VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wānanga o te Ūpoko o te Ika a Māui

School of Engineering and Computer Science
Te Kura Mātai Pūkaha, Pūrorohiko

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Internet: office@ecs.vuw.ac.nz

Creating Counterfactuals for Text Analysis (eXplainable AI)

Michael Blayney

Supervisors: Andrew Lensen and Hayden Andersen

Submitted in partial fulfilment of the requirements for
Bachelor of Engineering with Honours.

Abstract

Text-based machine learning classification models attempt to assign given text samples into a set of distinct classes. Many of the decisions made by these machine learning models are complex and difficult for humans to understand. A counterfactual statement is a conditional one that explores what hasn't happened, or what would be true under different circumstances. Inferences made by inspecting counterfactual statements can be useful for understanding complicated black-box machine learning models. There are several existing tools that can automatically generate counterfactual statements for text-based data, however, many of them attempt to maximise the number of statements generated and do not emphasise the quality of output to a significant extent. This project involves developing a framework that automatically generates counterfactual statements by modifying original text samples as little as possible, such that the final statements maximally resemble their originals. The results show that focusing on minimal alterations will inevitably reduce the success rate of generation, however, the final counterfactual statements are similar to the original texts and are of high quality.

Acknowledgments

I would like to thank my supervisors, Andrew Lensen and Hayden Andersen for not only their academic mentorship and guidance, but also for their personal support during the exceptionally difficult second half of this year.

Contents

Chapter 1

Introduction

In the modern day, machine learning model performances are increasing at an incredible rate. However, this increased performance generally does come with increased model complexity. A majority of current successful machine learning techniques take the form of complex black-box models, which train and produce predictions with limited human understanding [1]. More simplistic statistical models map inputs to outputs in an understandable and predictable way, and as such, can be comprehended and explained more easily.

An interpretable model is one where its processes are easily understood, producing expected and consistent results that can be predicted by human users [2]. The field of Explainable Artificial Intelligence (XAI) seeks to provide tools and techniques to peer inside complex black-box processes and increase interpretability [2]. The main goal of XAI is to produce models with outputs and predictions that can be more easily explained while retaining the increased performance benefits over more transparent statistical models.

1.1 Motivations

The concept of white and black-box in the context of machine learning models is similar to their more popular namesakes in the world of software testing. White-box testing refers to tests created with internal software implementation in mind, while black-box testing is blind to implementation and instead is based on testing external behaviours of a system. White-box machine learning models are ones where behaviours can be inspected and tracked, with inputs being transformed in understandable ways by relatively basic algorithms [3]. The behaviour of white-box models can be explained consistently due to their relative simplicity, allowing humans to track input transformations and accurately predict outputs. When a machine learning model is easily understood, they are labelled as "interpretable" [4].

Black-box models, on the other hand, are much less inherently understandable. When looking at models that rely on large neural networks, for example, it is generally unfeasible or logistically impossible to manually map inputs to produced outputs or to verify correct internal behaviour. An important note is that XAI techniques do not increase the transparency of black-box models to that seen in white-box ones, but instead they provide tools and auxiliary algorithms to approximate understandable functions which estimate produced outputs closely. When a machine learning model is unable to be readily understood by humans, but can be comprehended using additional tools and descriptions, they are labelled as "explainable"[4].

Many highly impactful calculations are currently handled by these loosely understood black-box machine learning processes [5]. It is difficult for people to place trust in systems that they inherently cannot understand. This problem is amplified when the results of these

complicated models are used for important and critical decisions such as medical diagnoses [4]. For a piece of software to be trusted it likely has to exhibit high amounts of robustness, which is generally assessed through testing if similar input conditions produce similar results [5]. This approach is inherently infeasible for black-box machine learning models due to the fact that small alterations to inputs can produce vastly different, relatively untraceable outputs and results. Accountability is another concern for black-box models, seeing as errors are often difficult to delineate, leading to blaming a specific individual or implementation step for any given mistake near impossible.

1.1.1 Counterfactual Statements

As outlined in my motivations, black-box models that exclusively generate outputs with little human understanding can prove to be problematic and require additional tools to comprehend. One XAI strategy is to produce alternative, modified versions of model inputs that produce different predicted outputs. These modified inputs are known as counterfactual statements and can provide context and insight into black-box models. A common example of a counterfactual statement describes a person whose request for a bank loan has been rejected [6]. If a bank customer earns $40,000 per year, and the bank requires a minimum of $50,000 per year to issue a loan, the request will be rejected. Rather than focusing on the statement of decision, "Your income is not sufficient for this loan", the counterfactual statement would be "If your income was $50,000 pa we could have issued you a loan". After a counterfactual statement is generated, it is up to the human user to interpret and make conclusions based on this altered output, as the counterfactual statement itself does not explicitly specify or explain a model's inner workings.

Very few tools for text-based counterfactual generation exist currently [7], with most if not all still in their infancy. Most existing frameworks and current research are based on generating counterfactual statements for tabular data, and as such, the specifics of text-based generation are limited. With text-based counterfactual statement generation, there are a few interesting things to consider. Continuous numerical or discrete categorical data, as seen in tabular datasets, allow for frameworks to work with a quantifiable number of input variations and permutations [8]. In comparison, text-based models can expect a seemingly near-infinite number of inherently different and complex sentences. For generated counterfactual text samples to be meaningful, they need to contain plausible sentences that somewhat resemble the original input text, as a random string of incoherent characters that can alter label predictions would provide little insight into the text-based model it is based around [7].

Successful counterfactual statement generation for complex text-based machine learning models could have many real-world applications, such as aiding the development of models intended for use in sensitive environments and increasing user trust in the systems built upon them. Major issues surrounding unpredictable text-based models have been seen throughout the industry over the years. A great example of these issues can be seen in the chatbot named Tay, which was produced by Microsoft and frequently produced extreme outputs including anti-Semitic and anti-democratic remarks [9]. Increasing the ability for developers to predict outputs and identify problematic processes through the use of counterfactual statements, may help avoid situations such as these and increase brand and technology trust.

1.2 Project Goals

My overall goal is to train black-box text classification models and produce a framework that can provide effective counterfactual statements, which in turn, can be used to infer more understanding about the base models' processes. My concrete project goals are as follows:

1. Produce multiple black-box machine learning models that can accurately classify text instances.
2. Produce a framework that can generate text-based counterfactual statements when provided with a set of instances or user input.
3. The produced counterfactual statements will be grammatically correct.
4. Generated statements will have minimal alterations from their original, base text samples

Producing tools that satisfy the specified goals will enable me to automatically generate grammatically correct counterfactual statements for text-based data that closely resemble their base strings, and in turn, provide means to make valuable inferences about the original black-box models.

Chapter 2

Background and Related Work

2.1 Counterfactual Statements

Table 2.1 shows a simplified example of an input instance and the corresponding generated counterfactual response for a hypothetical rental property application. We can see that a weekly income of $400 is not sufficient for the application to be accepted, given a rent price of $500. The generated counterfactual shows that an application where weekly income is $600 would be accepted under these circumstances. From this counterfactual information, one could make several inferences, such as that weekly income needs to exceed the cost of rent, however, this is still an inference and not something explicitly stated by the XAI technique. Due to this specific example stating $600 in the counterfactual statement, it is not completely clear if there is a certain fixed amount that rent per week needs to be exceeded by or some other relative ratio-based requirement. Since $600 is both a fixed $100 above, and 1.2 times the required rent, the requirements for an application to be approved are ambiguous and up for interpretation. If the rent per week was increased to $1000, the provided counterfactual would support weekly income increasing to either $1100 or $1200 equally, hinting at the uncertainty that still remains when using XAI approximations and techniques.

Although this example is overly simple, it demonstrates that context may need to be given during counterfactual generation to further help end users understand how to interpret the results. In this case, it depends on how exhaustive the counterfactual generation was and how it was implemented. If each dollar value was incrementally checked in ascending order to turn a negative application into a positive one, it can at least be correctly interpreted that $600 is the lowest weekly income that would satisfy the model's requirements to produce an accepted application prediction. However, with higher dimensionality and more complex features, this may still not be enough to make a sufficient inference and large numbers of counterfactual statements would be required to manually make a more accurate inference.

This ambiguity of interpreting counterfactual data and statements hint at the larger issue of XAI being less effective in the hands of non-experts and imperfect users [4]. A perfect user in this context would be one that always makes perfect inferences when given a counterfactual statement. The average user when presented with a counterfactual statement may be victim to both their own biases or common logical fallacies which could skew their interpretation in a number of ways. Additionally, a user may be falsely over-confident when provided with a seemingly simplistic counterfactual response, and as such, believe their potentially misled assumptions far too readily.

Table 2.1: Hypothetical counterfactual for an individual applying to a rent a property

	Weekly Income ($)	Rent Per Week ($)	Application Accepted
Input	400	500	No
Counterfactual	600	500	Yes

Table 2.2: Hypothetical generated counterfactual for movie review

	Text	Sentiment
Input	"The movie was excellent and an absolute treat to watch"	Positive
Counterfactual	"The movie was horrible and an absolute treat to watch"	Negative

2.1.1 Text based counterfactual statements

It is important to have a basic understanding of how a machine learning model can train on a single string of text. Sentiment analysis is a process where a piece of text is analysed and subsequently categorised [10]. This is typically performed on text in order to determine the attitude of the writer, with the categories often being labelled, "positive", 'negative", and occasionally including a third "neutral" class. These sentiment analysis classes are generally referred to as sentiments.

Table 2.2 shows an example of a text-based counterfactual statement, in this instance an altered movie review that changes a hypothetical sentiment analysis model's prediction from positive to negative [11]. Here we see that the word "excellent" was replaced with the word "horrible", which successfully altered the predicted sentiment. Similar to the tabular counterfactual example shown in Table 2.1, the generated statement alludes to the inner workings of the model, however, the text-based nature of the counterfactual statement makes it arguably more ambiguous. At most from this simple singular result, we can infer that the machine learning model likely places higher importance on the word "horrible" as opposed to "treat". Additionally, this result may indicate a bias towards assigning negative predictions to reviews with seemingly conflicted sentiments, however, like in the tabular example, more than a single generated statement is likely required for accurate inference.

2.2 Text Correctness and Plausibility

Plausible text samples are grammatically correct ones that can reasonably be expected as input [7]. For example, when performing sentiment analysis, a framework may be able to alter the output result of a given model by changing the input text into incoherent gibberish, which would provide little meaningful insight into the model. Plausibility can be upheld through modifications by making sure spelling and grammar are correct in the final text, in addition to ensuring that the content of the original text is preserved.

A counterfactual statement is most effective when it has a high level of similarity to the original text it was generated from [7]. If a piece of text's sentiment was successfully altered, however, in the process underwent so many modifications such that it no longer resembled the original in any way, it would be difficult to infer any meaningful insights and defeat the purpose of generated counterfactual statement almost entirely. The most direct approach to

solve this issue is to minimise the number of modifications during counterfactual statement generation in order to retain as much of the original text as possible. Content preservation is an alternative approach that involves assessing the similarity of content between two text samples, which can be quantified effectively with the assistance of machine learning models, as seen in the BERT language model developed by Google [12]. Incorporating content preservation entails using complex machine-learning models to ensure that the main subject of a sentence or text is preserved.

2.3 Evaluating Counterfactual Statement Generation

There are several different methods for counterfactual framework evaluation in the literature, the most notable being Label Flip Score [7]. Label flip score (LFS) is used as a measure of how successfully a framework can produce samples that alter or flip the outcome label. For my use case, that being text analysis, LFS is calculated after generating a set of counterfactual text samples using some input text. LFS represents the percentage of generated samples that successfully lead a given model to produce an alternate classification label to the original text. Equation 2.1 defines LFS [7], for which an input text x_i with label C_i, generates K counterfactual text samples.

$$LFS_i = \frac{1}{K} \sum_{k=1}^{K} I(C_i \neq \hat{C}_i) \times 100 \tag{2.1}$$

2.4 Term Frequency-Inverse Document Frequency

Term Frequency-Inverse Document Frequency (TF-IDF) is a measure used in the field of natural language processing that determines how relevant a word is in a given piece of text [13]. It works by calculating both the relative frequency of a piece of text in a given sample and the text's relative frequency throughout a larger collection of written texts. TF-IDF will be used on dataset instances and input text in order to determine the relative importance of words and to avoid modifications to common, unimportant ones. For a given document or sample collection D, a word w, and an individual document or sample d $\in$ D, Equation 2.2 defines TF-IDF, where $f_{w,d}$ equals the number of times w appears in d, $|D|$ is the size of the corpus, and $f_{w,D}$ equals the number of documents in which w appears in D [13]. w_d represents a given word w, in a specified document d. The calculated importance of a word scales with its frequency in a given sample, and this frequency is then logarithmically multiplied by the ratio of total samples in which that word appears. This logarithm results in diminishing relevance improvements for words found in a larger amount of samples, which helps to minimise the scores of extremely common words like prepositions and conjunctions.

$$w_d = f_{w,d} \times \log \frac{|D|}{f_{w,D}} \tag{2.2}$$

2.5 Applications of Counterfactual Statements

Another important note is the ability to use counterfactual statements nefariously. Data poisoning and label-flip attacks occur when a nefarious person or party attempts to provide a model with problematic input data in order to hinder machine learning performance or get a specific desired result [14], [15]. An example of such an attack describes an adversary encountering a machine learning-based anti-malware program. The attacker could investigate

which factors and inputs could result in a change from a threat detection reading to a non-threat reading. It has been shown in the literature that counterfactual frameworks are very effective at producing samples with high label-flip scores and may assist in strengthening software against these label-flip attacks via effective identification of weaknesses [7].

2.6 Related Work

2.6.1 Generate Your Counterfactuals

One of the main text-based counterfactual frameworks that relate to this project was designed by IBM and named Generate Your Counterfactuals (GYC) [7]. GYC's goal is to generate counterfactuals that can be used as test cases for model evaluation. GYC performs a large portion of the tasks which I am attempting to implement in this project, generating plausible counterfactual statements with high Label Flip Scores. The main limitation of this existing solution is its accuracy with longer individual text instances. The solutions will inherently be similar, due to both my and IBM's framework focusing on producing valid sentences that effectively modify predicted class labels. GYC's purpose is to produce counterfactual statements for use in data augmentation, whereas, my goal is to produce meaningful counterfactual statements that can provide insight when inspected by a human user. The main technical difference between my solution and IBM's framework will be the prioritisation of minimal alterations to original instances when generating samples.

2.6.2 Polyjuice: Automated, General-purpose Counterfactual Generation

Polyjuice is a general-purpose counterfactual text generator that provides a number of different functionalities [11]. While many counterfactual generation tools specialise in one specific use case such as data augmentation or error analysis, Polyjuice attempts to provide functionalities for both, additionally including counterfactual statement generation. Polyjuice makes use of complex machine learning models when vectorising input text, as opposed to the more simplistic TF-IDF, which is likely done in an attempt to maximise the sentiment analysis accuracies. Similar to GYC, Polyjuice places the largest amount of focus on its ability to generate counterfactuals from a maximal number of input instances. While the tool ensures that it produces valid and realistic sentences as output, the subject and content of the original text are often lost or shifted. In the example "It is great for kids", Polyjuice produces the counterfactual statement "It is great for no one", essentially replacing the sentence's subject, whereas a simpler approach of changing the word great could have potentially provided a counterfactual statement that retained more of the original content.

2.6.3 Literature Analysis

Many of the existing counterfactual generation tools are not specifically designed for the sole purpose of counterfactual statement generation, and as such, only focus on the quantity and validity of the generated text. The main goal of text-based counterfactual statement generation is to provide a maximal amount of insight into the model, and as such, I think the emphasis on model quality is lagging behind. I believe ensuring that minimal changes are made to the base instances will produce more meaningful and human-understandable samples, due to inherent content preservation and similarities to the original text samples.

Chapter 3

Design

3.1 Sentiment Analysis Model Design

I chose to use two largely different machine learning models for sentiment analysis such that they could be compared meaningfully. Logistic Regression was selected to represent a simple black-box model, with a deep learning multilayer perceptron (MLP) being the more complex alternative. Both of these models were specifically selected due to their frequent use in many classification tasks and reasonably simple methods of implementation [16]. I intend to use both of these models to perform sentiment analysis, after which, I will compare my findings and inferences from the counterfactual statements produced, to see if there is any meaningful difference between the baseline classifiers in this context.

3.2 Counterfactual Framework Design

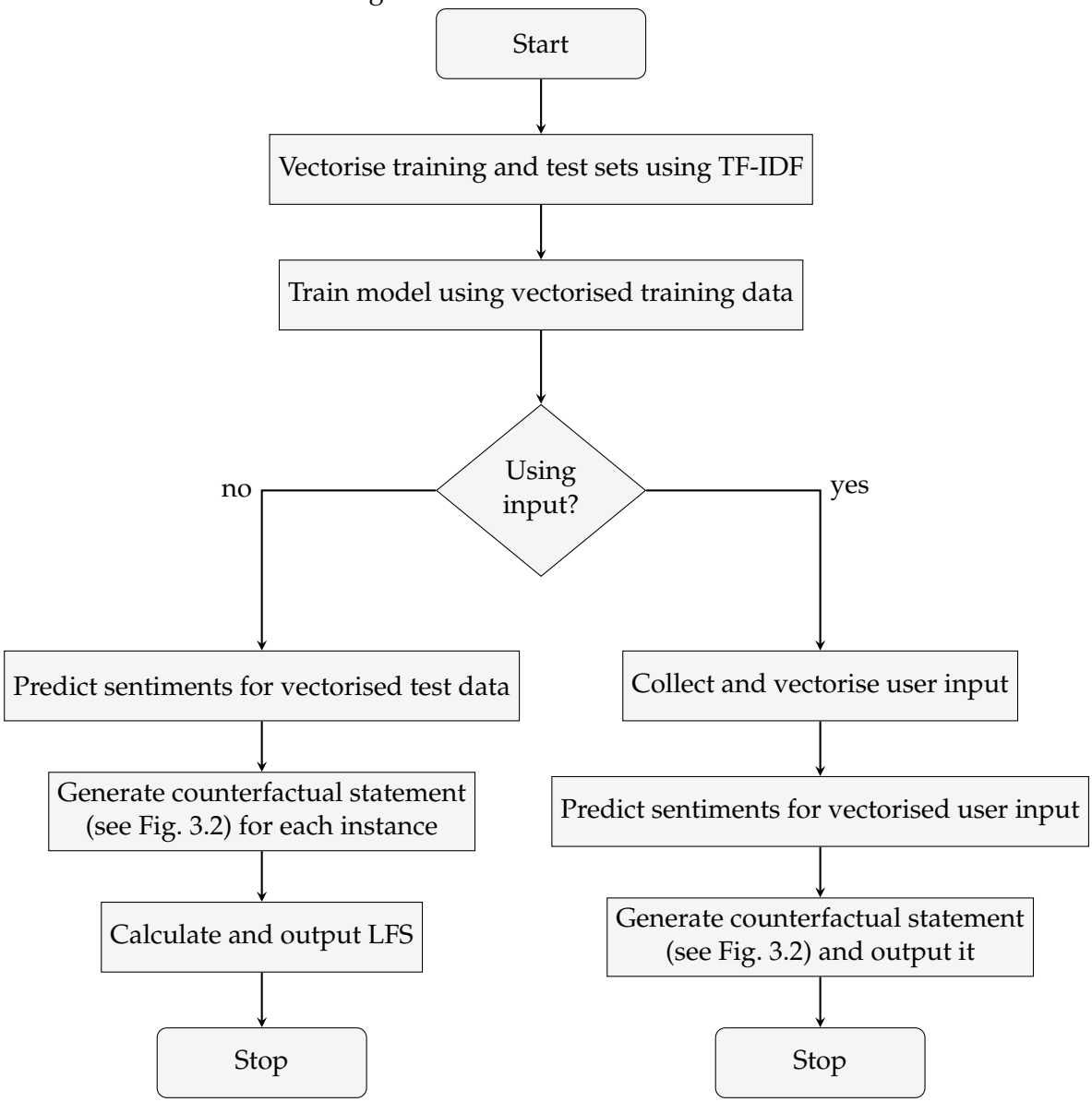
3.2.1 Counterfactual Statement Generation

Generating a counterfactual statement consists of a number of important steps which can be seen in Fig. 3.1. Once the data has been split during the initial preprocessing step, TF-IDF is performed on the training data and the vectorised corpus is stored. The selected machine learning model is then trained using the vectorised training text along with the corresponding sentiments.

At this point, the framework then offers two different types of processes that it can perform. The first of which is to generate counterfactual statements for each of the instances in the test set and calculate the resulting LFS. The test set is fed into the model to produce a baseline sentiment classification for each of the instances. Next, the test set is iterated over and counterfactual statements are created for each instance if possible, making use of the original text and baseline predicted sentiments. If a valid statement was generated, a variable counting number of successful modifications is incremented for later use in the label flip score calculation. Following the completion of the test set iteration, label flip score is calculated using the framework's number of successful modifications and the total number of instances in the test set, and is then output to the user. This LFS calculation functionality is represented by the left branch of Fig. 3.1, following the "Using input?" decision node.

Alternatively, the framework can allow a user to input text through a command line interface, generate the corresponding counterfactual statement, and finally, print that new statement back to the user. The user-provided text is vectorised using the same vectoriser and corpus as in the model training step, allowing it to retain and use the word importance rankings found during training. The vectorised user text is then fed to the model to produce a single prediction, which is then used to create a counterfactual statement. Once a

Figure 3.1: Framework Flowchart



valid counterfactual statement has been generated, it is simply output to the screen via the command line interface. If a counterfactual statement was unable to be created, the user is informed as such and presented with a reason for the generation failure instead. This functionality is represented by the right branch of Fig. 3.1, following the "Using input?" decision node.

The process of generating a counterfactual statement for a given piece of text consists of multiple steps, all of which are detailed in Fig. 3.2 and Algorithm 1. The counterfactual statement generation algorithm is provided with a single text instance and its baseline sentiment, found by classifying the original text using a selected model. The main steps for counterfactual statement generation are as follows:

- The framework first performs TF-IDF to get and store the importance of words in the provided text.
- The word with the highest importance is selected and modification is attempted.

Algorithm 1 Counterfactual Statement Generation Algorithm

```
wordImportance ← Perform TF-IDF to rank importance of words in text
sortedWordImportance ← Sort wordImportance with most the important word first
for word in sortedWordImportance do
    modifiedWord ←Perform selected modification on word.
    if modification is successful then
        modifiedText ← Replace all instances of word in text with modifiedWord
        isValid ← Check grammar of modifiedText
        if isValid then
            vectorised ← Vectorise modifiedText
            prediction ← Predict sentiment of vectorised
            if prediction != original prediction of text then
                return modifiedText
return None
```

- If a modification for the most important word was found, all instances of that word in the original text are replaced with the new modified one.
  - In the case that the selected modification involved appending "not" prior to the base word, the combination of both words replaces the single original one.
- The new text, with all instances of a selected word having been replaced, is then checked for grammatical correctness using existing natural language processing tools.
- If the modified text is deemed grammatically valid, it is subsequently vectorised and fed to the model for sentiment prediction.
- If the grammatically correct modified text produces a sentiment that is not the same as the original, a counterfactual statement has been generated and is returned.
- If modification could not be made to a piece of text, grammar was invalid post modification, or the modified version did not alter the sentiment, the next most important word is selected and the process begins again from the modification step.
- In the case that all words are exhausted, with no valid counterfactual statements being generated, a null value is returned and handled by the outer processes of the framework.

An important note is that this approach only makes modifications to a single word in a given text, as the main goal of my framework is to reduce the number of modifications made to original strings. A counterfactual statement that has undergone vast amounts of modification and bears little similarity to the original input text is less useful than one in which reduced numbers of modifications are made.

3.2.2 TF-IDF Vectorisation

For TF-IDF, I selected a pre-provided solution, seeing as comprehensive vectorisation tools already exist and vector normalisation nuances make manual implementations less effective. During the sentiment analysis model's training step, a corpus of all existing words in the dataset is generated. Any words contained within a set of pre-defined unimportant stop-words are removed from the corpus to increase future TF-IDF performance and remove the importance of a large number of auxiliary and conjunctive words entirely. TF-IDF

is then performed iteratively on each instance in the dataset to calculate the importance of each word. A sorted list containing all words and their corresponding importance values is generated by summing each of the TF-IDF scores for each word from each instance. What remains is a set of all words in the dataset with their corresponding total importance in the entire training set.

3.2.3 Text Modifications Strategies

The first and primary method of word modification is to simply find that word's antonym. If no antonym can be discerned for the most important word, the framework checks to see if the word is an adjective or adverb such as "good" or "slow", again using the natural language toolkit library. Most adjectives and adverbs can have their meanings reversed when the word "not" is inserted before it, and as such, that is what my framework does when it encounters these types of words in the absence of simple antonym replacement. Naively inserting "not" into sentences will often invalidate grammatical correctness, however, grammatical correctness is checked post-modification so any instances of this should be caught in most cases.

3.2.4 Generated Text Validation

For my framework, inspecting the validity of an instance or piece of text simply consists of checking both the spelling of constituent words and overall grammar. Correct spelling is ensured by performing a simple dictionary lookup on each word, whereas grammar checking is more nuanced and less viable to manually implement. Ensuring correct grammar can be done by using a wide range of comprehensive regular expressions to enforce good sentence structure, however, creating such a system would require a tremendous amount of work and would likely come nowhere close to many existing solutions. In an attempt to avoid re-inventing the wheel, I used a pre-existing library named language tool python which provides a simpler and more comprehensive solution to checking the basic grammar of strings.

3.2.5 User Interface and Interactions

My framework simply provides a basic command line interface for both input collection and output logging. A more comprehensive and user-friendly interface solution was out of the scope of this project. My goals were more research-oriented and pertained to creating a functional counterfactual statement generation framework that was not intended to be an end product for consumer use. As such, no outsourced testing for user experience was required or performed throughout the course of development.

3.3 Alternative Designs

3.3.1 Alternative Ways to Rank Word Importance

There are several other ways to rank the importance of words that could have alternatively been used in this project. The most simple of which would likely be a primitive ratio-based approach. During the training stage of the sentiment analysis model, the training set could be analysed, with each word in the corpus being assigned what is essentially a purity value for each sentiment. A ratio could be created for each word that details how many times it appears in instances with positive sentiments versus negative sentiments. Words in the

corpus that are exclusively found in text with a certain sentiment are likely to be very important and impact the classification heavily. The main drawback to this approach is to do with the nuances of evaluating each word's importance given its sentiment purity and the number of times it appears throughout the dataset. A word that only appears once will be seen as essentially having a maximum amount of purity, whereas it may only be obscure or misspelled. This approach would likely have to additionally incorporate a technique similar to TF-IDF anyway to resolve these difficulties, and it is doubtful that it would achieve the accuracy of more comprehensive, and well-supported solutions such as existing TF-IDF implementations.

More comprehensive machine learning-based word association approaches are also available such as Word2vec and the Bag-of-words model [17][18]. Where TF-IDF creates vectors using a basic statistical measure, the Word2vec model uses more complex machine learning to create vectors that likely need post-processing in order to use in a similar way. Word2vec has been shown to be more computationally intensive than TF-IDF, alongside providing lower accuracies when used for training many sentiment analysis models [17]. Due to the large number of instances contained within my selected datasets, reducing computational time is somewhat important and the downsides of more complex approaches appear to outweigh the benefits. The Bag-of-words model also appears to have similar drawbacks for my specific application. Even if Word2vec or the Bag-of-words model was to produce a more accurate ranking of the importance of words, it would likely not matter for this project, seeing as I expect the accuracies of my sentiment analysis model to be sufficient when using TF-IDF. Additionally, the exhaustive nature of my framework's text modifications means that the potentially higher likelihood of finding a successful counterfactual in the fewest amounts is not necessary, especially since at this modification stage, the computationally intensive training has already been completed.

3.3.2 Counterfactual Framework Text Modifications

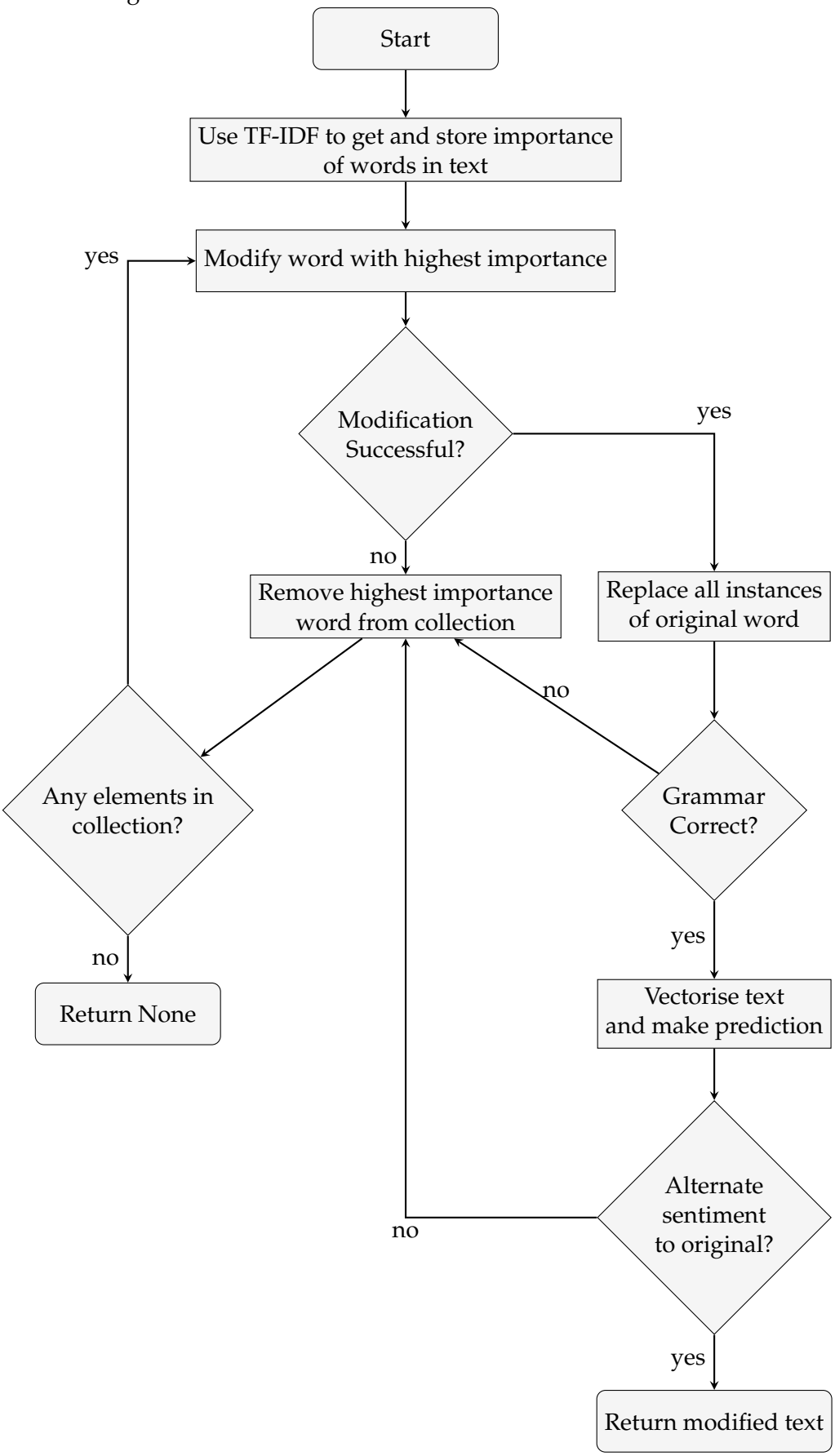
The most evident alternative design decision that would produce drastically different results is likely related to the number of modifications made per text instance. During counterfactual statement generation, I limit my text modifications to a single word per piece of text. This is with the intention to limit the magnitude of changes made to the original text, in accordance with my project goals. Textual counterfactual statements that are similar to their base texts are more useful due to their ease of inference, allowing human users to more easily discern what specific changes resulted in altered sentiments. A counterfactual statement that has little or no resemblance to the original input string could easily be generated but would provide negligible insight into the model in question. I could have exhaustively attempted to make modifications to all words and iteratively increased the number of words to be modified per instance, stopping at a valid counterfactual statement or other termination criteria, however, I felt that this fell outside of the scope of this project. Additionally, defining at what point the modified text is too dissimilar from the original would likely be difficult.

Some other strategies to modify words could have been implemented, although were omitted due to their suspected limited feasibility. In the absence of valid antonyms or words that can have their meanings inverted by the adverb "not", it may be viable in a small number of cases to remove certain words entirely. However, in practice, this is very likely to produce grammatically incorrect statements seeing as the vital words that would require modification are just that, vital to the sentence structure.

Another possible approach could be to incorporate a more primitive brute force modification method, where an increasing number of characters are changed at random and

after which are checked for correct spelling and grammar. This approach may end up finding extremely efficient solutions and counterfactual statements with a minimal number of characters changed, however, it is extremely unlikely to do so in a meaningful way. Take for instance the movie review “That movie was bad“, a brute force approach may discover that the text “That movie was rad“ produced an altered sentiment with only a single character change. My framework design might determine that the most accurate antonym for “bad“ would be the word “good“ instead and as such have a potentially inferior result and a higher number of characters changed. Quite obviously the computational cost of a simple brute force solution would be incredibly large, requiring an unfeasible amount of attempted modifications. To put the cost into perspective, a purely permutation-based brute force approach would take over 40 million years to exhaust all modifications in a 10-character string, provided modification and validation of the string took a single second.

Figure 3.2: Counterfactual Statement Generation Flowchart



Chapter 4

Implementation

4.1 Sentiment Analysis Model

4.1.1 Datasets

For this project, I chose to train my sentiment analysis model on two separate datasets. Both datasets contained a number of string text entries and an associated and predetermined sentiment as either positive or negative. The benefit of using more than one single dataset is that excessive over-fitting can be identified, allowing for future mitigation steps to be taken if required. Both of the datasets were retrieved from Kaggle, an online data science community, and were stored in CSV format. I chose to use pandas, a python data analysis library, to store these datasets in a DataFrame.

The first dataset contained movie reviews taken from The Internet Movie Database (IMDb) website [19]. The wide range of over 40,000 user-provided movie reviews provided a huge number of accurate text-sentiment pairs, seeing as correct user intent and sentiment were likely ensured through the accompanying rating scores. The dataset contained equal numbers of positive and negative reviews and had no missing or null values. The nature of user-provided movie reviews meant that the dataset contained a huge number of spelling and grammatical errors throughout a majority of the text instances.

The additional, secondary dataset contained roughly 14,600 tweets directed towards U.S. airline companies [20]. In contrast to the previous dataset, the airline tweet data contained an imbalanced spread of sentiments, with 21% of instances being labelled as positive, 63% being negative, and a further 16% of instances that were labelled as neutral. Additionally, null values are also present in the initial data and would have to be dealt with accordingly. Due to the nature of tweets being solely textual with no other supporting rating or context, the sentiments provided in this dataset were likely produced manually by humans or provided by another machine learning sentiment analysis model which may indicate lower accuracy.

4.1.2 Data Preprocessing

Some specific preprocessing steps were required for the Airline dataset, which contains more features than we require, and as such, the first required step is to remove all features except for text and sentiment. After the other features have been removed, there are no longer any null values contained within the dataset as they were all contained within the now-removed auxiliary features. The next step was to omit any instances that contained neutral sentiments, as the IMDB dataset does not contain them, nor is the framework designed around facilitating more than two sentiments. The removal of text with neutral or

ambiguous sentiments may skew the accuracy of any sentiment analysis model training on this dataset, as it will likely have a more difficult time classifying text with subtle sentiment indications.

Public online reviews and tweets are not known for their formality and grammatical correctness, as such, the datasets to be used for training naturally contain a wide range of samples with colloquial words, spelling mistakes, and incoherent text. I made use of regular expressions to return words with incorrectly repeated characters into their correct base form. For example, a review consisting of the string "It was a very loooong movie", would be corrected to "It was a very long movie" through pattern matching in combination with dictionary lookup comparisons. An important note here is that the first correct instance of a dictionary-defined word will be taken as the original intended word, whereas this may not be correct. Take the review "The director is a looooser" for example. This preprocessing step would begin reducing the amount of 'o' characters until a valid word is found. The intuitively correct word here would be "loser", however, "The director is a looser" would be the final string due to "looser" being a valid word.

I chose to remove all punctuation characters, except apostrophes, during the preprocessing stage despite their importance in sentence definitions. The vectorisation process is solely based on the Term Frequency Inverse Document Frequency (TF-IDF) algorithm, calculating the relative importance of specific words. The problem with this situation reveals itself if we look at the following subsection of a given review "horrible,". If punctuation is included in the vectorisation, the words "horrible" and "horrible," will have to be considered as completely separate words which doesn't make much intuitive sense and will likely muddy the results of TF-IDF. The reason apostrophes are avoided being removed is due to the nature of the punctuation mark. As seen in the words "its" and "it's", the apostrophe indicates a completely separate word and, as such, entries for both of these examples in the corpus should be expected.

I made the conscious decision to keep any unknown or misspelled words that could not be fixed through the previous steps. Undefined words being present in the training set is likely not as problematic as it may sound a first. A major portion of how the TF-IDF algorithm ranks words is based on how many times a word appears throughout the instances in the dataset. If a word is incorrectly misspelled, it is relatively unlikely that a vast majority of other instances also have the exact same erroneous spelling. On the other hand, the benefit of retaining undefined words is that widely used colloquial terms that fail dictionary look-ups can still be vectorised, ranked, and trained with.

Test samples have their punctuation removed and repeated characters changed, however, I do not discard instances with undefined words such that I can produce a label for each test instance. Words that have no definition are instead skipped for TF-IDF such that they are not perceived as extremely important and prioritised. During training, the complete set of samples used in the inverse document frequency calculation only considers the training set, with the test set being excluded to avoid future test bias.

I split the data into training and test sets containing 80% and 20% of the original dataset respectively. Both of the initial datasets are large enough to support this ratio, having over 10,000 total instances. I split the data such that both the training and test sets maintained an equal ratio of both positive and negative sentiments. In the case of the airline dataset which innately contains an unequal balance of sentiments, I incorporated oversampling techniques to remedy the class imbalance.

4.1.3 User Input Preprocessing

It is necessary to perform a few basic preprocessing steps on the user input, which are similar to the steps taken in the sentiment analysis model prior to training. As mentioned when looking at the original model datasets, punctuation can be problematic and cause issues during the vectorisation steps. Since users will likely include punctuation in their input, it may cause issues similar to that of the previously discussed "horrible" versus "horrible," discrepancy. If I were to omit preprocessing at this stage and a user was to include the substring "horrible," in their input text, it would be interpreted as a unique word and would not appear when looking up the word in the vectorised training corpus.

4.2 Model

For all of the training and model creation, I make extensive use of the python machine learning library scikit-learn as it is very effective, provides sufficient utilities for this project, and is widely supported and used throughout the industry [21]. Scikit-learn provides implementations of Logistic Regression and Multilayer Perceptron Classifier models, both of which I used as the basis of my training.

4.3 TF-IDF Vectorisation

Scikit-learn also provides an effective TF-IDF vectoriser implementation through its feature extraction module. The library's TfidfVectorizer converts a collection of raw text documents to a matrix of TF-IDF features, incorporates the ability to blacklist certain words and manually reduce the importance of auxiliary words, and provides a number of useful methods that range from normalisation to decoding vectors back into their original strings.

4.4 Text Modification

The python Natural Language Toolkit (NLTK) provides a suite of libraries that aid in processing English written language. A specific module of NLTK called wordnet provides the ability to very simply generate an antonym for any given word, provided one exists. I used wordnet in my framework to generate the required antonyms during text modification, in addition to using its word type analysis abilities to inspect the nature of words when required. I additionally used a python library named language-tool-python for checking the grammatical correctness of my generated counterfactual statements. While language-tool-python also provided corrective suggestions, I simply used it for a binary text correctness evaluator.

Chapter 5

Evaluation

5.1 Results

5.1.1 Sentiment Analysis Model Results

Table 5.1 shows the test prediction accuracies of the two different classifiers for each of the datasets. It serves as a simple, base classifier comparison when predicting class labels of vectorised text. The IMDB dataset contains 50000 instances, with 40000 and 10000 instances being in the training and test sets respectively. Similarly, the Airline dataset contains 11565 usable instances, 9252 training instances, and 2313 test instances. The presented accuracies for the sentiment analysis models over the two datasets are of expected and acceptable magnitudes, all falling between 80% to 90%. The complex nature of sentiment analysis on datasets containing imperfect, user-provided text makes achieving higher accuracies a difficult task. The results provided here are similar to results seen in a multitude of similar sentiment analysis tasks and published in current research [22].

The accuracies are likely sufficient for the purposes of counterfactual statement generation, however, it should be noted that a certain amount of error will be carried forward into the framework and cause difficulties in changing the sentiments of text that has been incorrectly labelled by the model. The imperfect accuracies, relating to the number of statements generated, won't necessarily hinder the possible inferences that can be obtained from the generated statements, seeing as counterfactuals are often used to investigate these exact problems found in the models themselves.

Provided users input similar data as seen in the original datasets, I can expect roughly 10%-18% of the generated counterfactual statements to have the same sentiment as the true input sentiments, due to these model inaccuracies. As a simplified illustration of this expectation, if the base model incorrectly predicts that the text, "I love Air New Zealand" is negative, the framework will exclusively attempt to generate and output a positive counterfactual statement.

As perhaps expected, the more complex and comprehensive multilayer perceptron outperformed logistic regression in terms of sentiment analysis accuracy over both datasets, however, only by a small margin. Additionally, it also appears that the Airline dataset proved to be more challenging to make accurate predictions for in general. This is likely due to there being less than a quarter of the training data in the Airline dataset when compared to the IMDB one , alongside the potentially less accurate pre-provided sentiments as discussed in the design chapter.

Table 5.1: Text sentiment analysis model accuracies

Dataset	Logistic Regression Accuracy (%)	MLP Accuracy (%)
IMDB	85.17	89.81
Airline	82.94	86.37

Table 5.2: Label Flip Scores for test datasets

Dataset	Number of Test Instances	Number of Successfully Altered Sentiments	Label Flip Score (%)
IMDB	10000	2216	22.16
Airline	2313	433	18.72

5.1.2 Framework Results

Table 5.2 shows the results and calculated Label Flip Scores produced when my framework was provided with the IMDB and Airline test sets. The results at first appear to be somewhat low but upon further inspection may prove to be deceiving. A lot of the instances in the test set contain text with numerous grammatically incorrect sentences, and seeing as my counterfactual framework will not create a counterfactual statement if no grammatically correct one can be found, it is unlikely that most of these incorrect original strings could ever produce a valid statement under my definition without explicit correction. The relatively low label flip scores are likely also a result of my attempt to minimise modifications made to the original instances. While more drastic changes to the base strings would have resulted in higher label flip scores, I believe these results still satisfy my goals to produce an adequate consistency of counterfactual generation with as few modifications as possible. Provided the end users input grammatically correct text, the valid statement generation rate will most definitely be much higher than the figures seen here.

Table 5.3 shows examples of generated text that successfully alters the predicted class label when the framework's model is trained on the IMDB dataset. Some table elements display only the altered sentence, rather than the entire review due to many instances being extremely large. From the generated text, you can clearly see that punctuation is retained. While punctuation is removed during preprocessing for use in vectorisation, the final text modification is done by simply replacing occurrences of the base word in the original text with the new, modified one. Similarly, Table 5.4 shows examples of generated text that successfully alters the predicted class label when the framework's model is trained on the Airline dataset.

Table 5.5 and Table 5.6 show the mean number of characters that were modified in order to produce counterfactual statements, with Table 5.5 displaying the results when using a logistic regression-based model, and Table 5.6 when using the MLP classifier. The general results are likely of expected magnitudes, seeing as only a single word is modified during counterfactual statement generation, and words have average lengths of around the mid-to-high tens. It is interesting to see that the results of both models are quite similar, however, these small differences can most likely be explained by the differences in their sentiment prediction accuracies, as the ratios seen here are approximately proportional.

Table 5.7 shows a set of user input strings for which no counterfactual statement could be generated. The table additionally provides one of the modified versions of the base text if applicable, alongside an explanation of why the framework could not produce a counter-

Table 5.3: Example IMDB text instances and their corresponding generated counterfactual statements

Original Text	Generated Text
”Where is it written that sequels must suck?”	”Where is it written that sequels must not suck?”
”It’s difficult to express how bad this movie is.”	”It’s difficult to express how good this movie is.”
”This film is very boring and so long.”	”This film is very interesting and so long.”
”The Rock’s acting was lackluster”	”The Rock’s acting was not lackluster”
”This show was an amazing, fresh & innovative idea”	”This show was an horrible, fresh & innovative idea”
”I didn’t really understand the movie, it was confusing from start to finish”	”I didn’t really understand the movie, it was simple from start to finish”
”i will never watch another terminator movie”	”i will always watch another terminator movie”
”Yes, I call this a perfect movie”	”Yes, I call this a flawed movie”
”I love Marvel movies”	”I hate Marvel movies”
”This was without a doubt the best in the series”	”This was without a doubt the worst in the series”
”This sequel proved to be better than the original”	”This sequel proved to be worse than the original”
”The lion king was excellent, I can’t wait to see what live action adaption they do next”	”The lion king was poor, I can’t wait to see what live action adaption they do next”
”There seems to be a bad smell surrounding anything Shyamalan produces in recent years”	”There seems to be a good smell surrounding anything Shyamalan produces in recent years”
”What a good way to waste money”	”What a good way to save money”

factual statement. Short descriptions of each of the instances are as follows:

0. This is the intended effect, seeing as no valid words are present.

1. The only modifiable word was spelled incorrectly and as such was not recognised. This is the intended effect of my implementation, however, suggestive correction could easily fix these cases if desired.

2. Simply no modifiable words were present in the original text.

3. The modification techniques use the python natural language toolkit which only works with English text.

4. This is an example of Garbage-In Garbage-Out. The original text made no sense, and as such, neither did the modified one.

5. Failure to generate a grammatically correct statement for similar reasons to that of the instance with index 4.

Table 5.4: Example Airline text instances and their corresponding generated counterfactual statements

Original Text	Generated Text
”YOU GUYS ARE HORRIBLE”	”YOU GUYS ARE amazing”
”I definitely was not the only person in line who thought it was absurd”	”I definitely was not the only person in line who thought it was reasonable”
”Another awful experience at the check in desk”	”Another wonderful experience at the check in desk”
”I didn’t enjoy my time”	”I didn’t dislike my time”
”You guys really do suck”	”You guys really do not suck”
”The plane was 3 hours late”	”The plane was 3 hours early”
”Our flight was delayed again for the second time”	”Our flight was not delayed again for the second time”
”I won’t be flying with you again”	”I will be flying with you again”
”That would be convenient seeing as we left there a day ago”	”That would be inconvenient seeing as we left there a day ago”
”i love flying with you guys”	”i hate flying with you guys”
”Smallest plane I have ever been on and smoothest landing ever”	”Smallest plane I have ever been on and roughest landing ever”
”Don’t know her last name, but Karen at your call center is terrific”	”Don’t know her last name, but Karen at your call center is dreadful”
”We had the greatest time”	”We had the worst time”
”The flights were affordable and extra bags were included”	”The flights were not affordable and extra bags were included”
”I couldn’t recommend you guys enough!”	”I couldn’t reject you guys enough!”
”There was enough time to watch a few movies”	”There was not enough time to watch a few movies”

6. Although the original text is closer to being grammatically correct than numbers 4 and 5, it is still invalid.

7. The original text, in this case, is correct, however, the only possible modification was to the word “few“, whose antonyms produced a grammatically incorrect modified sentence. This is working correctly, seeing as there was no valid counterfactual statement by my definition.

8. While the framework was able to modify a word in the input text, it had no impact on the sentiment when re-predicted, and as such, the statement was rejected. This failure to impact sentiment was likely due to the neutral nature of the original text.

9. Failure to alter sentiment for similar reasons to that of the instance with index 8.

10. Due to the relatively neutral nature of both the original and modified text, the model produced the same sentiment for both pieces of text, and since no other valid modifications could be made, it failed.

Table 5.5: Mean number of characters modified in successful counterfactual statements using Logistic Regression

Dataset	Mean Number of Charcters Changed	Mean Percent of Characters Changed (%)
IMDB	6.51	2.74
Airline	7.12	3.89

Table 5.6: Mean number of characters modified in successful counterfactual statements MLP

Dataset	Mean Number of Charcters Changed	Mean Percent of Characters Changed (%)
IMDB	6.54	2.75
Airline	7.07	3.87

11. The failure correlating to this instance is likely due to the overwhelmingly positive sentiment of the original text. The framework exhausted all possible modification options in an attempt to produce a negative sentiment but could not.

Table 5.7: Example user provided text for which valid counterfactual statements could not be generated (Trained on IMDB dataset using Logistic Regression)

Index	Original Text	Modified Text	Reason For Rejection
0	″salkdfaoishv″	-	No modifications could be made
1	″that movie was grood″	-	No modifications could be made
2	″Movie″	-	No modifications could be made
3	″den filmen var fantastisk″	-	No modifications could be made
4	″the the″	″not the not the″	All possible modifications fail grammar check
5	″in in in″	″out out out″	All possible modifications fail grammar check
6	″I like movie it good″	″I like movie it bad″	All possible modifications fail grammar check
7	″I saw the movie many times″	″I saw the movie few times″	All possible modifications fail grammar check
8	″A large hat″	″A small hat″	No statements alter the original sentiment
9	″A plane flew over the airport″	″A plane flew under the airport″	No statements alter the original sentiment
10	″That movie wasn't OK″	″That movie wasn't unsatisfactory″	No statements alter the original sentiment
11	″Simply fantastic! What a wonderful movie. It was amazing. I had a great time″	″Simply fantastic! What a wonderful movie. It was horrible. I had a great time″	No statements alter the original sentiment

Chapter 6

Conclusions and Future Work

6.1 Contributions

As discussed in the background chapter of this report, many of the existing text-based counterfactual generation systems attempt to maximise the number of counterfactual statements that can be generated when provided with a given test set. A lot of emphasis is placed on label flip score while paying little attention to the magnitude and consequences of variation from the original texts. A larger number of counterfactual statements that are heavily modified, and as such, harder to gain inference from is likely worse than fewer statements that resemble the original texts more closely and subsequently provide more insight to end users. The ultimate goal of counterfactual statements is to provide insights into machine learning models, so ease of interpretation is vital. I believe that the results from my approach to minimise the number of words changed during counterfactual statement generation serve as a proof of concept and may hint at the validity of prioritising original information retention during text modification.

6.2 Future Work

6.2.1 Batch User Input

I think that giving users the ability to upload a file containing a set of text instances would be beneficial to implement in the future. The framework could produce counterfactual statements for each instance, report a label flip score for the set, and provide more results such that more accurate model inferences can be made. This fell out of the scope of this particular project, seeing as my goals revolved around producing a counterfactual statement for a single user-provided string.

6.2.2 Improved User Experience

Since this project had no intended consumer or user base and instead was concerned with creating a functional counterfactual framework implementation, I did not design or implement any user interface or quality-of-life features. One approach to improve this lack of usability could be to develop a more comprehensive application with a Graphical User Interface (GUI) with user experience in mind. This would likely require some amount of user testing and feedback. Alternatively, my framework could potentially become fully realised as an API or library to be used in other applications and projects.

6.2.3 Framework Improvements

My framework currently rejects any counterfactual statement that is generated when provided with user input that fails grammar checks prior to modification. This is because, with the exception of a small number of cases, grammatically incorrect text with a single word modified will likely still be incorrect text, and as such, the generated counterfactual statements will be rejected. Perhaps implementing a more comprehensive grammar evaluation that assigns a quantifiable grammatical correctness value would provide a solution, as modifications that do not decrease the coherence and correctness could be accepted, even if errors still existed. I am unsure as to the availability or viability of such a technology, however, other approaches to solve this problem may also exist, such as, evaluating the grammar of only the modified sentence, or disabling grammar checking altogether if the original text fails validation.

Since counterfactual statements are only as good as the inferences that can be made from them, it would be interesting to incorporate some user testing to investigate whether the produced outputs are what users would expect and if they can discern any notable insights about the models that generated them. While counterfactual generation for tabular data can more easily be evaluated quantifiably, string outputs are much more difficult to assess and this approach could provide an additional evaluation metric for my framework's outputs.

6.3 Final Remarks

When reflecting on my initial goals for this project, it is evident that all goals have been met to a satisfactory level. My final framework can generate counterfactual statements for provided text, does not modify auxiliary words, such as prepositions and pronouns, and generates statements with efforts to minimise alterations from original text samples. It does this all while ensuring the grammatical correctness of generated text, and allowing users to input custom text instances that are then transformed into counterfactual statements with a decent success rate. While there could still be many improvements made to my framework and the evaluation scores appear low on the test datasets, it still meets my initial intentions and performs adequately.

Consideration

At the beginning of the second trimester, I endured severe and unforeseen personal hardship which rendered me largely unable to work on this project for several months. The process of recovering from and handling grief occupied a large amount of time that would otherwise have been spent on development and several stages of report writing. The delayed development resulted in a lower volume of code and encroached on my planned report writing time, leading to a final product that I believe ultimately doesn't reflect my abilities accurately. While I am still reasonably happy with the results of my hastened work, I believe the quality of both report and code would have been higher had this unexpected personal loss not been present. Any consideration, provided these circumstances, would be greatly appreciated.

Bibliography

[1] S. Sharma, J. Henderson, and J. Ghosh, *CERTIFAI: A Common Framework to Provide Explanations and Analyse the Fairness and Robustness of Black-Box Models*. New York, NY, USA: Association for Computing Machinery, 2020, p. 166–172. [Online]. Available: https://doi.org/10.1145/3375627.3375812

[2] C. Molnar, "A guide for making black box models explainable," *Interpretable Machine Learning, 2nd ed.*, 2022. [Online]. Available: https://christophm.github.io/interpretable-ml-book/

[3] O. Loyola-González, "Black-box vs. white-box: Understanding their advantages and weaknesses from a practical point of view," *IEEE Access*, vol. 7, pp. 154 096–154 113, 2019.

[4] B. Babic, S. Gerke, T. Evgeniou, and I. G. Cohen, "Beware explanations from ai in health care," *Science*, vol. 373, no. 6552, pp. 284–286, 2021.

[5] A. Barredo Arrieta, N. Díaz-Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado, S. Garcia, S. Gil-Lopez, D. Molina, R. Benjamins, R. Chatila, and F. Herrera, "Explainable artificial intelligence (xai): Concepts, taxonomies, opportunities and challenges toward responsible ai," *Information Fusion*, vol. 58, pp. 82–115, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1566253519308103

[6] S. Wachter, B. Mittelstadt, and C. Russell, "Counterfactual explanations without opening the black box: Automated decisions and the gdpr," *Harvard journal of law and technology*, vol. 31, no. 2, p. 841, 2018. [Online]. Available: https://arxiv.org/abs/1711.00399/

[7] N. Madaan, I. Padhi, N. Panwar, and D. Saha, "Generate your counterfactuals: Towards controlled counterfactual generation for text," 2020. [Online]. Available: https://arxiv.org/abs/2012.04698

[8] R. M. J. Byrne, "Counterfactuals in explainable artificial intelligence (xai): Evidence from human reasoning," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*. International Joint Conferences on Artificial Intelligence Organization, 7 2019, pp. 6276–6282. [Online]. Available: https://doi.org/10.24963/ijcai.2019/876

[9] M. Wolf, K. Miller, and F. Grodzinsky, "Why we should have seen that coming: Comments on microsoft's tay "experiment," and wider implications," *The ORBIT Journal*, vol. 1, no. 2, pp. 1–12, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2515856220300493

[10] W. Medhat, A. Hassan, and H. Korashy, "Sentiment analysis algorithms and applications: A survey," *Ain Shams engineering journal*, vol. 5, no. 4, pp. 1093–1113, 2014.

[11] T. Wu, M. T. Ribeiro, J. Heer, and D. S. Weld, "Polyjuice: Automated, general-purpose counterfactual generation," *CoRR*, vol. abs/2101.00288, 2021. [Online]. Available: https://arxiv.org/abs/2101.00288

[12] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing*, K. Inui, J. Jiang, V. Ng, and X. Wan, Eds. Association for Computational Linguistics, 2019, pp. 3980–3990. [Online]. Available: https://doi.org/10.18653/v1/D19-1410

[13] J. E. Ramos, "Using tf-idf to determine word relevance in document queries," 2003. [Online]. Available: https://www.researchgate.net/publication/228818851_Using_TF-IDF_to_determine_word_relevance_in_document_queries

[14] R. Taheri, R. Javidan, M. Shojafar, Z. Pooranian, A. Miri, and M. Conti, "On defending against label flipping attacks on malware detection systems," *Neural Comput. Appl.*, vol. 32, no. 18, pp. 14 781–14 800, 2020. [Online]. Available: https://doi.org/10.1007/s00521-020-04831-9

[15] E. Rosenfeld, E. Winston, P. Ravikumar, and J. Z. Kolter, "Certified robustness to label-flipping attacks via randomized smoothing," in *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, ser. Proceedings of Machine Learning Research, vol. 119. PMLR, 2020, pp. 8230–8241. [Online]. Available: http://proceedings.mlr.press/v119/rosenfeld20b.html

[16] D. G. Kleinbaum and M. Klein, *Introduction to Logistic Regression*. New York, NY: Springer New York, 2010, pp. 1–39. [Online]. Available: https://doi.org/10.1007/978-1-4419-1742-3_1

[17] X. Rong, "word2vec parameter learning explained," *CoRR*, vol. abs/1411.2738, 2014. [Online]. Available: http://arxiv.org/abs/1411.2738

[18] Y. Zhang, R. Jin, and Z.-H. Zhou, "Understanding bag-of-words model: a statistical framework," *International journal of machine learning and cybernetics*, vol. 1, no. 1, pp. 43–52, 2010.

[19] L. N, "Imdb dataset of 50k movie reviews," Mar 2019. [Online]. Available: https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews

[20] F. Eight, "Twitter us airline sentiment," Oct 2019. [Online]. Available: https://www.kaggle.com/datasets/crowdflower/twitter-airline-sentiment

[21] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[22] J. Hartmann, M. Heitmann, C. Siebert, and C. Schamp, "More than a feeling: Accuracy and application of sentiment analysis," *International Journal of Research in Marketing*, 2022.