Detection of fake reviews using a transparent machine learning approach

Artem Butbaev

Bachelor of Science in Computer Science with Business with Honours
The University of Bath
April 2021

Detection of fake reviews using a transparent machine learning approach

Submitted by: Artem Butbaev

Abstract

Recently, the domain of fake review detection has received strong attention from academia, businesses and governments due to the wide-ranging, unfavourable effects of fake reviews on individual consumers and businesses.

In the past, the literature within this domain has mostly focused on overall model performance and feature engineering, rather than on the aspect of explainability. In this dissertation, a fake review detection model is implemented based on a transparent machine learning approach combined with an innovative capability of presenting its decision-making to the end user when classifying a review sample through an explainable interface. A real-world labelled dataset from the Yelp business review website of around 60,000 restaurant reviews is used for training and evaluating the model's performance. The explainable interface is evaluated based on criteria examined in the transparency literature.

The dissertation concludes with a critical discussion of the implemented product and its performance, identifies the project's useful contributions to the domain and identifies opportunities for further work.

Contents

List of Figures

List of Tables

Acknowledgements

Dr Rob Wortham for his consistent direction, guidance and support as my supervisor.

Dr Mohammad Golbabaee for providing feedback as my second marker to help improve the dissertation and sharpen its focus.

Professor Bing Liu for providing access to the Yelp dataset used in the development of the system.

Chapter 1

Introduction

1.1 Motivation

The domain of fake review detection has received strong attention from academia, businesses and governments in the last few years due to the key role of reviews in shaping consumer behaviour and affecting purchasing decisions (Crawford et al., 2015). Research has shown that online reviews are a crucial component that consumers consider when purchasing products and services online (Xiao, Wei and Dong, 2016) with the UK government estimating that over 75% of UK internet users consider them in their purchase decision making process (Competition and Markets Authority, 2020). Arguably, online reviews are very useful for consumers when buying goods online.

Online reviews are also beneficial to organisations for purposes such as opinion mining and opinion stream mining. With these techniques, the goal is to assess the popularity of products and identify customers' changing attitudes and perceptions based on online reviews and electronic word-of-mouth (eWOM) in general, such that future iterations can be improved (Zimmermann, Ntoutsi and Spiliopoulou, 2016).

However, due in part to the important role of eWOM in consumer decision making and the rising commercial value of online reviews, there has been a large increase in the number of fake reviews - also called shill reviews, deceptive reviews or opinion spam - on online platforms (Cardoso, Silva and Almeida, 2018).

The effect of reviews on consumer decision making depends on a number of features, such as the type of product or service, and the sentiment of the review. Examples of types of products include search goods, i.e., products which can be assessed well before a first-hand experience, e.g., supplement tablets, and experience goods, which are more challenging to assess before a direct experience, e.g., a stay at a hotel (Lee and Shin, 2014). The latter type of product has been found to be influenced more strongly by online reviews (Park and Lee, 2009). In terms of sentiment, the literature seems to agree that negative reviews generally have a stronger effect on consumer decision making than positive reviews, in part due to the negativity effect, where people find negative information particularly attention grabbing (Park and Lee, 2009). As a result, the effect of negative reviews can lead to an unfavourable impact on sales and brand perception for organisations, as was found by Li, Wu and Mai (2019) for reviews relating to tablet computers and their sellers. Floyd et al. (2014) also found that the sentiment of reviews has a greater impact on sales as compared to volume, i.e., the quantity of reviews available.

The greater number of fake reviews present on online platforms has the effect of skewing the product perceptions of consumers, undermining consumers' trust and resulting in inaccurate use of opinion mining to identify product feedback for organisations. Fake reviews on online platforms may also have legal implications, as they are arguably a form of consumer misinformation distorting fair market competition (Hunt, 2015). An anecdotal and well-known example of legal action being taken in the UK for the creation of fake reviews was Orlando Figes, a British historian, who criticised competitors' books on Amazon to undermine their ideas and make his books look more favourable (Orlando Figes to pay fake Amazon review damages, 2010).

As part of their work with digital markets, the UK Government was undertaking, at time of writing, an assignment within the area of online consumer protection. In part, they were investigating major websites for the level of protection they offer for consumers from fake reviews (Competition and Markets Authority, 2020). This work highlights that the domain of fake reviews is not only of interest to individuals and businesses, but also to governments due to their legal implications.

1.2 Project Aims

Clearly, fake review detection is a problem worthy of addressing being a present problem and concern in the real world. For this project, the broad goal was to develop a system capable of detecting fake reviews to help in addressing some of the challenges presented in the previous section. Upon reviewing the literature, a gap was found with regards to transparency, with none of the existing literature presenting a model capable of classifying reviews, while at the same time being able to present its decision making to the end user. Hence, the objective and aspiration of the project was to explore the previous work done and develop a fake review detection system based on this work, while adding a layer of transparency and showcasing its usefulness for research and commercial applications.

A formal outline of the project aims follows below:

1. Critically review the relevant literature and previous work to establish a view of the state of the art in fake review detection and transparency research
2. Train, test and evaluate a classifier on a real-world fake review dataset with comparable performance to the models presented in the literature
3. Develop a transparent front end layer in the form of an explainable interface which effectively explains the decision making of the model to a naive user
4. Critically evaluate and discuss the performance of the developed system in terms of model performance and the level of transparency of the explainable interface
5. Present recommendations for future work in the domain of fake review detection based on the implemented product

1.3 Dissertation Outline

Chapter 1 Introduction

Outlines the motivations for undertaking this project, the aims of the project and the structure of the dissertation.

Chapter 2 Literature and Technology Survey

Critically reviews the literature on the topic areas of fake review detection and transparency, setting the foundation for the project in the wider problem domain.

Chapter 3 Requirements

Outlines the requirements gathering process, explains key requirement decisions and presents a formal requirements specification enabling the implementation of the machine learning model and explainable interface to satisfy the project aims.

Chapter 4 Implementation

Describes how the requirements were implemented in practice to achieve the project aims. Explains in detail the implementation of the two key parts of the system - the fake review detection model and the explainable interface. The chapter also presents an overview of the overall system and the technology stack used.

Chapter 5 Results and Analysis

Outlines and describes the results of the fake review detection model based on the relevant performance metrics. Presents the performance of the explainable interface through comparison to state of the art transparency criteria and best practices.

Chapter 6 Discussion

Critically examines the system implemented, outlines the contributions and insights of the project, and highlights limitations of the methodology used. The system, research and commercial contributions of the dissertation are also outlined.

Chapter 7 Conclusion and Future Work

Provides a summary of achievements in line with project aims and requirements, with future work suggested.

Chapter 2

Literature and Technology Survey

2.1 Aim

This chapter critically inspects and reviews the literature on the topic areas of fake review detection and transparency, setting the foundation for the project by identifying how it may contribute to the wider research in the problem domain, as well as helping to shape the technical requirements for the practical aspects of the project. The survey reviews the machine learning approaches that have been used previously to identify fake reviews, evaluates previous work and highlights current challenges in the problem domain. Likewise, the survey also reviews the state of the art in the problem domain of transparency and examines how its principles can be applied to the task of creating a transparent fake review detection model. The main goal of the chapter is to build an awareness of what has been tried and tested to help approach the problem of fake review detection from a different angle in our solution to contribute to the existing literature with original insights.

2.2 Fake Review Detection Approaches

This section critically explores the machine learning approaches that have been used in the literature over the last decade to create a system capable of detecting fake reviews. It also considers how such systems have been evaluated in the past in terms of performance. Based on their systematic survey of the fake review detection domain, Ren and Ji (2019) break down the task of fake review detection into two sub-tasks: the detection of spam (the reviews) and the detection of spammers (the subjects who write the reviews). This project focused on the former, and as such the outlined approaches relate predominantly to the detection of fake reviews based on the content of the reviews and their metadata, rather than detecting the individual or collective parties who craft fake reviews. Non-machine learning approaches are not considered in this Literature and Technology Survey as they are "rare, immature, and limited in use" (Zhang et al., 2016) within the domain of fake review detection.

2.2.1 Supervised Learning

According to Zhang et al. (2016), the most frequently used type of machine learning within the domain of fake review detection is supervised learning, with the most commonly used algorithms being support vector machines (SVM), Naive Bayes (NB), decision trees and random forest. In a supervised learning model, the detection of fake reviews is seen as a binary classification problem, where the task is to classify each review into one of two sets: genuine (authentic) or fake (deceptive). A supervised learning approach requires a dataset for training the machine learning model with examples describing both the inputs and corresponding output labels. Once such a model has been trained with example input-output pairs, it can make predictions based on the characteristics of unseen data. As will be seen in Section 2.3, a major challenge within the domain for applying a supervised learning approach is the scarcity of labelled datasets, one reason being that human readers are very poor at detecting fake samples from a set of reviews (Ott et al., 2011).

Critical to the process of supervised learning is feature engineering, where a selection of input features are crafted and used to train the model effectively. For the detection of fake reviews, these features may be both review-based, i.e., related to the review content, or metadata-based, such as the historical behaviour of the reviewer who posted the review. According to Zhang et al. (2016), review-based features have been the most commonly used features used in creating fake review detection models.

2.2.2 Unsupervised Learning

Due to the challenges in procuring accurate real-world labelled datasets within the domain of fake review detection (discussed in more detail later in Section 2.3), machine learning methods not relying on large and accurate datasets are currently of high interest to researchers within the domain of fake review detection. These methods include unsupervised and semi-supervised learning techniques. However, they do not currently achieve the same performance levels as supervised methods albeit research into these methods is limited and findings are inconclusive (Crawford et al., 2015), suggesting that there may be potential for better performance using such alternative methods.

An example of a proposed unsupervised model for detecting fake reviews was created by Mukherjee et al. (2013a), called the Author Spamicity Model (ASM). The principle with an unsupervised model such as this, is that it looks for patterns in the data with no pre-existing labels needed. In the given context of fake review detection, the authors used various behavioural footprints of reviewers to arrive at two clusters: those of spammers and those of genuine reviewers, where the model inference process resulted in learning the population distribution of these clusters. The results from this model were worse compared to supervised learning models. The authors argue that a key reason for this is the noise produced due to the clustering - the clusters identify spammers and non-spammers, but each of these groups of reviewers may produce both authentic and fake reviews.

Another example of a proposed unsupervised model is presented by Kumar et al. (2019), which is built on the basis of distributions describing different aspects of reviewing behavior. Although it outperforms previous unsupervised models within the domain, its performance is not compared to supervised models. As is clear, both of these approaches focus on the reviewer aspect of the fake review detection problem, rather than the review aspect as the majority of supervised learning implementations have done.

2.2.3 Natural Language Processing

Natural Language Processing (NLP) is a sub-field of artificial intelligence, spanning the disciplines of linguistics and computer science, concerning itself with the processing and understanding of text data, often utilising statistical analysis and machine learning techniques. (*Natural Language Processing (NLP) – University of Copenhagen*, 2020). As argued by Hovy and Spruit (2016), NLP can be seen as a "dual-use problem" in the domain of fake review detection, meaning NLP techniques can be used for both good and bad - to detect fake reviews and generate them in the first place. For the purposes of this project, the main focus will be on the former application of NLP.

For this project where the main goal was to build a model for detecting fake reviews, NLP techniques were certainly required. This is because the a key part of the data used for the fake review detection model will naturally be unstructured text data, which will need to be processed to be used as features in the machine learning model. For example, using NLP techniques, one may identify verbal features in reviews such as the number of nouns in a review, the sentiment of individual reviews or even the meaning of words in specific contexts (semantics).

In terms of NLP terminology, of note is a *corpus*, which is a collection of individual texts, also called documents. In the domain of fake reviews, the corpus will be the total collection of reviews. Second, a *document* is a fixed unit of text processing, which is part of a corpus. In the context of reviews, the intuitive granularity is for each review to be a separate document (Méndez et al., 2005).

Tokenisation

Focusing on NLP processing methods, tokenisation is one commonly used approach used within the field of spam and fake review detection. Also called lexical analysis, the approach divides each document into strings of characters called tokens. This is so the algorithm sees the document as individual tokens rather than a stream of characters. A common type of tokenisation is word tokenisation, where sentences are split into individual words. Within spam filtering systems, tokens are usually considered as strings of characters delimited by punctuation symbols or blanks (Manning, Raghavan and Schütze, 2009). Although punctuation and special characters are usually removed before tokenisation, Méndez et al. (2005) suggest punctuation may be important to include in features such as n-grams. Tokenisation is arguably one of the most essential steps in processing review data prior to feature extraction, as it is necessary for almost all text-based features.

Stopwords

Another method typically used in NLP is the use of a stopword list. This is a list of words that are deemed non-informative and thus removed during pre-processing of the corpus, e.g., high-frequency words that are common to all texts, such as "and". Using a stopword list greatly reduces the computational cost of pre-processing (Méndez et al., 2005). For their experiments on fake review detection models, Cardoso, Silva and Almeida (2018) did not remove stopwords as is common in the spam domain (Méndez et al., 2005), as certain stopwords are important features found in fake reviews and spam in general. Cardoso, Silva and Almeida (2018) also did not make use of stemming or lemmatisation and kept words in their original form as these techniques may remove important features important

True Review Class	Predicted as Fake	Predicted as Genuine
Fake	**tp**	**fn**
Genuine	**fp**	**tn**

Table 2.1: Confusion matrix for the problem of fake review binary classification

for detecting fake reviews.

Stemming and Lemmatisation

Two other methods used are stemming and lemmatisation. Stemming is a technique used to reduce words to their "stem" or common base form, i.e., their grammatical roots. For example, "running" and "runner" would both be stemmed to "run". Lemmatisation is similar to stemming but with a crucial difference in keeping the semantic meaning of word. Whereas stemming is a "crude" approach where endings of words are cut off, lemmatisation uses a dictionary or vocabulary to return the base or dictionary form of a word, which is known as the lemma (Manning, Raghavan and Schütze, 2009). For example, a stemming algorithm may simplify "ponies" to "poni", whereas a lemmatisation algorithm would simplify the same word to the correct "pony". Different algorithms may be used for the process of stemming and lemmatisation based on requirements.

2.2.4 Evaluation of Model Performance

The approach to evaluating the performance of machine learning models varies widely, depending on a number of factors such as the problem domain, the particular properties of the dataset and the classes of importance. It is generally agreed within the machine learning community that there is no single best measure with each having its strengths and weaknesses. Using a combination of different evaluation metrics is preferred as it gives a more balanced view of a given algorithm's performance (Sokolova, Japkowicz and Szpakowicz, 2006). In the literature investigated for the domain of fake review detection, there are a number of different evaluation metrics used, which will be described shortly.

Confusion Matrix

For evaluating binary classifiers, such as in the case of fake review detection, evaluation metrics are constructed based on the values from a confusion matrix (Sokolova, Japkowicz and Szpakowicz, 2006). This matrix records, for each class (fake and genuine reviews), how many samples were correctly and incorrectly classified.

This results in four possible values:

- True positives (tp) - the fake reviews correctly identified as fake
- False negatives (fn) - the fake reviews incorrectly identified as genuine
- True negatives (tn) - the genuine reviews correctly identified as genuine
- False positives (fp) - the genuine reviews incorrectly identified as fake

An illustration of this can be seen in Table 2.1.

Evaluation Metrics

The most commonly used metric of accuracy within machine learning (Sokolova, Japkowicz and Szpakowicz, 2006) is arguably problematic and misleading for use in the domain of fake review detection, in large part due to the data imbalance of genuine and fake reviews, as will be illustrated in the example of the naive classifier. Due to this data imbalance and the difference in importance of the two classes, we want to distinguish between the number of correct labels identified of the two types of reviews.

The most commonly used metrics in the domain of fake review detection can be seen in Equations (2.1) - (2.5) below, each expressed in terms of the possible values from the confusion matrix in Table 2.1. Each measure ranges from 0% to 100%. These have been used in almost all works surveyed during this literature review. In particular, the four metrics excluding BA are used most often for the evaluation of fake review detection classifiers (Zhang et al., 2016; Ren and Ji, 2019; Ott et al., 2011; Mukherjee et al., 2013c).

The balanced accuracy (BA) has been included in this list to better illustrate the example of the naive classifier, but also because it helps illustrate the model's average performance of detecting both fake and genuine reviews, being the mean of the True Positive Rate (also called Recall (R)) and the True Negative Rate.

$$A = \frac{tp + tn}{tp + tn + fp + fn} \tag{2.1}$$

$$BA = \frac{\frac{tp}{tp+fn} + \frac{tn}{tn+fp}}{2} \tag{2.2}$$

$$R = \frac{tp}{tp + fn} \tag{2.3}$$

$$P = \frac{tp}{tp + fp} \tag{2.4}$$

$$F = \frac{2PR}{P + R} \tag{2.5}$$

Performance of a Naive Classifier

To further illustrate the need for using a combination of metrics rather than simply using accuracy to evaluate the performance of a fake review detection classifier, we may consider the example of a theoretical naive classifier which predicts the majority class label. Let us assume that the probability of a given review being fake is 10%. In such a case the classifier will naively predict each review as genuine leading to an accuracy of 90%. However, the number of fake reviews correctly identified (true positives) will be 0. In fact, the balanced accuracy (BA) - another kind of evaluation metric, based on Equation (2.2) below will be 45% for this example.

As is clear with this simplified example, evaluating a model tested on a dataset with an imbalanced label distribution using the accuracy evaluation metric certainly does not tell the full story of the classifier's performance. The impact of the imbalanced class distribution on a given model's performance is further highlighted later in the Results chapter, where the evaluation metrics are compared for fake review detection models trained on balanced and imbalanced sets of fake and genuine reviews, with the former achieving significantly better performance at detecting fake reviews. As the focus of this project is on developing a fake review detection model, we are more interested in detecting the minority class rather than the majority class, and as such addressing the data imbalance problem is crucial at both the development and evaluation stages.

Cross-validation

For getting an estimate of how a machine learning model will perform in practice, k-fold cross-validation (CV) is commonly used in the fake review detection literature, typically with either 5 or 10 folds (Crawford et al., 2015; Zhang et al., 2016).

The process of k-fold CV is visualised in Figure 2.1. In this example, the number of folds used is 5 (k = 5). Firstly, the dataset is split randomly into k number of equally sized sub-samples. Then, effectively 5 rounds of cross-validation take place, with the performance of the model being estimated as the average of the 5 rounds. In each round of CV, k–1 folds of data are used for training the model, with the last fold being left for testing the model. The fold that is used for testing has not been seen by the model beforehand, so cross-validation simulates the situation of the model classifying unseen data. As can be seen in the figure, for k-fold CV the testing sets for each round of CV are mutually exclusive.

Stratified k-fold cross-validation is similar to k-fold CV, but has the additional requirement that all folds of data have approximately the same proportions of class labels. Stratified k-cross validation is recommended for evaluating model performance on datasets of imbalanced class distributions, such as the case of fake review detection (Cardoso, Silva and Almeida, 2018).

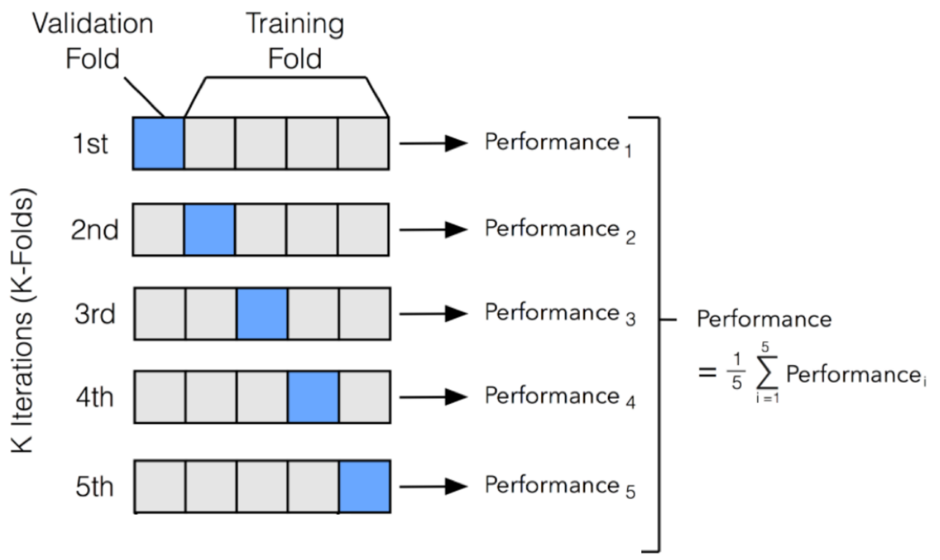


Figure 2.1: Diagram of k-fold cross-validation for 5 folds, from Unknown Author (2021).

2.3 Fake Review Data

This section critically explores some of the datasets that have previously been used in the literature for the development of fake review detection models. Unfortunately, there is a scarcity of labelled, open-source fake review datasets available; it is widely agreed within the domain that one of the main obstacles in effectively building models for detecting fake reviews based on supervised and semi-supervised approaches is the lack of publicly available labelled datasets (Zhang et al., 2016; Crawford et al., 2015; Mukherjee et al., 2013a).

One reason for the lack of labelled datasets is due to the fact that deceptive reviews are written to sound as authentic as possible, meaning they are often not identifiable by a human reader (Ott et al., 2011). This means that the manual labelling of authentic and fake reviews is a difficult and arguably a highly error-prone task. Arguably, with ever more advancements in AI and natural language processing in particular, it will only become more challenging in the future to manually distinguish fake from genuine reviews, at least on the basis of the text itself. To illustrate this challenge, two reviews from each class are shown below, sourced from a real-life dataset discussed later in this section. Arguably, they both look authentic at first glance.

> *'Imagination is rare, hence the line but imagination is worth waiting for. I had the game of the week (this time Ostrich with cherries and cheddar) which was mind blowing. The veggie dog while nothing as creative as the others tasted just like a real hot dog, which I suppose is quite creative if you've ever tasted one of those horrible Smart Dogs from Whole Foods. Although the line was long, it seemed directly matched up with tables making it easy to get one right after you ordered. Duck fat fries: put this on Randolph and we'd have an Iron Chef Doug.'* (**Example of a genuine review from the Yelp[1] restaurant dataset)**

> *'Tru was the best combination of excellent food and incredible service. We went there to celebrate a birthday, and the staff made it a memorable and special event. In addition to the extreme attention to detail, the staff was welcoming and friendly. While expensive, I would definitely recommend Tru because I felt the food and service were superior to other fine dining establishments. I enjoyed the experience much more than Everest, and maybe even Alinea.'* (**Example of a fake review from the Yelp restaurant dataset)**

2.3.1 Pseudo-Fake Reviews

An alternative approach that some researchers such as Ott, Cardie and Hancock (2013) have taken to overcome the challenge of lack of labelled datasets is hiring online workers to write fake and authentic reviews. These have been referred to as "pseudo-fake reviews" by other researchers in the field (Mukherjee et al., 2013c; Zhang et al., 2016). The platform that was used by Ott, Cardie and Hancock (2013) for hiring workers was Amazon Mechanical Turk (AMT)[2] - a crowd-sourcing marketplace that allows individuals, researchers or businesses to outsource tasks and jobs that can be completed virtually. AMT

[1]https://www.yelp.com/
[2]https://www.mturk.com/

is often used for outsourcing manual labelling activities required for machine learning, as well as other repetitive tasks.

According to Mukherjee et al. (2013c), all previous supervised learning methods within the area of fake review detection used a similar approach to Ott, Cardie and Hancock (2013), using pseudo-fake reviews in creating fake review detection models. However, this approach has been criticised by some researchers, with Zhang et al. (2016) arguing that such reviews are ultimately different in nature to reviews in the real world. In particular, they suggest that the online workers who craft the reviews have a different psychological state to real online spammers and that real-world fake reviews are likely to look more authentic, being more challenging for a model to detect. Arguably, one reason for this is that real spammers have more experience with crafting fake reviews and have a better idea of how to make fake reviews look more authentic, whereas hired AMT workers are likely to be doing a variety of tasks on the platform, and it is not known what their experience was in writing fake reviews.

In effect, Zhang et al. (2016) argue that fake review detection models using pseudo-fake reviews for the training dataset result in artificially high accuracies which are likely to perform less well on real fake reviews. For example, Ott et al. (2011) created a classifier with stated 90% accuracy using a dataset prepared using AMT, yet this classifier only reached detection accuracies near chance when tested on real-world fake reviews (Mukherjee et al., 2013c). This argument is also supported by the findings of Cardoso, Silva and Almeida (2018) where the researchers found that classifiers performed better on artificial reviews compared to real-world fake reviews.

2.3.2 Real-World Fake Reviews

Being against so-called pseudo-fake reviews, Zhang et al. (2016) instead assume the approach of using real-world fake reviews filtered by a commercial website - Yelp.com. The dataset used in the paper contains around 67,000 labelled reviews from over 200 hotels and restaurants where roughly 13% of reviews are filtered, i.e., fake or deceptive. This dataset has also been used by other researchers including Mukherjee et al. (2013c) and Cardoso, Silva and Almeida (2018). We acquired this dataset for use in this project by request from Bing Liu, Distinguished Professor in the Department of Computer Science at the University of Illinois at Chicago and author of numerous papers in the domain of fake reviews detection, including Mukherjee et al. (2013c).

A key challenge with the Yelp dataset and other website-filtered datasets, as mentioned by Kumar et al. (2019), is that the data is sourced from Yelp's proprietary, confidential algorithm meaning it is unknown what verbal or non-verbal features have been used by the website to filter the reviews. However, the Yelp algorithm's performance, including its accuracy has been supported by a number of studies such as Mukherjee et al. (2013a), where the researchers found its filtering to be reasonable and reliable, with the filtered reviews being correlated with abnormal spamming behaviours (Zhang et al., 2016). Another example of a similar real-world fake review dataset which has been used by researchers within the domain is sourced from Dianping - a Chinese online shopping platform (Ren and Ji, 2019). Similar to the Yelp dataset, this was also gathered as output from the confidential algorithm of the commercial website.

Advantages and Disadvantages of the Yelp Dataset

One of the advantages of the Yelp dataset is that the characteristics of the reviews within it are likely to be similar to other genuine and fake reviews on the Web, with Mukherjee et al. (2013b) arguing that the Yelp dataset is the closest ground-truth labelled dataset available in the real-life setting. Another related advantage is the natural class distribution of the reviews in the Yelp dataset, compared to pseudo-fake datasets where the researchers dictate an arbitrary class distribution, such as the pseudo-fake reviews crafted by Ott et al. (2011), where there was a 50/50 split of genuine and fake reviews. Having a dataset with a natural class distribution of genuine and fake reviews means that an evaluation of the model would lead to a more realistic preview of how the model would perform in a real-world setting (Mukherjee et al., 2013b). Furthermore, the Yelp dataset is of relatively large size with around 67,000 labelled reviews. This number of reviews would be very time-consuming and expensive to craft using crowd-sourcing. Finally, the Yelp dataset contains review metadata, such as the number of up-votes from other reviewers - this makes it possible to extract more features and thus arguably have a better description of the machine learning problem compared to only using the textual information within the review itself.

Although the dataset clearly has its advantages, there are also some disadvantages to consider. Firstly, the Yelp dataset only focuses on reviews within the context of restaurants and hotels. Arguably, this makes the review-centric features (the features extracted from the text within the reviews) less likely to generalise well to other domains of reviews, such as consumer goods. Another drawback is that we assume the labels in the dataset are correct, i.e., that the proprietary review filtering process is accurate. Although the literature surveyed suggests that Yelp's confidential algorithm and its filtering is reliable, the labels are likely not 100% accurate. Arguably, another drawback is that these reviews were posted in the time-period 2004-2012, since which spammers could have become more sophisticated in their approach and thus the inherent characteristics of fake reviews may have changed, meaning that a model trained on this data may not be as good at detecting contemporary fake reviews. Furthermore, the reviews are sourced from a limited geographical region in Chicago, US, where the researchers who sourced the dataset are based. Therefore, a model trained on this dataset may not generalise well to other English-speaking countries, like the UK.

Although other real-world datasets are publicly available within the domain as outlined by Ren and Ji (2019), such as reviews from Amazon, these have been labelled using non-machine learning approaches which are not proven to be reliable, for example rule-based methods, where researchers have determined a set of rules for automatically annotating fake reviews, e.g., repetitive reviews by a single user may be a rule that would lead to a review being flagged as fake (Ren and Ji, 2019).

2.3.3 Data Imbalance

A further consideration which was briefly mentioned in the previous subsection regarding labelled datasets in the domain of fake review detection is the natural imbalance of fake and authentic reviews in a real-world dataset(Al Najada and Zhu, 2014), which is expected and a challenge present in this problem domain. For example, in the aforementioned Yelp review dataset which was acquired for this project, fake reviews make up only 13% of the total reviews.

Training a classifier with such an imbalanced representation of classes will lead to the data imbalance problem and the performance of the model suffering as seen in Section 2.2.4 on Evaluation of Model Performance. In the case of the majority of samples being labelled as genuine reviews, this may result in the model classifying all fake reviews (the minority class) as authentic reviews (Zhang et al., 2016). This is because the model will see the classification of each review as authentic as the most probable label given the imbalance of the data. As seen in Section 2.2.4, this may lead to the model having a good overall accuracy, but this will only be because the model is mainly basing its decision on the natural class imbalance in the dataset. Therefore, training the model on the natural class distribution of reviews would lead to biased predictions and negatively impact the model's ability to detect fake reviews which is the main focus of this project.

There are are a number of techniques used in the literature to overcome this challenge. One approach is to artificially balance the class distribution of reviews using data oversampling (increasing the number of data points with the minority label) and/or undersampling (reducing the number of data points with the majority label) (Mukherjee et al., 2013c; Zhang et al., 2016). Both oversampling and undersampling have their disadvantages. Whereas oversampling may cause some overfitting due to more identical or similar input-output example pairs depending on the method used, undersampling causes a loss of information about a class as we are leaving out potentially informative examples. More advanced resampling techniques include oversampling whilst making small tweaks to features or synthesising new data points, and undersampling of the majority class after clustering analysis, the goal being to preserve as much information about the class as possible [3]. Other techniques to overcome the challenge of data imbalance include modifying the cost of misclassification in the model hyper-parameters and using a cost-sensitive machine learning approach for training the classifier.

[3]https://www.kaggle.com/rafjaa/resampling-strategies-for-imbalanced-datasets

2.4 Feature Extraction

This section explores feature extraction where new features are constructed from a given dataset as it is applied to the fake review detection domain. The types of features previously used in the literature are summarised including review-centric (verbal) and reviewer-centric (non-verbal) features, which have been used to train supervised fake review detection models.

Feature extraction or feature engineering is the process of constructing or devising features from a given dataset to be used for training a machine learning algorithm (Crawford et al., 2015). In the domain of fake review detection, we are looking to extract discriminating features that can help us to effectively distinguish deceptive from genuine reviews by better describing the discriminating characteristics of genuine and fake reviews. Crawford et al. (2015) break down the extraction of features into two sub-tasks: review-centric (verbal) and reviewer-centric (non-verbal) features.

Hence, for the problem of fake review detection, individual predictive features are elements extracted either from the text of the reviews, i.e., verbal features, or as metadata from each review in the form of non-verbal features. This broad categorisation of features used for building machine learning models for detecting fake reviews is also supported by Mukherjee et al. (2013c) and Zhang et al. (2016). Zhang et al. (2016) flag the importance of focusing on individual predictive features when building and evaluating machine learning models for detecting fake reviews, arguing that generalised approaches where the overall performance of models is assessed lack resulting insight and guidance on how to improve such models in the future.

Although it has been observed that models using a combination of verbal and non-verbal features achieve the best performance in identifying fake reviews, it remains unclear what the best set of features is and which features are the most beneficial (Crawford et al., 2015). Zhang et al. (2016) argue that non-verbal features are more important than verbal features in creating models with better performance and hypothesise that as online spammers become more sophisticated in crafting fake reviews through experience, models using only verbal features will become less effective over time. Although this may be true, arguably there will often be some linguistic footprint left behind by a spammer, which is detectable by an algorithm. Researchers believe that although fake and genuine reviews are not easily distinguishable, subtle linguistic cues set them apart (Zhang et al., 2016). Arguably, non-verbal features may be susceptible to "cheating", for example if a spammer creates a new account to post a review with no previous posting history. This example strengthens the idea that a combination of verbal and non-verbal features is needed to detect fake reviews effectively.

Outlined below are some of the most commonly used verbal and non-verbal features in the literature for building fake review detection models.

2.4.1 Review-centric (Verbal) Features

1. **Review length** - the total number of words in a given review (Zhang et al., 2016). As argued by Ren and Ji (2019), this feature may be important, as the average review length of reliable reviewers is typically longer than that of spammers.
2. **Bag of words (n-grams)** - n-grams may used to create features based on sequences

of words from a review text (Crawford et al., 2015). For example, using trigrams (n = 3), sets of three contiguous words from the review text are used to create a vector model. The values within the vector may be used as features values.

3. **Term Frequency** - similar to n-grams where n = 1, term frequency features count the number of occurrences of specific words, rather than simply having a flag to indicate if it appears in a given review or not (Crawford et al., 2015).
4. **Sentiment** - the ratio of sentiment indicators to the review length (Zhang et al., 2016). As suggested by Ott, Cardie and Hancock (2013), deceptive reviews may exaggerate the sentiment of their reviews (how positive or negative they are), compared to similar truthful reviews.
5. **Parts of speech (PoS)** - both the distribution of PoS tags and the number of nouns, verbs and adjectives can be used as features (Zhang et al., 2016). Ott et al. (2011) suggest that the frequency distribution of PoS tags in a review is a good feature due to it being indicative of the genre of a text, where fake reviews are assumed to be of an imaginative genre, where truthful reviews are of an informative genre.
6. **Linguistic Inquiry and Word Count (LIWC)** - used in numerous papers, LIWC is a software that can be used to derive features as it groups words in a given text into psychologically meaningful dimensions (Ott et al., 2011), such as social words (e.g., "mate") and self-references (e.g., "I", "my") (Crawford et al., 2015).
7. **Semantic** - in contrast to bag of words (n-grams), semantic features consider the underlying meaning of words in a specific context (Ren and Ji, 2019), rather than simply the occurrence of specific words in a review. For example, one particular word may have different meanings in different contexts.
8. **Lexical diversity** - the ratio of unique words (Zhang et al., 2016). Arguably, the intuition behind such a feature is that a genuine review will typically use a wider range of unique words specific to their personal experience, whereas a fake review may use words common to other reviews.

2.4.2 Reviewer-centric (Non-Verbal) Features

1. **Reviewer's past posting history** - Crawford et al. (2015) highlight that spammers often write more reviews on a given day than a genuine user. According to the researchers, 90% of genuine users never post more than one review within 24 hours. Specific features include total review count, maximum number of reviews posted over one day and average number of reviews posted on a given day (Zhang et al., 2016).
2. **Membership length** - the amount of time that a given reviewer has been a member of the online platform (Zhang et al., 2016). Arguably, spammers may have to create multiple accounts to minimise the number of reviews posted on each account. Therefore, an account with a short membership length and multiple reviews posted may be an indicator of a spammer.
3. **Reviewer deviation** - as noted by Crawford et al. (2015) and Ren and Ji (2019), it has been observed that spammers' ratings tend to deviate from the average user

rating more often than genuine reviewers'.

4. **Review votes** - Zhang et al. (2016) highlight the following non-verbal attributes available on the Yelp business review site: useful, funny and cool votes. Users of the platform can use these to react to reviews. Similar reactions can also be found on other websites, such as Amazon.

2.4.3 Principal Component Analysis (PCA)

Principal component analysis (PCA) is a technique of multivariate analysis that may be used to reduce the dimensionality of a given dataset or feature space by removing redundant or unimportant features to reduce the complexity of the machine learning model (Sedighi, Ebrahimpour-Komleh and Bagheri, 2017). This is especially useful when there are many features extracted from a dataset, as is likely the case with a model for detecting fake reviews with many possible features as highlighted in the two preceding subsections.

PCA works by preserving the features in the data which have the largest data variations, called the principal components and removing unnecessary features (Sedighi, Ebrahimpour-Komleh and Bagheri, 2017). This reduction of the dimensionality of the dataset is achieved by transforming to an updated set of features which are uncorrelated such as to preserve the variation present in the original features (Jolliffe, 2002).

By separating the discriminant features from the original feature space this should arguably help with reducing model overfitting, as intuitively the model would consider fewer attributes of a given review, making the decision making process less complex. Also, as one of the main goals of the project is to develop a transparent fake review detection model, PCA could potentially help with creating a simplified and more meaningful explanation of the model's internal decision making to the end user, as only the most important features could be considered in its decision making.

2.5 Transparency and Explainability

This section explores the state of the art in transparency and explainability as they relate to autonomous intelligent systems, and in particular fake review detection systems. Being one of the project aims to develop an explainable interface to help the user understand the decision-making of the fake review detection model, the literature was consulted to understand important considerations and best practices within this area along with approaches to assessing the level of transparency of a given system, applying these ideas to the problem of fake review detection.

2.5.1 Transparency

Transparency is generally recognised within the AI ethics community to be a must-have requirement for autonomous and intelligent systems (AIS), such as robots, AI-systems and algorithmic decision systems (Winfield et al., 2021). It is also one of the principles in EPSRC's Principles of Robotics, which outline the critical characteristics of an autonomous and/or robotic system (Wortham, 2020). In the last few years, transparency has also become a popular theme within the wider technology community and society in general. For example, Satya Nadella, CEO of Microsoft, outlined in an article for Slate Magazine a number of objectives and design considerations that the industry and society as a whole should have for AI, one of which is transparent AI: "We should be aware of how the technology works and what its rules are. We want not just intelligent machines but intelligible machines... People should have an understanding of how the technology sees and analyzes the world" (Nadella, 2016).

Definition of transparency

Despite its importance, transparency is a broad concept which has no single clear definition. The non-specificity of the concept is highlighted by the fact that there is no current standard on transparency that can be tested for in artificially intelligent systems (Winfield et al., 2021). Arguably the most common trait found in all definitions of transparency during the literature review was the extent of a given user's understanding of an intelligent system, which is dependent on a number of factors, such as "the extent to which a system discloses criteria of its functioning" (Spagnolli et al. (2017) as cited in Winfield et al. (2021)). This reflects the idea that transparency is not a property that a system either possesses or not, but rather that the transparency of a system lies on a continuous scale.

The concept of trust is commonly mentioned alongside transparency in the literature. For machine learning models in particular, it has been argued that trust is "fundamental if one plans to take action based on a prediction" (Ribeiro, Singh and Guestrin, 2016). In the case of a fake review detection system, the end user would arguably be expected to take an action based on the output of the system, such as ignoring the information in a review classified as fake or choosing to purchase a product because of reading reviews classified as genuine. Hence, the user would need to trust the system to consider its output in their decision-making process.

Two other concepts related to transparency are explainable AI (XAI) and interpretability which are concerned with making the output of AI systems understandable to a user instead of having a black box with only inputs and outputs. These two concepts are explored in more detail in the next sub-section of this chapter.

The need for transparency

The need for transparency arises for several reasons. Arguably the main goal of a transparent system is that it allows the stakeholder (researcher, consumer, organisation, etc.) to understand how it functions and generates specific output. This understanding can then help the human user manage situations when the AIS goes wrong, when accountability is needed or when the user does not know the underlying problem well. It also allows for building trust in stakeholders and users with regards to the system (Bond et al., 2019).

Let us consider these needs for the case of a fake review detection system. As with any AIS, the system would not be expected to be 100% correct at all times (Winfield et al., 2021). By having a transparent system, the relevant stakeholders would be able to better analyse when the system classifies a review incorrectly. This information could potentially be used to improve the system as a result. In contrast, with a black box system, the user can only make a guess as to why a review has been erroneously classified. Accountability is arguably particularly attractive in a commercial setting, as the organisation will be able to understand and explain their system's reasoning and thus be more confident in being held accountable for its predictions. With a black box system, accountability is not possible (Winfield et al., 2021). Furthermore, as mentioned previously in this chapter, human end users of a fake review detection system are not competent at telling the difference between fake and genuine reviews themselves, the main reason being that they are written to look and sound as authentic as possible (Ott et al., 2011). Arguably, a transparent fake review detection system would allow users to better understand the problem and become better at identifying suspicious reviews themselves.

Transparency and the user

It is clear from the literature that tailoring the interface to the specific user of the given system is of paramount importance in making that system transparent. For example, the draft standard IEEE P7001 proposed by Winfield et al. (2021) requires that information shown to the user about the system is at a level of abstraction suitable and meaningful to them. This means that transparency is a property of a system for a specific stakeholder group, rather than for all stakeholders. Intuitively, a transparent system is not useful if the user does not understand it. Thus, one could argue in such a case that it is not truly transparent. For example, Bond et al. (2019) pose the example of a deep neural network (DNN) which is presented to a user with all its nodes and weights and argue that even though it is transparent in the sense that the user can see the structure of the DNN, it is not truly transparent because this structure will likely not help them understand the decision making process of the system as a whole because the information presented is not meaningful to them. As is clear with this example, a system may be deemed transparent if the information displayed to the user helps them understand the reasoning of the intelligent system and not simply because information about the intelligent system is presented.

Applying these ideas to the domain of fake review detection, this means that the level of transparency of the system would be assessed differently depending on the type of user. For instance, a researcher in the domain of fake review detection would require information transferred from the system to be at a much lower level compared to the information a business representative or customer would need to understand why a review has been identified as suspicious. For example, the end user may not be familiar with the features/attributes the model uses to form its decision and so may require additional explanations of these, whereas the researcher will likely be familiar with them already.

Transparency and mental models

A concept which further illustrates the importance of focusing on the individual user is that of a mental model, which can be seen as a useful structure of biases in an individual's cognition which helps them better understand the world (Wortham, 2020). By definition, mental models are individual and biased, based on the person's previous experiences meaning that their application may lead to false or unexpected outcomes. The general hypothesis of transparency research is that, through designing increased transparency in a given autonomous and intelligent system, the end user can form a more accurate and thus better mental model of its functioning, which is likely to reduce the chances of a misunderstanding of how the system works (Wortham, 2020). This idea of designing for transparency is also echoed in Winfield et al. (2021) where the authors argue that "transparency does not come for free" and needs to be intentionally infused into a system. The difficulty of this task varies, as previously illustrated with the example of a DNN - it is very challenging to make such a system transparent due to its inherent complexity.

A pragmatic definition of transparency

Although the above concepts and ideas are useful to think about when designing and implementing a transparent system, due to the vague definition of transparency and lack of a standard, it is seemingly challenging to objectively assess a system's level of transparency. As part of their draft standard, one of the main aims of Winfield et al. (2021) was to set out an actionable and practical definition of transparency such that all intelligent systems can be objectively assessed and tested. This is of particular interest to this project, as the main aim was to develop a transparent fake review detection system and reflect on its performance in explaining its decision making. The key points from this definition presented are as follows:

1. There is a transfer of information from the system to the user
2. The information transferred is honest, meaning the information presented does not deceive the user about the decision-making of the system
3. The information presented to the user is relevant for their understanding of the system (and/or its decision making)
4. The information is presented in a form meaningful to the user

Considering these criteria for a fake review detection system, firstly, the key information to be transferred to the user, along with the classification of a review, is the explanation of how the model reached this classification. Secondly, the explanation should be accurate and not obscure the process by which the model reached its decision and with what accuracy. Thirdly, the explanation should be useful and meaningful to the specific user in helping them understand the model's decision making. If the information presented is not relevant or the user does not understand how the model has reached its classification of a given review, the system is not transparent based on this definition.

Arguably, satisfying Winfield et al. (2021)'s definition of transparency could be done most effectively by conducting a thorough user study and evaluation with a sample of the target users as participants. In particular, satisfying points 3 and 4 above can arguably only be done with certainty through a user study, because only the user can state whether the information presented is relevant, meaningful and whether it helps them understand the decision making of the system.

Standard practices and criteria

Although a formal user study with the target users as participants arguably provides the greatest indication of the transparency of a system, there are numerous criteria for transparency and standard practices which have been highlighted in the literature as practical approaches to make a system, in particular decision-making systems like fake review detection systems, more transparent in practice. Arguably, these are useful to consider in the evaluation of a transparent system. The key criteria of note found during the literature review are summarised in Table 2.2 and described in more detail below.

Criteria	Description	Source
1	Information should be presented at a suitable level of abstraction for the given audience	Winfield et al. (2021); Weld and Bansal (2019)
2	A natural language explanation of the system's decision making should be included	Winfield et al. (2021)
3	The explanation interface should be interactive, allowing the user to interrogate the system	Weld and Bansal (2019); Winfield et al. (2021); Bond et al. (2019)
4	A system fact sheet should accompany the system	Bond et al. (2019); Mojsilovic (2018)
5	The interface should present alternative system decisions	Bond et al. (2019)
6	A confidence index should be displayed alongside the output of the system	Bond et al. (2019)
7	The output of the system may be presented in different forms for accessibility	Winfield et al. (2021)
8	Past output and associated decisions should be traceable	Winfield et al. (2021)
9	Visualisations may support user understanding and decision making	Wang et al. (2019)
10	The explanation interface should be well-suited to different scopes of explanation	Liao, Gruen and Miller (2020)

Table 2.2: Summary of standard practices and criteria for transparency

Criteria 1 - Information should be presented at a suitable level of abstraction for the given audience

This criteria has a direct link to the target user(s) of the system. If the explanation of the system's decision-making is not presented at a suitable level of detail to the particular user who will be using that system, the system will not be fully transparent.

Criteria 2 - A natural language explanation of the system's decision making should be included

According to Winfield et al. (2021), including a natural language explanation of the system's reasoning in reaching a decision increases the transparency of a system for the end user. Arguably, providing the explanation in this form is likely to increase the level of trust that the user has in the system as it explains something typically technical and complicated in a natural language form.

Criteria 3 - The explanation interface should be interactive, allowing the user to interrogate the system

This criteria is concerned with allowing the user to question the presented model decision making and get further explanations for their potential questions. According to Bond et al. (2019), "giving the user the option to interrogate likely increases trust and integrity". Weld and Bansal (2019) argue that the need for the explanation interface to be interactive arises from psychological research which has shown that explanation is a process which is best compared to "a conversation between explainer and listener", rather than a one-way process of the explainer providing a single explanation to the listener.

This criteria is very similar to offering the user the ability to see information and system explanations at different levels of detail and being able to drill up or down depending on if they need a more high-level explanation or a low-level explanation of a decision.

Criteria 4 - A system fact sheet should accompany the system

A system fact sheet is described as "metadata related to [the] algorithmic decision" made by the system (Bond et al., 2019). The analogy of the system fact sheet is a nutrition label for a given item of food, which describes what the food is made up of, where it was produced, etc. (Mojsilovic, 2018). This has numerous advantages in terms of transparency. For example, taking the nutrition label analogy further, just like a person can look at a nutrition label and determine if the food is appropriate for them, they can also look at the fact sheet of an intelligent system and determine its applicability to a given problem, its strengths and weaknesses or what data the model was trained on.

The system fact sheet arguably helps the user understand the context of a given system and its foundations, rather than only being aware of specific decisions or the high-level rules contributing to the system decision-making process.

Criteria 5 - The interface should present alternative system decisions

This criteria relates to reducing the chances of automation bias, where the user over-trusts the decision made by a given system and complacently agrees with it (Bond et al., 2019). Instead of presenting a single output or decision, this criteria suggests presenting one or more alternative decisions that can be taken by the user, helping to transfer the responsibility and accountability to the end user (Bond et al., 2019).

Criteria 6 - A confidence index should be displayed alongside the output of the system

Similar to how an alternative decision shifts the responsibility to the user, arguably this criteria does the same, as by presenting the confidence index alongside the decision, the user is more involved in accepting or rejecting a given decision based on the confidence level. Arguably, a confidence index also helps to further build user trust at the output is more honest as less information about the decision is hidden from the end user.

Criteria 7 - The output of the system may be presented in different forms for accessibility

Intuitively, this criteria makes sense as it gives the user the option to understand a given decision in different ways depending on what is clearer for them, such as a natural language explanation or a visualisation displaying the reasoning process.

Criteria 8 - Past output and associated decisions should be traceable

Traceability is a further addition to the definition of transparency where transparency should not only be a real-time feature of the system, but that past decisions can also be understood. For example, if the general principles of a decision are understood, then this would satisfy this criteria.

Criteria 9 - Visualisations may support user understanding and decision making

Wang et al. (2019) argue that whereas some explanations may be well-suited to textual explanations or lists of decision clauses, complex concepts are more effectively explained using visualisation techniques. The researchers also suggest that if the visualisations are implemented and used properly, they can be used for sensitivity analysis, that is, reasoning over what would happen to the output if the input features were to change slightly. This criteria is related to Criteria 7 since visualisations arguably can help to make the system more accessible.

Criteria 10 - The explanation interface should be well-suited to different scopes of explanation

This criteria is concerned with the scope of the explanation. In particular, Liao, Gruen and Miller (2020) highlight two common types of explanations in transparent systems: global and local. A global explanation helps the user understand how the system as a whole makes predictions, and a local explanation provides information on a single prediction. These are also referred to as "How?" (How does the system make predictions?) and "Why?" (Why is this instance given this prediction?) questions, respectively. Arguably, this criteria is vaguely related to Criteria 3, in terms of offering the user the opportunity to understand the system's decision-making at different levels of detail.

Limitations of transparency

Although transparency clearly has its benefits, there are two major limitations that are worth highlighting.

One limitation that was briefly described earlier is that of automation bias. A form of cognitive bias, it is where end users "over trust and accept computer-based advice" and complacently accept decisions recommended by intelligent systems (Bond et al., 2019). This idea is also echoed in Winfield et al. (2021), where the authors argue that transparency may lead to over-confidence in the system by the user, arguably a side-effect of the system being transparent. The authors of Bond et al. (2019) also argue that if a user does not feel competent at the problem that the system is addressing, they are very likely to always trust the decision of the AI as they are not able to reason about the problem themselves. In the case of a fake review detection system, arguably an end user will feel incompetent at identifying fake reviews. For example, as mentioned in a previous section, Ott et al. (2011) argued that human readers will often not be able to tell the difference between fake and genuine reviews because they are written to look as authentic as possible. Therefore, there is arguably an increased risk of automation bias for the domain of fake review detection.

A second major limitation is that transparency arguably increases the risk of people gaming a given system (Hunt, 2015). This means that transparency may be in tension with other ethical principles like security. This is particularly important for the domain of fake review detection. In the case of a fake review detection system, if the spammers get access to the system, they are able to craft fake reviews which lead to that review not being classified as fake, as they will understand what reasoning leads to that decision. Arguably, one way to overcome the gaming of a transparent system is to show the explanation at a high enough level to not reveal the sensitive internal decisions of that system. For example, in the case of a fake review detection system, the system may explain that a given review has been deemed fake because of the reviewer's past behaviour, while not revealing the specific behaviour in detail.

2.5.2 Interpretability and Explainability

Two approaches closely related to transparency with the goal of making the output of AI systems understandable to a user are interpretability and explainability. Interpretable and explainable machine learning approaches both seek to make a machine learning model more understandable to humans compared to a black box model where a solution has inputs and outputs, but internal decision-making and functionality is hidden (see Figure 2.2).

Although they share the same goal, explainable and interpretable systems are fundamentally different. An explainable machine learning solution uses a black box model approach for the main predictive model, such as a neural network, with a second model created to explain the first. In contrast, interpretable models are constructed to be inherently understood by humans, and so do not require any additional technologies (Rudin, 2019). Explainable approaches, such as LIME (Hu et al., 2018), use a secondary tool to create approximations of how the main machine learning model makes predictions. As argued by Rudin (2019), this is inherently problematic and such approaches should be treated with caution, especially for use cases where high-stake decisions are made.

There are several reasons why we would desire our fake review detection model to be interpretable:

1. **User awareness** - Ott et al. (2011) argue that deceptive reviews are often not identifiable by a human reader. With an interpretable model, users can be given an explanation of why a given review is likely to be fake and become better at identifying them on online platforms themselves.

2. **Guiding future research** - as noted by Zhang et al. (2016), in the domain of fake review detection there has largely been a focus on general model assessment and

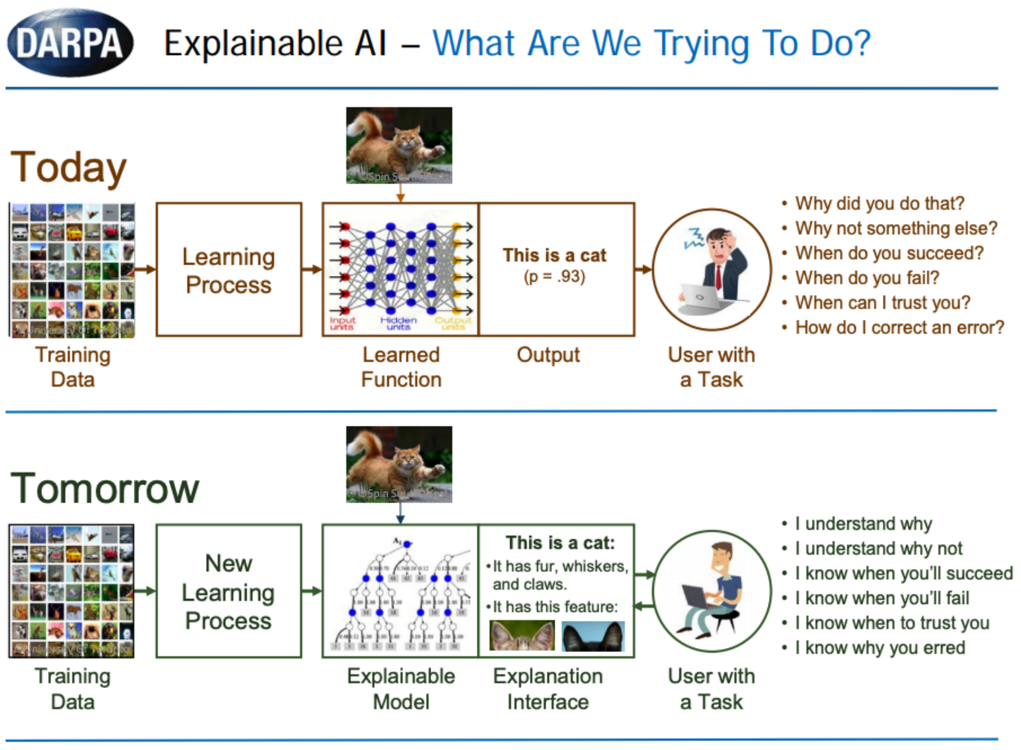


Figure 2.2: Illustration of the difference between an explainable and a black box model, from Gunning (2017).

performance, whilst theoretical insight and guidance for improving future models has been lacking. An interpretable model would provide a transparent view of the model's underlying decisions, helping researchers to narrow their focus and better understand the discriminant features of a fake review, and which features are most important for detecting a fake review. An interpretable model also has the benefit of making classification errors more transparent, meaning researchers can better understand why certain reviews may be incorrectly classified (Rudin, 2019).

3. **Accountability** - as the output of an interpretable model is explainable, such a model would be particularly attractive in a commercial setting, as its designers can be confident in how it has come to its conclusions and thus be more comfortable in being held accountable for its predictions.

Arguably, a potential danger of making fake review detection models interpretable is that people who write fake reviews acquire access to such models and learn based on the model's decision-making how to make their fake reviews look more genuine. Furthermore, there may be a trade-off between interpretability and accuracy, resulting in a model that is easier to understand but likely to be less accurate. However, as argued by Rudin (2019), this trade-off can be removed by using well-structured data with meaningful features.

In their survey of previously used methods for fake review detection model construction in academia, Ren and Ji (2019) mention some interpretable machine learning approaches, such as decision trees, Naive Bayes and linear regression, however there is a lack of reflection on what can be learnt from these models. Rather, as is the case with most other research, there is predominant focus on the machine learning models' respective accuracies. Cardoso, Silva and Almeida (2018) also compare a wide variety of classification approaches to fake review detection, but likewise do not focus on the aspect of model interpretability.

Although work has been done in the fake review detection literature to investigate discriminating features of fake reviews, to the best of the author's knowledge there has been no work done to create a transparent fake review detection system which would allow for further analysis to be done, such as understanding which features may lead to wrong or biased classification decisions.

From the literature investigated there was no work found combining the efforts of fake review detection and transparent/explainable decision-making systems. The closest work found combining these efforts was Yang et al. (2019), where the researchers developed a transparent system for fake news detection called XFake. Thus, there appeared to be an opportunity in approaching the problem of detecting fake reviews from a different angle and crafting an interpretable, transparent solution, such that end users and researchers alike could more effectively understand a fake review detection model's decision-making process.

2.6 Summary

Over the last decade or so, automated fake review detection has received strong attention from both academia and businesses due to the key role of reviews in shaping consumer behaviour and affecting purchasing decisions. This chapter has explored the key approaches and methods previously used in the literature for detecting fake reviews and established a foundation in the existing problem domain for this project.

Based on the existing literature, there was an innovative opportunity in combining a machine learning approach to fake review detection with principles of transparency and creating an interpretable model with an explainable interface to help the end user better understand how the model has come to a given decision in classifying a given review as fake or genuine. An interpretable model would also allow to better understand the features of reviews that are most important in helping to identify fake reviews to help direct further research in the domain.

Chapter 3

Requirements

Having established a solid foundation for the project with the literature review, the next step was to prepare the requirements enabling the implementation of the system that could effectively satisfy the project aims outlined in Chapter 1. This chapter outlines the requirements gathering process, key requirement decisions made during the project and the formal requirements specification enabling the creation of the software system.

3.1 Requirements Process

The main goal of the requirements process was to discover and outline the key capabilities required by the system such that it could be implemented to fulfill the project aims outlined in Chapter 1 as effectively as possible. At a high level, the requirements had to meet the two-fold project objective of creating a machine learning model capable of classifying reviews, which needed to work within a transparent framework such that its decision making could be effectively explained to the end user.

The requirements were predominantly derived from the state of the art and previous work done in the fake review detection and transparency domains (examined in Chapter 2), as well as the project aims and proposed aspects of originality for the system (described in Chapter 1). Although some requirements were known early on in the project, the requirements were generally iteratively updated during the project life-cycle as a better understanding of the problem was acquired and the domains within which it exists.

3.2 Key Requirement Decisions

This section presents the most important high-level requirement decisions considered during the project life-cycle which were crucial for achieving the five project aims outlined in Chapter 1 as effectively as possible and ensuring the success of the project. The key decisions relate to the machine learning approach, dataset, the explainability approach and the programming language used for the implementation.

Machine Learning Approach

Selecting the appropriate machine learning approach was a key requirement decision that would have an effect on the capabilities of the rest of the system. Specifically, due to the two-fold focus of the project of creating a fake review detection classifier with comparable performance to state of the art solutions combined with the novelty feature of creating an explainable interface, the approach had to satisfy both of these objectives.

A number of different machine learning approaches in the Literature and Technology Review (Section 2.2) were considered and given that supervised learning models have historically performed better than unsupervised learning approaches and is the most popular approach within the domain, as well as the fact that a well-regarded dataset was acquired, a supervised learning approach was seen as the best option to satisfy Project Aim 2 of creating a classifier with comparable performance to the literature.

To also effectively satisfy Project Aim 3 of developing a transparent solution, it was decided that a decision tree classifier would be the most appropriate choice. Not only does it have decent performance based on previous work explored in Section 2.2.1, but it is also a fundamentally interpretable approach which is not too complex to explain to a naive end user, as per Project Aim 3, as compared to other approaches such as random forest (RF) and other more complex classifiers. Furthermore, being an interpretable classifier, the model's decision-making would not have to be approximated as is the case with black box models, which in itself can be fallible. It was decided that using more than one type of classifier and comparing their performances would be out of scope and unnecessary for achieving the project aims, other than potentially helping to achieve a better performing classifier.

For feature engineering, it was decided that both review and reviewer-centric attributes would be used, as the literature reviewed suggested that reviewer-centric features are increasingly important for classifier performance (Section 2.4). Due to the natural time constraints of the project, an in-depth exploration of potential features and an investigation of feature importance was not possible. Thus, to meet Project Aim 2 of creating a classifier with comparable performance to the literature, most features used were sourced from previous systems highlighted in Section 2.4.1 in the Literature and Technology Survey.

Dataset

The dataset to be used for the machine learning model was another important consideration for the requirements. Due to the lack of publicly available and labelled datasets, at the initial stages of the project it was unknown whether it would be possible to acquire such a dataset or alternative approaches had to be considered, such as pseudo-fake reviews or using an alternative machine learning approach. Multiple researchers within the field were contacted by email and fortunately access was provided to a labelled dataset sourced from the business review website Yelp which has previously been used by a number of researchers within the domain including Zhang et al. (2016), Mukherjee et al. (2013b) and Cardoso, Silva and Almeida (2018). The dataset was provided by Professor Bing Liu, one of the authors of Mukherjee et al. (2013a) and Mukherjee et al. (2013c).

The acquired dataset was labelled, domain-specific (restaurant and hotel reviews) of a fair size of around 67,000 reviews (for restaurant reviews) and, based on the literature review, has been well-regarded within the domain and one of the closest ground-truth labelled datasets available in terms of real-life reviews (Mukherjee et al., 2013b). Having considered the advantages and disadvantages of this dataset in the Literature and Technology Review (Section 2.3.2), we argued that the acquired dataset is likely to be a better choice for building a fake review detection model compared to using alternative approaches like crafting pseudo-fake reviews using crowd-sourcing. Even though the Yelp dataset has some drawbacks, these were seen as minor compared to the drawbacks of using a pseudo-fake review dataset.

Explainability Approach

Another key requirement decision was deciding the approach to use for achieving the transparency of the machine learning model. As the decision tree was chosen as the supervised learning approach to satisfy Project Aims 2, 3 and 4, it was seen as reasonable to use the inherent explainable characteristics of the decision tree, namely the structure of the decision tree and decision paths, to explain the decision-making of the model to the end user. Therefore, being an interpretable classifier, the decision tree set the foundation for the explainable interface in allowing for its underlying structure and decision paths for specific samples to be extracted.

In achieving the non-functional aspect of Project Aim 3 in producing a transparent and explainable front end layer which presents the decision making of the model to a naive user in an understandable form, we decided early on that a formal user study and evaluation would be out of scope due to the time constraints of the project and the already two-fold focus on developing a machine learning model and creating a transparent front end layer, along with an evaluation of both (Project Aim 4). Instead, Project Aim 3 was to be achieved by following the transparency criteria and best practices presented in the state of the art, as per Section 2.5.1 in the literature review. Based on the literature surveyed, these practical approaches help to make an intelligent decision-making system more transparent in practice. These criteria were also to be used in evaluating the performance of the implemented explainable interface to satisfy Project Aim 4.

Programming Language

Another decision with a large effect on the success of the project was the choice of programming language and associated technologies, which would play an important role in helping to satisfy the practical aspects of the project, namely Project Aims 2, 3 and 4. The chosen languages or technologies would need to assist in loading and preparing the dataset acquired, constructing a feature set, selecting feature subsets, training and evaluating a decision tree classifier on the feature sets, and building a transparent front end layer to explain the decision making of the model. Being a general-purpose language with an extensive selection of libraries and frameworks capable of satisfying the listed capabilities, Python was selected for use in this project before development started. Although other languages were considered, Python was ultimately chosen for its versatility and general-purpose nature, combined with the author's previous experience using Python and their interest in further developing their skills.

Python is a hugely popular language with widespread usage, meaning it has a lot of support in the form of high-quality documentation and an active community of developers. Moreover, Python's use in the domain of fake review detection has been demonstrated in papers such as Kumar et al. (2019).

In terms of specific frameworks, Python is well-known for its machine learning libraries including *scikit-learn*[1] and *PyTorch*[2], natural language processing libraries (e.g., *NLTK*[3] and *spaCy*[4]), as well as general libraries for processing and manipulation of large datasets, particularly *pandas*[5]. It was discovered early on in the project that Python also has a module providing an interface to SQLite databases (*sqlite3*)[6] which would be needed to connect to and query the Yelp dataset acquired, which was in SQLite database format. Early on in the project, a simple preliminary program was written in Python which confirmed that the fake review dataset acquired could be successfully accessed and manipulated using sqlite3 and pandas.

[1]https://scikit-learn.org/stable/
[2]https://pytorch.org/
[3]https://www.nltk.org/
[4]https://spacy.io/
[5]https://pandas.pydata.org/
[6]https://docs.python.org/3/library/sqlite3.html

3.3 Requirements Specification

This section presents a formal description of the software system to be built. The specification consists of functional and non-functional requirements (denoted using NF) for the three main categories of requirements needed to implement the system. Together, the requirements from each category form the system capabilities needed for the transparent fake review detection system as a whole, based on which the system can be implemented.

Each requirement has a measure of importance associated with it: high, medium or low. These can be interpreted as critical, useful and desirable to have in the software product, based on their relative importance in achieving the five project aims outlined in Chapter 1.

Each requirement has an associated source in terms of requirements gathering, such as the related section in the Literature and Technology Review which informed the requirement or the specific project objective which the requirement supports in satisfying.

3.3.1 Data Requirements

The following requirements are concerned with the dataset that will be used to train the machine learning model to satisfy the project aims. Section 2.3 of the Literature and Technology Review covers these ideas in more detail.

	1. Data Requirements	**Priority**	**Source**
1.1	The dataset must contain labelled ("genuine"/"fake") reviews	High	Section 2.3; Project Aim 2
1.2	The reviews in the dataset must be from the same domain, e.g., restaurant reviews	High	Section 2.3.2; Project Aim 2
1.3	The dataset must be balanced to overcome the data imbalance problem	High	Section 2.3.3; Project Aim 2
1.4	(NF) The dataset should be sourced from a real-life application or commercial website	Medium	Section 2.3.2, Project Aim 2
1.5	(NF) If a real-life dataset is not available, the dataset must be crafted through AMT	High	Section 2.3.1; Project Aim 2
1.6	(NF) The dataset must be of a suitably large size for model training and testing	High	Project Aim 2, 4

3.3.2 Model Requirements

The requirements below outline the capabilities required to create the classifier and evaluate its performance to satisfy the relevant project aims. Sections 2.2 (Fake Review Detection Approaches) and 2.4 (Feature Engineering) of the Literature and Technology Review cover these ideas in more detail.

	2. Model Requirements	**Priority**	**Source**
2.1	The classifier must be trained on a combination of review-centric and reviewer-centric features	High	Section 2.4; Project Aims 2, 4
2.2	The features extracted must be similar to features previously used in the literature	High	Section 2.4.1; Project Aims 2, 4
2.3	The importance of individual features with regards to the class label must be analysed using a correlation metric and t-test	High	Project Aim 2
2.4	Principal Component Analysis (PCA) should be used to explore the correlation between sets of features	Medium	Section 2.4.2; Project Aims 2, 4
2.5	5-fold or 10-fold stratified cross-validation must be used to estimate the model's performance on unseen data	High	Section 2.2.4; Project Aims 2, 4
2.6	The following evaluation metrics must be used in evaluating the model's performance: accuracy, balanced accuracy, precision, recall and F1 score	High	Section 2.2.4; Project Aims 2, 4
2.7	The model must be trained with a balanced class distribution of fake and genuine reviews, and tested with a natural class distribution	High	Sections 2.2.4, 2.3.3; Project Aims 2, 4
2.8	The model's hyper-parameters must be tuned to reduce overfitting and improve the performance of the model	High	Project Aim 2
2.9	The model's hyper-paramaters may be optimised using Random Search (RS) or another numerical optimisation method	Low	Project Aim 2
2.10	(NF) The classifier chosen must be interpretable such that its decision making can be explained in the front end layer	High	Project Aims 3, 4
2.11	(NF) The final model must have comparable performance to models presented in literature	High	Project Aim 2

3.3.3 Front End and Transparency Requirements

The requirements below are concerned with the capabilities required to develop the transparent front end layer of the system and evaluate its performance in effectively explaining the decision making of the model to the end user (Project Aims 3 and 4). Section 2.5 (Transparency and Explainability) of the Literature and Technology Review covers the ideas of transparency and explainability in detail.

	3. Front End Requirements	**Priority**	**Source**
3.1	The decision making process for a given review classification must be presented in a text form outlining the series of decisions taken by the model	High	Section 2.5.1 ; Project Aim 3
3.2	The decision making process for a given review classification must be presented in a visual form, such as a diagram	High	Section 2.5.1; Project Aim 3
3.3	The user must be able to select a review to be classified and displayed in the explainable interface	High	Project Aim 3
3.4	The performance of the front end in terms of transparency must be evaluated using transparency criteria and best practices used in the state of the art	High	Section 2.5.1; Project Aim 3
3.5	The output for a given classification of a review should detail the confidence index of the decision	Medium	Section 2.5.1; Project Aim 3
3.6	(NF) The explanation of the model's decision making should be intuitive and understandable by a naive user or non-expert, such as a consumer	High	Project Aim 3
3.7	(NF) The interface must be simple and user-friendly	High	Section 2.5.1; Project Aim 3
3.8	(NF) A classification and its associated explanation should be generated and shown quickly to the user for a given review chosen	Medium	Project Aim 3

3.4 Summary

This chapter has presented the requirements process, the key decisions made in terms of requirements and a formal requirements specification describing the software system to be implemented. The requirements were based predominantly on the findings during the Literature and Technology Review, as well as the five project aims outlined in the first chapter, in order to ensure the system to be implemented had the capabilities needed for the project to be a success. In the next chapter, we will see how the requirements were implemented to create the transparent fake review detection system.

Chapter 4

Implementation

This chapter describes how the requirements presented in the previous chapter were implemented in practice to achieve the project aims. The chapter presents the two key parts of the implemented system - the classifier capable of detecting fake reviews and the front end in the form of a user interface which explains the decision-making of the classifier to the end user. Before describing each of these parts in more detail, the programming and technology stack is outlined, as well as an overview of the system as a whole.

4.1 Technology Stack

As briefly outlined in the previous chapter, a key requirements decision was to select an appropriate technology stack for implementing the project requirements. Python and Jupyter Notebook were selected for development and testing of both the classifier and the user interface. The Anaconda distribution was used to manage the various Python packages used for implementing the system. After development was finished, Python scripts were exported to be run in the terminal/command line. In particular, the user of the front end can run the program on their terminal/command line.

Python was chosen for a number of key reasons, including that it is a general-purpose language with an extensive selection of packages. Furthermore, the language is a popular choice within the machine learning community in particular, and has great online documentation. Further to these reasons, the author wanted to further improve their knowledge and practical experience in this language.

4.2 System Overview

The full implementation consisted of two key parts: the creation of the machine learning model and selection of the best-performing classifier, and the development of the user interface which displays the decision-making of the selected classifier.

Before outlining the implementation of the classifier and user interface in more detail, an illustration is presented in Figure 4.1 to the reader of the full implemented system to highlight the individual components that bring about the desired capabilities.

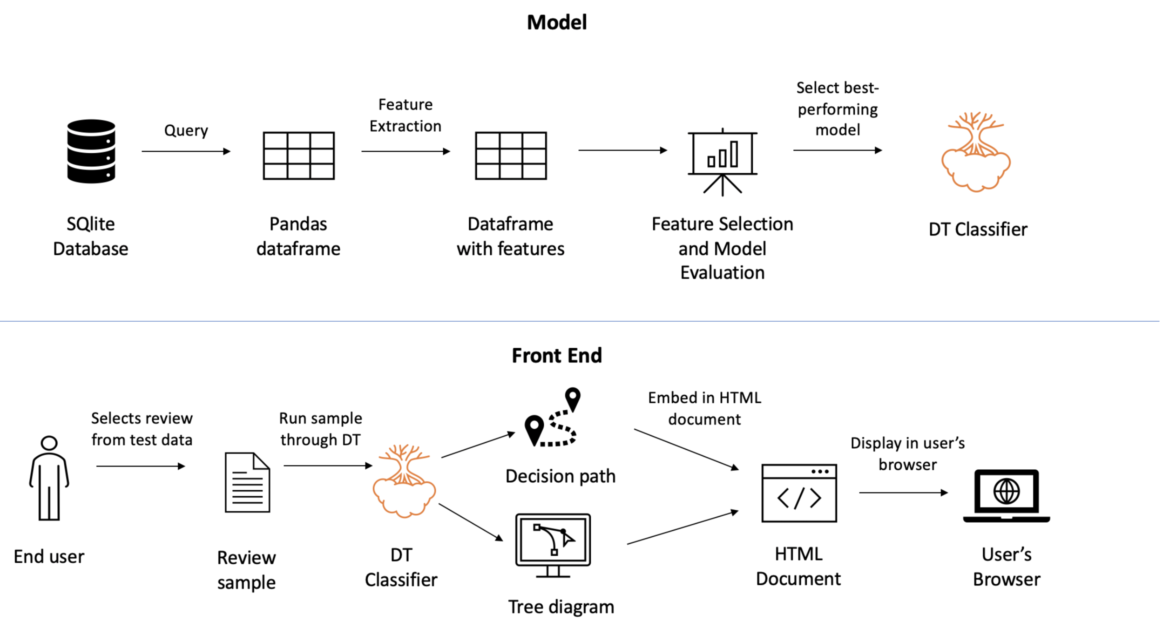


Figure 4.1: System overview illustration

Figure 4.1 presents the main components of the implementation for both the machine learning model and the front end (user interface). The implementation of the model is covered in Section 4.3, with the user interface implementation being covered in Section 4.4. Arguably, the main output of the model implementation, as highlighted in orange in Figure 4.1, is the decision tree (DT) classifier. This classifier is used in the front end to classify reviews that the user selects and explain the decision-making behind the classifications to the user. For this, the trained decision tree's inherently interpretable functioning is used to explain the decision making to the user in the form of the decision path and tree diagram, as can be seen in the figure.

4.3 Model Implementation

4.3.1 Dataset and Data Preparation

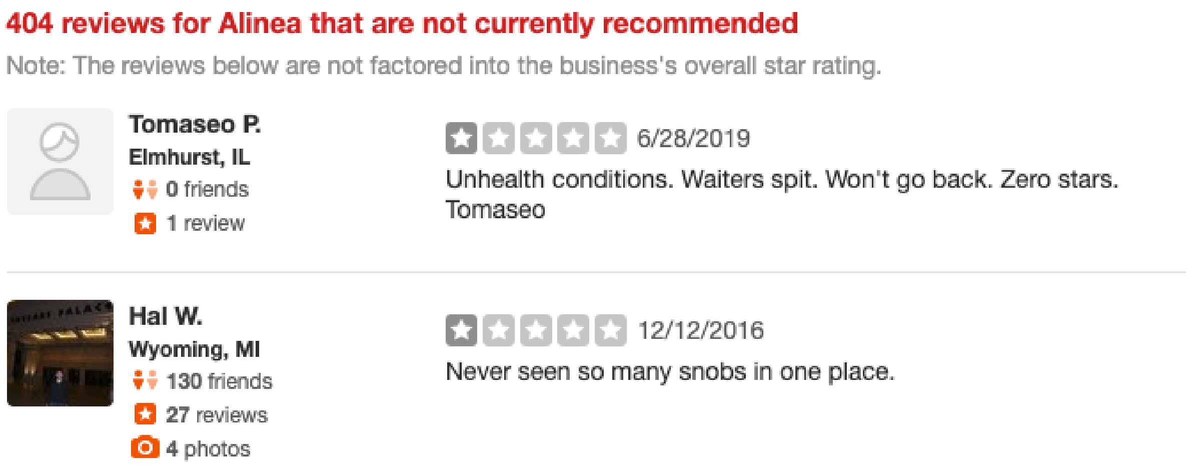
As the chosen machine learning approach was that of supervised learning, based on the requirements presented earlier, the classifier would need to learn from a set of input-output pairs, where the input is a set of features or attributes and the output is the binary classification: genuine or fake. Here we provide an overview of the dataset used and present a breakdown of the attributes contained within it.

About the Dataset

As briefly described in the previous chapter, the dataset used for this project was provided by Professor Bing Liu from the University of Illinois at Chicago (UIC). The dataset has previously been used in a number of research papers in the fake review detection domain, such as in Zhang et al. (2016) and Cardoso, Silva and Almeida (2018). The dataset is labelled and sourced by the researchers from the business review site Yelp containing reviews from a range of restaurants and hotels in the United States. For this project, only the restaurant reviews were chosen to keep the domain of reviews consistent in line with Requirement 1.2.

According to Mukherjee et al. (2013c), the labels for the dataset ("genuine" and "fake") were acquired through scraping the Yelp website, as the website has a proprietary, secret algorithm publicly flagging the reviews which it reasons are fake or suspicious, an example of which can be seen in Figure 4.2. In this figure, an example can be seen of two fake reviews according to Yelp for Alinea restaurant in Chicago[1]. These reviews, which are not recommended by Yelp to the user, have been used to label the "fake" reviews in this dataset. As can be seen in the figure, these flagged reviews are ignored in the business's overall ratings.

The genuine reviews in the dataset are those which Yelp recommends and are displayed on the business's review page, an example of which can be seen in Figure 4.3 for the same restaurant.


404 reviews for Alinea that are not currently recommended

Note: The reviews below are not factored into the business's overall star rating.

Tomaseo P.
Elmhurst, IL
0 friends
1 review

6/28/2019
Unhealth conditions. Waiters spit. Won't go back. Zero stars.
Tomaseo

Hal W.
Wyoming, MI
130 friends
27 reviews
4 photos

12/12/2016
Never seen so many snobs in one place.

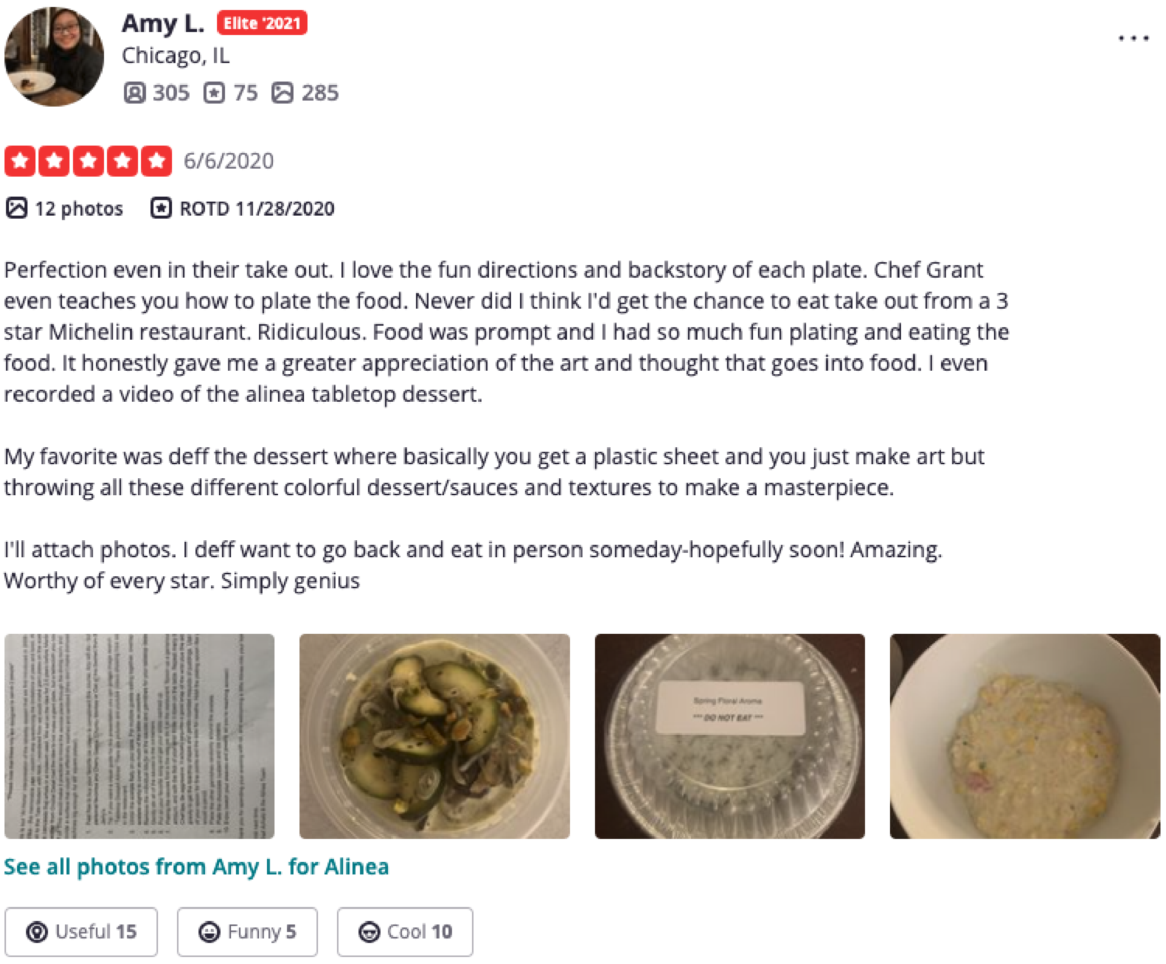
Figure 4.2: Example of fake reviews on Yelp.com

Tables and Dataset Statistics

The dataset is contained in a database file in SQLite format and contains three tables: Review (table 1), Reviewer (table 2) and Restaurant (table 3). Table 3 contains information on each restaurant, such as their phone number, whether their restaurant is wheelchair accessible, and other such attributes, which were not required for this project and thus this table was not used. The attributes found in tables 1 and 2 can be seen in Tables 4.1 and 4.2.

The restaurant dataset used contains a total of 67,019 reviews, of which 58,716 are labelled as genuine and 8,303 labelled as fake. Thus, a small percentage of 12.4% of reviews are fake, which highlights the large imbalance in the two classes. The reviews were posted from 2004-2012, with the large majority (92%) posted in the five years spanning 2008-2012.

[1]https://www.yelp.com/biz/alinea-chicago?osq=Alinea+Chicago



Amy L. Elite '2021

Chicago, IL

305 75 285

6/6/2020

12 photos ROTD 11/28/2020

Perfection even in their take out. I love the fun directions and backstory of each plate. Chef Grant even teaches you how to plate the food. Never did I think I'd get the chance to eat take out from a 3 star Michelin restaurant. Ridiculous. Food was prompt and I had so much fun plating and eating the food. It honestly gave me a greater appreciation of the art and thought that goes into food. I even recorded a video of the alinea tabletop dessert.

My favorite was deff the dessert where basically you get a plastic sheet and you just make art but throwing all these different colorful dessert/sauces and textures to make a masterpiece.

I'll attach photos. I deff want to go back and eat in person someday-hopefully soon! Amazing. Worthy of every star. Simply genius

See all photos from Amy L. for Alinea

Useful 15 Funny 5 Cool 10

Figure 4.3: Example of a genuine review on Yelp.com

Attribute	**Description**
reviewID	The unique identifier of the review posted
date	The date the review was posted
reviewContent	The review itself, as a string
flagged	The class label: N or Y, i.e., genuine or fake, respectively
reviewerID	The unique identifier of the reviewer who posted the review
restaurantID	The unique identifier of the restaurant
rating	The rating of the review from 1-5
usefulCount	The number of "useful" votes the review has received
coolCount	The number of "cool" votes the review has received
funnyCount	The number of "funny" votes the review has received

Table 4.1: Review table (table 1) attributes

Attribute	**Description**
reviewerID	The unique identifier of the reviewer
yelpJoinDate	The date the reviewer created a profile on Yelp
name	The name of the reviewer
location	The location of the reviewer
friendCount	The number of friends the reviewer has on Yelp
reviewCount	The number of reviews the reviewer has posted
usefulCount	The number of "useful" votes the reviewer has received
coolCount	The number of "cool" votes the reviewer has received
funnyCount	The number of "funny" votes the reviewer has received
complimentCount	The number of compliments the reviewer has received
tipCount	The number of tips the reviewer has received

Table 4.2: Reviewer table (table 2) attributes

"Anonymous" Reviewers

During data preparation it was found that there was a discrepancy between the reviewers found in the Reviewer and Review tables. Specifically, there was a total of 16,941 unique reviewerIDs in the Reviewer table (table 2) and 35,027 unique reviewerIDs in the Review table (table 1). A total of 18,352 reviewers found in table 1 were missing from table 2.

These missing reviewers were assumed to be anonymous or those without a profile on Yelp, or potentially those that deleted their profile after posting a review and thus had their information deleted. However, this assumption may have been wrong as the missing reviewers may be due to another reason, such as the tables being inconsistent or out of date.

Connecting to the Database

As mentioned in the Requirements chapter, the Python library *sqlite3* was used to provide an interface for accessing the SQLite database and retrieving the tables found in the database file.

In Listing 1, the code performing the function of connecting to the SQLite database file *yelpResData.db* can be seen (lines 10-14). After connecting to the database and creating a cursor object to interact with the database, we query the Master table which is a system table present in all SQLite databases returning all database objects present in the database. Querying the Master table for objects of type "table" (lines 20-21), we are given the three tables mentioned previously: review, reviewer and restaurant. At this point, the table names were used to write SQL queries to return these tables and store them in pandas dataframes (lines 23-25). As mentioned earlier, the restaurant table was not used and hence is commented out in the code.

```
 9  # set up connection to the database
10  connection = sqlite3.connect("yelpResData.db")
11  # create cursor object for interaction with the database
12  cur = connection.cursor()
13  # specify how to handle bytes in database
14  connection.text_factory = lambda x: str(x, 'utf-8', 'ignore')
15
16  # previously attempted text factories below
17  #connection.text_factory = bytes
18  #connection.text_factory = lambda x: str(x, 'iso-8859-1')
19
20  cur.execute("SELECT name FROM sqlite_master WHERE type='table';")
21  print(cur.fetchall())
22
23  review_table = pd.read_sql_query('SELECT * FROM review', connection)
24  #restaurant_table = pd.read_sql_query('SELECT * FROM restaurant', connection)
25  reviewer_table = pd.read_sql_query('SELECT * FROM reviewer', connection)
```

Listing 1: Importing Review and Reviewer tables into dataframes from database file

Data Preparation

After retrieving the rows from the two tables in the database, a series of data preparation steps were applied before feature engineering could commence. Firstly, the Unicode encoding in the reviewContent column of the review table was normalised as there were some errors present when the table was retrieved from the database. Then, the labelled reviews from the review table were extracted. Thirdly, the date column had some values with strings present - these were removed and the column was set to be of datetime format. The table was checked for null reviews, of which there were two - these were removed. Finally, a subset of columns in the review table were extracted to be used for feature extraction and selection: date, reviewerID, reviewContent and flagged. An additional column was added called hasProfile to signify if the reviewer associated with a given review was present in the reviewer table (table 2), as per the assumption made about "anonymous" reviewers. Finally, the attributes "usefulCount", "coolCount" and "funnyCount" from Table 4.1 were also added to this subset of columns.

4.3.2 Feature Extraction

The prepared dataset setting a solid foundation for the project, the next step was to carry out feature engineering to create a vector of feature values based on the dataset's attributes to serve as input to the learner with the corresponding labels for each review to produce a classifier. Based on the input vector of feature values $\mathbf{x} = (x_1, x_2, x_3, ..., x_n)$, where each x_n is a single feature, the binary classifier will then predict for a given review sample one of two classes: genuine ('N') or fake ('Y').

The process of feature engineering was broadly split into two iterative steps: feature extraction and feature selection, the former of which will be covered in this section. The feature extraction process is a key step in a machine learning project, and as argued by Domingos (2012), the quality of features is the single most important factor in successful machine learning projects.

Feature Extraction Process

The feature extraction process for this problem was manual in nature, each feature being explicitly defined, rather than making use of feature learning where an algorithm or model discovers features to be used for the classifier. There were several reasons for this approach. Firstly, the manual feature extraction approach is dominant in the fake review detection literature, as seen in Chapter 2. Secondly, arguably automated feature learning could have led to features being extracted that were not interpretable or hard for a potential user to understand as they are not explicitly defined, conflicting with the project aim of creating a transparent solution which can be understood.

Therefore, based on the project aims, the approach determined was to manually extract features based on the types of features previously used in the domain (explored in Chapter 2) and then apply feature selection techniques to select the best individual features and create feature sets to evaluate the classifiers and identify the best performing classifier among them.

Features Extracted

Tables 4.3 and 4.4 summarise the 15 features extracted, broken down by the two main categories of the attributes present in the dataset: review-centric features and reviewer-centric features. Whereas the former is based exclusively on the review text, the latter describes some information about the reviewer's behaviour or general review metadata. Both categories of features were extracted to satisfy Requirements 2.1 and 2.2.

All features were extracted from the review table (Table 4.1) of the dataset, except for the *hasProfile* feature which was extracted using both the review and reviewer tables (Tables 4.1 and 4.2). All features except for *hasProfile* were extracted based on the types of features previously used in the literature and reviewed in Chapter 2. The *hasProfile* feature was extracted based on the previously described "anonymous" reviewers identified during the data preparation stages of the project.

Features 13-15 were not modified from the data preparation stage and were extracted directly from the corresponding attributes found in the Review table (Table 4.1). Most of the other features extracted, such as *reviewLength*, *hasProfile* and *postCount* are regarded to be relatively trivial in implementation and will therefore not be covered in depth in

Feature	Feature Name	Description
1	reviewLength	the length of a given review in characters
2	avgSentiment	the average sentiment score of sentences in a given review
3	nounProp	the proportion of nouns in a given review of all word tokens
4	verbProp	the proportion of verbs in a given review of all word tokens
5	adjProp	the proportion of adjectives in a given review of all word tokens
6	propernounProp	the proportion of proper nouns in a given review of all word tokens
7	numCount	count of numbers in a given review
8	symCount	count of symbols in a given review
9	maxTfidf	the highest tf-idf score (term frequency – inverse document frequency) of a unigram (single word) in a given review

Table 4.3: Review-centric features extracted

Feature	Feature Name	Description
10	hasProfile	whether the reviewerID associated with a given review in the review table is found in the reviewer table in the database
11	maxReviews	the maximum number of reviews that the reviewer associated with a given review has posted in a single day
12	postCount	the number of posts that the reviewer associated with a given review has
13	usefulCount	number of "useful" likes for the given review (see Figure 4.2)
14	coolCount	number of "cool" likes for the given review (see Figure 4.2)
15	funnyCount	number of "funny" likes for the given review (see Figure 4.2)

Table 4.4: Reviewer-centric features extracted

this section (see Tables 4.3 and 4.4 for their description and Appendix C.2 for their implementation in Python). Instead, this section will cover the features that were less straight-forward to extract and requiring a number of external libraries, such as VADER for sentiment analysis, and Natural Language Tool Kit (NLTK) and SpaCy for natural language processing.

avgSentiment **feature**

A feature based on sentiment of the review has been previously used in the fake review detection literature by a number of researchers, such as Zhang et al. (2016). The intuition behind such a feature is that a fake review aiming to deceive the people reading it will aim to express a particular view more strongly on average compared to a genuine review (Ott, Cardie and Hancock, 2013). In the case of a genuine review, the reviewer is arguably stating their genuine opinion and not trying to influence the reader directly in a positive or negative way.

The feature was implemented using modules from the Natural Language Tool Kit (NLTK), a popular collection of libraries for natural language processing in Python. Specifically, the VADER (*Valence Aware Dictionary for sEntiment Reasoning*) module[2] was used to get the polarity of each sentence in the review. VADER is a sentiment analysis tool which has been tuned to texts from social media and microblogs, utilising both a lexicon of words with manually labelled sentiment polarity scores (e.g., "bad" will have a low score, and "great" will have a higher score), and a set of rules abstracting how sentiment is expressed in text (Hutto and Gilbert, 2014). VADER has been shown to perform well in diverse contexts, including movie and product reviews (Hutto and Gilbert, 2014).

The VADER sentiment analyser in Python calculates a unidimensional polarity score for a given sentence by summing the sentiment scores of each word in the lexicon and then adjusting this score according to the set of syntactical rules, and then normalising the score to be between -1 (most negative) and +1 (most positive) (Hutto and Gilbert, 2014). This is called the "compound" sentiment score, the computation of which can be seen in Listing 2 in lines 86-87. Before the sentiment analyser was applied, each review was tokenised into a list of sentences (line 80) using the tokeniser from NLTK, with the sentiment score being calculated for each sentence and averaged at the end, once all sentences were processed. Pre-processing to remove symbols such as "?" and "!", capitalisation of letters and numbers was not done, as these play a role in the calculation of sentiment, being part of the rules VADER uses to gauge the sentiment of a sentence.

To illustrate the avgSentiment feature further, below are two examples of reviews picked from the Yelp dataset, one with a high avgSentiment value and the other with a low avgSentiment value.

Example of a review with an avgSentiment value of 0.971 (very positive sentiment):

"Ok....Coast is my favorite Sushi place...we try to go as much as we can ..this are the things that I love: 1 is BYOB, there is nothing better than your own wine 2 White Dragon Roll........we order at least 2 or 3 everytime we go 3 Great ambiance and crowd Because of being so popular there is always a wait of an hour or more.. =(but they do take reservations, which is great, also the service is ok and sushi takes a while..but all the cons go away because of the awesome white dragon...love it"

[2]https://www.nltk.org/_modules/nltk/sentiment/vader.html

```python
# Extract Feature 2 - Average sentiment of review
###

# initialise Sentiment Intensity Analyzer (SIA)
analyzer = SIA()
# create empty list to store all sentiment values
sentiment_list = []

# calculate for each review an average sentiment
# value based on individual sentences
for review in df['reviewContent']:

    # 1. split review into individual sentences (tokenise)
    review_sentences = tokenize.sent_tokenize(review)

    # 2. get the average sentiment for the review
    # (average sentiment of sentences in review)
    total_sentiment = 0.0
    for sentence in review_sentences:
        sentiment_score = analyzer.polarity_scores(sentence)
        total_sentiment += sentiment_score["compound"]

    # calculate average sentiment
    average_sentiment = round(total_sentiment / no_sentences, 4)

    # 3. append average sentiment to sentiment_list
    sentiment_list.append(average_sentiment)

# add new feature to dataframe
df["avgSentiment"] = sentiment_list
```

Listing 2: Creating *avgSentiment* feature

Example of a review with an avgSentiment value of -0.691 (very negative sentiment):

"For me, if you idiots judge a pastry shop by there cup cakes you need a kick in the ass .the way i see it one day martha stewart had a show on cup cakes and all these wacked out prozac queens could not find vanilla extract because they drank it .so they made it some elses problem ..now it is a cup cake marrygoround out there .bull shit ..... Alliance has some great product eat that and shut up..."

Parts of Speech features

Features 3-8 were all extracted using a similar approach, namely parts of speech (PoS) tagging. As seen in the Literature and Technology Review, PoS features are commonly used in the fake review detection domain, with Zhang et al. (2016) suggesting that both the frequency distribution and number of PoS tags can be used as suitable features. The intuition behind using PoS as a feature in a fake review detection model is that PoS tags can be used to indicate the genre of a text, with fake reviews assumed to be of an imaginative genre and truthful reviews of an informative genre (Ott et al., 2011).

A popular natural language processing library called SpaCy was used for tokenising each review into a list of words and tagging each word with the appropriate parts of speech tag, such as "ADJ" for adjectives and "PUNCT" for punctuation[3]. At a high level, the SpaCy PoS tagger is a probabilistic model, which for each word predicts a given PoS tag based on patterns of PoS tags in the data that the tagger was trained on (Honnibal and Montani, 2017). An example of such a pattern given in the SpaCy documentation is a word following the token "the" is highly likely to be a noun in the English language[4]. Although it is possible to train the PoS tagger offered by SpaCy on any types of texts, in the interest of time a pre-trained pipeline from SpaCy was used called en_core_web_sm, which is trained on written web text including blogs, news and comments in the English language[5].

Listing 3 shows the code snippet from the feature engineering code which performs the function of extracting the PoS features summarised in Table 4.3.

[3]https://universaldependencies.org/docs/u/pos/

[4]https://spacy.io/usage/linguistic-features#pos-tagging

[5]https://spacy.io/models/en#en_core_web_sm

```python
# Extract Features 3-8 - Parts of Speech (PoS)
###

# load parts of speech tagger from spaCy library
tagger = spacy.load("en_core_web_sm")

noun_proportion_list = []
adj_proportion_list = []
verb_proportion_list = []
propnoun_proportion_list = []
num_count_list = []
symbol_count_list = []

for review in df['reviewContent']:

    # 1. tokenise and tag review
    tagged_review = tagger(review)

    # get count of tokens (words) in review
    token_count = len(tagged_review)

    noun_count = 0
    adj_count = 0
    verb_count = 0
    propnoun_count = 0
    num_count = 0
    symbol_count = 0

    # 2. Get counts of nouns, adjectives, verbs,
    # proper nouns, numbers and symbols
    for token in tagged_review:

        if(token.pos_ == "NOUN"):
            noun_count += 1
        if(token.pos_ == "ADJ"):
            adj_count += 1
        if(token.pos_ == "VERB"):
            verb_count += 1
        if(token.pos_ == "PROPN"):
            propnoun_count += 1
        if(token.pos_ == "NUM"):
            num_count += 1
        if(token.pos_ == "SYM"):
            symbol_count += 1

    # 3. append proportions to lists
    noun_proportion_list.append(noun_count / token_count)
    adj_proportion_list.append(adj_count / token_count)
    verb_proportion_list.append(verb_count / token_count)
    propnoun_proportion_list.append(propnoun_count / token_count)
    num_count_list.append(num_count)
    symbol_count_list.append(symbol_count)
```

Listing 3: Creating Parts of Speech (PoS) features

```python
# Extract Feature 9 - Maximum TF-IDF score for a given review
###

# create corpus - a list of all reviews
corpus = df["reviewContent"].tolist()
# initialise vectoriser and assign stop word list
# words appearing in more than 85% of documents will
# not be part of the vector
vectoriser = TfidfVectorizer(max_df = 0.85, stop_words = "english")
# fit and transform TF-IDF model on corpus
vocabulary = vectoriser.fit_transform(corpus)
max_tfidf_list = []

# loop through all reviews and
# retrieve their maximum TF-IDF value
for document_index in range(len(corpus)):

    max_value = vocabulary[document_index].max()
    max_tfidf_list.append(max_value)

# add new feature to dataframe
df["max_tfidf"] = max_tfidf_list
```

Listing 4: Creating *maxTfidf* feature

maxTfidf **feature**

TF-IDF has two main components: term frequency (TF) and inverse document frequency (IDF). A high TF-IDF score indicates that the frequency of the word in a given review was high, whereas the frequency of the word across the corpus as a whole was low, indicating that the word is significant or unique to a given document. Arguably, the logic behind this feature in a fake review detection model is that either fake reviews will tend to repeat important words often or spammers may use very similar words to other reviews, leading to very low or very high scores for words in the review.

This feature was extracted using the TF-IDF vectoriser from scikit-learn[6]. Firstly, the corpus (collection of all documents to be analysed) was created as a list of all reviews in the dataset. Next, the TF-IDF vectoriser from scikit-learn was fit to the corpus, creating a vocabulary matrix with each row signifying the specific document (review) and columns signifying the TF-IDF values for individual words, i.e., this feature was based on uni-grams. Stop words like "the" and "and" were removed by the vectoriser. Furthermore, words appearing in more than 85% of documents were added to the stop word list.

After creating the TF-IDF matrix, the TF-IDF vector for each review was analysed for the highest TF-IDF value, which was the feature value used. The implementation of this feature can be seen in Listing 4.

[6]https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

4.3.3 Feature Selection

After extracting the 15 review-centric and reviewer-centric features presented in the previous section, the next stage of the feature engineering process was to apply feature selection techniques to satisfy Requirement 2.3 of assessing the importance of individual features.

Feature Selection Process

The goal of the feature selection process was two-fold. Firstly, feature selection was done as an exploratory exercise to investigate the importance of the individual features extracted, helping to better understand the discriminant features for the problem of detecting fake reviews. Secondly, the discriminant features found from the feature selection activity were used to form a number of feature sets, i.e., subsets of features, which were used to train separate models to be compared, with the end goal of creating a better performing and less complex classifier. These feature sets, which were informed by the results of the feature selection process are presented after outlining the feature selection approaches used.

Three main quantitative approaches were used for feature selection:

1. **Point-biserial correlation** – was used to assess the correlation of individual features and the label of a given review ("genuine" or "fake"). Point-biserial correlation is a type of correlation co-efficient which expresses the strength of association between a binary variable, in this case - the label of the review, and a continuous variable - the particular feature value.
2. **T-test** – was used to assess the significance of the difference between the mean values of the two groups (genuine and fake reviews) for a given feature.
3. **Decision tree feature importance** – was used to assess the importance of individual features when training the decision tree classifier on all features. Also called the Gini importance, it estimates how much each feature has reduced the Gini impurity in the decision tree (Pedregosa et al., 2011).

For implementing the above metrics, the package SciPy[7] was used for conducting the point-biserial correlation and t-tests (Virtanen et al., 2020). For retrieving the Gini importance of individual features, the feature_importances_ method from scikit-learn was used after training the decision tree classifier with all features. This was repeated 10 times to get the average feature importance of each feature. The implementation for the above feature selection methods can be seen in Appendix C.2.

The results of these three metrics for individual features are displayed in Figures 4.4, 4.5 and 4.6. The raw results can be seen in Appendix B. The two features *propernounProp* and *verbProp* features were disregarded because their p value for both the point-biserial correlation and the t-test was greater than the recommended threshold of $p < 0.05$. Interestingly, the point-biserial correlation and t-test results show an almost identical pattern for the features. A slightly different pattern emerges when plotting the Gini importance against each feature.

[7]https://www.scipy.org/

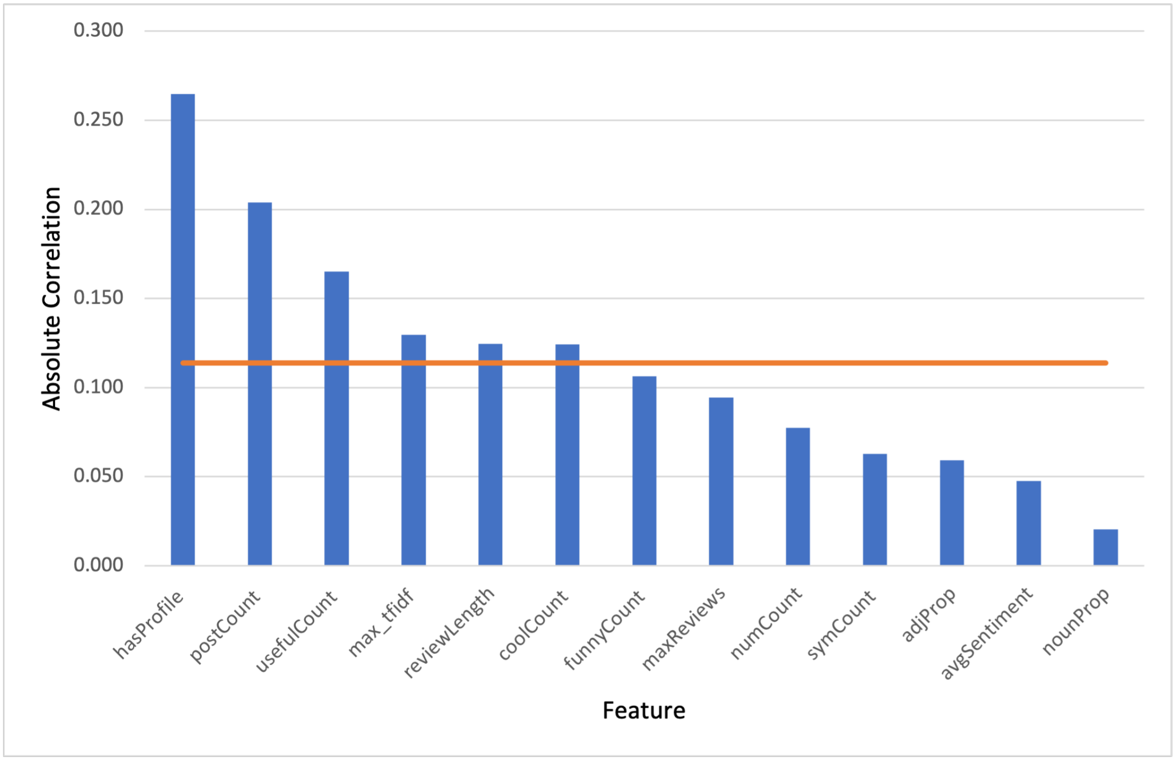


Figure 4.4: An illustration of the absolute correlation value plotted against each feature, with the orange line representing the average feature correlation.

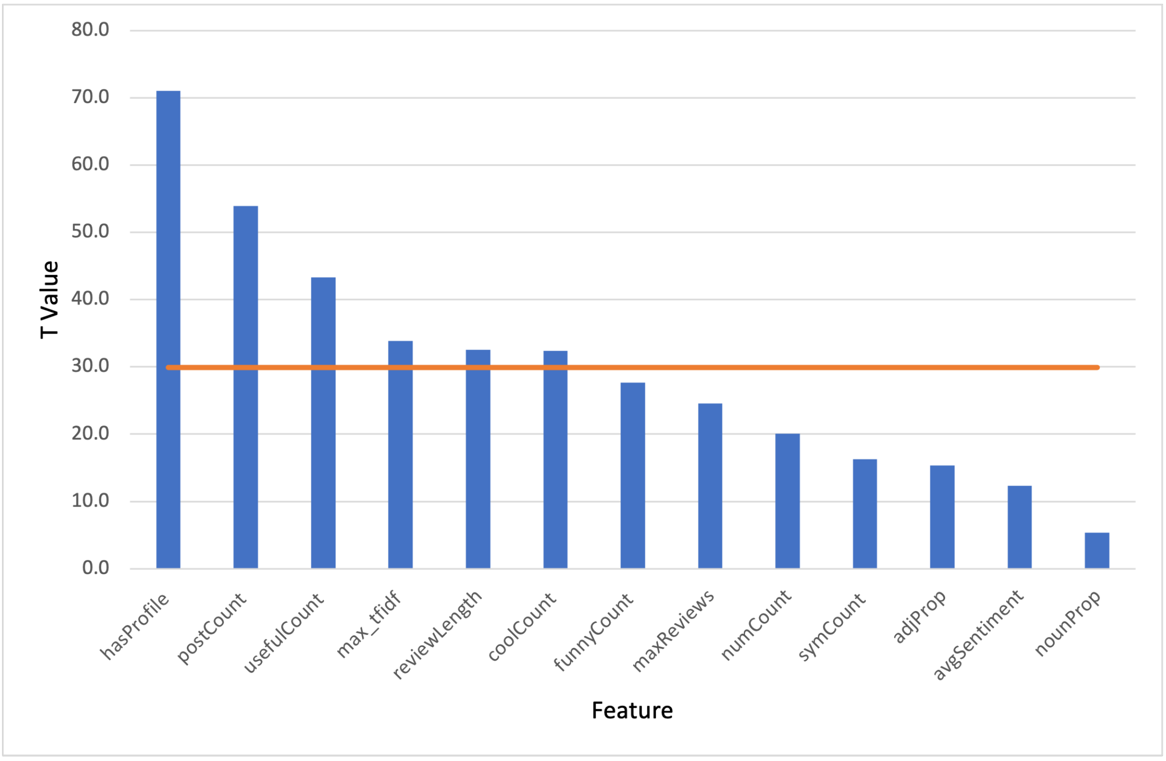


Figure 4.5: An illustration of the absolute T value plotted against each feature, with the orange line representing the average absolute T value.

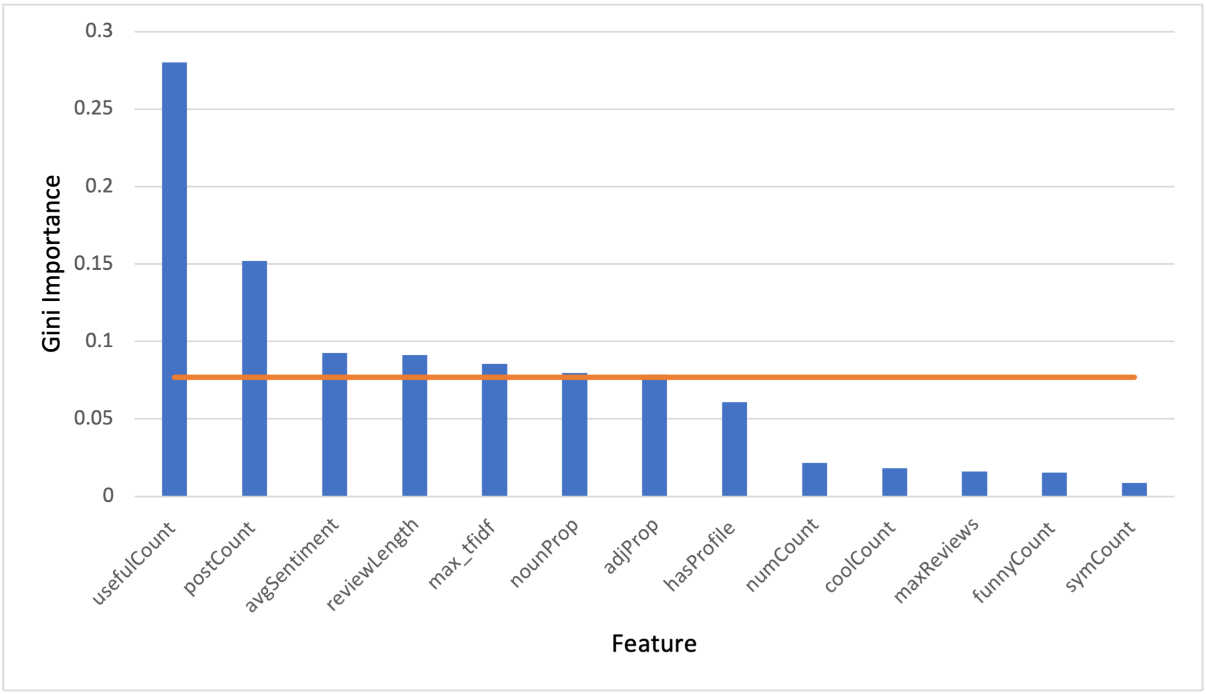


Figure 4.6: An illustration of the average Gini importance (stratified 10-fold CV, classifier trained on FS3) plotted against each feature, with the orange line representing the average Gini importance value.

Feature Sets

Outlined below are the seven feature sets (FS) created based on the results of the feature selection process. Feature sets 1 and 2 were created to contrast the performance of the classifiers trained on review-centric and reviewer-centric features only, respectively. The features contained in each feature set are outlined in Table 4.5.

1. **FS1** - review-centric features only.
2. **FS2** - reviewer-centric features only.
3. **FS3** - all features (except for where p-value > 0.05 in t-test).
4. **FS4** - top 10 features overall based on point-biserial correlation and t-test.
5. **FS5** - top 5 features overall based on on point-biserial correlation and t-test.
6. **FS6** - top 5 features overall based on feature importances of classifier trained with FS3.
7. **FS7** - top 10 features overall based on feature importances of classifier trained with FS3.

4.3.4 Model Evaluation

The decision tree classifier was evaluated using stratified 10-fold cross-validation (CV) as per Requirement 2.5. For training the model, a combination of review-centric and reviewer-centric features were used (Requirement 2.1) using the seven feature sets in

FS1	**FS2**	**FS3**	**FS4**	**FS5**	**FS6**	**FS7**
maxTfidf	hasProfile	maxTfidf	hasProfile	hasProfile	usefulCount	usefulCount
reviewLength	postCount	reviewLength	postCount	postCount	postCount	postCount
numCount	usefulCount	numCount	usefulCount	usefulCount	avgSentiment	avgSentiment
symCount	coolCount	symCount	maxTfidf	maxTfidf	reviewLength	reviewLength
adjProp	funnyCount	adjProp	reviewLength	reviewLength	maxTfidf	maxTfidf
avgSentiment	maxReviews	avgSentiment	coolCount			nounProp
nounProp		nounProp	funnyCount			adjProp
		hasProfile	maxReviews			hasProfile
		postCount	numCount			numCount
		usefulCount	symCount			coolCount
		coolCount				
		funnyCount				
		maxReviews				

Table 4.5: Feature Sets (FS) created based on feature selection process

Table 4.5. The evaluation metrics included accuracy, balanced accuracy, precision, recall and F1 score (Requirement 2.6). As per Requirement 2.7, the model had to be trained using a balanced class distribution and tested using a natural class distribution of genuine and fake reviews. Finally, the model's hyper-parameters needed to be tuned as stated in Requirement 2.8 to reduce overfitting, as otherwise the decision tree would grow fully until all leaf nodes were pure.

Resampling

To achieve a balanced class distribution of genuine and fake reviews for training the model, resampling techniques had to be applied to the training folds of data during each iteration of the CV. In this implementation, both oversampling and undersampling were used to balance the class distribution of genuine and fake reviews. For oversampling, each of the fake reviews in each fold were duplicated once. For undersampling, a random subset of genuine reviews of equal size to the oversampled fake review fold was selected. The function resample used for this can be seen in Listing 5.

Hyper-parameter tuning

The scikit-learn implementation of the decision tree grows fully if no hyper-parameters are specified in the initialisation of the decision tree instance. Due to a fully grown tree being likely to overfit to the training data and being less likely to perform well on unseen data, hyper-parameter tuning was implemented.

Specifically, a one-dimensional grid search approach was taken, where the optimal value for the max_depth parameter was found for each feature set. max_depth was selected as the particular hyper-parameter because it was thought to have the greatest impact on both the model's performance and the interpretability of the model. Essentially, this hyper-parameter limits the growth of the decision tree structure to a particular depth. Hence, in a single hyper-parameter, we both reduce the likelihood of the model overfitting and reduce the complexity of the model for the end user to interpret.

The grid search was done by testing a range of possible values for the hyper-parameter, from 3 to 25 and selecting the best performing (on average, during 10-fold CV) parameter value for each feature set. The performance metric used was the F1 score. The relevant code for this can be seen in Appendix C.3.

```python
def resample(x_train, y_train):

    # 1. "combine" x_train and y_train
    x_train = x_train.copy()
    x_train["label"] = y_train

    # create a sub-dataframes
    genuine = x_train[x_train["label"] == "N"]
    fake = x_train[x_train["label"] == "Y"]

    # 1. oversampling

    # 2x oversampling
    fake_copy = fake.copy()
    fake = pd.concat([fake, fake_copy])

    # 2. undersampling

    # shuffle rows
    genuine = genuine.reindex(np.random.permutation(genuine.index))
    # under-sample majority class
    genuine = genuine[:len(fake)]

    # combine two sub-dataframes
    x_train = pd.concat([genuine, fake])
    x_train = x_train.reset_index(drop = True)

    # shuffle rows
    x_train = x_train.reindex(np.random.permutation(x_train.index))

    # 3. "separate" x_train and y_train again
    y_train = pd.Series(x_train["label"])
    x_train = x_train.drop("label", 1)

    return x_train, y_train
```

Listing 5: Function for balancing review dataset using oversampling and undersampling

Recording results

Once the necessary functionality was implemented to carry out the model evaluation, the seven feature sets were individually run through the 10-fold CV script and the average score for each metric output by the script was recorded in a spreadsheet. This process was repeated three times for each of the following model configuration scenarios:

1. unbalanced dataset for training, no hyper-parameter tuning
2. balanced dataset for training, no hyper-parameter tuning
3. balanced dataset for training, hyper-parameter tuning applied

As will be noted in the next chapter, these 3 sets of models were evaluated to compare the relative performances of unbalanced vs. balanced training data, and compare the performance of a fully grown decision tree (i.e., no hyper-parameter tuning applied) to a decision tree which is not fully grown. The performances of these models are presented and compared in the next chapter.

4.4 Front End Implementation

Having completed the implementation of the review classifier involving the activities of data preparation, feature engineering and model evaluation, the next step was to create a transparent front end layer providing an explanation of the model's decision making to a naive end user (Project Aim 3). Specifically, the user would be interested in understanding the reasoning behind the binary output of the model ("fake" or "genuine") requiring the front end to provide an explanation of this output. The user would also potentially be interested in understanding the general, high-level decision-making of the model.

Based on the Front End and Transparency Requirements presented in Chapter 3, the implemented interface needed to provide both a natural language (text) explanation and visual depiction of the model's decision making. The former was achieved through analysing the decision path of a given review sample through the decision tree classifier and converting this to a series of decisions in text form (covered in Section 4.4.1). The latter was implemented in the form of a tree diagram, visualising the decision tree structure of the classifier with individual nodes and links between them, and the decision path taken by a given review sample through the structure (covered in Section 4.4.2).

To create a single interface, the natural language explanation of the decision path and the tree diagram were embedded in a single HTML document, and displayed in the user's web-browser (covered in Section 4.4.3), with the user being able to select the review to be classified through the user interface (covered in Section 4.4.4). This served as the key practical output for the project to showcase the capability of the system as a whole and illustrate the potential of a full-fledged version of the product.

4.4.1 Decision Path

The first critical component which was required for attaining an explanation of the model's decision making was to obtain the decision path of a given sample review when input to the classifier, that is, the list of nodes that the sample traversed and the associated decisions at each node, from the root node to the leaf node, leading to a classification of that review. This functionality set the foundation for presenting the decision making of the model in text form (Requirement 3.1) and visual form (Requirement 3.2).

The decision path is an inherently interpretable attribute of the decision tree classifier and its underlying structure. When a sample of interest is classified by a decision tree, the sample will start at the root node of the structure and traverse the tree based on the sample's feature values and the feature thresholds at each subsequent node until it arrives at a leaf node, which dictates the final classification of the sample. The final classification is based on the majority label ("genuine" or "fake") of training samples at the leaf node. If the leaf node only has training samples belonging to a single class label, the node is referred to as a pure leaf node.

To achieve the desired functionality of retrieving the decision path for a given review sample, the first task was to retrieve the list of nodes that a given sample traverses when input to the classifier. After obtaining this list of nodes, it would then be possible to inspect each node in terms of the feature it splits on and threshold value. Finally, by inspecting the leaf node (the last node that the sample traverses), specifically the proportion of class labels based on the training samples, it would be possible to retrieve the final classification decision ("genuine" or "fake").

The practical implementation of this in Python was heavily inspired by the scikit-learn documentation, in particular the decision_path method provided by the scikit-learn library[8] (Pedregosa et al., 2011). Applying this method on a trained classifier and a set of input samples, it returns a $i * j$ matrix where each sample is represented by the elements in i, and the nodes in the tree are represented by elements in j. If only a single sample is supplied to the method, as was done in this implementation (the user selects a single review to be classified and explained), the matrix returned is reduced to a row vector. As per the scikit-learn documentation, a non-zero element j in this row vector signifies that the given sample i has traversed node j, where j represents the node identifier (ID) in the decision tree structure of the classifier (Pedregosa et al., 2011). For example, if the tree structure contains 56 nodes, there will be a total of 56 columns in the row vector, with non-zero elements j representing the nodes the given sample has traversed. The method *apply* from the scikit-learn library was used to retrieve the ID of the leaf node that the sample ended its decision path on.

After obtaining the row vector highlighting the nodes traversed by a given review sample provided by the decision_path method, it was then possible to inspect each of these nodes traversed for their feature and threshold values to create a list of features used in the model's decision making for the given review. Similarly, by inspecting the leaf node provided by the *apply* method it was possible to retrieve the classification decision by inspecting the proportion of class labels at that node. Listing 6 presents the implementation of the function get_decision_path_info utilising the decision_path (line 35) and *apply* (37) methods in Python. This function takes three arguments: the decision tree classifier, the feature names and the feature values of the sample to be classified. The implemented function returns a list of all features considered by the classifier at each node in the decision path (the feature that each node is split on), two lists of corresponding thresholds for each node split, consisting of the threshold sign and value (e.g., "<=" and "4"), and a string summarising the classification decision including the associated probabilities based on the proportion of class labels at the leaf node.

Having a list of the features considered in the decision path and the associated value thresholds, the next step was to create a series of written statements which could be displayed in the user interface as a user-friendly, natural language (text) explanation of the decision path. In Appendix C.5 lines 87-149 the function write_decisions can be seen which fulfilled this purpose. This function was added to the second version of the user interface described later in this section, one of the major capabilities of this being summarising multiple nodes in the decision path being split on the same feature. Rather than presenting a list of all nodes and the decisions at each as done in the first version of the user interface, this function summarises the decisions to make them more readable and appear less arbitrary to the end user. Another key capability of this function was to round up threshold values that were arbitrarily set as floats by the classifier but are always integer values, e.g., the length of a review in characters can never be a float value. This was also done to make the explanation more readable and clear to the user.

[8]https://scikit-learn.org/stable/auto_examples/tree/plot_unveil_tree_structure.html#decision-path

```python
def get_decision_path_info(classifier, feature_names, sample):

    # retrieve probabilities for classification for sample
    probabilities = classifier.predict_proba(sample)
    # retrieve decision paths for test sample
    node_indicator = classifier.decision_path(sample)
    # retrieve leaf IDs reached by test sample
    leaf_identifier = classifier.apply(sample)
    # get a list of all thresholds (of all nodes in tree)
    threshold = classifier.tree_.threshold
    # get node id of leaf that the sample "lands on"
    leaf_id = leaf_identifier[0]
    # obtain ids of the nodes sample goes through
    nodes_traversed = node_indicator.indices[node_indicator.indptr[0]:
                                    node_indicator.indptr[1]]

    features_used = []
    threshold_values = []
    threshold_signs = []

    for node_id in nodes_traversed:

        # break for loop
        if(node_id == leaf_id):
            break

        # check if value of the split feature for sample i is below threshold
        if (sample.iloc[0][classifier.tree_.feature[node_id]] <= threshold[node_id]):
            threshold_sign = "<="
        else:
            threshold_sign = ">"

        features_used.append(feature_names[classifier.tree_.feature[node_id]])
        #thresholds.append(threshold_sign + " " + str(round(threshold[node_id], 3)))
        threshold_values.append(round(threshold[node_id], 3))
        threshold_signs.append(threshold_sign)

    # get proportions of labels at leaf node and calculate probabilities of the output
    class_proportion = classifier.tree_.value[leaf_id]
    class_probability = round(100 * max(probabilities[0]), 1)
    samples_at_leaf = int(class_proportion[0][0]) + int(class_proportion[0][1])
    genuine_probability = round(100 * probabilities[0][0], 1)
    fake_probability = round(100 * probabilities[0][1], 1)

    predicted_string = "According to the model, this review is likely to be Genuine " \
    "with probability {prob1}% and Fake with probability {prob2}%. " \
    .format(prob1 = genuine_probability, prob2 = fake_probability)
    predicted_string += "These probabilities are based on " + str(samples_at_leaf) + \
    " other reviews with the same decision path."

    return features_used, threshold_values, threshold_signs, predicted_string
```

Listing 6: Retrieving the decision path information for a review sample

4.4.2 Tree Diagram

After the implementation of the decision path functionality and the natural language explanation of decisions made by the model for a given review sample, the next component implemented was the tree diagram, serving as a visual depiction of the model's decision making to satisfy Requirement 3.2.

Scikit-learn and pydotplus libraries were used to achieve the required functionality. Pydotplus[9] is a package that provides an interface to the DOT language, which is a graph depiction language. The function export_graphviz from scikit-learn was used to create a DOT format representation of the decision tree structure of the classifier, which was then used by pydotplus to modify the visual representation of the graph.

In particular, pydotplus was used to create a graph of the DOT format representation, the nodes of which were then modified to visually distinguish the nodes with the majority label being "genuine" and those with the majority label being "fake", and also highlight the decision path in a separate colour. The decision_path method from scikit-learn was used again to retrieve the decision path for the given sample. After retrieving the decision path, the nodes part of the decision path were coloured in a separate colour using pydotplus. The colours were specified using the DOT colour names outlined by Graphviz for the DOT language[10]. The specific colours were picked to make the decision path stand out to the user from the rest of the nodes, whilst also visually distinguishing other nodes which have their majority label/prediction as "genuine" or "fake". Hence, two shades of grey were used for the latter task. Finally, pydotplus was used to save the modified graph as a png image file, to be later embedded together with the natural language explanation in an HTML document.

Listing 7 displays the code that was used to achieve the described functionality. This code is from the second version of the user interface but the general functionality is the same for both versions.

4.4.3 HTML Wrapper

Having implemented the core functionality to illustrate the classifier's decision making in text and visual form, the next step was to integrate these components and display them to the user in an interface. For this task, Turkel and Crymble (2012) served as inspiration to integrate the decision path and tree diagram in an HTML document and display the document in the user's web browser by embedding (wrapping) the text output and tree diagram in an HTML file.

Key to the implementation of the HTML wrapper functionality was string formatting, which allowed the decision path text and tree visualisation file location to be embedded dynamically at the appropriate positions in the HTML file. This functionality can be seen in the function wrap_output_in_HTML in Listing 8. In line 286 it can be seen how string formatting is used to wrap the title, the current date-time, the decision path explanation and the file location of the tree diagram in the string wrapper, which contains the HTML document structure. Prior to embedding the decision path explanation, a function was implemented to prepare the string for embedding by formatting it correctly.

[9]https://pydotplus.readthedocs.io/
[10]https://www.graphviz.org/doc/info/colors.html

```python
def create_tree_diagram_png(dt, feature_names, class_names, filename, sample):

    # get decision tree classifier in DOT (graphviz) format
    dot_data = tree.export_graphviz(dt, out_file=None,
                                feature_names=feature_names,
                                    class_names=class_names, filled=True,
                                    rounded=True, special_characters=True,
                                    node_ids = False)

    graph = pydotplus.graph_from_dot_data(dot_data)

    # set all nodes to be of a white colour
    for node in graph.get_node_list():
        node.set_fillcolor('white')

    # set colour of leaf nodes depending on their class (fake or genuine)
    for node in graph.get_node_list():
        d = node.get_attributes()
        if("label" in d):
            labels = node.get_attributes()['label'].split('<br/>')
            for i, label in enumerate(labels):
                if(label.startswith('class = ')):
                    # colour nodes depending on their class
                    if('Fake' in label):
                        # set colour of nodes with fake as majority vote
                        node.set_fillcolor('gray75')
                    elif('Genuine' in label):
                        # set colour of nodes with genuine as majority vote
                        node.set_fillcolor('gray97')

    # get decision path for chosen review sample
    decision_path = dt.decision_path(sample)

    # loop through each node in tree and its associated decision-path value,
    # i.e., if the node is part of the decision path
    # then it has a value of 1, otherwise, it has a value of 0
    for node_counter, node_value in enumerate(decision_path.toarray()[0]):

        # if node is not part of decision path for sample, continue
        if(node_value == 0):
            continue
        # set colour of node part of decision path for sample
        dot_node = graph.get_node(str(node_counter))[0]
        dot_node.set_fillcolor('darkseagreen1')

    # export updated graph to a png
    graph.write_png(filename)
```

Listing 7: Creating the tree diagram of the decision tree classifier with the decision path highlighted

This included steps such as creating a bulleted list of the decisions made by the model and creating a header to serve as the title of the document. These steps can be seen in the prepare_string_to_wrap function in Appendices C.4 and C.5.

After the creation of this HTML document, the functionality to open it in the user's web browser was provided by the webbrowser[11] module in Python by providing the path of the saved HMTL document to the open_new_tab method.

4.4.4 User Interface

A single user interface was implemented by combining the previously outlined elements in this section, i.e., the decision path functionality, tree diagram and the HTML wrapping functionality. Along with these elements, functionality to get user input from the keyboard was added to satisfy Requirement 3.3 - the user being able to select a review to be classified and explained to them. Two versions of the user interface were created, with the first version providing the core functionality of the aforementioned components, with the second version building on the usability of the first version to make it more transparent for the end user.

The following is a summary of the steps taken by the code to achieve the functionality of the user interface (both versions) after the user has selected a review to classify:

1. Retrieve review chosen by user from test data and its associated review information
2. Get decision path information for chosen review and write as a series of decisions (see Section 4.4.1)
3. Generate tree diagram with the decision path highlighted (see Section 4.4.2)
4. Prepare text explanation and tree diagram image to wrap in HTML document (see Section 4.4.3)
5. Embed the text explanation and tree visualisation in HTML document and open in user's web browser (see Section 4.4.3)

The above steps can also be seen in code form in the main user input loop in Appendix C.5, lines 329-371.

User Interface v1.0

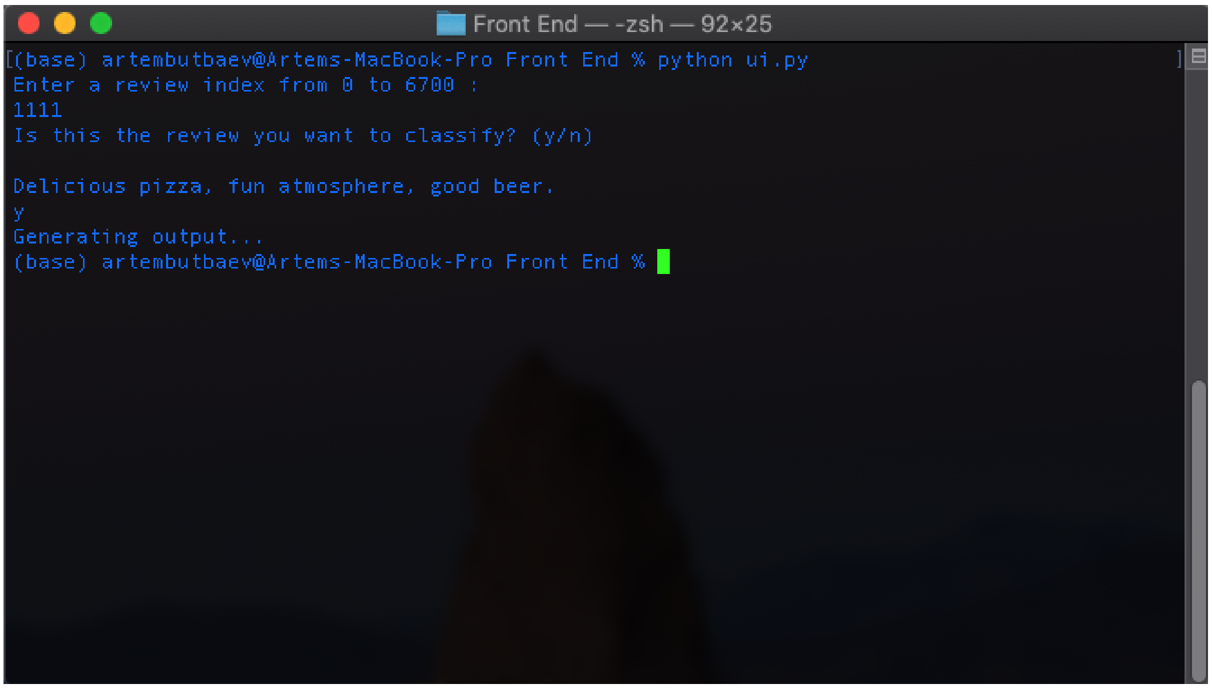
The main goal of the first version of the user interface was to create the functionality for user input (to select a review to be classified) and to implement the basic functionality to achieve the list of five steps outlined above. This was done by integrating the individual components of the decision path function, the tree diagram function, and the HTML wrapper functionality in a single script.

Functionality to allow the user to select a review was added using the native **input**() function in Python which reads a line from input and returns it as a string. As can be seen in Figure 4.7, when the user runs the user interface script from the command line/terminal, they are prompted to select a review by its index in the testing data. In this case, they can select from 6700 reviews, which is the size of the testing data, i.e.,

[11]https://docs.python.org/3/library/webbrowser.html

```python
def wrap_output_in_HTML(title, text, image):

    current_datetime = datetime.datetime.today().strftime("%d/%m/%Y - %H:%M:%S")

    # create a new HTML file
    filename = title + '.html'
    f = open(filename,'w')

    # create basic HTML structure with indications
    # of where strings will be embedded
    wrapper = """<html>
    <head>
    <style>
    body {
        background-color: WhiteSmoke;
    }
    </style>
    <title>%s - %s</title>
    </head>
    <body>

    <p>%s</p>
    <a href=%s target="_blank">Click here to open visualisation</a>
    </body>
    </html>"""

    # embed the document title, current datetime, decision path text
    # and tree visualisation file location in the HTML document
    whole = wrapper % (title, current_datetime, text, image)
    f.write(whole)
    f.close()

    path = "file://" + str(Path().absolute()) + "/%s"
    path = path % (filename)

    return path
```

Listing 8: Wrapping the decision path explanation and tree diagram in an HTML file



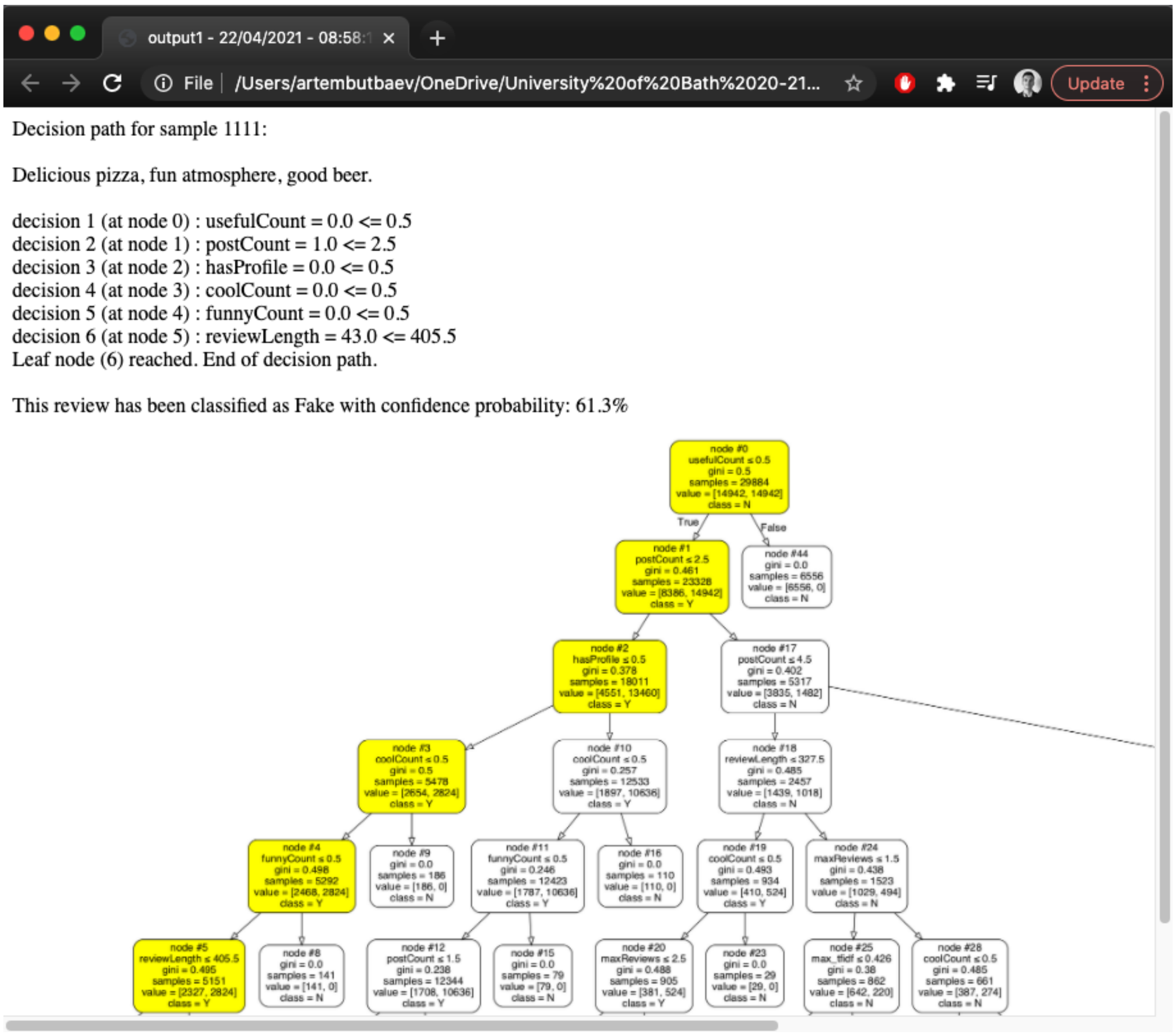
```
[(base) artembutbaev@Artems-MacBook-Pro Front End % python ui.py
Enter a review index from 0 to 6700 :
1111
Is this the review you want to classify? (y/n)

Delicious pizza, fun atmosphere, good beer.
y
Generating output...
(base) artembutbaev@Artems-MacBook-Pro Front End %
```

Figure 4.7: User Interface v1.0/v2.0 - user input

data the classifier has not been trained on and has not seen before. Once the user has entered an index, the review is shown to the user and they are asked if they would like this review classified. If the user enters anything else than "y", the prompt to select a review index appears again and the user can choose another index. Otherwise, the user interface generates the HTML file output and displays the user interface main screen in the user's browser (Figure 4.8 (v1.0), Figure 4.9 (v2.0)).

User Interface v2.0

The second version of the user interface used the basic functionality of the first version and made the interface more transparent for the end user, with updates based on feedback received from the project supervisor and further research by consulting the transparency literature. These updates can be seen in Figures 4.9 and 4.10, which display screenshots of the user interface. The same example review is used as for the previously shown screenshots for the first version of the user interface (Figure 4.8).


output1 - 22/04/2021 - 08:58:1
File | /Users/artembutbaev/OneDrive/University%20of%20Bath%2020-21...
Update
node #0 usefulCount ≤ 0.5 gini = 0.5 samples = 29884 value = [14942, 14942] class = N
True
False
node #1 postCount ≤ 2.5 gini = 0.461 samples = 23328 value = [8386, 14942] class = Y
node #44 gini = 0.0 samples = 6556 value = [6556, 0] class = N
node #2 hasProfile ≤ 0.5 gini = 0.378 samples = 18011 value = [4551, 13460] class = Y
node #17 postCount ≤ 4.5 gini = 0.402 samples = 5317 value = [3835, 1482] class = N
node #3 coolCount ≤ 0.5 gini = 0.5 samples = 5478 value = [2654, 2824] class = Y
node #10 coolCount ≤ 0.5 gini = 0.257 samples = 12533 value = [1897, 10636] class = Y
node #18 reviewLength ≤ 327.5 gini = 0.485 samples = 2457 value = [1439, 1018] class = N
node #4 funnyCount ≤ 0.5 gini = 0.498 samples = 5292 value = [2468, 2824] class = Y
node #9 gini = 0.0 samples = 186 value = [186, 0] class = N
node #11 funnyCount ≤ 0.5 gini = 0.246 samples = 12423 value = [1787, 10636] class = Y
node #16 gini = 0.0 samples = 110 value = [110, 0] class = N
node #19 coolCount ≤ 0.5 gini = 0.493 samples = 934 value = [410, 524] class = Y
node #24 maxReviews ≤ 1.5 gini = 0.438 samples = 1523 value = [1029, 494] class = N
node #5 reviewLength ≤ 405.5 gini = 0.495 samples = 5151 value = [2327, 2824] class = Y
node #8 gini = 0.0 samples = 141 value = [141, 0] class = N
node #12 postCount ≤ 1.5 gini = 0.238 samples = 12344 value = [1708, 10636] class = Y
node #15 gini = 0.0 samples = 79 value = [79, 0] class = N
node #20 maxReviews ≤ 2.5 gini = 0.488 samples = 905 value = [381, 524] class = Y
node #23 gini = 0.0 samples = 29 value = [29, 0] class = N
node #25 max_tfidf ≤ 0.426 gini = 0.38 samples = 862 value = [642, 220] class = N
node #28 coolCount ≤ 0.5 gini = 0.485 samples = 661 value = [387, 274] class = N

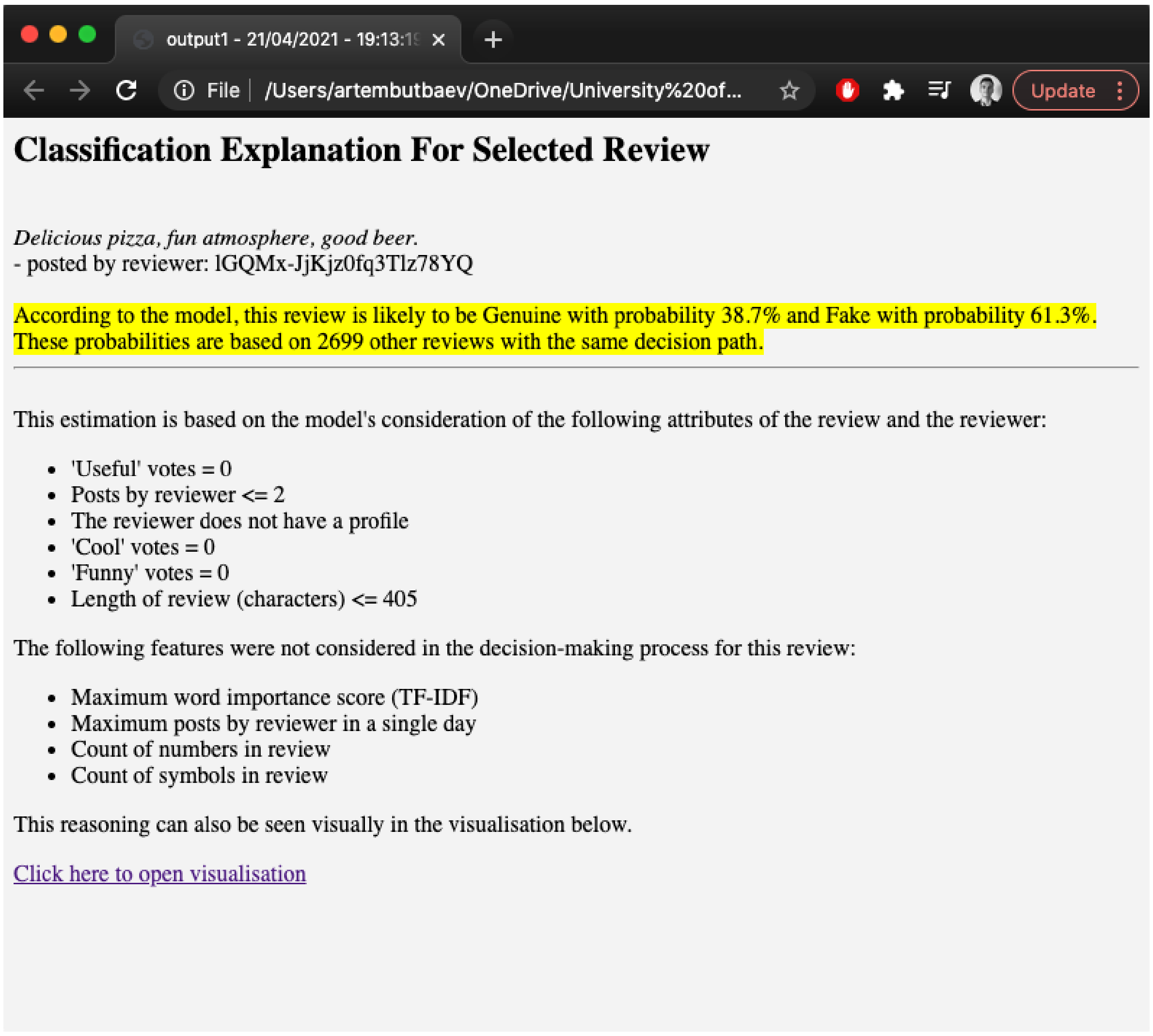
Decision path for sample 1111:

Delicious pizza, fun atmosphere, good beer.

```
decision 1 (at node 0) : usefulCount = 0.0 <= 0.5
decision 2 (at node 1) : postCount = 1.0 <= 2.5
decision 3 (at node 2) : hasProfile = 0.0 <= 0.5
decision 4 (at node 3) : coolCount = 0.0 <= 0.5
decision 5 (at node 4) : funnyCount = 0.0 <= 0.5
decision 6 (at node 5) : reviewLength = 43.0 <= 405.5
Leaf node (6) reached. End of decision path.
```

This review has been classified as Fake with confidence probability: 61.3%

Figure 4.8: User Interface v1.0 - main screen



output1 - 21/04/2021 - 19:13:1

File | /Users/artembutbaev/OneDrive/University%20of...

Update

Classification Explanation For Selected Review

Delicious pizza, fun atmosphere, good beer.
- posted by reviewer: lGQMx-JjKjz0fq3Tlz78YQ

According to the model, this review is likely to be Genuine with probability 38.7% and Fake with probability 61.3%. These probabilities are based on 2699 other reviews with the same decision path.

This estimation is based on the model's consideration of the following attributes of the review and the reviewer:

- 'Useful' votes = 0
- Posts by reviewer <= 2
- The reviewer does not have a profile
- 'Cool' votes = 0
- 'Funny' votes = 0
- Length of review (characters) <= 405

The following features were not considered in the decision-making process for this review:

- Maximum word importance score (TF-IDF)
- Maximum posts by reviewer in a single day
- Count of numbers in review
- Count of symbols in review

This reasoning can also be seen visually in the visualisation below.

Click here to open visualisation

Figure 4.9: User Interface v2.0 - main screen

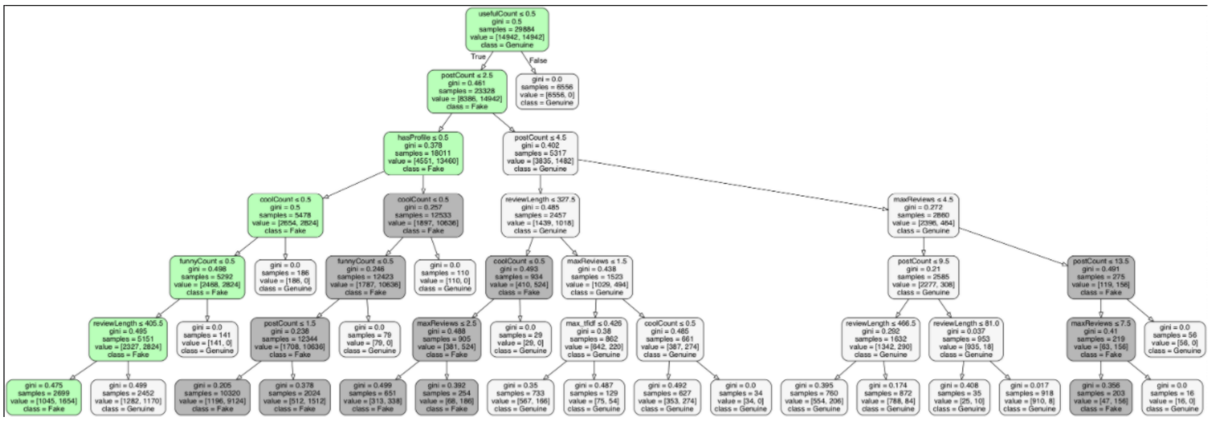


Figure 4.10: User Interface v2.0 - tree visualisation. The green nodes are those part of the decision path, the dark grey nodes are those with "Fake" as the classification decision, and the light grey nodes have "Genuine" as the classification decision

The following is a list of the updates included in the second version of the user interface:

- Multiple nodes which are split using the same feature in the decision path are summarised to a single decision in the explanation
- For features of type integer the decision threshold values are rounded, e.g., posts by reviewer feature, which cannot be a non-integer value
- Updating feature names to be more readable and understandable to the end user, rather than using the raw feature names
- Including an alternative decision (the probability of the opposite classification is shown) to transfer accountability to the user
- Including information about features not used in the model's decision making
- Creating a hyperlink to the tree visualisation, rather than having it on the same page as the text explanation - removes the need for scrolling (see Figure 4.8) and does not overwhelm the user with too much information at once
- Updated tree visualisation with nodes highlighted to show the classification decision at each node (i.e., the majority label)
- Added reviewer information (reviewer ID)
- Added information on how many training samples the classification decision was based on
- Aesthetic updates including highlighting the model prediction in a yellow colour, changing the background, updating page layout, adding a title

Going through the screenshot example of the second version of the user interface in Figure 4.9, the user can see the review they selected and the reviewer who posted it at the top of the page. Highlighted in yellow they read the model's prediction and the number of previous review examples (training data points) with the same decision path. Below the prediction, the individual decisions which led to this classification are outlined, as well as the features that were not considered by the model in its decision making for this particular review. The user may click on the hyperlink "Click here to open visualisation", which opens the tree visualisation shown in Figure 4.10 in a separate tab. This visualises the decision path by the given review sample as it moves through the nodes in the decision tree structure in a green colour. The light grey and dark grey nodes represent the nodes with the majority classification being "Genuine" and "Fake", respectively.

4.5 Summary

This chapter has outlined in detail the implemented system as part of this dissertation to satisfy the relevant requirements from the preceding chapter. The system presented consists of two key components: the fake review classifier and the transparent front end layer which explains the output of the model to an end user. This chapter has also presented the technology stack used for development and an overview of the system as a whole. In the next chapter, the performance of the classifier and the level of transparency offered by the user interface will be investigated.

Chapter 5

Results and Analysis

This chapter outlines and discusses the performance of both the implemented decision tree classifier and the transparency of the user interface developed. The results are presented to the reader and then interpreted and examined in more detail.

5.1 Model Performance

5.1.1 Model Evaluation Process

The decision tree classifier was evaluated by running the evaluation code implemented based on the relevant requirements, as outlined in the preceding chapter in Section 4.3.4, with the raw results being recorded in Tables A.1, A.2 and A.3.

As a quick summary of Section 4.3.4, the evaluation code consisted of 10-fold cross-validation (CV), which was repeated three times, once each for the following scenarios:

1. **Model 1** - unbalanced dataset for training, no hyper-parameter tuning
2. **Model 2** - balanced dataset for training, no hyper-parameter tuning
3. **Model 3** - balanced dataset for training, hyper-parameter tuning applied

These three sets of models were tested and compared, firstly to illustrate the problem of an imbalanced dataset. Secondly, they were used to show that Models 1 and 2 overfit to the training data because the growth of the decision trees is not limited as no hyper-parameter tuning is applied.

Each of these three sets of models were evaluated for the 7 feature sets outlined in Section 4.3.3 in pursuit of the best performing feature set and classifier to use in the explainable interface.

Model performance was evaluated using five evaluation metrics as per Requirement 2.6: accuracy, balanced accuracy, precision, recall, and F1 score (also called F score), which are defined formally in Section 2.2.4 of the Literature and Technology Review. All of the measures range from 0–100. For all classifiers evaluated, the testing data was unmodified throughout, with a natural class distribution of genuine and fake reviews to simulate a real-world set of reviews as much as possible, as per Requirement 2.7.

For the reader's convenience, the 7 feature sets (FS) evaluated are listed again below (outlined in more detail in Section 4.3.3 on Feature Selection in the preceding chapter):

1. **FS1** - review-centric features only.
2. **FS2** - reviewer-centric features only.
3. **FS3** - all features (except for where p-value > 0.05 in t-test).
4. **FS4** - top 10 features overall based on point-biserial correlation and t-test.
5. **FS5** - top 5 features overall based on on point-biserial correlation and t-test.
6. **FS6** - top 5 features overall based on feature importances of classifier trained with FS3.
7. **FS7** - top 10 features overall based on feature importances of classifier trained with FS3.

5.1.2 Performance Overview

Figures 5.1, 5.2, 5.3, 5.4 and 5.5 each display the performance of the decision tree classifiers for each of the five metrics, by the model set and the feature set. Below we highlight and discuss the patterns seen in these figures for each performance metric.

Accuracy

Looking at Figure 5.1, we can see that there is a general pattern (for all feature sets) of the accuracy being at its highest for model set 1, with model sets 2 and 3 achieving lower accuracies. For example, for Feature Set 1 (FS1), there is a drop in accuracy from around 78% for model 1 to 65% and 60% for models 2 and 3, respectively. Although the drop in accuracy is not as pronounced for the other feature sets, it is still noticeable.

Although these results could be concerning at first glance, this performance metric does not tell the full story, due to the imbalanced training and testing data used for model set 1. As the accuracy measure does not distinguish between the model's ability to detect genuine reviews and detect fake reviews (recall), model 1 can be good at detecting genuine reviews but poor at detecting fake reviews without the accuracy suffering, due to the fact that there is a much greater number of genuine reviews compared to fake reviews, as explained in Section 2.2.4 in the Literature and Technology Review. Looking ahead to Figure 5.3, this is exactly what we see, with FS1 achieving a recall of only 18% for model 1, and none of the feature sets achieving a greater recall than 62% for model 1.

Balanced Accuracy

Due to the highly imbalanced nature of the testing dataset, a more meaningful picture of the models' accuracies is portrayed by the balanced accuracy performance metric. Compared to the accuracy metric discussed above, Figure 5.2 portrays a more reassuring pattern, as there is a general increase in balanced accuracy for models 2 and 3, compared to model 1, with the balanced accuracy reaching around 80% for feature sets 2-7 for model 3.

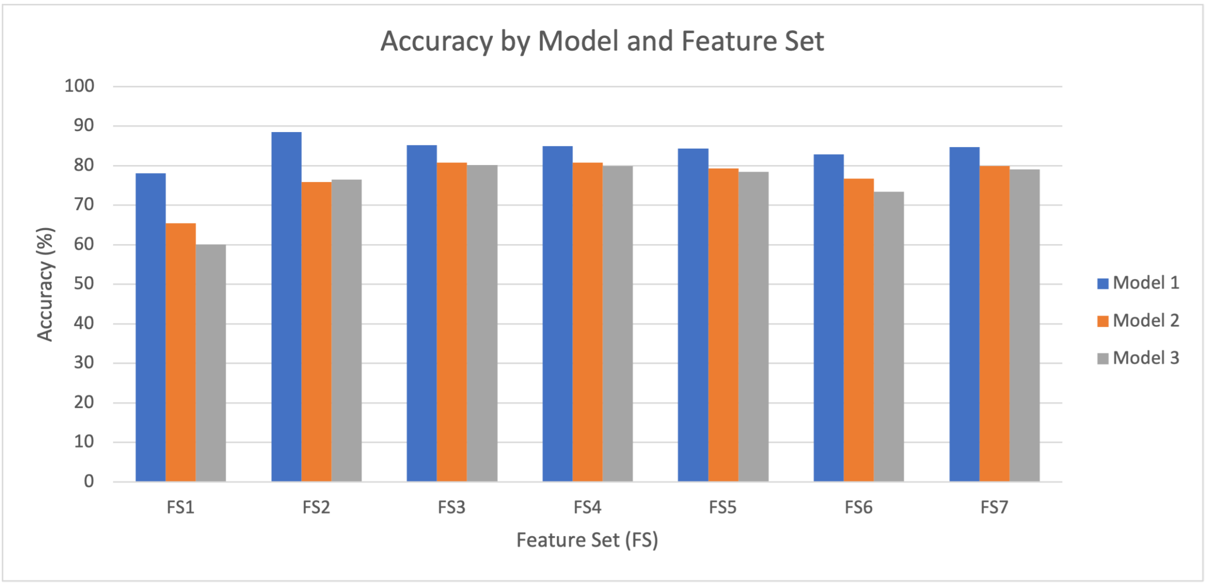


Figure 5.1: Performance of each feature set by Accuracy (A) for the three sets of models

Arguably, one factor leading to this increase is due to the training dataset being balanced for models 2 and 3, which reduces the classifier's bias in detecting reviews based mainly on the majority of training samples being genuine in the imbalanced dataset. Intuitively, with a balanced training dataset the model becomes exposed to an equal number of genuine and fake review samples and therefore it does not have this bias, becoming better at detecting fake reviews.

Another factor which arguably leads to the general increase in balanced accuracy from model 2 to 3 is the hyper-tuning applied to the classifier. Whereas the decision tree grows fully for models 1 and 2, model 3 has the hyper-parameter of max_depth applied which limits the growth of the tree and makes it less susceptible to overfitting to the training data compared to models 1 and 2. As the testing data used has not been previously seen by the model, these results indicate that the model hyper-tuning benefits the balanced accuracy of the classifier.

Recall

Recall assesses the model's ability of detecting fake reviews as it calculates the proportion of fake reviews that are correctly identified. Similarly to the reassuring pattern seen for balanced accuracy, Figure 5.3 shows a clear pattern of a significant increase in recall for models 2 and 3 for all feature sets, with feature sets 2-7 reaching 85%+ recall for model 3. As seen with the balanced accuracy metric, as the training dataset is balanced and hyper-parameter tuning is applied to reduce overfitting, the classifiers perform significantly better at detecting fake reviews. There is one outlier in Figure 5.3 for FS2, where there is no improvement from model 2 to model 3, which there is for all other feature sets, which is likely due to model 2 with FS2 leading to a simpler decision tree structure, even with no hyper-parameter tuning applied.

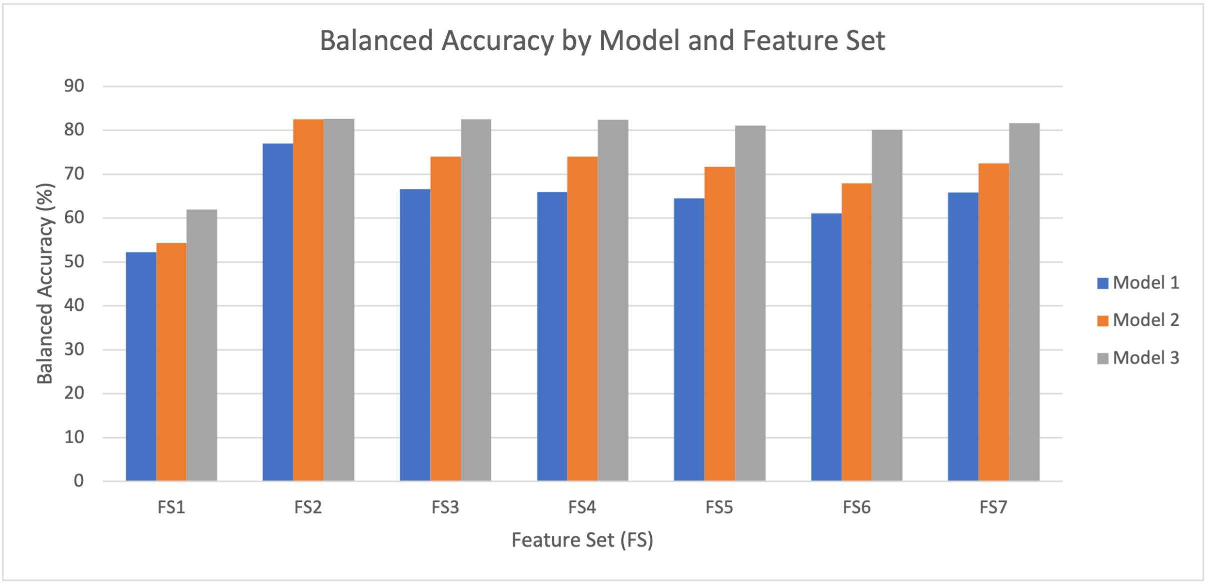


Figure 5.2: Performance of each feature set by Balanced Accuracy (BA) for the three sets of models

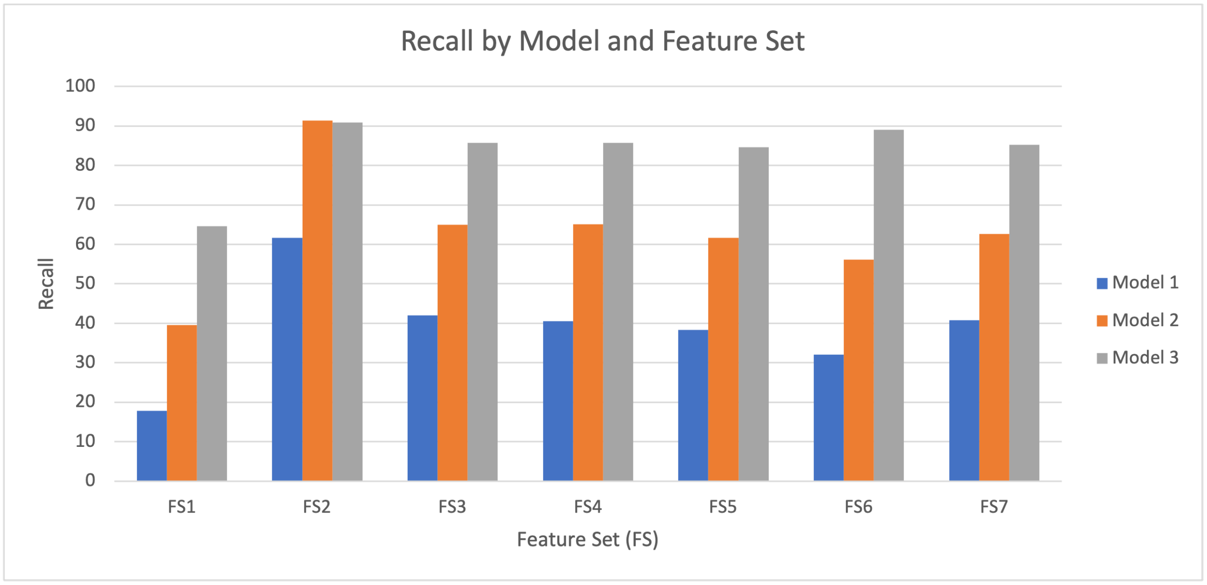


Figure 5.3: Performance of each feature set by Recall (R) for the three sets of models

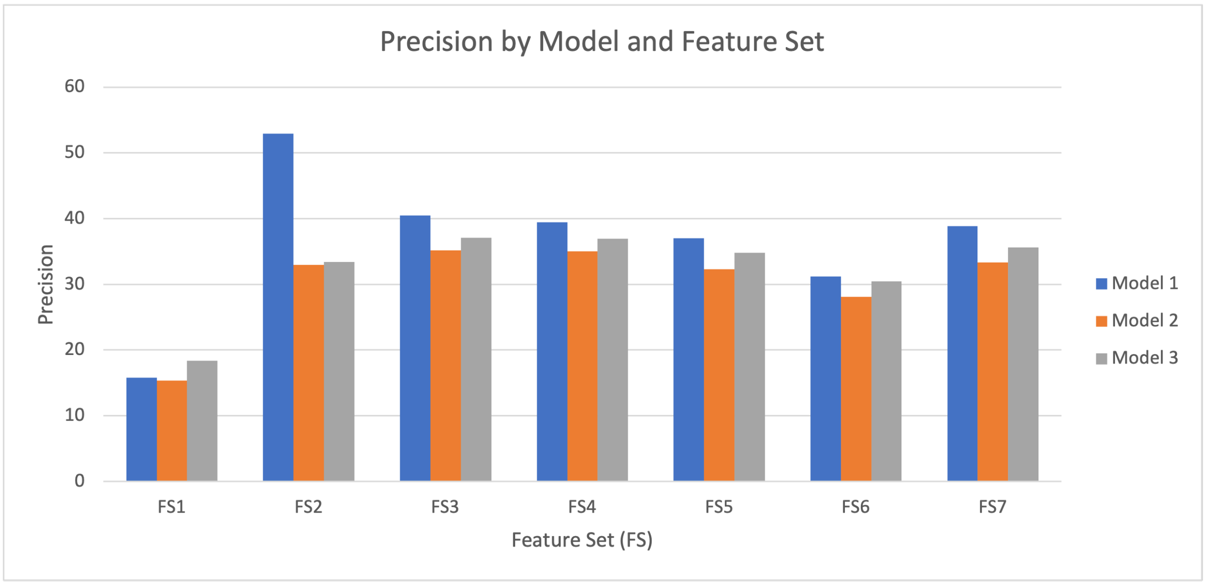


Figure 5.4: Performance of each feature set by Precision (P) for the three sets of models

Precision

Precision, also called the Positive Predictive Value (PPV), measures the proportion of true positives of the total number of true positives and false positives. In other words, it is the share of truly fake reviews from the reviews predicted as fake. Figure 5.4 displays the results for this metric. Although not a very reassuring picture at first glance, the precision is very similar to that achieved by model 1, for all feature sets except for FS2. Arguably, the relatively poor performance of this metric is due to the highly imbalanced testing data, where approximately 9 in 10 reviews are genuine for the dataset used. Therefore, there is arguably a higher chance of false positives for models 2 and 3 as they are trained on a balanced dataset of genuine and fake reviews, whereas model 1 arguably performs better in general in this metric because it is trained on an imbalanced dataset, and thus the classifier learns the pattern of the skewed relative proportion of genuine and fake reviews, as also discussed in Section 2.2.4 of the Literature and Technology Survey. Arguably, the false positives are worth the much higher recall of models 2 and 3.

F score

The F score (or F1 score) is calculated from the precision and recall of the given model, as the harmonic mean of the two measures. The general pattern for this measure is also reassuring (see Figure 5.5), with the F-score increasing from model 1 to model 2 to model 3, as the training data is balanced and hyper-parameter tuning is applied to the classifier. The only outlier, as was the case for precision, is model 1 trained with FS2, which has the highest F-score of all model and feature set combinations. However, looking at the recall and precision for this particular model, it is clear why this occurs: model 1 trained with FS2 has a recall of 62% and a precision of 53%, leading to a harmonic mean of 57%. It is clear in this case that the high precision in particular brings up the F-score for this model, as the recall is significantly lower than that achieved by all model 3 classifiers. In the case of fake review detection, as argued above for precision, a higher level of recall is

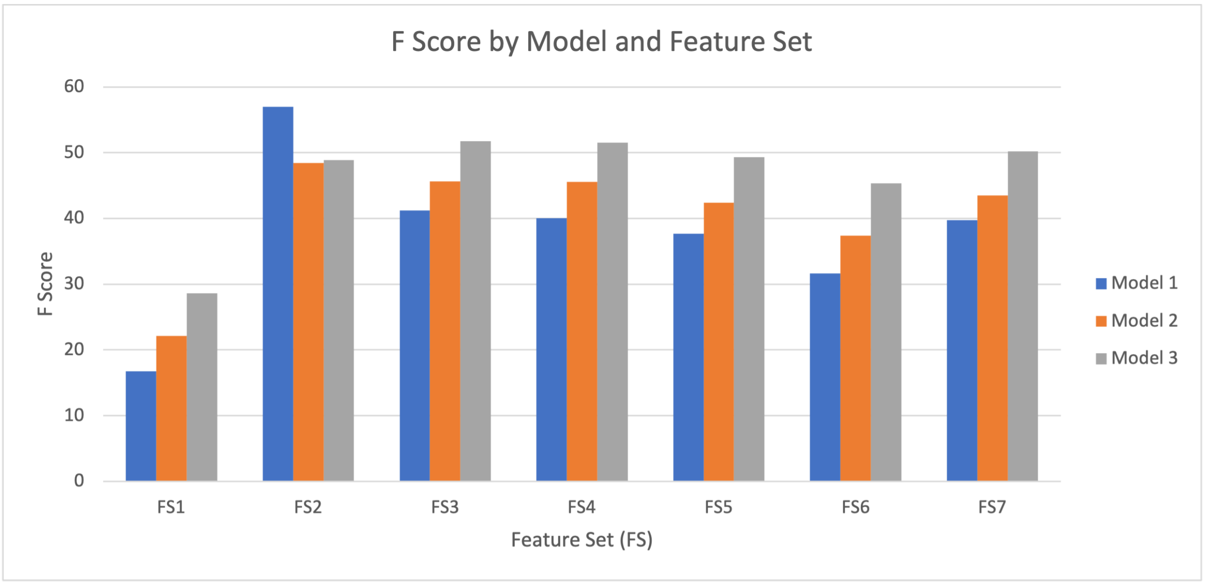


Figure 5.5: Performance of each feature set by F score (F) for the three sets of models

arguably more important than a higher F1 score.

Selecting the Best Classifier

Having evaluated the performance of the different models and feature sets for the five evaluation metrics, the next step was to select the best-performing feature set to be used in the user interface. The best-performing classifier was selected from model set 3 by considering two factors: the total score for all metrics, as per Figure 5.6, and a consideration of the optimal max_depth value for the hyper-parameter. As can be seen in the figure, FS3 and FS4 were the two best performing feature sets by a small margin, and based on the fact that the optimal max_depth value for FS3 was 7 and for FS4 it was 6, the latter was picked for its simpler model with nearly equivalent performance.

Arguably, however, it would not have made a large difference to the overall performance of the classifier if a different feature set was picked (except FS1), as all feature sets except for FS1 achieved similar results during stratified 10-fold CV.

The classifier evaluated with the best performance by a very small margin across all performance metrics was Model 3 trained on Feature Set 3 (FS3) with an average accuracy of 80.16, precision of 37.10, recall of 85.76 and F score of 51.75.

5.1.3 Comparison to Literature

Having presented an overview of the evaluated classifiers' performance achieved for the five evaluation metrics, the next step was to compare these results to those found in the fake review detection literature.

A clear point of variability between the results reported in this project and the results surveyed in the literature is the class distribution of the testing data, which has a significant effect on the evaluation metrics. Whereas a natural (unbalanced) class distribution of genuine/fake reviews was used for this project, it seems that the majority of the papers

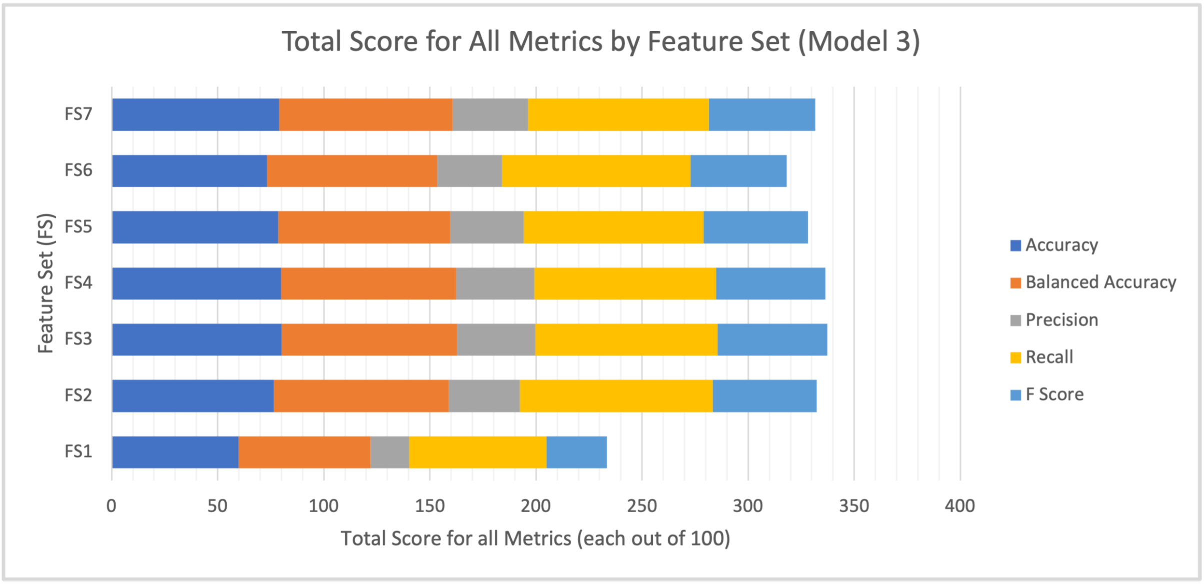


Figure 5.6: A comparison of the performance of the seven feature sets for all five evaluation metrics for Model 3

surveyed use a balanced distribution of genuine and fake reviews for both training and testing. This helps to explain the reason for the Precision and F scores being significantly lower for the results reported in this project compared to those reported in the literature, as will be seen. Other reasons for some disparities in results are due to diverse datasets used, different classifiers, different feature sets used (review-based and/or behavioural features) and differing hyper-parameter tuning applied.

Thus, directly comparing our results to the majority of results reported in the literature is challenging, as it is unclear whether testing data was balanced in terms of class distribution in a given paper. For example, in Zhang et al. (2016), the researchers outlined that they used a balanced set of authentic and fake reviews, but it is unclear whether the class distribution used for testing was also balanced. Likewise, Ren and Ji (2019) mention that the evaluation metrics of accuracy, precision, recall and F score were used on a "balanced dataset".

One of the first major publications within the domain by Ott et al. (2011) reported classification performance on balanced testing data with their pseudo-fake reviews, so many researchers after this have also reported classification performance on balanced data. However, arguably this makes the results look better than they would be in a real-world scenario where the classifier would be faced with an imbalanced dataset. This was the reason for a natural class distribution being used for testing in this project. Nevertheless, Ott et al. (2011) reported an accuracy, precision and recall of approximately 90% on their pseudo-fake dataset using only review-based features, for which our classifier (FS1) achieves significantly worse results. However, as seen in the Literature and Technology Survey, researchers such as Zhang et al. (2016) have critiqued this pseudo-fake dataset for making the results artificially better by not using real-world reviews.

Possibly the paper that has the most similar experimental scenario to this project is Mukherjee et al. (2013b), where the authors used and compared a balanced class distribution and natural class distribution of fake and genuine reviews, having used the same dataset used for this project (Yelp). For the restaurant data subset, which was used for this project, they reported a maximum of approximately 82% accuracy for a classifier trained on both review-based and behavioural features, with a natural class distribution of genuine and fake reviews for the testing dataset. For the same experimental scenario, they report an F score of around 63%, recall of 87% and precision of 49%. Although these results are somewhat higher than those reported for our classifier, the accuracy and recall are within around 2%. Arguably, the better performance reported for the F score and precision is due to more complex behavioural features used in their paper, which boosts the performance of their classifier. The authors also used the SVM classifier in contrast to the decision tree used in our classifier, which likely outperforms the latter classifier.

Zhang et al. (2016) reported for the Yelp dataset and the decision tree classifier (CART algorithm) an accuracy of approximately 83% for their classifier trained on review-based and behavioural features, with a precision of 86%, recall of 88%, and F Score of 87%. As is evident, although the accuracy and recall reported are not far from the results reported by Mukherjee et al. (2013b) and the results obtained in this project, the precision and F score are much higher, suggesting that the authors used a balanced testing dataset, rather than a natural class distribution of reviews. Indeed, these results are similar to the results reported by Mukherjee et al. (2013b) for the balanced testing dataset in their work.

All in all, although it is challenging to directly compare the results obtained for our classifier with those reported in the literature because of different experimental conditions, the results reported by Mukherjee et al. (2013b) are based on the same dataset used, a natural class distribution in the testing data and a combination of review-based and behavioural features, as was used for this project. Although their precision and F score reported are marginally higher than those obtained by our classifier, this is likely due to more complex behavioural features extracted from the data and a better performing classifier used (SVM). The accuracy and recall are comparable.

5.2 Front End Performance

As was outlined in the Requirements chapter, due to the two-fold focus of the dissertation on the creation of a machine learning model capable of detecting fake reviews and the development of a front end solution which can display its decision making in a transparent manner, it was decided early on together with the project supervisor that a formal user study would be out of scope for this dissertation. Such an approach for evaluating the transparency of the system is considered in the Future Work section in the concluding chapter. Conducting instead a less formal, smaller user study was decided to be a wrong approach which would likely yield weak results. Instead, evaluation of the front end's performance was to be done by comparing its capabilities and level of explainability to a number of transparency criteria and standards explored in the surveyed transparency literature in the Literature and Technology Survey.

5.2.1 Transparency Criteria

The transparency criteria and standards presented in the Literature and Technology Survey are repeated in Table 5.1 below for the reader's convenience. These criteria are explained in more detail in Section 2.5.1 of the Survey. Two columns have been added to this version of the table, each indicating whether the given version of the implemented user interface (v1.0 or v2.0) satisfies the specific criteria. Below, each criteria is considered in more detail and to what extent it has been satisfied in the front end.

Criteria	Description	In v1.0	In v2.0
1	Information should be presented at a suitable level of abstraction for the given audience	N/A	N/A
2	A natural language explanation of the system's decision making should be included	No	Yes
3	The explanation interface should be interactive, allowing the user to interrogate the system	No	No
4	A system fact sheet should accompany the system	No	No
5	The interface should present alternative system decisions	No	Yes
6	A confidence index should be displayed alongside the output of the system	Yes	Yes
7	The output of the system may be presented in different forms for accessibility	Yes	Yes
8	Past output and associated decisions should be traceable	Yes	Yes
9	Visualisations may support user understanding and decision making	Yes	Yes
10	The explanation interface should be well-suited to different scopes of explanation	Yes	Yes

Table 5.1: Transparency criteria satisfied in User Interface v1.0 and v2.0

Criteria 1: Information should be presented at a suitable level of abstraction for the given audience

This criteria was arguably satisfied for the updated version (v2.0) of the user interface due to the updates made to the initial version of the user interface, in particular the summarisation of node decisions, rounding of decision thresholds and updating feature names to be more intuitive to the user. However, arguably this criteria can only truly be satisfied by conducting a formal user study, as the user has the final say of whether the information presented is at the right level of abstraction, as seen in the transparency literature. Hence, this criteria has been marked as "not available", since the target user of the system, the researcher, product owner or customer, have the final say about the satisfaction of this criteria.

Criteria 2: A natural language explanation of the system's decision making should be included

This criteria was arguably not satisfied for the first version of the user interface, with the textual output lacking a natural language explanation, and rather being an unmodified output of the function used by the developer to acquire the necessary decision path information. However, the criteria was satisfied for the updated version of the interface, where a conscious effort was made to configure the output through the write_decisions function, explained in Section 4.4.1, to make the output more readable, less technical and less arbitrary to the user, as well as appear more like an explanation given by a human in a natural language style.

Criteria 3: The explanation interface should be interactive, allowing the user to interrogate the system

This criteria was not satisfied for either of the two versions of the user interface. Arguably, the user can reflect on explanations at different levels of detail through the implemented user interface, such as understanding the basic classification decision, the attributes used in the decision, the attributes not used in the decision, and a global view of the decision tree structure and the decision path within it. However, the two versions of the interface do not provide the capability of allowing the user to ask the system questions about the decision or the global reasoning of the classifier, or tweak feature values to see their effect on the classification decision, i.e., sensitivity analysis.

Criteria 4: A system fact sheet should accompany the system

This criteria was not satisfied for the two versions of the user interface. Although there is some metadata relating to the classification decision, such as the number of training samples with the same decision path as the classified sample, other important information as outlined by Mojsilovic (2018) that should be included in a system fact sheet, such as what data the algorithm was trained on, is not included. A formal fact sheet should accompany the system if developed into a full-fledged product.

Criteria 5: The interface should present alternative system decisions

This criteria was satisfied for the second version of the user interface. Specifically, the interface presents two probabilities of the review being genuine or fake, rather than stating a single classification decision with a single probability, as is done in the first version of the interface. The addition of an alternative decision in User Interface v2.0 should arguably help transfer more of the responsibility and accountability to the end user as they will be faced with both classification decisions and associated probabilities and will need to decide on the "final" classification themselves.

Criteria 6: A confidence index should be displayed alongside the output of the system

Both v1.0 and v2.0 present a confidence index alongside the output, satisfying this criteria. Specifically, the interpretable confidence index from the decision tree structure in the form of the distribution of class labels at the leaf is presented. For example, if at a given leaf node there are 500 genuine review training samples and 1000 fake review training samples, then the confidence index for the review being genuine is 33.3% and 66.7% for being fake.

Criteria 7: The output of the system may be presented in different forms for accessibility

This criteria is satisfied by both v1.0 and v2.0 of the user interface. Both versions present the output classification decision in two different forms: text and diagram form.

Criteria 8: Past output and associated decisions should be traceable

The output for both v1.0 and v2.0 is traceable, satisfying this criteria. This is because the general rules of the decision tree classifier are visible to the end user, and as such they can form an understanding of the reasoning for a specific classification decision, even after the output is first generated, unless the classifier is updated or changed.

Criteria 9: Visualisations may support user understanding and decision making

Both v1.0 and v2.0 include a tree diagram visualising the decision path of the selected review sample. The same decision path is also explained in text form in the output. The tree diagram also illustrates the overall decision tree structure and the rules within it, including the decision features and thresholds.

Criteria 10: The explanation interface should be well-suited to different scopes of explanation

Versions v1.0 and v2.0 of the user interface both present a local and global explanation of the model's decision making, with the text output and tree diagram, respectively. Where the text explanation explains the decision making at the local level, i.e., the decision making behind the specific, selected review, the decision tree diagram presents the decision at a global level, as the end user is able to see the rules contained within the entire decision tree structure and understand how samples traverse through it, including the sample they selected.

5.3 Summary

This chapter has presented and reflected on the results achieved by the implemented decision tree classifier, comparing the recorded results with the results reported in the literature, and evaluated the performance of the front end implemented using a set of transparency criteria and best practices from the transparency literature surveyed. The next chapter will discuss the implemented system in more detail, including the wider project methodology, as well as the limitations and contributions of the project.

Chapter 6

Discussion

In this chapter, we will continue the discussion of the performance of the implementation and consider the limitations of the wider implemented system, both the review classifier and the transparency offered by the user interface. The high-level project and development methodology will also be critiqued. All of these points discussed will be built on in the Future Work section of the concluding chapter. This chapter also summarises the system and research contributions of this project.

6.1 Project Methodology Limitations

Reflecting on the project methodology, including the literature review, requirements gathering and development process, there were a number of limitations of the approaches pursued.

6.1.1 Literature Review

Although a wide subset of the literature was consulted for both fake review detection machine learning approaches and transparency of autonomous systems, some areas could have been explored in more detail. For example, more advanced machine learning approaches that still provide elements of explainability for transparent systems could have been explored. Furthermore, in-depth examples of other explainable systems could have been investigated in more detail to get a better idea of how such systems are implemented in practice and their limitations. Also, the decision tree classifier, which was chosen as the classifier for the fake review detection model, could have been probed further from a theoretical stand-point to better understand how they are built from the feature set, and the differences between different decision tree algorithms like CART and ID3.

6.1.2 Requirements Gathering

As explained in Chapter 3, the requirements were largely derived from the literature surveyed and previous systems made within the fake review detection literature. Some requirements were also independently derived from the project aims outlined in the first chapter, such as the proposed aspects of originality. Arguably, this process worked well for setting the foundation for the fake review classifier implemented, however it would arguably have benefited the project by more closely understanding the user requirements for the transparent front end and the explainability aspect of the system implemented. A user study would have also helped to assess the demand for such a tool (or the need for better understanding fake reviews on the Web in general). However, as discussed earlier in this dissertation, a user study was deemed out of scope for this project due to its natural time constraints and the additional complexities of carrying out such a study.

6.1.3 Development Process

The development process at a high level was largely bottom-up, starting from the acquired labelled dataset and working up to the front end solution. In other words, the review classifier was implemented first, with the transparent user interface being built on top of the interpretable structure of the decision tree classifier. However, arguably the best possible explainable interface would result from a top-down approach, where the front end solution is envisioned and designed first, being shaped by the user's transparency requirements, such as the depth of explanation and elements of interactivity of the user interface required. Although the front end was made more transparent in the second version of the user interface implemented, its capabilities were largely constrained by the underlying structure of the decision tree classifier. For example, in explaining the individual decisions taken by the model, the underlying decision path from the decision tree was extracted.

In terms of the programming languages used, the vast majority of the system was implemented using Python, due to its general-purpose nature and large array of packages. Even though the front end implemented is not meant as a full-fledged product, it would have arguably offered a better user experience if made using a language like JavaScript to offer better interactive capabilities for the end user.

6.2 Model Limitations

This section considers the limitations of the machine learning model created for the fake review detection system.

6.2.1 Dataset

As discussed in the Literature and Technology Survey, a large assumption made about the acquired dataset was that the labels ("genuine" and "fake") were correct, as there is no feasible way to test them, being based on the proprietary algorithm used by Yelp. Although the dataset is highly regarded in the fake review detection literature, it is very likely that some labels are incorrect.

Another limitation of the dataset is the limited domain of the reviews (restaurants) and the limited geographical region from where the reviews are sourced - Chicago, US. Furthermore, the reviews are from the time period 2004-2012, since when spammers and fake review writers have arguably become more sophisticated in their approaches to crafting fake reviews.

These limitations of the dataset result in the risk of the model being based on a limited subset of characteristics of fake reviews on the Web in general, limiting its generalisation potential. Furthermore, as the dataset used is labelled by another algorithm, there is the risk of the model implemented being based on the representation of genuine and fake reviews as seen by the Yelp algorithm, rather than being an indication of genuine and fake reviews in general.

6.2.2 Classifier

The decision tree classifier, which was chosen in the implementation of the fake review detection model for this project, has a high level of interpretability and explainability, but likely produces worse performance compared to similar models based on other classifiers such as DNNs and various statistical models, as seen with the SVM classifier in the preceding chapter. Naturally, for this project there was a a compromise to be made in selecting the classifier to be used as two of the project aims were conflicting, namely creating a fake review detection model with comparable performance to the literature and developing an explainable interface such that the user can understand the reasoning of the model. Although the decision tree classifier was selected for both its good historical overall performance within the domain and its high level of interpretability, potentially other machine learning techniques could have been used offering better performance and a similar level of explainability to the decision tree, such as graphical models like Bayesian Belief Nets or ensemble methods like random forests (Gunning, 2017). Future work should explore the viability of such approaches in more detail.

Furthermore, although the supervised learning approach is dominant in the fake review detection literature, the limitations of labelled datasets, such as the one used for this project, may lead to biased models as described above. Other machine learning techniques like semi-supervised learning and unsupervised learning are of high recent interest to the domain since no labelled dataset is required. However, the challenge for such techniques is two-fold: creating a model with comparable performance to supervised learning techniques and making the model explainable, as it is likely to be more complex compared to interpretable techniques like the decision tree classifier or statistical models. Again, future work should further explore non-supervised approaches to fake review detection.

6.2.3 Feature Engineering

The feature engineering process was largely founded on the literature surveyed, in particular for feature extraction, yet there are a number of limitations of the process to outline.

For feature extraction, one limitation was the limited number of extracted features. Although both review-centric and reviewer-centric features were extracted to be used in the fake review detection model, a larger selection of both types of features could have been extracted, such as features based on the ratings of reviews which were not used, or a larger variety of NLP-based features, such as features based on the semantics of the reviews. Furthermore, the extracted features were mostly arbitrarily selected from a master list of previously extracted features in the domain (see Section 2.4.1 and 2.4.2 in the Literature and Technology Survey), except for ensuring a selection of both review-centric and reviewer-centric features. In future work, the feature extraction process should arguably be founded on the historically best-performing features to ensure the high performance of a fake review detection model.

The approach to feature extraction was manual in nature, which is the dominant approach used in the fake review detection domain, with features being defined explicitly and then extracted. Arguably, the manual approach helped to craft interpretable features. However, automatic feature extraction techniques could have been explored, which hold the potential of leading to better features being extracted, although the challenge with such techniques would be to ensure the interpretability of the features for the explainability of the system. For example, automatically extracted features could lead to better overall model performance, but may be too complex for the user (or even researcher) to understand. In future work, automatic feature extraction should be explored for the fake review detection domain, as this was not seen in the literature surveyed.

For the process of feature selection only the importance of individual features was assessed, with the creation of feature sets used for the model evaluation largely based on the results of this analysis. Principal Component Analysis (PCA), although part of the model requirements presented in the Requirements chapter, was not done due to time constraints and prioritising the development of v2.0 of the user interface. Although the comparison of subsets of features allowed for understanding the best-performing sub-sets of features, conducting PCA would arguably have lead to a better-performing model and a better understanding of the discriminant features and the relationships between individual features.

6.2.4 Model Evaluation

Similarly to the process of feature extraction, the evaluation of the classifier was largely influenced by the surveyed literature. In particular, the commonly used 10-fold cross-validation approach and the four metrics of accuracy, recall, precision and F1 score were used as these appeared in a majority of fake review detection papers surveyed and would lead to a fair comparison.

A component of the model evaluation worth critiquing was the naive approach to resampling the fake review dataset prior to training the classifier. For oversampling, there was a duplication of characteristics of the fake reviews leading to the model overfitting to this data to an extent. For undersampling, valuable information was potentially lost from the genuine reviews discarded, and as this was done at random, not all valuable characteristics of genuine reviews may have been captured at the learning stage. Further work could utilise more complex methods for resampling, such as creating synthetic data samples for oversampling and clustering-based undersampling to ensure as little information as possible is lost about the majority class in the discarded samples.

Also, the hyper-parameter tuning approach for reducing the complexity and increasing the performance of the model had its limitations, even though it satisfied these goals. In particular, only the `max_depth` parameter was investigated in detail using a form of grid search, with other parameters not actively explored. Future work should consider using a grid search on a larger subset of parameters to further increase the performance of the classifier and potentially help to make the tree more interpretable by reducing its complexity.

6.3 Front End Limitations

This section considers the limitations of the transparent front end created with the goal of explaining the decision-making of the implemented fake review detection model.

6.3.1 User Study

Arguably the most important limitation to note regarding the front end is the lack of a user study for empirically assessing the usefulness of the implemented user interface and its design. As outlined in the limitations of the requirements gathering process, the lack of a user study meant that the front end implemented was not based directly on user requirements and needs, but rather the criteria and standards used within the transparency literature for autonomous systems in general, and the explanation interpretable from the decision tree classifier, i.e., the chain of decisions leading to the classification. Future work must assess the transparency and usability of such an interface and its usefulness for both consumers and organisations.

6.3.2 Transparency

Although the decision tree classifier which was used for the fake review detection model is interpretable by its nature with a developer or user being able to understand its reasoning by inspecting the decision path or structure of the decision tree, as was done in the implemented front end, one may argue that the output is not fully explainable. Arguably, the decision path and overall decision tree structure is arbitrary to an extent, especially to a naive user. For example, the user may question why the classifier has chosen the particular threshold features and values for its nodes. As noted by Bond et al. (2019), "decision trees can easily provide decision explanations..., however these rules can often be counter-intuitive". Based on this argument, one may argue that the implemented user interface (which is based on the rules interpretable from the decision tree classifier) may be confusing to the user to an extent, thus harming the transparency and explainability of the user interface.

Further discussing the level of explainability offered by the front end, it also worth noting the transparency of the extracted features for use in the classifier, which are shown to the user in the user interface. Arguably, as they were manually extracted they are more transparent than potentially machine extracted features, however, some features are arguably a bit arbitrary, such as maxTfidf. Although this feature was re-phrased in the user interface to a more user-friendly explanation ("Maximum word importance score (TF-IDF)"), it arguably requires more explanation for the user to understand it well and build a mental model similar to the developer.

Ultimately, as is clear from the transparency literature, the transparency of an intelligent system depends on the kind of user who will be using the system. The implemented system was designed for a naive end user, as per Requirement 3.6, and as such should make sense to a person with no experience in machine learning. Arguably, such users would find it more challenging to interpret the explanation and tree diagram displayed in the front end. Related to this is the extent to which the user may trust the system and to what extent they form an accurate mental model of the model's functioning. Therefore, as outlined earlier in this section, a user study would help in assessing the transparency of the front end as accurately as possible.

On the opposite end of the spectrum, there is the argument that the implemented front end is too transparent for a naive end user, such as a customer scrolling through reviews on an e-commerce website. Arguably, the decision path explanations and tree diagrams presented in the implemented user interface are prone to being gamed by a spammer, as they would be aware of how the model classifies a review as genuine or fake. Therefore, a middle ground is needed if a similar classifier is used in a public setting like an e-commerce website. Arguably, the internal decision-making of the model should only be disclosed within the business or organisation who own the model, with the end user facing explanation being drastically simplified to avoid potentially harmful individuals such as spammers gaining valuable knowledge about the model's decision making rules. For example, taking inspiration from the implemented user interface, an option could be to only present the classification probabilities, such as "According to the model, this review is likely to be Genuine with probability 65% and Fake with probability 35%". On the other hand, employees within the business or organisation could still use a more detailed and transparent solution as implemented in this project, so they could improve the system, investigate review misclassifications and ensure that the model gives reliable predictions.

6.3.3 User Interface

Another limitation of the front end implemented which was briefly mentioned in the limitations of the development process is the static nature of the user interface with a lack of interactivity in the current implementation. This idea is also reflected by Criteria 3 of the transparency criteria evaluated in the Results chapter - a criteria that the user interface did not satisfy. Although the front end is not meant as a full-fledged product, rather being a demonstration of the potential of such a solution, a future solution should be developed using technologies such as JavaScript to allow for better interactivity and usability for the end user. Better interactivity in the user interface would also help with satisfying some of the transparency criteria that were not satisfied by the implementation in this project, such as the ability of the user to interrogate and ask the system questions.

6.4 System Limitations

This section considers the high-level limitations of the implemented solution as a whole.

Firstly, it is worth highlighting the lack of integration between the individual components of the implemented system. The components of the system, including data preparation, feature engineering, feature selection and the user interface are arguably not well integrated. For future work, developers of a similar system should consider creating a more integrated pipeline, which would be very useful for situations such as the dataset being updated or when re-training of the underlying model is needed based on new features extracted.

Secondly, as seen during the evaluation of the front end in the Results chapter, the system does not currently have a fact sheet (transparency Criteria 4 in surveyed literature). Important information such as what data the algorithm was trained on and the results of bias analysis of this dataset should be included in a fact sheet document if the system is made into a full-fledged product. This would allow potential product owners, users and other stakeholders to be clear about questions such as the underlying foundation of the decision making of the classifier and the classifier's applicability to a given problem.

Thirdly, it is worth considering to what extent the system is applicable and generalisable to other online platforms and other domains of reviews. Arguably, in part due to the limitations of the dataset discussed earlier in this chapter, the generalisation potential of the system as a whole is limited. Also, websites not offering a large variety of review metadata like Yelp (e.g., funny votes, useful votes) may result in a poorly performing model if it is only based on the review text content, as illustrated in the results presented in the preceding chapter.

Another high-level consideration which emerged during the project was whether the problem of detecting fake reviews is being framed in the correct way in the surveyed fake review detection literature. In the majority of articles surveyed in the literature review, the problem of fake review detection is of a binary nature, which is well-aligned with the approach of binary classification. However, perhaps reviews should be plotted on a scale of "suspiciousness", rather than being either genuine or fake.

6.5 Contributions

This dissertation has applied a range of machine learning techniques and principles from the transparency literature to develop a transparent fake review detection classifier, with comparable performance to the literature, with the additional capability of being able to explain its decision making through an explainable interface to a naive end user. To the best of the author's knowledge, the combined fake review detection classifier and explainable user interface together serve as a first system of its kind within the domain of fake review detection, highlighting the potential of a full-fledged version of such a product in commercial and research contexts.

6.5.1 System Contributions

The system developed for this dissertation offers a number of contributions to the field of fake review detection. In particular, it serves as the first fake review detection system which is capable of explaining its decision making to an end user based on principles of transparent intelligent systems. The following list presents a more detailed view of the system contributions of this dissertation:

- Proposed and developed a fake review detection classifier with manually extracted features based on data from an acquired dataset sourced from the business review site *Yelp.com*.
- Trained, tested and evaluated different configurations of classifiers with the best-performing classifier having comparable performance to the models surveyed in literature.
- Designed and developed a front end layer capable of explaining the classifier's decision making to the end user based on a review that they select.
- Developed a front end layer with the capability of explaining model decision making in both natural language form and visual form to the end user.

- Developed the system using mainly open-source technologies, making the system generally replicable for future work in both research and commercial applications.
- Used a generalisable approach for the developed user interface, with the front end being compatible with different decision tree structures.

6.5.2 Research Contributions

This dissertation offers a number of research contributions to the domains of fake review detection and transparency of intelligent systems.

- Produced an in-depth overview of the latest fake review detection literature, transparency research and machine learning techniques pertaining to fake review detection, in the form of the Literature and Technology Survey.
- Evaluated the performance of the developed fake review detection classifier with manually extracted review-centric and reviewer-centric features from the Yelp restaurant dataset.
- Compared and analysed the performances of different configurations of fake review detection classifiers, including different sets of features, balanced and unbalanced training data and hyper-parameter tuning. In particular, the stark difference between training and testing fake review detection classifiers with balanced and unbalanced data was highlighted.
- Developed a fake review detection classifier with an innovative transparent front end to serve as a first system of its kind.

6.5.3 Commercial Contributions

Further to the research contributions, the project also presents commercial contributions.

- The project has demonstrated the potential of a full-fledged version of the implemented transparent fake review detection system for use in commercial applications.
- Discussed the viability and limitations of a transparent fake review detection system for use in organisations.

6.6 Summary

This chapter has outlined through discussion the key limitations of the implemented system and the limitations of the high-level project and development methodology, inspiring future work, which is outlined in the next chapter. This chapter has also presented the key contributions of the project.

Chapter 7

Conclusion and Future Work

In this chapter the major achievements of the dissertation are reviewed in the light of the original project aims set out in the introductory chapter. This chapter also identifies and offers recommendations for future work, based on the limitations of the project highlighted in the preceding chapter.

7.1 Conclusion

This dissertation had the high-level aim of addressing the currently present issue of fake reviews on the Web, which have detrimental effects on individual consumers, organisations and governments. After a thorough review of the literature, an innovative opportunity was identified in crafting a transparent fake review detection model capable of identifying genuine and fake reviews, and having the capability to present its decision making to the end user. The classifier was implemented using a real-world labelled dataset acquired from the business review website Yelp.com, from which review-based and behavioural features were extracted and selected. The performance of different configurations of classifiers were evaluated, with the best performing classifier having comparable performance to classifiers reviewed in the literature.

A front end was developed presenting the output of the implemented classifier in the form of a natural language explanation and diagram illustrating its decision making based on a review sample selected by the end user. The user interface was evaluated using transparency criteria and best practices identified in the transparency literature.

Despite the limitations of the project approach outlined in the Discussion, the results of the implemented classifier and the level of transparency offered by the system demonstrate the potential of full-fledged version of such a system in research and commercial applications, especially as the domains of fake review detection and explainable intelligent systems are both becoming increasingly more salient within the technology community and society in general.

7.2 Future Work

Having highlighted and discussed the limitations of the implemented system and the general project methodology in the previous chapter, this section suggests appropriate future work that could help to address these limitations and improve on the work done.

7.2.1 Fake Review Detection Model

One key limitation that was highlighted in the previous chapter with regards to the fake review detection model implemented were the various limitations of using a supervised machine learning approach which requires a labelled dataset. As mentioned, recently other machine learning approaches have become of interest to the domain, in particular semi-supervised and unsupervised learning, as labelled datasets within this domain are a challenge for a number of reasons, as explained in the Literature and Technology Survey. Future work should explore the viability of such approaches in more detail.

Future work should also explore using other classifiers offering better performance compared to the decision tree, whilst maintaining a similar level of transparency offered by the implemented system for the end user. One of the techniques that future work may explore are model induction approaches, where an explainable model is inferred from a black box model, as described in Gunning (2017). In general, further work should explore other machine learning techniques for further improving the learning performance of the classifier, while maintaining the aspect of explainability offered by the system implemented in this dissertation.

Another limitation of the implemented system was the lack of principal component analysis (PCA) of the features used for training the model. Using PCA or similar techniques for understanding the relationships between individual features would assist in selecting better combinations of features to be used in the classifier. As the feature extraction process was largely manual for this project, further work should also explore automatic feature extraction techniques and assess their usefulness for the fake review detection domain. Another consideration for future work in feature engineering would be to summarise best historically performing features to inform manual feature extraction going forwards.

As has been evident throughout this project, a key challenge with datasets in this domain is that the labels are highly imbalanced. This requires resampling techniques to be applied prior to model training to overcome the data imbalance problem and the classifier being highly biased in detecting reviews as genuine, whilst achieving poor performance in identifying fake reviews. Future work should explore more sophisticated methods for resampling, such as developing synthetic fake reviews (oversampling) and undersampling genuine reviews using approaches such as clustering to retain as much information about the class as possible.

7.2.2 Explainable User Interface

As explained in the preceding chapter, one of the major limitations of the front end developed in this dissertation and the application of transparency principles, was the lack of a formal user study to assess the level of explainability and transparency offered by the system, as well as its design. As such, future work must empirically evaluate the viability of such an interface and usefulness for both end users and organisations.

User studies would also assist in tailoring the user interface to the specific user. For now, the front end in is based generally on transparency principles. A user study would allow for fine-tuning of the front end depending on what the user prefers. Furthermore, it is clear from the transparency literature that the transparency of a system ultimately depends on the user who will be working with the system or its output. Therefore, any future work in this area should be clear on the user, along with their goals and intentions for the designer of the system. As the authors of Bond et al. (2019) suggest, an explainable interface should be co-designed with the end users of the given system to achieve the level of transparency desired. Specific aspects of the implemented system which may be explored in a user study include the level of explainability of the features and the logic of the overall classifier decision making.

From assessing the transparency and performance of the implemented user interface in this dissertation, one key transparency criteria which was left unsatisfied was the ability of the end user to interrogate and ask the system questions. As such, future work should aim to add more interactivity to the user interface. For example, a user may want to adjust the feature values of a given review to gauge the sensitivity of the model. This would benefit the system in numerous ways, including increasing the user's trust of the system, and the user being able to build a more accurate mental model of the system's functioning. Finally, future work should also investigate the extent to which the transparency offered by the system may be harmful, with the risk of spammers being potentially able to game the system.

Bibliography

Al Najada, H. and Zhu, X., 2014. isrd: Spam review detection with imbalanced data distributions. *Proceedings of the 2014 ieee 15th international conference on information reuse and integration (ieee iri 2014)*. IEEE, pp.553–560.

Bond, R.R., Mulvenna, M., Finlay, D., Wong, A., Koene, A., Brisk, R., Boger, J. and Adel, T., 2019. Human Centered Artificial Intelligence: Weaving UX into Algorithmic Decision Making. *Rochi 2019: International conference on human-computer interaction.*

Cardoso, E.F., Silva, R.M. and Almeida, T.A., 2018. Towards automatic filtering of fake reviews. *Neurocomputing*, 309(1), pp.106–116.

Competition and Markets Authority, 2020. Digital markets: using our existing tools and emerging thoughts on a new regime [Online]. Andrea Coscelli, Chief Executive of the CMA, delivers a keynote speech to the annual Fordham Competition Law Institute conference on the CMA's existing tools and a new regime to support competition and innovation in digital markets. [Accessed: 5 November 2020]. Available from: `https://www.gov.uk/government/speeches/digital-markets-using-our-existing-tools-and-emerging-thoughts-on-a-new-regime`.

Crawford, M., Khoshgoftaar, T.M., Prusa, J.D., Richter, A.N. and Al Najada, H., 2015. Survey of review spam detection using machine learning techniques. *Journal of big data*, 2(1), p.23.

Domingos, P., 2012. engA few useful things to know about machine learning. *Communications of the acm*, 55(10), pp.78–87.

Floyd, K., Freling, R., Alhoqail, S., Cho, H.Y. and Freling, T., 2014. How online product reviews affect retail sales: A meta-analysis. *Journal of retailing*, 90(2), pp.217–232.

Gunning, D., 2017. Explainable artificial intelligence (xai). *Defense advanced research projects agency (darpa), nd web*, 2(2).

Honnibal, M. and Montani, I., 2017. *spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing*. Unpublished.

Hovy, D. and Spruit, S.L., 2016. The social impact of natural language processing. *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 2: Short papers)*. pp.591–598.

Hu, L., Chen, J., Nair, V.N. and Sudjianto, A., 2018. Locally interpretable models and effects based on supervised partitioning (lime-sup). *arxiv preprint arxiv:1806.00663.*

Hunt, K.M., 2015. Gaming the system: Fake online reviews v. consumer law. *Computer law security review*, 31(1), pp.3–25.

Hutto, C. and Gilbert, E., 2014. Vader: A parsimonious rule-based model for sentiment analysis of social media text. *Proceedings of the international aaai conference on web and social media*. vol. 8.

Jolliffe, I.T., 2002. eng*Principal component analysis*, Springer series in statistics. 2nd ed. New York: Springer.

Kumar, N., Venugopal, D., Qiu, L. and Kumar, S., 2019. Detecting anomalous online reviewers: an unsupervised approach using mixture models. *Journal of management information systems*, 36(4), pp.1313–1346.

Lee, E.J. and Shin, S.Y., 2014. When do consumers buy online product reviews? effects of review quality, product type, and reviewer's photo. *Computers in human behavior*, 31(1), pp.356–366.

Li, X., Wu, C. and Mai, F., 2019. The effect of online reviews on product sales: A joint sentiment-topic analysis. *Information management*, 56(2), pp.172–184.

Liao, Q.V., Gruen, D. and Miller, S., 2020. Questioning the ai: informing design practices for explainable ai user experiences. *Proceedings of the 2020 chi conference on human factors in computing systems*. pp.1–15.

Manning, C.D., Raghavan, P. and Schütze, H., 2009. *An introduction to information retrieval* [Online]. Cambridge University Press. Accessed: 30-11-2020. Available from: `https://nlp.stanford.edu/IR-book/`.

Méndez, J.R., Iglesias, E.L., Fdez-Riverola, F., Díaz, F. and Corchado, J.M., 2005. Tokenising, stemming and stopword removal on anti-spam filtering domain. *Conference of the spanish association for artificial intelligence*. Springer, pp.449–458.

Mojsilovic, A., 2018. *Factsheets for AI Services* [Online]. Accessed: 07-04-2021. Available from: `https://www.ibm.com/blogs/research/2018/08/factsheets-ai/`.

Mukherjee, A., Kumar, A., Liu, B., Wang, J., Hsu, M., Castellanos, M. and Ghosh, R., 2013a. Spotting opinion spammers using behavioral footprints. *Proceedings of the 19th acm sigkdd international conference on knowledge discovery and data mining*. pp.632–640.

Mukherjee, A., Venkataraman, V., Liu, B., Glance, N. et al., 2013b. Fake review detection: Classification and analysis of real and pseudo reviews. *Uic-cs-03-2013. technical report*.

Mukherjee, A., Venkataraman, V., Liu, B. and Glance, N.S., 2013c. What yelp fake review filter might be doing? *International aaai conference on web and social media (icwsm)*, 2013, pp.409–418.

Nadella, S., 2016. The Partnership of the Future: Microsoft's CEO explores how humans and A.I. can work together to solve society's greatest challenges. *Slate magazine* [Online]. Available from: `https://slate.com/technology/2016/06/microsoft-ceo-satya-nadella-humans-and-a-i-can-work-together-to-solve-societys-challenges.html`.

Natural Language Processing (NLP) – University of Copenhagen, 2020. [Online]. Accessed: 01-12-2020. Available from: `https://di.ku.dk/english/research/groups/nlp/`.

2010. Orlando figes to pay fake amazon review damages. *Bbc news* [Online]. Available from: `https://www.bbc.co.uk/news/uk-10670407` [Accessed 2020-11-05].

Ott, M., Cardie, C. and Hancock, J.T., eds, 2013. *Negative deceptive opinion spam*. North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL2013), Atlanta, Georgia: Association for Computational Linguistics.

Ott, M., Choi, Y., Cardie, C. and Hancock, J.T., 2011. Finding deceptive opinion spam by any stretch of the imagination. *Corr* [Online], abs/1107.4557. `1107.4557`, Available from: `http://arxiv.org/abs/1107.4557`.

Park, C. and Lee, T.M., 2009. Information direction, website reputation and ewom effect: A moderating role of product type. *Journal of business research*, 62(1), pp.61–67.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. and Duchesnay, E., 2011. Scikit-learn: Machine learning in Python. *Journal of machine learning research*, 12, pp.2825–2830.

Ren, Y. and Ji, D., 2019. Learning to detect deceptive opinion spam: A survey. *Ieee access*, 7, pp.42934–42945.

Ribeiro, M.T., Singh, S. and Guestrin, C., 2016. " why should i trust you?" explaining the predictions of any classifier. *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. pp.1135–1144.

Rudin, C., 2019. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature machine intelligence*, 1(5), pp.206–215.

Sedighi, Z., Ebrahimpour-Komleh, H. and Bagheri, A., 2017. Rlosd: Representation learning based opinion spam detection. *2017 3rd iranian conference on intelligent systems and signal processing (icspis)*. IEEE, pp.74–80.

Sokolova, M., Japkowicz, N. and Szpakowicz, S., 2006. Beyond accuracy, f-score and roc: A family of discriminant measures for performance evaluation. In: A. Sattar and B.h. Kang, eds. *Ai 2006: Advances in artificial intelligence*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp.1015–1021.

Spagnolli, A., Frank, L.E., Haselager, P. and Kirsh, D., 2017. Transparency as an ethical safeguard. *International workshop on symbiotic interaction*. Springer, pp.1–6.

Turkel, W. and Crymble, A., 2012. Output data as an html file with python [Online]. Available from: `https://programminghistorian.org/en/lessons/output-data-as-html-file`.

Unknown Author, 2021. Model selection [Online]. [Online; accessed April 08, 2021]. Available from: `http://ethen8181.github.io/machine-learning/model_selection/model_selection.html`.

Virtanen, P., Gommers, R., Oliphant, T.E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S.J., Brett, M., Wilson, J., Millman, K.J., Mayorov, N., Nelson, A.R.J., Jones, E., Kern, R., Larson, E., Carey, C.J., Polat, İ., Feng, Y., Moore, E.W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E.A., Harris, C.R., Archibald, A.M., Ribeiro, A.H., Pedregosa, F., van Mulbregt, P. and SciPy 1.0 Contributors, 2020. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature methods* [Online], 17, pp.261–272. Available from: `https://doi.org/10.1038/s41592-019-0686-2`.

Wang, D., Yang, Q., Abdul, A. and Lim, B.Y., 2019. Designing theory-driven user-centric explainable ai. *Proceedings of the 2019 chi conference on human factors in computing systems*. pp.1–15.

Weld, D.S. and Bansal, G., 2019. The challenge of crafting intelligible intelligence. *Communications of the acm*, 62(6), pp.70–79.

Winfield, A.F., Dennis, L.A., Egawa, T., Olszewska, J.I., Theodorou, A., Wortham, R.H. and Watson, E., 2021. *IEEE P7001: a new standard on Transparency*. Unpublished.

Wortham, R.H., 2020. *Transparency for Robots and Autonomous Systems: Fundamentals, technologies and applications: Fundamentals, technologies and applications*. Stevenage: The Institution of Engineering and Technology.

Xiao, S., Wei, C.P. and Dong, M., 2016. Crowd intelligence: Analyzing online product reviews for preference measurement. *Information management*, 53(2), pp.169–182.

Yang, F., Pentyala, S.K., Mohseni, S., Du, M., Yuan, H., Linder, R., Ragan, E.D., Ji, S. and Hu, X., 2019. Xfake: explainable fake news detector with visualizations. *The world wide web conference*. pp.3600–3604.

Zhang, D., Zhou, L., Kehoe, J.L. and Kilic, I.Y., 2016. What online reviewer behaviors really matter? effects of verbal and nonverbal behaviors on detection of fake online reviews. *Journal of management information systems* [Online], 33. Available from: `https://doi.org/10.1080/07421222.2016.1205907`.

Zimmermann, M., Ntoutsi, E. and Spiliopoulou, M., 2016. Extracting opinionated (sub)features from a stream of product reviews using accumulated novelty and internal re-organization. *Information sciences*, 329(1), pp.876–899.

Appendix A

Model Performance - Raw Results

Feature Set	Performance Metric				
	Accuracy	*B. Accuracy*	*Precision*	*Recall*	*F score*
FS1	78.02	52.18	15.76	17.82	16.72
FS2	88.45	76.94	52.92	61.63	56.94
FS3	85.16	66.62	40.47	41.98	41.19
FS4	84.93	65.88	39.47	40.56	40.00
FS5	84.28	64.55	37.02	38.31	37.64
FS6	82.83	61.02	31.22	32.03	31.61
FS7	84.72	65.84	38.88	40.73	39.77

Table A.1: Average performance of 1st set of models (no resampling, no hyper-parameter tuning)

Feature Set	Performance Metric				
	Accuracy	*B. Accuracy*	*Precision*	*Recall*	*F score*
FS1	65.46	54.36	15.35	39.60	22.12
FS2	75.86	82.54	32.96	91.41	48.43
FS3	80.83	74.02	35.19	64.96	45.64
FS4	80.73	74.01	35.05	65.08	45.55
FS5	79.25	71.71	32.32	61.68	42.41
FS6	76.74	67.89	28.06	56.11	37.41
FS7	79.86	72.45	33.35	62.60	43.51

Table A.2: Average performance of 2nd set of models (with resampling, no hyper-parameter tuning)

Feature Set	Performance Metric				
	Accuracy	*B. Accuracy*	*Precision*	*Recall*	*F score*
FS1	59.98	61.97	18.38	64.62	28.58
FS2	76.44	82.62	33.43	90.84	48.86
FS3	80.16	82.56	37.10	85.76	51.75
FS4	79.88	82.40	36.95	85.75	51.52
FS5	78.45	81.08	34.79	84.58	49.30
FS6	73.41	80.09	30.46	88.97	45.35
FS7	79.05	81.68	35.58	85.17	50.19

Table A.3: Average performance of 3$^{\text{rd}}$ set of models (with resampling, with hyper-parameter tuning)

Appendix B

Feature Performance - Raw Results

Feature Name	T Value	Abs T Value
hasProfile	-71.05	71.05
postCount	53.93	53.93
usefulCount	43.30	43.30
maxTfidf	-33.83	33.83
reviewLength	32.53	32.53
coolCount	32.41	32.41
funnyCount	27.69	27.69
maxReviews	24.54	24.54
numCount	20.09	20.09
symCount	16.31	16.31
adjProp	-15.36	15.36
avgSentiment	-12.36	12.36
nounProp	-5.32	5.32
verbProp	-1.18	1.18
propernounProp	-0.82	0.82

Table B.1: T-test results

Feature Name	Correlation Value	Abs Correlation Value
hasProfile	0.26	0.26
postCount	-0.20	0.20
usefulCount	-0.16	0.16
maxTfidf	0.13	0.13
reviewLength	-0.12	0.12
coolCount	-0.12	0.12
funnyCount	-0.11	0.11
maxReviews	-0.09	0.09
numCount	-0.08	0.08
symCount	-0.06	0.06
adjProp	0.06	0.06
avgSentiment	0.05	0.05
nounProp	0.02	0.02
verbProp	0.00	0.00
propernounProp	0.00	0.00

Table B.2: Point-biserial correlation results

Feature Name	Gini Importance
usefulCount	0.280
postCount	0.152
avgSentiment	0.093
reviewLength	0.091
max_tfidf	0.085
nounProp	0.080
adjProp	0.078
hasProfile	0.061
numCount	0.022
coolCount	0.018
maxReviews	0.016
funnyCount	0.016
symCount	0.009

Table B.3: Average feature (Gini) importance results during 10-fold CV

Appendix C

Code

C.1 Data Preparation: data_preparation.py

```python
# import framework to work with sqlite database in Python
import sqlite3
# pandas will be used for manipulating data sets retrieved from the database file
import pandas as pd
import unicodedata
# pickle will be used to serialise the prepared dataframe to be loaded later
import pickle

# set up connection to the database
connection = sqlite3.connect("yelpResData.db")
# create cursor object for interaction with the database
cur = connection.cursor()
# specify how to handle bytes in database
connection.text_factory = lambda x: str(x, 'utf-8', 'ignore')

# previously attempted text factories below
```

```
#connection.text_factory = bytes
#connection.text_factory = lambda x: str(x, 'iso-8859-1')

cur.execute("SELECT name FROM sqlite_master WHERE type='table';")
print(cur.fetchall())

review_table = pd.read_sql_query('SELECT * FROM review', connection)
#restaurant_table = pd.read_sql_query('SELECT * FROM restaurant', connection)
reviewer_table = pd.read_sql_query('SELECT * FROM reviewer', connection)

series = pd.Series(review_table['reviewContent'])
series = series.str.normalize("NFKD")
review_table['reviewContent'] = series.values

reviews = review_table[(review_table["flagged"] == 'Y') | (review_table["flagged"] == 'N')].reset_index(drop = True)

number_of_reviews = len(reviews)
number_of_genuine_reviews = (reviews["flagged"] == 'N').sum()
number_of_filtered_reviews = (reviews["flagged"] == 'Y').sum()

print("Number of reviews:", number_of_reviews)
print("Number of genuine reviews:", number_of_genuine_reviews)
print("Number of filtered reviews:", number_of_filtered_reviews)

# check if there are any duplicate reviews
reviews["reviewID"].nunique()

# save file to csv for further inspection
#df.to_csv("dataframe.csv", encoding='utf-8', index=False)

# identify indeces containing "Updated" to remove the strings
date_column = pd.Series(reviews["date"])
indeces = date_column.str.contains('updated', case=False, regex=True)
print(indeces.value_counts())

true_indeces = indeces[indeces == True].index

```

```python
# print values to fix
for index in range(0, len(true_indeces)):
    print(date_column.iloc[true_indeces[index]])

# "slice" the individual values to remove "Updated -"
for index in true_indeces:
    date_column.iloc[index] = date_column.iloc[index][10:]

# check that update is successful
indeces = date_column.str.contains('updated', case=False, regex=True)
print(indeces.value_counts())

# make date formatting consistent in the date column of the dataframe
reviews["date"] = date_column
reviews["date"] = pd.to_datetime(reviews["date"])

# show count of reviews by year
years = reviews["date"].dt.year
print(years.value_counts())

# remove empty reviews
empty_review_index_list = [62005, 62792]
reviews = reviews.drop(reviews.index[empty_review_index_list])

reviews = reviews.sort_index()

columns = [reviews["date"], reviews["reviewerID"], reviews["reviewContent"], reviews["flagged"]]
headers = ["date","reviewerID", "reviewContent", "flagged"]
reviews = pd.concat(columns, axis=1, keys=headers).reset_index(drop = True)

reviews

# temporary - get usefulCount, coolCount and funnyCount features from dataframe
columns = [reviews["usefulCount"], reviews["coolCount"], reviews["funnyCount"]]
headers = ["usefulCount","coolCount", "funnyCount"]
extra_features = pd.concat(columns, axis=1, keys=headers).reset_index(drop = True)

```

```python
# pickle
extra_features.to_pickle("./extra_features.pkl")

# explore "anonymous" reviewers
# i.e., those not found in the reviewer table (from the exercise in the above cell)

# "anonymous" means ID in reviews table but NOT in reviewerID
anonymous_reviewers = []
has_profile = []

reviewers_in_table1 = pd.Series(list(reviews["reviewerID"].unique()))
reviewers_in_table2 = set(list(reviewer_table["reviewerID"]))

for reviewer in reviewers_in_table1:
    if(reviewer not in reviewers_in_table2):
        anonymous_reviewers.append(reviewer)
        has_profile.append(0)
    else:
        has_profile.append(1)

print("Number of unique reviewers in reviews table: ", len(reviews['reviewerID'].unique()))
print("Number of unique reviewers in reviewer table: ", len(reviewer_table['reviewerID'].unique()))

print("Number of \"anonymous\" reviewers: ", len(anonymous_reviewers))
# e.g. reviewer ID "xMYPc5tzV2PSryKFK_y1PQ" is found in reviews but not reviewers table
print()
print("Examples of \"anonymous\" reviewers: ")
print("Number of reviews in reviews table with reviewer ID xMYPc5tzV2PSryKFK_y1PQ: ",
      len(reviews[reviews["reviewerID"] == "xMYPc5tzV2PSryKFK_y1PQ"]))
print("Number of reviews in reviewers table with reviewer ID xMYPc5tzV2PSryKFK_y1PQ: ",
      len(reviewer_table[reviewer_table["reviewerID"] == "xMYPc5tzV2PSryKFK_y1PQ"]))
print()
print("Number of reviews in reviews table with reviewer ID ciAaaK5kBPGM1y8CtkJtXQ: ",
      len(reviews[reviews["reviewerID"] == "ciAaaK5kBPGM1y8CtkJtXQ"]))
print("Number of reviews in reviewers table with reviewer ID ciAaaK5kBPGM1y8CtkJtXQ: ",
      len(reviewer_table[reviewer_table["reviewerID"] == "ciAaaK5kBPGM1y8CtkJtXQ"]))

```

```
reviews[reviews["reviewerID"] == "xMYPc5tzV2PSryKFK_y1PQ"]
# for this particular reviewer, we can see that they are not present in the reviewers' table, yet have posted
# many times at different dates, from 2007-2009

# check if any two groups of users do not have fake reviews
x1 = len(reviews[(reviews['hasProfile'] == 1) & (reviews['flagged'] == 'Y' )])
x2 = len(reviews[(reviews['hasProfile'] == 1) & (reviews['flagged'] == 'N' )])
print("Proportion of fake reviews by reviewers with a profile: ")
print(x1 / (x1 + x2))
print("Proportion of genuine reviews by reviewers with a profile: ")
print(x2 / (x1 + x2))
print("Total: ", x1 + x2)
print()
x3 = len(reviews[(reviews['hasProfile'] == 0) & (reviews['flagged'] == 'Y' )])
x4 = len(reviews[(reviews['hasProfile'] == 0) & (reviews['flagged'] == 'N' )])
print("Proportion of fake reviews by reviewers without a profile: ")
print(x3 / (x3 + x4))
print("Proportion of genuine reviews by reviewers without a profile: ")
print(x4 / (x3 + x4))
print("Total: ", x3 + x4)

# add column in reviews table to indicate whether a reviewer is "anonymous"
reviews["hasProfile"] = has_profile

reviews.to_pickle("./reviews.pkl")

cur.close()
connection.close()
```

C.2 Feature Engineering: feature_engineering.py

```python
# Import required libraries

import pandas as pd
import numpy as np
import pickle
import time
import nltk
# a tokenizer will be used for pre-processing the review strings for feature extraction
from nltk import tokenize
nltk.download('punkt')
# Vader and its lexicon will be used for the extraction of a sentiment analysis-based feature
from nltk.sentiment.vader import SentimentIntensityAnalyzer as SIA
nltk.download('vader_lexicon')
# spaCy will be used for tokenising (into words) reviews and tagging parts of speech within them
import spacy
# from scikit-learn import the TF-IDF vectoriser to be used for the n-grams feature
from sklearn.feature_extraction.text import TfidfVectorizer
# stats package will be used for feature selection
from scipy import stats
# pyplot will be used for creating boxplots
import matplotlib.pyplot as plt

# Load dataframe to be used for feature engineering

# load dataframe with no features (output by data preparation code)
### SET OWN PATH HERE
###
path = "/Users/artembutbaev/OneDrive/University of Bath 20-21 " + \
"(Year 4)/CM - Individual Project/2. Code/Yelp data/reviews.pkl"
###
###

df = pd.read_pickle(path)
```

```

###
# Extract Feature 1 - Review length (number of characters in review string)
###

number_of_reviews = len(df)
reviewLengthList = []
for review_index in range(number_of_reviews):
    length_of_review = len(df["reviewContent"].iloc[review_index])
    reviewLengthList.append(length_of_review)
df["reviewLength"] = reviewLengthList
df.reset_index(drop = True)

###
# Extract Feature 11 - Maximum number of reviews written by reviewer in a single day
###

# create series with all unique reviewerID values
unique_reviewers = pd.Series(df["reviewerID"].unique())
# create empty column to be used for filling in values
df["maxReviews"] = np.nan

for reviewer_index in range(0, len(unique_reviewers)):

    # 1. get maximum number of reviews posted in a single day for given reviewer
    max_reviews_reviewer = df["date"][df["reviewerID"] == unique_reviewers[reviewer_index]].value_counts()[0]
    # 2. update the column "maxReviews" for every row with the given reviewer
    df.loc[df.reviewerID == unique_reviewers[reviewer_index], "maxReviews"] = max_reviews_reviewer

# convert column to integers after all values filled in
df["maxReviews"] = df["maxReviews"].apply(np.int64)

###
# Extract Feature 2 - Average sentiment of review
###

# initialise Sentiment Intensity Analyzer (SIA)
```

```python
analyzer = SIA()
# create empty list to store all sentiment values
sentiment_list = []

# calculate for each review an average sentiment
# value based on individual sentences
for review in df['reviewContent']:

    # 1. split review into individual sentences (tokenise)
    review_sentences = tokenize.sent_tokenize(review)

    # 2. get the average sentiment for the review
    # (average sentiment of sentences in review)
    total_sentiment = 0.0
    for sentence in review_sentences:
        sentiment_score = analyzer.polarity_scores(sentence)
        total_sentiment += sentiment_score["compound"]

    # calculate average sentiment
    average_sentiment = round(total_sentiment / no_sentences, 4)

    # 3. append average sentiment to sentiment_list
    sentiment_list.append(average_sentiment)

# add new feature to dataframe
df["avgSentiment"] = sentiment_list

###
# Extract Features 3-8 - Parts of Speech (PoS)
###

# load parts of speech tagger from spaCy library
tagger = spacy.load("en_core_web_sm")

noun_proportion_list = []
adj_proportion_list = []
verb_proportion_list = []
```

```python
propnoun_proportion_list = []
num_count_list = []
symbol_count_list = []

for review in df['reviewContent']:

    # 1. tokenise and tag review
    tagged_review = tagger(review)

    # get count of tokens (words) in review
    token_count = len(tagged_review)

    noun_count = 0
    adj_count = 0
    verb_count = 0
    propnoun_count = 0
    num_count = 0
    symbol_count = 0

    # 2. Get counts of nouns, adjectives, verbs,
    # proper nouns, numbers and symbols
    for token in tagged_review:

        if(token.pos_ == "NOUN"):
            noun_count += 1
        if(token.pos_ == "ADJ"):
            adj_count += 1
        if(token.pos_ == "VERB"):
            verb_count += 1
        if(token.pos_ == "PROPN"):
            propnoun_count += 1
        if(token.pos_ == "NUM"):
            num_count += 1
        if(token.pos_ == "SYM"):
            symbol_count += 1

    # 3. append proportions to lists
    noun_proportion_list.append(noun_count / token_count)
    adj_proportion_list.append(adj_count / token_count)
    verb_proportion_list.append(verb_count / token_count)
    propnoun_proportion_list.append(propnoun_count / token_count)
    num_count_list.append(num_count)
    symbol_count_list.append(symbol_count)

# add new features to dataframe
df["nounProp"] = noun_proportion_list
df["adjProp"] = adj_proportion_list
df["verbProp"] = verb_proportion_list
df["propernounProp"] = propnoun_proportion_list
df["numCount"] = num_count_list
df["symCount"] = symbol_count_list
df = df.reset_index(drop = True)

###
# Extract Feature 9 - Maximum TF-IDF score for a given review
###

# create corpus - a list of all reviews
corpus = df["reviewContent"].tolist()
# initialise vectoriser and assign stop word list
# words appearing in more than 85% of documents will
# not be part of the vector
vectoriser = TfidfVectorizer(max_df = 0.85, stop_words = "english")
# fit and transform TF-IDF model on corpus
vocabulary = vectoriser.fit_transform(corpus)
max_tfidf_list = []

# loop through all reviews and
# retrieve their maximum TF-IDF value
for document_index in range(len(corpus)):

    max_value = vocabulary[document_index].max()
    max_tfidf_list.append(max_value)

```

```
# add new feature to dataframe
df["max_tfidf"] = max_tfidf_list

###
# Feature X - Reviewer membership length (skipped)
###

# length of time reviewer has been a member of the online platform (Yelp)
# this feature has been skipped because the reviewer table provided in the data is incomplete,
# with over 40,000 of the 60,019 reviewerIDs unmatched in the reviews table

###
# Feature X - Friend count (skipped)
###
# skipped due to same reason as the previous feature (unmatched reviewerIDs)

###
# Extract Feature 12 - Count of reviews posted by reviewer
###

# create series with all unique reviewerID values
unique_reviewers = pd.Series(df["reviewerID"].unique())
print("Number of unique reviewers: ", len(unique_reviewers))
# create empty column to be used for filling in values
df["postCount"] = np.nan

for reviewer_index in range(0, len(unique_reviewers)):

    # 1. get number of reviews posted by a given reviewer
    posts_by_reviewer = df["reviewerID"][df["reviewerID"] == unique_reviewers[reviewer_index]].value_counts().sum()

    # 2. update the column "postCount" for every row with the given reviewer
    df.loc[df.reviewerID == unique_reviewers[reviewer_index], "postCount"] = posts_by_reviewer

# convert column to integers after all values filled in
df["postCount"] = df["postCount"].apply(np.int64)

```

```
# Pickle (serialise) dataframe with all features ready to be used in model evaluation
df.to_pickle("./df3.pkl")

# Load dataframe for feature selection

# load dataframe with all features extracted
### SET OWN PATH HERE
###
path = "/Users/artembutbaev/OneDrive/University of Bath 20-21 " + \
"(Year 4)/CM - Individual Project/2. Code/Model Building/df4.pkl"
###
###

df = pd.read_pickle(path)

##########################

# Feature Selection

# Define functions to be used for feature selection

# print quartiles and spread of data points for a given feature
def print_quartiles(df, feature, bins_count):

    df_genuine = df[df["flagged"] == 'N']
    df_fake = df[df["flagged"] == 'Y']

    feature_genuine = df_genuine[feature]
    feature_fake = df_fake[feature]

    print(feature + " - Genuine reviews: ")
    print("Median (Q2): ", np.quantile(feature_genuine, .50))
    print("Q1: ", np.quantile(feature_genuine, .25))
    print("Q3: ", np.quantile(feature_genuine, .75))
    print("IQR: ", np.quantile(feature_genuine, .75) - np.quantile(feature_genuine, .25))
    print()
    print(feature + " - Fake reviews: ")
```

```python
    print("Median (Q2): ", np.quantile(feature_fake, .50))
    print("Q1: ", np.quantile(feature_fake, .25))
    print("Q3: ", np.quantile(feature_fake, .75))
    print("IQR: ", np.quantile(feature_fake, .75) - np.quantile(feature_fake, .25))

    # value counts (in bins) - use for bar charts to visually compare
    print("Total: ", len(feature_genuine))
    print("Value counts - Genuine reviews: ")
    print(feature_genuine.value_counts(bins=bins_count, dropna=False))
    print()
    print("Total: ", len(feature_fake))
    print("Value counts - Fake reviews: ")
    print(feature_fake.value_counts(bins=bins_count, dropna=False))

# create a boxplot for a given feature and an attribute to group it by (typically, the label)
def create_boxplot(df, feature, group_by):

    boxplot = df.boxplot(column=[feature], by=group_by, return_type=None, showfliers=False)
    figure = boxplot.get_figure()
    figure.suptitle('')
    plt.show()

# Print quartiles and create associated boxplots

# 1. reviewLength feature
print_quartiles(df, "reviewLength", 5)
create_boxplot(df, "reviewLength", "flagged")
# 2. maxReviews feature
print_quartiles(df, "maxReviews", 1)
create_boxplot(df, "maxReviews", "flagged")
# 3. avgSentiment feature
print_quartiles(df, "avgSentiment", 5)
create_boxplot(df, "avgSentiment", "flagged")
# 4. adjective proportion feature
print_quartiles(df, "adjProp", 10)
create_boxplot(df, "adjProp", "flagged")
# 5. max tf-idf feature
```

```
print_quartiles(df, "max_tfidf", 5)
create_boxplot(df, "max_tfidf", "flagged")
# 6. post count feature
print_quartiles(df, "postCount", 5)
create_boxplot(df, "postCount", "flagged")
# 7. noun proportion feature
print_quartiles(df, "nounProp", 5)
create_boxplot(df, "nounProp", "flagged")
# 8. verb proportion feature
print_quartiles(df, "verbProp", 5)
create_boxplot(df, "verbProp", "flagged")
# 9. propernoun proportion feature
print_quartiles(df, "propernounProp", 5)
create_boxplot(df, "propernounProp", "flagged")
# 10. numcount feature
print_quartiles(df, "numCount", 10)
create_boxplot(df, "numCount", "flagged")
# 11. symcount feature
print_quartiles(df, "symCount", 10)
create_boxplot(df, "symCount", "flagged")
# 12. hasProfile feature
print_quartiles(df, "hasProfile", 1)
create_boxplot(df, "hasProfile", "flagged")
# 13. usefulCount feature
print_quartiles(df, "usefulCount", 1)
create_boxplot(df, "usefulCount", "flagged")
# 14. coolCount feature
print_quartiles(df, "coolCount", 1)
create_boxplot(df, "coolCount", "flagged")
# 15. funnyCount feature
print_quartiles(df, "funnyCount", 1)
create_boxplot(df, "funnyCount", "flagged")
%matplotlib inline

# calculates point-biserial correlation between labels and a given feature
def point_biserial(labels, feature):

    # convert feature to numpy array
    feature = np.asarray(feature)

    # Output: -1 indicates a perfect negative association,
    # +1 indicates a perfect positive association, and 0 indicates no association
    #return stats.pointbiserialr(labels, feature)
    result, p_value = stats.pointbiserialr(labels, feature)
    return result

# Calculate point-biserial correlation for each feature and the associated labels
###

# convert labels to binary integers ("Y" = 1, "N" = 0)
labels = df["flagged"].tolist()
labels_binary = list(labels)
for label_index in range(len(labels)):
    if(labels[label_index] == 'Y'):
        labels_binary[label_index] = 1
    elif(labels[label_index] == 'N'):
        labels_binary[label_index] = 0
# convert labels list to numpy array
labels_binary = np.asarray(labels_binary)

# feature 1 - length of a review
feature_1 = df["reviewLength"].tolist()
print("reviewLength feature: ")
print(point_biserial(labels_binary, feature_1))
# feature 2 - maximum number of reviews by reviewer in a single day
feature_2 = df["maxReviews"].tolist()
print("maxReviews feature: ")
print(point_biserial(labels_binary, feature_2))
# feature 3 - average sentiment of a review
feature_3 = df["avgSentiment"].tolist()
print("avgSentiment feature: ")
print(point_biserial(labels_binary, feature_3))
# feature 4 - proportion of nouns
feature_4 = df["nounProp"].tolist()
```

```
print("nounProp feature: ")
print(point_biserial(labels_binary, feature_4))
# feature 5 - proportion of adjectives
feature_5 = df["adjProp"].tolist()
print("adjProp feature: ")
print(point_biserial(labels_binary, feature_5))
# feature 6 - proportion of verbs
feature_6 = df["verbProp"].tolist()
print("verbProp feature: ")
print(point_biserial(labels_binary, feature_6))
# feature 7 - proportion of proper nouns
feature_7 = df["propernounProp"].tolist()
print("propernounProp feature: ")
print(point_biserial(labels_binary, feature_7))
# feature 8 - max tf-idf
feature_8 = df["max_tfidf"].tolist()
print("max_tfidf feature: ")
print(point_biserial(labels_binary, feature_8))
# feature 9 - count of numbers in review
feature_9 = df["numCount"].tolist()
print("numCount feature: ")
print(point_biserial(labels_binary, feature_9))
# feature 10 - count of symbols in review
feature_10 = df["symCount"].tolist()
print("symCount feature: ")
print(point_biserial(labels_binary, feature_10))
# feature 11 - count of posts
feature_11 = df["postCount"].tolist()
print("postCount feature: ")
print(point_biserial(labels_binary, feature_11))
# feature 12 - hasProfile
feature_12 = df["hasProfile"].tolist()
print("hasProfile feature: ")
print(point_biserial(labels_binary, feature_12))
# feature 13 - usefulCount
feature_13 = df["usefulCount"].tolist()
print("usefulCount feature: ")
```

```python
print(point_biserial(labels_binary, feature_13))
# feature 14 - coolCount
feature_14 = df["coolCount"].tolist()
print("coolCount feature: ")
print(point_biserial(labels_binary, feature_14))
# feature 15 - funnyCount
feature_15 = df["funnyCount"].tolist()
print("funnyCount feature: ")
print(point_biserial(labels_binary, feature_15))

# Calculate t-test for groups of features for genuine and fake review
###

df_genuine = df[df["flagged"] == "N"]
df_fake = df[df["flagged"] == "Y"]

# 1. review length
feature_genuine = df_genuine["reviewLength"]
feature_fake = df_fake["reviewLength"]
print('\033[1m' + "1. Review length:" + '\033[0m')
statistic, p_value = stats.ttest_ind(feature_genuine, feature_fake)
print(statistic)
# 2. maximum reviews posted in a single day by reviewer
feature_genuine = df_genuine["maxReviews"]
feature_fake = df_fake["maxReviews"]
print('\033[1m' + "2. Maximum reviews posted by reviewer in a single day:" + '\033[0m')
statistic, p_value = stats.ttest_ind(feature_genuine, feature_fake)
print(statistic)
# 3. average sentiment of sentences in review
feature_genuine = df_genuine["avgSentiment"]
feature_fake = df_fake["avgSentiment"]
print('\033[1m' + "3. Average sentiment of sentences in review:" + '\033[0m')
statistic, p_value = stats.ttest_ind(feature_genuine, feature_fake)
print(statistic)
# 4. noun proportion out of all words in review
feature_genuine = df_genuine["nounProp"]
feature_fake = df_fake["nounProp"]
```

```
print('\033[1m' + "4. Noun proportion:" + '\033[0m')
statistic, p_value = stats.ttest_ind(feature_genuine, feature_fake)
print(statistic)
# 5. adjective proportion out of all words in review
feature_genuine = df_genuine["adjProp"]
feature_fake = df_fake["adjProp"]
print('\033[1m' + "5. Adjective proportion:" + '\033[0m')
statistic, p_value = stats.ttest_ind(feature_genuine, feature_fake)
print(statistic)
# 6. verb proportion out of all words in review
feature_genuine = df_genuine["verbProp"]
feature_fake = df_fake["verbProp"]
print('\033[1m' + "6. Verb proportion:" + '\033[0m')
statistic, p_value = stats.ttest_ind(feature_genuine, feature_fake)
print(statistic)
# 7. proper noun proportion out of all words in review
feature_genuine = df_genuine["propernounProp"]
feature_fake = df_fake["propernounProp"]
print('\033[1m' + "7. Proper noun proportion:" + '\033[0m')
statistic, p_value = stats.ttest_ind(feature_genuine, feature_fake)
print(statistic)
# 8. maximum tf-idf value in review
feature_genuine = df_genuine["max_tfidf"]
feature_fake = df_fake["max_tfidf"]
print('\033[1m' + "8. Maximum tf-idf value:" + '\033[0m')
statistic, p_value = stats.ttest_ind(feature_genuine, feature_fake)
print(statistic)
# 9. maximum tf-idf value in review
feature_genuine = df_genuine["postCount"]
feature_fake = df_fake["postCount"]
print('\033[1m' + "9. Post count:" + '\033[0m')
statistic, p_value = stats.ttest_ind(feature_genuine, feature_fake)
print(statistic)
# 10. symbol count in review
feature_genuine = df_genuine["symCount"]
feature_fake = df_fake["symCount"]
print('\033[1m' + "10. Symbol count:" + '\033[0m')
```

```
statistic, p_value = stats.ttest_ind(feature_genuine, feature_fake)
print(statistic)
# 11. number count in review
feature_genuine = df_genuine["numCount"]
feature_fake = df_fake["numCount"]
print('\033[1m' + "11. Number count:" + '\033[0m')
statistic, p_value = stats.ttest_ind(feature_genuine, feature_fake)
print(statistic)
# 12. HasProfile
feature_genuine = df_genuine["hasProfile"]
feature_fake = df_fake["hasProfile"]
print('\033[1m' + "12. Has reviewer got a profile:" + '\033[0m')
statistic, p_value = stats.ttest_ind(feature_genuine, feature_fake)
print(statistic)
# 13. usefulCount
feature_genuine = df_genuine["usefulCount"]
feature_fake = df_fake["usefulCount"]
print('\033[1m' + "13. Useful count:" + '\033[0m')
statistic, p_value = stats.ttest_ind(feature_genuine, feature_fake)
print(statistic)
# 14. coolCount
feature_genuine = df_genuine["coolCount"]
feature_fake = df_fake["coolCount"]
print('\033[1m' + "14. Cool count:" + '\033[0m')
statistic, p_value = stats.ttest_ind(feature_genuine, feature_fake)
print(statistic)
# 15. funnyCount
feature_genuine = df_genuine["funnyCount"]
feature_fake = df_fake["funnyCount"]
print('\033[1m' + "15. Funny count:" + '\033[0m')
statistic, p_value = stats.ttest_ind(feature_genuine, feature_fake)
print(statistic)
```

C.3 Model Evaluation: model_evaluation.py

```python
# Import required libraries

import pandas as pd
import numpy as np
import pickle
# import decision tree classifier
from sklearn.tree import DecisionTreeClassifier
# import functionality to split data into training and testing
from sklearn.model_selection import train_test_split
# import functionality for creating stratified k-fold cross validation
from sklearn.model_selection import StratifiedKFold
# import metrics module to construct confusion matrix
from sklearn import metrics

# function for balancing dataset using oversampling and undersampling
def resample(x_train, y_train):

    # 1. "combine" x_train and y_train
    x_train = x_train.copy()
    x_train["label"] = y_train

    # create a sub-dataframes
    genuine = x_train[x_train["label"] == "N"]
    fake = x_train[x_train["label"] == "Y"]

    # 1. oversampling

    # 2x oversampling
    fake_copy = fake.copy()
    fake = pd.concat([fake, fake_copy])

    # 2. undersampling

    # shuffle rows
    genuine = genuine.reindex(np.random.permutation(genuine.index))
    # under-sample majority class
    genuine = genuine[:len(fake)]

    # combine two sub-dataframes
    x_train = pd.concat([genuine, fake])
    x_train = x_train.reset_index(drop = True)

    # shuffle rows
    x_train = x_train.reindex(np.random.permutation(x_train.index))

    # 3. "separate" x_train and y_train again
    y_train = pd.Series(x_train["label"])
    x_train = x_train.drop("label", 1)

    return x_train, y_train

# Load dataframe for model evaluation

# SET PATH HERE FOR DATAFRAME WITH ALL FEATURES
# load dataframe with all 15 features ready
path = "/Users/artembutbaev/OneDrive/University of Bath 20-21 " + \
"(Year 4)/CM - Individual Project/2. Code/Model Building/df4.pkl"
df = pd.read_pickle(path)

# Fake and genuine review examples included in dissertation
# df[df['flagged'] == 'N'].iloc[2333].reviewContent
# df[df['flagged'] == 'Y'].iloc[233].reviewContent
# df

### Define feature sets

labels = pd.Series(df["flagged"])

# feature set 1 - review-centric features only
columns = [df["max_tfidf"], df["reviewLength"], df["numCount"],
            df["symCount"], df["adjProp"], df["avgSentiment"], df["nounProp"]]
headers = ["max_tfidf", "reviewLength", "numCount",
            "symCount", "adjProp", "avgSentiment", "nounProp"]
fs1 = pd.concat(columns, axis=1, keys=headers)

# feature set 2 - reviewer-centric features only
columns = [df["hasProfile"], df["postCount"], df["usefulCount"],
            df["coolCount"], df["funnyCount"], df["maxReviews"]]
headers = ["hasProfile", "postCount", "usefulCount",
            "coolCount", "funnyCount", "maxReviews"]
fs2 = pd.concat(columns, axis=1, keys=headers)

# feature set 3 - all features
columns = [df["max_tfidf"], df["reviewLength"], df["numCount"],
            df["symCount"], df["adjProp"], df["avgSentiment"], df["nounProp"],
            df["hasProfile"], df["postCount"], df["usefulCount"],
            df["coolCount"], df["funnyCount"], df["maxReviews"]]
headers = ["max_tfidf", "reviewLength", "numCount",
            "symCount", "adjProp", "avgSentiment", "nounProp",
            "hasProfile", "postCount", "usefulCount",
            "coolCount", "funnyCount", "maxReviews"]
fs3 = pd.concat(columns, axis=1, keys=headers)

# feature set 4 - both categories, top 10 overall based on correlation and t-test
columns = [df["hasProfile"], df["postCount"],
            df["usefulCount"], df["max_tfidf"], df["reviewLength"],
            df["coolCount"], df["funnyCount"], df["maxReviews"],
            df["numCount"], df["symCount"]]
headers = ["hasProfile", "postCount", "usefulCount", "max_tfidf",
            "reviewLength", "coolCount", "funnyCount",
            "maxReviews", "numCount", "symCount"]
fs4 = pd.concat(columns, axis=1, keys=headers)

# feature set 5 - both categories, top 5 overall based on correlation and t-test
columns = [df["hasProfile"], df["postCount"],
            df["usefulCount"], df["max_tfidf"], df["reviewLength"]]
headers = ["hasProfile", "postCount", "usefulCount",
           "max_tfidf", "reviewLength"]
fs5 = pd.concat(columns, axis=1, keys=headers)

# feature set 6 - both categories, top 5 overall based on feature
# importance of model trained on FS3 (all features)
columns = [df["usefulCount"], df["postCount"],
           df["avgSentiment"], df["reviewLength"], df["max_tfidf"]]
headers = ["usefulCount", "postCount",
           "avgSentiment", "reviewLength", "max_tfidf"]
fs6 = pd.concat(columns, axis=1, keys=headers)

# feature set 7 - both categories, top 10 overall based on feature
# importance of model trained on FS3 (all features)
columns = [df["usefulCount"], df["postCount"], df["avgSentiment"],
           df["reviewLength"], df["max_tfidf"], df["nounProp"],
           df["adjProp"], df["hasProfile"], df["numCount"], df["coolCount"]]
headers = ["usefulCount", "postCount", "avgSentiment", "reviewLength",
           "max_tfidf", "nounProp", "adjProp", "hasProfile", "numCount", "coolCount"]
fs7 = pd.concat(columns, axis=1, keys=headers)

###
# Evaluate average model performance using stratified 10-fold cross-validation
###

###
# SPECIFY FEATURE SET HERE
features = fs4
###

max_depth = 6
number_of_folds = 10
skf = StratifiedKFold(n_splits = number_of_folds, shuffle = True)
skf.get_n_splits(features, labels)
print(skf)
count = 1

acc_sum = 0
```

```python
precision_sum = 0
recall_sum = 0
f_sum = 0
balanced_sum = 0

# loop through train/test data splits
for train_index, test_index in skf.split(features, labels):
    x_train, x_test = features.iloc[train_index], features.iloc[test_index]
    y_train, y_test = labels.iloc[train_index], labels.iloc[test_index]

    ###
    # 1. re-sample training set ONLY (keep test data class distribution the same)
    ###
    # Comment out the following line to train model on natural
    # class distribution of fake/genuine reviews
    x_train, y_train = resample(x_train, y_train)

    # check effect of resampling on the number of fake
    # and genuine review samples in training dataset
    #print("Fake reviews in training set:")
    #print(len(y_train[y_train == 'Y']))
    #print("Genuine reviews in training set:")
    #print(len(y_train[y_train == 'N']))
    #print()

    ###
    # 2. train classifier
    ###
    # use the following line for training a fully grown
    # tree with no hyper-parameter tuning
    # dt = DecisionTreeClassifier()
    # use the following line for training a tree with
    # hyper-parameter tuning applied
    dt = DecisionTreeClassifier(max_depth = max_depth)

    # build decision tree from training data
    dt.fit(x_train, y_train)

    ###
    # 3. predict test data
    predictions = dt.predict(x_test)
    ###

    # create confusion matrix based on predictions
    # vs. actual labels of reviews
    confusion_matrix = metrics.confusion_matrix(y_test, predictions, labels=['N','Y'])
    tn, fp, fn, tp = metrics.confusion_matrix(y_test, predictions).ravel()

    ###
    # 4. calculate evaluation metrics
    ###

    # a. accuracy
    accuracy = (tp + tn) / (tp + tn + fp + fn)
    # b. precision
    if(tp == 0 and fp == 0):
        precision = 0
    else:
        precision = tp / (tp + fp)
    # c. recall
    if(tp == 0 and fn == 0):
        recall = 0
    else:
        recall = tp / (tp + fn)
    # d. f score
    if(precision == 0 and recall == 0):
        f_score = 0
    else:
        f_score = (2 * precision * recall) / (precision + recall)
    if(tn == 0 and fp == 0):
        true_negative_rate = 0
    else:
        true_negative_rate = tn / (tn + fp)
    # e. balanced accuracy
    balanced_accuracy = (true_negative_rate + recall) / 2

    # get feature importances - Gini importance
    # it represents the normalised total reduction of the criterion by a given feature
    #importances = dict(zip(features.columns, dt.feature_importances_))
    #feature_importances = dt.feature_importances_
    #for index in range(len(feature_importances)):
        #list_of_feature_importance[index] += feature_importances[index]
        #print(feature_importances[index])
    #print()

    print("Run ", count, ": ")
    print("Overall accuracy: ", accuracy * 100)
    print("Balanced accuracy: ", balanced_accuracy * 100)
    print("Precision: ", precision * 100)
    print("Recall: ", recall * 100)
    print("TNR: ", true_negative_rate * 100)
    print("F Score: ", f_score * 100)
    print()
    count += 1

    acc_sum += accuracy
    precision_sum += precision
    recall_sum += recall
    f_sum += f_score
    balanced_sum += balanced_accuracy

print()
print("Average accuracy over", number_of_folds, "runs: ",
      round(acc_sum / number_of_folds * 100, 2))
print("Average balanced accuracy over", number_of_folds, "runs: ",
      round(balanced_sum / number_of_folds * 100, 2))
print("Average precision over", number_of_folds, "runs: ",
      round(precision_sum / number_of_folds * 100, 2))
print("Average recall over", number_of_folds, "runs: ",
      round(recall_sum / number_of_folds * 100, 2))
print("Average F score over", number_of_folds, "runs: ",
        round(f_sum / number_of_folds * 100, 2))
print()

# get average feature importance of each feature during k-fold CV
#for index in range(len(list_of_feature_importance)):
    #list_of_feature_importance[index] = list_of_feature_importance[index] / number_of_folds


###
# Hyper-parameter optimisation (max_depth feature)
###


###
# SPECIFY FEATURE SET HERE
features = fs4
###

number_of_folds = 10
skf = StratifiedKFold(n_splits = number_of_folds, shuffle = True)
skf.get_n_splits(features, labels)

# specify range of values here to test for hyper-parameter optimisation
max_depth_range = (range(3,26))
max_depth_avg_f1 = []

for max_depth in max_depth_range:

    print("Performing 10-fold cross-validation for max_depth: ", max_depth)

    f_sum = 0

    # loop through train/test data splits
    for train_index, test_index in skf.split(features, labels):
        x_train, x_test = features.iloc[train_index], features.iloc[test_index]
        y_train, y_test = labels.iloc[train_index], labels.iloc[test_index]

        # balance training set
        x_train, y_train = resample(x_train, y_train)

        dt = DecisionTreeClassifier(max_depth = max_depth)
        dt.fit(x_train, y_train)

        predictions = dt.predict(x_test)
        confusion_matrix = metrics.confusion_matrix(y_test, predictions, labels=['N','Y'])
        tn, fp, fn, tp = metrics.confusion_matrix(y_test, predictions).ravel()

        # b. precision
        if(tp == 0 and fp == 0):
            precision = 0
        else:
            precision = tp / (tp + fp)
        # c. recall
        if(tp == 0 and fn == 0):
            recall = 0
        else:
            recall = tp / (tp + fn)
        # d. f score
        if(precision == 0 and recall == 0):
            f_score = 0
        else:
            f_score = (2 * precision * recall) / (precision + recall)

        f_sum += f_score
    avg_score = round(f_sum / number_of_folds * 100, 2)
    #print("Average F score over", number_of_folds, " + \
    #"runs for max_depth", max_depth, "was:", avg_score)
    #print()
    #print("...")
    #print()
    max_depth_avg_f1.append(avg_score)

scores = dict(zip(max_depth_range, max_depth_avg_f1))
print(scores)
```

```

# plot the average F1 score against the max_depth value on a scatter plot
import matplotlib.pyplot as plt
plt.scatter(scores.keys(), scores.values())
plt.show()

###
# get feature importances and decision tree characteristics
###

# Print feature importances in construction of tree

# Also known as Gini importance, it represents the normalised total
# reduction of the criterion by a given feature
# importances add up to 1.0
importances = dict(zip(features.columns, dt.feature_importances_))
for value in importances:
    print(value, ":", importances[value])
print()

# Print decision tree characteristics

print("decision tree characteristics: ")
print("depth of decision tree: ", dt.get_depth())
print("number of leaves: ", dt.get_n_leaves())
print()
dt.get_params(deep=True)

#### Serialise classifier

# Serialise decision tree classifier and train/test
# data and test reviews for front end

# serialise classifier
with open('dt_final.pkl', 'wb') as f:
    pickle.dump(dt, f)

```

```
# serialise train/test data
with open('x_train.pkl', 'wb') as f:
    pickle.dump(x_train, f)
with open('x_test.pkl', 'wb') as f:
    pickle.dump(x_test, f)
with open('y_train.pkl', 'wb') as f:
    pickle.dump(y_train, f)
with open('y_test.pkl', 'wb') as f:
    pickle.dump(y_test, f)

# serialise reviews associated with test data
df1 = df[df.index.isin(x_test.index)]
with open('df_test.pkl', 'wb') as f:
    pickle.dump(df1, f)
```

C.4 User Interface v1.0: ui_v1.0.py

```
# Import required libraries

import numpy as np
import pickle
from sklearn import tree
from matplotlib import pyplot as plt
# import graphviz - library which will be used to visualise the decision tree and decision paths
import graphviz
# import pydotplus - an interface to the DOT language used by graphviz, which will be used to modify graphs
import pydotplus
# import webbrowser - will be used to open a new tab with the content of the HTML file created
import webbrowser
# import datetime - for including the current date and time in the HTML file generated
```

```
import datetime
# import functionality to get the current directory path
from pathlib import Path

# Define functions for getting the decision path for a given review and to create HTML output

# creates a diagram of the decision tree with the decision path for a given review highlighted and saves it as a png
def create_decision_path_png(dt, feature_names, filename, sample):

    # export decision tree in DOT (graphviz) format
    dot_data = tree.export_graphviz(dt, out_file=None,
                                feature_names=feature_names, class_names=dt.classes_,
                                filled=True, rounded=True, special_characters=True, node_ids = True)

    graph = pydotplus.graph_from_dot_data(dot_data)

    # set all nodes to be of white colour
    for node in graph.get_node_list():
        node.set_fillcolor('white')

    # get decision path for chosen review sample
    decision_path = dt.decision_path(sample)

    # loop through each node in tree and its associated decision-path value,
    # i.e., if the node is part of the decision path then it has a value of 1, otherwise, it has a value of 0
    for node_counter, node_value in enumerate(decision_path.toarray()[0]):

        # if node is not part of decision path for sample, continue
        if(node_value == 0):
            continue
        # update colour of nodes part of decision path to be of yellow colour
        dot_node = graph.get_node(str(node_counter))[0]
        dot_node.set_fillcolor('yellow')

    # export updated graph to a png
    graph.write_png(filename)

```

```
# gets for a sample review the decision path in text form
# inspiration from:
# https://scikit-learn.org/stable/auto_examples/tree/plot_unveil_tree_structure.html#decision-path
def get_decision_path_text(classifier, feature_names, sample):

    # retrieve probabilities for classification for sample
    probabilities = classifier.predict_proba(sample)
    # retrieve decision paths for test sample
    node_indicator = classifier.decision_path(sample)
    # retrieve leaf IDs reached by test sample
    leaf_identifier = classifier.apply(sample)

    # get list of feature names
    feature_names = feature_names
    # get a list of all thresholds (of all nodes in tree)
    threshold = classifier.tree_.threshold
    # get node id of leaf that the sample "lands on"
    leaf_id = leaf_identifier[0]
    # obtain ids of the nodes sample goes through
    nodes_traversed = node_indicator.indices[node_indicator.indptr[0]:
                                        node_indicator.indptr[1]]

    decision_counter = 0
    decisions_made = []

    for node_id in nodes_traversed:

        # increment counter
        decision_counter += 1

        # break for loop if it is a leaf node
        if(node_id == leaf_identifier[0]):
            leaf_node_string = "Leaf node ({node}) reached. End of decision path.".format(node = node_id)
            decisions_made.append(leaf_node_string)
            break

        # check if value of the split feature for sample i is below threshold
```

```python
        if (sample.iloc[0][classifier.tree_.feature[node_id]] <= threshold[node_id]):
            threshold_sign = "<="
        else:
            threshold_sign = ">"

        current_decision = "decision {decision_no} (at node {node}) : {feature_name} = {value} {inequality} {threshold}".format(
                  decision_no=decision_counter,
                  node=node_id,
                  feature_name=feature_names[classifier.tree_.feature[node_id]],
                  value=round(sample.iloc[0][classifier.tree_.feature[node_id]], 3),
                  inequality=threshold_sign,
                  threshold=round(threshold[node_id], 3))

        decisions_made.append(current_decision)

    # get predicted classification of sample based on samples at the leaf node
    predicted_classification = classifier.classes_[np.argmax(classifier.tree_.value[leaf_id])]
    class_proportion = max(classifier.tree_.value[leaf_id])
    class_probability = round(100 * max(probabilities[0]), 1)

    # get the predicted classification and confidence probability
    if(predicted_classification == "N"):
        predicted_string = "This review has been classified as Genuine with confidence probability: {prob}%".format(prob=class_probability)
    else:
        predicted_string = "This review has been classified as Fake with confidence probability: {prob}%".format(prob=class_probability)

    return decisions_made, predicted_string

# prepares the string to wrap in the HTML document to be displayed to the user
def prepare_string_to_wrap(index, review_text, decisions_made, predicted_string):

    # prepare string to wrap in html document

    string_to_wrap = ""

    # add title
    index_string = "Decision path for sample " + str(index) + ":"
    string_to_wrap += index_string
    string_to_wrap += "<br/><br/>"

    # add review text
    string_to_wrap += review_text
    string_to_wrap += "<br/><br/>"

    # add decisions made to string
    for decision in decisions_made:
        string_to_wrap += decision
        string_to_wrap += "<br/>"

    # add classification and accuracy to string
    string_to_wrap += "<br/>"
    string_to_wrap += predicted_string
    string_to_wrap += "<br/>"

    return string_to_wrap

# wraps decision path text, decision path image and other information in an HTML document
# and displays it in the users browser
def wrapOutputInHTML(title, body, image):

    now = datetime.datetime.today().strftime("%d/%m/%Y - %H:%M:%S")

    filename = title + '.html'
    f = open(filename,'w')

    # set style for image
    style = "height:75%;"

    wrapper = """<html>
    <head>
    <title>%s - %s</title>
    </head>
    <body><p>%s</p>
    <a href="decision_path_1.png" download></body>
    <img src=%s style=%s>
    </html>"""

    whole = wrapper % (title, now, body, image, style)
    f.write(whole)
    f.close()

    path = "file://" + str(Path().absolute()) + "/%s"
    path = path % (filename)

    return path

# Load decision tree classifier, test data and associated reviews

###
# SET PATH FOR CLASSIFIER AND TEST DATA HERE
path = "/Users/artembutbaev/OneDrive/University of Bath 20-21 " + \
"(Year 4)/CM - Individual Project/2. Code/Model Building/"
###

# load decision tree classifier
with open(path + 'dt_final.pkl', 'rb') as f:
    dt = pickle.load(f)

# load test data
with open(path + 'x_test.pkl', 'rb') as f:
    x_test = pickle.load(f)

# load associated reviews with test data
with open(path + 'df_test.pkl', 'rb') as f:
    df_test = pickle.load(f)


# Main user input loop - user selects a review for which to show its decision path

while(True):

```

```python
    print("Enter a review index from 0 to", len(x_test), ":")
    user_input = input()
    index_chosen = int(user_input)

    review_text = df_test["reviewContent"].iloc[index_chosen]
    print("Is this the review you want to classify? (y/n)\n")
    print(review_text)
    user_input_2 = input()

    if(user_input_2 == 'y'):
        print("Generating output...")
        # get review sample from test data based on chosen index
        sample = x_test.iloc[index_chosen:index_chosen+1]

        # 1. get review text for chosen index
        review_text = df_test["reviewContent"].iloc[index_chosen]

        # 2. get decision path (text)
        decisions_made, predicted_string = get_decision_path_text(dt, x_test.columns, sample)

        # 3. get decision path (visual - save as png)
        create_decision_path_png(dt, x_test.columns, 'decision_path_1.png', sample)

        # 4. embed review text/decision path/visual in HTML document and open in browser
        string_to_wrap = prepare_string_to_wrap(index_chosen, review_text, decisions_made, predicted_string)
        image_to_wrap = "decision_path_1.png"
        path = wrapOutputInHTML("output1", string_to_wrap, image_to_wrap)
        webbrowser.open_new_tab(path)

        break
```

C.5 User Interface v2.0: ui_v2.0.py

```python
# Import required libraries

# pickle will be used to load the saved classifier and
# test data to be used for the front end
import pickle
# scikit-learn will be used for getting the decision path
# of samples and generating the tree diagram
from sklearn import tree
# pydotplus will be used to create an interface to the
# DOT language which will be used to modify the tree diagram
import pydotplus
# webbrowser will be used to open a new tab with the
# content of the HTML file created
import webbrowser
# import datetime - for including the current date
# and time in the HTML file generated
import datetime
# import functionality to get the current directory path
from pathlib import Path

####################################################
####################################################


# Define functions to be used for generating the front end

# function 1 - generates for a sample review the decision path information
# inspiration from:
# https://scikit-learn.org/stable/auto_examples/tree/plot_unveil_tree_structure.html#decision-path
def get_decision_path_info(classifier, feature_names, sample):

    # retrieve probabilities for classification for sample
    probabilities = classifier.predict_proba(sample)
    # retrieve decision paths for test sample
    node_indicator = classifier.decision_path(sample)
    # retrieve leaf IDs reached by test sample
    leaf_identifier = classifier.apply(sample)
    # get a list of all thresholds (of all nodes in tree)
    threshold = classifier.tree_.threshold
    # get node id of leaf that the sample "lands on"
    leaf_id = leaf_identifier[0]
    # obtain ids of the nodes sample goes through
    nodes_traversed = node_indicator.indices[node_indicator.indptr[0]:
                                    node_indicator.indptr[1]]

    features_used = []
    threshold_values = []
    threshold_signs = []

    for node_id in nodes_traversed:

        # break for loop
        if(node_id == leaf_id):
            break

        # check if value of the split feature for sample i is below threshold
        if (sample.iloc[0][classifier.tree_.feature[node_id]] <= threshold[node_id]):
            threshold_sign = "<="
        else:
            threshold_sign = ">"

        features_used.append(feature_names[classifier.tree_.feature[node_id]])
        #thresholds.append(threshold_sign + " " + str(round(threshold[node_id], 3)))
        threshold_values.append(round(threshold[node_id], 3))
        threshold_signs.append(threshold_sign)

    # get proportions of labels at leaf node and calculate probabilities of the output
    class_proportion = classifier.tree_.value[leaf_id]
    class_probability = round(100 * max(probabilities[0]), 1)
    samples_at_leaf = int(class_proportion[0][0]) + int(class_proportion[0][1])
    genuine_probability = round(100 * probabilities[0][0], 1)
    fake_probability = round(100 * probabilities[0][1], 1)

    predicted_string = "According to the model, this review is likely to be Genuine " \
    "with probability {prob1}% and Fake with probability {prob2}%. " \
    .format(prob1 = genuine_probability, prob2 = fake_probability)
    predicted_string += "These probabilities are based on " + str(samples_at_leaf) + \
    " other reviews with the same decision path."

    return features_used, threshold_values, threshold_signs, predicted_string

# function 2 - this function writes the series of decisions in a natural language form
# based on the features used and their associated thresholds
# in the decision path
# returns the series of decisions and the
# features not used in the decision path
def write_decisions(features_used, feature_names, threshold_values, threshold_signs):

    unique_features = []
    features_not_used = []
    list_of_statements = []

    # create list of unique features (in the order they appear in decision path)
    # (not efficient but our lists will be very small)
    for feature in features_used:
        if(feature not in unique_features):
            unique_features.append(feature)

    # create list of features not used in decision path
    # (not efficient but our lists will be very small)
    for feature in feature_names:
        if(feature not in unique_features):
            features_not_used.append(feature)

    # create a dict with all unique features and the number of times
    # they appear in the decision path
    feature_counts = dict((element,0) for element in unique_features)

    # loop through each decision
    for index in range(len(features_used)):

        # update feature count in dict
        feature_counts[features_used[index]] += 1

        # handle case when feature is 'hasProfile'
        if(features_used[index] == "Does the reviewer have a profile?"):
            if(threshold_signs[index] == ">"):
                list_of_statements.append("The reviewer has a profile")
            elif(threshold_signs[index] == "<="):
                list_of_statements.append("The reviewer does not have a profile")
            continue
        # handle the cases where the threshold should
        # be rounded to an integer
        # i.e., when the feature is not max_tfidf or 'hasProfile'
        elif(features_used[index] != "Maximum word importance score (TF-IDF)"):

            # if threshold sign is '>', add 0.5 to the value and change sign to '>='
            if(threshold_signs[index] == ">"):
                threshold_values[index] += 0.5
                threshold_signs[index] = ">="
            # if threshold sign is '<=', subtract 0.5 from the value and keep the same sign
            elif(threshold_signs[index] == "<="):
                threshold_values[index] -= 0.5

            # if threshold sign is '<=' and value is 0, update sign to be '='
            if(threshold_signs[index] == "<=" and threshold_values[index] == 0):
                threshold_signs[index] = "="

        # create statements based on feature counts
        if(feature_counts[features_used[index]] == 1):
            list_of_statements.append(features_used[index] \
                                      + " " + \
                                      threshold_signs[index] \
                                      + " " + str(int(threshold_values[index])))
        else:
            list_of_statements[unique_features.index(features_used[index])] \
            += " and " + threshold_signs[index] + " " + str(int(threshold_values[index]))

    return list_of_statements, features_not_used

# function 3 - creates a diagram of the decision tree with the decision path
# for a given review highlighted and saves it as a png
def create_tree_diagram_png(dt, feature_names, class_names, filename, sample):

    # get decision tree classifier in DOT (graphviz) format
    dot_data = tree.export_graphviz(dt, out_file=None,
                                feature_names=feature_names,
                                    class_names=class_names, filled=True,
                                    rounded=True, special_characters=True,
                                    node_ids = False)

    graph = pydotplus.graph_from_dot_data(dot_data)

    # set all nodes to be of a white colour
    for node in graph.get_node_list():
        node.set_fillcolor('white')

    # set colour of leaf nodes depending on their class (fake or genuine)
    for node in graph.get_node_list():
        d = node.get_attributes()
        if("label" in d):
            labels = node.get_attributes()['label'].split('<br/>')
            for i, label in enumerate(labels):
                if(label.startswith('class = ')):
                    # colour nodes depending on their class
                    if('Fake' in label):
                        # set colour of nodes with fake as majority vote
                        node.set_fillcolor('gray75')
                    elif('Genuine' in label):
                        # set colour of nodes with genuine as majority vote
                        node.set_fillcolor('gray97')

    # get decision path for chosen review sample
    decision_path = dt.decision_path(sample)

    # loop through each node in tree and its associated decision-path value,
    # i.e., if the node is part of the decision path
    # then it has a value of 1, otherwise, it has a value of 0
    for node_counter, node_value in enumerate(decision_path.toarray()[0]):

        # if node is not part of decision path for sample, continue
        if(node_value == 0):
            continue
        # set colour of node part of decision path for sample
        dot_node = graph.get_node(str(node_counter))[0]
        dot_node.set_fillcolor('darkseagreen1')

    # export updated graph to a png
    graph.write_png(filename)

# function 4 - prepares the string to wrap in the
# HTML document to be displayed to the user
def prepare_string_to_wrap(index, review_text, reviewer_id,
                           decisions_made, features_not_used, predicted_string):

    string_to_wrap = ""

    # add title
    index_string = "<h2>Classification Explanation For Selected Review</h2>"
    string_to_wrap += index_string
    string_to_wrap += "<br/>"

    # add review text
    string_to_wrap += "<em>" + review_text + "</em>"
    string_to_wrap += "<br/>"

    # add reviewer ID
    string_to_wrap += "- posted by reviewer: " + reviewer_id
    string_to_wrap += "<br/><br/>"

    # add classification and accuracy to string
    string_to_wrap += "<mark>" + predicted_string + "</mark>"
    string_to_wrap += "<br/>"

    # add line to separate classification from decisions made
    string_to_wrap += "<hr>"

    # add decisions made to string
    string_to_wrap += "<br/>"
    string_to_wrap += "This estimation is based on the model's " + \
    "consideration of the following attributes of the review and the reviewer:<br/>"
    string_to_wrap += "<ul>"
    for decision in decisions_made:
        string_to_wrap += "<li>"
        string_to_wrap += decision
        string_to_wrap += "</li>"
    string_to_wrap += "</ul>"
    #string_to_wrap += "<br/>"

    # add info about features NOT considered in decisions
    string_to_wrap += "The following features were not " + \
    "considered in the decision-making process for this review:"
    string_to_wrap += "<ul>"
    for feature in features_not_used:
        string_to_wrap += "<li>"
        string_to_wrap += feature
        string_to_wrap += "</li>"
    string_to_wrap += "</ul>"

    # add information about visualisation
    string_to_wrap += "This reasoning can also be " + \
    "seen visually in the visualisation below."

    return string_to_wrap

```

```
# function 5 - wraps decision path text, decision path image
# and other information in an HTML document and displays it in the users browser
def wrap_output_in_HTML(title, text, image):

    current_datetime = datetime.datetime.today().strftime("%d/%m/%Y - %H:%M:%S")

    # create a new HTML file
    filename = title + '.html'
    f = open(filename,'w')

    # create basic HTML structure with indications
    # of where strings will be embedded
    wrapper = """<html>
    <head>
    <style>
    body {
        background-color: WhiteSmoke;
    }
    </style>
    <title>%s - %s</title>
    </head>
    <body>

    <p>%s</p>
    <a href=%s target="_blank">Click here to open visualisation</a>
    </body>
    </html>"""

    # embed the document title, current datetime, decision path text
    # and tree visualisation file location in the HTML document
    whole = wrapper % (title, current_datetime, text, image)
    f.write(whole)
    f.close()

    path = "file://" + str(Path().absolute()) + "/%s"
    path = path % (filename)

    return path

######################################################
######################################################


# Main code with user input loop

###
# SET PATH FOR CLASSIFIER AND TEST DATA HERE
path = "/Users/artembutbaev/OneDrive/University of Bath 20-21 " + \
"(Year 4)/CM - Individual Project/2. Code/Model Building/"
###

# load decision tree classifier from path above
with open(path + 'dt_final.pkl', 'rb') as f:
    dt_final = pickle.load(f)
# load test data (features only) from path above
with open(path + 'x_test.pkl', 'rb') as f:
    x_test = pickle.load(f)
# load associated review data from path above
with open(path + 'df_test.pkl', 'rb') as f:
    df_test = pickle.load(f)

# set feature names to be displayed in front end
feature_names_readable = ['Does the reviewer have a profile?', 'Posts by reviewer', \
                          '\'Useful\' votes', 'Maximum word importance score (TF-IDF)', \
                          'Length of review (characters)', '\'Cool\' votes', \
                          '\'Funny\' votes', 'Maximum posts by reviewer in a single day', \
                          'Count of numbers in review', 'Count of symbols in review']
# set output class names to be displayed in front end
class_names_readable = ['Genuine', 'Fake']

# User input loop - user selects a review they would like classified and explained
# an explanation and tree diagram is generated, embedded within an HTML document
# and displayed in the user's web-browser in a new tab
while(True):

    print("Enter a review index from 0 to", len(x_test) - 1, ":")
    user_input = input()
    index_chosen = int(user_input)
    review_text = df_test["reviewContent"].iloc[index_chosen]
    print("Is this the review you want to classify? (y/n)\n")
    print(review_text)
    user_input_2 = input()

    if(user_input_2 == 'y'):
        print("Generating output...")

        # 1. get review sample from test data for chosen index
        # and review text and reviewer ID from review data
        sample = x_test.iloc[index_chosen:index_chosen+1]
        review_text = df_test["reviewContent"].iloc[index_chosen]
        reviewer_id = df_test["reviewerID"].iloc[index_chosen]

        # 2. get decision path information (features used and the threshold values)
        features_used, threshold_values, threshold_signs, predicted_string = \
        get_decision_path_info(dt_final, feature_names_readable, sample)
        # write the decision path information
        # as a series of decisions in text form, detailing
        # the features used and not used in the model's decision making
        decisions_made, not_used = \
        write_decisions(features_used, feature_names_readable, threshold_values, threshold_signs)

        # 3. generate tree diagram with highlighted decision path (save as png)
        create_tree_diagram_png(dt_final, x_test.columns, \
                                 class_names_readable, 'decision_path_1.png', sample)

        # 4. prepare the written explanation to wrap in the HTML document
        string_to_wrap = prepare_string_to_wrap(index_chosen, review_text, reviewer_id,
                                                decisions_made, not_used, predicted_string)
        image_to_wrap = "decision_path_1.png"

        # 5. embed the written explanation and tree diagram in HTML document
        path = wrap_output_in_HTML("output1", string_to_wrap, image_to_wrap)
        # open the prepared HTML document in the user's web browser
        webbrowser.open_new_tab(path)

        break
```