BSc (Hons) Software Engineering

Final Year Project Report

2016/2017

Assessing Gender Bias using Natural Language Processing techniques

Abdul Haseeb Hussain

Apostolos Antonacopoulos

@00368091

Abstract

The purpose of the project is the detection of Gender Bias using Natural Language Processing techniques. Given that the project is research orientated there is no necessity on the need to provide a user interface rather the project should prove that it is possible to detect Gender Bias using Natural Language Processing techniques.

The project has successfully been able to prove that it is possible to detect Gender Bias using Natural Language Processing techniques, given that no previous public research has been conducted within this area, the project lays out much of the groundwork in terms of development for future researchers to extend especially given that the project within its present form could potentially lead to many false positives unless weaknesses are addressed within future extensions. Such extensions could include support for gender classification of non-named entities within text, sentiment analysis weightings and sentiment analysis based on the objects and subjects of given text sentences.

The project has contributed positively to the field of Natural Language Processing and lay the foundations for future research upon the automated detection of Gender Bias using Natural Language Processing techniques. The undertaking of the project has allowed me to build a considerable knowledge base within the field of Natural Language Processing given the amount of research I was able to undertake and the application of such research.

Acknowledgements

I would like to give thanks to and acknowledge the following people who supported me throughout the project:

- Apostolos Antonacopoulos for his advice and supervision throughout the duration of the project in the role of project supervisor, his expertise and experience within the domain of Natural Language Processing proved invaluable as the project progressed, without him a successful outcome to the project could not have been achieved.
- Ian Drumm for his critical advice and feedback at a key stage within the project in the role of project assessor.
- Team NLTK for their invaluable contribution to the NLTK open source libraries which formed the base for my project to build upon, without these libraries my project could not have been achieved since the suite offers a simple interface to very complex functionality of Natural Language Processing theory and is the most commonly used tool for any work within the field.
- Finally, I would like to give thanks to my Family and Friends for their patience and encouragement throughout the duration of the project.

Contents

Chapter 1 Introduction

1.1 Motivation for project

The conglomeration of textual data within the modern age is mostly of the unstructured type which is where the need to apply meaning and context to such unstructured textual data becomes important. When this data is concerned with Natural Language then the field of Natural Language Processing comes forward.

Gender Bias analysis is the processing of determining gender bias given some input, this input may be in the format of text or speech, the level of difficulty in detecting gender bias within text is considerably difficult than speech as the human emotional element is removed with an abstract view presented. Therefore, when attempting to detect Gender Bias using text it is important to study Natural Language Processing given that it allows a structured methodology in which to deliver more context to such text for analysis. It is important to study Gender Bias so that inequalities within natural language can be detected, this is especially the case within many domains whereby gender neutrality needs to be maintained e.g. within law, recruitment, business etc. From a language perspective utilising English as an initial starting point allows for an objective perspective with a focus on specific texts rather than dealing with the gender bias inherit within languages as many other languages are gendered to varying degrees.

Natural Language Processing in particular interested me given the exponential growth within the field alongside the growth of Big Data, the applicability of the field to so many domains gave me a perspective on the field of Software Engineering through a fresh lens which is what ultimately led to me deciding to do a project within the field of Natural Language Processing.

The motivation for the project came over the summer of 2016 when I was researching ideas for a final year project, having scored well in my first two years at university I had confidence in my technical ability to readily pick up on a project within a field outside my comfort zone as long as I could apply myself to produce some technical deliverables. My confidence not only came from my results from the previous years but also my keen interest in broadening my horizons beyond traditional projects such as a mobile app or a website. Initially I thought of doing a project related to low-level security having just taken a gap year to take part in an exchange programme to study a semester in Computer Security at the University of Applied Sciences, Amsterdam, though I decided against this since I was looking to operate beyond my boundaries which is where I stumbled upon Artificial Intelligence and Natural Language Processing.

The initial direction of the project was defined as the general analysis of textual data which later evolved into the assessment of Gender Bias using Natural Language Processing techniques, this evolvement came as research indicated very little work had been conducted on the subject with interest in Natural Language Processing techniques focused on social media analysis and search engine filtering. This only furthered my motivation with the opportunity to contribute positively to a particular domain and problem.

1.2 Aims and Objectives

In order to achieve a successful outcome for the project, the following aims and objectives were outlined for the project:

Aim 1 – Detection of Gender Bias using Natural Language Processing techniques:

Objective 1. To implement functionality for Text Tokenization on a word and sentence level alongside testing.
Objective 2. To implement functionality for custom layered Part of Speech Tagging alongside testing.
Objective 3. To implement functionality for Stemming and Lemmatization using the Porter Stemmer and WordNetLemmatizer alongside testing.
Objective 4. To implement functionality for recognition of Named Entities, especially Persons alongside testing.
Objective 5. To implement functionality for Sentiment Extraction alongside testing.
Objective 6. To implement functionality for Gender Classification alongside testing.
Objective 7. To implement functionality for the detection of Gender Bias alongside testing.

Naturally following the above aim comes the aim to extend the project with the below objectives which add additional features.

Aim 2 – Possible Extension of project with additional features:

Objective 1. To implement functionality for the detection of Topics through Topic Modelling alongside testing.

Objective 2. To implement functionality for a Graphical User Interface exposing Natural Language Processing techniques previously implemented and tested.

1.3 Approach adopted

Following on from the above defined aims and objectives a formal approach and methodology was applied to the project. This approach factored in the current tools that are available that operate within the scope of the project and how I can further extend the functionality of these given tools to achieve the given aims.

In alignment with professional standards, the formal approach was set at the beginning of the project and continued until a successful outcome to the project was achieved and deliverables were able to be produced. The approach followed was a hybrid Agile Incremental methodology which included the use of applicable Agile techniques such as Test Driven Development and deliverables being produced through a Minimal Viable Product encapsulating key features of the project with additional iterations producing optional additional extensions to the Minimal Viable Product. The Incremental nature of approach allowed me make up for my initial shortfall in knowledge of the field of Natural Language Processing by constructing the product from simple objectives progressing to advanced objectives which required a strong foundational understanding of the field whilst churning out deliverables satisfying the potential customer who was also my project supervisor Apostolos Antonacopoulos given his research interest within the field of Natural Language Processing.

1.4 Summary of Project Plan

The overall goal of the project is providing functionality for the assessment of Gender Bias using Natural Language Processing techniques.

The plan was reconsidered many times as the project progressed, these reconsiderations included the modification of the Minimal Viable Product objectives to become optional additional objectives and vis versa.

The plan outlines the importance of the Minimal Viable Product to encapsulate the deliverance of the overall goal with optional additional objectives providing functionality building upon this overall goal.

Management of deadlines and deliverables was done through the use of an online Gantt chart tool.

Below is a list of the given tasks and their given deadlines:

28/10/2016 - To implement functionality for Text Tokenization on a word and sentence level alongside testing.

04/11/2016 - To implement functionality for custom layered Part of Speech Tagging alongside testing.

25/11/2016 - To implement functionality for Stemming and Lemmatization using the Porter Stemmer and WordNetLemmatizer alongside testing.

15/12/2016 - To implement functionality for recognition of Named Entities, especially Persons alongside testing.

10/02/2017 - To implement functionality for Sentiment Extraction alongside testing.

17/02/2017 - To implement functionality for Gender Classification alongside testing.

24/02/2017 - To implement functionality for the detection of Gender Bias alongside testing.

Deadlines for the below optional tasks are not defined as they were deemed non-achievable so were not delivered:

- To implement functionality for the detection of Topics through Topic Modelling alongside testing.
- To implement functionality for a Graphical User Interface exposing Natural Language Processing techniques previously implemented and tested.

1.5 Structure of Dissertation

The structure of the dissertation is as following:

- Chapter 2 – Outlining background work associated with the given project including a discussion on methodologies, tools, standards, algorithms and techniques that were used in the project including the rationale for the use of these techniques.
- Chapter 3 – Outlining the Specification and Design aspects of the project including the requirements phase, the design of the software including the rationale and justification for design decisions made throughout the project.
- Chapter 4 – Outlines a detailed view of the Development and Implementation aspect of the project by describing how the designed system was implemented to meet the requirements and including a description of the main problems faced and the solutions adopted. Also provides a justification of the methods and tools adopted.
- Chapter 5 – Outlines the Testing Strategy implemented and the results from adopting this strategy.
- Chapter 6 – Critical evaluation offers a review of the projects plan and achievements against its objectives and any deviations. This Chapter also gives a critical evaluation of the product including strengths and weaknesses. This Chapter also outlines lessons learnt during the course of the project.
- Chapter 7 – Concludes the report with a review of the work done. Possible improvements to the product are included including discussing how further work could be done upon the project by next year's final year students given that it is a research orientated project. This chapter also reflects on the legal/social/ethical/professional issues associated with the given project.

Chapter 2 Background

2.1 Background Knowledge of Field

This section presents a background knowledge of the field of Natural Language Processing beginning from an overview of the field to individual Natural Language Processing techniques progressing in order of complexity.

2.1.1 Natural Language Processing

In recent years, Big Data has grown exponentially and has become the latest buzz word within Technology, as Big Data has grown so has the need to classify and categorize such large volumes of data so that context and meaning can be applied to such data, when such data is unstructured text (natural language) then this is where the field of Natural Language Processing steps in. The field of Natural Language Processing is concerned with the interaction between computers and human natural language, the field stems from the fields of Computer Science, Computational Linguistics and Artificial Intelligence. (Chowdhury, 2003) Using established Natural Language Processing techniques, a machine can be made to extract information from and understand Natural Language which by its nature can be ambiguous and context-specific. Examples include Social Media platforms who store large volumes of data including natural language data, given that these platforms manage such data it makes sense they would like to make use of such data which is where Natural Language Processing techniques can be used to categorize and apply context through techniques such as Sentiment Analysis and Topic Modelling. These techniques are also increasingly utilised by search engines to deliver more meaningful results to users. (Cafarella & Etzioni, 2005)

There is a standard sequence of the processing of exposing Natural Language Processing components, these components include Text Tokenization, Part of Speech Tagging, Stemming and Lemmatization, Named Entity Recognition, Sentiment Analysis and Topic Modelling. This sequence is illustrated below:

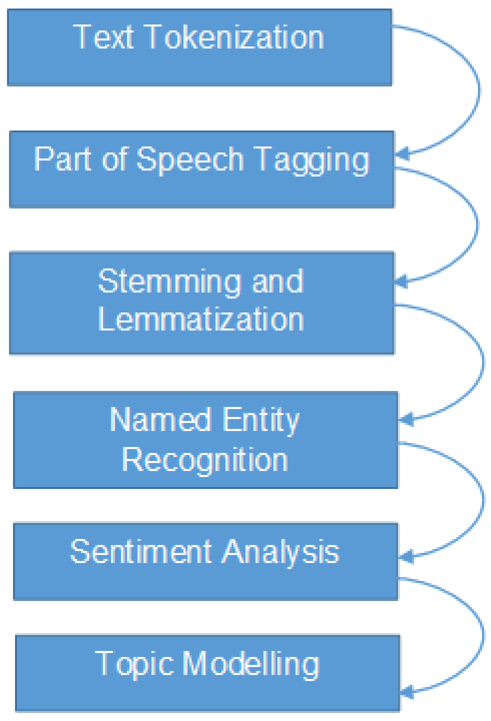


Figure 2.1 – Natural Language Processing Sequence Diagram

2.1.2 Text Tokenization

Within the field of Natural Language Processing and Lexical Analysis, Text Tokenization is fundamental and provides a base to initiate further Natural Language Processing Techniques such as Tagging, Stemming etc. Text Tokenization is the process of breaking down streams of text into elements such as sentences, words, phrases etc. This process allows for more meaningful context to be determined from text. The extracted elements using the tokenization process are known as tokens. (Kaplan, 2005) Within certain languages such as Chinese and Korean with their being no marking for word boundaries the use of Whitespace tokenization is not a suitable technique, each language presents unique challenges to the task of Tokenization with differences in language structure, punctuation etc. Resolutions to meet these challenges include the need to develop increasingly complex heuristics or the creation of language models. Whitespace is the greatest indication of word tokenization though whitespace itself can be a misleading factor e.g. within the word "San Francisco", according to whitespace tokenization they are two independent words whereas in reality it is expected to be managed as one word.

Generally, Tokenization is done to extract sentences and words, techniques such as the use of regular expressions may be used to remove whitespace and punctuation to extract the targeted content, punctuation can increase the challenge of the Tokenization process within many languages as punctuation is dependent upon to convey context and unique meaning to words, phrases and sentences e.g. Should the following word “Aren’t” be tokenized inclusive of the apostrophe punctuation then the tokens extracted will provide no indication of meaning or context and depending on the method utilised for tokenization a variety of unique outputs could be produced such as “are n’t”, “aren t” and “arent”. Within the tokenization process many challenges can become apparent when dealing with various text structures e.g. urls, emoticons, hyphens etc.

2.1.3 Part of Speech Tagging

Part of Speech Tagging is the process of marking individual tokenized words within their given contexts i.e. Its relationship to related words adjacent within given phrases, sentence and paragraphs. Part of Speech tags utilised for identification will be linguistic terms such as Nouns, Verbs, Adjectives, Adverbs etc. in the most basic form with additional sub terms offering increasingly complex analysis. Words by themselves are generally ambiguous and can be freely applied within a variety of contexts and language structures which is why correct grammatical context needs to be determined. (Mitkov, 2005)

Unlike English many languages utilise extensive grammatical cases with linguistics inflecting mainly through suffixes to indicate meaning, the use of cases was utilised heavily within Old English to indicate grammatical function though Modern English is more analytical with more importance given on the order of words. Within English there are three main cases (Leech & Svartvik, 2013):

- Nominative Case – Indicates subject of sentence
- Genitive Case – Indicates possession or ownership
- Accusative Case – Indicates receiver of an action

Within many languages gender is used as a means of breaking nouns into class, this differs from natural gender association where gender is the gender of a person, animal or character. These languages are generally restricted to Male, Female classification though certain languages have “neuter” class and others have different genders for inanimate and animate objects. This utilisation of gender independent of natural gender is known as Grammatical Gender. Traditionally English was a gendered language though has long stopped classifying nouns by gender. Cases and Grammatical Gender complicate the tagging process as more work will need to be undertaken to ascertain context.

The following are common methods for Part of Speech Tagging:

- Stochastic Tagging – This method utilises probability to ascertain maximum likelihood of tags against specific words. Within this method a training corpus is utilised, an example of a training corpus would be the Brown corpus which contains 500 samples of written text within the English language totalling around 1 million words, this corpus has been painstakingly tagged by hand. Within the targeted test data tags are associated to a given word based on the most frequent tag for that given word within the training data. The problem with this simplistic approach is that the mostly likely picked tagged could not be the correct tag within the given context of the test data.
- Hidden Markov Models – This method accumulates information upon previous n words and tags and using this information conducts a most probable decision to select a tag for the given word through training cycles. Assumptions within this method include that a given words associated tag is dependent upon previous n tags and that this does not change over time as information on following words and tags become known. This model allows the Handling of observed events and hidden events, observed events would be the words within a given sentence and hidden events would be the part of speech tags. (Kupiec, 1992)
- Maximum Entropy Tagging – This method makes no assumptions upon the unknown and models only what is known upon some set of constraints. For the unknown the maximum entropy is selected as the uniform distribution. (Ratnaparkhi, 1996)
- Transformation Rule Based Tagging – This method utilises a concept called Transformation Based Learning which is a rule based algorithm to automatically tag given words within some text. Transformation Based Learning extracts linguistic information from corpora allowing the transformation of state using rules to assign a tag to a given word. These

rules will be dependent on factors such as the preceding and following word. Transformation Based Learning initially begins with a simple solution and loops through cycles, within each cycle decisions are made on whether an improved transformation can be made, this continues until transformations don't add value and there is no more transformation to be made. Transformation Based Learning does not utilise probability within the decision making process which can be a disadvantage depending on the training and testing data utilised, Transformation Based Learning can also be impractical with continuous cycles leading to an exponential increase within the time and resources needed to produce results. A popular example of a Transformation Based Tagger is the Brill Tagger. (Nguyen, Nguyen, Pham, & Pham, 2016)

The output of Part of Speech Tagging provides meaningful input for further advanced Natural Language Processing techniques such as Chunking and Named Entity Recognition.

2.1.4 Stemming and Lemmatization

Stemming is the process of reducing words to their stem or root. This stem or root may not be the morphological root of the word though it is generally sufficient that similar words are mapped to the same stem even if the stem is not a valid root by itself. For Example, the following words "particle" and "particles" are stems of "particl" which is not an actual word by itself. (Hull, 1995)

Lemmatization is the process whereby words are grouped together based upon inflections e.g. the lemma for the word 'better' is 'good', the lemma is single item that any number of words may be grouped by, as can be seen lemma of a word is not necessarily it's stem with Lemmatization being a complex approach to stemming. (Barnbrook, Danielsson, & Mahlberg, 2006) Unlike Stemming, Lemmatization also requires the associated part of speech tag of a word alongside the word itself as input as it requires information on the context of the given word so it can discriminate words that are affected on parts of speech. Inflections are common within many languages. English is a moderately inflected language in comparison to other languages such as Latin and Arabic, the process of Lemmatization comes in use with advanced Natural Language Processing techniques such as Topic Modelling where similar concepts can be grouped into abstract relatable topics.

2.1.5 Named Entity Recognition

Named Entity Recognition is the process of extracting named entities from a given tagged text, this process is dependent upon the accuracy of the tagging process as inaccurate tags will lead to inaccurate entities being deduced from such tags. Generally, before Named Entity Recognition can take place a sub-process called Chunking is needed whereby natural word groups are formed from multiple tags e.g. noun or verb phrases. There are two types of Chunking, namely Chunking-Up and Chunking-Down. Chunking-Up refers to moving from small scale information to generalised concepts. Chunking- Down is the opposite and is the extraction of small scale information from generalised concepts.

Another sub-process called Chinking could optionally take place following Chunking to optimise the Chunking process, Chinking removes superfluous words from existing phrases to formulate useful chunks. (Bird, Klein, & Loper, 2006)

The entities recognised through the Named Entity Recognition process are generally Noun entities that are classified using a machine learning classifier according to predefined categories such as e.g. Persons, Organisations, Countries etc.

2.1.6 Sentiment Analysis

Sentiment Analysis is the process of identifying and quantifying sentiment within a given text. Generally, Sentiment is exposed through types such as Positive or Negative or Neutral. This proves particularly useful when identifying opinions regarding a particular subject matter. The most practical use of Sentiment Analysis is opinion mining social media given the large amount of opinionatedly diverse data that social media platforms manage. An example of this would be to opinion mine such social media data for political sentiment on given candidates. Sentiment Analysis also has the potential to deliver Emotional Intelligence to machines offering a powerful component to integrate with existing machine learning approaches. Methods of acquiring Sentiment Analysis include a Lexicon Based approach and

Machine Learning approach. The Lexicon Based approach may utilise a Dictionary Based approach or a Corpus Based Approach through Statistical methods or Semantic methods. (Liu, 2012) The Machine Learning approach may utilise Supervised Learning or Unsupervised Learning with Supervised Learning using classifiers such as Probabilistic, Rule-Based, Linear and Decision Tree Classifiers.

2.1.7 Topic Modelling

Topic Modelling is the process of identifying abstract “topics” within a given text or document, such “topics” can be used to organize and summarise given documents or text. Topic Modelling helps organize, summarize and understand large collections of data. From an abstract perspective Topic Modelling aims to determine structure within given unstructured text or collection of documents. Topic Modelling is still a research orientated field though in future it could have many applications upon the summarization and categorization of social media data and large quantities of text such as news and books.

The most popular model for determining Topics is the Latent Dirichlet Allocation (LDA). LDA defines a “topic” as distribution over words. (Blei, 2012) LDA assumes that the document given as a bag of words, the topic is assumed as a distribution over a fixed vocabulary and that words are generated from specific topic distributions. These topics found can then be utilised to find trends within specific topics and connecting topics.

2.2 Review of Similar Work

Gender Bias detection is not a widely researched area, as I will show below within subsection 2.2.2 there is a method which uses pattern recognition which unlike Natural Language Processing looks at the problem from a less contextual backdrop with less context on input data. The general focus presented within this dissertation is on contextual understanding of natural language rather than pattern recognition so that nuances can be detected.

To the best knowledge of the author the only work present within this subject is from a company called Kanjoya that has applied Natural Language Processing techniques to employee surveys to detect bias with gender being one metric. (Captain, 2015)

Given the fact that the research done by this company is not open source it is difficult from a research perspective to accurately gauge the work done and contribution to the field of Natural Language Processing with there being no option to build upon such research.

2.2.1 Gender Bias Issues

Though there has been no open source work conducted upon the given subject to the authors best knowledge as discussed above. There has been research upon Gender Bias within machine learning algorithms such as Word2Vec which utilise vector modelling though contain gender stereotypes which may be inappropriate e.g. an inappropriate stereotype analogy is tall : man :: short : woman whilst an appropriate stereotype analogy is King : man :: Queen : woman, such analogies exhibit the relationships that are commonly associated with genders inappropriately and appropriately and how this can drastically exaggerate and mislead results. (Bolukbasi, Chang, Zou, Saligrama, & Kalai, 2016) The findings of such research strongly indicate that such Gender Bias within words themselves could exaggerate and mislead understanding of given Gender Bias by machine learning algorithms such as Word2Vec. From my perspective, presently I do not utilise the Word2Vec machine learning algorithm so it does not presently affect me though in future as the gender classification procedure is extended and weaknesses are addressed I may need to utilise Word2Vec so I will be affected by such negativities in certain cases. Trying to re-create a machine learning algorithm such as Word2Vec is far beyond the scope of this project and is impractical given the resources of those who manage these algorithms so the hope is that such sexism within machine learning algorithms such as Word2Vec are addressed independently.

2.2.2 Pattern Recognition

Aside from the field of Natural Language Processing the analysis of Gender Bias has done using Pattern Recognition. (Ali, et al., 2010) This method gathers and annotates large volumes of textual data to extract patterns and trends. Although this method offers meaningful results in terms of Gender Bias it has many weaknesses such as the non-recognition of nuances within text such as sentiment and gender classification. The benefit of this methods is that large datasets can be used to offset such results though from an end user application it is difficult to gauge the practicality and reusability of this method as a component in comparison to the use of Natural Language Processing.

2.2.3 Sentiment Analysis

Sentiment Analysis is the most used technique within Natural Language Processing and is commonly applied on social media platforms given the vast variety of data at disposal. Previous research has been able to detect mood using a combination of a lexicon based and machine learning approaches with weights applied to given words indicating the impact of words within given tweets. (Gaikwad & Joshi, 2017) This weight is utilised alongside the term frequency with tweets classified as Funny, Sad, Happy, Angry and None using machine learning algorithms such as NB, SVM and KNN.

2.2.4 Gender Imbalance

Work has also been conducted to analyse Gender imbalance within the Association of Computational Linguistics Anthology Network using Topic Modelling Machine Learning Algorithms. (Vogel & Jurafsky, 2012) This research found that women publish more on dialog, discourse and sentiment whilst men publish more than women on parsing, formal semantics and finite state models. This study illustrated the difference in tendencies towards particular research topics including a historical context. This method manually labelled the name of authors which can be utilised as a meaningful resource within further studies on gender.

2.3 Tools

NLTK – Natural Language Toolkit (NLTK) is an open source library for Natural Language Processing that exposes implementation and corpuses for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for tokenization, parsing, lemmatization, stemming, tagging, classification, and semantic reasoning. Alternatives to NLTK include TextBlob which expose similar functionality without the same level of support.

SciPy – SciPy is an open source Python library that is used for technical and scientific computing. It features key functions and algorithms core to Pythons scientific computing capabilities and it is depended upon by Natural Language

Processing libraries such as NLTK. This is required for certain probability, tagging, clustering and classification tasks.

NumPy – NumPy is a fundamental open source Python library utilised for scientific computing. It is commonly utilised alongside SciPy given the dependencies between them and is also depended upon by Natural Language Processing libraries such as NLTK. This is required for certain probability, tagging, clustering and classification tasks.

Gensim – Gensim is an open source Python library utilised for vector space modelling and topic modelling. It includes implementations of algorithms such as document2vec, word2vec, random projections, tf-idf, hierarchical Dirichlet processes (HDP), latent semantic analysis (LSA) and latent Dirichlet allocation (LDA).

Pickle – Pickle is a Python module for serializing and de-serializing Python Object structures. This can be utilised to store trained classifiers and then loaded from a Pickle file rather than re-training classifiers improving performance at no negative consequence.

Below is a list of alternative tools that could have been potentially utilised within the project:

TextBlob – TextBlob provides much of the functionality that NLTK offers. It is also coded in Python so was a viable alternative though NLTK is more widely used so was chosen due to the level of user support that could be offered should any issues crop up within development.

Pattern – Pattern is a web mining tool for Python. It provides functionality for Natural Language Processing similar to NLTK. Unlike NLTK Pattern is not as widely used so does not come with the same level of support.

Weka – Weka is a collection of Machine Learning algorithms for data mining tasks. This toolkit is coded in Java and can be utilised to independently or in conjunction with other tools provide functionality for Natural Language Processing. As it was decided to use Python within development for the project this tool was not utilised.

GATE – GATE is a text mining toolkit coded in Java. This toolkit also provides functionality for Natural Langauge Processing and Machine Learning independently and through plugins with other tools such as OpenNLP and Stanford CoreNLP.

Freeling – Freeling provides much of the functionality that NLTK offers. It is coded in C++ so usage of this library would necessitate usage of C++ throughout the project which was ultimately decided against as it was decided that usage of Python would be more suited to the project.

OpenNLP – OpenNLP is a Java Machine Learning toolkit for the processing of Natural Language text. It provides a similar depth of functionality to NLTK though is coded with Java. Had Java been chosen as a development as the language for development this toolkit would have been considered for use within the project.

LingPipe – LingPipe is a toolkit for processing text using computational linguistics. It is coded in Java and provides much of the Natural Language Processing functionality that other tools offer, unlike other tools it is a commercial product and is not open source. Had Java been chosen as the language for development this toolkit would have been considered for use within the project.

Mallet – Mallet is a Machine Learning for Language toolkit that provides functionality for Statistical Natural Language Processing, document classification, topic modelling, cluster analysis and machine learning. This toolkit is coded in Java though NLTK provides an interface to this machine learning package which is why it was useful to decide against using this package as an alternative to NLTK.

Standard Core NLP – Stanford Core NLP provides similar functionality to OpenNLP and is also coded with Java. This was similarly not chosen due to the decision to choose Python as the language for development.

2.4 Programming Languages

Development within any project requires a choice to be made on the desired programming language(s) to be utilised. Each programming language has advantages and disadvantages with them being integral in platform support, performance, efficiency etc. The choice of programming language to be utilised can also be dependent upon the choice of the project, Natural Language Processing is a niche field with 2 main programming languages utilised due the support they offer within the field of Natural Language Processing and the related fields of Computational Linguistics and Machine Learning:

- Java – Whilst not as popular as Python within the domain of Natural Language Processing, Java also provides access to many Natural Language Processing and Machine Learning libraries such as Freeling, OpenNLP, LingPipe, Weka, Mallet and Standard Core NLP.
- Python – Python is the most popular programming language utilised within the field of Natural Language Processing with it providing easy integration with a variety of Natural Language Processing and Machine Learning libraries. The most popular Natural Language Processing tool NLTK is implemented within Python so utilisation of the language would provide easy support, other relevant libraries Python provides support for are TextBlob, SciPy, NumPy, Gensim and Pattern. Python also provides simpler functionality for the manipulation of textual data in comparison to other bulkier languages such as Java.

Upon weighing up the pros and cons of Java and Python it was decided that utilisation of Python would be far more advantageous within the project.

2.5 Text Editor

Having chosen Python as the programming language I chose Sublime Text 3 as the text editor for the project. There are many viable alternatives to Sublime Text 3 that could have been utilised such as Notepad++ and Atom. I found that Sublime Text 3 was very good looking with its colour coding, easy to use interface and navigation. Notepad++ in comparison could be seen as too simplistic with not so easy navigation. Atom in comparison could be seen as being more beneficial to Sublime Text 3 with its integration with Git and GitHub, though from a personal perspective I preferred managing version control independently of the text editor. In comparison to other text editors Sublime Text 3 is also faster and more stable.

2.6 Code Repository

GitHub was chosen as the code repository mainly due it's interactive interface, initially Git was going to be used for managing version control though given my inexperience with version control and the learning curve associated with using Git it was decided that it was more suitable to use a web-based graphical interface such as GitHub. There are many other alternatives to GitHub that could have been chosen such as SourceForge, BitBucket etc. to equal measure though from a personal perspective I found the GitHub user interface more intuitive which is why it was ultimately chosen.

Chapter 3 Specification and Design

This chapter discusses the requirements phase and the design of the product as a whole including a rationale and justification for decisions made. Specification and Design are essential components in the successful outcome of any project.

Given that the project is research orientated with an emphasis on the application of Natural Language Processing techniques the specifications throughout the project were very broad with the project supervisor and customer offering very little in the way of formal requirements other than the unique application of Natural Language Processing techniques with proof. This one broad requirement eventually morphed into me in consultation with the customer laying down some specific requirements to manage scope and the direction of the project which would aid me in the prevention of scope creep and the satisfaction of the customer with regular updates on tracking the status of the project.

3.1 Requirements

The following are core requirements of the project:

1) To implement functionality for Text Tokenization on a word level alongside testing
2) To implement functionality for Text Tokenization on a sentence level alongside testing
3) To implement functionality for custom layered Part of Speech Tagging alongside testing
4) To implement functionality for Stemming using the Porter Stemmer alongside testing
5) To implement functionality for Lemmatization using the WordNetLemmatizer alongside testing
6) To implement functionality for recognition of Named Entities, especially Persons alongside testing
7) To implement functionality for Sentiment Extraction alongside testing
8) To implement functionality for Gender Classification alongside testing
9) To implement functionality for the detection of Gender Bias alongside testing

The above core requirements encapsulate the scope of the Minimal Viable Product and it was decided the satisfaction of such requirements would provide a successful outcome to the project.

Optional additional requirements:

1) To implement functionality for the detection of Topics through Topic Modelling alongside testing
2) To implement functionality for a Graphical User Interface exposing Natural Language Processing techniques previously implemented and tested

The above optional requirements encapsulate the scope of the additional iterations naturally following on from the Minimal Viable Product objectives. It was decided that the setting of such requirements would not only value to the project should time constraints allow, alongside they could provide indication of the extension of the project given the research orientated nature of the project.

3.2 Audience

The audience for the given project are researchers who have an interest in the detection of Gender Bias especially the detection of Gender Bias using Natural Language Processing techniques. Such researchers may wish to extend the project for utilisation within other projects or they may independently extend this project with the addressing of weaknesses and extending implementation through to Topic Modelling and the creation of a Graphical User Interface. The audience for this project is a very niche audience with the potential to reach a wider more mainstream audience as the project is extended into a fully-fledged application or integrated with an existing application.

3.3 Design

Having decided to a do a research orientated project the Design aspect of the project was minimal with an emphasis placed on research, implementation and testing. Had the project completed the additional optional objective of the creation of a Graphical User Interface the Design aspect of the project could have been much more broad with designs of the interface.

Below is displayed the overall class diagram for the project illustrating the relationships and exhibited functionality of individual classes that contributed to the overall goal of the detection of Gender Bias using Natural Language Processing techniques.

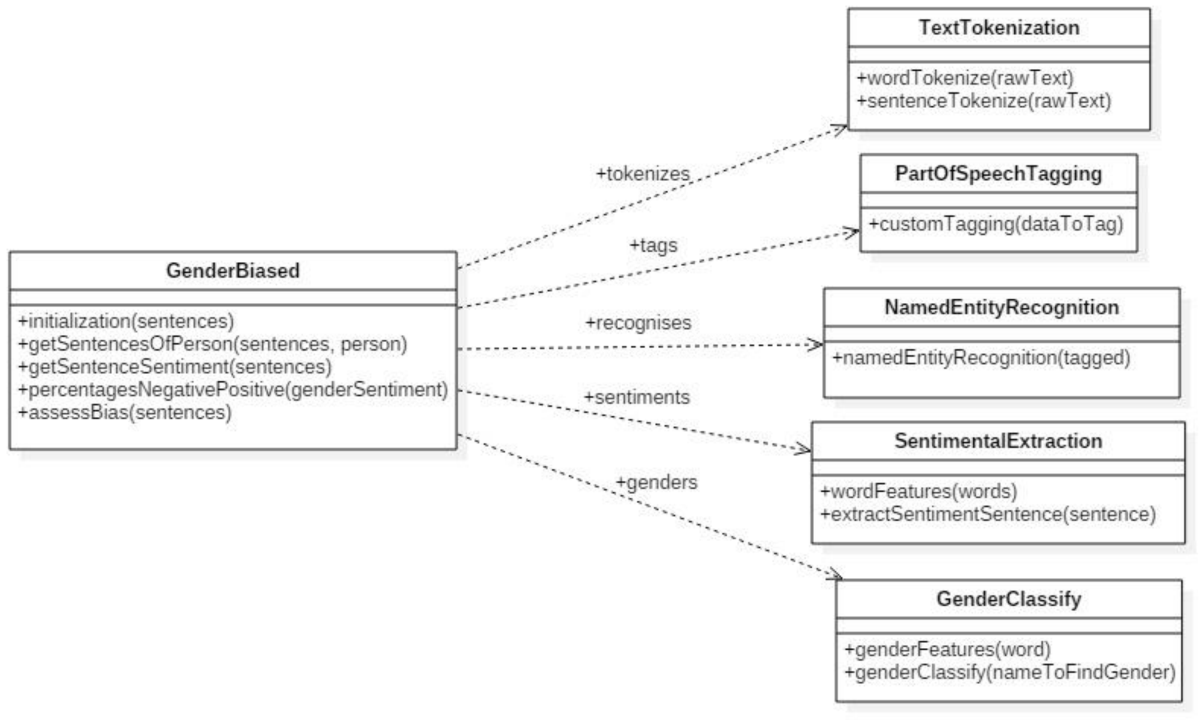

TextTokenization
+wordTokenize(rawText)
+sentenceTokenize(rawText)
PartOfSpeechTagging
+customTagging(dataToTag)
GenderBiased
+initialization(sentences)
+getSentencesOfPerson(sentences, person)
+getSentenceSentiment(sentences)
+percentagesNegativePositive(genderSentiment)
+assessBias(sentences)
NamedEntityRecognition
+namedEntityRecognition(tagged)
SentimentalExtraction
+wordFeatures(words)
+extractSentimentSentence(sentence)
GenderClassify
+genderFeatures(word)
+genderClassify(nameToFindGender)
+tokenizes
+tags
+recognises
+sentiments
+genders

Figure 3.1 - Gender Bias Class Diagram

- The TextTokenization class encapsulates the independent functionality of Text Tokenization that is used by the GenderBiased class.
- The PartOfSpeechTagging class encapsulates the independent functionality of Part of Speech Tagging that is used by the GenderBiased class.
- The NamedEntityRecognition class encapsulates the independent functionality of Named Entity Recognition that is used by the GenderBiased class.
- The SentimentExtraction class encapsulates the independent functionality of Sentiment Extraction that is used by the GenderBiased Class.
- The GenderClassify class encapsulates the independent functionality of Gender Classification that is used by the GenderBiased Class.

- The GenderBiased class encapsulates functionality that utilises all the Above classes to assess the overall goal of Gender Bias.

<u>Rationale and Justification for Class Design Choices</u>

- Stemming and Lemmatization is not included within the above Class Diagram because these two Natural Language Processing techniques were not utilised within the Gender Bias Analysis, had the scope of the project extended to the optional additional requirements of Topic Modelling then these two Natural Language Processing techniques could have been utilised within the overall Class Diagram of the project. The above Class diagram represents the overall goal of the detection of Gender Bias using Natural Language Processing techniques.
- The separation of classes exhibited utilised the concept of Object Orientated Programming whereby the Natural Language Processing techniques are independent objects and then utilised through a GenderBiased object. Alternatively, all functionality could have been implemented within the GenderBiased class though from a design perspective this would lead to many complications further down the line as the project is extended and may move within a different direction, such a decision would lead to the need for code refactoring. The separation of classes at an early stage within the Design process averted many further complications given that the project is research orientated with a volatile project scope dealing with unknown variables.

Chapter 4 Development and Implementation

This chapter outlines the Development and Implementation aspect of my project. Also included within this chapter is a discussion of Problems Faced and Solutions Adopted at each phase alongside a justification of the methods and tools adopted.

4.1 How designated system was implemented to requirements

4.1.1 Development Environment – Text Editor

Before starting the project, it was necessary to set up the development environment. Sublime Text 3 was installed as a lightweight text editor with built in support for the Python language which was utilised throughout development and implementation.

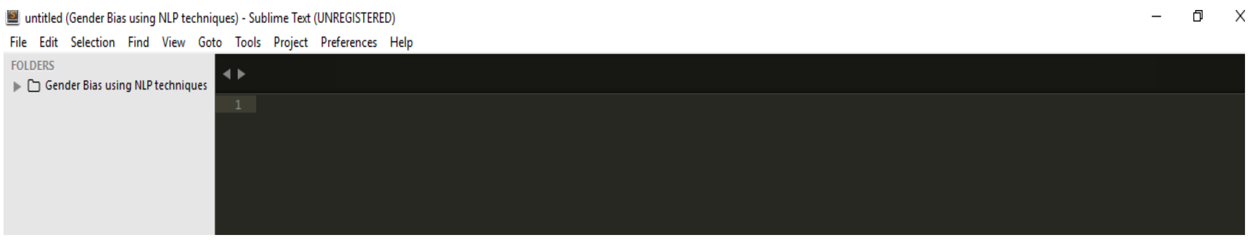

untitled (Gender Bias using NLP techniques) - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help
FOLDERS
Gender Bias using NLP techniques

Figure 4.1 - Sublime Text 3 Environment

Problems Faced and Solutions Adopted:

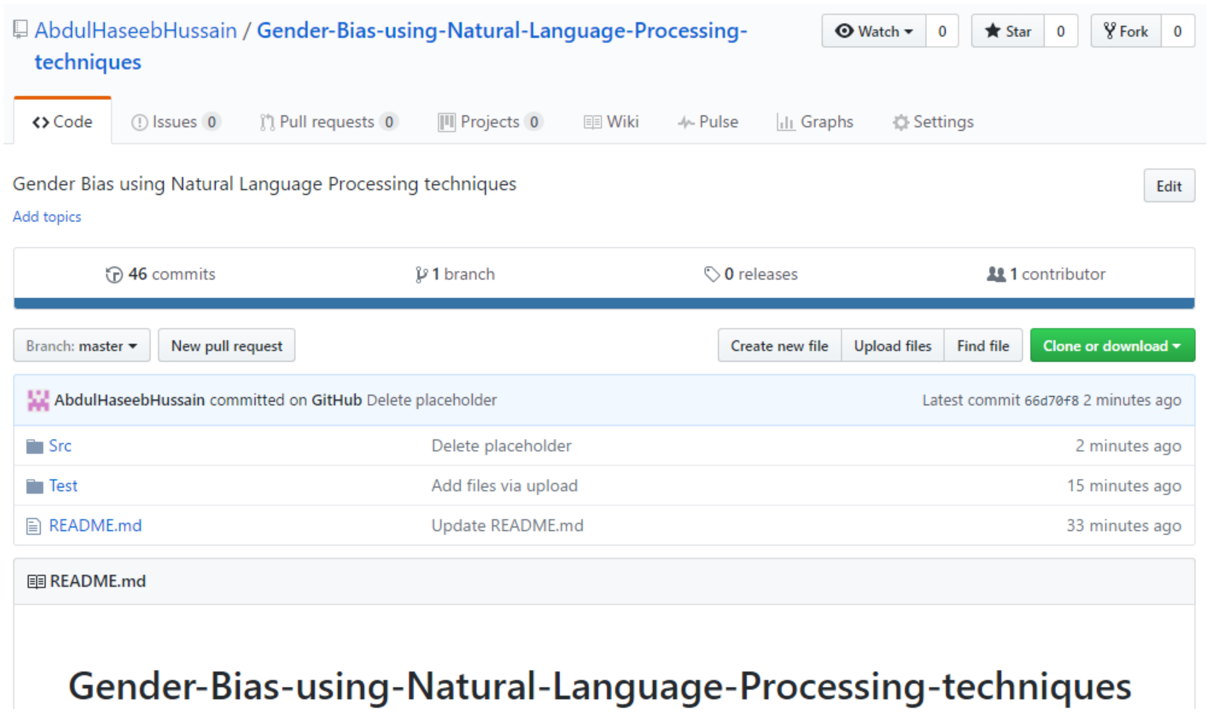
There were no issues with the choice of Sublime Text 3 as the text editor.

Justification of methods and Tools Adopted:

There are many viable alternatives to Sublime Text 3 that could have been utilised such as Notepad++ and Atom. I found that Sublime Text 3 was very good looking with its colour coding, easy to use interface and navigation. Notepad++ in comparison could be seen as too simplistic with not so easy navigation. Atom in comparison could be seen as being more beneficial to Sublime Text 3 with its integration with Git and Github, though from a personal perspective I preferred managing version control independently of the text editor. In comparison to other text editors Sublime Text 3 is also faster and more stable.

4.1.2 Code Hosting and Version Control

Github was utilised as the code hosting platform managing version control. Upon the completion of each objective the project was updated on Github as a backup measure for storing and managing the code.



Gender Bias using Natural Language Processing techniques

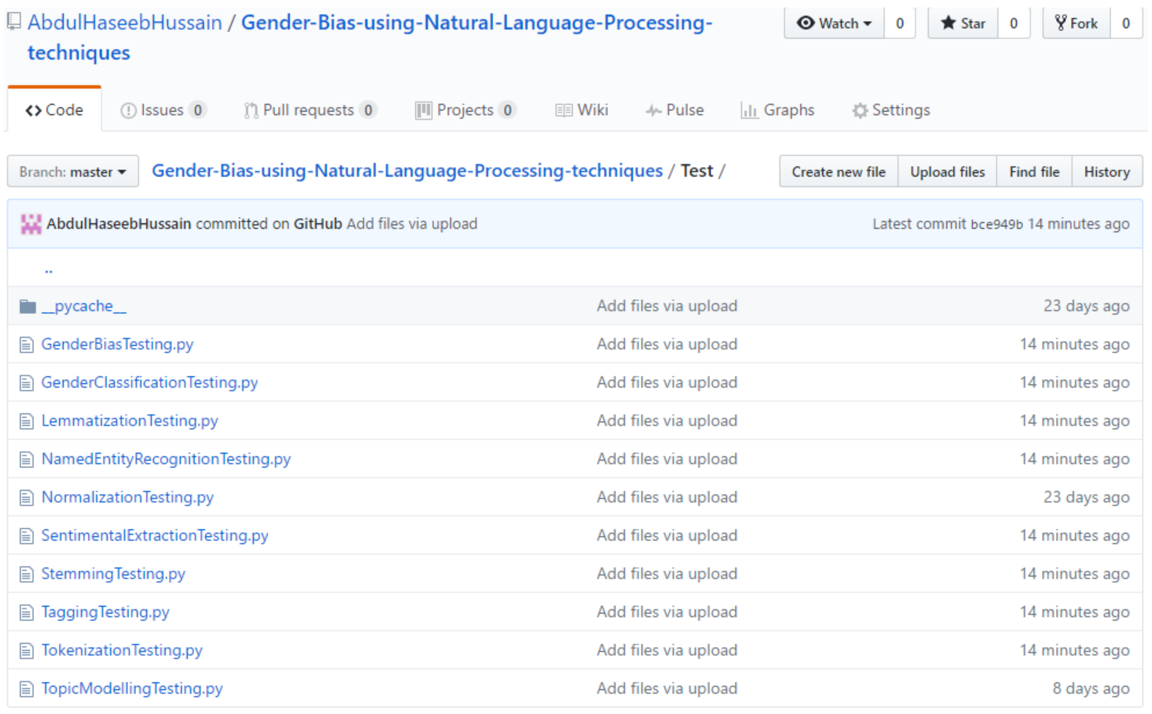
46 commits | 1 branch | 0 releases | 1 contributor

AbdulHaseebHussain committed on GitHub Delete placeholder — Latest commit 66d70f8 2 minutes ago

Src	Delete placeholder	2 minutes ago
Test	Add files via upload	15 minutes ago
README.md	Update README.md	33 minutes ago

Gender-Bias-using-Natural-Language-Processing-techniques

Figure 4.2 - GitHub Project folder



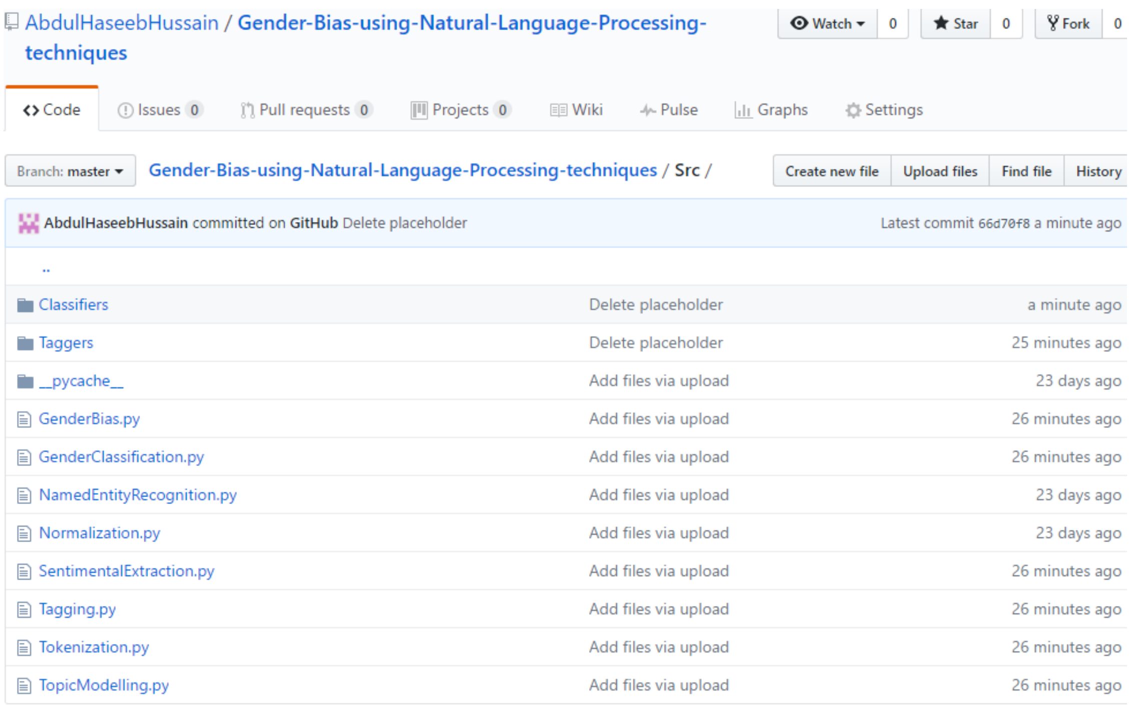
AbdulHaseebHussain / Gender-Bias-using-Natural-Language-Processing-techniques

Branch: master / Gender-Bias-using-Natural-Language-Processing-techniques / Test /

AbdulHaseebHussain committed on GitHub Add files via upload — Latest commit bce949b 14 minutes ago

..		
__pycache__	Add files via upload	23 days ago
GenderBiasTesting.py	Add files via upload	14 minutes ago
GenderClassificationTesting.py	Add files via upload	14 minutes ago
LemmatizationTesting.py	Add files via upload	14 minutes ago
NamedEntityRecognitionTesting.py	Add files via upload	14 minutes ago
NormalizationTesting.py	Add files via upload	23 days ago
SentimentalExtractionTesting.py	Add files via upload	14 minutes ago
StemmingTesting.py	Add files via upload	14 minutes ago
TaggingTesting.py	Add files via upload	14 minutes ago
TokenizationTesting.py	Add files via upload	14 minutes ago
TopicModellingTesting.py	Add files via upload	8 days ago

Figure 4.3 - GitHub Test Folder



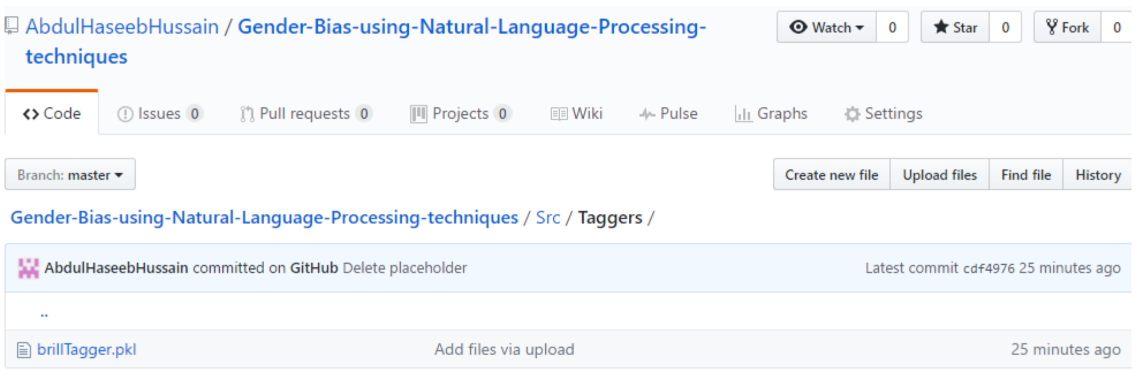
AbdulHaseebHussain / Gender-Bias-using-Natural-Language-Processing-techniques

Branch: master / Gender-Bias-using-Natural-Language-Processing-techniques / Src /

AbdulHaseebHussain committed on GitHub Delete placeholder — Latest commit 66d70f8 a minute ago

..		
Classifiers	Delete placeholder	a minute ago
Taggers	Delete placeholder	25 minutes ago
__pycache__	Add files via upload	23 days ago
GenderBias.py	Add files via upload	26 minutes ago
GenderClassification.py	Add files via upload	26 minutes ago
NamedEntityRecognition.py	Add files via upload	23 days ago
Normalization.py	Add files via upload	23 days ago
SentimentalExtraction.py	Add files via upload	26 minutes ago
Tagging.py	Add files via upload	26 minutes ago
Tokenization.py	Add files via upload	26 minutes ago
TopicModelling.py	Add files via upload	26 minutes ago

Figure 4.4 - GitHub Src Folder

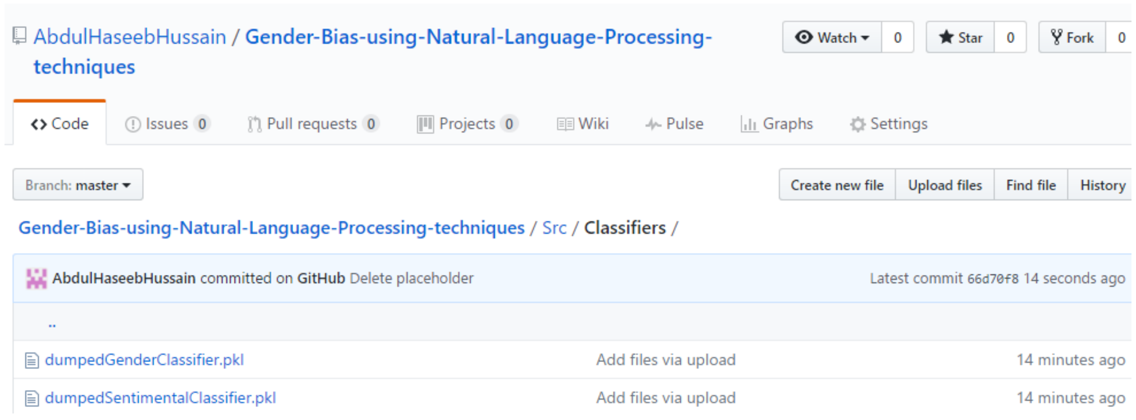


Gender-Bias-using-Natural-Language-Processing-techniques / Src / Taggers /

AbdulHaseebHussain committed on GitHub Delete placeholder — Latest commit cdf4976 25 minutes ago

..		
brillTagger.pkl	Add files via upload	25 minutes ago

Figure 4.5 - GitHub Taggers Folder



Gender-Bias-using-Natural-Language-Processing-techniques / Src / Classifiers /

AbdulHaseebHussain committed on GitHub Delete placeholder — Latest commit 66d70f8 14 seconds ago

..		
dumpedGenderClassifier.pkl	Add files via upload	14 minutes ago
dumpedSentimentalClassifier.pkl	Add files via upload	14 minutes ago

Figure 4.6 - GiHub Classifiers Folder

Problems Faced and Solutions Adopted:

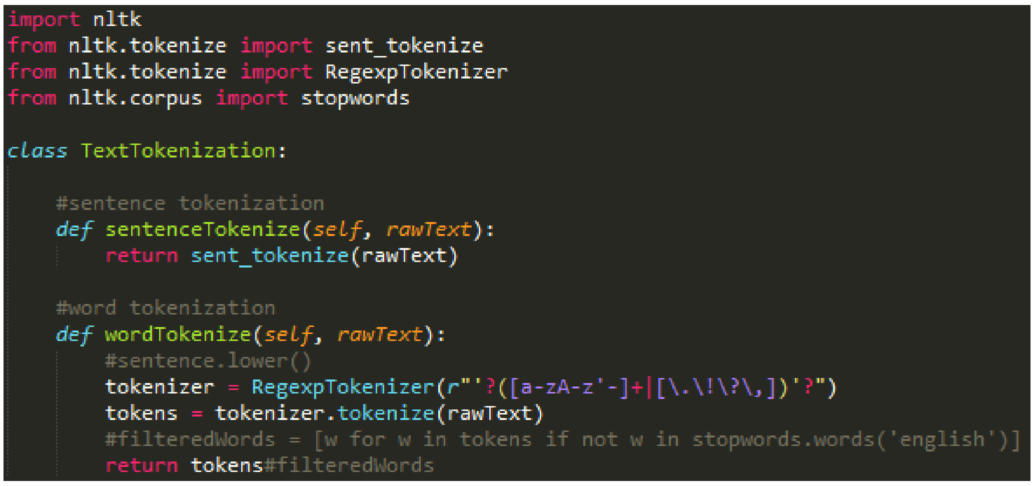
There were no problems associated with code hosting and version control.

Justification of methods and Tools Adopted:

Initially Git was going to be used for managing version control though given my inexperience with version control and the learning curve associated with using Git it was decided that it was more suitable to use the web-based graphical interface with GitHub.

4.1.3 Text Tokenization

Text Tokenization refers to the splitting of Text by some measure, within this project this will be by words and sentences. The topic of Text Tokenization is discussed in detail in Section 2.1.2.



```
import nltk
from nltk.tokenize import sent_tokenize
from nltk.tokenize import RegexpTokenizer
from nltk.corpus import stopwords

class TextTokenization:

    #sentence tokenization
    def sentenceTokenize(self, rawText):
        return sent_tokenize(rawText)

    #word tokenization
    def wordTokenize(self, rawText):
        #sentence.lower()
        tokenizer = RegexpTokenizer(r"'?([a-zA-z'-]+|[\.\!\?\,])'?")
        tokens = tokenizer.tokenize(rawText)
        #filteredWords = [w for w in tokens if not w in stopwords.words('english')]
        return tokens#filteredWords
```

Figure 4.7 - TextTokenization class

The Sentence Tokenization and Word Tokenization methods shown above draw heavily from the NLTK suite of libraries, within the TextTokenization class there are two methods:

- sentenceTokenize – This method is utilised for sentence tokenization within the project. This method uses the standard sent_tokenize method drawn from the NLTK suite of libraries namely PunktSentenceTokenizer which splits text into sentences based upon machine learning on the Punkt corpus that comes along with NLTK. This method learns parameters (a list of abbreviations) trained upon the corpus using unsupervised learning and uses this to sentence tokenize the given text. PunktSentenceTokenizer provides support for multiple European languages including English.
- wordTokenize – This method is utilised for word tokenization within the project. This method does not utilise the NLTK standard word tokenization method word_tokenize because it includes numbers as words which is unnecessary within my application. Instead NLTK's RegexpTokenizer has been utilised to build a custom Tokenizer which detects words and certain punctuation which will be of value. This custom regular expression tokenizer uses a method whereby words are split by based on punctuation such as full

stops, commas, exclamation marks and question marks. This method ignores hyphens as a mechanism for word splitting as a hyphen can be utilised as a linking mechanism between words so that they are expected to be treated as one word e.g. hyphens between person or place names.

Problems Faced and Solutions Adopted:

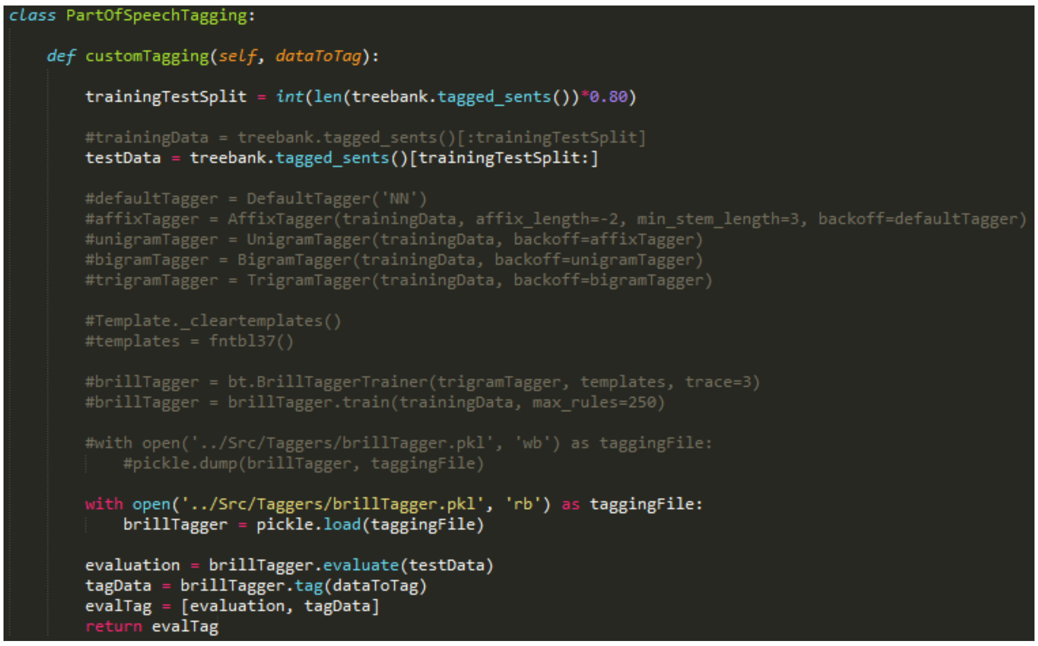
- Problems with word_tokenize – Originally my application utilised NLTK's word_tokenize method though it was realised that this could cause complications later with numbers being accepted as words with further Natural Language Processing techniques trying to include such numbers incorrectly within computations.
- Problems with Stop Word removal – Originally the Word Tokenization process removed Stop Words. Stop Words are the most common words within a language e.g. 'in', 'and' etc. The reasoning behind removing these high frequency Stop Words was that their removal would allow an increased focus on the remaining more important words though as the project progressed issues cropped up with entities not being recognised within the Named Entity Recognition aspect of the project as such stop words provided context within Tagging and entity recognition.
- Problems with Punctuation – Originally certain punctuation was not included within word tokenization. This punctuation was full stops, commas, question marks and exclamation marks, as the project progressed into further more advanced Natural Language Processing techniques it was recognised that many entities especially Persons were not being recognised / recognised correctly within the Named Entity Recognition phase due to slight incorrectness in tagging which was ultimately caused by certain punctuation not being included. The reversal of this decision and the inclusion of these punctuation marks allowed increased accuracy within the detection of entities especially Persons which are a key component of the application. As the original decision was made it was anticipated that the inclusion of such punctuation would be unnecessary and only words would be needed within further Natural Language Processing techniques though as development reached the Named Entity Recognition phase, Person entities were not being recognised independently e.g. "Abdul Haseeb Hussain, Adam Jones" was being recognised as one entity rather than two entities.

Justification of methods and Tools Adopted:

The justification for using the NLTK suite of libraries for Text Tokenization as opposed to TextBlob (the other alternative using Python) which is a library that can perform similar functions is that NLTK is more popular thereby there is more support should any issues crop up later within development as the project delves into more advanced Natural Language Processing techniques.

4.1.4 Part of Speech Tagging

Part of Speech Tagging refers to the marking up of given text by grammatical classifications e.g. Nouns, Verbs, Adverbs etc. This is discussed in detail in Section 2.1.3.



```python
class PartOfSpeechTagging:

    def customTagging(self, dataToTag):

        trainingTestSplit = int(len(treebank.tagged_sents())*0.80)

        #trainingData = treebank.tagged_sents()[:trainingTestSplit]
        testData = treebank.tagged_sents()[trainingTestSplit:]

        #defaultTagger = DefaultTagger('NN')
        #affixTagger = AffixTagger(trainingData, affix_length=-2, min_stem_length=3, backoff=defaultTagger)
        #unigramTagger = UnigramTagger(trainingData, backoff=affixTagger)
        #bigramTagger = BigramTagger(trainingData, backoff=unigramTagger)
        #trigramTagger = TrigramTagger(trainingData, backoff=bigramTagger)

        #Template._cleartemplates()
        #templates = fntbl37()

        #brillTagger = bt.BrillTaggerTrainer(trigramTagger, templates, trace=3)
        #brillTagger = brillTagger.train(trainingData, max_rules=250)

        #with open('../Src/Taggers/brillTagger.pkl', 'wb') as taggingFile:
            #pickle.dump(brillTagger, taggingFile)

        with open('../Src/Taggers/brillTagger.pkl', 'rb') as taggingFile:
            brillTagger = pickle.load(taggingFile)

        evaluation = brillTagger.evaluate(testData)
        tagData = brillTagger.tag(dataToTag)
        evalTag = [evaluation, tagData]
        return evalTag
```

Figure 4.8 - PartOfSpeechTagging class

Within the PartOfSpeechTagging class there is one method:

- customTagging – This method is utilised for Part of Speech Tagging within the application. This method draws heavily from NLTK's tagging libraries. This method uses a layered tagging approach with the Brill Tagger (Brill, 1992) at the top with a Trigram backoff Tagger. A Brill Tagger is a transformational rule-based tagger which assigns a default tag sequence and then applying an ordered list of transformational rules to correct the tags of individual tokens. These rules are learned through training the Brill Tagger with training data and the templates which in this case were from fntbl37 which offers 37 templates.

A Trigram Tagger is a 2nd order tagger which utilises the knowledge about the token alongside previous 2 tokens and their associated tags to statistically determine the tag for the current token. The backoff Tagger for the Trigram Tagger is a Bigram Tagger. A Bigram Tagger is a 1st order tagger which utilises knowledge about the token alongside the previous token and its associated tag to statistically determine the tag for the current token. The backoff Tagger for the Bigram Tagger is the Unigram Tagger. A Unigram Tagger is 0th order tagger which only uses the token itself to determine tag for the token. The backoff Tagger for the Unigram Tagger is the Affix Tagger. An Affix Tagger utilises knowledge about the affix of the word to determine the probability of the tag for the given word. The backoff Tagger for the Affix Tagger is the Default Tagger which is assigned a default tag of 'NN' which Is the most common tag. The backoff Tagger is utilised if the given tagger cannot find a tag. (Loper, Klein, & Bird, 2009)

All of the layered taggers not just the brill tagger is trained and tested using the treebank corpus. An 80:20 training:test split was decided. The trained Brill Tagger is used to tag data and then this is serialized and saved using pickle so that there is no need to retrain taggers and instead trained Taggers can be subsequently loaded from the pickle file. This method returns the tagged data for the given input alongside an evaluation of the general accuracy of the method implemented.

Problems Faced and Solutions Adopted:

- Training and Testing Split - Throughout the process of formulating the Part of Speech Tagging method, experimentation need to be conducted to assess the right balance between testing and training data. Eventually an 80:20 split was settled upon as less training data provides parameter estimates with greater variance and with less testing data the performance statistic will have a greater variance.
- Non-Layered Tagging Approach – Initially a Layered Tagging approach was not utilised which led to fast though inaccurate results which is why a layered tagging was utilised beginning from providing more context to more specific until finally

the most probable tag of “NN” is applied, this change of approach increased the accuracy of the Part of Speech method.

- Problems with Speed – Initially each time the Part of Speech Tagging method was called the layered Taggers would need to be trained, this was very time consuming therefore a solution was adopted whereby the trained Tagger would be serialized and saved using Pickle to a pickle file and then loaded each time the Part of Speech Tagging method is called, the code for training is commented out should it be needed again to retrain the Tagger should any changes need to be implemented. This solution led to considerable improvements with speed at no negative consequence. Another issue with speed was caused by the choice of the corpus for the training and test data. Initially the training and test corpus utilised was the Brown corpus which is 57340 in length, given that a layered Brill Tagger was utilised this was very time consuming which is why I decided to use the treebank corpus which is 3914 in length, this decision speeded up performance alongside delivering just as meaningful tagging results.

Justification of methods and Tools Adopted:

The Treebank corpus (Marcus, Marcinkiewicz, & Santorini, 1992) was utilised as it provided a cost effective solution with increased performance and approximately just as meaningful results.

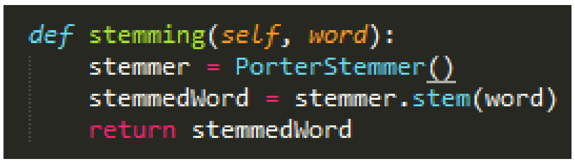
The use of the fntbl37 templates returns 37 templates as opposed to other templates such as nltkdemo18 which returns 18 templates, nltkdemo18plus which returns 18 templates and brill24 which returns 24 templates. Thereby fntbl37 allows for more templates and better results from training.

The use of Pickle (Python Software Foundation, 2017) for serializing, saving and loading trained taggers increased performance as it meant that there was no need to retrain taggers and instead the trained taggers could be loaded from the pickle file.

The use of NLTK provided much of the auxiliary functionality for the custom tagging approach, any attempt to recreate such functionality would be impractical and unreasonable.

4.1.5 Stemming

Stemming refers to the process of extracting the stem of a given word as discussed in Section 2.1.4. Ultimately as the project progressed Stemming was not utilised within the overall goal of gender bias analysis though it's implementation is available for later use within projected extensions such as Topic Modelling.



```python
def stemming(self, word):
    stemmer = PorterStemmer()
    stemmedWord = stemmer.stem(word)
    return stemmedWord
```

Figure 4.9 - Stemming method

The stemming method within the application utilises the Porter Stemmer from the NLTK suite of libraries. The Porter Stemmer is an implementation of the Porter Algorithm for stemming. (Savoy, 1999)

Problems Faced and Solutions Adopted:

There were no problems faced though this is likely because the stemming process was not taken further, an alternative route for the application was Topic Modelling where stemming could have proved useful with an increased likelihood of problems becoming apparent.

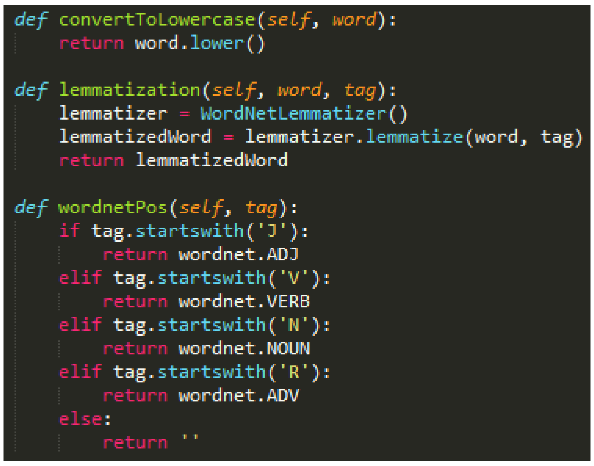
Justification of methods and Tools Adopted:

Another plausible alternative to the Porter Stemmer is the Snowball Stemmer. (Porter, 2001) The Porter Stemmer is based on the Porter Algorithm for Stemming with the Snowball Stemmer extending this Algorithm. Unlike the Porter Stemmer, the Snowball Stemmer is able to accommodate many languages, the Snowball Stemmer also offers improved results in comparison to the Porter Stemmer. The justification for utilising the Porter Stemmer over the Snowball Stemmer comes from it being more popular and there being more support for it, given that such support may need to be utilised as the application extends its capabilities this was chosen.

The use of NLTK provided much of the auxiliary functionality for the Porter Stemmer approach, any attempt to recreate such functionality would be impractical and unreasonable.

4.1.6 Lemmatization

Lemmatization refers to the process of extracting the lemma of a given word as discussed in Section 2.1.4. Ultimately as the project progressed Lemmatization was not utilised within the overall goal of gender bias analysis though it’s implementation is available for later use within projected extensions such as Topic Modelling.



```python
def convertToLowercase(self, word):
    return word.lower()

def lemmatization(self, word, tag):
    lemmatizer = WordNetLemmatizer()
    lemmatizedWord = lemmatizer.lemmatize(word, tag)
    return lemmatizedWord

def wordnetPos(self, tag):
    if tag.startswith('J'):
        return wordnet.ADJ
    elif tag.startswith('V'):
        return wordnet.VERB
    elif tag.startswith('N'):
        return wordnet.NOUN
    elif tag.startswith('R'):
        return wordnet.ADV
    else:
        return ''
```

Figure 4.10 - Methods utilised within lemmatization

The lemmatization method utilises the WordNetLemmatizer from the NLTK suite of libraries. WordNetLemmatizer itself utilises WordNet’s built-in morphy function. By default, WordNetLemmatizer will assume a word is noun if the tag is not specified which is why this has been explicitly expected. The wordnetPos method allows the conversion of treebank tags passed as input to WordNet tags for use by WordNetLemmatizer.

Problems and Solutions:

- Lowercase - Initially the word given as input to the lemmatization method was not converted to lowercase which caused the method to return the word back which caused the test case to fail, after further review of the documentation for WordNetLemmatizer it was decided to lowercase the word which validated correctly.

There was no other problem faced though this is likely because the Lemmatization process was not taken further, an alternative route for the application was Topic Modelling where Lemmatization would have proved useful with an increased likelihood of problems becoming apparent.

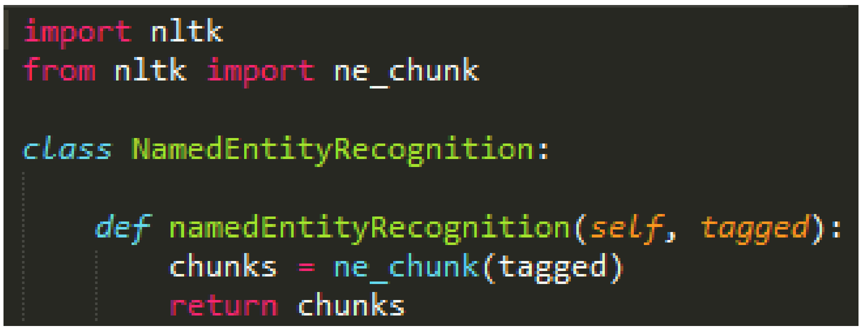
Justification of methods and Tools Adopted:

The justification for using WordNetLemmatizer is that since WordNetLemmatizer is WordNet's built-in morphy function it has access to the large lexical database that is WordNet which includes synonyms and synsets and is seen as a combination of a thesaurus and dictionary. (Miller, Beckwith, & Fellbaum, 1990)

The use of NLTK provided much of the auxiliary functionality for the lemmatization approach, any attempt to recreate such functionality would be impractical and unreasonable.

4.1.7 Named Entity Recognition

Named Entity Recognition refers to the process of extracting Named Entitities from given text, within this project Person named entities were ultimately useful within the overall goal of the project. This topic is discussed in detail within Section 2.1.5.



```python
import nltk
from nltk import ne_chunk

class NamedEntityRecognition:

    def namedEntityRecognition(self, tagged):
        chunks = ne_chunk(tagged)
        return chunks
```

Figure 4.11 - NamedEntityRecognition class

Within the NamedEntityRecognition class there is one method:

- namedEntityRecognition – This method utilises NLTK suite of libraries in-built ne_chunk method which automatically recognises entities through Part of Speech tagged input data. The ne_chunk method acts as a chunker producing 2-level trees. Nodes on Level 1 are outside any chunk e.g. "and/CC". Nodes on Level 2 are inside a chunk with the label of a chunk denoted by the label of the subtree e.g. "Martin/NNP" is part of the "PERSON" chunk.

Problems and Solutions:

Stop Words and Punctuation – As noted within the Text Tokenization phase originally the Tokenization process included the removal of Stop Words and certain punctuation such as commas, full stops, exclamation and questions marks.

The reasoning for this decision was that stop words which are common words occur in such high frequently that they may offer little distinction between texts. They also complicate the process of sentiment extraction should it be done on a word by word level using SentiWordNet. (Esuli & Sebastiani, 2006) The decision to go against the previous reasoning was that sentiment extraction was not done on a word by word level, collection of frequency of words was also not recorded as anticipated early within the project when Text Tokenization was started.

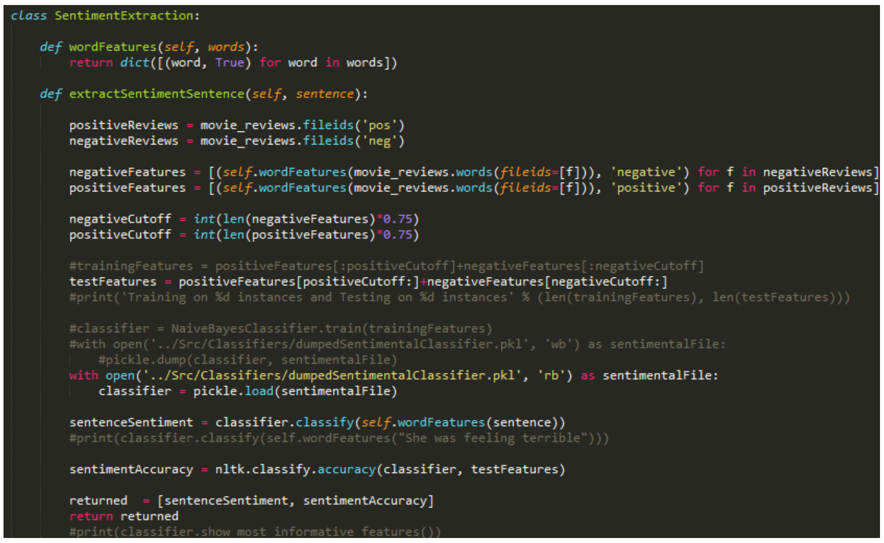
Most notably pertaining to Named Entity Recognition the exclusion of Stop Words and the mentioned punctuation leads to Persons not being accurately detected as such exclusions offer greater context e.g. commas and words such as “and” separate various entities so it can be recognised that the Persons are independent and Persons are not incorrectly jumbled together due to lack of connecting Stop Words. The inclusion of Stop Words also allows at an earlier stage the correct tags to be allocated to words so Persons and other entities can be detected later on.

Justification of methods and Tools Adopted:

The Justification for using NLTK’s in-built ne_chunk method is that it makes little sense in attempting to recreate a named entity recognition chunking mechanism as given the constraints I am working with I would not be able to recreate the functionality that ne_chunk offers.

4.1.8 Sentiment Extraction

Sentiment Extraction refers to the process of extracting sentiment from given text, within this project sentiment is categorised as either positive or negative. This topic is discussed in detail in Section 2.1.6.



```
class SentimentExtraction:

    def wordFeatures(self, words):
        return dict([(word, True) for word in words])

    def extractSentimentSentence(self, sentence):

        positiveReviews = movie_reviews.fileids('pos')
        negativeReviews = movie_reviews.fileids('neg')

        negativeFeatures = [(self.wordFeatures(movie_reviews.words(fileids=[f])), 'negative') for f in negativeReviews]
        positiveFeatures = [(self.wordFeatures(movie_reviews.words(fileids=[f])), 'positive') for f in positiveReviews]

        negativeCutoff = int(len(negativeFeatures)*0.75)
        positiveCutoff = int(len(positiveFeatures)*0.75)

        #trainingFeatures = positiveFeatures[:positiveCutoff]+negativeFeatures[:negativeCutoff]
        testFeatures = positiveFeatures[positiveCutoff:]+negativeFeatures[negativeCutoff:]
        #print('Training on %d instances and Testing on %d instances' % (len(trainingFeatures), len(testFeatures)))

        #classifier = NaiveBayesClassifier.train(trainingFeatures)
        #with open('../Src/Classifiers/dumpedSentimentalClassifier.pkl', 'wb') as sentimentalFile:
            #pickle.dump(classifier, sentimentalFile)
        with open('../Src/Classifiers/dumpedSentimentalClassifier.pkl', 'rb') as sentimentalFile:
            classifier = pickle.load(sentimentalFile)

        sentenceSentiment = classifier.classify(self.wordFeatures(sentence))
        #print(classifier.classify(self.wordFeatures("She was feeling terrible")))

        sentimentAccuracy = nltk.classify.accuracy(classifier, testFeatures)

        returned  = [sentenceSentiment, sentimentAccuracy]
        return returned
        #print(classifier.show_most_informative_features())
```

Figure 4.12 - SentimentExtraction class

Within the Sentiment Extraction class there are two methods:

- wordFeatures – This method implements the bag-of-words model which constructs a word presence feature set from all the words of an instance, it converts all the list of words into a dictionary (mapping keys to values e.g. mapping positive and negative reviews) feature set which is what the classifier expects as input. Supervised machine learning as done with the classifier in this example requires this feature extractor to capture information about input that will be used for classification. (S, Raj, & S.Rajaraajeswari, 2016)
- extractSentimentSentence– This method takes as input the sentences. This method utilises the movie_reviews corpus available through NLTK to aggregate all negative and positive reviews to train a Naïve Bayes Classifier on a 75:25 training:testing split. Using the Pickle Python module as mentioned in Chapter 2.3, the classifier is serialized and saved into a pickle file so that there is no need to subsequently retrain the classifier. This classifier is then utilised to classify the given sentences passed as input. This sentiment is returned alongside a general accuracy of the trained classifier on the given test data.

Problems Faced and Solutions Adopted:

Sentiment Analysis – Due to the ambiguities associated with Natural Languages within testing there were certain inaccuracies with sentiment not being classified correctly, these issues were resolved for the given test cases through experimenting with the training:test split though there is an accepted understanding that there could be many cases where a more complex classifier could be applied though for this application at this stage the given Naïve Bayes Classifier is satisfactory.

Justification of methods and Tools Adopted:

The use of Pickle to serialize and save the trained classifier to a Pickle file allows the trained classifier to be loaded from the pickle file instead of the need to train the classifier again which increased performance.

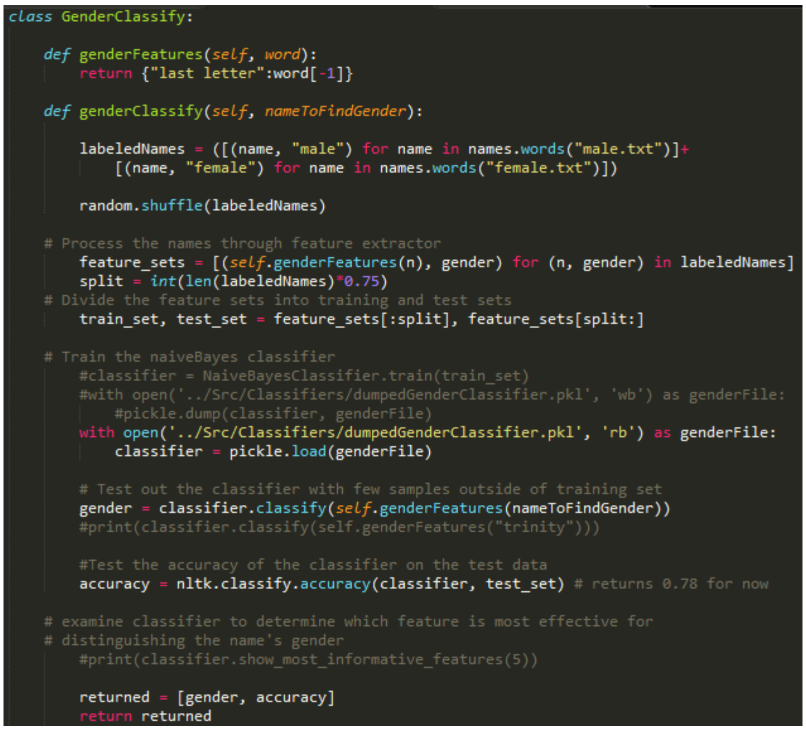
The use of the 75:25 training and test split was judged based on experimentation with the classifier and the given test results alongside an understanding of non-movie related data within unit testing to prevent the classifier from become too specific.

The use of the Naïve Bayes Classifier provided a simple mechanism for supervised learning which satisfied objectives.

The use of NLTK provided much of the auxiliary functionality for the sentiment extraction approach, any attempt to recreate such functionality would be impractical and unreasonable.

4.1.9 Gender Classification

Gender Classification similar to Sentiment Extraction uses a bag of words model to train a classifier by given training data which are files containing a list of female and male names so that the classifier can accurately deduce exact matches or outside cases using probablity. Section 4.1.7 addressed the extraction of person named entities which would need to be classified by gender within this section.



```python
class GenderClassify:

    def genderFeatures(self, word):
        return {"last letter":word[-1]}

    def genderClassify(self, nameToFindGender):

        labeledNames = ([(name, "male") for name in names.words("male.txt")]+
            [(name, "female") for name in names.words("female.txt")])

        random.shuffle(labeledNames)

    # Process the names through feature extractor
        feature_sets = [(self.genderFeatures(n), gender) for (n, gender) in labeledNames]
        split = int(len(labeledNames)*0.75)
    # Divide the feature sets into training and test sets
        train_set, test_set = feature_sets[:split], feature_sets[split:]

    # Train the naiveBayes classifier
        #classifier = NaiveBayesClassifier.train(train_set)
        #with open('../Src/Classifiers/dumpedGenderClassifier.pkl', 'wb') as genderFile:
            #pickle.dump(classifier, genderFile)
        with open('../Src/Classifiers/dumpedGenderClassifier.pkl', 'rb') as genderFile:
            classifier = pickle.load(genderFile)

        # Test out the classifier with few samples outside of training set
        gender = classifier.classify(self.genderFeatures(nameToFindGender))
        #print(classifier.classify(self.genderFeatures("trinity")))

        #Test the accuracy of the classifier on the test data
        accuracy = nltk.classify.accuracy(classifier, test_set) # returns 0.78 for now

    # examine classifier to determine which feature is most effective for
    # distinguishing the name's gender
        #print(classifier.show_most_informative_features(5))

        returned = [gender, accuracy]
        return returned
```

Figure 4.13 - GenderClassify class

Within the Gender Classify class there are two methods:

- genderFeatures – Just as in the Sentiment Extraction there is a feature extractor method so too within gender classification there is a feature extractor which is this method, this method extracts the final letter of the names from the corpus used for training and testing and uses this as input for the classifier.
- genderClassify – This method takes a name as input, it then using the names corpus trains the classifier differentiating male and females based on the last letters determined from the feature extractor. This classifier is then used to classify the input name given and the gender is output alongside the accuracy of the classifier on the testing data. Using Pickle, the classifier is serialized and saved in a pickle file where it can be subsequently loaded from so that there is no need to retrain the classifier. A 75:25 training:split was settled upon based on experimentation to improve accuracy whilst not becoming too specific.

Problems and Solutions:

Gender Neutral Names – There are some names that are Gender neutral which presents a problem considering that Gender Classification requires an output of either Male or Female, since many of these gender neutral names are surnames the allowance of forenames within the method allows a more accurate distinction to be made since surnames are more likely to be gender neutral, such issues of gender neutrality are not restricted to English names.

Justification of methods and Tools Adopted:

The use of Pickle to serialize and save the trained classifier allows the trained classifier to be loaded from the pickle file instead of the need to retrain the classifier again which increased performance.

The use of the names corpus allows a ready solution to the problem of providing training and testing data. To attempt to recreate my own corpus would be impractical and would take away from achieving objective which is why the names corpus was utilised.

The use of the Naïve Bayes Classifier provided a simple mechanism for supervised learning which satisfied objectives. (Manning, Raghavan, & Schütze, 2009)

The use of NLTK provided much of the auxiliary functionality for the gender classification approach, any attempt to recreate such functionality would be impractical and unreasonable.

4.1.10 Gender Bias Analysis

Gender Bias Analysis is the process of detecting gender bias by bringing together previously implemented functionality. It is discussed thoroughly within this section.

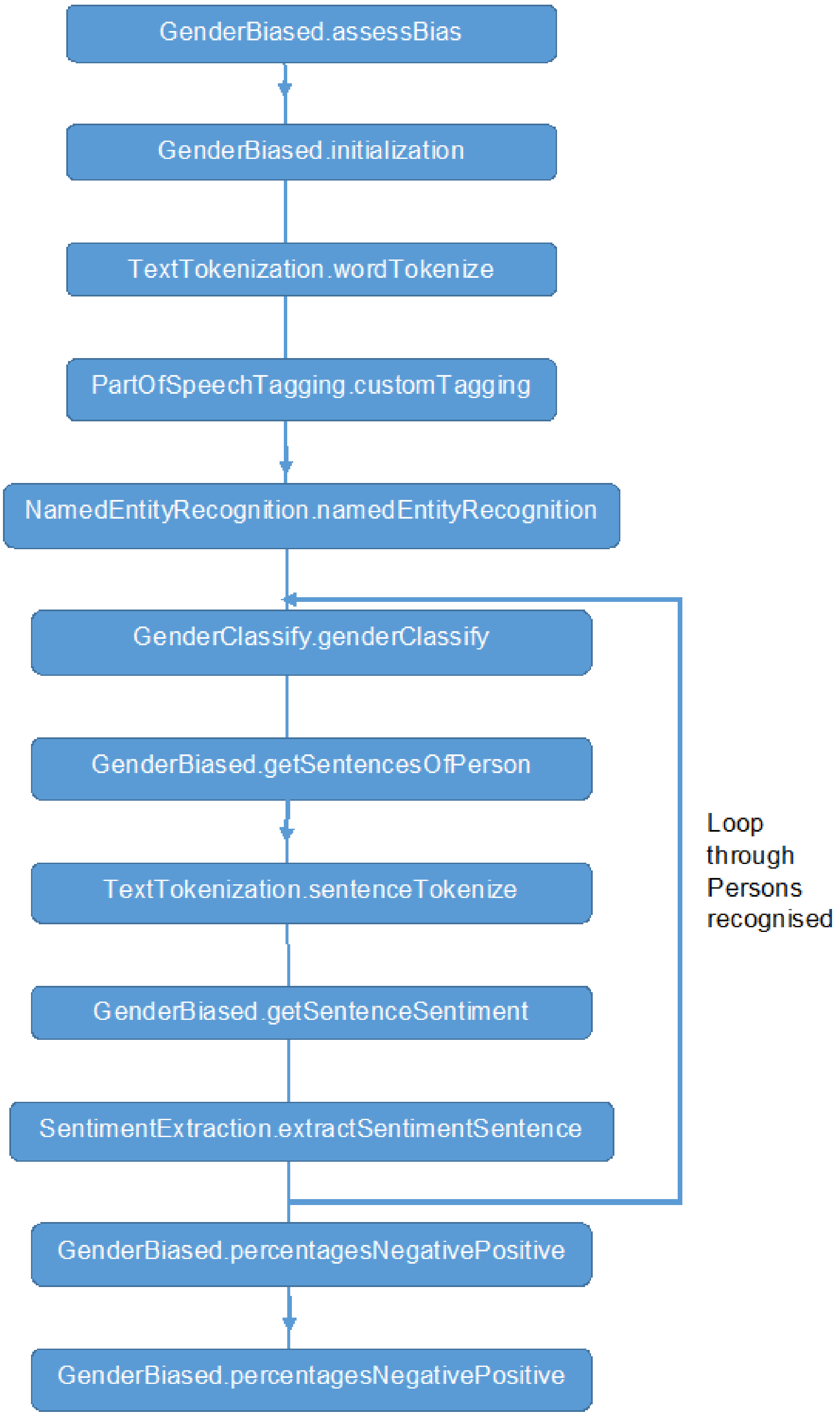
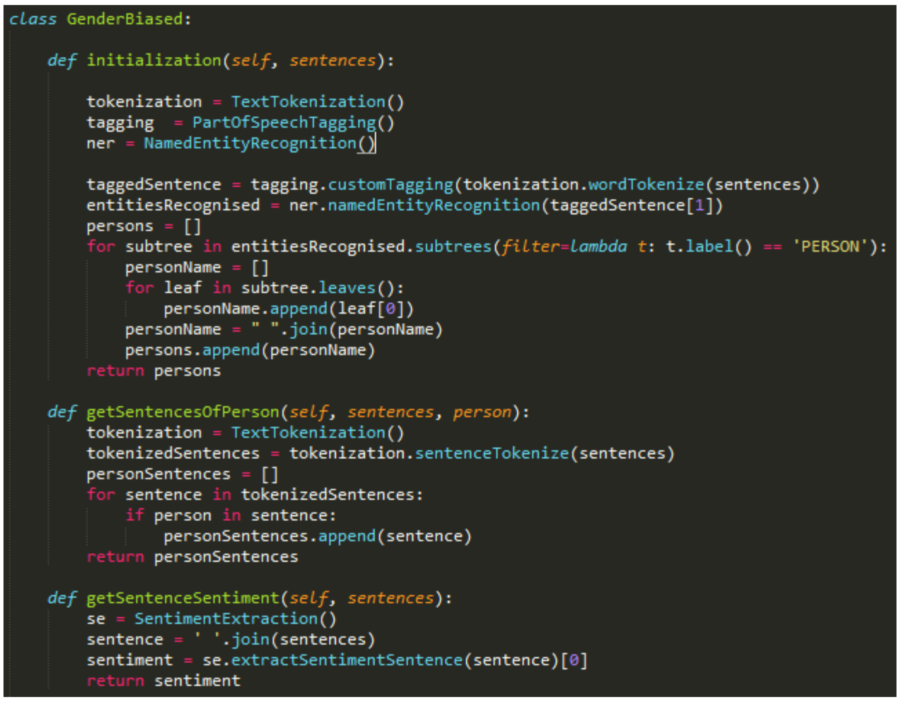
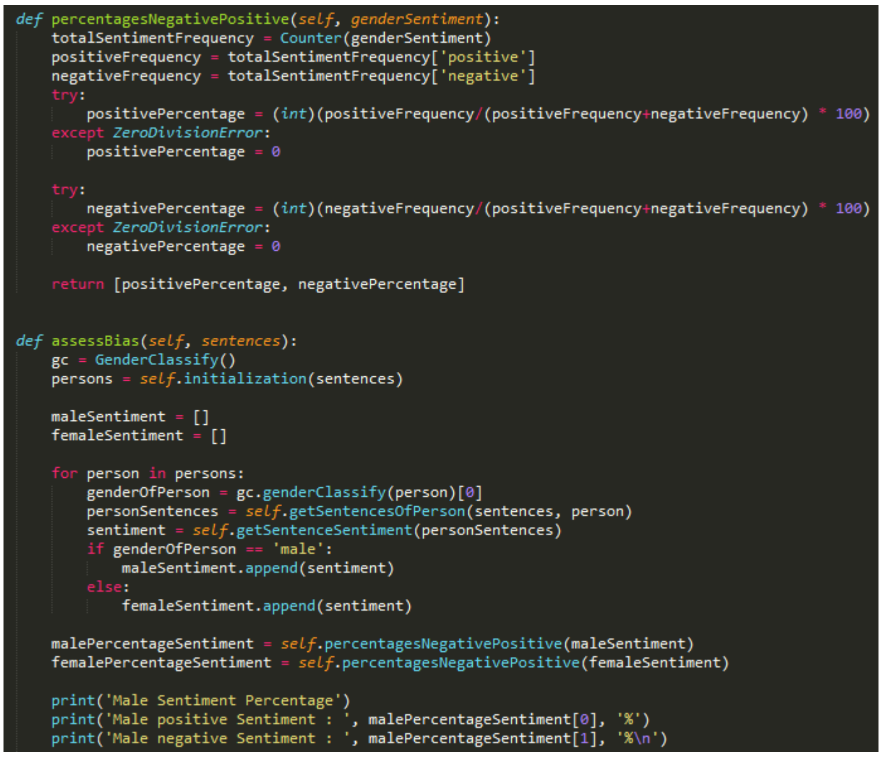

GenderBiased.assessBias
GenderBiased.initialization
TextTokenization.wordTokenize
PartOfSpeechTagging.customTagging
NamedEntityRecognition.namedEntityRecognition
GenderClassify.genderClassify
GenderBiased.getSentencesOfPerson
TextTokenization.sentenceTokenize
GenderBiased.getSentenceSentiment
SentimentExtraction.extractSentimentSentence
GenderBiased.percentagesNegativePositive
GenderBiased.percentagesNegativePositive
Loop through Persons recognised

Figure 14.14 – Method Execution Sequence



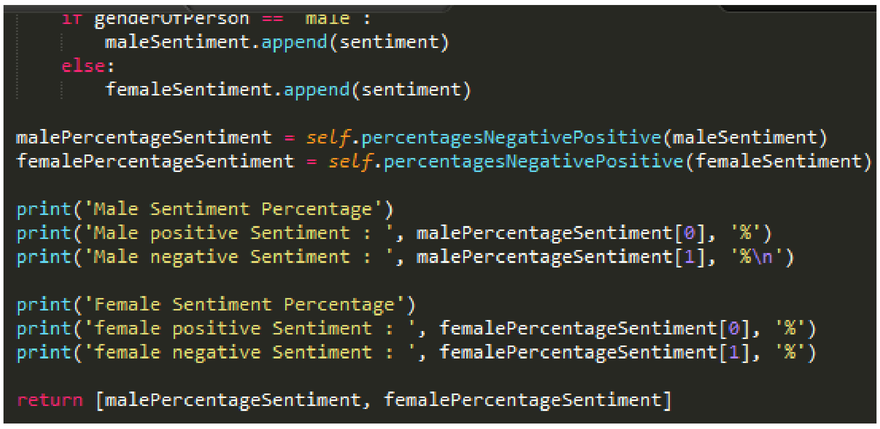
```python
class GenderBiased:

    def initialization(self, sentences):

        tokenization = TextTokenization()
        tagging  = PartOfSpeechTagging()
        ner = NamedEntityRecognition()

        taggedSentence = tagging.customTagging(tokenization.wordTokenize(sentences))
        entitiesRecognised = ner.namedEntityRecognition(taggedSentence[1])
        persons = []
        for subtree in entitiesRecognised.subtrees(filter=lambda t: t.label() == 'PERSON'):
            personName = []
            for leaf in subtree.leaves():
                personName.append(leaf[0])
            personName = " ".join(personName)
            persons.append(personName)
        return persons

    def getSentencesOfPerson(self, sentences, person):
        tokenization = TextTokenization()
        tokenizedSentences = tokenization.sentenceTokenize(sentences)
        personSentences = []
        for sentence in tokenizedSentences:
            if person in sentence:
                personSentences.append(sentence)
        return personSentences

    def getSentenceSentiment(self, sentences):
        se = SentimentExtraction()
        sentence = ' '.join(sentences)
        sentiment = se.extractSentimentSentence(sentence)[0]
        return sentiment
```

Figure 4.15 - GenderBiased class



```python
def percentagesNegativePositive(self, genderSentiment):
    totalSentimentFrequency = Counter(genderSentiment)
    positiveFrequency = totalSentimentFrequency['positive']
    negativeFrequency = totalSentimentFrequency['negative']
    try:
        positivePercentage = (int)(positiveFrequency/(positiveFrequency+negativeFrequency) * 100)
    except ZeroDivisionError:
        positivePercentage = 0

    try:
        negativePercentage = (int)(negativeFrequency/(positiveFrequency+negativeFrequency) * 100)
    except ZeroDivisionError:
        negativePercentage = 0

    return [positivePercentage, negativePercentage]


def assessBias(self, sentences):
    gc = GenderClassify()
    persons = self.initialization(sentences)

    maleSentiment = []
    femaleSentiment = []

    for person in persons:
        genderOfPerson = gc.genderClassify(person)[0]
        personSentences = self.getSentencesOfPerson(sentences, person)
        sentiment = self.getSentenceSentiment(personSentences)
        if genderOfPerson == 'male':
            maleSentiment.append(sentiment)
        else:
            femaleSentiment.append(sentiment)

    malePercentageSentiment = self.percentagesNegativePositive(maleSentiment)
    femalePercentageSentiment = self.percentagesNegativePositive(femaleSentiment)

    print('Male Sentiment Percentage')
    print('Male positive Sentiment : ', malePercentageSentiment[0], '%')
    print('Male negative Sentiment : ', malePercentageSentiment[1], '%\n')
```

Figure 4.16 - GenderBiased class cont.



```
    if genderOfPerson == 'male':
        maleSentiment.append(sentiment)
    else:
        femaleSentiment.append(sentiment)

malePercentageSentiment = self.percentagesNegativePositive(maleSentiment)
femalePercentageSentiment = self.percentagesNegativePositive(femaleSentiment)

print('Male Sentiment Percentage')
print('Male positive Sentiment : ', malePercentageSentiment[0], '%')
print('Male negative Sentiment : ', malePercentageSentiment[1], '%\n')

print('Female Sentiment Percentage')
print('female positive Sentiment : ', femalePercentageSentiment[0], '%')
print('female negative Sentiment : ', femalePercentageSentiment[1], '%')

return [malePercentageSentiment, femalePercentageSentiment]
```

Figure 4.17 - GenderBiased class cont.

Within the Gender Biased class there are five methods:

- initialization – This method does Word Tokenization, Part of Speech Tagging and Named Entity Recognition from sentences passed as input returning recognised Person entities. Word Tokenization is called from TextTokenization.wordTokenize which is then used as input for tagging. Part of Speech Tagging is called from PartOfSpeechTagging.customTagging which is used as input for named entity recognition. Named Entity Recognition is called from NamedEntityRecognition.namedEntityRecognition and returns all named entities, then the method filters and returns only PERSON entities recognised ignoring all others.
- getSentencesOfPerson – This method takes in given text and a Person as input and returns the sentences within the given text related to the person. This method calls TextTokenization.sentenceTokenize which covers the functionality for sentence tokenization and then loops through all sentences and returns all those that concern the Person given as input.
- getSentenceSentiment – This method takes in sentences as input and return the overall sentiment. This method calls SentimentExtraction.extractSentimentSentence which covers the functionality for extracting sentiment from a given sentence which is returned.
- percentagesNegativePositive – This method takes as input the genderSentiment and returns the percentage of the genderSentiment that is positive and negative. The method returns what percentage of the total Sentiment is positive and what percentage is negative.

- assessBias – This method takes as input sentences and uses the above methods to recognise Persons, their gender and difference in the overall percentage of positive vs negative sentiment for males and females to determine bias. This method calls initialization and then for each Person named entity recognised calls GenderClassify.genderClassify which encapsulates the functionality of gender classification, then getSentencesOfPerson is called to get all sentences related to that person, then getSentenceSentiment is called to get the sentiment of such sentences, such sentiment is recorded and categorised by gender with percentagesNegativePositive called and male sentiment passed as input and then female sentiment passed as input to percentagesNegativePositive independently, the results are printed to the console and returned.

Problems Faced and Solutions Adopted:

Issues with other components – Within this phase issues with previous phases became apparent as this stage attempted to bring together all the previous phases together to provide meaningful results and complete the project. The solution to this was to tweak elements of these other phases until issues were fixed.

Justification of methods and Tools Adopted:

The Gender Bias method utilised presents an abstract view of the project as a whole tailoring to completing the overall goal by bringing together other phases to provide meaningful results.

Chapter 5 Testing and Results

5.1 Testing Strategy and Plan Adopted

For any project that includes Development and Implementation there is a requirement on the production of a consistent Testing approach which can validate and verify all developments made within the project. The Test Approach adopted with a project is dependent on the project itself as just as projects can be varied so too testing strategies can be varied.

Within large projects where a traditional waterfall methodology is adopted the Testing strategy may commence following development whereas an Agile project adopts a Test Driven Development approach with Testing done alongside development.

Just as Testing approaches can be dependent on factors such as the organisation, project size and methodology chosen so too the testing methods can also vary. A website may require testing methods such as Penetrative Testing which would not be conducted within other projects that are not exposed to any security threat. Platform Independence Testing may be conducted should the project implementation aim to seek platform independence with the product working consistently across multiple platform though such Testing would be useless should the implementation be purposely platform dependent.

There are many types of Testing with specific Testing methods utilised dependent on the given project. Below are the two types of Testing included within the project:

- Unit Testing – Testing method whereby individual units of code are tested independently for functional correctness of standalone modules.
- User Acceptance Testing – Testing method whereby the customer/ end user validates the product against requirements.

The Testing strategy that I have adopted includes Unit Testing and User Acceptance Testing. Unit Testing is utilised to validate individual components of the project with User Acceptance Testing accepting each of these components from the project supervisor and customer's perspective.

The plan that I have outlined includes tests for the following aspects of the project:

1. Text tokenization
2. Part of Speech Tagging
3. Stemming
4. Lemmatization
5. Named Entity Recognition
6. Sentiment Extraction
7. Gender Classification
8. Gender Bias

The Tests outlined will seek to ensure that the above aspects of the project will have been verified and validated to an appropriate measure though given that testing will be done on Natural Language which can be ambiguous and obscure it has been deemed acceptable to assume that there will be many outside cases that will not be satisfied by the unit tests specified, within the grand scheme of the testing within this project it is only necessary to prove that Gender Bias can be detected on textual data using Natural Language Processing techniques.

5.2 Natural Language Processing Techniques Validation – Unit Testing

This section covers the validation of all the Natural Language Processing techniques implemented. Each Subsection includes notes on specific input for tests and expected output, alongside Actions Taken to achieve validation and an Analysis of Test Results.

For each validated Natural Language Processing technique implemented there are unique tests which are designed to validate the correctness of the system given plausible obstacles. Below is a list of the Test Cases alongside the input and output for unit tests within these Test Cases:

- Text Tokenization – There are two unit tests testSentenceTokenization and testWordTokenization. For testSentenceTokenization the input data is “Hello World. How are you?” and the expected output is two sentences “Hello World.” and “How are you?”. For testWordTokenization the input data is “Hello World’s. How are’ You today in Calledonian-Thistle! 12345” and the expected output is 10 words “Hello”, “World’s”, “.”, “How”, “are’”, “You”, “today”, “in”, “Calledonian-Thistle” and “!”.

- Part of Speech Tagging – There is one unit test testTagging. The input data for testTagging is “Tom thinks John is terrible. John thinks Tom is great.”. The expected output for testTagging is the general accuracy of the method which should be greater than 0.90 or 90% alongside the combination of words and part of speech tags of the input given i.e. [('Tom', 'NNP'), ('thinks', 'VBZ'), ('John', 'NNP'), ('is', 'VBZ'), ('terrible', 'JJ'), ('.', '.'), ('John', 'NNP'), ('thinks', 'VBZ'), ('Tom', 'NNP'), ('is', 'VBZ'), ('great', 'JJ'), ('.', '.')].

- Stemming – There is one unit test testStemming. The input data for testStemming is 4 words “Walking”, “Walks”, “walked” and “walk”. The expected output for testStemming is one word “walk”.

- Lemmatization – There is one unit test testLemmatization. The input data for testLemmatization is the word “Is” and the tag “V” indicating it’s within a verb context. The expected output for testLemmatization is the word “be”.

- Named Entity Recognition – There is one unit test testNER. The input data for testNER is “Yesterday Ivan Blake, Abdul Haseeb Hussain and Adam Allen Jones went to work for the summer after working at Google. Johnathon Allen, Mahatma Ghandi and Tom Smith are also planning to go tomorrow to Apple. Anna Rose and Fiona Smith.”. The expected output for testNER is all the Persons recognised “Ivan Blake”, “Abdul Haseeb Hussain”, “Adam Allen Jones”, “Johnathon Allen”, “Mahatma Ghandi”, “Tom Smith”, “Anna Rose” and “Fiona Smith”.

- Sentiment Extraction – There are five unit tests testGeneralSentimentAccuracy, testStronglyPositiveSentiment, testStronglyNegativeSentiment, testMildlyPositiveSentiment and testMildlyNegativeSentiment. The expected input for testGeneralSentimentAccuracy is “The prom was pretty good and worthwhile, the expected output for testGeneralSentimentAccuracy is that the accuracy for the sentiment extraction method be greater than 0.65 or 65%. The expected input for testStronglyPositiveSentiment is “The prom was set in a beautiful venue with a marvellous stage” with the expected output “positive”. The expected input for testStronglyNegativeSentiment is “The prom was set in a disastrous venue with a tacky stage” with the expected output “negative”. The expected input for testMildlyPositiveSentiment is “The prom was great though it was slightly long” with the expected output “positive”. The expected input for testMildlyNegativeSentiment is “The prom was terrible though finished quick” with the expected output “negative”.

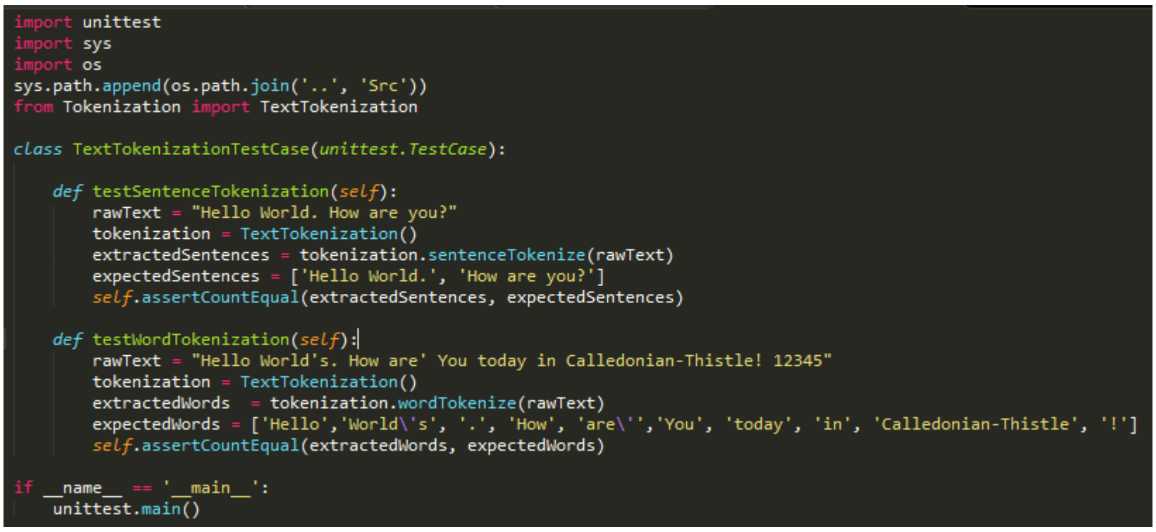
- Gender Classification – There are three unit tests testGeneralGenderAccuracy, testMale and testFemale. The input for testGeneralGenderAccuracy is “Adam” with expected output the general accuracy for the gender classification method which should be greater than 0.7 or 70%. The input for testMale is the names “Abdul”, “Apostolos”, “John” and “Thomas” with the expected output of “male”, “male”, “male” and “male” representing the gender of the names. The input for testFemale is the names “Anna”, “Katrina”, “Sabrina” and “Samantha” with the expected output of “female”, “female”, “female” and “female” representing the gender of the names.

- Gender Bias Validation – There are four unit tests testPositiveMaleBias, testNegativeMaleBias, testPositiveFemaleBias and testNegativeFemaleBias. The input for all unit tests is “Thomas Smith was a man of upstanding character and grace. Anna Smith was a women of terrible character.”. The expected output for testPositiveMaleBias is 100 indicating the percentage of male positivity. The expected output for testNegativeMaleBias is 0 indicating the percentage of male negativity. The expected output for testPositiveFemaleBias is 0 indicating the percentage of female positivity. The expected output for testNegativeFemaleBias is 100 indicating the percentage of female negativity.

5.2.1 Text Tokenization Validation

This subsection covers the validation of the Text Tokenization aspect of the project.

For the testSentenceTokenization test the input is stored within the variable rawText with the output stored within the variable extractedSentences and expected output stored within the variable expectedSentences. For the testWordTokenization test the input is stored within the variable rawText with the output stored within the variable extractedWords and expected output stored within the variable expectedWords.



```python
import unittest
import sys
import os
sys.path.append(os.path.join('..', 'Src'))
from Tokenization import TextTokenization

class TextTokenizationTestCase(unittest.TestCase):

    def testSentenceTokenization(self):
        rawText = "Hello World. How are you?"
        tokenization = TextTokenization()
        extractedSentences = tokenization.sentenceTokenize(rawText)
        expectedSentences = ['Hello World.', 'How are you?']
        self.assertCountEqual(extractedSentences, expectedSentences)

    def testWordTokenization(self):
        rawText = "Hello World's. How are' You today in Calledonian-Thistle! 12345"
        tokenization = TextTokenization()
        extractedWords  = tokenization.wordTokenize(rawText)
        expectedWords = ['Hello','World\'s', '.', 'How', 'are\'','You', 'today', 'in', 'Calledonian-Thistle', '!']
        self.assertCountEqual(extractedWords, expectedWords)

if __name__ == '__main__':
    unittest.main()
```

Figure 5.1 - TextTokenizationTestCase unit test

Please refer to the subsection introduction for an understanding of input and output representations within the above test case.

Actions Taken:

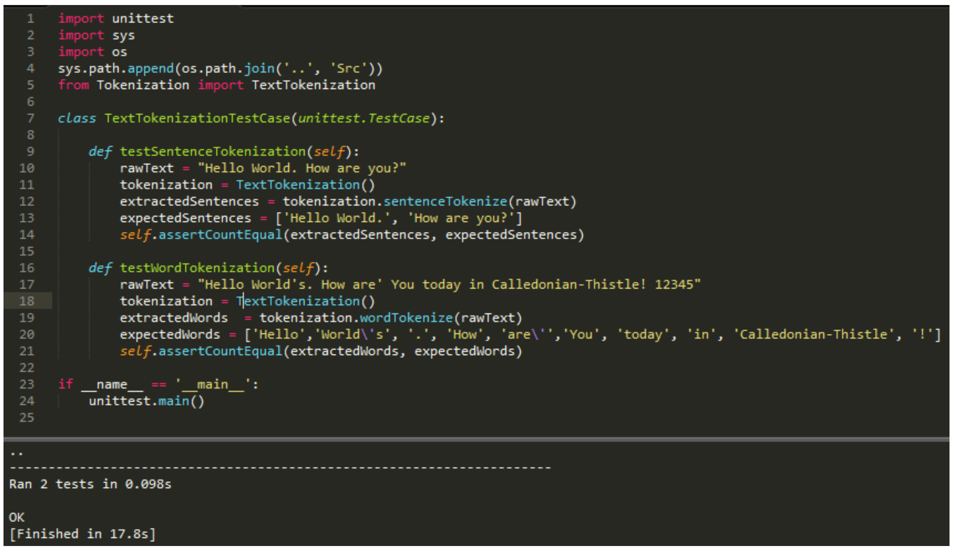
The TextTokenizationTestCase test case tests the sentence and word level tokenization within the project. The test case exhibits two methods:

- testSentenceTokenization - The sentence tokenization method is given the text “Hello World. How are you?” as input and the expected output are two sentences, “Hello World.” and “How are you?” The utilisation of the question marks tests the common occurrence of alternative end of sentence punctuation appearing at the end of given sentence e.g. a question ends with question mark and an assertion may end with an exclamation mark.

- testWordTokenization - The word tokenization method is given the text “Hello World’s. How are’ You today in Calledonian-Thistle! 12345” as input and the expected output is 10 words “Hello”, “World’s”, “.”, “How”, “are’”, “You”, “today”, “in”, “Calledonian-Thistle” and “!”. From a testing perspective the text input is sufficient as it covers the most likely obstacles to adequate Persons Named Entity Recognition which is one of the high level uses of Text Tokenization, another high level use of Text Tokenization is Sentiment Extraction which could be inhibited by incorrectly tokenized words though the given input text test adequately validates common obstructions to word tokenization. Within testing there needs to be a balance when constructing tests covering the majority of potential of obstacles as given the nature of Natural Languages there will be outside cases not covered by this specific test which is acceptable. The use of hyphens within input text tests whether hyphened words are being recognised as one which is useful later when Text Tokenization is utilised within Persons Named Entity Recognition and Persons names are hyphened. The use of numbers at the end of the text tests whether number are incorrectly being detected as words. The use of the exclamation mark tests whether end of sentence punctuation is being incorrectly counted as being part of the word. The use of apostrophes test whether words with apostrophes are being detected within their full form. Certain punctuation such as exclamation marks, question marks, commas and full stops are included as part of the word tokenization method since they are utilised within further objectives for providing context.

Analysis of Test results:

The initial two dots printed within the command line in the bottom section of the image below indicates that both tests passed. Which is indicated later within the command line as it is printed “Ran 2 tests” and later “OK”.



```
import unittest
import sys
import os
sys.path.append(os.path.join('..', 'Src'))
from Tokenization import TextTokenization

class TextTokenizationTestCase(unittest.TestCase):

    def testSentenceTokenization(self):
        rawText = "Hello World. How are you?"
        tokenization = TextTokenization()
        extractedSentences = tokenization.sentenceTokenize(rawText)
        expectedSentences = ['Hello World.', 'How are you?']
        self.assertCountEqual(extractedSentences, expectedSentences)

    def testWordTokenization(self):
        rawText = "Hello World's. How are' You today in Calledonian-Thistle! 12345"
        tokenization = TextTokenization()
        extractedWords  = tokenization.wordTokenize(rawText)
        expectedWords = ['Hello','World\'s', '.', 'How', 'are\'','You', 'today', 'in', 'Calledonian-Thistle', '!']
        self.assertCountEqual(extractedWords, expectedWords)

if __name__ == '__main__':
    unittest.main()

```

```
..
----------------------------------------------------------------------
Ran 2 tests in 0.098s

OK
[Finished in 17.8s]
```

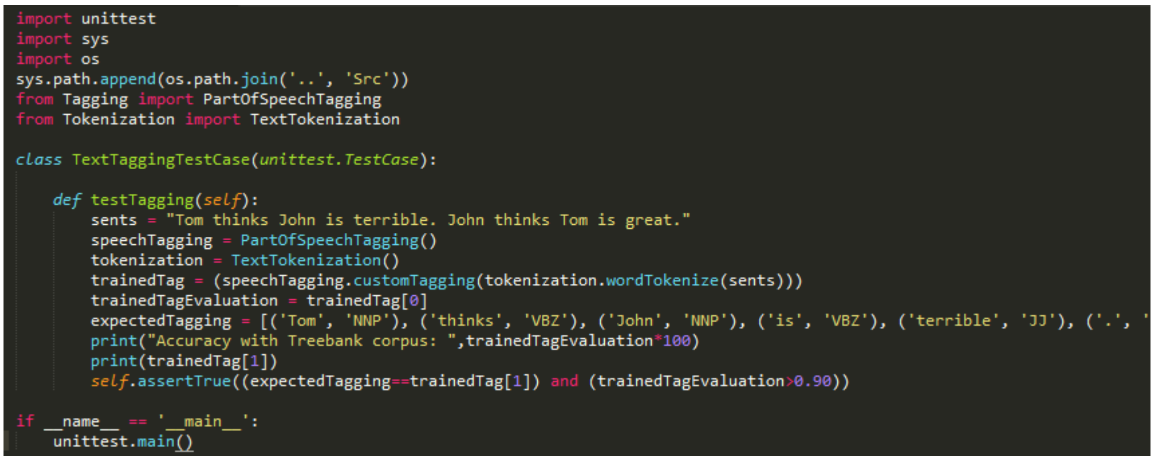
Figure 5.2 - TextTokenizationTestCase unit test results

Both tests within the TextTokenizationTestCase passed with expected output matching actual output.

5.2.2 Part of Speech Tagging Validation

This subsection covers the validation of the Part of Speech Tagging aspect of the project.

For the testTagging test the input is store within the variable sents and output stored within the variables trainedTag and trainedTagEvaluation. The expected output is greater than 0.90 for the trainedTagEvaluation and the variable expectedTagging stores the expected tags.



```
import unittest
import sys
import os
sys.path.append(os.path.join('..', 'Src'))
from Tagging import PartOfSpeechTagging
from Tokenization import TextTokenization

class TextTaggingTestCase(unittest.TestCase):

    def testTagging(self):
        sents = "Tom thinks John is terrible. John thinks Tom is great."
        speechTagging = PartOfSpeechTagging()
        tokenization = TextTokenization()
        trainedTag = (speechTagging.customTagging(tokenization.wordTokenize(sents)))
        trainedTagEvaluation = trainedTag[0]
        expectedTagging = [('Tom', 'NNP'), ('thinks', 'VBZ'), ('John', 'NNP'), ('is', 'VBZ'), ('terrible', 'JJ'), ('.', '
        print("Accuracy with Treebank corpus: ",trainedTagEvaluation*100)
        print(trainedTag[1])
        self.assertTrue((expectedTagging==trainedTag[1]) and (trainedTagEvaluation>0.90))

if __name__ == '__main__':
    unittest.main()
```

Figure 5.3 - TextTaggingTestCase unit test

Please refer to the subsection introduction for an understanding of input and output representations within the above test case.

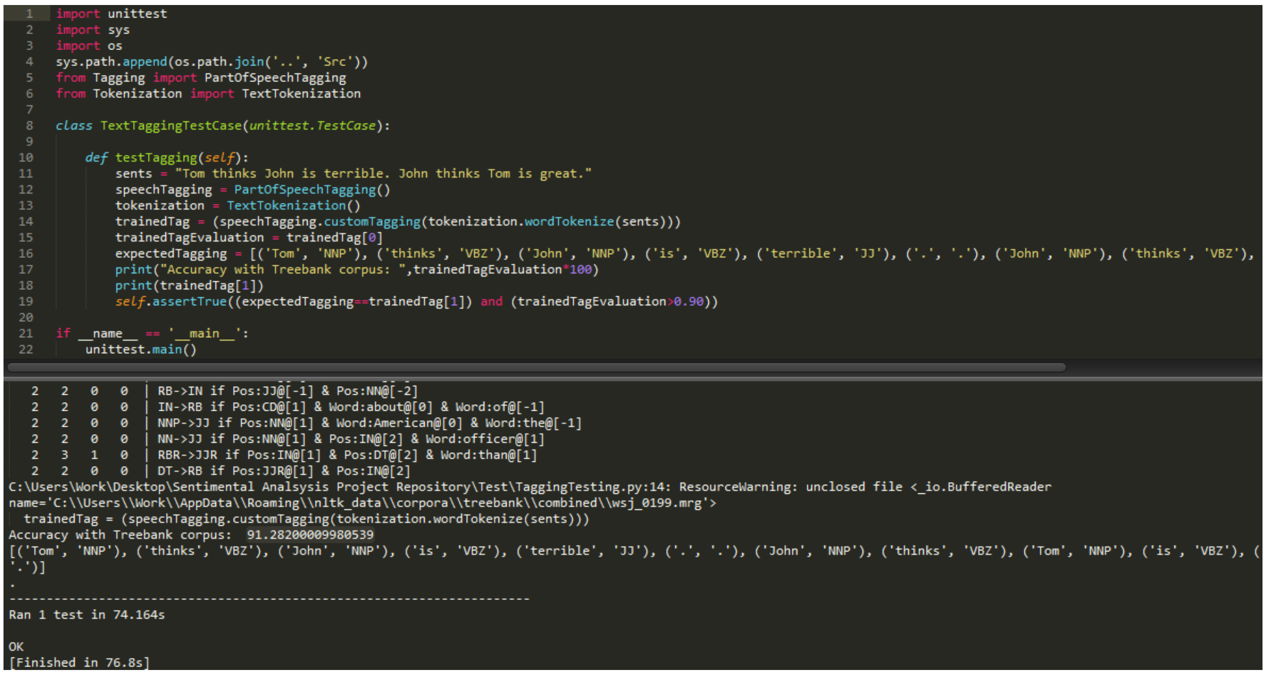
Actions Taken:

The TextTaggingTestCase test case tests the Part of Speech tagging within the project. The test case exhibits one method:

- testTagging – The tagging method is given the text “Tom thinks John is terrible. John thinks Tom is great.” word tokenized as input. As output the correct tagging sequence is expected alongside an evaluation of how the Part of Speech Tagging method evaluates against the given test data when trained against the given training data. The method is expected to deliver the correct tags to the relatively simple input given which will be [('Tom', 'NNP'), ('thinks', 'VBZ'), ('John', 'NNP'), ('is', 'VBZ'), ('terrible', 'JJ'), ('.', '.'), ('John', 'NNP'), ('thinks', 'VBZ'), ('Tom', 'NNP'), ('is', 'VBZ'), ('great', 'JJ'), ('.', '.')]. The method is also expected to deliver an evaluation on the accuracy of the method on the given test data with an accuracy greater than 90%. It is expected that 100% accuracy cannot be achieved but an accuracy exhibiting that a majority would validate the method given the complexities and ambiguities associated with working upon Natural Language.

Analysis of Test results:

The dot printed on its own within the command line in the bottom section of the image below indicates that the one test passed. Which is indicated later within the command line as it is printed “Ran 1 test” and later “OK”. Additionally, within the command line the tags output from the Part of Speech Tagging method and the accuracy of the method are printed.



```python
import unittest
import sys
import os
sys.path.append(os.path.join('..', 'Src'))
from Tagging import PartOfSpeechTagging
from Tokenization import TextTokenization

class TextTaggingTestCase(unittest.TestCase):

    def testTagging(self):
        sents = "Tom thinks John is terrible. John thinks Tom is great."
        speechTagging = PartOfSpeechTagging()
        tokenization = TextTokenization()
        trainedTag = (speechTagging.customTagging(tokenization.wordTokenize(sents)))
        trainedTagEvaluation = trainedTag[0]
        expectedTagging = [('Tom', 'NNP'), ('thinks', 'VBZ'), ('John', 'NNP'), ('is', 'VBZ'), ('terrible', 'JJ'), ('.', '.'), ('John', 'NNP'), ('thinks', 'VBZ'),
        print("Accuracy with Treebank corpus: ",trainedTagEvaluation*100)
        print(trainedTag[1])
        self.assertTrue((expectedTagging==trainedTag[1]) and (trainedTagEvaluation>0.90))

if __name__ == '__main__':
    unittest.main()
```

```
  2   2   0   0  | RB->IN if Pos:JJ@[-1] & Pos:NN@[-2]
  2   2   0   0  | IN->RB if Pos:CD@[1] & Word:about@[0] & Word:of@[-1]
  2   2   0   0  | NNP->JJ if Pos:NN@[1] & Word:American@[0] & Word:the@[-1]
  2   2   0   0  | NN->JJ if Pos:NN@[1] & Pos:IN@[2] & Word:officer@[1]
  2   3   1   0  | RBR->JJR if Pos:IN@[1] & Pos:DT@[2] & Word:than@[1]
  2   2   0   0  | DT->RB if Pos:JJR@[1] & Pos:IN@[2]
C:\Users\Work\Desktop\Sentimental Analsysis Project Repository\Test\TaggingTesting.py:14: ResourceWarning: unclosed file <_io.BufferedReader
name='C:\\Users\\Work\\AppData\\Roaming\\nltk_data\\corpora\\treebank\\combined\\wsj_0199.mrg'>
  trainedTag = (speechTagging.customTagging(tokenization.wordTokenize(sents)))
Accuracy with Treebank corpus:  91.28200009980539
[('Tom', 'NNP'), ('thinks', 'VBZ'), ('John', 'NNP'), ('is', 'VBZ'), ('terrible', 'JJ'), ('.', '.'), ('John', 'NNP'), ('thinks', 'VBZ'), ('Tom', 'NNP'), ('is', 'VBZ'), (
'.')]
.
----------------------------------------------------------------------
Ran 1 test in 74.164s

OK
[Finished in 76.8s]
```

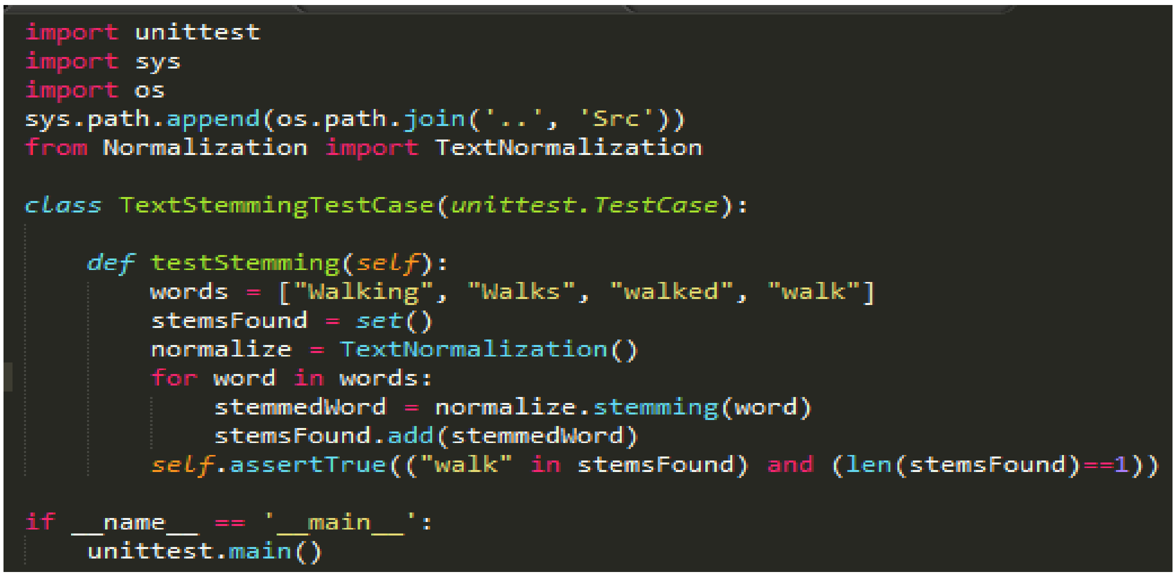
Figure 5.4 - TextTaggingTestCase unit test results

The test within the TextTaggingTestCase passed with the correct tags and the accuracy of the trained method greater than 90% on the given test data. The corpus used for the training and test data is the treebank corpus, usage of this corpus on an 80:20 training : test split achieves an accuracy of 91.3%. As mentioned in the Development and Implementation chapter other corpuses were tested though issues with performance and accuracy led to this corpus being utilised based on the results within unit testing.

5.2.3 Stemming Validation

This subsection covers the validation of the Stemming aspect of the project. Please find an image of the Test Case below.

For the testStemming test the input is stored within the variable words, the output is stored within the variables stemsFound. The expected output is outlined within the parameter of the assertTrue method.



```python
import unittest
import sys
import os
sys.path.append(os.path.join('..', 'Src'))
from Normalization import TextNormalization

class TextStemmingTestCase(unittest.TestCase):

    def testStemming(self):
        words = ["Walking", "Walks", "walked", "walk"]
        stemsFound = set()
        normalize = TextNormalization()
        for word in words:
            stemmedWord = normalize.stemming(word)
            stemsFound.add(stemmedWord)
        self.assertTrue(("walk" in stemsFound) and (len(stemsFound)==1))

if __name__ == '__main__':
    unittest.main()
```

Figure 5.5 - TextStemmingTestCase unit test

Please refer to the subsection introduction for an understanding of input and output representations within the above test case.

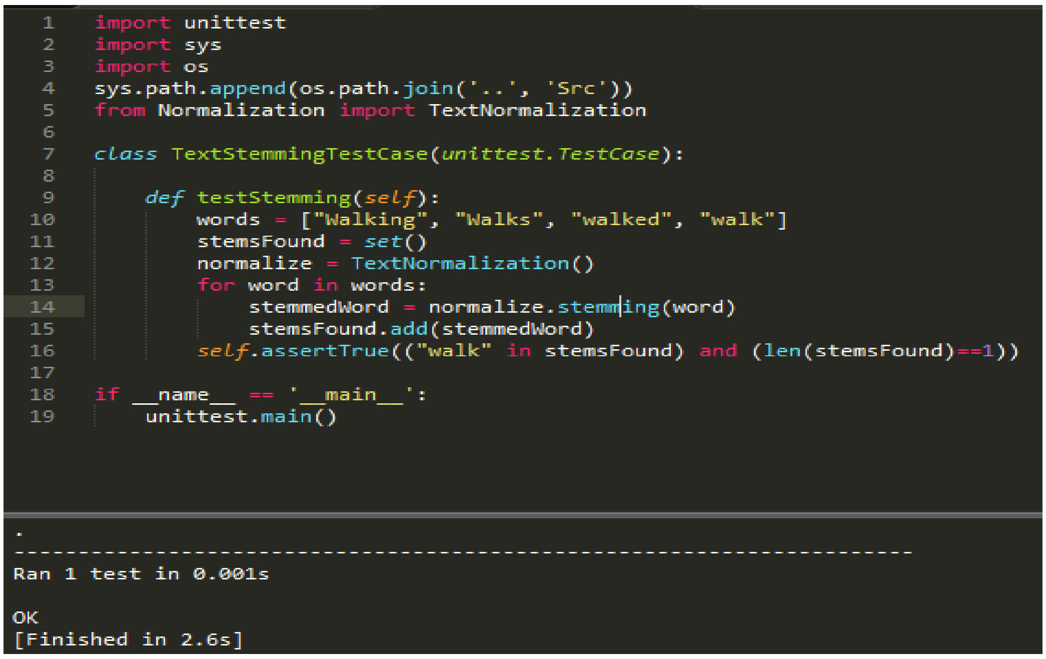
Actions Taken:

The TextStemmingTestCase test case tests the Stemming within the project. The test case exhibits one method:

- testStemming - The Stemming method is given 4 words “Walking”, “Walks”, “walked”, “walk” with the given output expected to be the stem of the word which in this case would be the common stem of “walk”. Two of the words begin with uppercase letters and two begin with lowercase letters to test whether this makes a difference in the stem returned.

Analysis of Test results:

The initial dot printed within the command line in the bottom section of the image below indicates that the one test passed. Which is indicated later within the command line as it is printed “Ran 1 test” and later “OK”.



```
import unittest
import sys
import os
sys.path.append(os.path.join('..', 'Src'))
from Normalization import TextNormalization

class TextStemmingTestCase(unittest.TestCase):

    def testStemming(self):
        words = ["Walking", "Walks", "walked", "walk"]
        stemsFound = set()
        normalize = TextNormalization()
        for word in words:
            stemmedWord = normalize.stemming(word)
            stemsFound.add(stemmedWord)
        self.assertTrue(("walk" in stemsFound) and (len(stemsFound)==1))

if __name__ == '__main__':
    unittest.main()
```

```
.
----------------------------------------------------------------------
Ran 1 test in 0.001s

OK
[Finished in 2.6s]
```

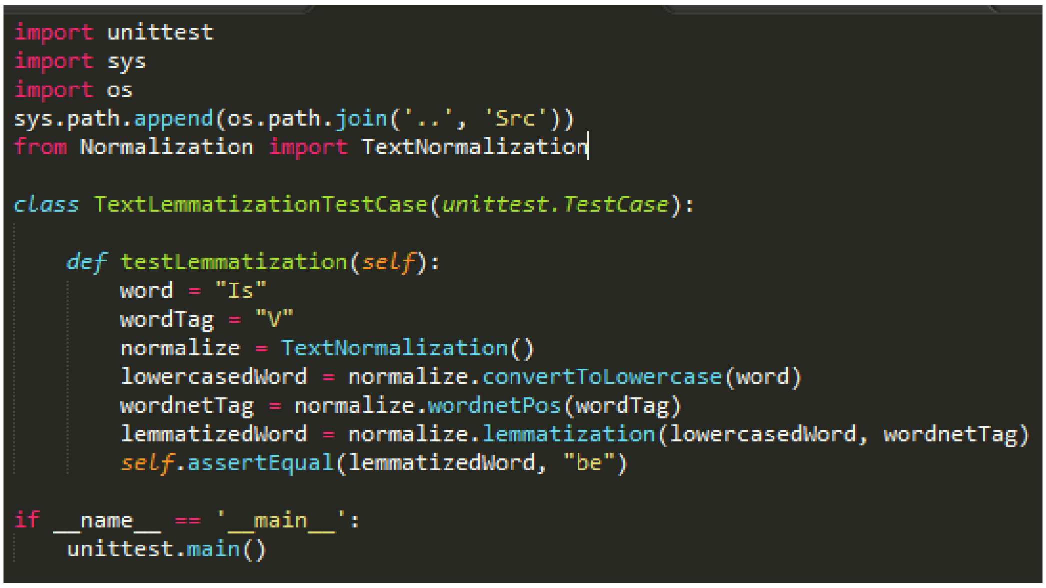
Figure 5.6 - TextStemmingTestCase unit test results

The one test within the TextStemmingTestCase test case passed with the correct stem found for the words.

5.2.4 Lemmatization Validation

This subsection covers the validation of the Lemmatization aspect of the project.

For the testLemmatization test the input is stored within the word and wordTag variables, the output is stored within lemmatizedWord variable. The expected output is outlined within the parameter of the assertEqual method.



```
import unittest
import sys
import os
sys.path.append(os.path.join('..', 'Src'))
from Normalization import TextNormalization

class TextLemmatizationTestCase(unittest.TestCase):

    def testLemmatization(self):
        word = "Is"
        wordTag = "V"
        normalize = TextNormalization()
        lowercasedWord = normalize.convertToLowercase(word)
        wordnetTag = normalize.wordnetPos(wordTag)
        lemmatizedWord = normalize.lemmatization(lowercasedWord, wordnetTag)
        self.assertEqual(lemmatizedWord, "be")

if __name__ == '__main__':
    unittest.main()
```

Figure 5.7 - TextLemmatizationTestCase unit test

Please refer to the subsection introduction for an understanding of input and output representations within the above test case.

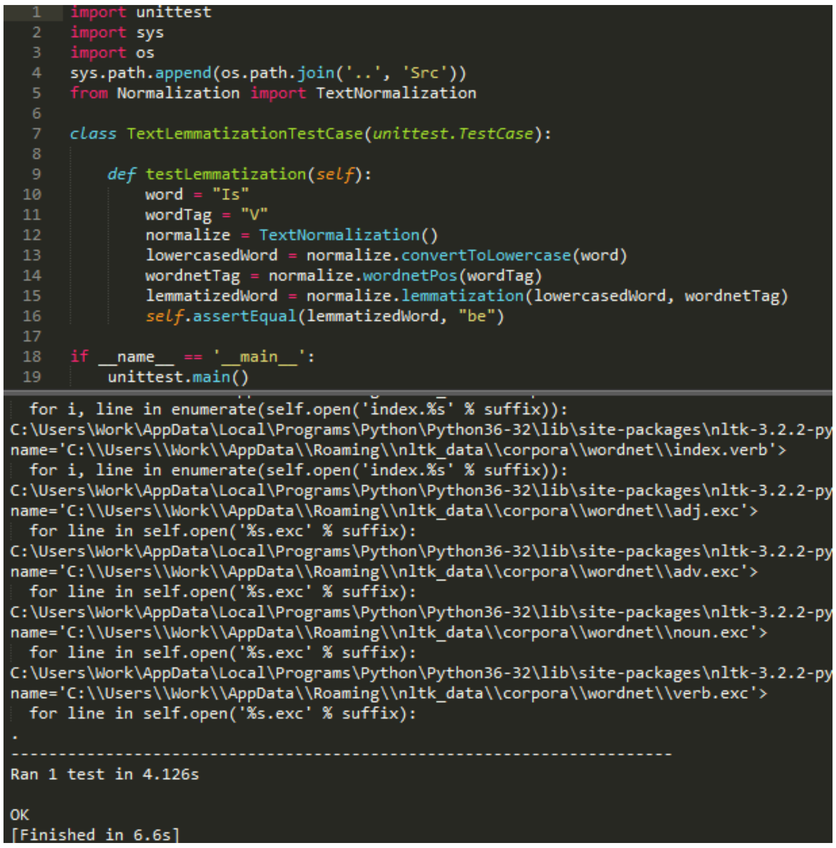
Actions Taken:

The TextLemmatizationTestCase test case tests the lemmatization of the project. The test case exhibits one method:

- testLemmatization - As explained within previous sections Lemmatization places a greater emphasis on the context of a given word to be input, this context is delivered through the given tag of the word. In this test case the word “Is” is provided as input with it being in the context of a Verb. The Lemmatization method expects that the word will be given in lowercase fashion. As mentioned previously within the development and implementation chapter that since WordNet is utilised for the lemmatization process then the associated tag needs to be according to the WordNet standard, this WordNet standardized tag is given as the second parameter to the method. The word chosen within this test case has been purposely made to have a different root and stem to its lemma since many times the lemma of a word can be the same and with this test case both conditions can be covered within testing. Within this test case the expected lemma of the word “Is” within the context of a Verb is “be”.

Analysis of Test results:

The dot printed on its own within the command line in the bottom section of the image below indicates that the one test passed. Which is indicated later within the command line as it is printed “Ran 1 test” and later “OK”.



```
import unittest
import sys
import os
sys.path.append(os.path.join('..', 'Src'))
from Normalization import TextNormalization

class TextLemmatizationTestCase(unittest.TestCase):

    def testLemmatization(self):
        word = "Is"
        wordTag = "V"
        normalize = TextNormalization()
        lowercasedWord = normalize.convertToLowercase(word)
        wordnetTag = normalize.wordnetPos(wordTag)
        lemmatizedWord = normalize.lemmatization(lowercasedWord, wordnetTag)
        self.assertEqual(lemmatizedWord, "be")

if __name__ == '__main__':
    unittest.main()
```

```
  for i, line in enumerate(self.open('index.%s' % suffix)):
C:\Users\Work\AppData\Local\Programs\Python\Python36-32\lib\site-packages\nltk-3.2.2-py
name='C:\\Users\\Work\\AppData\\Roaming\\nltk_data\\corpora\\wordnet\\index.verb'>
  for i, line in enumerate(self.open('index.%s' % suffix)):
C:\Users\Work\AppData\Local\Programs\Python\Python36-32\lib\site-packages\nltk-3.2.2-py
name='C:\\Users\\Work\\AppData\\Roaming\\nltk_data\\corpora\\wordnet\\adj.exc'>
  for line in self.open('%s.exc' % suffix):
C:\Users\Work\AppData\Local\Programs\Python\Python36-32\lib\site-packages\nltk-3.2.2-py
name='C:\\Users\\Work\\AppData\\Roaming\\nltk_data\\corpora\\wordnet\\adv.exc'>
  for line in self.open('%s.exc' % suffix):
C:\Users\Work\AppData\Local\Programs\Python\Python36-32\lib\site-packages\nltk-3.2.2-py
name='C:\\Users\\Work\\AppData\\Roaming\\nltk_data\\corpora\\wordnet\\noun.exc'>
  for line in self.open('%s.exc' % suffix):
C:\Users\Work\AppData\Local\Programs\Python\Python36-32\lib\site-packages\nltk-3.2.2-py
name='C:\\Users\\Work\\AppData\\Roaming\\nltk_data\\corpora\\wordnet\\verb.exc'>
  for line in self.open('%s.exc' % suffix):
.
----------------------------------------------------------------------
Ran 1 test in 4.126s

OK
[Finished in 6.6s]
```

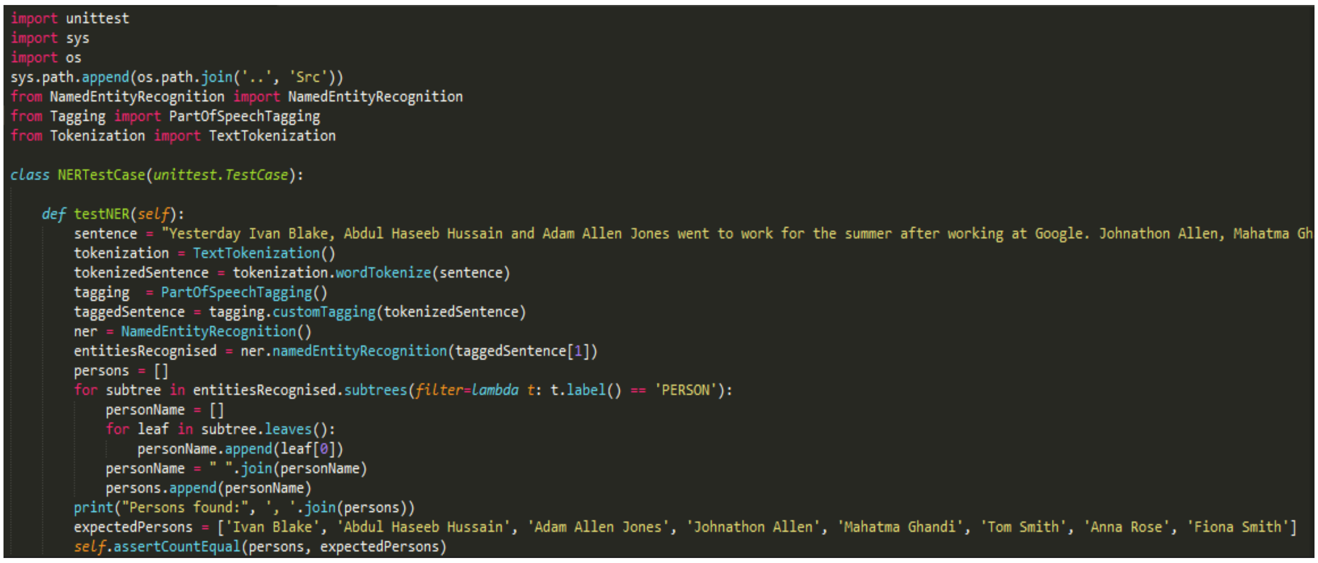
Figure 5.8 - TextLemmatizationTestCase unit test results

The one test within TextLemmatizationTestCase test case passed with the correct lemma found for the word provided as input.

5.2.5 Named Entity Recognition Validation

This subsection covers the validation of the Named Entity Recognition aspect of the project.

The testNER test input is stored within the sentence variable. The output is stored within the variable persons and the expected output is stored within the variable expectedPersons.



```
import unittest
import sys
import os
sys.path.append(os.path.join('..', 'Src'))
from NamedEntityRecognition import NamedEntityRecognition
from Tagging import PartOfSpeechTagging
from Tokenization import TextTokenization

class NERTestCase(unittest.TestCase):

    def testNER(self):
        sentence = "Yesterday Ivan Blake, Abdul Haseeb Hussain and Adam Allen Jones went to work for the summer after working at Google. Johnathon Allen, Mahatma Gh
        tokenization = TextTokenization()
        tokenizedSentence = tokenization.wordTokenize(sentence)
        tagging  = PartOfSpeechTagging()
        taggedSentence = tagging.customTagging(tokenizedSentence)
        ner = NamedEntityRecognition()
        entitiesRecognised = ner.namedEntityRecognition(taggedSentence[1])
        persons = []
        for subtree in entitiesRecognised.subtrees(filter=lambda t: t.label() == 'PERSON'):
            personName = []
            for leaf in subtree.leaves():
                personName.append(leaf[0])
            personName = " ".join(personName)
            persons.append(personName)
        print("Persons found:", ', '.join(persons))
        expectedPersons = ['Ivan Blake', 'Abdul Haseeb Hussain', 'Adam Allen Jones', 'Johnathon Allen', 'Mahatma Ghandi', 'Tom Smith', 'Anna Rose', 'Fiona Smith']
        self.assertCountEqual(persons, expectedPersons)
```

Figure 5.9 - NERTestCase unit test

Please refer to the subsection introduction for an understanding of input and output representations within the above test case.

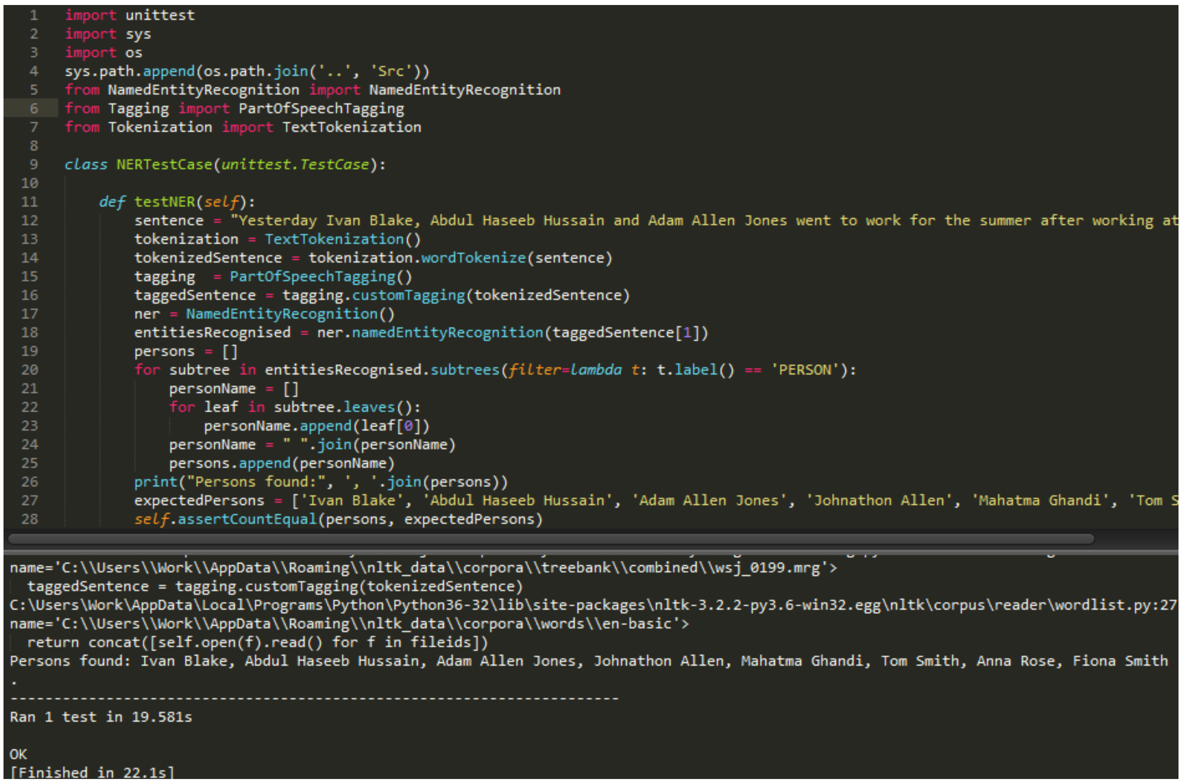
Actions Taken:

The NERTestCase test case tests the Named Entity Recognition aspect of the project through the lens of Persons detection which is the only entity needed within the overall goal of the project. The test case exhibits one method:

- testNER - The text given as input is "Yesterday Ivan Blake, Abdul Haseeb Hussain and Adam Allen Jones went to work for the summer after working at Google. Johnathon Allen, Mahatma Ghandi and Tom Smith are also planning to go tomorrow to Apple. Anna Rose and Fiona Smith." The named entity recognition method returns all entities recognised, these entities are iterated through until labelled persons extracted can be returned. These returned persons are compared against the expected output of "Ivan Blake", "Abdul Haseeb Hussain", "Adam Allen Jones", "Johnathon Allen", "Mahatma Ghandi", "Tom Smith", "Anna Rose", "Fiona Smith". The names selected were purposely varied with names with middle names to ensure that such cases would be detected, the names given as input were also varied in that are a mixture of names from many cultures and different genders to ensure that these would not be a limiting factor though the successful inclusion of these given names accepts that there are many outside cases of names from various cultures that may not be detected with the given method.

Analysis of Test results:

The dot printed on its own within the command line in the bottom section of the image below indicates that the one test passed. Which is indicated later within the command line as it is printed “Ran 1 test” and later “OK”. Additionally, within the command line the persons output from the Named Entity Recognition method are printed.



```
import unittest
import sys
import os
sys.path.append(os.path.join('..', 'Src'))
from NamedEntityRecognition import NamedEntityRecognition
from Tagging import PartOfSpeechTagging
from Tokenization import TextTokenization

class NERTestCase(unittest.TestCase):

    def testNER(self):
        sentence = "Yesterday Ivan Blake, Abdul Haseeb Hussain and Adam Allen Jones went to work for the summer after working at
        tokenization = TextTokenization()
        tokenizedSentence = tokenization.wordTokenize(sentence)
        tagging  = PartOfSpeechTagging()
        taggedSentence = tagging.customTagging(tokenizedSentence)
        ner = NamedEntityRecognition()
        entitiesRecognised = ner.namedEntityRecognition(taggedSentence[1])
        persons = []
        for subtree in entitiesRecognised.subtrees(filter=lambda t: t.label() == 'PERSON'):
            personName = []
            for leaf in subtree.leaves():
                personName.append(leaf[0])
            personName = " ".join(personName)
            persons.append(personName)
        print("Persons found:", ', '.join(persons))
        expectedPersons = ['Ivan Blake', 'Abdul Haseeb Hussain', 'Adam Allen Jones', 'Johnathon Allen', 'Mahatma Ghandi', 'Tom S
        self.assertCountEqual(persons, expectedPersons)
```

```
name='C:\\Users\\Work\\AppData\\Roaming\\nltk_data\\corpora\\treebank\\combined\\wsj_0199.mrg'>
  taggedSentence = tagging.customTagging(tokenizedSentence)
C:\Users\Work\AppData\Local\Programs\Python\Python36-32\lib\site-packages\nltk-3.2.2-py3.6-win32.egg\nltk\corpus\reader\wordlist.py:27
name='C:\\Users\\Work\\AppData\\Roaming\\nltk_data\\corpora\\words\\en-basic'>
  return concat([self.open(f).read() for f in fileids])
Persons found: Ivan Blake, Abdul Haseeb Hussain, Adam Allen Jones, Johnathon Allen, Mahatma Ghandi, Tom Smith, Anna Rose, Fiona Smith
.
----------------------------------------------------------------------
Ran 1 test in 19.581s

OK
[Finished in 22.1s]
```

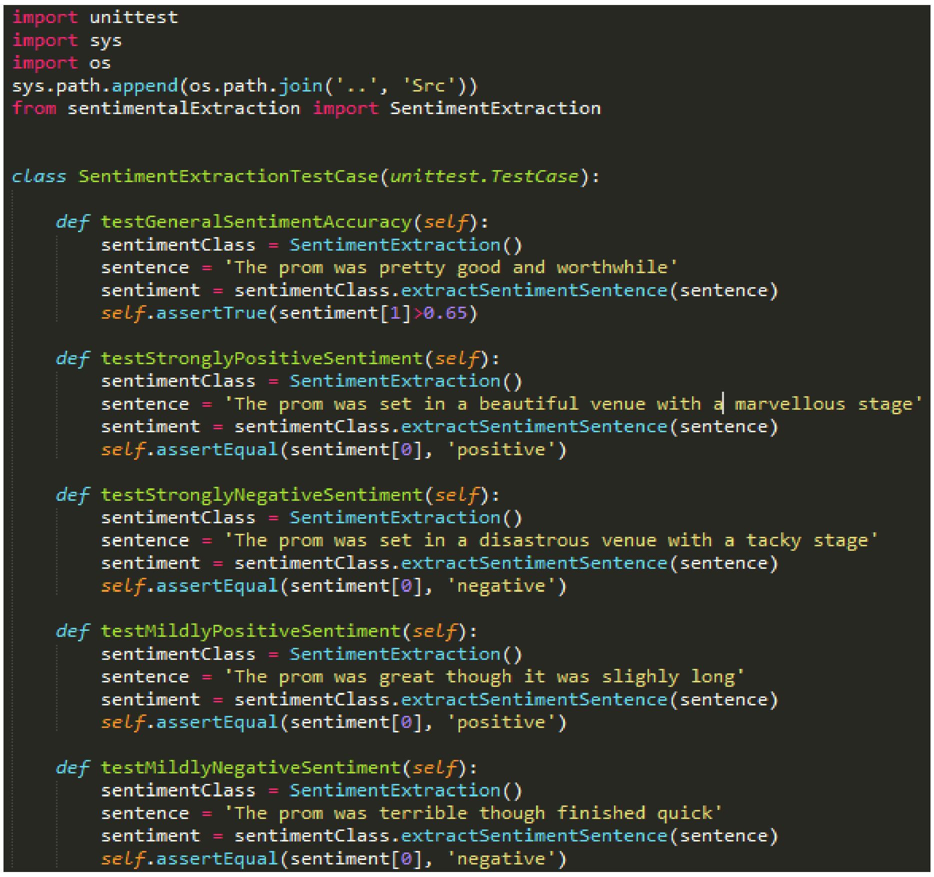
Figure 5.10 - NERTestCase unit test results

The one test within NERTestCase test case passed with all Persons entities recognised from the given test.

5.2.6 Sentiment Extraction Validation

This subsection covers the validation of the Sentiment Extraction aspect of the project.

For all five tests the input is stored within the variable sentence and the output is stored within the variable sentiment, the expected output for all tests is outlined within the assert methods of each test.



```python
import unittest
import sys
import os
sys.path.append(os.path.join('..', 'Src'))
from sentimentalExtraction import SentimentExtraction


class SentimentExtractionTestCase(unittest.TestCase):

    def testGeneralSentimentAccuracy(self):
        sentimentClass = SentimentExtraction()
        sentence = 'The prom was pretty good and worthwhile'
        sentiment = sentimentClass.extractSentimentSentence(sentence)
        self.assertTrue(sentiment[1]>0.65)

    def testStronglyPositiveSentiment(self):
        sentimentClass = SentimentExtraction()
        sentence = 'The prom was set in a beautiful venue with a marvellous stage'
        sentiment = sentimentClass.extractSentimentSentence(sentence)
        self.assertEqual(sentiment[0], 'positive')

    def testStronglyNegativeSentiment(self):
        sentimentClass = SentimentExtraction()
        sentence = 'The prom was set in a disastrous venue with a tacky stage'
        sentiment = sentimentClass.extractSentimentSentence(sentence)
        self.assertEqual(sentiment[0], 'negative')

    def testMildlyPositiveSentiment(self):
        sentimentClass = SentimentExtraction()
        sentence = 'The prom was great though it was slighly long'
        sentiment = sentimentClass.extractSentimentSentence(sentence)
        self.assertEqual(sentiment[0], 'positive')

    def testMildlyNegativeSentiment(self):
        sentimentClass = SentimentExtraction()
        sentence = 'The prom was terrible though finished quick'
        sentiment = sentimentClass.extractSentimentSentence(sentence)
        self.assertEqual(sentiment[0], 'negative')
```

Figure 5.11 - SentimentExtractionTestCase unit test

Please refer to the subsection introduction for an understanding of input and output representations within the above test case.

Actions Taken:

The given test case tests the Sentiment Extraction within the project.

The Sentiment Extraction test case exhibits five methods:

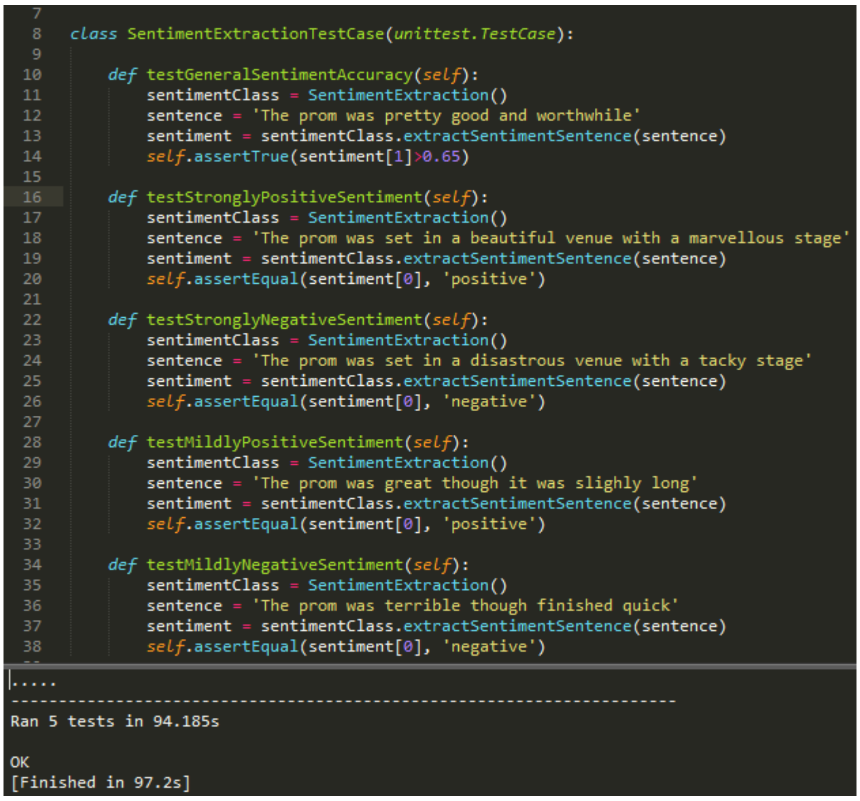
- testGeneralSentimentAccuracy – The Sentiment Extraction method within this test method takes the text "The prom was pretty good and worthwhile" as input, this test method tests the general accuracy of the classification of the sentiment method using the given training data against the given test data. The expected output should determine an accuracy greater than 0.65 or 65%.
- testStronglyPositiveSentiment – The Sentiment Extraction method within this test method takes the text "The prom was set in a beautiful venue with a marvellous stage" as input, from a sentiment perspective this given test is strongly positive with "marvellous" and "beautiful" the key sentiment

parameters with both delivering strongly positive sentiment. The expected output from the Sentiment Extraction method is “positive”.

- testStronglyNegativeSentiment – The Sentiment Extraction method within this test method takes the text “The prom was set in a disastrous venue with a tacky stage” as input, from a sentiment perspective this given test is strongly negative with “disastrous” and “tacky” the key sentiment parameters with both delivering strongly negative sentiment. The expected output from the Sentiment Extraction method is “negative”.
- testMildlyPositiveSentiment – The Sentiment Extraction method within this test method takes the text “The prom was great though it was slightly long” as input, from a sentiment perspective this given test is mildly positive with “great” and “slightly long” the key sentiment parameters with “great” delivering strongly positive sentiment and “slightly long” delivering mildly negative sentiment thereby the combination of these delivers mildly positive sentiment. The expected output from the Sentiment Extraction method is “positive”.
- testMildlyNegativeSentiment – The Sentiment Extraction method within this test method takes the text “The prom was terrible though finished quick” as input, from a sentiment perspective this given test is mildly negative with “terrible” and “finished quick” the key sentiment parameters with “terrible” delivering strongly negative sentiment and “finished quick” delivering mildly positive sentiment when used in this context thereby the combinations of these parameters delivers mildly negative sentiment. The expected output from the Sentiment Extraction method is “negative”.

Analysis of Test results:

The initial five dots printed within the command line in the bottom section of the image below indicates that all five tests passed. Which is indicated later within the command line as it is printed “Ran 5 tests” and later “OK”.



```

class SentimentExtractionTestCase(unittest.TestCase):

    def testGeneralSentimentAccuracy(self):
        sentimentClass = SentimentExtraction()
        sentence = 'The prom was pretty good and worthwhile'
        sentiment = sentimentClass.extractSentimentSentence(sentence)
        self.assertTrue(sentiment[1]>0.65)

    def testStronglyPositiveSentiment(self):
        sentimentClass = SentimentExtraction()
        sentence = 'The prom was set in a beautiful venue with a marvellous stage'
        sentiment = sentimentClass.extractSentimentSentence(sentence)
        self.assertEqual(sentiment[0], 'positive')

    def testStronglyNegativeSentiment(self):
        sentimentClass = SentimentExtraction()
        sentence = 'The prom was set in a disastrous venue with a tacky stage'
        sentiment = sentimentClass.extractSentimentSentence(sentence)
        self.assertEqual(sentiment[0], 'negative')

    def testMildlyPositiveSentiment(self):
        sentimentClass = SentimentExtraction()
        sentence = 'The prom was great though it was slighly long'
        sentiment = sentimentClass.extractSentimentSentence(sentence)
        self.assertEqual(sentiment[0], 'positive')

    def testMildlyNegativeSentiment(self):
        sentimentClass = SentimentExtraction()
        sentence = 'The prom was terrible though finished quick'
        sentiment = sentimentClass.extractSentimentSentence(sentence)
        self.assertEqual(sentiment[0], 'negative')
```

```
.....
----------------------------------------------------------------------
Ran 5 tests in 94.185s

OK
[Finished in 97.2s]
```

Figure 5.12 - SentimentExtractionTestCase unit test results

All five tests within the SentimentExtractionTestCase test case passed with sentiment correctly analysed.

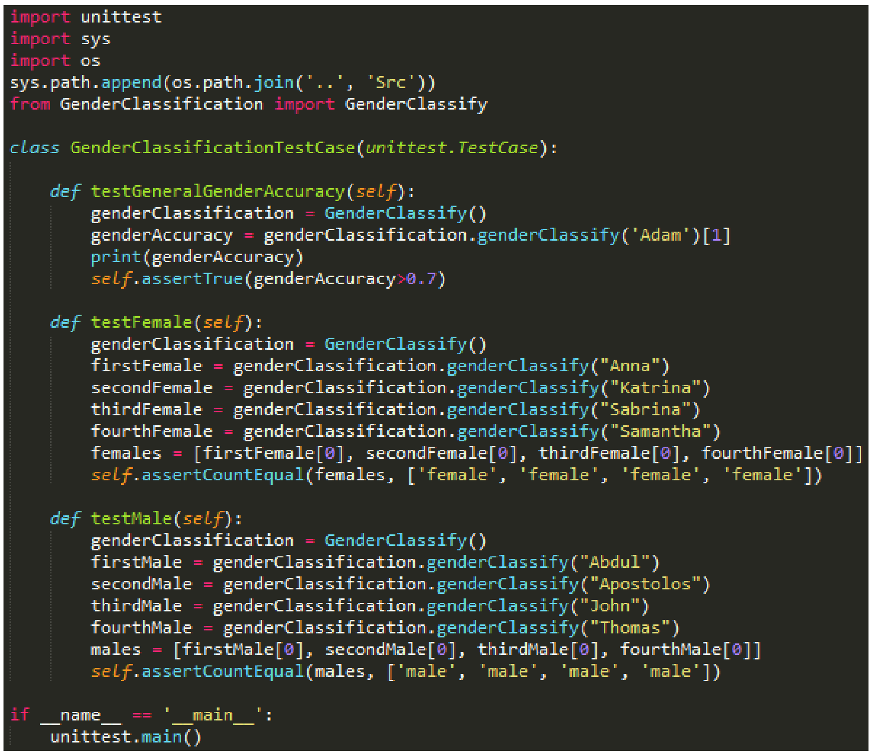
5.2.7 Gender Classification Validation

This subsection covers the validation of the Gender Classification aspect of the project.

For the testGeneralGenderAccuracy test the input is specified as a parameter to the genderClassify method, the output is stored within the variable genderAccuracy and the expected output is outlined within the parameter of the assertTrue method.

For the testFemale test the input is stored within the variables firstFemale, secondFemale, thirdFemale and fourthFemale. The output is stored within the variable females and the expected output is outlined within the parameter of the assertCountEqual method.

For the testMale test the input is stored within the variables firstMale, secondMale, thirdMale and fourthMale. The output is stored within the variable males and the expected output is outlined within the parameter of the assertCountEqual method.



```python
import unittest
import sys
import os
sys.path.append(os.path.join('..', 'Src'))
from GenderClassification import GenderClassify

class GenderClassificationTestCase(unittest.TestCase):

    def testGeneralGenderAccuracy(self):
        genderClassification = GenderClassify()
        genderAccuracy = genderClassification.genderClassify('Adam')[1]
        print(genderAccuracy)
        self.assertTrue(genderAccuracy>0.7)

    def testFemale(self):
        genderClassification = GenderClassify()
        firstFemale = genderClassification.genderClassify("Anna")
        secondFemale = genderClassification.genderClassify("Katrina")
        thirdFemale = genderClassification.genderClassify("Sabrina")
        fourthFemale = genderClassification.genderClassify("Samantha")
        females = [firstFemale[0], secondFemale[0], thirdFemale[0], fourthFemale[0]]
        self.assertCountEqual(females, ['female', 'female', 'female', 'female'])

    def testMale(self):
        genderClassification = GenderClassify()
        firstMale = genderClassification.genderClassify("Abdul")
        secondMale = genderClassification.genderClassify("Apostolos")
        thirdMale = genderClassification.genderClassify("John")
        fourthMale = genderClassification.genderClassify("Thomas")
        males = [firstMale[0], secondMale[0], thirdMale[0], fourthMale[0]]
        self.assertCountEqual(males, ['male', 'male', 'male', 'male'])

if __name__ == '__main__':
    unittest.main()
```

Figure 5.13 - GenderClassificationTestCase unit test

Please refer to the subsection introduction for an understanding of input and output representations within the above test case.

Actions taken:

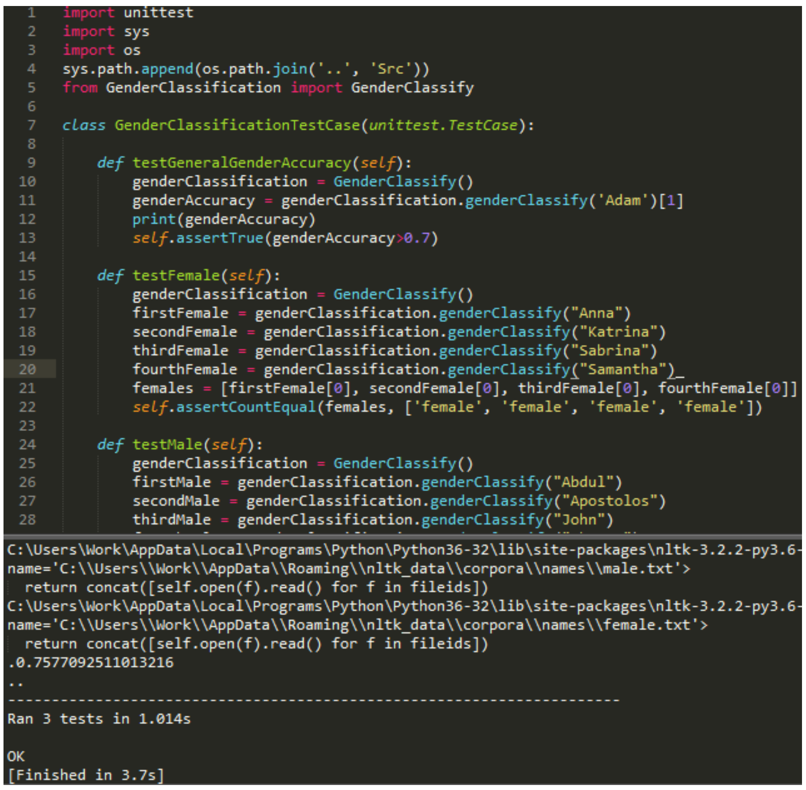
The GenderClassificationTestCase test case tests the Gender Classification of the project. It exhibits three methods:

- testGeneralGenderAccuracy – The Gender Classification method within this test method takes as input the name “Adam” with output as the percentage of the accuracy of the gender classification method through testing the gender classification with training data on method against the given test data, the accuracy output should be greater than 0.7 or 70%.
- testFemale – The Gender Classification method within this test method takes as input four female names “Anna”, “Katrina”, “Sabrina” and “Samantha” with the expected output for each of these being an assertion of “female”.
- testMale – The Gender Classification method within this test method takes as input four male name “Abdul”, “Apostolos”, “John” and “Thomas” with the expected output for each of these being an assertion of “male”.

The cultural variety of the names given as input to both of these tests examines whether this method is restricted to names from certain cultures.

Analysis of Test results:

The one dot followed by a decimal printed on their own within the command line in the bottom section of the image below indicates that the testGeneralGenderAccuracy test passed and the decimal represents the accuracy which as a percentage is 75%. The two dots printed on their own in the command line indicate that the testFemale and testMale tests passed. The passing of all three tests is indicated later within the command line as it is printed "Ran 3 tests" and later "OK".



```python
import unittest
import sys
import os
sys.path.append(os.path.join('..', 'Src'))
from GenderClassification import GenderClassify

class GenderClassificationTestCase(unittest.TestCase):

    def testGeneralGenderAccuracy(self):
        genderClassification = GenderClassify()
        genderAccuracy = genderClassification.genderClassify('Adam')[1]
        print(genderAccuracy)
        self.assertTrue(genderAccuracy>0.7)

    def testFemale(self):
        genderClassification = GenderClassify()
        firstFemale = genderClassification.genderClassify("Anna")
        secondFemale = genderClassification.genderClassify("Katrina")
        thirdFemale = genderClassification.genderClassify("Sabrina")
        fourthFemale = genderClassification.genderClassify("Samantha")
        females = [firstFemale[0], secondFemale[0], thirdFemale[0], fourthFemale[0]]
        self.assertCountEqual(females, ['female', 'female', 'female', 'female'])

    def testMale(self):
        genderClassification = GenderClassify()
        firstMale = genderClassification.genderClassify("Abdul")
        secondMale = genderClassification.genderClassify("Apostolos")
        thirdMale = genderClassification.genderClassify("John")
```

```
C:\Users\Work\AppData\Local\Programs\Python\Python36-32\lib\site-packages\nltk-3.2.2-py3.6-
name='C:\\Users\\Work\\AppData\\Roaming\\nltk_data\\corpora\\names\\male.txt'>
  return concat([self.open(f).read() for f in fileids])
C:\Users\Work\AppData\Local\Programs\Python\Python36-32\lib\site-packages\nltk-3.2.2-py3.6-
name='C:\\Users\\Work\\AppData\\Roaming\\nltk_data\\corpora\\names\\female.txt'>
  return concat([self.open(f).read() for f in fileids])
.0.7577092511013216
..
----------------------------------------------------------------------
Ran 3 tests in 1.014s

OK
[Finished in 3.7s]
```

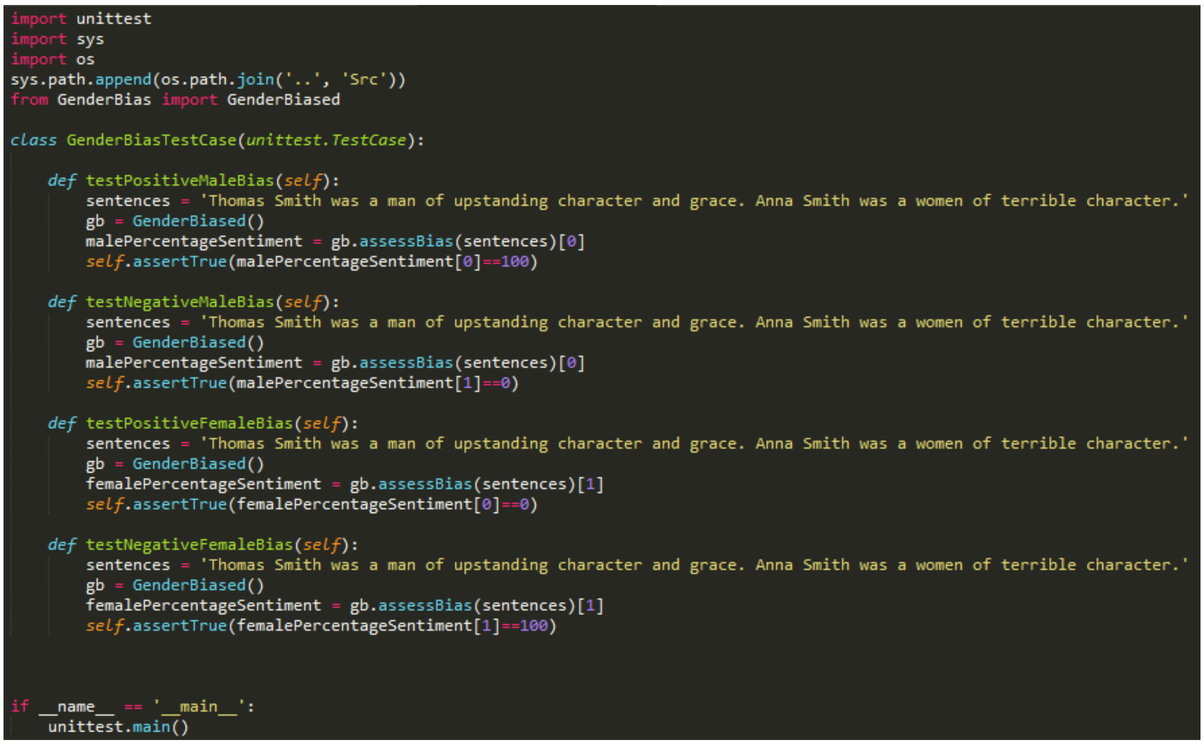
Figure 5.14 - GenderClassificationTestCase unit test results

All three tests within the GenderClassificationTestCase test case passed with gender correctly classified with the accuracy of the Gender Classification method scoring 75%.

5.2.8 Gender Bias Validation

This subsection covers the validation of the Gender Bias aspect of the project.

For all four tests the input is stored within the variable sentences. For the testMalePositiveBias and testMaleNegativeBias tests the actual output is stored within the variable malePercentageSentiment. For the testFemalePositiveBias and testFemaleNegativeBias tests the actual output is stored within the variable femalePercentageSentiment. The expected output for all tests is outlined within the parameters of the assertTrue method.



```python
import unittest
import sys
import os
sys.path.append(os.path.join('..', 'Src'))
from GenderBias import GenderBiased

class GenderBiasTestCase(unittest.TestCase):

    def testPositiveMaleBias(self):
        sentences = 'Thomas Smith was a man of upstanding character and grace. Anna Smith was a women of terrible character.'
        gb = GenderBiased()
        malePercentageSentiment = gb.assessBias(sentences)[0]
        self.assertTrue(malePercentageSentiment[0]==100)

    def testNegativeMaleBias(self):
        sentences = 'Thomas Smith was a man of upstanding character and grace. Anna Smith was a women of terrible character.'
        gb = GenderBiased()
        malePercentageSentiment = gb.assessBias(sentences)[0]
        self.assertTrue(malePercentageSentiment[1]==0)

    def testPositiveFemaleBias(self):
        sentences = 'Thomas Smith was a man of upstanding character and grace. Anna Smith was a women of terrible character.'
        gb = GenderBiased()
        femalePercentageSentiment = gb.assessBias(sentences)[1]
        self.assertTrue(femalePercentageSentiment[0]==0)

    def testNegativeFemaleBias(self):
        sentences = 'Thomas Smith was a man of upstanding character and grace. Anna Smith was a women of terrible character.'
        gb = GenderBiased()
        femalePercentageSentiment = gb.assessBias(sentences)[1]
        self.assertTrue(femalePercentageSentiment[1]==100)


if __name__ == '__main__':
    unittest.main()
```

Figure 5.15 - GenderBiasTestCase unit test

Please refer to the subsection introduction for an understanding of input and output representations within the above test case.

Actions Taken:

The given test case tests the Gender Bias of the project.

The following text is given as input to all tests within the test case: “Thomas Smith was a man of upstanding character and grace. Anna Smith was a women of terrible character.”

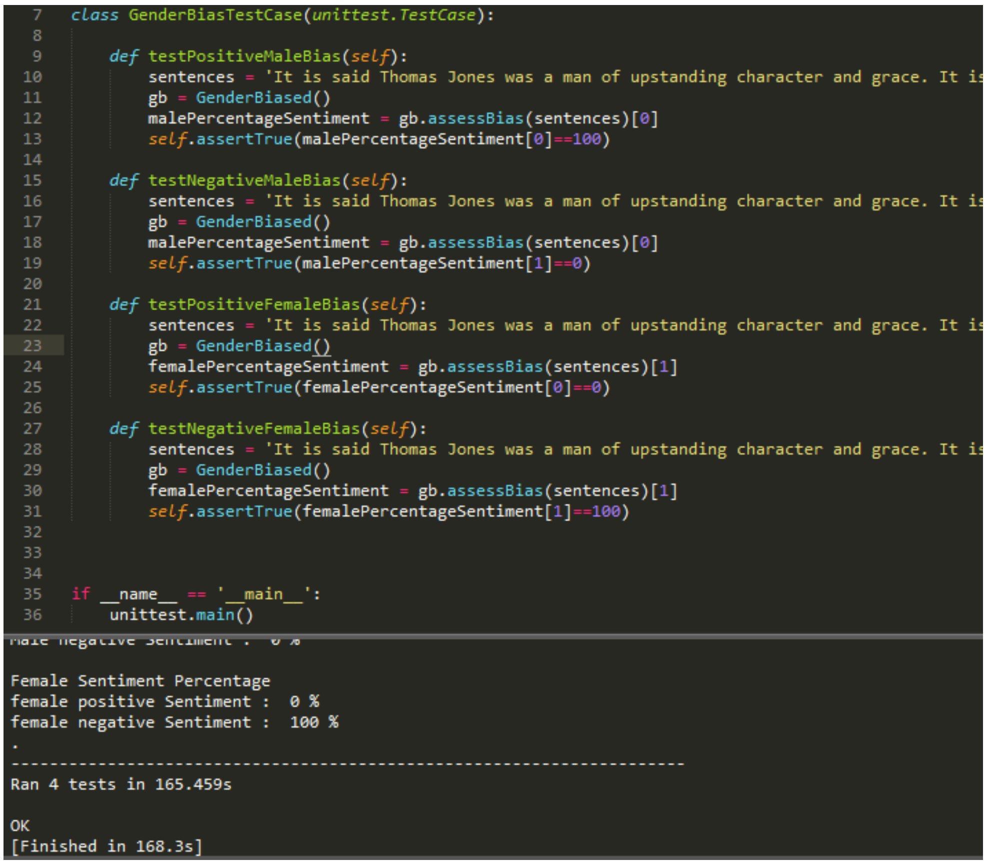
There are four tests:

- testPositiveMaleBias – This test method takes given text as input and outputs the male and female Sentiment after Gender Bias method assesses the bias, the expected male bias outputted is expected to be 100% male positive.

- testNegativeMaleBias – This test method takes given text as input and outputs the male and female Sentiment after Gender Bias method assesses the bias, the expected male bias outputted is expected to be 0% male negative.
- testPositiveFemaleBias – This test method takes given text as input and outputs the male and female Sentiment after Gender Bias method assesses the bias, the expected female bias outputted is expected to be 0% female positive.
- testNegativeFemaleBias – This test method takes given text as input and outputs the male and female Sentiment after Gender Bias method assesses the bias, the expected female bias outputted is expected to be 100% female negative.

Analysis of Test results:

The four dots printed independently on their own within the command line in the bottom section of the image below indicate that the four tests passed. Which is indicated later within the command line as it is printed “Ran 4 tests” and later “OK”. Additionally, within the command line the gender bias sentiment output from the Gender Bias method called over the tests is printed.



```python
class GenderBiasTestCase(unittest.TestCase):

    def testPositiveMaleBias(self):
        sentences = 'It is said Thomas Jones was a man of upstanding character and grace. It is
        gb = GenderBiased()
        malePercentageSentiment = gb.assessBias(sentences)[0]
        self.assertTrue(malePercentageSentiment[0]==100)

    def testNegativeMaleBias(self):
        sentences = 'It is said Thomas Jones was a man of upstanding character and grace. It is
        gb = GenderBiased()
        malePercentageSentiment = gb.assessBias(sentences)[0]
        self.assertTrue(malePercentageSentiment[1]==0)

    def testPositiveFemaleBias(self):
        sentences = 'It is said Thomas Jones was a man of upstanding character and grace. It is
        gb = GenderBiased()
        femalePercentageSentiment = gb.assessBias(sentences)[1]
        self.assertTrue(femalePercentageSentiment[0]==0)

    def testNegativeFemaleBias(self):
        sentences = 'It is said Thomas Jones was a man of upstanding character and grace. It is
        gb = GenderBiased()
        femalePercentageSentiment = gb.assessBias(sentences)[1]
        self.assertTrue(femalePercentageSentiment[1]==100)



if __name__ == '__main__':
    unittest.main()
```

```
Male negative Sentiment :  0 %

Female Sentiment Percentage
female positive Sentiment :  0 %
female negative Sentiment :  100 %
.
----------------------------------------------------------------------
Ran 4 tests in 165.459s

OK
[Finished in 168.3s]
```

Figure 5.16 - GenderBiasTestCase unit test results

All four test within the GenderBiasTestCase test case passed with gender bias correctly analysed.

5.3 User Acceptance Testing

Within the project the customer role was played by my Project Supervisor Apostolos Antoncopoulos who has an invested research interest within the field of Natural Language Processing. Although the project implemented was heavily practical, from the customer's perspective all that mattered was that the testing corroborated implementation especially within the grand scheme of applying Natural Language Processing techniques in some meaningful way to some domain which later matured into determining gender bias using Natural Language Processing techniques. Upon the implementation of each objective within the project a simple user acceptance test was taken whereby the customer accepted/rejected the objective completed culminating within user acceptance of the project as a whole and of results produced within testing.

User Acceptance Tests	Accepted/Rejected
Text Tokenization	Accepted
Part of Speech Tagging	Accepted
Stemming	Accepted
Lemmatization	Accepted
Named Entity Recognition	Accepted
Sentiment Extraction	Accepted
Gender Classification	Accepted
Gender Bias	Accepted

Table 5.1 - User Acceptance Testing table

Within the requirements of the project there was an optional objective related to the creation of a Graphical User Interface for the product though through consultation with the customer and project supervisor Apostolos Antonacopoulos this was deemed unreachable with a project emphasis on the practical results of the Natural Language Processing techniques applied, had the project taken a different course and the Graphical User Interface been created then Testing would have included Usability Testing alongside User Acceptance Tests.

Chapter 6 Critical Evaluation

6.1 Review of Projects achievements against its objectives

The project was successfully able to meet all its objectives for the Minimal Viable Product. Please refer to Section 6.2 for an in-depth review of the achievement of the planned objectives.

Objective 1. Achieved functionality for Text Tokenization on a word and sentence level alongside testing.
Objective 2. Achieved functionality for custom layered Part of Speech Tagging alongside testing.
Objective 3. Achieved functionality for Stemming and Lemmatization using the Porter Stemmer and WordNetLemmatizer alongside testing.
Objective 4. Achieved functionality for recognition of Named Entities, especially Persons alongside testing.
Objective 5. Achieved functionality for Sentiment Extraction alongside testing.
Objective 6. Achieved functionality for Gender Classification alongside testing.
Objective 7. Achieved functionality for the detection of Gender Bias alongside testing.

The project was unable to meet the optional additional objectives:

Objective 1. Unable to implement functionality for the detection of Topics through Topic Modelling alongside testing.
Objective 2. Unable to implement functionality for a Graphical User Interface exposing Natural Language Processing techniques previously implemented and tested.

6.2 Review of the Plan and explanations of any deviation from it

The overall goal of the project is the detection of Gender Bias using Natural Language Processing techniques. All implementation will be done using the Python language. All development will be done using the Sublime Text 3 Text Editor. TeamGantt.com will be utilised for the management of the Gantt chart to manage deadline, deliverables and project documents. GitHub will be utilised as a repository to store code.

As mentioned within the previous section of this chapter all objectives for the Minimal Viable Product were met which satisfied the needs of the customer. Throughout the project there were many deviations from the initial project plan:

- Word Sense Disambiguation – Within the initial project plan this objective was part of the Minimal Viable Product. Initially this was a placeholder objective that was an abstract representation of what I have now specified as the objectives Text Normalization (Stemming and Lemmatization) and Named Entity Recognition as my knowledge of the field increased and the direction of the project became more clearly defined.
- Graphical User Interface – Within the initial project plan this objective was a part of the Minimal Viable Product. Gradually as the project progressed with development of preceding objectives it was decided through consultation with the project supervisor and customer that focus be applied on expanding upon the practical functionality of applying Natural Language Processing techniques rather than providing a user interface since the project is research orientated. The scale of the task of creating a Graphical User Interface also factored into the decision of moving this to become an optional additional objective.
- Sentiment analysis – Initially the overall goal of the project was to provide Sentiment Analysis over a given text within the Minimal Viable Product, as the project progressed the scope of the project changed to provide general analysis of textual data with Sentiment Analysis as an optional additional objective, later the project changed direction again with Sentiment Analysis (also known as Sentiment Extraction) being factored into the Minimal Viable Product with the need to apply Natural Language Processing to some domain in some unique/meaningful way in consultation with project supervisor and customer.
- Machine Learning – Initially given my inexperience within the field I made an assumption that Machine Learning is separate from Natural Language Processing and would be used to supplement the implementation of Natural Language Processing techniques so the application of Machine Learning was

seen as an optional additional objective. As my knowledge of the field increased I quickly realised that many of the Natural Language Processing techniques that I was implementing integrated with machine learning techniques so that they were inseparable, thereby this optional additional objective was unnecessary.

- Able to work on multiple corpuses – Initially this was an additional optional objective with the need to provide Sentiment Analysis on multiple corpuses as initially the project supervisor and customer desired the application of Natural Language Processing techniques over historical datasets though later in consultation the project deviated from this path as it was realised that historical datasets are very difficult to categorise and it would be more suitable to shift the direction of the project to the application of Natural Language Processing techniques to some other domain to provide more meaningful results.
- Application to Domain **-** As the project progressed the project deviated such that the Project supervisor and customer specified that Natural Languages Processing techniques be applied in such unique way to some domain so Application to Domain was a placeholder objective that was an abstract representation of what I have now specified as the Sentiment Extraction (same as Sentiment Analysis), Gender Classification and Gender Bias Assessment Objectives.
- Visualization of sentiment analysis results – Initially Sentiment Analysis was the overall goal of the project with visualization of the results from Sentiment Analysis exposed in some form of interface, as the scope of the project changed through consultation with the project supervisor and customer with it being decided that this visualization was unnecessary.

Finally, as the project progressed it became apparent that additional optional iterations would not be possible and that the contingency plan should be put in to finish the Minimal Viable Product which satisfied the customer and project supervisor.

6.3 Evaluation (with hindsight) of the product, including strengths and weaknesses

In summary, the product was able to successfully meet core objectives to complete the Minimal Viable Product (MVP) which encapsulated the functionality of demonstrating gender bias using Natural Language Processing techniques. The product satisfied the expectations of the customer.

Strengths

Able to form a research foundation on demonstrating the ability to deduce gender bias using Natural Language Processing techniques.

The project was able to successfully complete all aims and objectives set out. Given the lack of research on the subject of assessment of Gender Bias using Natural Language Processing techniques, the project provides a foundation from a research and implementation point of view for others to extend.

Weaknesses

Although the project was able to successful deliver upon the aims and objectives that were set out there are many weaknesses associated with the project which could likely be factored into extensions of the project. Such weaknesses are not necessarily caused by faults within the implementation.

The first weakness is that Gender Bias is not done on **words other than Person names and entities**, using the given test cases which utilise Person names and entities to imply Gender Bias it is unneeded to factor in gender words e.g. “She”, “He” etc. Practically, this exposes a major gap within implementation as gender words are used more frequently within natural language than Person Names exposing another interface needed for assessing gender bias. Unlike the use of Person names and entities which is relatively simple to implement, the extension to allow gender words can exponentially complicate the problem with many languages extensively using such words.

The second weakness is that currently sentiment extraction does not factor the **object and subject of the sentence** within its formulation, within the given test cases this does not pose a problem since each sentence only has an individual person name and entity so there is only one object and subject though commonly sentences contain multiple person name entities whereby the object and subject of sentences needs to be determined to assess upon whom positive/negative sentiment is applied.

The third weakness ties in with the second weakness. Within a given sentence even if a resolution to the second weakness can be determined upon whom is sentiment applied when determining Gender Bias e.g. Within the given sentence “Thomas thinks Anna is terrible”, is a negative sentiment only applied to Anna or Thomas too since it could reflect negatively upon him that he believes Anna is terrible. This is a matter which needs further research.

The fourth weakness is associated with the ambiguity associated with determining sentiment, this is evident when using Natural Languages. Natural Language itself can be ambiguous and words can be used within a variety of contexts to have a variety of different meanings, this itself poses an issue when determining sentiment analysis. For example, the word “ok” could be perceived as negative or positive depending upon the context it is used.

The fifth weakness is the sentiment variables utilised i.e. Positive and Negative. An alternative would have been to utilise weighting values for positive, negative and neutral sentiment. This was initially the case though to counter the use of such weighting also complicates the process of providing meaningful gender bias results which is why the more simplistic method was implemented.

6.4 Lessons learnt during course of the project

Given that this project was my first experience of managing an individual project of such scope, there were many lesson along the duration of the project.

I learned lessons upon the implementation of a hybrid project methodology, initially I assumed that since my project utilised an Agile Incremental approach that I would need to apply the Agile methodology in its entirety, I quickly realised that since Agile is a team-orientated methodology I would not be able to apply it to my individual project in its entirety so I applied individual techniques such as Test Driven Development with Unit Testing and User Acceptance Testing and Iterative development with the Minimal Viable Product (MVP) as a deliverable alongside any optional additional extensions though further iterations.

Having Time managed the project according to the Gantt chart I learnt how to manage and manipulate deadlines using a Gantt chart, utilising time constraints to manage scope, deliverables and customer expectations.

Given that my initial knowledge of the field of Natural Language Processing was nil prior to taking up the project, many lessons were learnt as the project progressed with the order of implementation aligned as I became more familiar with the field.

Before taking up this project I had no prior experience in developing with the Python language, though I quickly picked up the syntax to implement the project to a successful outcome, the choice of the Python language was a lesson learnt as it demonstrated that programming languages should not determine the choice of project rather the choice of programming language utilised for development should be determined by the choice of project. This lesson would serve me well in future with developing within a professional environment operating with this concept.

Chapter 7 Conclusions

7.1 Review of the work done and findings compared with the state-of-the art

The undertaking of the project has allowed me to gather a considerable knowledge base upon the field of Natural Language Processing. The project has also been able to lay the groundwork through implementation to prove the detection of Gender Bias using Natural Language Process techniques with open source research within the area currently lacking.

The product has been able to leverage existing Natural Language Processing projects namely the NLTK suite of libraries to positively contribute to the field of Natural Language Processing. The project has been able to extend the functionality of the NLTK suite of libraries to detect Gender Bias which is not currently a feature of the tool. This enables works into a more contextual direction unlike the pattern recognition method described within subsection 2.2.2.

Current research within the field of Natural Language Processing focuses on the Sentiment Analysis of Social Media, as the field grows then research could expand further into obscure topics such as the determination of Gender Bias using Natural Language Processing techniques whereby the product could be integrated into the other projects.

7.2 Possible improvements to the product and future work

The product provides a research foundation for the detection of gender bias using Natural Language Processing techniques. There are many improvements that can be made to the product especially those weaknesses addressed within the previous chapter outlining a Critical Evaluation of the product, these weaknesses were all concerned with outside cases that would not be satisfied with the existing product and could work given extensions of the methods be implemented though some are inherit problems within the ambiguous nature of the field so may not be possible within the near future. Given that the project is a research project, future work can be conducted given that it is open source upon extending the functionality and eventually providing a user interface whereby users can maybe input documents to the product with it outputting a deep analysis of Gender Bias using Natural Language Processing techniques. Such future work could be conducted by subsequent final year students with an interest in the subject of Natural Language Processing, given that I have implemented much of the groundwork for the project all that will be needed will be a reading of this dissertation to gain an understanding of the field and of the project and of the weaknesses within the product that need addressing prior to any development extensions. Given that the all development has been done using the Python language, this should simplify the process of understanding of code and also further implementation given the easy to grasp nature of the Python language.

7.3 Reflection on any legal/social/ethical/professional issues

From a legal perspective the product poses many legal issues especially with regards to the law regarding Equal Opportunities and Discrimination as the given determined results from the product could shift perspective upon Gender Discrimination and Equal Opportunities. Equal Opportunity and Discrimination laws could be extended to include Natural Language Processing techniques should the product be extended with false positives reduced given the social, professional and ethical effects of the product.

From a professional perspective the product could change business practices and communication mechanisms in a bid to maintain gender neutrality. The project is a public open source project which has been made publicly available as a contribution to knowledge within the field of Natural Language Processing and given the lack of current research on the Detection of Gender Bias using Natural Language Processing techniques.

From a social perspective the utilisation of the product could shift the pendulum of Gender Bias to opposite extremes in a desperate aim to maintain gender neutrality. The product would also radically change the way user’s express information within text leading to an effort to divert from straight forward communication in a bid to maintain gender neutrality.

From an ethical perspective the product poses concerns regarding the ethics of monitoring Gender Bias especially given that the product has the potential to provide many false positives given that it operates upon the ambiguous nature of natural languages, such false positives could provoke reactive policies and controls that are unfair contravening ethical principles. The utilisation of the product upon given data would also bring into disrepute the ownership of the data since writers of specific text may be pressured to conduct assessment using the product.

References

Ali, O., Flaounas, I., Bie, T. D., Mosdell, N., Lewis, J., & Cristianini, N. (2010). Automating News Content Analysis: An Application to Gender Bias and Readability. *Journal of Machine Learning Research*, 7.

Barnbrook, G., Danielsson, P., & Mahlberg, M. (2006). *Meaningful Texts: The Extraction of Semantic Information from Monolingual and Multilingual Corpora.* A&C Black.

Bird, S., Klein, E., & Loper, E. (2006, 03 14). *Chunk Parsing.* Retrieved from University Of Malta: http://staff.um.edu.mt/mros1/csa2050/docs/chunktutorial.pdf

Blei, D. M. (2012, April). Probablistic Topic Models. *Communications of the ACM, 55*(4), 77-84 .

Bolukbasi, T., Chang, K.-W., Zou, J., Saligrama, V., & Kalai, A. (2016, July 21). *Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings.* Retrieved from Cornell University Library: https://arxiv.org/abs/1607.06520v1

Brill, E. (1992). A simple rule-based part of speech tagger. *In Proceedings of the workshop on Speech and Natural Language* (pp. 112-116). Harriman, New York: Association for Computational Linguistics.

Cafarella, M. J., & Etzioni, O. (2005). A search engine for natural language applications. *WWW '05 Proceedings of the 14th international conference on World Wide Web* (pp. 442-452). Chiba, Japan: ACM.

Captain, S. (2015, 10 10). *How Artificial Intelligence Is Finding Gender Bias At Work*. Retrieved from Fast Company: https://www.fastcompany.com/3052053/how-artificial-intelligence-is-finding-gender-bias-at-work

Chowdhury, G. G. (2003). Natural language processing. *Annual Review of Information Science and Technology, 37*(1), 51-89.

Esuli, A., & Sebastiani, F. (2006). *SentiWordNet: A High-Coverage Lexical Resource for Opinion Mining ∗.* Kluwer Academic Publishers.

Gaikwad, G., & Joshi, D. J. (2017). *Multiclass mood classification on Twitter using lexicon dictionary and machine learning algorithms.* IEEE Xplore.

Hull, D. A. (1995, January). Stemming Algorithms - A Case Study for Detailed Evaluation. *JOURNAL OF THE ASSOCIATION FOR INFORMATION SCIENCE AND TECHNOLOGY, 47*(1), 70-84.

Kaplan, R. M. (2005). *A Method for Tokenizing Text. In Festschrift in Honor of Kimmo Koskenniemi's 60th anniversary.* CSLI Publications.

Kupiec, J. (1992). Robust part-of-speech tagging using a hidden Markov model. *Computer Speech & Language, 6*(3), 225-242.

Leech, G., & Svartvik, J. (2013). *A Communicative Grammar of English.* Routledge.

Liu, B. (2012). *Sentiment Analysis and Opinion Mining.* Morgan & Claypool Publishers.

Loper, E., Klein, E., & Bird, S. (2009). *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit.* O'Reilly Media, Inc.

Manning, C. D., Raghavan, P., & Schütze, H. (2009). *An Introduction to Information Retrieval.* Cambridge: Cambridge University Press.

Marcus, M. P., Marcinkiewicz, M. A., & Santorini, B. (1992, 6). Building a large annotated corpus of English: the penn treebank. *Computational Linguistics - Special issue on using large corpora: II, 19*(2).

Miller, G. A., Beckwith, R., & Fellbaum, C. (1990). Introduction to WordNet: An On-line Lexical Database. *International Journal of Lexicography*, 235-312.

Mitkov, R. (2005). *The Oxford Handbook of Computational Linguistics.* Oxford: OUP Oxford.

Nguyen, D. Q., Nguyen, D. Q., Pham, D. D., & Pham, S. B. (2016). A Robust Transformation-Based Learning Approach Using Ripple Down Rules for Part-of-Speech Tagging. *AI Communications, 29*(3), 409-422.

NLTK Project. (2017, 1 2). *NLTK 3.0 documentation*. Retrieved 3 23, 2017, from Natural Language Toolkit: http://www.nltk.org/

Porter, M. (2001, 10). *Snowball: A language for stemming algorithms*. Retrieved from Snowball: http://snowball.tartarus.org/texts/introduction.html

Python Software Foundation. (2017, 3 30). *pickle — Python object serialization*. Retrieved from Python docs: https://docs.python.org/2/library/pickle.html

Python Software Foundation. (2017, 4 26). *Python 2.7.13 documentation*. Retrieved 3 22, 2017, from Python docs: https://docs.python.org/2/

Ratnaparkhi, A. (1996). A Maximum Entropy Model for Part-Of-Speech Tagging. *In EMNLP 1*, 133-142.

S, D., Raj, P., & S.Rajaraajeswari. (2016). A Framework for Text Analytics using the Bag of Words (BoW) Model for Prediction. *International Journal of Advanced Networking & Applications (IJANA)*, 320-323.

Savoy, J. (1999). A STEMMING PROCEDURE AND STOPWORD LIST FOR GENERAL FRENCH CORPORA. *Journal of the American Society for Information Science*, 19.

The Scipy community. (2017, 1 16). *NumPy v1.12 Manual*. Retrieved 3 17, 2017, from SciPy: https://docs.scipy.org/doc/numpy/

The Scipy community. (2017, 3 9). *SciPy docs*. Retrieved 3 15, 2017, from SciPy: https://docs.scipy.org/doc/scipy/reference/

Vogel, A., & Jurafsky, D. (2012). *He Said, She Said: Gender in the ACL Anthology.* Jeju, Republic of Korea: Association for Computational Linguistics.

Wallach, H. M., Murray, I., Salakhutdinov, R., & Mimno, D. (2009). *Evaluation Methods for Topic Models.* ICML.

Appendix A – Project Logbook

Week 1 – (26st September 2016 - 3rd October 2016)

Work Achieved and Things Learnt:

Meeting with Apostolos, researched Natural Language Processing tool NLTK and researched project management tools to use. Discussed potential project ideas with Apostolos and finalized the project title as Sentimental Analysis of Textual Data. Decided that Aposotolos will be supervisor, Github will be used as a code repository, decided ok to potentially use online project management tool. Apostolos talked about the university's access to large datasets. Researched general overview of NLTK and its NLP functionality. I learnt about how NLTK implements Natural Language Processing techniques.

Problems:

Unsure of what type of data to perform sentiment on.

Outcomes and Solutions:

Research variety of data to do sentiment on e.g. social media, articles etc.

Next Task:

Need to start project plan.

Week 2 – (3rd October 2016 – 10th October 2016)

Work Achieved and Things Learnt:

Decided to use TeamGantt.com as an online portal for the Gantt chart and repository project documentation. Finished Initial Gantt chart for project. I have also started the project plan. Learnt about the TeamGantt.com interface.

Problems:

At this stage there are many unknown variables, this combined with my inexperience within the field and as a project manager makes it tricky to be specific within the Gantt chart.

Outcomes and Solutions:

I kept the Gantt chart as abstract as possible until I can gain more familiarity within the field.

Next Task:

Gain access to datasets through Apostolos, this could be used as a starting point for deciding on datasets to potentially use.

Week 3 – (10th October 2016 – 17th October 2016)

Work Achieved and Things Learnt:

Meeting with Apostolos, Requested Apostolos to provide access to large datasets. Access was granted through online primaresearch portal. Learnt about the structure of many of these datasets that Apostolos provided.

Problems:

There are many issues with the datasets that Apostolos provided namely a language barrier with many datasets only available in non-English.

Outcomes and Solutions:

Will mention this to Apostolos within next meeting.

Next Task:

Need to decide on initial dataset to use for sentimental analysis.

Week 4 – (17th October 2016 – 24th October 2016)

Work Achieved and Things Learnt:

Discussed with Apostolos about Issues with PrimaResearch as many are only available in non-English with lack of access to British Library Datasets. Apostolos also sent British Library dataset through email. I learnt about the structure about these British Library datasets.

Problems:

Apostolos's datasets are unlikely to be suitable for sentimental analysis.

Outcomes and Solutions:

Decided on placeholder of BBC Sport dataset until I've chosen final dataset, this dataset may not be needed though since I am dealing with many unknown variables I may need a dataset which I can experiment on in the short term. This topic of choosing a dataset will be revisited when the unique Application to Domain becomes the focus objective.

Next Task:

Attempt to complete Text Tokenization alongside Unit Testing for 28th October.

Week 5 – (24th October 2016 – 31st October 2016)

Work Achieved and Things Learnt:

Completed Text Tokenization alongside complementary unit testing tasks using NLTK. Apostolos mentioned that any research papers I read and tools I use, I should document it so it can be mentioned within the dissertation, he also accepted the use of the BBC Sport dataset as a placeholder. I leant about the technique of Text Tokenization within Natural Language Processing.

Problems:

Historical documents may not be viable source for application of NLP techniques due to inconsistencies from the British Library and also historical documents are not formatted and categorised as they are now.

Outcomes and Solutions:

Apostolos recommended looking into other potential datasets such as sports.

Next Task:

Attempt to complete custom layered Tagging alongside complementary Unit Testing for next week. Learn version control to back up code.

Week 6 – (31st October 2016 – 7th November 2016)

Work Achieved and Things Learnt:

Completed custom layered Tagging alongside unit testing tasks using NLTK. Have started using version control through GitHub. Apostolos mentioned that eventually I will need to eventually work on extracting named entities from words and sentences. I learnt how to use version control through GitHub interface. Learnt about the layered approach within the Tagging techniques within Natural Language Processing with various taggers and how to train and test these using machine learning algorithms.

Problems:

Had issues with using Git whilst experimenting on the command line. Had many issues with trying to implement custom layered tagging from an implementation level with the use of a machine learning algorithm for training and testing the taggers. The training of the classifier was very time consuming especially since the classifier needs to be trained each time the method is called.

Outcomes and Solutions:

Decided to settle for GitHub over Git on the command line. Experimentation with layered tagging allowed the degree of accuracy exhibited within testing to be increased with each layer. I continued with retraining the classifier each time the method is called until later within the project I discovered Pickle which is a Python module to save the trained tagger so it can be loaded from a file rather than the need for retraining which is time consuming.

Next Task:

Do Text Normalization alongside complementary Unit Testing for 25th November with additional time allocated for working on the submission of two assignments for other modules.

Week 7 – (7th November 2016 – 14th November 2016)

Work Achieved and Things Learnt:

Done WordNet morphology and lowercase conversion aspect of Text Normalization using NLTK. Apostolos mentioned he is away on the 5th December and we will make up for it on the 12th December, he discussed his trip to Mexico for the conference and the importance of this in the grand scheme of the field. He also discussed how sentiment could be extracted in two ways, one from individual words and secondly from phrases/sentence context. He mentioned this could be something I look into once I reach the Sentimental Analysis objective. I learnt about the WordNet database and its widespread utilisation within the field of Natural Language Processing.

Problems:

Very busy with other assignments.

Outcomes and Solutions:

Managing time strategically with less time being afforded to the project due to these constraints.

Next Task:

Do Stemming aspect of Text Normalization for next week.

Week 8 – (14th November 2016 – 21st November 2016)

Work Achieved and Things Learnt:

Done Stemming aspect of Text Normalization with unit testing. Apostolos mentioned if possible an additional objective could be the provision of support for multiple datasets within product. He also recommended that I start the dissertation after I have submitted the progress report. I learnt about the application and use of stemming within the field of Natural Language Processing and Linguistics especially the PorterStemmer.

Problems:

Managed to submit two assignments though two more assignments are due for early December.

Outcomes and Solutions:

Continuing managing time strategically between project and assignments.

Next Task:

Do Lemmatization alongside complementary Unit Testing for next week.

Week 9 – (21st November 2016 – 28th November 2016)

Work Achieved and Things Learnt:

Done Lemmatization aspect of Text Normalization alongside Unit Testing. During meeting with Apostolos it was decided that the next stage could extract relations and maybe identify similes and metaphors etc. I learnt about the application and use of lemmatization within the field of Natural Language Processing and Linguistics especially the WordNetLemmatizer.

Problems:

Busy with assignments that need submitting in early December. The potential next stage of the project discussed with Apostolos with the identification of similes and metaphors has been deemed unrealistic and unfeasible.

Outcomes and Solutions:

Continuing managing time strategically for the project. The next objective of the project was decided to be Named Entity Recognition.

Next Task:

Complete Named Entity Recognition alongside complementary Unit Testing for the 15th December.

Week 10 – (28th November 2016 – 5th December 2016)

Work Achieved and Things Learnt:

Started researching Named Entity Recognition. I learnt about various strategies on the tackling of Named Entity Recognition within Natural Language with various machine learning algorithms.

Problems:

Busy with assignments, with 2 to submit by next week.

Outcomes and Solutions:

Allocated adequate time for objectives so I can continue managing time between project and assignments.

Next Task:

Continue researching Named Entity Recognition.

Week 11 – (5th December 2016 – 12th December 2016)

Work Achieved and Things Learnt:

Research on Named Entity Recognition. I learnt about the utilisation of custom chunking for Named Entity Recognition. Apostolos stated that he will introduce me to a PHD student via email whose experiences I can potentially learn from. Apostolos asks that the Application to Domain objective is key as it represents how the product would add value.

Problems:

Submitted two assignments though another 2 major assignments are due for mid-January. Was worried about time consuming nature of implementing custom chunking for Named Entity Recognition.

Outcomes and Solutions:

Managing Time strategically for project alongside working on other assignments. Decided to use NLTK's standard Named Enmity recognition method.

Next Task:

Finish development aspect of Named Entity Recognition objective alongside complementary Unit Testing.

Week 12 – (12th December 2016 – 19th December 2016)

Work Achieved and Things Learnt:

Managed to finish Named Entity Recognition objective using NLTK alongside Unit, I'm taking a week off to apply extra focus to assignments for other modules.

Problems:

Busy with assignments.

Outcomes and Solutions:

Will get back to the project next week.

Next Task:

Free week.

Week 13 – (19th December 2016 – 26th December 2016)

Work Achieved and Things Learnt:

Conducted research on how custom Natural Language Processing techniques implemented can be applied to the domain of social media. Decided this is not a viable option to consider for my project. Next Meeting with Apostolos is in early January. I learnt there are many example of applications of Natural Language Processing to Social Media so within this domain my project is not likely to add value.

Problems:

Busy with other assignments.

Outcomes and Solutions:

Strategically managing time to deal with project and assignments for other modules.

Next Task:

Research on how custom Natural Language Processing techniques implemented can be applied to domain of sport

Week 14 – (26th December 2016 – 2nd January 2017)

Work Achieved and Things Learnt:

Conducted research on how custom Natural Language Processing techniques implemented can be applied to domain of sport through online media. Decided this is not a viable option to consider for my project as I learnt that there are many examples of Natural Language Processing to Sport and Online Media so within this domain my project is not likely to add value.

Problems:

Busy with other assignments.

Outcomes and Solutions:

Strategically managing time to deal with project and assignments for other modules.

Next Task:

Research on how custom Natural Language Processing techniques implemented can be applied to domain of news.

Week 15 – (2nd January 2017 – 9th January 2017)

Work Achieved and Things Learnt:

Conducted research on how custom Natural Language Processing techniques implemented can be applied to domain of news through online media. Decided this is not a viable option to consider for my project as I learnt that there are many examples of Natural Language Processing to News and Online Media so within this domain my project is not likely to add value.

Problems:

Deadlines for both assignments are approaching.

Outcomes and Solutions:

Strategically managing time to deal with project and assignments for other modules.

Next Task:

Research on how custom Natural Language Processing techniques implemented can be applied to domain of Stories. Begin working on progress report.

Week 16 – (9th January 2017 – 16th January 2017)

Work Achieved and Things Learnt:

Conducted research on how my custom Natural Language Processing techniques implemented can be applied to the domain of stories through online media. Decided this is not a viable option to consider. Met with Apostolos and gave him an update on the progress of the project, he recognised the stagnation and recommended that I pick up the pace once the progress report has been submitted and interview is done so I can provide more focus towards the final key objective within the MVP. Also Apostolos put me into contact with the PHD student, I have kept correspondence with him and will organise a meeting with him once I return to university for the second semester. I also need to agree a meeting time with Apostolos for the second semester, previously it was 4pm every Monday though this may need to change. Decided this is not a viable option to consider for my project as I learnt that there are many examples of Natural Language Processing to Stories and Online Media so within this domain my project is not likely to add value.

Problems:

Managed to submit both assignments though need to provide an additional small deliverable that will take up the rest of the week.

Outcomes and Solutions:

Completed the small deliverable.

Next Task:

Focus on progress report for the next week.

Week 17 – (16th January 2017 – 23th January 2017)

Work Achieved and Things Learnt:

Completed the progress report and submitted it.

Problems:

I wasn’t sure how to structure the progress report from a professional standard.

Outcomes and Solutions:

After much research I decided on a report structure that satisfies the criteria.

Next Task:

Continue general research on a domain to apply my custom Natural Language Processing techniques to and organise a new meeting time with Apostolos based on my timetable for the second semester.

Week 18 – (23rd January 2017 – 30th January 2017)

Work Achieved and Things Learnt:

Organised Thursday 4pm as the meeting time with Apostolos beginning from next week.

Upon researching upon the independent use of Sentimental Analysis and Gender Classification within Natural Language Processing I decided that I could do a project which utilises both of them in some unique way which refined into the study of Gender Bias using Natural Language Processing techniques. The lack of public research upon the field increased my conviction of following through with this idea to complete the functionality for the Minimal Viable Product. I learnt about the lack of public research within the area of Gender Bias and the viability of development within the area.

Problems:

N/A

Outcomes and Solutions:

N/A

Next Task:

Begin Sentimental Extraction alongside complementary Unit Testing

Week 19 – (30th January 2017 – 6th February 2017)

Work Achieved and Things Learnt:

I have researched on the subject of Sentimental Analysis within Natural Language Processing by studying related work. I have learnt how to extract Sentiment using NLTK and the various machine learning algorithms involved.

Apostolos was satisfied with the unique application to domain being followed through with the detection of Gender Bias.

Problems:

N/A

Outcomes and Solutions:

N/A

Next Task:

Complete Sentimental Extraction alongside complementary Unit Testing by next week.

Week 20 – (6th February 2017 – 13th February 2017)

Work Achieved and Things Learnt:

Apostolos satisfied with work being carried and output from testing.

I learnt how to implement Sentimental Extraction alongside complementary Unit Testing. I utilised machine learning libraries to train and test classifiers to classify given text. I learnt about Pickle a Python module which allows me to save the classifiers so they can be loaded from file rather than the need to be re-trained which is time consuming.

Problems:

I had issues with performance related to the need to re-train the classifier each time the method is called as previously within the implementation of custom tagging.

Outcomes and Solutions:

After some research I found Pickle a Python module which allows me to save the classifiers so they can be loaded from file rather than the need to be re-trained which is time consuming. I also fixed this issue with the implementation of custom tagging.

Next Task:

Complete Gender Classification alongside complementary Unit Testing by next week.

Week 21 – (13th February 2017 – 20th February 2017)

Work Achieved and Things Learnt:

Meeting rescheduled for next week.

I learnt about related work on Gender Classification using Natural Language Processing techniques. I implemented Gender Classification alongside complementary Unit Testing utilising machine learning algorithms alongside Pickle.

Problems:

From testing it was deduced that 100% accuracy could not be achieived.

Outcomes and Solutions:

It was decided that as long as there was a high degree of degree then a certain failure percentage is negligible given the ambiguous nature of the field.

Next Task:

Complete Gender Bias detection alongside complementary Unit Testing by next week.

Week 22 – (20th February 2017 – 27th February 2017)

Work Achieved and Things Learnt:

Apostolos satisfied with project and agrees with the ceasing of development.

Using all previous implemented Natural Language Processing techniques, I implemented a method for the detection of Gender Bias using Natural Language Processing techniques alongside complementary Unit Testing.

Having completed all development associated with the project it was decided in consultation with the supervisor that my full focus associated with the project should targeted towards the dissertation.

Problems:

N/A

Outcomes and Solutions:

N/A

Next Task:

Complete Development and Testing chapters of report by next week.

Week 23 – (27th February 2017 – 6th March 2017)

Work Achieved and Things Learnt:

Apostolos clarified issues upon dissertation structure.

I managed to complete both the Development and Testing chapters of the report. Given the target set from the previous week both of these chapters were expected to around half of the dissertation in terms of length and so this week was very intensive in terms pushing myself to complete these as scheduled.

Problems:

Initially I was unsure of the structure of dissertation as a whole with potential discrepancies with the mark sheet and the project guide

Outcomes and Solutions:

After consultation with Apostolos it was clarified that I should follow the structure specified within the project guide

Next Task:

Complete Background Work chapter of report by next week

Week 24 – (6th March 2017 – 13th March 2017)

Work Achieved and Things Learnt:

Meeting reschedule due to timetable issues.

I managed to complete the Background Work chapter, this chapter was the second longest chapter and unusually long for a standard project, I learnt that for research oriented projects it is expected that this section will be long.

Problems:

N/A

Outcomes and Solutions:

N/A

Next Task:

Complete Critical Evaluation and Specification and Design by next week.

Week 25 – (13th March 2017 – 20th March 2017)

Work Achieved and Things Learnt:

I managed to complete the Critical Evaluation and Specification and Design chapters of the dissertation.

Apostolos satisfied with progress of dissertation. No meeting for next week as Apostolos away next week.

Problems:

Given the research orientated nature of the project my Specification and Design chapter was very short in comparison to other chapters of similar grading percentage.

Outcomes and Solutions:

Through consultation with Apostolos it was clarified that it was understood that this chapter will be shorter given the research orientated nature of the project and that this is not an issue.

Next Task:

Complete Conclusion, Introduction, contents, Acknowledgements and Abstract chapters by next week.

Week 26 – (20th March 2017 – 27th March 2017)

Work Achieved and Things Learnt:

I have completed the dissertation by completing the Conclusion, Introduction, Contents, Acknowledgements and Abstract chapters. I am scheduled to meet with Apostolos on the 30th where the project dissertation will be approved with the potential for minor corrections later.

Problems:

N/A

Outcomes and Solutions:

N/A

Next Task:

Need to begin preparation for the project presentation with 1 month in advance so I have plenty of time to finish other assignments which were given less priority over the past month whereby the focus was on completing the dissertation.

Week 27 – (27th March 2017 – 3rd April 2017)

Work Achieved and Things Learnt:

Gained feedback from Apostolos on half of the dissertation via email.

Problems:

N/A

Outcomes and Solutions:

N/A

Next Task:

Will finish minor corrections by next week.

Week 28 – (3rd April 2017 – 10th April 2017)

Work Achieved and Things Learnt:

Managed to complete minor corrections suggested by Apostolos on half of the dissertation

Problems:

N/A

Outcomes and Solutions:

N/A

Next Task:

Wait for feedback on the rest of the dissertation

Week 29 – (10th April 2017 – 17th April 2017)

Work Achieved and Things Learnt:

Received feedback regarding the rest of the dissertation from Apostolos via email

Problems:

N/A

Outcomes and Solutions:

N/A

Next Task:

Complete minor corrections upon rest of the dissertation within 2 weeks

Week 30 – (17th April 2017 – 24th April 2017)

Work Achieved and Things Learnt:

Continuing minor corrections.

Problems:

N/A

Outcomes and Solutions:

N/A

Next Task:

Complete corrections by meeting with Apostolos on the 27th April and submit dissertation on the 28th April.

Week 31 – (24th April 2017 – 31st April 2017)

Work Achieved and Things Learnt:

Completed corrections upon dissertation. Apostolos satisfied and suggests submitting on the 28th April. Submitted dissertation on the 28th April.

Problems:

N/A

Outcomes and Solutions:

N/A

Next Task:

Begin preparation for showcase/demo of product

Appendix B – Project Plan

Project Plan

Student Name

Abdul Haseeb Hussain

Student Supervisor

Apostolos Antonacopoulos

Project Title

Assessing Gender Bias using Natural Language Processing techniques

Introduction

The goal of the project is to demonstrate the detection of Gender Bias using Natural Language Processing techniques.

Objectives and Tasks

1) Text Tokenization

    - Sentence Tokenization

    - Word Tokenization

2) Part of Speech Tagging

    - Design Layered Tagging approach

    - Tagging using given Training and Test Data

3) Text Normalization

    - Stemming using Porter Stemmer

    - Lemmatization using WordNetLemmatizer

5) Named Entity Recognition

    - Named Entity Recognition using NLTK's build-in ne_chunk method

6) Gender Classification

    - Gender Features

- Gender Classify using Training and Test Data

7) Sentiment Extraction

- Sentiment Features
- Sentiment Extraction using Training and Test Data

8) Gender Bias Analysis

- Assessing Gender Bias using Natural Language Processing techniques implemented

Additional (Optional) Objectives:

1) Topic Modelling

- Topic Modelling using Gensim

2) GUI

- Creation of a Graphical User Interface

Proposed Methodology

The proposed methodology is incremental development with an initial Minimal Viable Product (MVP) and two further iterations expanding upon the MVP to complete the key objectives of the project, at each key stage customer feedback and testing will be undertaken and feed into further development cycles. A Gantt chart will give an overview of the project. Test Driven Development will be used throughout the project with Unit Testing and User Acceptance Testing upon each objective within development.

Software, equipment, facilities etc.

Python Natural Language Processing Libraries such as NLTK. Microsoft Windows will be used as the operating system in development. Sublime Text 3 is being used as the text editor for development. GitHub will be used as a repository to store code. TeamGantt.com will be used to store the Gantt chart for the project alongside project documentation such as the project plan etc. A laptop will be used for all coding.

Appendix C – Progress Report

Written Plan

Project Background

Over the summer of 2016 I stumbled upon the topics of Artificial Intelligence and Natural Language Processing, this led me to conduct general research on the subject of Natural Language Processing and influenced my decision to choose a project within the domain of Natural Language Processing. My initial decision was to choose a project wholly focused on the Sentiment Analysis of Textual Data though this decision was later revised as challenges within the project became apparent as discussed in-depth further within the report.

Although the field of Natural Language Processing is in its infancy much work has been conducted on many Natural Language Processing techniques such as Tagging and Sentiment Analysis within a variety of domains. My project will be unique in that it will apply Natural Language Processing techniques in a unique way to a particular domain contributing to knowledge extension within the field, how this uniqueness will be implemented is yet to be established at this stage within the project.

So far my knowledge of Natural Language Processing techniques has been restricted to those objectives that have been accomplished with me learning as the project progresses which is typical for projects which utilise an Agile Incremental approach with knowledge on the respective field being limited thereby the product is built incrementally as opposed to having the project fully planned out at a pre-development stage leading the project to run linearly.

Project Accomplishments

Much of the Minimal Viable Product has been completed with Text Tokenization, Tagging, Text Normalization and Named Entity Recognition objectives and their respective nested tasks completed. Each of these objectives is accompanied with unit test cases for testing purposes. Once the final objective of Application to Domain is applied to the MVP, the MVP will be ready for delivery to the customer.

Project Problems Arisen

Many of the issues that have arisen such as the revision of objectives and deadlines have been caused by a lack of familiarity with the field so as work was being incrementally completed expectations needed to be revised, stagnation within the project caused deadlines to be extended due to the need manage external constraints such as the need for me to complete assignments for other modules, these constraints were recognised early which allowed for deadlines to be easily extended, the steps of early detection have been taken further and have allowed for contingency plans to have already been drawn for subsequent stages within the project as a precautionary measure.

Aims and Objectives

The following are the aims and objectives to cover the Minimal Viable Product (MVP) for the project

1. Text Tokenization (100% completed)
   - Sentence Tokenization
   - Word Tokenization
2. Tagging (100% completed)
   - Layered Tagging
3. Text Normalization (100% completed)
   - Lowercase Conversion
   - Wordnet morphology
   - Stemming
   - Lemmatization
4. Named Entity Recognition (100% completed)
   - Chunking
5. Application to Domain (0% completed)

The following are additional aims and objectives that may be covered within additional iterations within the project:

1. Sentiment Analysis (0% completed)
   - Using SentiWordnet to extract sentiment from words
   - To measure sentiment of whole text based on above result
2. GUI (0% completed)

- Use a library to create a windows desktop application in Python
- Integrate command line version with GUI

The above aims and objectives differ from the tasks specified within the initial project plan. Initially Sentiment Analysis, the GUI and the visualization of sentiment analysis results were also seen as being a part of the MVP though it was quickly realised that this was not practical with time constraints and also with so many features the quality of the MVP could be diminished with the increased likelihood of bugs. The initial word sense disambiguation objective was a placeholder objective that was an abstract representation of what I have now specified as the objectives Text Normalization and Named Entity Recognition. The additional objectives have been changed from Machine Learning Classification and Able to Work on Multiple Corpuses to Sentiment Analysis and GUI, the reason for these changes are the factoring in of time constraints within the project and the fact that machine learning classifications have already been carried out within the layered custom Tagging and Named Entity Recognition objectives so it makes no sense to revisit this, these additional objectives will be handled by additional iterations should time allow moving forward.

Progress Reflection

Throughout the duration of the project much experience has been gained within a variety of topics. This project is the first project that the project manager has managed leading to experience gained on working with the Agile Incremental project methodology and with project management tools such as Team Gantt to manage the schedule, milestones and overall deadlines for the project. For the project owner this was their first foray developing with the Python programming language and with supporting work within the field of Natural Language Processing being predominately integrated with the Python language, the need to learn the language was a necessity on the development side of the project. Experience has also been gained within software engineering and project management best practices such as Version control and coding standards especially relating to Python. With regards to version control Github is being used as a repository to store code, the use of a graphical interactive online tool such as Github allowed me to bypass much of the learning curve that comes with understanding version control with the command line tool Git.

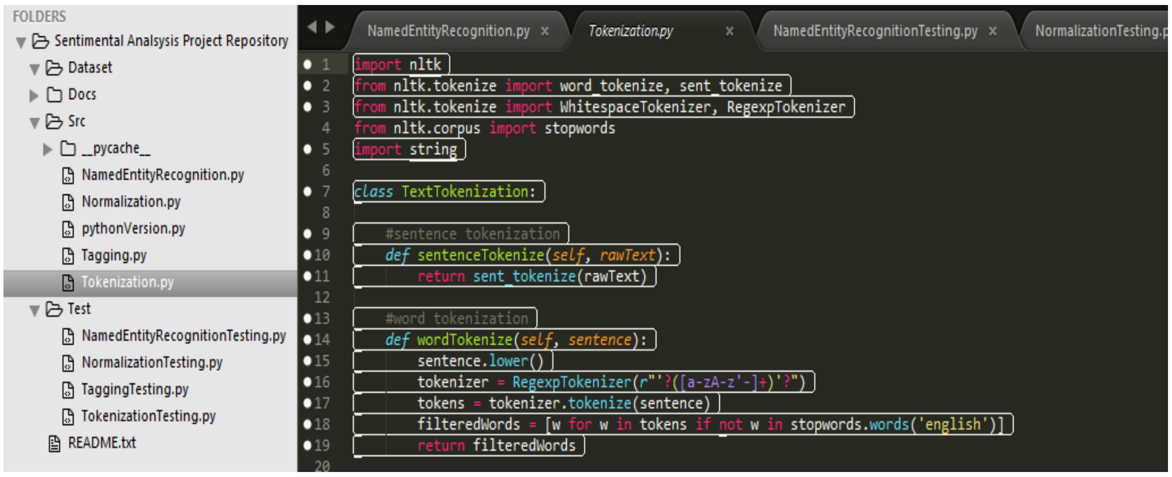
Much experience has been gained in the field of Natural Language Processing with the need to conduct a plethora of research on tools such as NLTK and theory before the development work at each objective as a necessity for that objective to be completed.

Up until this point many decisions have been made on the direction of the project especially as the complexities and nuances of the project were exposed as familiarity with the subject increased, as discussed in the previous section initially Sentiment Analysis, the GUI and the visualization of sentiment analysis results were also seen as being a part of the MVP though it was quickly realised that this was not practical with time constraints, these constraints were caused by external factors such as me needing to work on assignments for other modules and the reduction of scope within the project as with so many features the quality of the MVP could be diminished with an increased likelihood of bugs. The initial word sense disambiguation objective was a placeholder objective that was an abstract representation of what I have now specified as the objectives Text Normalization and Named Entity Recognition, initially I was unsure of what this would represent with research on a variety of topics such as relation extraction, identifying similes and metaphors and chunking etc. for me to narrow down. The additional objectives have been changed from Machine Learning Classification and Able to Work on Multiple Corpuses to Sentiment Analysis and GUI, the reason for these changes are the factoring in of time constraints within the project and the fact that machine learning classifications have already been carried out within the custom layered Tagging and Named Entity Recognition within the backend of the NLTK tool used to implement these objectives so it makes no sense to revisit this.

Adoption of methodology

An Agile Incremental approach was adopted for the project methodology which followed an outline of the creation of a Minimal Viable Product (MVP) with any additional iterations building upon this MVP. Initially it was expected that the Minimal Viable Product and each subsequent iteration would be approximately the same duration though as experience was gained on the project methodology and within the domain of Natural Language Processing this assumption was revised so that the Minimal Viable Product would be the provide the bulk of the work for the project and will last the longest of all the stages and potentially leave no time for additional iterations due to time constraints. The reason the MVP will last so long is due to the initial learning curve and the fact that it implements core functionality within the project, any subsequent iterations provide non-critical additions to this MVP. These additional iterations cover additional objectives. Should the additional objectives not be achievable then the MVP will be delivered to the customer and will encompass the product as a coverage of the main aims and objectives set out within the project.

The choice of the Agile Incremental project methodology is ideal for a project such as this where there are many unknown variables within the field of NLP as it is in its infancy which leads to a steep learning curve for an inexperienced project manager, the project also requires deliverables which makes this Agile Incremental Approach suitable as deliverables can be produced at a quicker pace.



```python
import nltk
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.tokenize import WhitespaceTokenizer, RegexpTokenizer
from nltk.corpus import stopwords
import string

class TextTokenization:

    #sentence tokenization
    def sentenceTokenize(self, rawText):
        return sent_tokenize(rawText)

    #word tokenization
    def wordTokenize(self, sentence):
        sentence.lower()
        tokenizer = RegexpTokenizer(r"'?([a-zA-z'-]+)'?")
        tokens = tokenizer.tokenize(sentence)
        filteredWords = [w for w in tokens if not w in stopwords.words('english')]
        return filteredWords
```

The above screenshot shows evidence of the implementation of the Agile Incremental project methodology within the code with the code being built up in increments with unit testing at each of these stages.

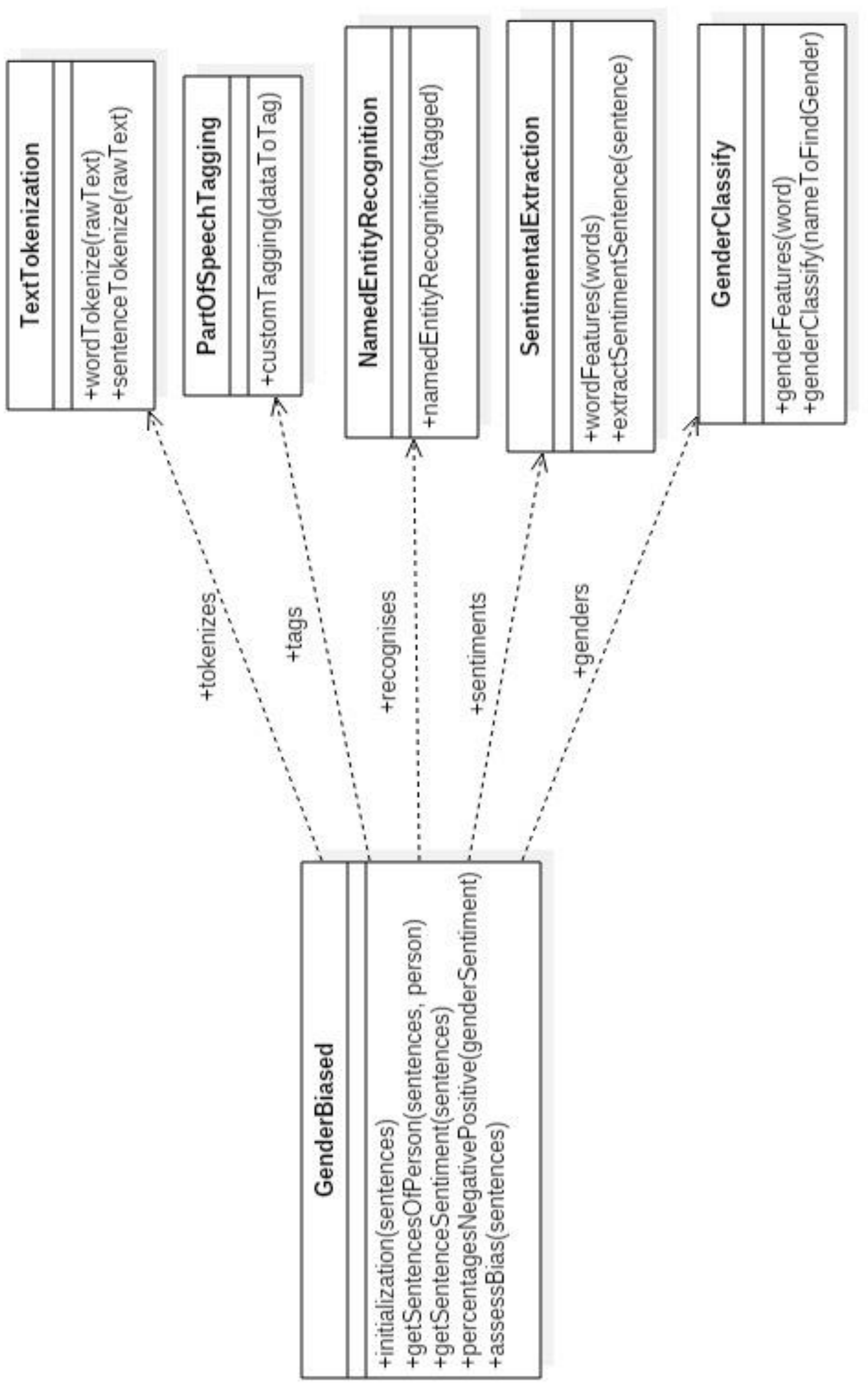
Work that lies ahead

Moving forward the Application to Domain objective should be completed according to its deadline otherwise should this deadline be exceeded then contingency plans will need to be implemented to manage the delivery of the Minimal Viable Product to the customer as an end product and leave additional objectives. Should the MVP be delivered early or on schedule then any additional iterations may be carried out and delivered with the option to fall back on the MVP as a contingency option.

Time Plan

The main deliverable to the customer will be the Minimal Viable Product, this is estimated to be ready by the 21st February 2017. The first optional additional objective is estimated to be ready for the 23rd March 2017. The second optional additional objective is estimated to be ready for the 27th April 2017. The deadlines for deliverables are all dependent on the project running smoothly though since there are many unknown variables that need to be considered then there is a strong likelihood that the project will not run smoothly and contingency plan may need to implemented. Should the additional objectives fail to be ready in any way e.g. the MVP extending its deadline further or bugs are detected within these additional objectives then the MVP will be the product delivered to the customers. Plenty of time has been allocated for the project to meet its minimum target of the delivery of the MVP with the potential to move the deadline further and set aside additional objectives to ensure that this is delivered to the customer. Please find the Gantt Chart for the project within Appendix A of this report.

Appendix D – Design Diagrams



TextTokenization
+wordTokenize(rawText)
+sentenceTokenize(rawText)
PartOfSpeechTagging
+customTagging(dataToTag)
NamedEntityRecognition
+namedEntityRecognition(tagged)
SentimentalExtraction
+wordFeatures(words)
+extractSentimentSentence(sentence)
GenderClassify
+genderFeatures(word)
+genderClassify(nameToFindGender)
+tokenizes
+tags
+recognises
+sentiments
+genders
GenderBiased
+initialization(sentences)
+getSentencesOfPerson(sentences, person)
+getSentenceSentiment(sentences)
+percentagesNegativePositive(genderSentiment)
+assessBias(sentences)