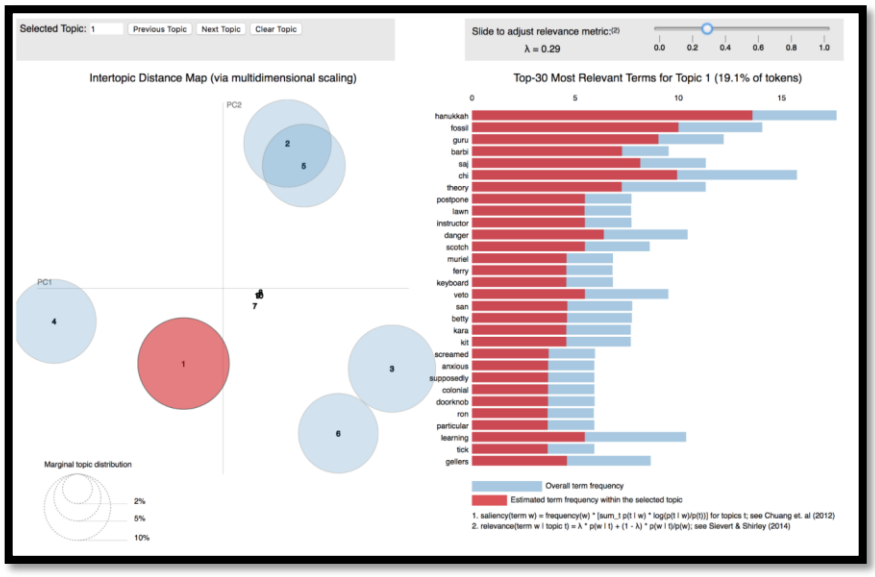


Figure 5 – pyLDAvis (Qi, 2018)

3.2.2 Web Development

3.2.2.1 Getting Started

I've decided to use Django as my web development framework due to its MVT (model, view and template) structure which I am familiar with. Due to the fact that the back end is coded in python it aligned well to use a python based web framework to allow compatibility and ease of integration. The methodology of a Django web application is that when the user sends a request a URLs system directs the request to a view for processing. Views behave as the logic that determines what input is received and what output is returned. For example when a button is clicked; the request is directed to a view which then returns a html page. In my case an appropriate scenario would be when I run the topic modelling algorithm, the pyLDAvis html page would be returned taking into consideration a user specified number of topics. Django projects are designed in a very specific way and there isn't much deviating from that. When starting a project all compulsory components are preconfigured and one is presented with a project skeleton.

django-admin startproject 'mysite'

Entering the above command in the command prompt where 'mysite' = project name, creates the project skeleton which I can then add to. Another command that I repetitively needed to use was the command to locally run the website so that I could see how it looked in production mode. This command is:

python manage.py runserver

3.2.2.2 File Management

The first thing that I needed to do was create some sort of file storage system. The types of files that are stored fall under two categories; static and media. Static files are the files that wouldn't change in production. Files such as the ones that control the visual representation of the web page e.g. HTML, JavaScript and CSS files. Media files are the user-uploaded content such as the files the user would upload to run the topic modelling on. In my settings.py file I instantiated a MEDIA_ROOT and STATIC_ROOT which controlled where the files of the respective type would be stored.

In my models.py file I needed to define an object of file type. Doing this allowed me to draw a schema in the database for the files that will be uploaded by the user. For user to be able t upload their files I created a ModelForm which essentially uses the pre-existing model to create a form. The ModelForm only takes a file input through a FileField and processes it through the model schema which then gets stored in the MEDIA_ROOT.

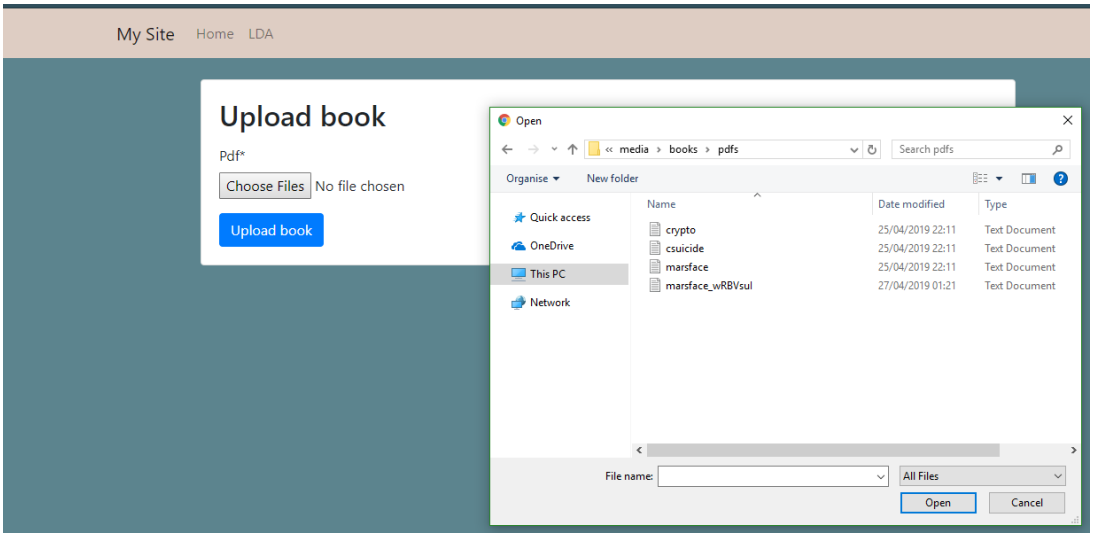


Figure 6 – Implemented File Upload ModelForm.

When I first created the file upload feature, it would only upload one file at a time. This would quickly become an inconvenience for users who intend to upload a large quantity of files. It quickly became apparent that I needed to allow multiple file upload. This proved to be one of my harder tasks of the project. Due to the request/response interaction that Django uses between browser and server; it wasn't as straightforward as looping through all the files the user uploads and storing each one individually. The ModelForm may have allowed multiple files to be submitted however submitted files were stored as TemporaryUploadedFile type and there were many restrictions that were in place for security purposes. If there weren't it would create vulnerabilities where an attacker could upload a malicious file to breach the static files that a user should not have access to. The TemporaryUploadedFiles were stored in a REQUEST.FILES[] list. My solution was to instantiate each TemporaryUploadedFile as a File model and manually pass it through the ModelForm resulting in it being written to the MEDIA_ROOT.

I implemented the functionality to allow the user to delete files that they have uploaded. The way I've done this is when the user clicks the delete button next to the specific file; its primary key is sent to the view that handles this process. It searches through the model schema and locates the file with the matching primary key, removes it and updates the database. I've also created a 'download' button where the user can download and view the uploaded file by traversing to its URL location.

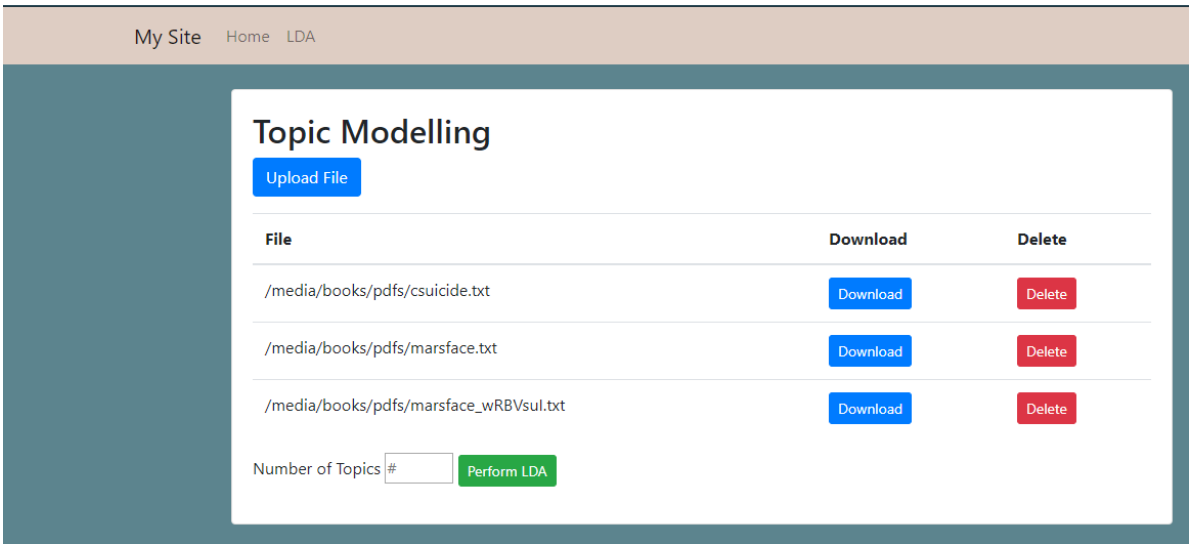


Figure 7 – Uploaded File List with Download and Delete buttons.

3.2.2.3 Connecting Everything Together

Once it completely implemented to efficiently control uploaded files, it was time to link the topic modelling algorithm to the MEDIA_ROOT to perform LDA on all the files in the directory. I transferred the LDA code from the Jupyter Notebook and into tp_lda.py located within the project folder. I created tp_lda.py to be an extension of views.py solely to handle the functionality of performing the topic modelling. Using a REQUEST.POST signal from the file list page, a user could define the number of topics they desired the LDA to take as a parameter. Once the algorithm was complete its pyLDAvis file is locally stored. Naturally, a view would return a predefined html template as a response. However, in this case the pyLDAvis file is created and returned within the same view.

3.2.2.4 Improved Algorithm

Once I completed and was satisfied with the LDAs performance on the user uploaded files, I decided to start working on an LDA2VEC algorithm which would make the topic model superior in quality due to LDA2VECs word embedding characteristics. My first goal was to successfully be able to compute vector mathematics of words within a vector space as mentioned in the related work chapter. I successfully managed to produce working word embedding's where I could test my vector mathematics.

I defined a function 'most_similar' which takes a word as input and returns the top 20 most similar words by discovering the words with the closest vector values. Of course, the vector space is a vastly large set of data with a high dimensionality therefore the data is dimensionally reduced and normalised to only use the defining features to find these words.

I created another function 'vecmath' which calculates the cosine similarity of two words based on their word vectors. This function allows me to test my vector maths and returns the top 20 results with the closest vector sum value. Example vector math returned from this function: California + Technology =

[silicon valley', 'in', 'new york', 'u.s.', 'west', 'tech', 'usa', 'san francisco', 'japan', 'america', 'dc', 'industry', 'canada', 'new york city', 'nyc', 'area', 'valley', 'china']

Unfortunately, I couldn't finish the LDA2VEC implementation as time was a resource I didn't have much left of and it proved quite difficult combining the word embedding to the already implemented LDA.

CHAPTER 4. TESTING

4.1 Unit Testing

Unit testing is a procedure in software testing where individual components of whole software are tested separately to examine if they behave expectedly. The main units in my implementation as outlined include:

- Pipelining
- Algorithm
- Web App

By testing each unit I could establish confidence in my implementation to produce the rightful outcome. To test the pipelining unit, I gathered a set of unclean documents and manually cleaned them (removing stop words, stemming and lemmatising). I ran the unclean versions through the pipeline unit and compared the output with the manually cleaned versions. Of the 5 documents that I tested there was an error count of 0 resulting in a 100% pass in the pipeline test.

Testing the algorithm unit was less straightforward because the LDA algorithm produces abstract topics. I decided for this test I needed human participants. I ran the algorithm 5 times on 5 different sets of documents with a desired number of topics set to 5. I gathered 10 users to fill out a questionnaire where they were asked if they could identify each topic based on the highest frequency words and what was their level of confidence in their answers on a scale of 1-5. If they could not identify a topic was it because they couldn't

name it or because there was no pattern that led to a topic. I documented the results in the table below.

LDA SET	Avg Topics Identified / 5	Avg Confidence / 5
1	5	4.8
2	4	4.3
3	5	4.5
4	5	4.5
5	5	4.7

Figure 8 – Table of results from the questionnaire

I count the unit test of the algorithm a success because on average 96% of the topics were identified with an average confidence rating of 4.56/5.

To test the web application I simply compared its functionality against the first 3 specifications that were stated in project aims. Could files uploaded, deleted and downloaded. The web app proved successful against each requirement.

CHAPTER 5. PROJECT MANAGEMENT

In this chapter I will talk about my chosen software development methodology (SDM), why I chose it. Furthermore, I will reflect upon my timeline by comparing and contrasting with my original aims and milestones.

5.1 Software Lifecycle Method

Choosing a software development life cycle enforced a plan of action to aid in the software being realised and established from the project aims/specification to a completed product. I have chosen to follow a waterfall model methodology because the project aims were firmly established at the start and verification of each stage provided early detection of errors. The waterfall model provides a structured approach of design, implementation and testing with each stage having to be completed before being able to move to the next.

Another software development life cycle model that I contemplated using was the V-Shaped model. In comparison to the waterfall model, it too has specific deliverables in each phase. I chose to stay with the waterfall model because the V-Shaped model was more costly, required more time and the path is not clear if errors occur in the testing phases.

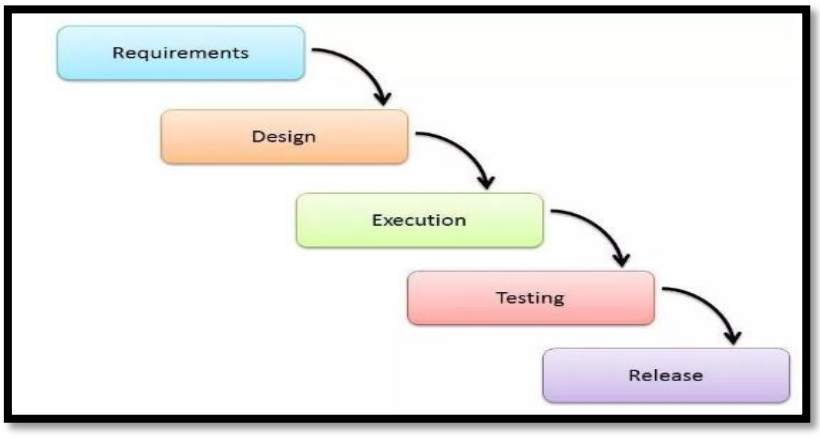


Figure 9 – Waterfall SDLC (Sami, 2012)

5.2 Schedule

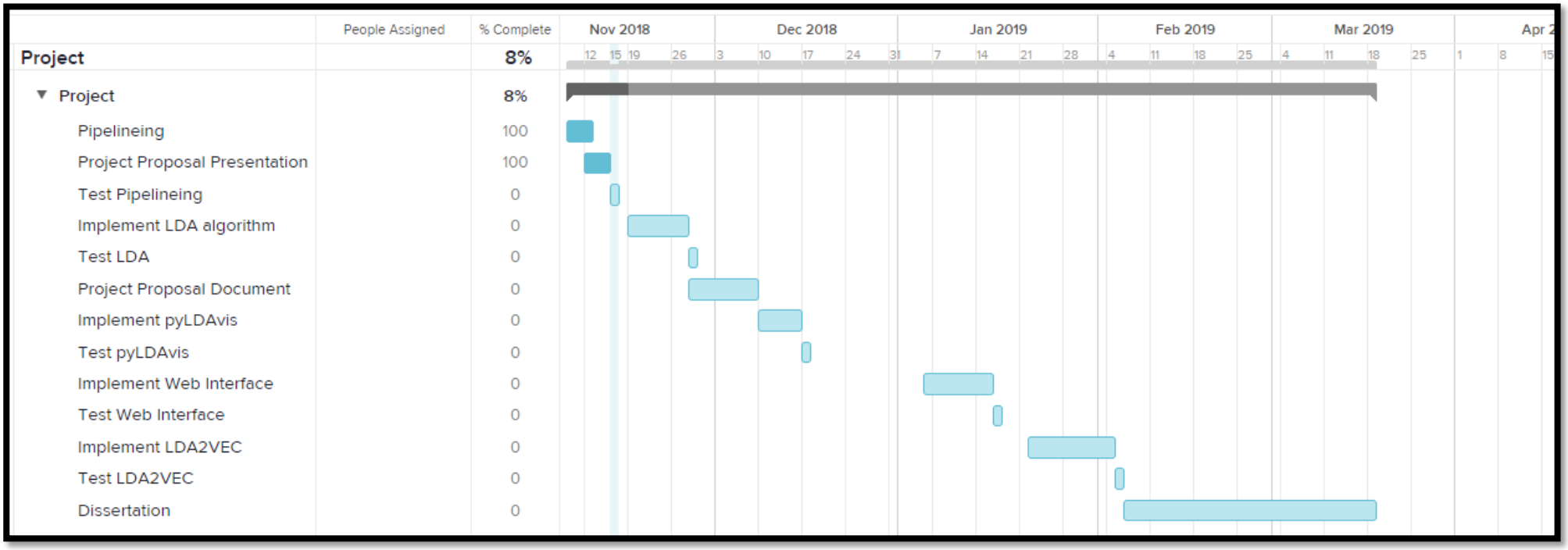


Figure 10 – Gantt Chart

In retrospect, following the Gantt chart that I created at the start really facilitated me to manage my time accordingly and provide order specifically to the implementation. Looking at the work that I currently have, the things that I would change would be to have given myself more time for the LDA2VEC implantation as it took much longer than initially thought.

5.3 Risk Assessment

Risk	**Severity / 10**	**Mitigation**
Loss of data	10	Backup Files / Version control such as Git
Malicious security attack	8	Use anti CSRF token in every form
Incomplete Project	7	Follow schedule and manage time accordingly

Figure 11 – Risk assesment table

5.4 Coding Convention

To standardize my code I used Bob's concise coding conventions which I was taught throughout my university years. The main rules to follow are:

Rule 1: Functions

Defined functions cannot exceed a line count of 75

• They should visible on one page.

• One can view the whole function without having to scroll the page. The only exceptions being main methods and functions that make use of switch statements.

Rule 2: Indentation

Functions cannot exceed an indentation level of 5. There are no exceptions.

Rule 3: Single Line Code length

A single line of code must not go above a character count of 80. There is non need for horizontal scrolling. There are no exceptions.

(Laramee, 2017)

CHAPTER 6. EVALUATION

6.1 Limitations

Topic modelling being a data mining strategy for big data it requires a lot of computation power. For LDA the time and space complexity O(n) is linear to the size of the data and the number of topics. With large sizes of collective documents I found that my laptop ran really slowly and it would take a considerable amount of time to complete the topic model. My laptop has an Intel Core i5-5200U CPU @ 2.20GHz 2 Core(s). Since, I am locally hosting the web application the response time to execute the LDA is dependent on the specs of my laptop. If hosted on a proper server with better processing power, the return speed would be much faster and would better satisfy the user.

LDA2Vec is a fairly new and modern concept; most of the resources that I could find were simply theoretical, any practical advances had been depreciated. This slowed my progression in the implementation of LDA2Vec.

During the unit testing of the LDA algorithm, I worried that the wording of the questions in the questionnaire could induce a bias in the participants. That they may actively be over searching for a topic to when it isn't so clear. However, I was reassured that it wasn't about topic they assigned but if they were confident in saying that there was a topic present.

All in all, I am glad that LDA was implemented and everything integrated well to create an end product that I can present.

6.2 Overview

More specifically, I believe that the quality of the inner functionality of the project equally depends on how the raw data is pre-processed and fed to the algorithm. The better non – topic defining words are removed from the raw data, the more accurate the topic assignments would be. I used a dictionary containing stop words from the natural language toolkit module. Improvements to this dictionary could result in better document cleaning.

As I had never previously used the Django framework before, there was a learning curve before I could really get started on the front end of the project. It's well known that Django takes a while to learn, but when you do then it is very quick to use.

CHAPTER 7. CONCLUSION

7.1 Project Outline

In summary, the project end goal was to create a topic modelling system that could be executed on the web increasing its accessibility to the wider community. Firstly, we stated the project aims and set the specifications to work towards. Following the waterfall methodology the next stage was design. I knew that I was making a web application thus I separated my workload into the front end and the back end. The front end dealt with

display, data inputs and pyLDAvis whereas the back end was responsible for the pre-processing, topic modelling algorithm and data storage/validation. Once front end and back end were individually complete it was time to integrate them as a whole product. By using HTTP requests I directed inputs and outputs to their necessary locations. User uploaded files were stored in a MEDIA.ROOT directory which was structured by a model schema. I enabled a multiple file upload feature which greatly improved efficiency for users. Users had permission to download and delete their files from the server. After the user uploaded the set of documents that they wanted to run LDA on, they could select the number of desired topics which would then be modelled by the algorithm. On completion of the topic model the user would be redirected to the pyLDAvis graphical representation of the results where they can interact and view topic details.

I began the implementation of LDA2Vec and successfully completed a vector space where vector mathematics could be carried on word embedding's. The next stages would have been to equip LDA with the ability to recognise word similarities and use that data to better assign topics.

7.2 Further Development

Currently, the only accepted file formats are .csv and .txt which greatly limit the standardisation of the project. There are so many different file formats that users may want to upload such as a pdf file. A possible way to overcome this drawback would be to implement a data stream where the text of the uploaded files are simply streamed into the topic modelling algorithm.

Another future development proposal is to allow session based file upload. To do this there would need to be a user authentication process such as a login system. when users log in they can upload there files to a specific subfolder within the MEDIA.ROOT which is dedicated to the user. This would definitely need a lot of computational power to uphold due to multiple users being able to run their topic models at the same time.

Since the process of performing LDA is potentially slow, a notification system could be created to let the user know when their model is complete. An alternative would be to send the pyLDAvis html file to the users' email address.

Bibliography

Introducing our Hybrid lda2vec Algorithm. (2016, May 27). Retrieved from MultiThreaded: https://multithreaded.stitchfix.com/blog/2016/05/27/lda2vec/

Anand Rajaraman, J. D. (2011). *Mining of Massive Datasets.* California: Cambridge University Press.

Blei, D. M., & Lafferty, J. D. (2006). *Dynamic topic models.* New York: Association for Computing Machinery.

Deerwester, S. (1990). *Indexing by latent semantic analysis.* Bell Communications Research.

Geiger, D., & Heckerman, D. (2011). *A characterization of the Dirichlet.* Institute of Mathematical Statistics.

Joshi, P. (2018, October 1). *Text Mining 101: A Stepwise Introduction to Topic Modeling using Latent Semantic Analysis (using Python)*. Retrieved from Analytics Vidhya: https://www.analyticsvidhya.com/blog/2018/10/stepwise-guide-topic-modeling-latent-semantic-analysis/

Laramee, R. S. (2017). *Bob's Concise Coding Conventions.* Retrieved May 2, 2018, from CS.Swansea: http://cs.swan.ac.uk/~csbob/teaching/laramee10codeConventionSlides.pdf

Levy, O., & Goldberg, Y. (2014). *word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method.* Cornell.

Mihalcea, R., Corley, C., & Strapparava, C. (2006). *Corpus-based and knowledge-based measures of text semantic similarity.* Boston: AAAI Press.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). *Efficient Estimation of Word Representations in.* Proceedings of Workshop at ICLR.

Qi, X. (2018, June 3). *LDA Topic Modeling and pyLDAvis Visualization*. Retrieved from medium: https://medium.com/@sherryqixuan/topic-modeling-and-pyldavis-visualization-86a543e21f58

Sami, M. (2012, March 15). *Software Development Life Cycle Models and Methodologies*. Retrieved March 23, 2019, from Melsatar: https://melsatar.blog/2012/03/15/software-development-life-cycle-models-and-methodologies/

Xu, J. (2018). *Topic Modeling with LSA, PLSA, LDA & lda2Vec.* medium.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., & Dean, J. (2013). *Distributed Representations of Words and Phrases.* Advances in neural information processing systems.