Final Year Project: Dissertation

Sentiment Analysis with Deep Learning in Amazon Products' Reviews

Computer Science B.Sc.

Supervisors: Dr C. Bryant, Dr S. Belguith

Student: [redacted]

02/05/21

Word Count: 14,769

Abstract

Knowing the feelings portrayed towards a product through consumer reviews is essential in the internet age for consumers and product owners alike. This knowledge may influence purchase decisions or lead to improvements in the product. Without going through the time-consuming process of looking into each review, a deep learning model may predict sentiments towards the product from its reviews and provide quick feedback to product owners.

For sentiment analysis with deep learning, the literature into Amazon reviews excluded three-star reviews from a five-star review system, referred to as ambiguous data, in the training of such models (Zhang & LeCun, 2015). Subsequently, this project aimed to justify removing the ambiguous data by evaluating its effect on a model's accuracy.

The experiments found that the inclusion of ambiguous data had no statistically significant impact on the accuracy of models trained with the ambiguous data as positive or negative and models trained without ambiguous data.

From the perspective of the impact on a model's accuracy, these findings concluded that it is of no consequence if ambiguous data is included in the training as negative, positive, or excluded altogether.

Table of Contents

Table of Tables

Table of Figures

1 – Introduction

Advancements in e-commerce have led to numerous online retailers displaying consumer feedback, openly, in the form of product reviews and ratings. Both of which can influence consumers' purchasing decisions (von Helversen, Abramczuk, Kopeć, & Nielek, 2018), are indicative of consumers' satisfaction (Engler, Winter, & Schulz, 2015), and affect brand reputation (Wahyudi & Kristiyanti, 2016). Subsequently, the need to timely evaluate the feelings and opinions conveyed through reviews has become a critical business interest.

Businesses meet this need by assessing the; "attitude, views, feelings, opinions" (Yadav & Vishwakarma, 2020, p. 1) of consumers, known collectively as sentiments, within reviews – a process known as sentiment analysis. Furthermore, the use of deep learning, a subset of machine learning (ML) based on biological neurons, is gaining popularity for sentiment analysis; incentivised by the increased performance it offers when utilising an abundant amount of data.

1.1 Sentiment Analysis (SA) using Deep Learning (DL)

SA using DL models has proven to be able to predict whether an Amazon review is positive or negative, up to an accuracy of "83.90%" (Baktha & Tripathy, 2017, p. 4). However, the inner workings of such models are hidden and challenging to interpret when evaluating the justifications behind isolated predictions; thus, "behave like a "black box"" (Yadav & Vishwakarma, 2020, p. 31).

Concerning this black-box problem, popular solutions work towards creating a separate model in parallel to a given black-box model. This new model focuses on capturing the inputs and outputs of the latter model to, transparently, determine the relationships between them to produce more readily interpreted outputs. Sometimes this process may involve input modifications.

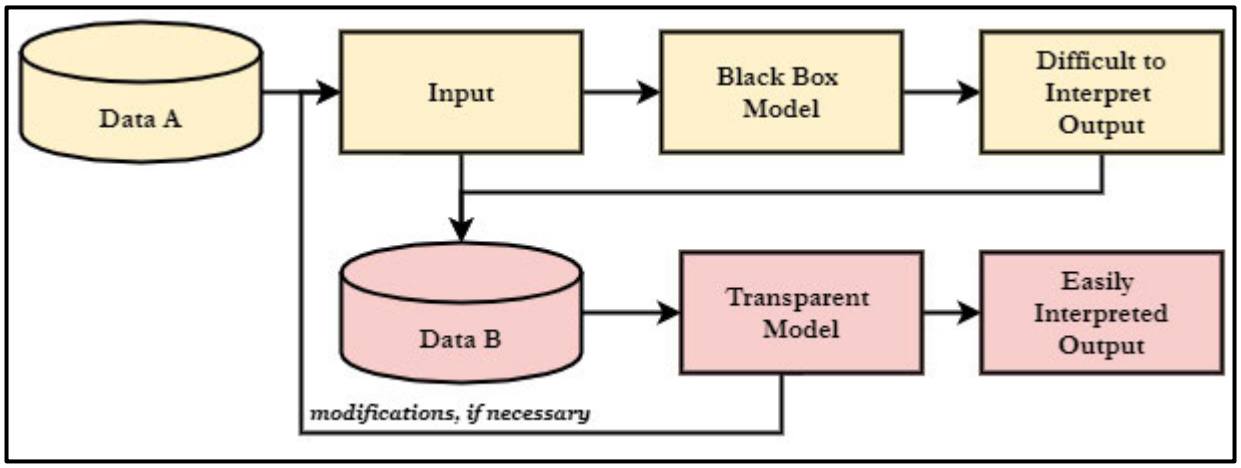

Figure 1: Black-box problem's solution

Figure 1, above, represents a generalised solution to the black-box problem derived from research on explaining predictions (Ribeiro, Singh, & Guestrin, 2016) and explaining black-box models (Guidotti et al., 2018). A point of interest about this representation is that while the black-box model is a DL model, the transparent model could be an alternative algorithmic or mathematical model.

Another pitfall of DL models for SA is that they require large amounts of pre-labelled data (confirmed positives and negatives) to learn to make predictions acceptably. From experience, the process of collecting and labelling data in such a manner can require many resource hours and present unexpected difficulties. Yadav and Vishwakarma (2020) further support this by describing the process as "difficult and tedious".

To address this data problem for Amazon reviews, one may increase the number of people labelling reviews with the consequence of increased labour costs. Another approach is to heuristically label data, that is to say, that if a review has a rating of higher or lower than three, respectively labelling it positive or negative – with the selected ratings defined as rating boundaries. This approach to labelling is evident in a commonly cited dataset (Blitzer, Dredze, & Pereira, 2007). This heuristic approach, while reasonably successful in producing results, does not explicitly express the relationship between predicted sentiments and their respective ratings or whether such a relationship exists – validating its use beyond the results it yields.

1.2 Contribution

This project aims to determine the viability of SA in predicting products' ratings. An investigation into Amazon review predictions and their respective ratings shall produce the desired results to achieve this aim. Subsequently, Amazon reviews will be collected, processed, and evaluated through software produced during this project. The results of this evaluation will lead to more transparency on the validity of the heuristic approach above, which is in line with the justification principles of the black-box problem's solutions. Ultimately, working towards improving the labelling process.

1.3 Objectives

1.3.1 Main Objectives – Seven

1. To learn deep learning techniques and implementations that are involved in sentiment analysis to a sufficient capacity where a functional implementation with reasonably high accuracy can be demonstrated to the supervisor by 11/11/2020

2. To deliver a solution that can collect, create, and save new records of data consisting of Amazon product reviews, collected directly from Amazon's website, requiring minimal product identifiers during the collection process by 25/11/2020

3. To provide a self-collected dataset of Amazon product reviews, targeted towards fifty-thousand records by 11/12/2020

4. To demonstrate a proof-of-concept deep-learning network, on a reduced dataset that works towards being used on two-thousand records, to the supervisor by 30/01/2021

5. To deliver a trained deep-learning network that uses six-thousand records with comparable relative accuracy to established networks in the subject area by 19/02/2021

6. To demonstrate a solution to the supervisor that can query the above-trained network with newly requested examples, aiming to display accuracy-related insights on said examples by 12/03/2021

7. To provide user-friendly extensions to the solution above which have comparable usage experiences to software similar in nature by 09/04/2021

1.3.2 Bonus Objectives – Two

1. To provide integrations into the system that accept different sources of data – one week from main objectives completion

2. To provide usage insights between newly accepted data and previous data – two weeks from bonus objective one completion

1.4 Review Structure

Shown below in figure 2, is an overview of the structure of this review, with section 1 covered thus far; section 2 presents a survey of current literature around the topics related to this project. Section 3 details the requirements, methodologies, and designs adopted during the development and investigation stages of the project. Section 4 summarises and concludes this review.

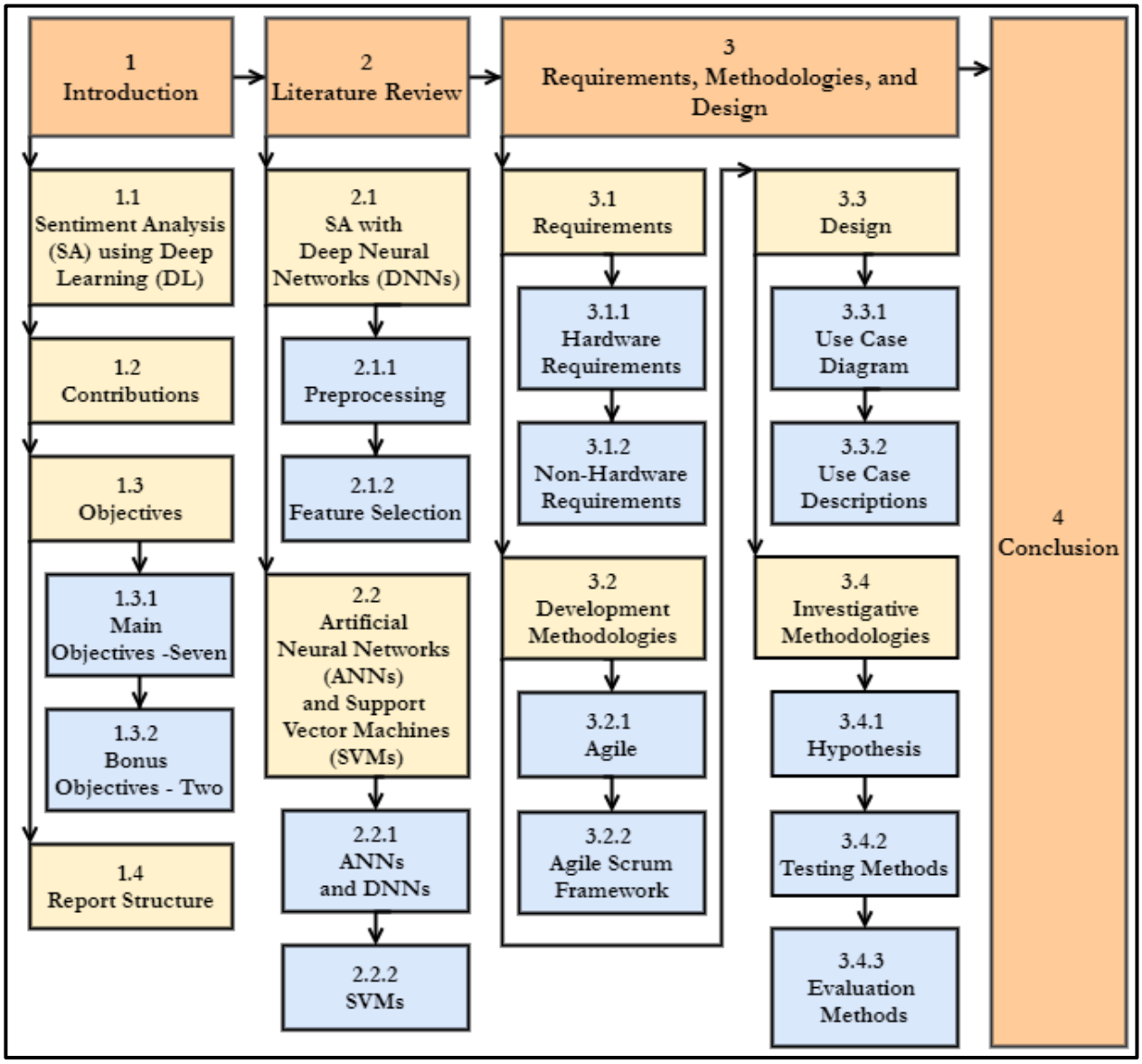

Figure 2: Review Structure

2 – Literature Review

2.1 SA with Deep Neural Networks (DNNs)

This section details how text data is prepared and represented for use in SA, focusing on data intended for use within DL models.

2.1.1 Preprocessing

Preprocessing data involves reducing the amount of irrelevance, and increasing the amount of relevance, data has for its intended use. For SA and DL, preprocessing data aims to make the data suitable for ready translation into being machine-interpretable without any unnecessary clutter. In this section, a discussion on the following preprocessing techniques shall take place:

- Stop word removal
- Stemming
- Lemmatisation
- Tokenisation

Stop word removal refers to the removal of words that provide "little or no information"(Bhadane, Dalal, & Doshi, 2015, p. 810) in regards to sentiments within a given context. Subsequently, the removal of stop words in the context of this project leads to prioritising words that impact the overall sentiment of a review.

The removal of stop words is also evident in all nine of the popular SA datasets identified by Yadav and Vishwakarma (2020, p. 4369) in their review of deep learning architectures, which further validates this technique's use.

These findings have resulted in the decision to use this preprocessing technique alongside punctuation and whitespace removal by inference, within this project.

Stemming is a process of reducing "a word to a common base form"(Manning, Raghavan, & Schütze, 2008, p. 32). Lammatisation is also a process that aims to do this. However, the former process utilises an algorithm which heuristically cuts words without considering its morphological validity (Willett, 2006). In contrast, the latter process considers morphological validity through the analysis of word forms. This analysis, however, requires increased processing. Katić and Milićević (2018) advocate the use of stemming over lemmatisation due to the increased performance it offers by decreasing the amount of space required to store words.

There is a need to store a large number of words in this project. The benefit of utilising stemming is clear and is the chosen technique for reducing words to their base form.

Tokenisation is the process of breaking a section of text into single sub-units, referred to as tokens, that are useful to process (Manning et al., 2008, p. 22). This technique's adoption is mandatory at this stage as applying this technique after all of the chosen techniques above will result in data that is ready for feature selection.

2.1.2 Feature Selection

Once the data preprocessing stage is complete, the need to reduce a large amount of text data into a relatively quick-to-process mathematical representation arises. The process to reduce data in such a manner is known as feature selection.

Word embedding (WE) and term frequency-inverse document frequency (TF-IDF) are two popular methods of feature selection. The WE method mathematically maps words to vectors that are in proximity to words with similar connotations and meanings. In contrast, TF-IDF maps words to vectors by counting word frequencies in a given body of data. The reason for the latter method being frequency-inverse is to increase the value of low-frequency words when encountering new examples (Dang, Moreno-García, & De la Prieta, 2020).

Dang et al. (2020) have also demonstrated, through a series of tests, that TF-IDF performs better in the case of Amazon reviews when paired with a Deep Neural Network (DNN). Their evidence provides the basis for the decision to adopt TF-IDF over WE despite it falling behind in other types of DL networks. Due to DNN being the network used for the project, the following section looks into Artificial Neural Networks and DNNs.

2.2 Artificial Neural Networks (ANN) and Support Vector Machines (SVMs)

This section explores ANNs and SVMs, providing research-backed reasonings for the final choice.

2.2.1 ANNs and DNNs

An artificial neural network is a collection of layers consisting of input-output nodes referred to as neurons, with an input layer, an arbitrary number of hidden layers, and an output layer. Apart from the final output layer, all remaining layers are interconnected.

A DNN is a multi-layered ANN which consists of more than two hidden layers (Dang et al., 2020). DNNs employ complex mathematical models during training. These models are capable of detecting non-linear relationships and then utilising them for predictions. This project requires investigating relationships, irrespective of type; for this reason, a DNN is the DL model of choice for this project.

2.2.2 SVMs

SVMs are a type of ML model which aims to create an optimal hyperplane at the most significant distance between each boundary case of the two outcomes for classification – this type of classification is known as binary classification (Kaur, Mangat, & Nidhi, 2017; Uysal & Murphey, 2017).

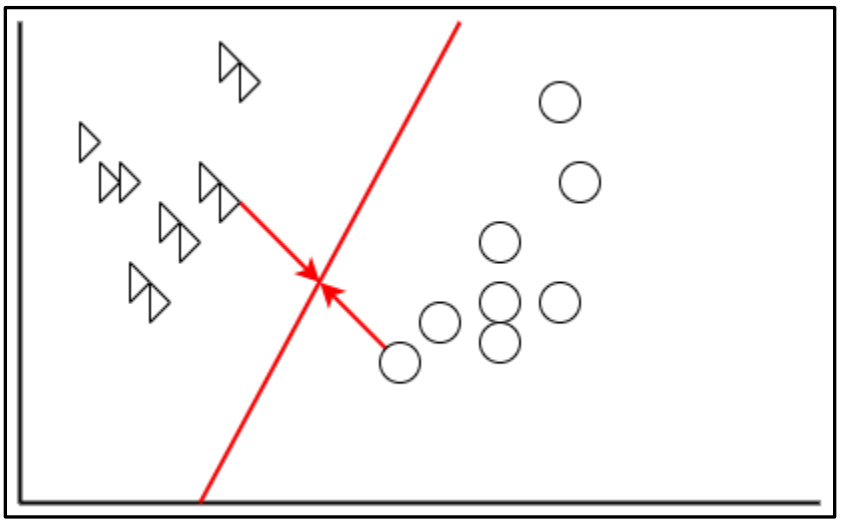

Figure 3: SVM's Hyperplane

Figure 3 above shows the approximated hyperplane represented by the arrowless red line. While an SVM performs well in binary classification tasks, the hyperplane is a linear model (Kaur et al., 2017), thus, can only determine linear relationships. This project's investigation aims to establish a relation between two variables; thus, excluding a relationship type, namely, non-linear does not fit with the aim of this project's investigation. Therefore, apart from research, this project excludes working with SVMs.

Yadav and Vishwakarma (2020) also propose that traditional ML approaches, such as SVM suffer from the time-costs of engineering, designing, and extracting features. They also argue that the large amounts of data in the internet ecosystem result in DNNs outperforming traditional methods due to the increased amount of extractable features, which can be non-linear, present in large amounts of data.

3 – Requirements, Methodologies, and Design

3.1 Requirements

This project requires specific software and hardware to come to fruition. This section will document these requirements.

3.1.1 Hardware Requirements

Due to the high demand for computation, the DL model in this project requires a graphical processing unit (GPU) capable of significantly accelerating computation through various optimisations. Subsequently, the core specifications for the computer system housing this GPU is detailed below:

Table 1: Hardware System

System Component	Name
Central Processing Unit (CPU)	Intel i5 7600K
Motherboard	ASUS Maximus IX Hero
GPU	NVIDIA GeForce GTX 1070 Ti
Random Access Memory (RAM)	16 Gigabytes
Memory	2TB Solid State Drive

The data and software used during this project will, at various stages, interact with each system component listed in Table 1. While such a GPU may not be mandatory

3.1.2 Non-Hardware Requirements

Keras is a popular application programming interface (API) that interfaces with the TensorFlow ML platform to facilitate intuitive DL development in the Python programming language (Chollet, 2015). Due to Keras' ease of use through experience, this API a requirement for this project. Subsequently, anything which Keras in contingent on, TensorFlow and Python, become requirements as well.

Additionally, a Python integrated development environment (IDE) with a focus on analysis is required to aid the development process. Spyder (Spyder, 2020), a scientific IDE for Python, is being used to meet this need.

3.2 Development Methodologies

A part of this project involves creating a piece of software. This section details the development methodologies adopted during this process. The “Design” section documents the design process that arises from the methodologies listed below.

3.2.1 Agile

Due to the experimental and applied nature of this project, software requirements are likely to change. A traditional software development methodology (SDM) such as waterfall will suffer from changing requirements as detailed requirements-based specifications must be written before the implementation of such software. In contrast, Agile is an SDM that divides the development of software into iterative increments over time. These increments aim to reduce risks when dealing with changes in requirements as each increment handles a single or multiple end-to-end functionalities.

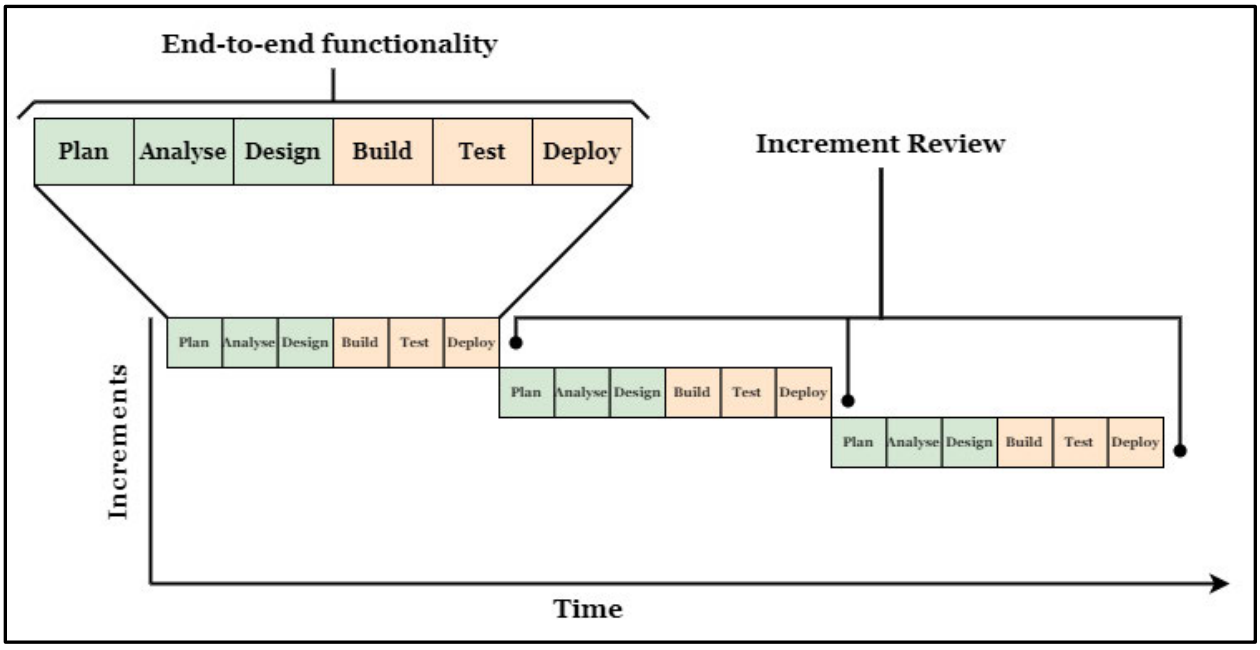

Figure 4: Agile Methodology

Figure 4 above shows the stages within an Agile increment and how the increments fit into the timeline of software development alongside end-of-increment reviews. The following sub-section on Scrum will detail how this Agile model fits into the project.

3.2.2 Agile Scrum Framework

Scrum is an agile framework that establishes set agile principles for use in project management. Within the context of this project and based on Scrum, a product backlog of software requirement shall be produced based on requirements elicited through "user interaction scenarios" (Bass, 2019, p. 62), referred to as use cases. Sub-section 3.3.1 "Use Cases" documents the use cases for the software developed during this project.

The previously used term "increments" in figure 4 are known as sprints in Scrum. Each sprint works on a portion of the product backlog, a sprint backlog, for completion within that sprint. The duration of a sprint varies between projects. However, this project has three-week sprints, and each sprint will bring to fruition an end-to-end product feature or multiple features.

Additionally, at the end of each sprint, a review will take place that details progress up to that point and updates the product backlog. This review also provides a point in time where a demonstration of progress to the project supervisor can take place.

In table 2 below are the current details of each sprint.

Table 2: Sprint Dates

Sprint No.	Date From – To (dd/mm/yy)	Date Justifications
1	08/01/21 – 30/01/21	Objective 4 of the project
2	30/01/21 – 19/02/21	Objective 5 of the project
3	19/02/21 – 12/03/21	Objective 6 of the project
4	12/03/21 – 09/04/21	Objective 7 of the project

Table 2 above shows the dates of the software development sprints of the project accompanied by the justification for their dates.

3.3 Design

This section documents the various design standards and techniques used in the software development process, which leads to a well-defined structure based around requirements.

3.3.1 Use Case Diagram

Presented in Figure 5 below is a use case diagram consisting of software use cases for a product owner on Amazon and a researcher – referred to as actors. Both of whom are users of the final product.

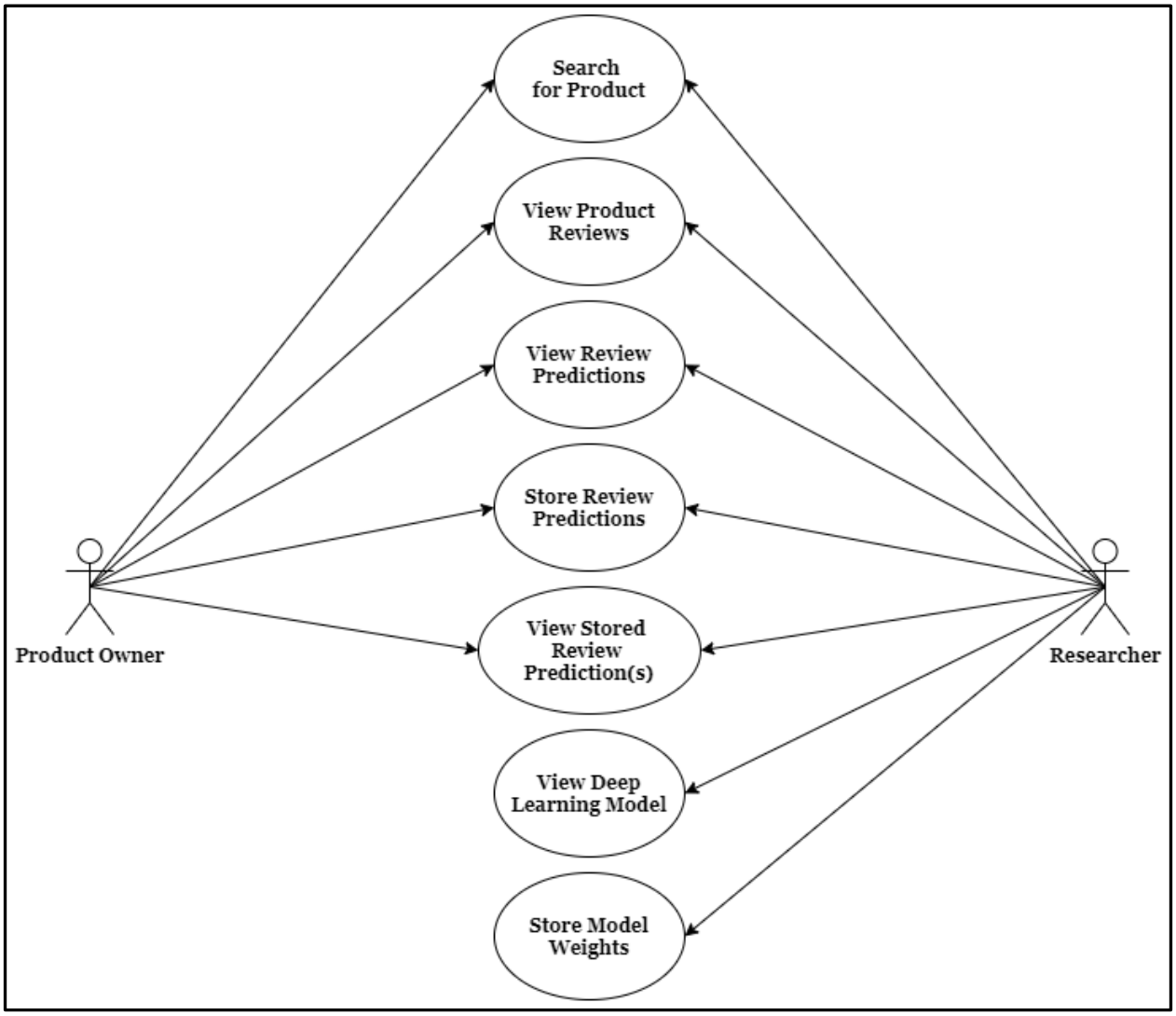

Figure 5: Use Case Diagram

The diagram in figure 5 above will help in creating use case descriptions, which are use cases in detailed tables in the next section.

3.3.2 Use Case Descriptions

Table 3: Search for Product

Use Case Title	1: Search for Product
Primary Actors	Product Owner, Researcher
Goal	Search for a product on the Amazon website
Scope	System
Preconditions	Amazon Standard Identification Number (ASIN) acquired.
Postconditions	Product with searched ASIN returns to the system
Main Success Scenario	1. Product ASIN is input into the search box 2. Product ASIN is searched for 3. Correct Product is returned
Extensions	2a. Invalid ASIN 3a. Wrong product is returned

Table 4: View Product Reviews

Use Case Title	2: View Product Reviews
Primary Actors	Product Owner, Researcher
Goal	View reviews associated with a product
Scope	System
Preconditions	Product search has returned correct product
Postconditions	Product reviews are displayed
Main Success Scenario	1. Receive request to display reviews 2. Reviews are formatted in a user-friendly manner 3. Reviews are displayed to screen
Extensions	1a. Product has no reviews

Table 5: View Review Predictions

Use Case Title	3: View Review Predictions
Primary Actors	Product Owner, Researcher
Goal	View the predicted sentiment of a product's reviews
Scope	System
Preconditions	Reviews are loading into the system The deep learning model is loaded
Postconditions	Product's reviews predictions are displayed
Main Success Scenario	1. The deep-learning model processes reviews 2. Review predictions are formatted in a user-friendly format 3. Review predictions are displayed on the screen
Extensions	3a. Displayed reviews are not displayed

Table 6: Store Review Predictions

Use Case Title	4: Store Review Predictions
Primary Actors	Product Owner, Researcher
Goal	Store displayed predictions into a local file
Scope	System
Preconditions	Review data exists in the system
Postconditions	Displayed reviews are stored by new line into a text file
Main Success Scenario	1. File saving directory is selected 2. Prediction data is ready to write to file 3. File is saved to local system
Extensions	1a. Directory does not exist 3a. File fails to save

Table 7: View Stored Review Predictions

Use Case Title	5: View Stored Review Predictions
Primary Actors	Product Owner, Researcher
Goal	View review prediction details stored in a previously saved data file
Scope	System
Preconditions	A file containing prediction data exists.
Postconditions	Review data for the product(s) are displayed on the screen
Main Success Scenario	1. File is loaded from the local system 2. File data is processed 3. Products' Reviews are accessible on screen
Extensions	1a. File does not exist 1b. Incorrect File 2a. Data has been modified to be incompatible with the system

Table 8: View Deep Learning Model

Use Case Title	6: View Deep Learning Model
Primary Actors	Researcher
Goal	View summary of deep-learning model's structure
Scope	System
Preconditions	Deep-learning model is loaded into the system
Postconditions	A summary of the deep-learning model' structure is displayed
Main Success Scenario	1. Deep-learning model's summary is requested 2. Deep learning model's summary is displayed on a screen
Extensions	1a. Summary request fails

Table 9: Store Model Weights

Use Case Title	7: Store Model Weights
Primary Actors	Researcher
Goal	To save the weights of the trained deep learning model to the local system within a directory of choice
Scope	System
Preconditions	Deep-learning model is loaded into the system
Postconditions	Deep learning model weights will be stored on the local system
Main Success Scenario	1. Deep-learning model's weights are requested to be saved 2. File saving directory is selected 3. Deep learning weights are saved to the selected directory
Extensions	2a. Directory does not exist 3a. System fails to save file

In the week before the commencement of the first sprint, using the above use case descriptions, a product backlog will be created and documented in the next report.

3.4 Investigative Methodologies

This project will investigate the relationship between review sentiment predictions and ratings to achieve its aim. This section will document the investigative approaches used.

3.4.1 Hypothesis

This project hypothesises that there exists a statistical relationship between the predicted sentiment of an Amazon review and its respective rating. Furthermore, this relationship can assist in improving the accuracy of heuristically labelling data.

3.4.2 Testing Methods

Testing this hypothesis will require a modified model based on the black-box problem's solution illustrated through Figure 1 in the introduction of this review.

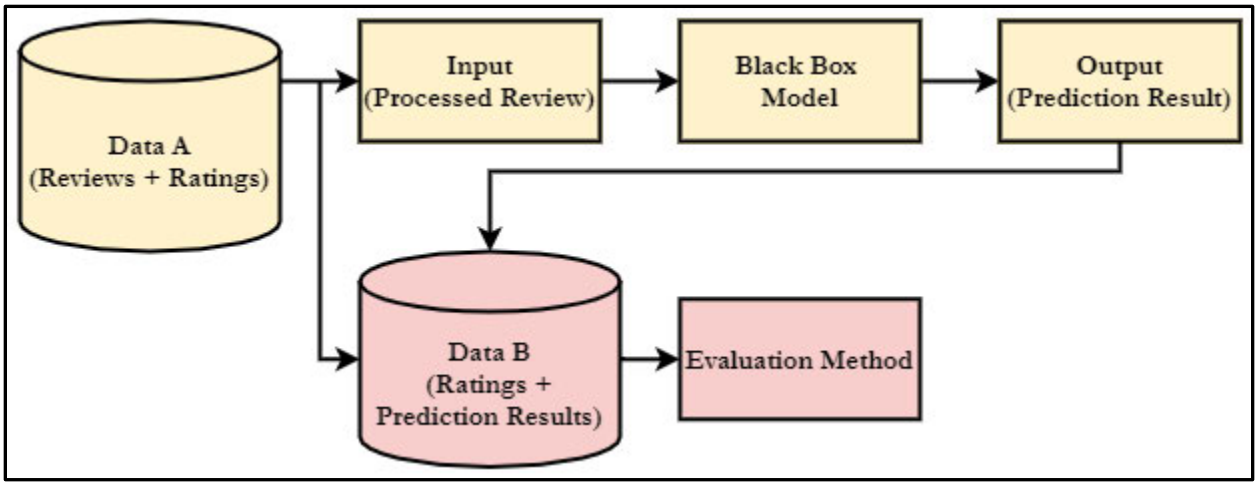

Figure 6: Investigation Model

Figure 6 above shows the process of capturing ratings and prediction results before moving to the evaluation stage. As the DL model predicts sentiment, product ratings do not enter the black-box model above; for this reason, it is necessary to store the ratings before data is formatted for entry into the model.

The data collected for this experiment consists of more than six-thousand unlabelled reviews, accounting for contingencies, stored in a text document.

The experiment will divide two-thousand unlabelled Amazon reviews into positive and negative reviews – above three-star as positive reviews, under three-star as negative reviews. The model then trains on half of the data provided until the model attains an acceptable level of accuracy, defined as seventy-five per cent.

Once the model has achieved this level of accuracy, the model will evaluate the unseen half of the data and save results in the container “Data B” shown in figure 6. The entire process is repeated on the data at least five times, retraining the model each time and collecting many instances of the said container as results for evaluation.

3.4.3 Evaluation Methods

Evaluating the gathered results involves plotting each result, consisting of a single instance of the container “Data B” in figure 6, onto a graph. Subsequently, this produces a collection of graphs based on many instances. A statistical analysis of the graphs aims to establish or refute a statistical relationship between review sentiments and ratings.

Understanding the reason(s) for the establishment or rebuttal of the relationship will, ultimately, inform designing a process to improve labelling data. An establishment will result in providing more accurate rating boundaries to label data upon while a rebuttal will result in questioning the impact heuristic data labelling has on training DL models.

In the event of establishing a relationship, testing new rating boundaries based on the relationship, with the proposed testing methods, shall take place – evaluating the results against the previous experiment.

In the event of rebutting a relationship, a portion of data labelling will occur manually, said portion would go through the proposed testing methods – evaluating the results against the previous experiment.

The results, in either case, will answer the aim of this project which is to determine the viability of SA in predicting products’ ratings.

4 – Revisions

This section documents revisions on the previous work that meaningfully contribute to the work ahead and clarify early obscurities presented towards the start of the project. Any project adjustments or developments found belonging to later sections of the report, such as software design, are elaborated upon in their respective sections.

4.1 Revisions – Introduction

This section details context added to the project's contribution and a change in a single objective. This contribution adds clarity to the original contribution, which stated that the project aims to determine the viability of SA in predicting products' ratings – the meaning for which is unclear as the context is left out. The objective revises a single word due to better understanding user requirements.

4.1.1 Contribution Revision

This project aims to justify the removal of three-star review data, suggested by the literature, in the binary sentiment classification of Amazon reviews using sentiment analysis (SA) with deep learning (DL). An investigation into Amazon review predictions and their respective accuracies shall produce the desired results to achieve this aim.

4.1.2 Objective Six Revision

Only objective six of the project required a single word adjustment.

Old objective six:

"To demonstrate a solution to the supervisor that can query the above-trained network with newly requested examples, aiming to display **accuracy**-related insights on said examples by 12/03/2021"

New objective six:

"To demonstrate a solution to the supervisor that can query the above-trained network with newly requested examples, aiming to display **sentiment**-related insights on said examples by 12/03/2021"

This adjustment is due to the user of the project's software not requiring the accuracy of the prediction but the prediction itself.

4.2 Revision – Development Methodologies

This section presents a modification to the sprint dates documented in the previous work.

4.2.1 Sprint Dates (M)

Table 10: Revised Sprint Dates

Sprint No.	Date From – To (dd/mm/yy)	Date Justifications
1	01/02/21 – 19/02/21	Objective 5 of the project
2	22/02/21 – 12/03/21	Objective 6 of the project
3	15/03/21 – 09/04/21	Objective 7 of the project

Table 10 above details the final sprint dates and removal of what was previously sprint one as application development starts at the revised sprint one start date. The sprint date adjustments also provide two days for sprint reviews during development as, previously, this was not possible.

5 – Requirements Specification and Design

This section documents the finalised requirements and designs, with justifications, that served as the basis of the project's software and experiments' development.

5.1 Software Requirements

This section details software requirements based on an agile user journey structure and proceeds to elaborate on the requirements elicitation process resultant of this journey.

5.1.1 User Experience – User Stories

In agile, user stories are descriptions that aim to capture the desired experience users expect from software usage in a high-level manner with minimal technical jargon to provide clarity and structure to development. Furthermore, user stories are split between sprints – structuring development time and workloads of the project around them.

User stories derive from a software description provided by a software's product owner. Under normal circumstances, a product owner who is separate from the development process provides this description; however, in this project, both are one. Thus, presenting the following software description for the project that centres on providing development overlaps that accommodate for the project's experiments in addition to the user experience:

"The software must allow a product analyst (PA) to search and retrieve relevant reviews of an Amazon product based on a unique identifier. Retrieved reviews must present a prediction of their sentiment as positive or negative to the PA. Upon the PA's request, the software must be able to save and load acquired search results."

To inform the projects development time and work commitments in sprints, the description above splits into the following user stories:

- As a PA, they must be able to search for Amazon product reviews to view the most relevant reviews of the product
- As a PA, they must be able to view the predicted sentiment of searched amazon reviews to know if they are positive or negative
- As a PA, they must be able to store retrieved Amazon reviews, with their sentiment predictions, for access at a later point in time

These stories break down further into use cases in the coming section.

5.1.2 Use Cases (Revised)

Section 3.3 of the report details use cases that reflect an early conceptualisation for the software. This section provides the finalised use case revisions based on the application development cycle – not early conception.

The user stories mentioned in section 5.1.1 are not enough to constitute a product backlog from which development can commence. Other entities contributing to the interaction within the software system, referred to as actors, must be identified beyond the product analyst.

The identification process produced the diagram shown in Figure 7 below and based on the actors' role and their associated simple verb-noun interactions with the overall system.

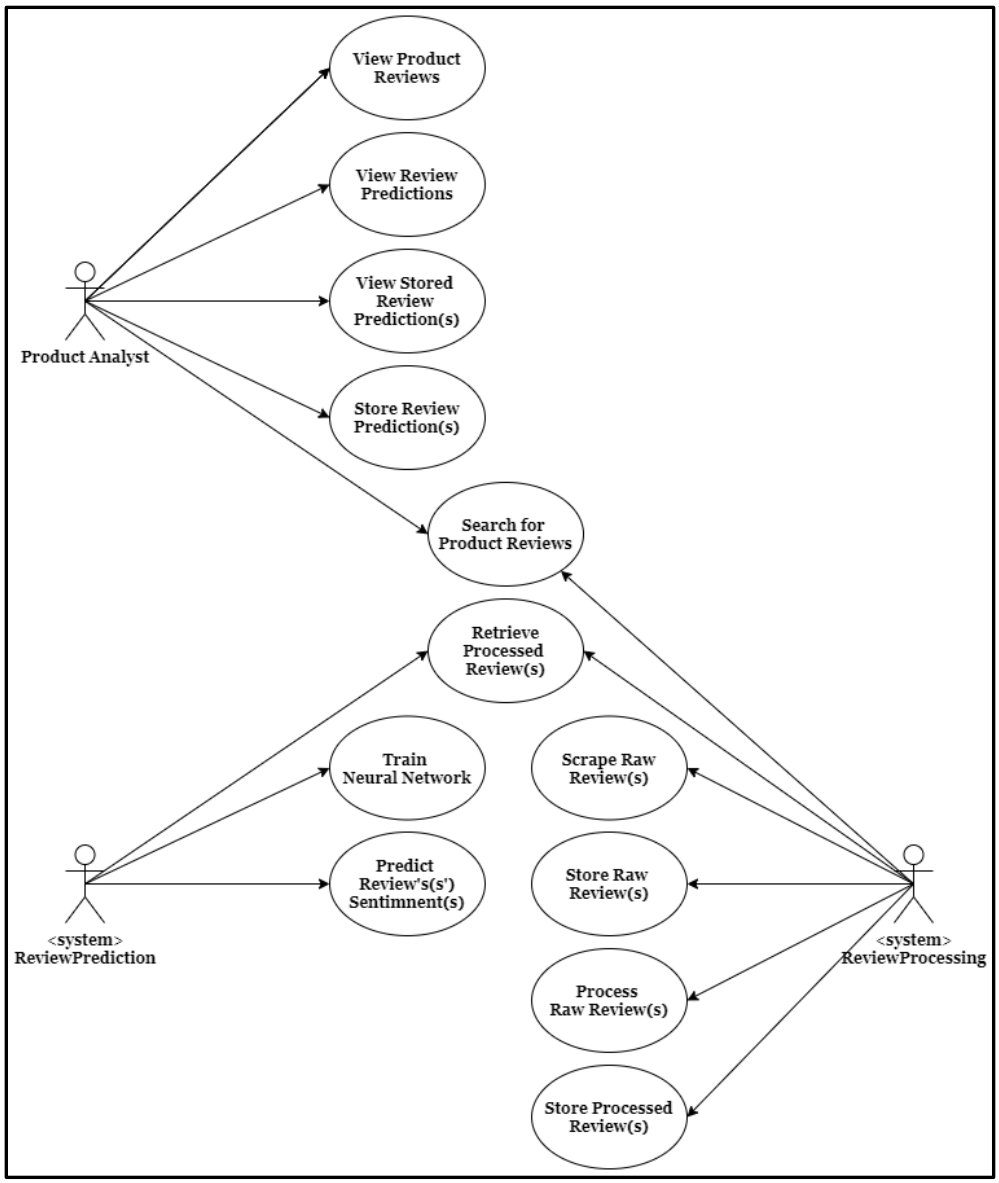

Figure 7: Use Case Diagram - Final

In Figure 7, the stick figures denote actors, and ellipses pointed to are their respective interactions – which two actors may have in common. The basis for the actors and interactions in the diagram lies in three main requirement areas identified; user interaction, review processing and review prediction.

User interaction concerns itself with user inputs and usage of the system. Review processing facilitates the gathering and processing of reviews for storage or passing them to other areas for further use. Review prediction allows for preparing data for use in a deep learning model and predicting newly acquired data based on a pre-trained model.

The review prediction system also accommodates experimentation with deep learning models, which becomes apparent in a later section discussing experiment design. The reason for excluding experimentation from software design, despite an implementation overlap, is that the experiment functionalities are reserved for project use and excluded from software deployment.

5.1.3 Use Case Descriptions (Revised)

This section features revisions on, additions to, and removal of use case descriptions detailed in section 3.3.2 of the report and serves as their replacement. These descriptions derive from the use cases in Figure 7, found in section 5.1.2.

Table 11: Search for Product Reviews

Use Case Title	1: Search for Product Reviews
Primary Actors	Product Analyst, ReviewProcessing
Goal	Search for a product on the Amazon website
Scope	System, ReviewProcessing System
Preconditions	Amazon Standard Identification Number (ASIN) acquired.
Postconditions	Product with searched ASIN returns to the system
Main Success Scenario	1. Product ASIN is input into the search box 2. Product ASIN is searched for 3. Product Reviews are returned
Extensions	2a. Invalid ASIN 3a. Wrong product is returned

Table 12: View Product Reviews (Revised)

Use Case Title	2: View Product Reviews
Primary Actor	Product Analyst
Goal	View reviews associated with a product
Scope	System
Preconditions	Product search has returned correct product
Postconditions	Product reviews are displayed
Main Success Scenario	1. Receive request to display reviews 2. Reviews are formatted in a user-friendly manner 3. Reviews are displayed to screen
Extensions	1a. Product has no reviews

Table 13: View Review Predictions (Revised)

Use Case Title	3: View Review Predictions
Primary Actor	Product Analyst
Goal	View the predicted sentiment of a product's reviews
Scope	System
Preconditions	Reviews are loading into the system The deep learning model is loaded
Postconditions	Product's reviews predictions are displayed
Main Success Scenario	1. The deep-learning model processes reviews 2. Review predictions are formatted in a user-friendly format 3. Review predictions are displayed on the screen
Extensions	3a. Displayed reviews are not displayed

Table 14: Store Review Prediction(s)(Revised)

Use Case Title	4: Store Review Prediction(s)
Primary Actor	Product Analyst
Goal	Store displayed predictions into a local file
Scope	System
Preconditions	Review data exists in the system
Postconditions	Displayed reviews are stored by new line into a text file
Main Success Scenario	1. File saving directory is selected 2. Prediction data is ready to write to file 3. File is saved to local system
Extensions	1a. Directory does not exist 3a. File fails to save

Table 15: View Stored Review Prediction(s)(Revised)

Use Case Title	5: View Stored Review Prediction(s)
Primary Actor	Product Analyst
Goal	View review prediction details stored in a previously saved data file
Scope	System
Preconditions	A file containing prediction data exists.
Postconditions	Review data for the product(s) are displayed on the screen
Main Success Scenario	1. File is loaded from the local system 2. File data is processed 3. Products' Reviews are accessible on screen
Extensions	1a. File does not exist 1b. Incorrect File 2a. Data has been modified to be incompatible with the system

Table 16: Retrieve Processed Review(s)

Use Case Title	6: Retrieve Processed Review(s)
Primary Actors	ReviewPrediction, ReviewProcessing
Goal	To retrieve raw review data from a previously stored file
Scope	ReviewPrediction System, ReviewProcessing System
Preconditions	A file containing review data exists. Request for processed review exists.
Postconditions	Review data loaded into the system for use
Main Success Scenario	1. File is loaded from the local system 2. File data is processed 3. Products' Reviews are accessible to the involved systems
Extensions	1a. File does not exist 1b. Incorrect File 2a. Data has been modified to be incompatible with the system

Table 17: Train Neural Network

Use Case Title	7: Train Neural Network
Primary Actors	ReviewPrediction
Goal	To train a neural network for later use in sentiment classification
Scope	ReviewPrediction System
Preconditions	Compatible review data exist to train a neural network
Postconditions	A retrievable trained neural network exists for system use
Main Success Scenario	1. Neural Network is trained to 80%+ accuracy 2. Trained network is saved to local system
Extensions	2. Network fails to save to the local system

Table 18: Predict Review's(s') Sentiment(s)

Use Case Title	8: Predict Review's(s') Sentiment(s)
Primary Actors	ReviewPrediction
Goal	To provide predicted sentiments of review data to required systems
Scope	ReviewPrediction System
Preconditions	Compatible review data exists. Request for review prediction exists. Trained neural network model loaded for use in predictions
Postconditions	Sentiment prediction data provided to the requesting system
Main Success Scenario	1. Sentiments for review data predicted by the model 2. Data passed to required system
Extensions	2. Data is not passed to the required system

Table 19: Scrape Raw Review(s)

Use Case Title	9: Scrape Raw Review(s)
Primary Actors	ReviewProcessing
Goal	To collect live Amazon review data
Scope	ReviewProcessing System
Preconditions	Internet Connection is present. Request to scrape data is present. Valid Amazon Serial Identification Number is present
Postconditions	Raw review data provided to the requesting system
Main Success Scenario	1. Amazon product page found 2. Review data scraped 3. Review data returned for system use
Extensions	1. Amazon product page does not exist 2. No valid review data exists for the product

Table 20: Store Raw Review(s)

Use Case Title	10: Store Raw Review(s)
Primary Actors	ReviewProcessing
Goal	To store raw reviews
Scope	ReviewProcessing System
Preconditions	Use Case 9
Postconditions	Raw review data stored in the local system
Main Success Scenario	1. Raw review data is ready to write to file 2. File is saved to local system
Extensions	2a. File fails to save to the system 2b. Text is not encoded correctly

Table 21: Process Raw Review(s)

Use Case Title	11: Process Raw Review(s)
Primary Actors	ReviewProcessing
Goal	To label reviews with their rating, ready for writing to file
Scope	ReviewProcessing System
Preconditions	Use Case 9
Postconditions	Raw reviews turned into processed reviews ready for file write
Main Success Scenario	1. Processed reviews written to files
Extensions	1. Text not encoded correctly

Table 22: Store Processed Review(s)

Use Case Title	12: Store Processed Reviews(s)
Primary Actors	ReviewProcessing
Goal	To store review data from Use Case 11
Scope	ReviewProcessing System
Preconditions	Use Case 11
Postconditions	Processed review data stored in the local system
Main Success Scenario	1. Raw review data is ready to write to file 2. File is saved to local system
Extensions	2a. File fails to save to the system 2b. Text is not encoded correctly

The project incorporates the above descriptions into the software implementation – discussed in section 6 of the dissertation.

5.2 Software Design

This section discusses the decisions and justifications for the software's architecture choice, class design, and interface design.

5.2.1 Software Architecture

Based on the requirements gathered and previous experience, the software adopts a Model-View-Controller (MVC) architecture. MVC makes implementing data presentation to the user, logical changes to data handling, and general functional changes more manageable through separating data presentation from data handling. Furthermore, implementing this architecture allows for extensions in the model, which do not interact with the View or Controller, for use in the experimentation part of this project.

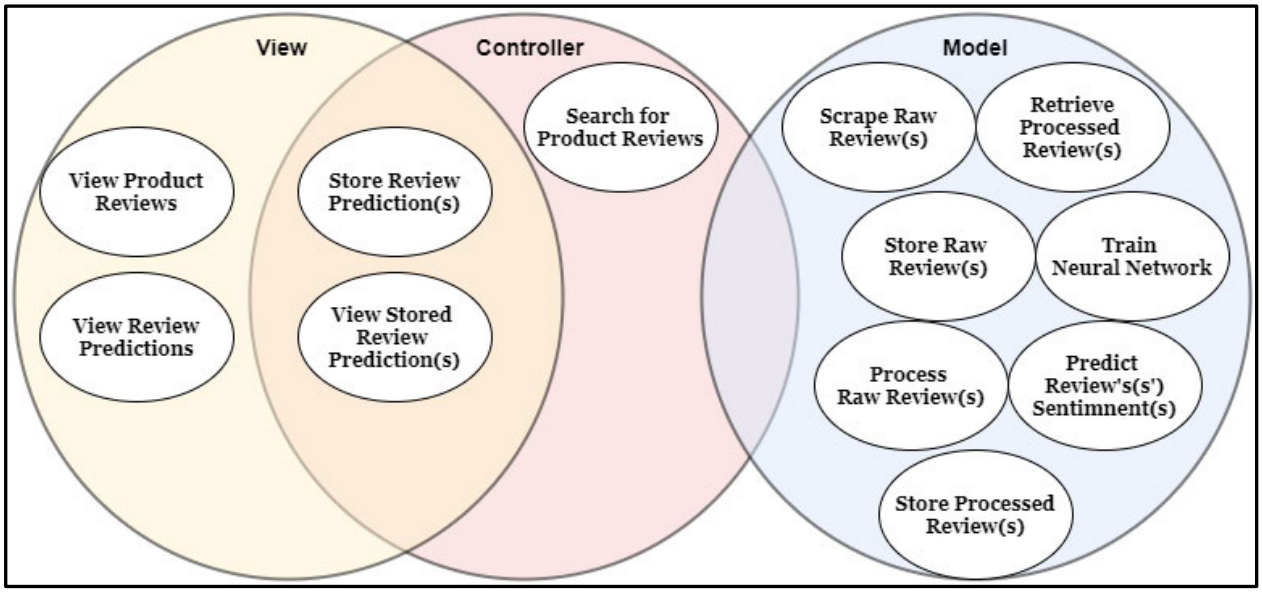

Figure 8: Venn Diagram for Architecture Justification

Figure 8 above justifies the architecture though allocating the respective use cases to their part in the software architecture. Searching for product reviews is the use case that links the Controller to both the View and Model. Two use cases fall in between the View and Controller; this is due to the interactions being handled solely between the View and Controller.

5.2.2 Class Design

Based on the MVC architecture, the software contains a View and Controller class. The Model portion of the architecture contains the classes; ReviewCollectorHTTP, NetworkModel, and ReviewUtils from early prototyping.

The View class formats the presentation of the software for the user and accepts user input. Should the user input request interactions with data already loaded into the View, the View independently handles the request; If no such data exists in the View, the Controller handles the request.

The Controller class takes requests from the View class and responds accordingly by consulting the Model classes for information given to the Controller and provided to the View for presentation. The Controller also contains self-contained functionalities such as saving and loading data from the View when requested.

The ReviewUtils class encapsulates general review-based functions which both the NetworkModel and ReviewCollectorHTTP classes inherit and may utilise. Functions include, but are not limited to, saving and loading reviews and processing reviews into a compatible format for further processing by the system at a later point – usually by the NetworkModel class; however, they are made available to both subclasses for usability and future expansion.

The ReviewCollectorHTTP class encapsulates all the necessary functions related to gathering review data from the Amazon website and either passing them to another function or saving them to a master file – for use in experimentation.

The NetworkModel class encapsulates all functions related to processing review data for deep learning use and predicting the sentiment of presented data. For software use, the Controller receives predictions; for experimentation use, the NetworkModel retains the predictions for analysis.

In the Python language, variables and functions are, by default, public. There exists a way to make functions private; however, they are still accessible through Pythonic means. To avoid this unfamiliar pseudo-encapsulation and limitations thereof, variables and functions of classes remain public in the software.

Thus far, class design follows a conceptualisation from requirements and early prototyping. The following section, 5.2.3, presents a class diagram, in Figure 9, reflecting the final product. The diagram clarifies how classes interact and details their functions – leaving function parameters out for consistency in documentation as expected values for function parameters in Python are not specified and can be handled differently according to variable type.

Earlier prototypes of the software retained the same relationships and classes in Figure 9 with fewer variables and functions as the need for them had not arisen at an early stage.

5.2.3 Class Diagram

In Figure 9 below, a class diagram representing the final product.

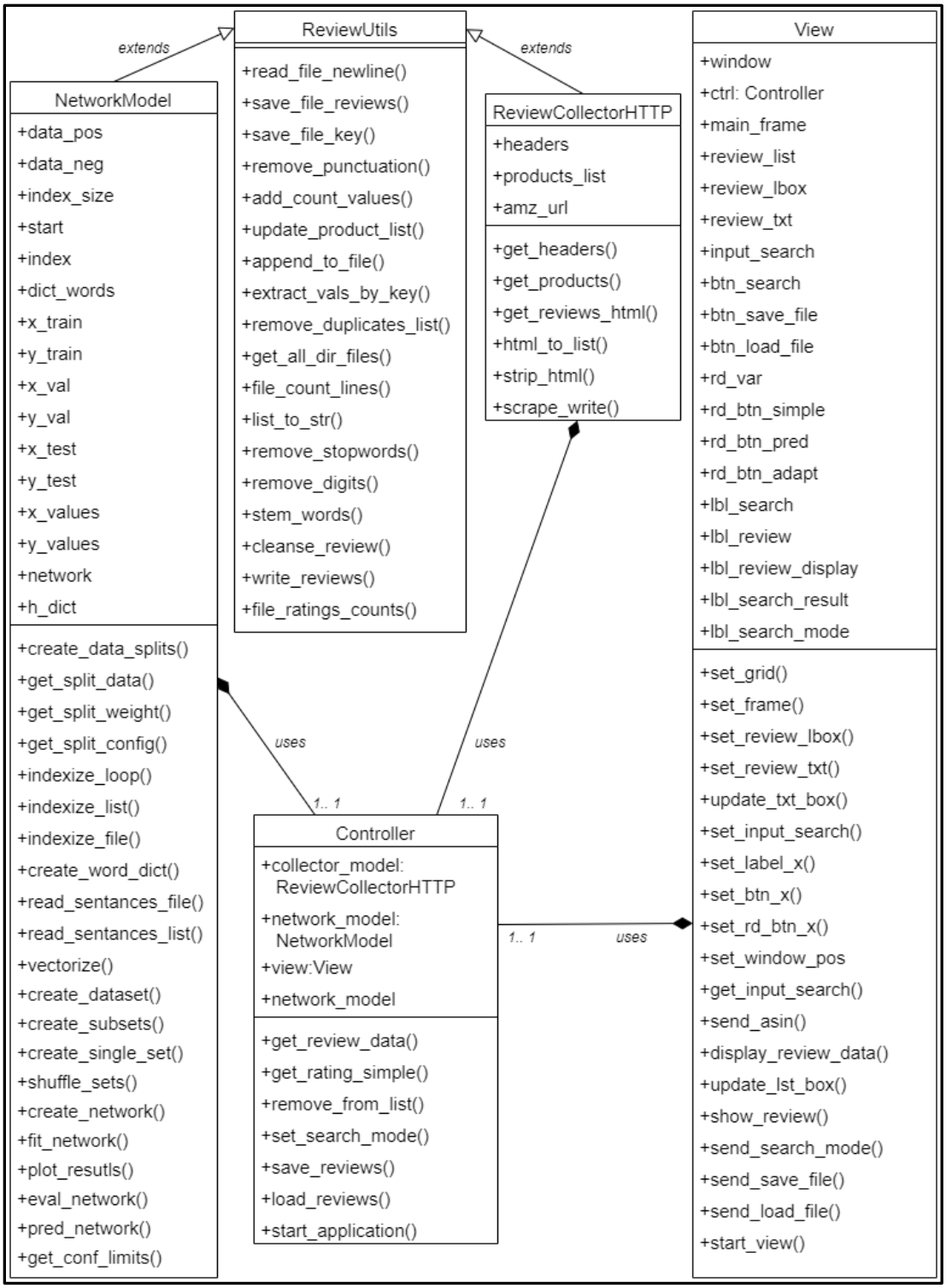

Figure 9: Class Diagram

5.2.4 Interface Design

The design of the software's interface reflects the desire to keep user input and display locations clear and intuitive and leave some space for expansion – resulting in the design shown in Figure 10 below.

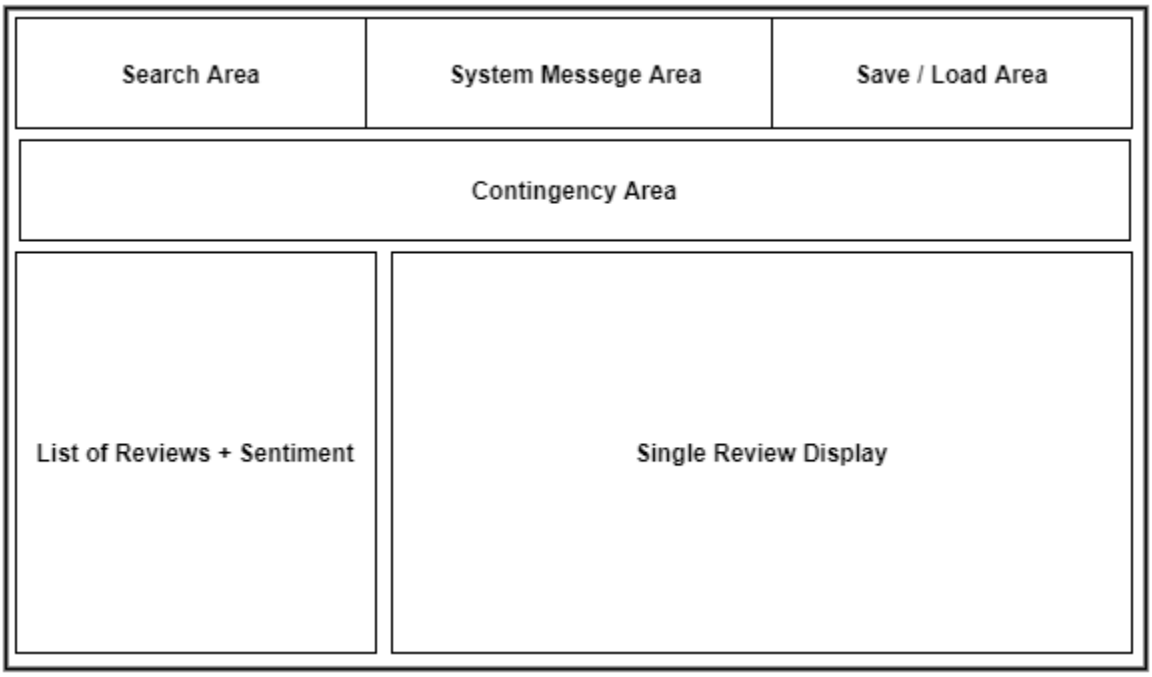

Figure 10: Interface Design

The search area is where the user may enter their search query in a text box and press a button to enter it into the system. The system message area is reserved to provide the user with feedback in regards to their input. Users may save or load review data through buttons in the save load area.

The contingency area remains a space to include future features. The list of reviews area displays results from the system from which the user may select an entry to view an individual review in the single review display area.

5.5 Summary

In summary, this section details how the software has progressed from a description of desired outcomes to architecture with accompanying class and interface design.

6 Development and Implementation

This section elaborates on the project's software development and implementation from the perspective of classes and their functionalities, briefly described in section 5.2.2 and how they satisfy the use cases described in section 5.1.3.

This section does not follow the chronological order of development to make explicit links to the requirements each class meets. Additionally, the ReviewUtils class serves as a utility class for its children; subsequently, its children's subsections detail its relevant functionalities.

6.1 Class - ReviewCollectorHTTP

Responsible for the retrieval of review data from the Amazon website, the ReviewcollectorHTTP class' final state of implementation falls into two key areas:

- Emulating Human Behaviour
- Retrieval of Review Data

The following subsections contain the details on each area above.

6.1.1 Emulating Human Behaviour

Lacking experience in scraping web data, this class encountered an issue in retrieving amazon review data as amazon would block scrape attempts by flagging them as coming from a bot through returning non-review-related HTML. This block was due to a naive attempt at using Python's web requests library without considering anti-scrape mechanisms.

The class utilises web headers that make requests look like they come from web browsers to address this issue. While introducing this technique resulted in the retrieval of review-related HTML data, after scraping for five minutes, Amazon blocked further requests after receiving requests at inhuman speeds. Subsequently, the class also uses one to one and a half seconds delay between requests – resulting in the successful and repeatable retrieval of Amazon review data.

6.1.2 Retrieval of Review Data

Having addressed the issue of Amazon blocks, the focus of this class returns to processing the HTML data. From previous experience, HTML page sources are reviewable through inspection tools in modern browsers – in this case, Google Chrome. Inspecting Amazon product reviews in such a manner led to the discovery of the "review" tag, an identification tag Amazon uses to label reviews in the HTML source code of product pages.

The discovery of this tag allowed the class to extract reviews from collected HTML data and store them in a list data structure. Subsequently, by discovering tags for review text and ratings, the list iterates into a list of strings containing review text with their ratings – with each string represented as shown in Figure 11 below.

```
It surprisingly works better than the broken one <3. #rating=5
```

Figure 11: String Representation

Finally, depending on the request, the class may return the list for further processing, process it through its parent, or store it in a file through functionalities provided by its parent. Additionally, through its parent, files containing review data may be read into the system.

The core algorithm which defines this class's review collection process is as follows:

1. Variable: Reviews List
2. Amazon Standard Identification Number (ASIN) provided for search purposes
3. FOR five times (Arbitrary)
    a. Sleep for between 1 and 1.5 seconds
    b. Set web header
    c. Request HTML page based on ASIN provided
    d. Find reviews on the page and add them to Reviews List
4. IF Reviews List contains reviews,
    a. return Reviews List
5. ELSE
    a. return 0

The above two areas covered by this class satisfy the following requirements in section 5.1.3:

- Use Case 6 – this class can load review data through its parent
- Use Case 9 – this class can scrape review data from Amazon
- Use Case 10 – this class can store collected review data to a file
- Use Case 11 – this class can process review data further through parent functionalities
- Use Case 12 – this class can store processed review data to a file through parent

After collecting this data, the NetworkModel class may load the data from a file or pass it directly after collection to NetworkModel for use.

6.2 Class – NetworkModel

Responsible for delivering predictions of review data through a deep neural network (DNN), the NetworkModel class' final state of implementation falls into three key areas:

- Review Data Sanitisation
- Preprocessing Sanitised Data
- Experiments

The following subsections contain the details on each area above.

6.2.1 Review Data Sanitisation

Retrieving review data from a file or through ReviewCollectorHTTP presents this class with a list of reviews altered to the point shown in Figure 12 of section 6.1.2. However, following the findings in section 2.1.1 of this work, this class utilises its parent's functions to exclude components of the review that provide little to no information (Bhadane et al., 2015, p. 810) and stems them to "a common base form" (Manning et al., 2008, p. 32). Such components include stopwords which are words such as "it", "then", and "the".

Furthermore, the class adds a ":1" to the end of words, left after the above process, to make them enumerable for later numerical representation. It also tags the review as positive or negative by adding a polarity label at the end of the review string with ": positive" or ": negative" – replacing its rating. Though the class does not utilize this polarity label, it ensures tags are available should they be required in the future.

The above processes are essential as it reduces the time a DNN takes to train and excludes words that have no meaningful impact on learning or assessing the polarity of a review by the DNN.

Figure 13 on the following page illustrates the algorithm used to sanitise each review within a list of reviews, accompanied by the outputs at each step.

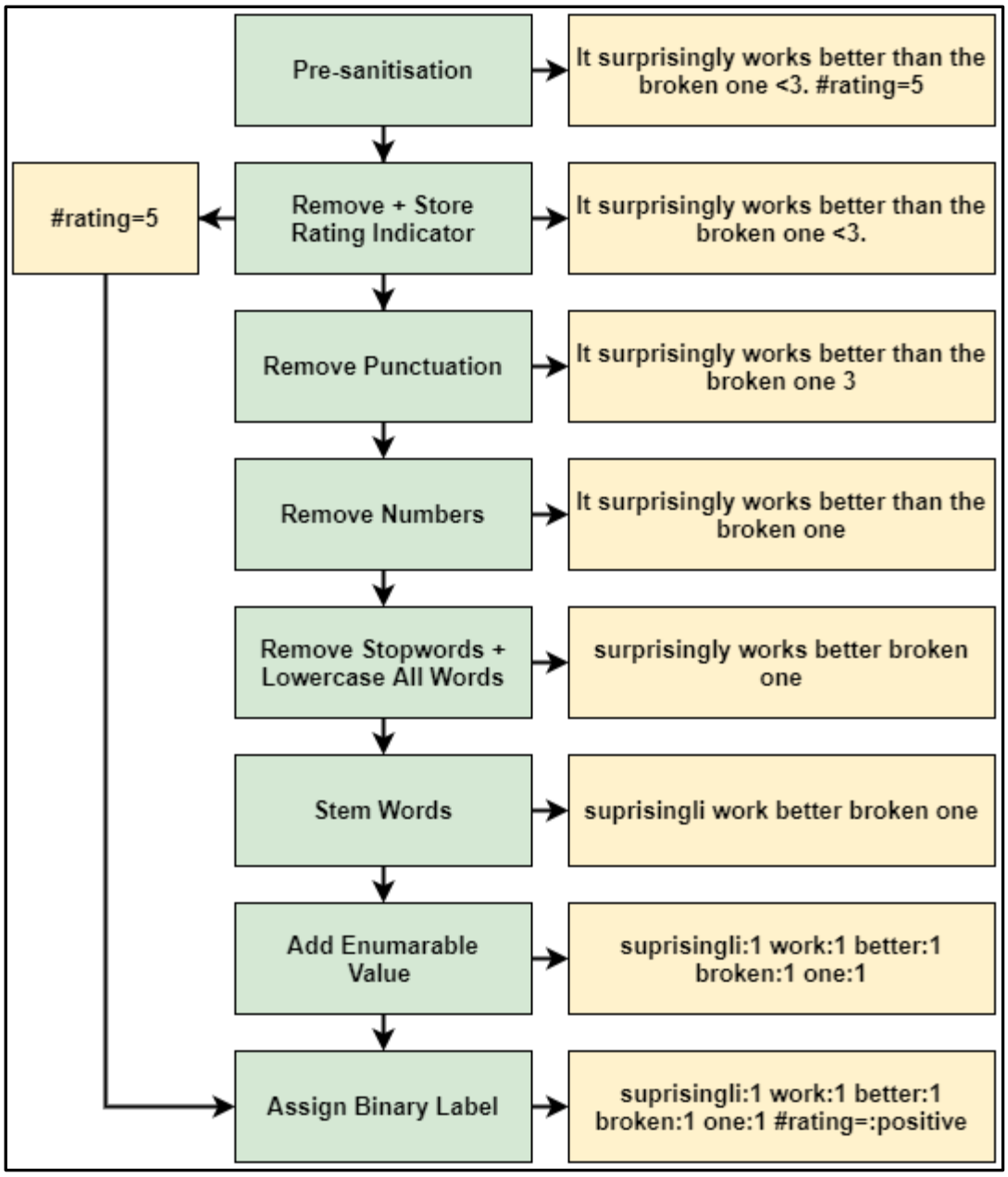

Figure 12: Data Sanitisation Algorithm

The peppermint colour represents the processes in Figure 12 with directional arrows towards successive steps and results — the above algorithm results in two lists for negative and positive reviews. The class allocates the reviews that go into each list by their rating – the rating at which the class determines this is three by default; however, it can change for experimentation purposes.

While most of this process occurs through using the powerful string manipulation tools in Python, removing stops words and the stemming process required using the list data structure due to having issues with implementing regular expressions in Python.

6.2.2 Feature Selection from Sanitised Data

After the process described in section 6.2.1, an algorithm within the class has two lists of reviews, positive and negative datasets, which require transforming into a form that a deep learning network can efficiently take as inputs without losing meaningful data – a process known as feature selection.

This subsection discusses a class instance with a single positive and negative review in their respective datasets for illustrative purposes whilst addressing feature selection, as shown in Figure 13 below.

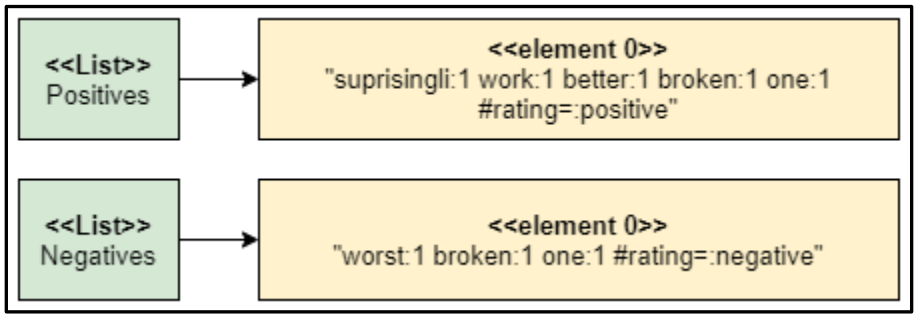

Figure 13: Two Review Lists

The algorithm first implements a dictionary data structure that uses unique words found in each list in Figure 13 as keys and their frequency in the datasets as values that serve as indicators towards their importance in the concept to be learnt by the neural network. Additionally, the algorithm sorts the dictionary in reverse order of frequency to provide greater significance to the less frequent words. It also replaces the count values in ascending order from zero being the least significant to the length of the dictionary minus one. Any words that have the same frequency are arbitrarily assigned their numeric value based on the order of computation. The results of the steps thus far resulted in the dictionary in Figure 14 below.

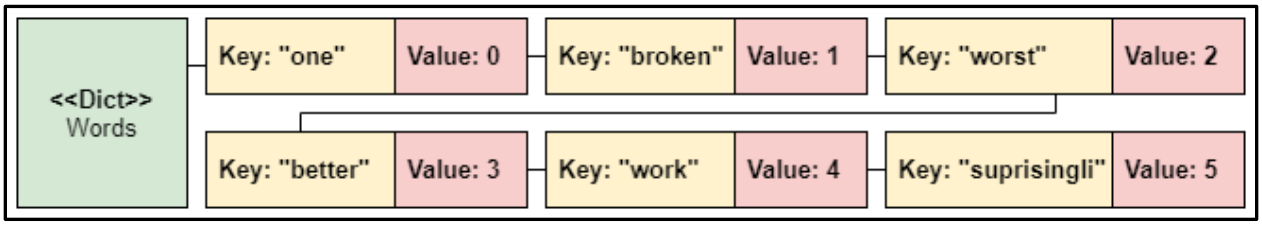

Figure 14: Word Frequency Dictionary

Above, the words “one” and “broken” are most frequent in the dataset, reflecting their values being the lowest – the goal of the algorithm thus far.

At this point, both datasets and the dictionary are in system memory. The algorithm then creates a Numpy (Harris et al., 2020) array of zeros equal to the dictionary's length. The created array allows for storing sentences as an array of ones and zeros by mapping their dictionary value to the indices of the array. This process, known as vectorisation, allows the neural network to compute epochs faster due to the Keras (Chollet, 2015) neural network APIs optimisations for this task.

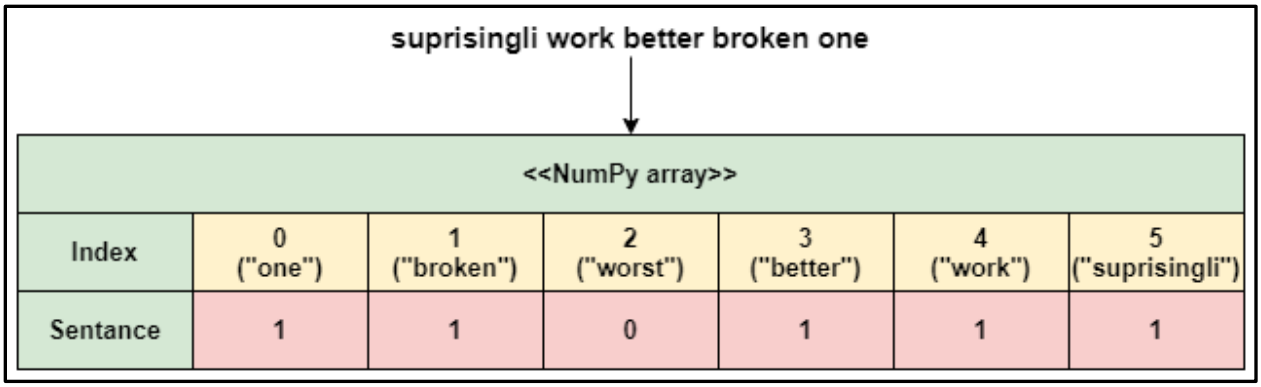

Figure 15: Words into vectors

In Figure 15 above, the algorithm distributes the sentence into a single entry in the array. The dictionary determines its index in the array, and a "1" represents the finding of a word. Words not in the dictionary do not appear in the array, thus, discarded. While Figure 16 showcases the results from a six-key dictionary, the software utilises a dictionary of ten thousand words for better training.

After vectorising both datasets, the algorithm creates two more Numpy arrays the length of each set, positive and negative, with zeros to represent "positive" in one and ones for "negative" in the other array. The algorithm creates these arrays to allow the neural network to know the concept it is trying to learn through validating against the respective labels in the arrays – confirming results as it trains.

The algorithm then combines datasets and labels and shuffles them to ensure no bias arises during network training. The algorithm ends by providing the neural network with the vectorized dataset containing the negative and positive data.

The network then trains on this data and is used to predict the sentiments of unseen data. The average accuracy of the deep neural network (DNN) configuration implemented in this class is 81.46% – comparable to those in literature (Dang et al., 2020, p. 21).

6.2.3 Experiments

While unrelated to the software requirements, this class also features options to configure data and the network for experiment use.

One such feature is changing the rating boundary though which reviews count as positive or negative; the boundary can be four-stars, three-stars, or two-stars. It also features the ability to include or exclude three-star review, the focal point of this project, from the dataset going to the DNN. During experimentation, the entire test set may also be of a particular class, such as one-star reviews only.

Furthermore, to better assess results, the ability to generate confidence intervals is present after supervisor feedback.

The above three areas covered by this class satisfy the following requirements in section 5.1.3:

- Use Case 6 – this class can load review data through its parent
- Use Case 7 – this class can train deep neural networks after processing data
- Use Case 8 – this class can predict the sentiments of review data and pass them on
- Use Case 11 – this class can also process review data further through parent functionalities

Concerning software use, when this class predicts reviews, they ultimately reach the user in the View class, which follows in the coming subsection.

6.3 Class – View

Responsible for interfacing the user with the software, the View class' final state of implementation falls into three key areas:

- Interface Layout
- Input
- Output

The following subsections contain the details on each area above.

6.3.1 Interface Layout

Inexperienced in using python graphical user interfaces (GUIs), this class uses web application design techniques, from previous experience, for its layout. Thus a 24x24 grid system makes the layout of the software's interface.

When laying out components such as buttons and text boxes, applying this technique resulted in misplaced components when the user resized the window. Difficult to debug as no inspection tools could look into the Python Tkinter (Hughes, 2000) library's variables; the class implements a fixed widow size to save time developing visual features.

Shown in Figure 16 below is the final result of this layout when the software launches.

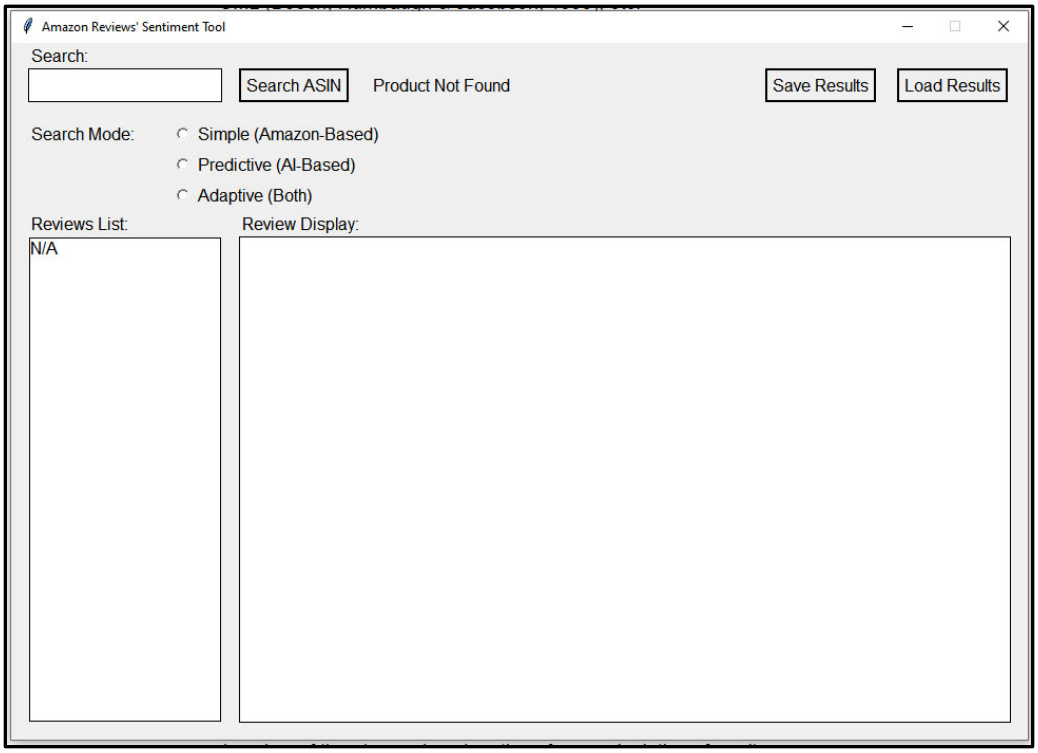

Figure 16: User Interface

The upper portion of the interface in Figure 16 contains inputs and the large areas below house outputs. The coming subsections discuss these areas.

6.3.2 Input

This class presents the user with four input options, all of which request the Controller to take action based on the input provided.

Firstly, the user may search for Amazon product reviews through the search box that accepts text input and a button to search – found at the top left of Figure 16.

Secondly, the user may choose their search mode from a selection of radio buttons under the search box to show that the input expects a single choice.

Thirdly, a “Save Results” button is to the top right of Figure 16 to suggest that the user can request to save review prediction results.

Finally, next to the previous button, a “Load Results” button suggests the user may request to load previously saved results.

6.3.3 Output

The user sees two large boxes under the inputs. The left box is a list box that the View may use to contain a reviews list that the Controller sends back, and the text box to the right of it updates based on the user selection from the list box. Furthermore, another output is a system message text box, which sits between the “search” and “save” buttons, shown in Figure 16.

The above three areas covered by this class, in combination with the Controller class, satisfy the following requirements in section 5.1.3:

- Use Case 1 – this class can request a product reviews search from the Controller
- Use Case 2 – this class can view product reviews sent from the Controller
- Use case 3 – this class can view review predictions sent from the Controller
- Use Case 4 – this class can store product reviews on request through the Controller
- Use Case 5 – this class can retrieve stored review predictions through the Controller

All in all, this class relies on the Controller class to respond to input requests provide feedback for outputs. Due to this, the Controller follows up in the next section.

6.3 Class – Controller

Responsible for handling user requests and consulting the Model, the Controller class' final state of implementation falls into a single area – managing user requests. The subsection below discusses this area.

6.3.1 Managing User Requests

The controller class, in essence, serves as a bridge between the User and Model Classes. When a user requests product reviews, the controller class forwards this request to Model functions and formats the Model's data to Update the View.

Two self-contained functionalities of this class are the ability to load and save review data upon user request. Python's Tkinter library handles these functions by providing user dialogue windows which the Controller then validates to prevent errors.

One of the core issues this class encountered was the need to load the application swiftly without training a model at launch. A pre-trained model based on experimentation is accessible and loads when the application launches to address this issue.

Another issue was that network predictions alone did not suffice in yielding results for the user as domain-specific jargon or concise reviews would result in inaccurate sentiment predictions. Through consulting the project's supervisor, the Controller provides multiple modes in which a user receives sentiment predictions.

One option is to use a default of anything above a three-star rating is counted as positive, and below negative. Another option utilises the neural network to predict all reviews. The last option provides the neural network predictions only if reviews are of a certain length and the rest based on the first mode – thus adaptive.

The above area covered by this class satisfies the following requirements in section 5.1.3:

- Use Case 1 – allows the user to search for product reviews through View-Model Interactions
- Use Case 4 – allows the user to store predictions loaded in the View
- Use Case 5 – allows the user to load previously stored predictions

This lead to the summary of this section.

6.5 Summary

In summary, the Model-related classes serve to satisfy the back-end functionalities. The View class handles the front-end, and the Controller glues them together to complete the system. They all contribute together to meet all the requirements set out in section 5.1.3.

7 Testing and Analysis

This section details the testing of the final product software. Additionally, it presents and analyses the experimentation process and results used to answer this project's aim.

7.1 Software

This section documents tests based on a valid and invalid input structure. Each subsection covers aspects of the software a user may interact with and tests for an expected outcome against inputs, known as functional testing, making adjustments as necessary if tests fail or succeed with actionable results. Additionally, the tests in this section fall under white box testing as the tester knows the software's implementation.

7.1.1 User Search

User search is when the user enters an Amazon Standard Identification Number (ASIN) into the search box of the software and presses the search button. When this happens, the user should receive one of two outcomes: the search returns product's reviews or does not.

The problem with ASINs is that, while ten characters long, the mixture of letters and numbers always differs, making any catch-all regular expression match for this incompatible. Thus the system relies on a valid "review" HTML tag present in the scraping of data to confirm reviews exist for a product. The system validates if the input data is ten characters long to resolve this issue. Furthermore, the web address used to retrieve reviews is hard-coded; thus, the user cannot change it.

Each test considers; the user action, the input data, the outcome of the test, supporting evidence, whether the test passed or failed, and any need for further testing based on the results. This structure is made evident in the coming subsections.

7.1.1.1 Test 1 – Search Valid ASIN

User Action: The user inputs data into the search box that is an ASIN and presses the search button.

Input Data: "B08CH9P42B"

Outcome: The system returns valid product review data and informs the user of "Product Found".

Evidence:

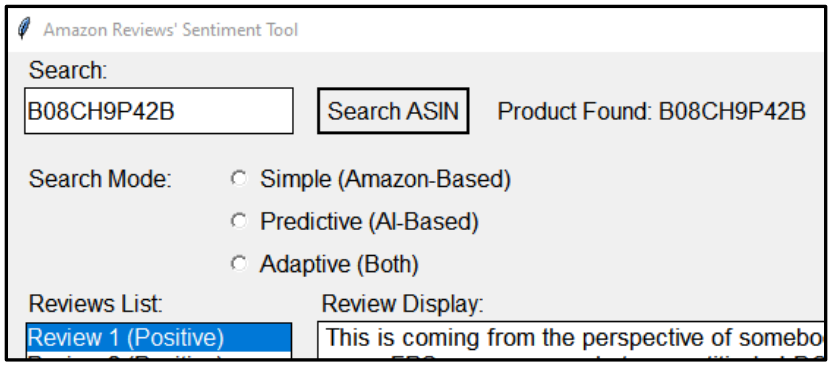

Figure 17: Test 1 Evidence

Pass/Fail: Pass

Further Actions: None

7.1.1.2 Test 2 – Search Invalid Data

User Action: The user inputs data into the search box that is not an ASIN and presses the search button.

Input Data: "asdasd", "12345"

Outcome: The system does not return any review data and informs the user of "Product Not Found".

Evidence:

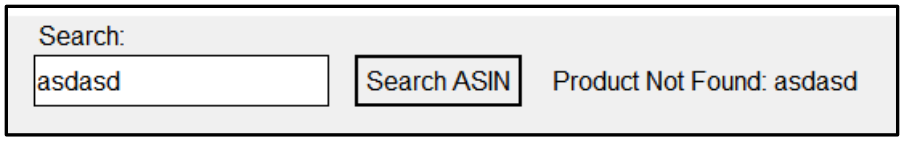

Figure 19: Test 2 Evidence A

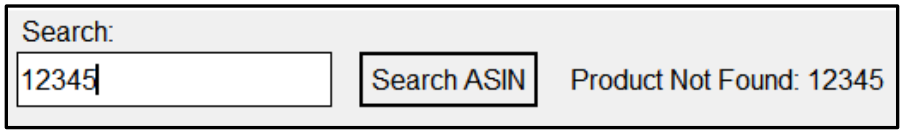

Figure 18: Test 2 Evidence B

Pass/Fail: Pass

Further Actions: None

The search component of the software has passed the input tests. The following section presents testing on the results that come from this search.

7.1.2 User Search Results

User search results are the reviews that return from a valid user search. The user may choose between three modes of search that affect the reviews' sentiments. The three modes are simple, predictive, and adaptive. Firstly, the "simple" mode classes reviews rated three or higher as "positive" and three or lower as "negative" – the default on the Amazon website. Secondly, the "predictive" mode relies on the deep neural network (DNN) to class all reviews' sentiments. Finally, the "adaptive" mode uses the basic mode logic for review less than an arbitrary length, fifty characters, due to the assumption that it is insufficient data for a DNN and uses the second mode's logic for anything longer than fifty characters.

This subsection tests the above modes in the results they produce, given a valid user search that is the input for Test 1, "B08CH9P42B".

7.1.2.1 Test 3 – Search Mode: Simple

User Action: User selects “Simple” as their search mode before a valid search

Input Data: Simple (Amazon-Based) radio button

Outcome: The review list shows sentiments according to the “Simple” mode description

Evidence:

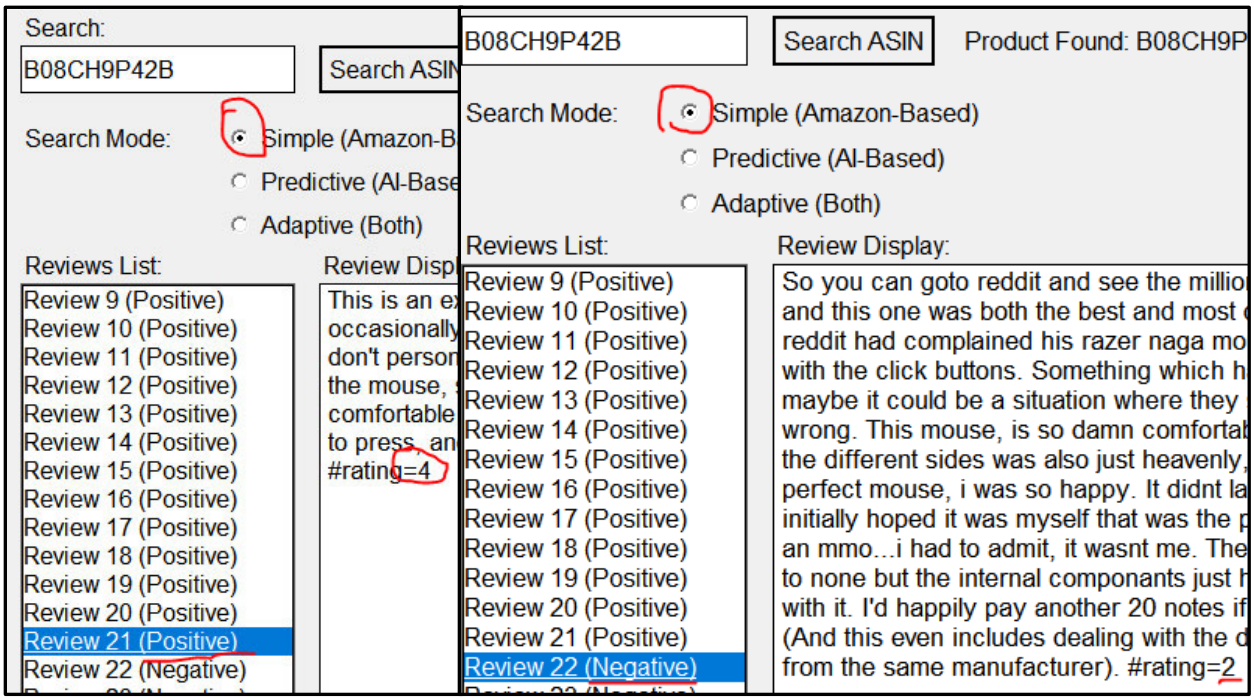

Figure 20: Test 3 Evidence A

Pass/Fail: Pass

Further Actions: None

7.1.2.2 Test 4 – Search Mode: Predictive

User Action: User selects “Predictive” as their search mode before a valid search

Input Data: Predictive (AI-Based) radio button

Outcome: The review list shows sentiments according to the “Predictive” mode description

Evidence:

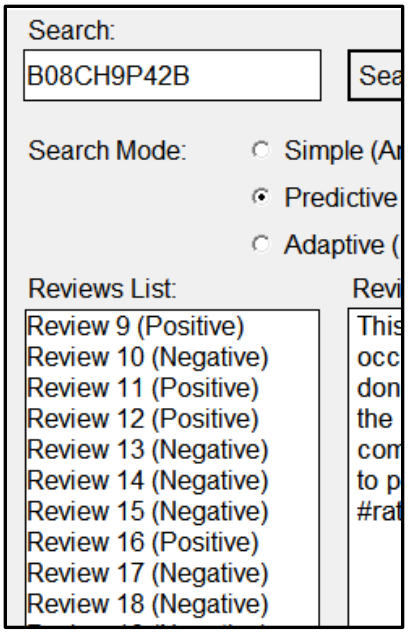

Figure 21: Test 4 Evidence A

Pass/Fail: Pass

Further Actions: None

7.1.2.3 Test 5 – Search Mode: Adaptive

User Action: User selects “Adaptive” as their search mode before a valid search

Input Data: Adaptive (Both) radio button, ASIN used in this case to highlight shorter reviews “B07CTSGQ16”

Outcome: The review list shows sentiments according to the “Adaptive” mode description

Evidence:

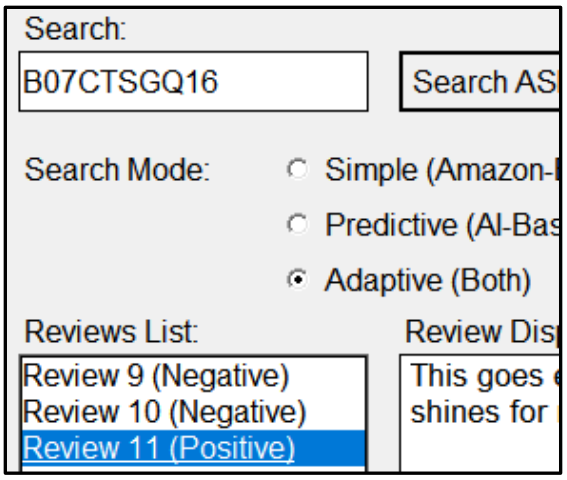

Figure 22: Test 5 Evidence A

Pass/Fail: Pass

Further Actions: None

7.1.3 File Save and Load

Given that users have successfully loaded their review data, the interface allows them to save and load the review data to their local system. The following subsections test these two functionalities based on valid and invalid data to read and write. Invalid data being either the absence of data or modified data.

7.1.3.1 Test 6 – Save File Valid

User Action: The user presses the "Save Results" button and uses the dialogue box to save review data

Input Data: Reviews List

Outcome: The system saved the review list to a file

Evidence:

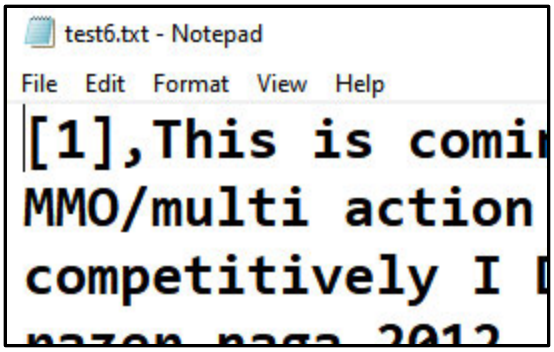

Figure 23: Test 6 Evidence A

Pass/Fail: Pass

Further Actions: None

7.1.3.2 Test 7 – Save File Invalid

User Action: The user presses the "Save Results" button and uses the dialogue box to save review data

Input Data: No Review List present

Outcome: The system saved gave an error message

Evidence:

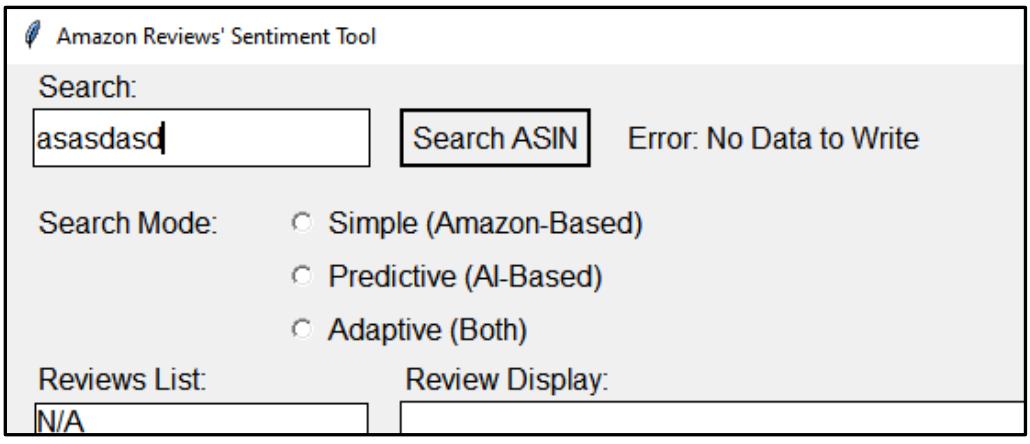

Figure 24: Test 7 Evidence A

Pass/Fail: Pass

Further Actions: None

7.1.3.3 Test 8 – Load File Valid

User Action: The user presses the “Load Results” button and uses the dialogue box to load review data

Input Data: Review File

Outcome: The system loaded the review list from a file

Evidence:

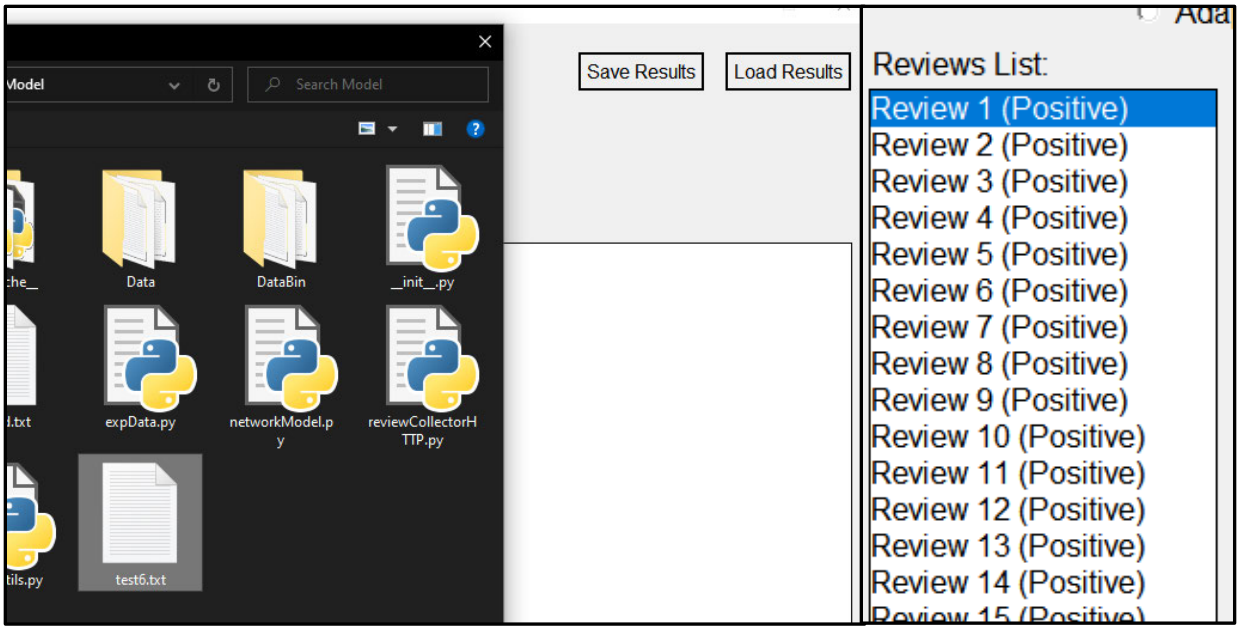

Figure 25: Test 8 Evidence A

Pass/Fail: Pass

Further Actions: None

7.1.3.4 Test 9 – Load File Invalid

User Action: The user presses the "Load Results" button and uses the dialogue box to load review data

Input Data: Wrong File Type

Outcome: The system saved gave an error message

Evidence:

Figure 26: Test 9 Evidence A

Pass/Fail: Pass

Further Actions: None

7.1.3.15 Test 10 – Load File Modified

User Action: The user presses the "Load Results" button and uses the dialogue box to load review data

Input Data: Review File Modified

Outcome: The system saved gave an error message

Evidence:

Figure 27: Test 10 Evidence A

Pass/Fail: Pass with the same result as Test 9

Further Actions: None, as the user should maintain the files without modification

7.2 Experiments

This section details the terms, data used, tests conducted and results obtained from the project's experiments. It also provides an analysis of the results. Additionally, it puts forward a null hypothesis and alternative hypothesis to form the basis of experimentation and any requirements before commencing experiments.

7.2.1 Terms

For clarity purposes, this subsection defines terms that may confuse or misdirect the content:

- **Class** – this term shall refer to outputs a deep neural network (DNN) is trying to learn and uses. This project concerns itself with to classes, positive and negative (sentiments).
- **Label** – this term shall refer to a category of data that filter into classes for DNN training. This project makes use of five such labels, which filter into the two classes during training:
    - Five-Star Reviews (always classed as positive)
    - Four-Star Reviews (always classed as positive)
    - Three-Star Reviews (may be classed as positive or negative)
    - Two-Star Reviews (always classed as negative)
    - One-Star Reviews (always classed as negative)

- **Training Set** – A dataset that a DNN uses to learn the problem area during training.
- **Validation Set** – A dataset that a DNN verifies against training data to update its weights during training.
- **Training Data** – training and validation sets combined
- **Test Set** – A dataset unseen by the network during training and should be from the same problem area as the training set.
- **Epoch** – a single instance of the deep learning algorithm going through the entire dataset during training a DNN
- **Model** – A term used to describe a trained DNN that can predict the class of unseen data based on its abstract representation of the problem.
- **Ambiguous Data** – 3-star review data, due to their polarity (negative or positive) being ambiguous (Blitzer et al., 2007, p. 442)
- **Non-ambiguous Data** – 5-star, 4-star, 2-star, and 1-star review data

With these terms in mind, the following sections detail the tests, results and analysis of the project's experiments.

7.2.2 Problem Description

The project aims to justify removing three-star review data from the training of mentioned deep learning models. The literature mentions the reasoning for removing the data is to provide “a clearer indication of positivity or negativity.” (Zhang & LeCun, 2015, p. 5) from the included review data (above three-star, below three-star). However, results to evaluate the impact on the models’ accuracies on non-ambiguous data when including ambiguous data in training such models does not exist. Thus, the project undertakes experiments to justify the removal of three-star reviews, either through validating literature or new conclusions drawn from experimentation.

7.2.3 Null Hypothesis

The inclusion of 3-star review data in training a binary deep learning-based sentiment classification model has no impact on the model's accuracy when determining the review polarity of non-ambiguous data.

7.2.4 Alternative Hypothesis

The inclusion of 3-star review data in training a binary deep learning-based sentiment classification model negatively impacts the model's accuracy when determining the review polarity of non-ambiguous data.

7.2.5 Experiments

In testing the above hypotheses, the project uses three models; one trained without ambiguous data, one trained with ambiguous data classed as positive, and the last trained with ambiguous data classed as negative. Each model utilises a dataset taken from equally distributed and shuffled classes during its training process of sixteen thousand samples of negative and positive reviews – eight thousand in the training set and eight thousand in the validation set. Furthermore, each experiment trains the neural network of each model for twenty epochs.

After that, the model predicts all the data from a test set containing one thousand of each label from the non-ambiguous data, four thousand in total. An evaluation of the results of the predictions provide data on the model’s accuracy and confidence interval.

Figure 28 below details the amounts of data used within each model – kept the same throughout experimentation.

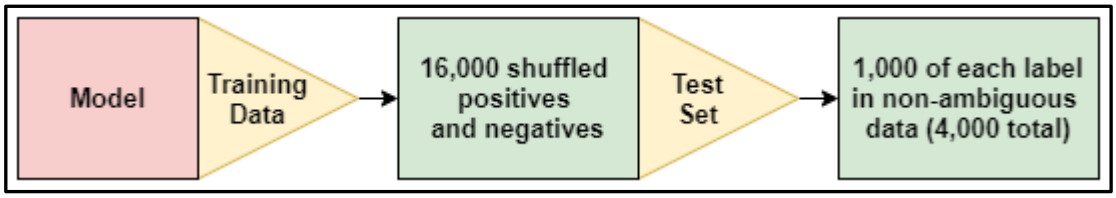

Figure 28: Experiment: Model's Quantities

Each model uses in the experiments follow the above quantities. While the testing set always features one thousand samples from each non-ambiguous label, the distribution of labels within the models training with ambiguous data differs as positives and negatives divide between five labels instead of just four.

The project conducts a total of three experiments. Each experiment creates ten instances of each model and evaluates the model on the test set – the average accuracy and 99.8% confidence intervals of these ten instances become the final result. Figure 29 presents the final results of the experiments.

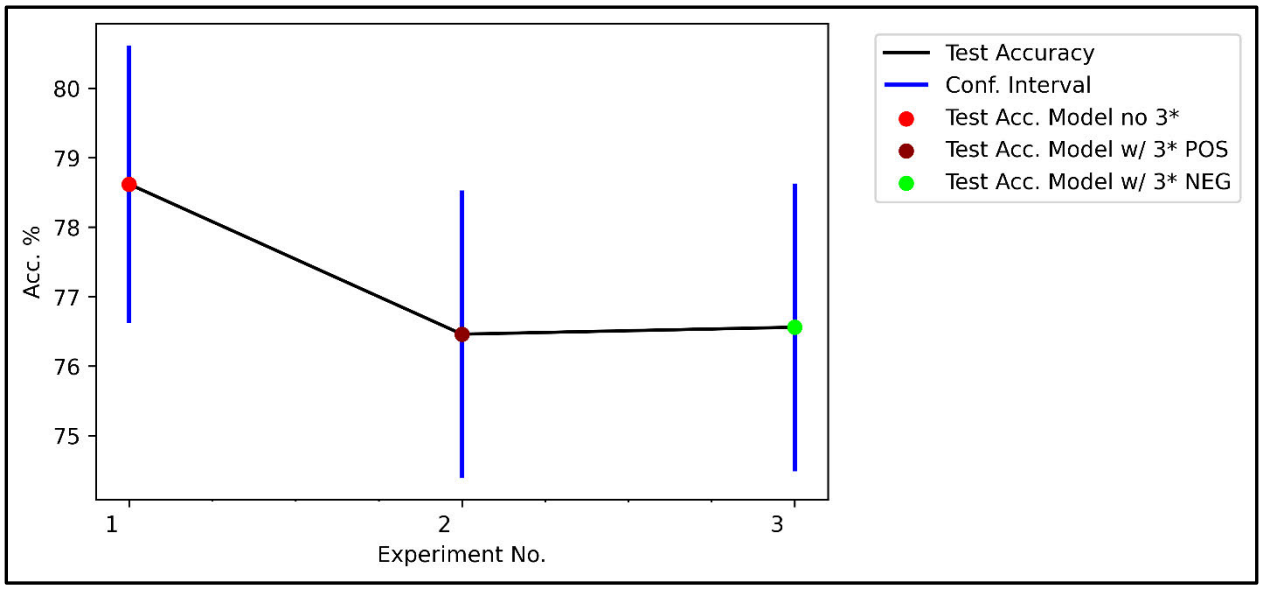

Figure 29: Experiment Results

In Figure 29, the black line follows the test accuracy average between models, and the blue vertical lines are the confidence intervals. Each dot indicates the average test accuracy of each model test, bright red being for the model trained without ambiguous data, darker red being for the model trained with ambiguous as positive, and lime for the model trained with ambiguous data as negative. See Appendix B for all experimental data.

Furthermore, Figure 29 shows that the difference between including ambiguous data (3-star review data) and not including the data is statistically insignificant when comparing the models' accuracy, not rejecting the null hypothesis and disproving the alternative hypothesis through the statistical comparison.

The project aimed to justify the removal of 3-star review data by comparing the models' accuracies; however, the results of the experiments have shown that, based on model accuracy, removing three-star data is not justified – thus, a negative result in light of the aim.

7.3 Summary

In summary, this section documents the various software functionality testing to ensure that it is running as expected – furthermore, it details the experimentation method and how they have related the project's aim to a negative result.

8 Critical Evaluation

This section provides a personal review of the project's objectives and time plan. Additionally, it reviews the product produced's strengths and weaknesses. Lastly, It details the lessons learnt and reflects on the project's first two deliverables.

8.1 Objectives

This subsection details the project's achievements against its objectives.

8.1.1 Objective One – Learning Sentiment Analysis with Deep Learning

Objective one:

> "To learn deep learning techniques and implementations that are involved in sentiment analysis to a sufficient capacity where a functional implementation with reasonably high accuracy can be demonstrated to the supervisor by 11/11/2020"

I started learning about sentiment analysis with deep learning from another module called "Deep Learning" at this point in the project. While the concepts learnt from that module at that stage allowed me to meet the objective, I needed to make changes through my previous software implementing experience to accommodate the encapsulation required when developing the product later on in the project. At this point, my implementation of a neural network achieved accuracy close to eighty per cent. Additionally, this was shown to my supervisor and confirmed to be reasonably accurate.

Additionally, as I progressed through the project, I needed to refactor the implementation to meet the architecture for my software and add functionality to control the experimental parameters used during the project's experimentations. Later on in the project, the neural network implementation learnt at this early stage was optimised further through experimentation and allowed me to use this in the project's product when predicting reviews.

While a lot of my focus went on getting a good accuracy, less attention focused on the data that went into the network. For instance, I realized it was more important to control how the data split between classes before entering the network, which directly contributed to the project's aim. However, at the time, I was trying to figure out what I wanted to do in the project, so the aim was kept broad enough to decide but precise enough to provide direction.

8.1.2 Objectives Two and Three – Data Collection

Objective two:

"To deliver a solution that can collect, create, and save new records of data consisting of Amazon product reviews, collected directly from Amazon's website, requiring minimal product identifiers during the collection process by 25/11/2020"

Objective Three:

"To provide a self-collected dataset of Amazon product reviews, targeted towards fifty-thousand records by 11/12/2020"

One of the key challenges faced during this project was creating a self-contained web scrapper that retrieved and stored data but, most importantly, was also open to refactoring. When completing the above abjective, the implementation could only retrieve data and write it to a file all in one function. This type of code is against good programming practice; however, I had minimal experience creating a web scraper and needed to get something that worked to meet the objective. This relatively quick manner of programming created a fair amount of technical debt for me later on in the project when having to refactor this code to fit into the architecture of the software; however, I set the objective completion dates with such prototyping in mind.

I had successfully gathered more than eighty thousand reviews using a list of Amazon products extracted from an external Amazon dataset (Blitzer et al., 2007). However, I realized that I was saving the reviews to a file in a preprocessed manner. For example, "amazinli:1 good:1 #rating=5". So while the objective was complete, I was not happy with how the reviews were saved to a file as they did not provide much flexibility in changing the preprocessing of data should this be required. Thus, I refactored the code to save the reviews without any preprocessing. The refactoring did mean that I had to recollect reviews; however, I could leave it running while completing other tasks since this process was fully automated.

I showed my supervisor the achievement of the above objectives and confirmed them to be complete through showing the process and collected data.

8.1.3 Objectives Four and Five – Neural Network

Objective Four:

"To demonstrate a proof-of-concept deep-learning network, on a reduced dataset that works towards being used on two-thousand records, to the supervisor by 30/01/2021"

Objective Five:

"To deliver a trained deep-learning network that uses six-thousand records with comparable relative accuracy to established networks in the subject area by 19/02/2021"

I managed to create a neural network in objective one with relatively high accuracy. However, that was using external data (Blitzer et al., 2007). For these objectives, I needed to format my collected data and ensure that it was fit for use in the network. Using the external data as a reference, I refactored code and ensured that my data and the external data were functionally the same at the point of neural network input. Once the functions to appropriately organize my data were in place, I managed to get the required data quantities into the network. For objective five, I also showed my supervisor the relatively high accuracy against other networks in the area (Dang et al., 2020, p. 21). Furthermore, starting objective five was when I started to refactor my implementation and go into agile sprints to develop the project's product.

8.1.4 Objectives Six and Seven – The Product

Objective Six:

"To demonstrate a solution to the supervisor that can query the above-trained network with newly requested examples, aiming to display sentiment-related insights on said examples by 12/03/2021"

Objective Seven:

"To provide user-friendly extensions to the solution above which have comparable usage experiences to software similar in nature by 09/04/2021"

From objective six onwards, work on the product continued. Concerning objective six, I managed to create a system that could show the sentiments of predicted reviews collected from Amazon; however, their results did not go to the front-end of the product.

For this reason, I decided to implement dummy data so the product could be displayed in an agile manner to represent the desired outcome and have actual results in the back-end. I met this objective through the back-end alone shown this to my supervisor for confirmation of achievement.

I showed the product to my supervisor after connecting the back-end to the front-end. Through this meeting, I found that it would be best to implement alternative modes to the prediction system for the user as accurate results may vary due to several factors such as review length and domain-specific jargon that the system had not seen before. I implemented two extra modes, as discussed previously in the dissertation, to accommodate this.

Objective seven proved challenging as I realized late that it would require usability testing due to user-friendliness. However, at this point, I had completed all practical work and also required changes to my experimentation that took up a lot of the extra time allocated to this project.

I can only contribute a fraction towards this objective by briefly self-evaluating its usability below in Figure 30. Ensuring all inputs and outputs are clearly labelled and describe their place in the product. The product shows clear labelling for the functionalities it provides.

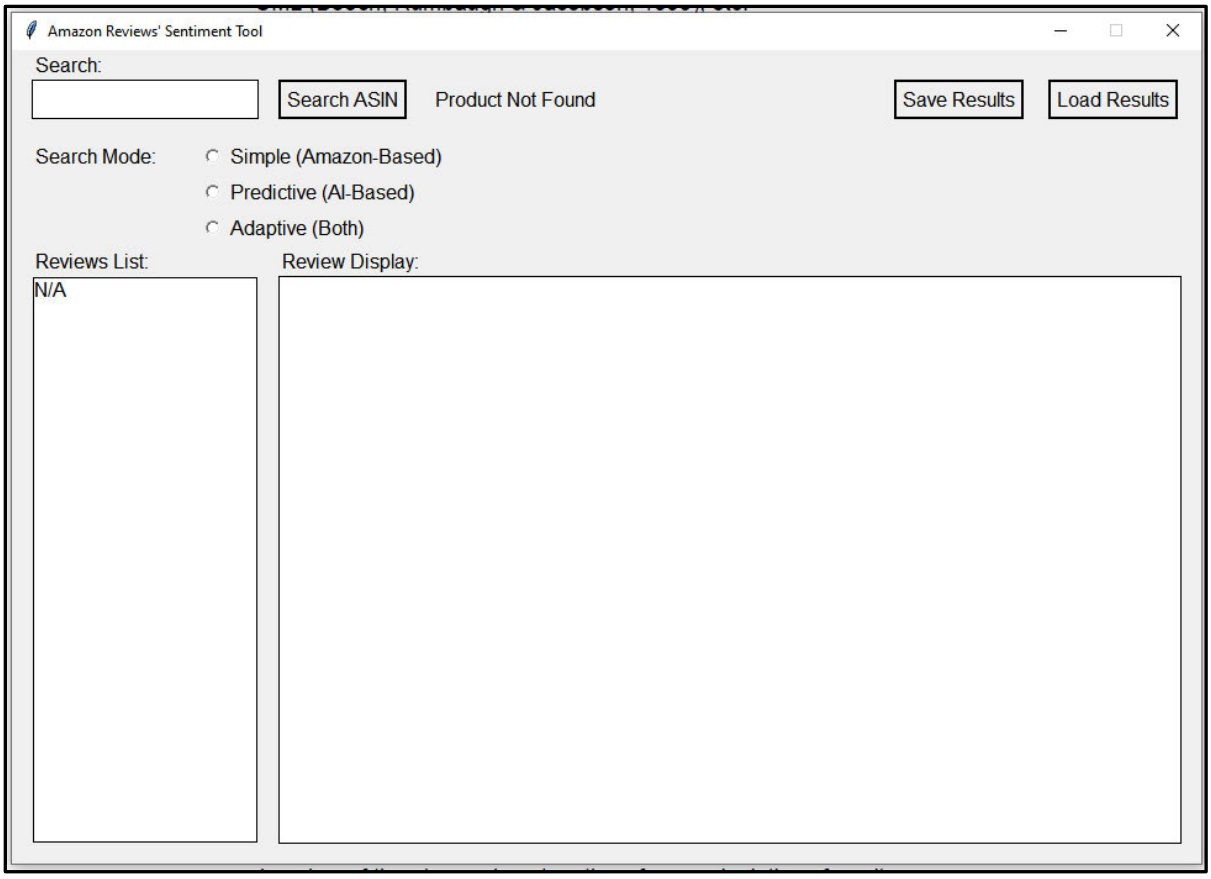

Figure 30: Self-evaluated Usability

8.1.5 Bonus Objectives

Bonus Objective One:

"To provide integrations into the system that accept different sources of data – one week from main objectives completion."

Bonus Objective Two:

"To provide usage insights between newly accepted data and previous data – two weeks from bonus objective one completion."

I did not complete any of the bonus objectives for the project. I formulated them when I did not realise the implications they had on the scope of practical work. It would have required collecting and formatting data from a new source or creating a system that allowed users to input data themselves. Their scope was too much work for the time I had leftover after correcting issues found towards the end of the project.

8.2 Review – Time Plan

For comparison, Figure 31 presents the original timeline proposed for the project.

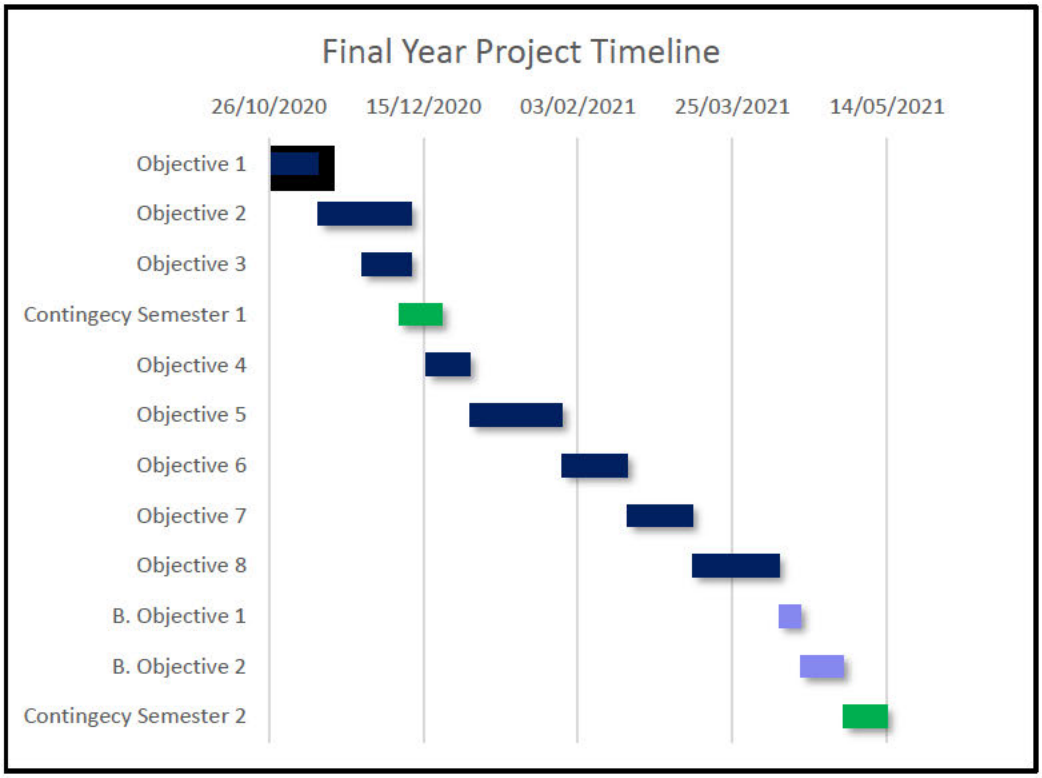

Figure 31: Timeline - Proposal

Figure 32 below presents the revised timeline for the project.

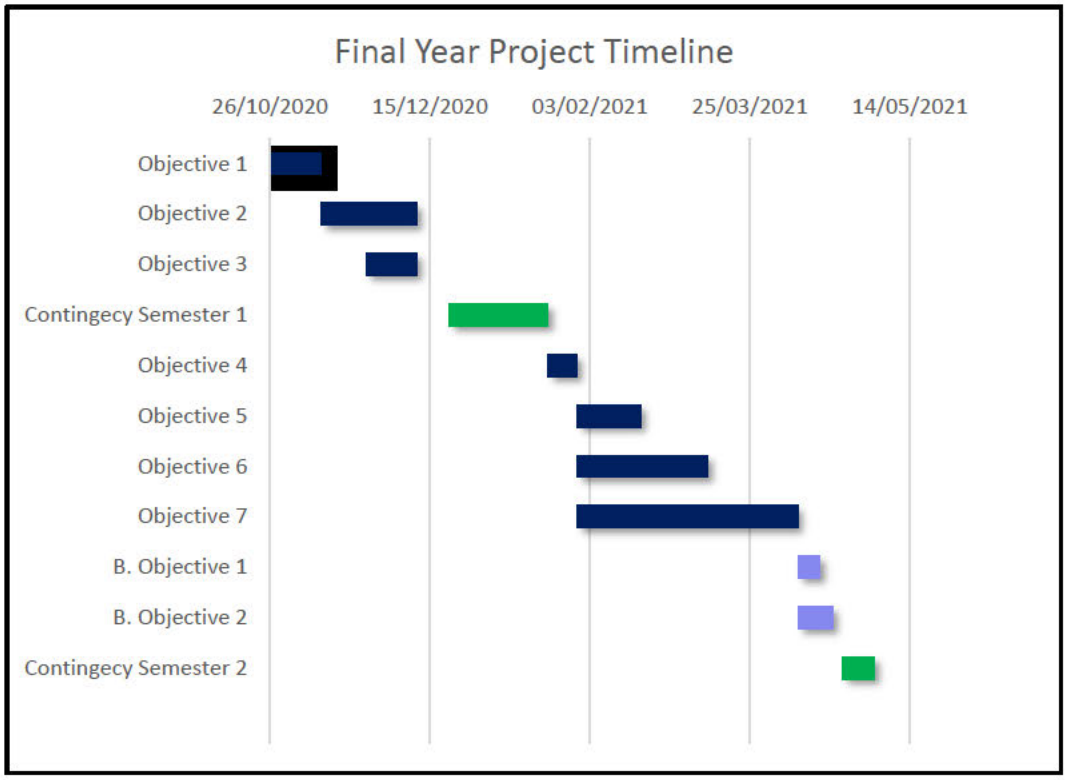

Figure 32: Timeline - Revised

There are five differences:

1. Increased contingency time at the end of Semester 1
2. There is less time to complete objective 4
3. Progress on Objective 5, 6, and 7 occurs at the same time until their respective completion dates
4. Progress on the bonus objectives occur at the same time until their respective completion dates
5. Reduced contingency time at the end of semester two

The reason for increasing the contingency time at the end of semester one was that I needed time for other module deadlines. While I considered this in the proposal, the amount of time allocated was insufficient to accommodate the time needed. Furthermore, through prototyping, I realized objective 4 did not require the amount of time allocated for it and redistributed the saved time towards the contingency time – reducing the time for objective 4 in the process.

Initially, I saw the project as a deadline-to-deadline affair, and the objectives naturally followed a waterfall approach. However, once the initial design for the product started, I realise that the previous timeline for objectives 5, 6, and 7 was not compatible with an agile approach. I discussed this with my supervisor and made this change to accommodate my agile approach. Although I did not attempt the bonus objectives, their change followed the same reasoning for objectives 5, 6, and 7.

I reduced the contingency time at the end of semester two as I thought I would complete all my practical work earlier to focus on writing; however, this proved to be untrue as I worked for a week and two days longer than expected.

8.3 Lessons Learnt

One of the personal lessons I learnt throughout the project was to open myself to scrutiny as much as possible. Although I felt embarrassed by my lack of cognisance in areas I have covered in the course previously, I needed to keep in mind that doing so and seeking opportunities can only help me irrespective of how I feel.

I also learnt that small details in experiments could snowball into significant problems if left unchecked. One such problem was towards the end of the project, where I realised that my experiments were not comparing models with the same test set. I did panic at this point; however, through supervision, I found myself overcoming this through experimentation.

This predicament leads to another lesson learnt – the need to keep code modular and polymorphic, as it was throughout my final product's implementation. While at the time, I gained benefits of quick and interchangeable code use throughout the development of the product, maintaining good coding practices allowed me to perform new experiments quickly so that it did not negatively impact my time.

Another helpful lesson learnt throughout the project was switching focus between various workloads. Throughout the project, I split my focus between the project and other modules. While switching between modules was nothing new, the incremental objective deadlines throughout the year were something I have not had to deal with before. While workload bursts for other modules existed, the project required a more sustained effort throughout, which I learned to do.

8.4 Reflection – Deliverables

For the first deliverable, I feel as though I was still determining the exact direction I wanted to take the project. Although I, generally, knew which area I wanted to go to, I feel as though I included irrelevant subjects in the first deliverable that I did not use at all in the second deliverable – for example, support vector machines.

I found that it is hard to estimate the impact of ambition increasing the project's scope to unrealistic levels against the grounding reality of what I can achieve in the time given to achieve it. Scope management is what I would say is the single most impacting factor between the two deliverables. In the first deliverable, scope management was not a pressing issue early on in the project. In the second deliverable, managing the project's scope became a more pressing issue due to the magnitude of the deliverable.

Overall, I am happy with both deliverables as they reflect the time and effort put in for the work done.

8.5 Summary

In summary, I feel the project has taught me a lot about how I manage an independent workload and provided me with insights into areas of improvement and has given me a better understanding of the deep learning process in a software environment.

9 Conclusions

This final section presents the dissertation's conclusions as a whole, the work completed, and any future work.

9.1 Conclusions – Dissertation

The dissertation set out to answer whether including ambiguous data in the training stages of a deep neural network (DNN) impacted the network's accuracy in predicting non-ambiguous data. The answer to this question is that including ambiguous data does not impact the accuracy of a DNN in a statistically significant manner. In the process of answering this question, the project produced software that embodied the principles of previously taught modules in its design and implementation. This software utilises a trained model without ambiguous data as this would not impact its accuracy – based on the conducted experiments.

9.2 Conclusions – Current Work, State-of-the-art, and Future Work

As mentioned, the current work focuses on the impact of ambiguous data on the model's accuracy. Current state-of-the-art research such as that of Zhang and LeCun (2015) concerning text understanding, Yadav and Vishwakarma (2020) concerning sentiment analysis in deep learning architectures, and Dang et al. (2020)'s comparative study on sentiment analysis with deep learning contains references to datasets that exclude ambiguous data. This use means that there is more to investigate than the limited scope covered in this project. Future work should explore the impact of including ambiguous data on each label within the classes in the test set to see if there is a statistically significant reason beyond accuracy alone to justify ambiguous data's removal. Future work may also increase the sample sizes during training and be domain-specific, e.g., electronics, books, or clothes.

In conclusion, ambiguous data is still an open area of investigation when not concerning the accuracy and further experimentation that explores the labels in more detail may yield exciting results.

References

Baktha, K., & Tripathy, B. K. (2017, 6-8 April 2017). *Investigation of recurrent neural networks in the field of sentiment analysis.* Paper presented at the 2017 International Conference on Communication and Signal Processing (ICCSP).

Bass, J. M. (2019). *Skills for Agile Software Engineering: An Introduction*. Electronic Book. University of Salford Manchester.

Bhadane, C., Dalal, H., & Doshi, H. (2015). Sentiment Analysis: Measuring Opinions. *Procedia Computer Science, 45*, 808-814. doi:https://doi.org/10.1016/j.procs.2015.03.159

Blitzer, J., Dredze, M., & Pereira, F. (2007). *Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification.* Paper presented at the Proceedings of the 45th annual meeting of the association of computational linguistics.

Chollet, F., others. (2015). Keras Retrieved from https://keras.io

Dang, N. C., Moreno-García, M. N., & De la Prieta, F. (2020). Sentiment Analysis Based on Deep Learning: A Comparative Study. *Electronics, 9*(3), 483. Retrieved from https://www.mdpi.com/2079-9292/9/3/483

Engler, T. H., Winter, P., & Schulz, M. (2015). Understanding online product ratings: A customer satisfaction model. *Journal of Retailing and Consumer Services, 27*, 113-120. doi:https://doi.org/10.1016/j.jretconser.2015.07.010

Guidotti, R., Monreale, A., Ruggieri, S., Turini, F., Giannotti, F., & Pedreschi, D. (2018). A Survey of Methods for Explaining Black Box Models. *ACM Comput. Surv., 51*(5), Article 93. doi:10.1145/3236009

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., . . . Oliphant, T. E. (2020). Array programming with NumPy. *Nature, 585*(7825), 357-362. doi:10.1038/s41586-020-2649-2

Hughes, P. (2000). Python and Tkinter Programming. *Linux J., 2000*(77es), 23–es.

Katić, T., & Milićević, N. (2018, 13-15 Sept. 2018). *Comparing Sentiment Analysis and Document Representation Methods of Amazon Reviews.* Paper presented at the 2018 IEEE 16th International Symposium on Intelligent Systems and Informatics (SISY).

Kaur, H., Mangat, V., & Nidhi. (2017, 10-11 Feb. 2017). *A survey of sentiment analysis techniques.* Paper presented at the 2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC).

Manning, C. D., Raghavan, P., & Schütze, H. (2008). *Introduction to Information Retrieval*: Cambridge University Press.

Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). *"Why Should I Trust You?": Explaining the Predictions of Any Classifier*. Paper presented at the Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, California, USA. https://doi-org.salford.idm.oclc.org/10.1145/2939672.2939778

Spyder. (2020). Retrieved from https://www.spyder-ide.org/

Uysal, A. K., & Murphey, Y. L. (2017, 21-23 Aug. 2017). *Sentiment Classification: Feature Selection Based Approaches Versus Deep Learning.* Paper presented at the 2017 IEEE International Conference on Computer and Information Technology (CIT).

von Helversen, B., Abramczuk, K., Kopeć, W., & Nielek, R. (2018). Influence of consumer reviews on online purchasing decisions in older and younger adults. *Decision Support Systems, 113*, 1-10. doi:https://doi.org/10.1016/j.dss.2018.05.006

Wahyudi, M., & Kristiyanti, D. A. (2016). Sentiment Analysis Of Smartphone Product Review Using Support Vector Machine Algorithm-Based Particle Swarm Optimization. *Journal of Theoretical & Applied Information Technology, 91*(1).

Willett, P. (2006). The Porter stemming algorithm: Then and now. *Program electronic library and information systems, 40*. doi:10.1108/00330330610681295

Yadav, A., & Vishwakarma, D. K. (2020). Sentiment analysis using deep learning architectures: a review. *Artificial Intelligence Review, 53*(6), 4335-4385. doi:10.1007/s10462-019-09794-5

Zhang, X., & LeCun, Y. (2015). Text understanding from scratch. *arXiv preprint arXiv:1502.01710*.

Appendix A

Links to External Cloud Documents

[redacted]

Appendix B

Raw Experiment Data (p1 and p2 are confidence intervals)

Model with No Ambiguous Data	Positive Ratings in Training	Negative Ratings in Training	Placeholder
	5, 4	2, 1	x
Experiment No.	Test Acc. (%)	p1 (%)	p2 (%)
1	78.3	76.28	80.31
2	79.02	77.03	81.01
3	78.1	76.08	80.12
4	78.97	76.98	80.96
5	78.62	76.62	80.62
6	78.87	76.88	80.86
7	78.7	76.7	80.7
8	77.82	75.79	79.85
9	78.37	76.36	80.38
10	79.42	77.45	81.39
Average:	78.62	76.62	80.62

Figure 33: Experiment Data - No Ambiguous Data

Model with Ambiguous Data as Positive	Positive Ratings in Training	Negative Ratings in Training	Placeholder
	5, 4, 3	2, 1	x
Experiment No.	Test Acc. (%)	p1 (%)	p2 (%)
1	76.22	74.14	78.3
2	76.37	74.3	78.45
3	76.1	74.02	78.18
4	77.75	75.72	79.78
5	76.47	74.4	78.54
6	77.05	74.99	79.1
7	75.67	73.58	77.77
8	76.4	74.33	78.47
9	77.17	75.12	79.22
10	75.4	73.3	77.5
Average:	76.46	74.39	78.53

Figure 34: Experiment Data - Ambiguous Data as Positive

Model with Ambiguous Data as Negative	*Positive Ratings in Training*	*Negative Ratings in Training*	*Placeholder*
	5, 4	3, 2, 1	x
	Test Acc. (%)	*p1 (%)*	*p2 (%)*
1	78.42	76.41	80.43
2	76.72	74.66	78.79
3	77	74.94	79.05
4	77	74.94	79.05
5	75.25	73.14	77.36
6	75.85	73.76	77.94
7	76.3	74.22	78.38
8	77.2	75.15	79.25
9	75.75	73.66	77.84
10	76.12	74.04	78.21
Average:	76.56	74.49	78.63

Figure 35: Experiment Data - Ambiguous Data as Negative