Vulnerability of Natural Language Classifiers to Evolutionary Generated Adversarial Text

Manjinder Singh

Dr Alexander Brownlee

February 2021

Dissertation submitted in partial fulfilment for the degree of Bachelor of Science with Honours Computing Science

Computing Science and Mathematics

University of Stirling

Abstract

Reputation, reliability, and branding are all variables that can be used to categorise an organisation as either trustworthy or not trustworthy. Automated text analysis models are increasingly used to auto-generate summaries which drive, for example, recommender websites. The models used to this aim need to be reliable and robust enough to avoid manipulation by bad actors. Testing these models before deployment into a production environment is therefore essential.

This project aims to generate adversarial examples against a sentiment analysis model, using genetic algorithms. To generate adversarial examples, we first need a sentiment analysis model to act as the victim model in our project. To this end the first step was to create our own model and to train it to be sufficiently accurate as a predictor of sentiment on a test dataset.

The project had two main aims. The first of which was to generate grammatically correct adversarial examples i.e., examples that when read by a human would read "naturally". The second aim was to ensure we make the fewest number of changes to the original text i.e., to swap the least number of words when generating our adversarial examples. To ensure we made the least number of changes to our original text, after each target word is swapped, we test the entire text against our model and check the sentiment value returned. Once we have successfully flipped the original sentiment of our original text, we stop the algorithm run and record this as a success.

To achieve the first aim, we used three different algorithms to select a replacement word. the first of the algorithms selected a word based on special distance, and the closest word was chosen i.e., a synonym of our target word. The second algorithm selected the replacement word after consideration of the surrounding words and selected the correct word based on context. The final algorithm used a combination of the first method, selection of a synonym, and then a check of the correctness of the sentence after we swapped our target word with its synonym.

We conducted multiple experiments to test the time taken by each algorithm, and selection of replacement words. It was found that out of the three algorithms developed the algorithm using context within surrounding words to select the replacement word performed best, in terms of least words swapped and most grammatically correct adversarial example generated.

The objectives of this project were met. The results for each algorithm are presented along with detailed reasoning, experiment design and discussion of outputs. This research provides a good foundation for further work and presents potential directions for future extensions of this project.

Attestation

I understand the nature of plagiarism, and I am aware of the University’s academic misconduct policy.

I certify that this dissertation reports original work by me during my University project.

Signature: Manjinder Singh Date: 10/03/2021

Acknowledgements

I would like to thank my supervisor, Alexander Brownlee, for his support and invaluable advice and detailed feedback throughout the project. Thank you for being a great supervisor and for always nudging me in the right direction.

To Kate Howie for showing me that statistics is fun and not something to be feared. For igniting in me an interest in mathematics, that I never knew I had. You are missed.

To my wife, Masami Singh (シン正美), for the endless cups of coffee, your patience and support over the years.

Table of Contents

List of Figures

List of Tables

1 Introduction

Reputation, reliability, and branding are all variables that can be used to categorise an organisation as either trustworthy or not trustworthy. Automated text analysis is increasingly used to auto-generate summaries which drive, for example, recommender websites. These then echo and amplify these perceptions. Comments made on social media platforms, on news channels, customer review sections could all be used as input to a natural language processing (NLP) model to examine the sentiment of these reviews. The output of this model could then be used to inform investors on buy/sell decisions based on the prevailing sentiment within these reviews[1][2]. Once trained, the NLP model could, potentially, be directed toward an undesirable outcome, by creating reviews that have been perturbed slightly, to give false negatives. This could impact major investment decisions and potentially disrupt local economies.

This project will explore the use of evolutionary or genetic algorithms to generate adversarial examples against NLP models. Adversarial examples are inputs created to manipulate a trained neural model to make an incorrect prediction. The adversarial examples for this project will be generated without information of the model's underlying architecture nor implementation details. The aim is to generate adversarial examples by substituting some words by their synonyms whilst attempting to minimizing the number of such substitutions. This research could be applied to help build more robust NLP models.

Genetic Algorithms perform well in combinatorial optimization problems where we have a large search space[31], such as the problem investigated in this project. We use a Genetic Algorithm to generate the most optimal solution, the solution with the least number of modifications to our original text, and one that maintains semantic similarity after modifications.

The first part of the project will be to build the NLP model that we will use to run the tests against. The second part of the project will be to implement a genetic algorithm library that we will use to generate the adversarial examples. The aim is to build a library of Python code that can be easily adapted and used to target a wide range of NLP models.

1.1 Background and Context

Deep learning models such as deep neural networks (DNNs) have been applied to a vast range of fields including speech recognition, natural language processing and computer vision to name a few. In some of these areas they have performed as well if not better than human experts[32]. However, they are vulnerable to adversarial inputs; these are inputs with perturbations that may be imperceptible to humans but can cause DNNs to misclassify the output[33].

A DNN is normally represented as a non-linear function[7] $f_\theta\colon X \rightarrow Y$, where X is the input attributes, $x \in X$ denotes a single sample, and $\mathbf{Y}$ is the output predictions. The output predictions can be either a discrete set of classes or sequence of objects. θ is the DNN parameters learned during the training of the model. The parameters are calculated by minimising the distance between the model's prediction of $f_\theta(X)$ and the correct, supplied, value of Y. The distance is measured by a loss function $J(f_\theta(X), Y)$ [8]. An adversarial example $\mathbf{x}'$ would be created by perturbation of the input, a robust DNN model would still classify the perturbed input correctly as y, however a vulnerable DNN model would, with high confidence, wrongly classify x'. An adversarial attack can have the aim of simply changing the classification of a previously correctly classified item to any other classification, such that $(f(x') \neq y)$ or to switch the classification to one specifically selected by the attacker $(f(x') = y')$. In this project we will look at the case such that $(f(x') \neq y)$.

Imperceptible changes to image data input can be relatively trivial to achieve, due to the continuous nature of image data, but it is much more challenging when using text as input. This is because text data is discrete, this means that any perturbations to the text can, in many cases, be immediately noticeable. The difficulty lies in being able to perturb a sentence, and, at the same time, preserve not only the semantics but also the syntactical structure of the sentence.

This makes this problem particularly interesting because the aim is not only to deceive the NLP model, but to do so in a way that will also deceive the human user of the system. This project will achieve this outcome through 'mutations' to the given input text, by replacing words with their synonyms and then 'evolve' these new sentences towards the best solution. The goal being to find the 'generation' with the fewest mutations that results in a misclassification of the input. After generating the adversarial example, the original text and the mutated solution will be displayed so that the two sentences can be compared in terms of semantics and syntactical structure.

1.2 Scope and Objectives

The projects main goals were:

1. Create a supervised NLP sentiment analysis model that will be used as the target against which we will generate our adversarial examples.
2. Testing the NLP system's accuracy on benchmark data, then testing on adversarial examples
3. Design and code a Python module that will contain the code for the genetic algorithm.
4. Analysis of the solutions produced by the Genetic Algorithm module
5. Potentially, ask a human subject to classify the adversarial examples produces by the system and compare against the NLP model classification
6. The aim of this project is to produce more robust models. To this end we will correctly classify any adversarial examples produced and incorporate the adversarial examples back into the original model, to make the model more robust to this form of attack.

Training of a sentiment classification model will be done using the IMDB Movie Reviews dataset[22]. The training dataset contains 50,000 reviews, split between 25,000 training and 25,000 test datasets. The reviews in both the training and test datasets have been annotated (0 = negative, 1 = positive). For the NLP model we will train a binary sentiment classification model. Using GloVe[4] each word from the review will be projected into a 300-dimensionl vector space. The Keras library will be used to train a Recurrent Neural Network (RNN) using the Long Short-Term Memory (LSTM) architecture.

Testing and refining the classification model will be of critical importance. The adversarial examples generated against this model will only be of value if the classification model is robust.

On completion of this project a fully implemented Genetic Algorithm (GA) framework will be available. This will be applied to generate black-box adversarial examples against the sentiment analysis model. The examples generated should be the near optimal solutions, in the regard that the text generated will be both semantically and syntactically correct yet be misclassified. It is hoped that the GA framework developed would be applicable to a wide range of NLP data sets and domains.

Evaluation of the adversarial examples generated will be key to assessing the effectiveness of the GA framework developed. The key statistics to investigate, other than the syntax and semantics of the adversarial text, will be the mean percentage perturbation to the original sentence and the percentage of successful attacks generated out of total attacks performed. As a final test the perturbed text could be analysed visually and classified by a human and the classification compared to the one produced by the classification model.

An interesting extension to this research could be to combine it with a time series model and work towards development of a predictive model for stock-market buy/sell decisions or predicting future share value based on time series data and sentiment analysis of pre-selected groupings of organisations or even currencies.

This project has shown that it is possible to generate grammatically correct adversarial examples against a sentiment analysis model. Theses adversarial examples can be used as additional input for the original model, to harden it against these types of attacks. The process of generating adversarial examples as outlined in this project, in our opinion, is one that should be performed on models of this type prior to introduction to a production environment to increase confidence in the model with the aim of reducing, as much as is possible, the vulnerability of the model to adversarial examples.

2 State-of-The-Art

Adversarial attacks were first applied in the field of computer vision to image classifier models, with the term 'adversarial example' first introduced by Szegedy et al. [6]. It was noticed that the researchers could cause the DNN to misclassify images by introducing small perturbations to the images. Application of 'noise' to the images led to misclassification by the DNN even when there was no perceptible difference to human observers. As neural networks have increasingly become more popular in a diverse range of fields within the AI space and specifically within the NLP field, the need to ensure the robustness of the models and the predictions they generate has become more urgent. This chapter will first begin with a review of the assumptions made with regards to the attackers, then move to a detailed discussion of the two main types of attacks. We will then look at white-box and then black-box attacks, and the most popular algorithms for generating adversarial examples within each of these two attack methods.

2.1 Attack Assumptions

This project will consider two possible assumptions we can make of a potential attacker, mainly white-box and black-box attack, as seen in *Figure 1*[10].

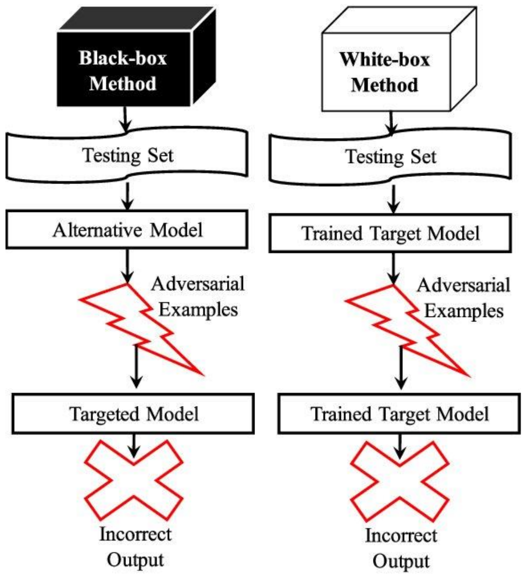


Figure 1 *Black-box vs. white-box adversarial methods. In the black-box settings, the adversary can only query and observe the targeted model's output. Consequently, it generates its adversarial examples using an alternative model. In white-box settings, the adversarial example[10]*

In white-box attacks, the attacker has full access to the model. This includes the model's architecture, loss function, activation function all inputs, outputs, and weights. A black-box attack assumes only access to the inputs and outputs of the model. We will look at the state-of-the-art research for both types of attacks.

2.2 White-box Attack

We will look at two types of popular white-box attacks. Firstly, the Fast Gradient Sign Method (FGSM) and secondly the Jacobian-based Saliency Map (JBSM) Attack method.

2.2.1 Fast Gradient Sign Method Attack

A very common and widely used attack for image classification is the Fast Gradient Sign Method Attack (FGSM)[11]. In the algorithm presented by Goodfellow et al.[11] the perturbation Δx is calculated as the sign of the models cost gradients, the partial differentials of the variables, i.e. $\Delta x = \varepsilon \mathrm{sign}(\nabla_{\mathrm{x}} \mathrm{J}(\mathrm{F}, \mathrm{x}, \mathrm{c}))$, where c is the actual class of the input x and $J(F, x, c))$ is the cost function used to train the model F and ε is set sufficiently small to make Δx undetectable. As illustrated in *Figure 2*.

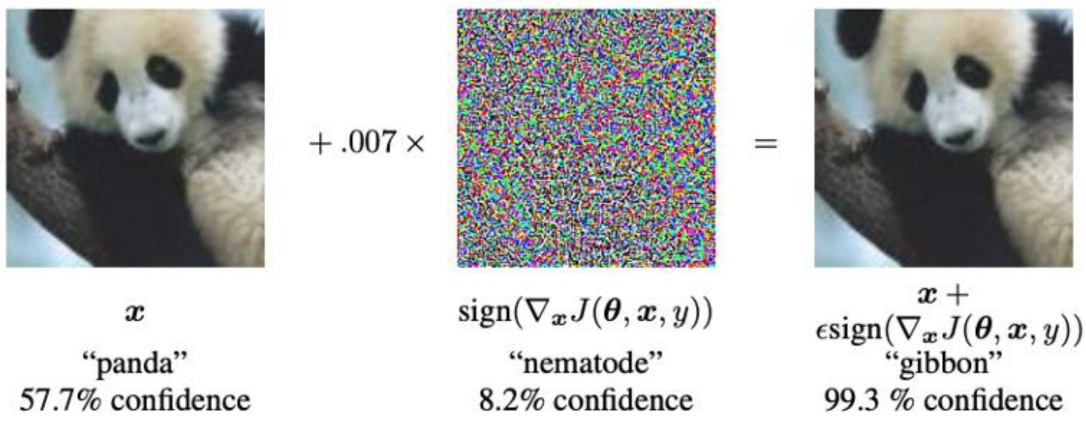

$+ .007 \times$
$=$
$\boldsymbol{x}$
"panda"
57.7% confidence
$\mathrm{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"nematode"
8.2% confidence
$\boldsymbol{x} + \epsilon \mathrm{sign}(\nabla_{\boldsymbol{x}} J(\boldsymbol{\theta}, \boldsymbol{x}, y))$
"gibbon"
99.3 % confidence

Figure 2 *FGSM adversarial example generation[11]*

This attack method was successfully applied to an NLP model by Liang, Bin et al.[12]. Rather than use the $sign$ of the cost gradient, they used the magnitude of the largest gradient. As state in the paper, "sometimes, only manipulating the input pixels with large gradient magnitude can also do the trick since a pixel with larger gradient magnitude often contributes more to the current prediction". Liang, Bin et al. outlined three attack methods: insertion, removal, and modification. They first calculated the cost gradient $\nabla_x J(F, x, c')$ for each training sample x, where c' is the target classification. Then they identified the top 50 characters containing the dimensions with maximum highest magnitude i.e. hot characters[12]. All phrases that then contain hot characters and occur most frequently are labelled Hot Training Phrases (HTPs)[12].

- ***Insertion***: the adversarial text is generated by inserting some HTPs of the target class c' close to the phrases with the most contribution to the original class c
- **Removal**: all non-essential adjectives or adverbs in the HSPs were removed
- **Modification**: *Hot Sample Phrases* (HSPs) to the current classification were identified in the same way as HTPs were identified in the insertion strategy, then characters in the HTPs were replaced by common misspellings and/or visually similar characters.

What was of significance in this algorithm was that when using only one strategy on its own, for example the removal strategy, it was very difficult to change the classification. However, combining all three produced much better results and avoided excessive modification of the original text, as shown in *Figure 3*.

The Old Harbor Reservation Parkways are three ~~historic~~ roads in the Old Harbor area of Boston. **Some exhibitions of Navy aircrafts were held here**. They are part of the Boston parkway system designed by Frederick Law Olmsted. They include all of William J. Day Boulevard running from Cast**1**e Island to Kosciuszko Circle along Pleasure Bay and the Old Harbor shore. The part of Columbia Road from its northeastern end at Farragut Road west to Pacuska Circle (formerly called Preble Circle). Old Harbor Reservation

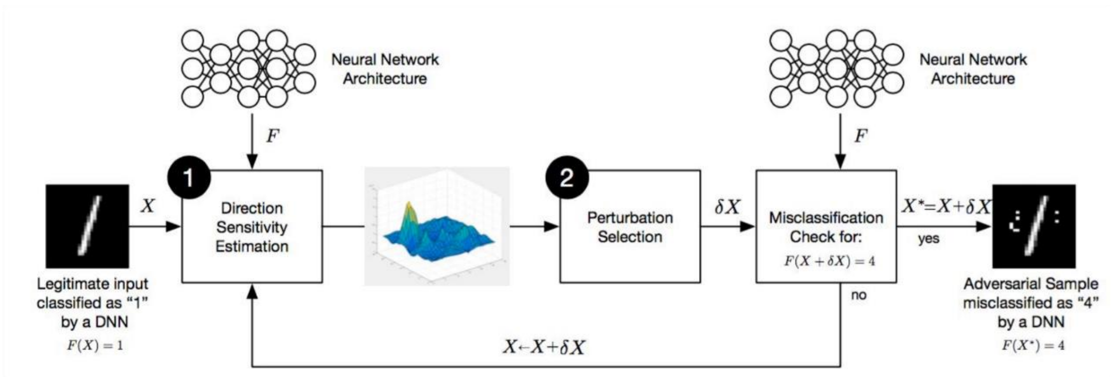
Figure 3 *Combination of three strategies (83.7% Building to 95.7% Means of Transportation)[12]*

The algorithm developed in [12] was however flawed in the sense that the grammar and semantics of the generated adversarial text was generally not preserved. This algorithm was improved upon by Samanta et al.[13] with the inclusion of an additional strategy of replacement: *insertion*, *removal*, and *replacement*.

This algorithm first checks if the word under consideration (w_i) is an adverb, then compares its contribution to the text classification, with the goal being to remove the adverb (w_i) that contributes the most to the classification task (contribution is measured using loss gradient). The motivation behind this heuristic is that adverbs put emphasis on the meaning of a sentence, and quite often do not alter the grammar of the sentence[13]. If after removal of the adverb, the text is grammatically incorrect, the algorithm will insert a word p_j before w_i. The word p_j is selected from a candidate pool P. P consists of the synonyms, typos, and sub-category specific keywords. If the output does not change the classification of the text for all the p_j then the algorithm replaces w_i with p_j. The algorithm selects each w_i in order of its contribution to the text, using the greedy method, this ensures that the least amount of change is made to the text. This algorithm was shown to be much more effective than [12]. The algorithm ordered the words ranked by contribution, and adversarial examples were constructed according to this order i.e., it is a greedy method that will always return the minimum changes required to change the text classification. The main improvement of this method over [12] was the limiting of replaced and added words, so as not to impact the grammar and Part-Of-Speech (POS) of the original words. The results were an improvement over [12], however as noted in the paper, the steps adopted for modifications are heuristic in nature , and this can be improved and automated to obtain better results[13].

2.2.2 Jacobian Saliency Map Attack

Another popular method is the Jacobian-based Saliency Map Attack (JSMA). This is a greedy algorithm originally used to generate adversarial examples against image data. The algorithm generates adversarial sample perturbations based on saliency maps. A saliency map is used to calculate the direction of sensitivity of the sample with regards to the target classification. The aim of the method being to find which of the attributes contributes the most to the classification and hence is most likely to cause a change of classification. Using the sensitivity map a few features are selected as possible candidates for perturbation, the model is then checked to see if this change resulted in a change of classification. If no change of classification occurs, then the next most sensitive attribute is selected and so on until an adversarial example is generated that causes a misclassification[14]. The method is illustrated in *Figure 4*.



Neural Network Architecture
F
1
X
Direction Sensitivity Estimation
2
Perturbation Selection
δX
Neural Network Architecture
F
Misclassification Check for: $F(X+\delta X)=4$
$X^*=X+\delta X$
yes
no
Legitimate input classified as "1" by a DNN $F(X)=1$
$X \leftarrow X+\delta X$
Adversarial Sample misclassified as "4" by a DNN $F(X^*)=4$

Figure 4 *JSMA adversarial example generation[14]*

This method of attack was successfully applied to an NLP model[15]. The authors used the JSMA method to find sequences of the text that contributed the most to classification of the text. This was then used to produce the adversarial example. The adversarial examples were produced for two types of DNNs, the first a categorial model and the second a sequential model. We will focus on the categorical model in this project. In the case of the categorical model the architecture used was Long Short-Term Memory (LSTM), see *Figure 5* for an outline of the DNN training process. The LSTM is used to solve issues that DNNs have with short term memory. If a sequence of text is too long, the DNN will have problems with propagating information from earlier time steps to later ones. In practice this means that if the DNN is processing some unstructured text as input, the DNN may leave out entirely some important pieces of information from the beginning of the text, an issue known as 'vanishing gradient'. During the back-propagation phase of learning the DNN will suffer from the vanishing gradient problem, where the gradient is reduced to a point where it no longer contributes to the learning. The use of the LSTM architecture helps to prevent this problem.

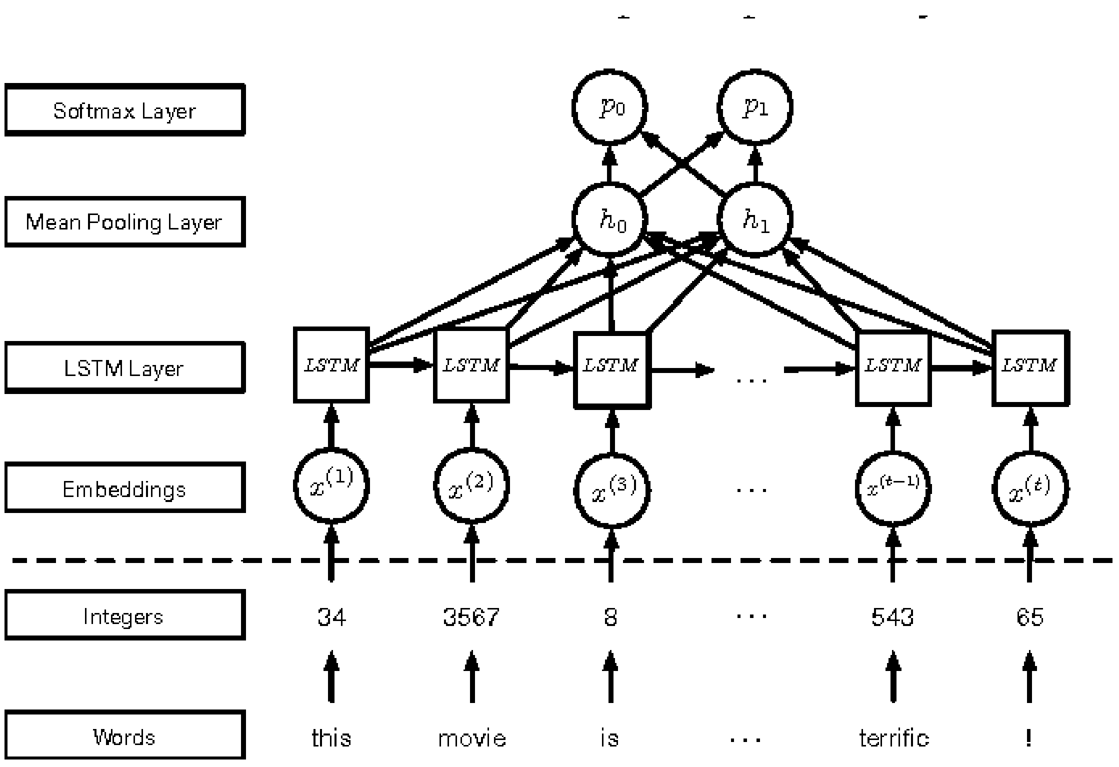


Figure 5 *LSTM-based DNN: this recurrent model classifies movie reviews [15]*

Adversarial examples were generated on the input feature vectors by use of the JSMA method. Firstly, the model's Jacobian was computed, and used to estimate the direction of the perturbation of each of the word embeddings. The major difficulty being the availability of feasible words in such a finite text space. To overcome this issue, the replacement word chosen is the one closest to the direction calculated by the Jacobian. This heuristic is then applied to the entire text, to eventually generate an adversarial example. The results of this were an error rate of 100% on the training set by changing on average 9.18 words in each of the 2000 training reviews, which is on average is a change of 12.9% to the original text. As stated in the paper the authors had access to the underlying model architecture, and the generation of adversarial examples against a model under a black-box scenario would need further investigation.

We now move on to look at black-box attacks. In this method of attack, we have no access to the underlying model, only the input and outputs are available, which makes the creation of adversarial examples much more challenging.

2.3 Black-box Attack

A black-box attack relies solely on the input and output of the DNN, to generate adversarial examples. These types of attacks are more applicable to a real-life scenario, since most classifier models in service provide for a user to input a test set and then are given an output classification. The attacker is reliant upon heuristics to generate the adversarial examples.

We will focus on black-box attacks that produce adversarial examples that are more natural[16]. There are many ways to generate adversarial examples, however the majority of them reply on perturbation to the original text that generate examples that may cause a misclassification by the DNN, however the resultant text would be clearly identified by a human as being incorrect.

We will look at two types of popular black-box attacks. Firstly, attacks that employ Generative Adversarial Networks (GANs). Secondly at the creation of a substitute model, to represent the actual DNN we wish to attack.

2.3.1 Generative Adversarial Networks

Attacks that employ Generative Adversarial Networks (GANs)[17] can generate examples that would be considered natural. Zhao et al.[16] present an algorithm that generates adversarial examples that are made up of two parts, firstly a GAN, to generate a fake dataset X of fixed-length vectors and secondly an inverter that maps an input $x \in X$ to its latent representation z'. These two parts are then trained on the original input, the aim being to minimise the reconstruction error between the original input and the adversarial example. The perturbations are introduced in the latent dense space by identifying the perturbed sample $\hat{z}$ in the neighbourhood of z'. To identify the correct $\hat{z}$, two search methods are used: iterative stochastic search and hybrid shrinking search. The main drawback of this approach is that after identification of a potential perturbed example $\hat{z}$ we must query the DNN and check the classification returned, as shown in *Figure 6*. This makes this algorithm extremely time consuming.

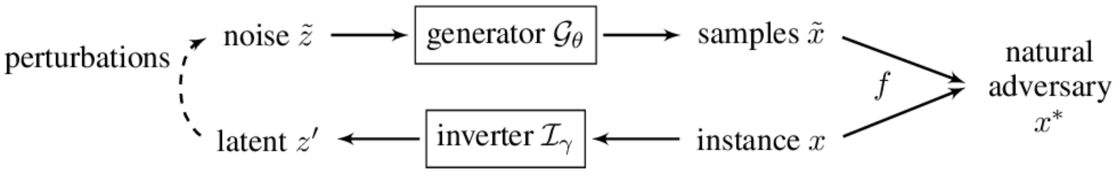


Figure 6 *Natural Adversary Generation. Given an instance of x, our framework generates natural adversaries by perturbing inverted z' and decoding perturbations $\hat{z}$ via G_θ to query the classifier f [16]*

Perturbations are done in a continuous space; hence this approach can be applied to both text and image data, this approach removes the issue of the discrete nature of text data. The method outlined by the authors successfully generated adversarial examples against textual entailment and a machine translation model. The method presented here is built upon GANs and therefore the capabilities of GANS have a direct impact upon the quality of the adversarial examples generated.

The iterative stochastic algorithm used to identify adversaries is computationally intensive, and while switching to a hybrid shrinking search does offer some performance gain, is still computationally expensive.

The adversarial examples generated were however grammatically and semantically like the original input. This is a major step in the right direction. However, the major drawback is the time-consuming nature of this approach.

2.3.2 Substitute Models

Without access to the underlying DNN model we can look to creating a substitute model to represent the actual DNN. Hu et al.[18] outline a substitution method that attacks a DNN for malware detection.

Malware detection is an interesting problem because it is a sequential classification problem. Sequence classification is a modelling problem where you have a sequence of inputs over time and the task is to classify the sequence correctly. The problems are the sequence can vary in length, consist of a large vocabulary of input symbols, and could potentially involve the model having to learn the long-term context or even dependencies between these symbols.

Hu et al.[18] outline two models, firstly a generative DNN model and secondly a substitution DNN model. The first model aims to generates a small adversarial API sequence and insert it after the original input sequence. The second model is a substitute DNN trained to mimic the victim DNN. So, any adversarial examples generated will not query the actual DNN but will instead query this substitute model. The substitute model is trained on a mixture of actual malware sequences and non-malware sequences, as well as the Gumbel-Softmax outputs.

The Gumbel-Softmax is used to jointly train the two DNNs because the original output of the generative DNN model is discrete. The Gumbel-Softmax distribution is a continuous distribution and so allows the gradient to be back-propagated from the generative DNN to the substitute DNN, as shown in *Figure 7*. This approach was novel in the way sequential adversarial examples were generated against a malware detection system. However, it required the training of two DNNs a victim model and an adversarial generator. The issue with this being that the tests were run against the substitute DNN and so the results are not proven to be transferable to a real-world DNN.

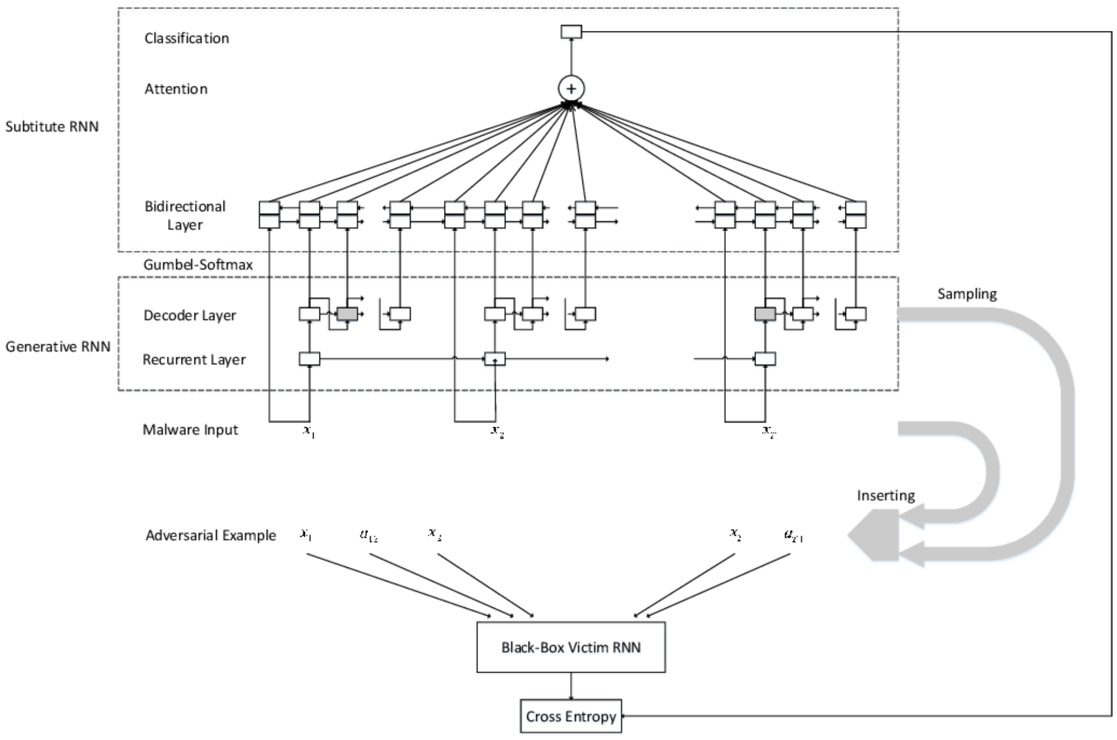


***Figure 7** The architecture of the proposed model when trained on malware[18]*

Creating a substitute model could be of use, where stealth is important. For example, so as not to arouse suspicion through generation of multiple queries to a real-world model. However, the shortcomings of such an approach would be creating a substitute model that sufficiently represents the model under attack.

2.4 Conclusion

Research into adversarial attacks originated in the field of computer vision[6]. A substantial amount of research has been conducted within this domain. Research into adversarial attacks on NLP models, until recently[12,13,18], has been sparse, due to the difficulties of applying methods from the image to the text domain. The difficulties are mainly due to the discrete nature of text data. With discrete data, it becomes much more difficult to make imperceivable and semantically correct changes to the input to generate good adversarial examples. This presents a potential gap in the current research, that will be investigated in this project.

We will create a sentiment analysis model that we will then use to generate adversarial examples against. In the next chapter we will outline the project approach and the steps in creating and testing the sentiment analysis model. We will then move on to present an attack model based on a genetic algorithm, and present how the results of the model will be assessed.

3 Background

This project will investigate the creation of adversarial examples against an NLP model. This research area, compared to similar work with image models, is relatively new and so this project will aim to combine work already done in several similar areas[3,5,7,12,16].

When generating adversarial examples, we will aim to maintain the grammatical and syntactical structure of the input text and so will be attempting to not only replace words with their synonyms, but to minimise the number of words replaced. We will take a black-box approach and assume we have access to only the input and output of the system.

To generate the adversarial examples, we will use a Genetic Algorithm to iteratively evolve a population of adversarial examples. Genetic Algorithms find a solution(s) based on the concept of fitness seen in *Sections 3.1.2*.

3.1 Genetic Algorithms

Genetic Algorithms (GAs) are one branch within the field of evolutionary computation. They are just one type of optimisation algorithm and are used to find the optimal solution for some given computational problem. The aim is to minimise or maximise a given function f. In our case, we are looking to minimise the number of words that we need to replace in the original text to misclassify the input.

Genetic Algorithms use the concept of biological processes of evolution and natural selection, to find answers to problems that have very large search spaces. This is done by continuously generating solutions, evaluating these solutions against a fitness function, and then iteratively refining these solutions, and moving closer to the most optimal one. After evaluation of the fitness of a particular solution, the algorithm moves onto the selection function, which decides which characteristic to keep for future generations. Any characteristics that do not generate an improved fitness score are disregarded from the next iteration.

Most Genetic Algorithms will have five main parts to them, as shown in *Figure 8*.

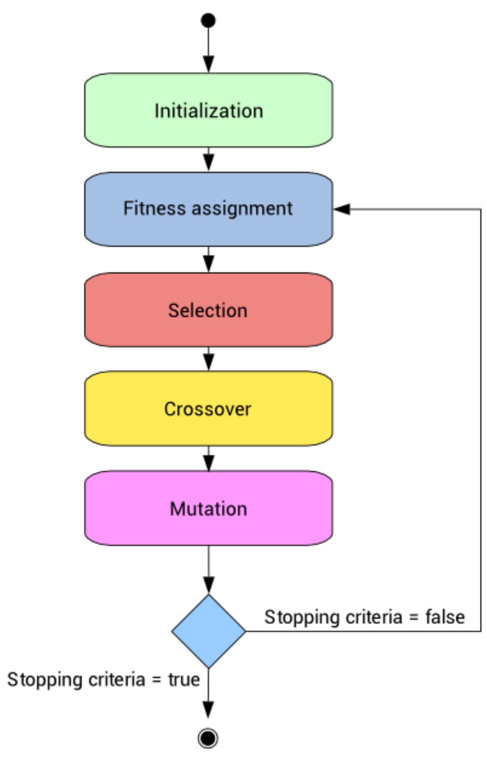


Figure 8 *State diagram for feature selection process of a genetic algorithm*

*(**https://www.neuraldesigner.com/blog/genetic_algorithms_for_feature_selection**)*

As represented in *Figure 8*, the main parts to a Genetic Algorithm are initialisation of a starting population, fitness assignment, selection, crossover, and mutation.

3.1.1 Initialisation

The first step in our Genetic Algorithm will be to generate an initial population. To start with we will generate a population P_0 of size S by calling a function S times to create a set of unique modifications to one of the original training texts. These S individual solution will then be checked for fitness.

3.1.2 Fitness Assignment

In this project we have only two possible classifications for our text: *negative*, and *positive*, our aim is to flip the classification of our original text. If our original text has a classification of *positive*, we will define our *target* classification as *negative*.

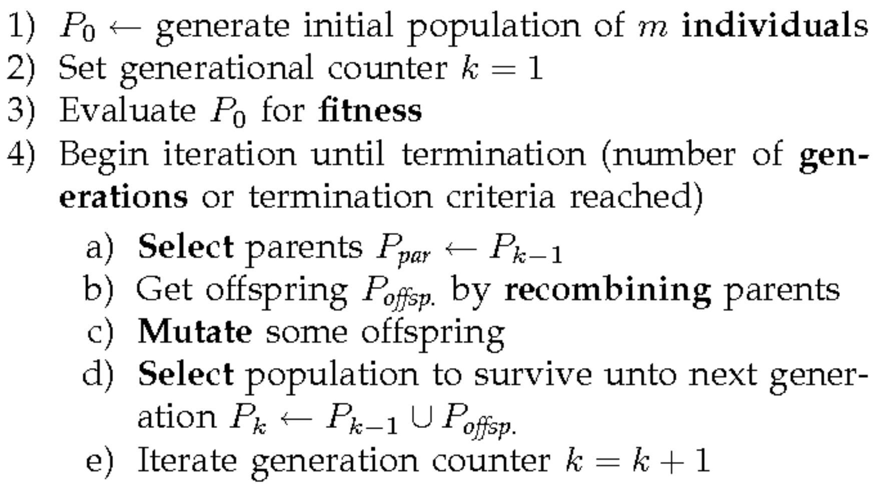
The fitness of each member of the population will be calculated as the probability of that member being labelled as the *target* classification. We will calculate the *target* classification probability by querying our NLP model. If any member of our population has a classification equal to the target, then we have successfully generated an adversarial example, and we stop. If none of the population members has a classification equal to the target, then we move on to the next step, selection.

3.1.3 Selection

The aim of the selection phase is to select the fittest members of the population and let them pass their genes on to the next generation of our population. If none of our population has a classification equal to our target, then we select the pair from our population with the highest fitness score, i.e., the highest probability values, and these two parent text strings will generate a new child text string by calling the crossover and mutation function.

3.1.4 Crossover and Mutation

We assume the two texts strings are statistically independent of each other, and so by independently sampling from both using a uniform distribution, we construct a new child text string. This new child string will then be used to generate a new population and again passed to the fitness function. This loop will repeat either until one of the population members is equal to the target classification, or until we have iterated through the loop a set number of generations, whichever occurs first. As shown in Figure 9[19].



```
1) $P_0 \leftarrow$ generate initial population of $m$ **individuals**
2) Set generational counter $k = 1$
3) Evaluate $P_0$ for **fitness**
4) Begin iteration until termination (number of **generations** or termination criteria reached)
    a) **Select** parents $P_{par} \leftarrow P_{k-1}$
    b) Get offspring $P_{offsp.}$ by **recombining** parents
    c) **Mutate** some offspring
    d) **Select** population to survive unto next generation $P_k \leftarrow P_{k-1} \cup P_{offsp.}$
    e) Iterate generation counter $k = k + 1$
```

Figure 9 *Genetic Algorithm Pseudocode[19]*

4 Tools Used

4.1 Anaconda

We will use Anaconda, a free distribution of the programming language Python. This comes with a package manager, Conda, which we will use to ensure a consistent virtual development environment.

Python will be the main language used throughout this project. The reason for this choice is the availability of stable and tested libraries specifically aimed at AI-based and ML projects. The implementation of AI and ML algorithms from first principles for this project would be an unreasonable task to undertake, give the time constraints and scope of the project. Python offers several libraries and frameworks to enable rapid prototyping and testing of ML models.

Anaconda is by far the most popular free machine learning environment, with far greater maturity and community support than any alternative, hence was a clear choice for this kind of development.

The libraries used in this project are outlined briefly in the following sections.

4.2 TensorFlow, Keras

TensorFlow[24] is an open-sourced deep learning library developed and maintained by Google. TensorFlow excels at numerical and large-scale computations required for most ML tasks. TensorFlow includes a library of common deep learning models and algorithms. These models and algorithms are coded in C++; however, a Python API is provided to access them, which means that the actual mathematical operations are not performed in Python, instead these operations are redirected by Python code via 'hooks' to the high-performant C++ binaries. The major benefit to this project of TensorFlow is *abstraction*, this will enable us to focus on the logic of the code without having to consider the details of implementation of individual algorithms.

TensorFlow excels at executing low-level tensor operations on the CPU and GPU. This requires a certain level of understanding of the mathematics behind each algorithm. If we are planning to build a model that is fit for a production environment, we should indeed understand and have lower-level knowledge of each algorithm used and its implementation details. However, for rapid prototyping and iteration of different models we will use Keras[25]. Keras provides another level of abstraction from TensorFlow, it is a Python based framework, which also means it is much simpler to debug and explore. The framework enables building models one block at a time, this will enable us to rapidly build models and to add or remove layers to quickly test our model.

4.3 Libraries

We will make use of the following libraries:

NumPy for scientific computing and data analysis, we will used it for working with arrays, and for linear algebra, and matrix calculations

Pandas for importing our data and general data analysis

5 Methodology

To evaluate our adversarial attack method, we first need to train a sentiment analysis model to use as our victim model. Below we will detail the steps required to create our NLP sentiment analysis model. We will outline the dataset used in *Section 5.1*, and then describe in detail word embeddings with a focus on the Global Vectors model (GloVe)[4] word embeddings model in *Section 5.2*.

In *Section 5.3*, we outline the training of our sentiment model along with some summary statistics on the accuracy of the model in *Section 5.3.6*.

The Python code for the creation of our NLP model is listed in Appendix A.

5.1 Dataset

We trained our sentiment analysis model using the IMDB Movie Review[22] dataset of 50,000 annotated reviews (0 = negative, 1 = positive). This dataset is split into 25,000 positive and 25,000 negative movie reviews.

A review is considered negative if it had a score of ≤ 4 out of 10 stars on the IMDB website, and positive if it had a score of ≥ 7 out of 10 stars. Any reviews with ratings of 5 or 6 were excluded for the dataset.

5.2 Word Embeddings

Image data is continuous therefore to measure the similarity between a clean data point and a perturbed data point we would typically use the Minkowski Distance[9], L_p norm measure, using the below generalised formula:

$$\left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{1/p}$$

Some common values of p are:

- $p = 1$, Manhattan distance
- $p = 2$, Euclidean distance
- $p = \infty$, Chebychev distance

Text data is discrete. So, the above measure of similarity would not be appropriate. To overcome this, we need some way of transforming the discrete text data to continuous data.

We will focus on word embeddings, which is a general term for any model that maps a set of words or phrases in a corpus to a set of real valued vectors.

A popular model is the Global Vectors for Word Representation (GloVe)[4]. GloVe allows for better word embeddings by creating a word-context matrix. Basically, it creates a measure to indicate that certain words are more likely to be seen in context of others. For example, the word “lettuce” is likely to be seen in the context of “salad” but not with “train

These vector representations of words can capture context. Context is implied through measuring the spatial distance between these word vectors, the shorter the distance between word vectors the stronger the relationship between the actual words.

For example, if we have the following two sentences (our corpus):

i. "I enjoy playing golf"

ii. "I enjoy playing tennis"

If we construct a vocabulary V from these two sentences, we have:

$$V = \{I, enjoy, playing, golf, tennis\}$$

From this vocabulary we can create a one-hot encoded vector for each word in our vocabulary:

$$I = [1,0,0,0,0]^T$$

$$enjoy = [0,1,0,0,0]^T$$

$$playing = [0,0,1,0,0]^T$$

$$golf = [0,0,0,1,0]^T$$

$$tennis = [0,0,0,0,1]^T$$

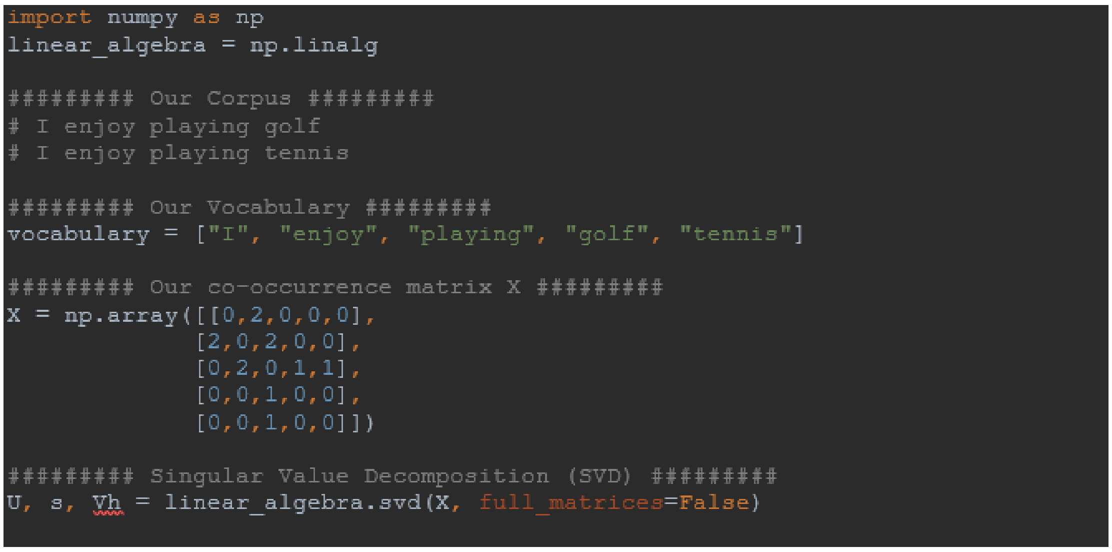
We can imagine we have 5 (i.e., size of V) dimensions in space, and each of the words in our vocabulary is in one of these dimensions, which also implies that none of the words in our vocabulary are related as they occupy a separate point in our 5-dimensional space. However, the words $golf$ and $tennis$ are related, we know this intuitively. Our aim is to have words that are similar be close to each other in our n-dimensional space i.e., the value for Euclidian distance between word vectors should be small for closely related words.

In our one-hot encoded vector, all the words are thought of as being independent of one another. However, we want to be able to represent the dependence of one word on another, so that words in the same context receive a greater share of this dependence compared to those outside of this context. We first go through the entire corpus and create a co-occurrence matrix, where we count each word (i.e., the rows in the matrix), and calculate how frequently we see the word in the same context (i.e., the column) in our corpus, as shown in *Figure 10*.

counts	I	enjoy	playing	golf	tennis
I	0	2	0	0	0
enjoy	2	0	2	0	0
playing	0	2	0	1	1
golf	0	0	1	0	0
tennis	0	0	1	0	0

Figure 10 *Example Co-occurrence matrix, with window size = 1*

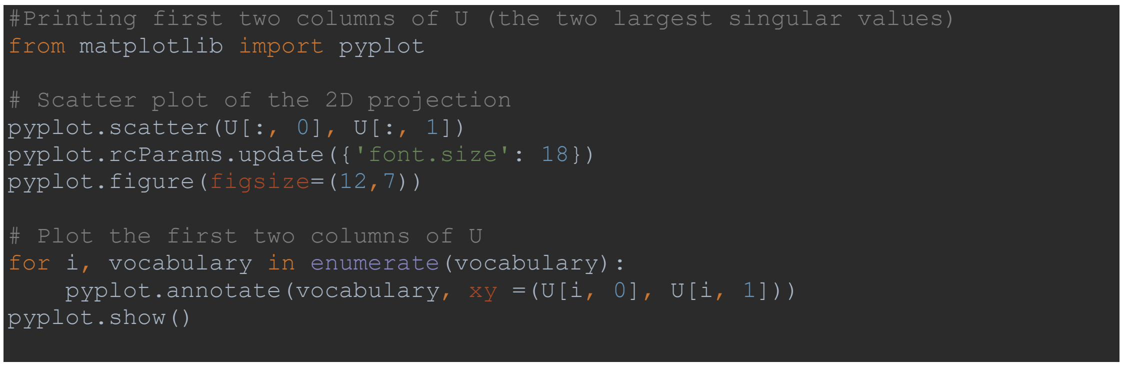
From *Figure 10*, we can see that in building the co-occurrence matrix we have already started to capture some very simple overlaps, between $\{golf\}$ and $\{tennis\}$ and we see they both co-occur with $\{playing\}$. This is a simple co-occurrence matrix, however for a larger corpus this would grow rapidly. Therefore, the next step is to reduce the dimensionality of this matrix via Singular Value Decomposition (SVD) of our co-occurrence matrix. Shown in the code listing in *Figure 11* below is an example of how this simple example would translate into Python code.



```python
import numpy as np
linear_algebra = np.linalg

######### Our Corpus #########
# I enjoy playing golf
# I enjoy playing tennis

######### Our Vocabulary #########
vocabulary = ["I", "enjoy", "playing", "golf", "tennis"]

######### Our co-occurrence matrix X #########
X = np.array([[0,2,0,0,0],
              [2,0,2,0,0],
              [0,2,0,1,1],
              [0,0,1,0,0],
              [0,0,1,0,0]])

######### Singular Value Decomposition (SVD) #########
U, s, Vh = linear_algebra.svd(X, full_matrices=False)
```

Figure 11 *Singular Value Decomposition in Python code*

If we now plot the first two columns of the matrix U (left singular vectors). We can see that words that are similar are plotted closer together, as show in *Figure 12*. The words $\{golf, tennis\}$ are very close together, overlapping, and we can say are related in some way.



```python
#Printing first two columns of U (the two largest singular values)
from matplotlib import pyplot

# Scatter plot of the 2D projection
pyplot.scatter(U[:, 0], U[:, 1])
pyplot.rcParams.update({'font.size': 18})
pyplot.figure(figsize=(12,7))

# Plot the first two columns of U
for i, vocabulary in enumerate(vocabulary):
    pyplot.annotate(vocabulary, xy =(U[i, 0], U[i, 1]))
pyplot.show()
```

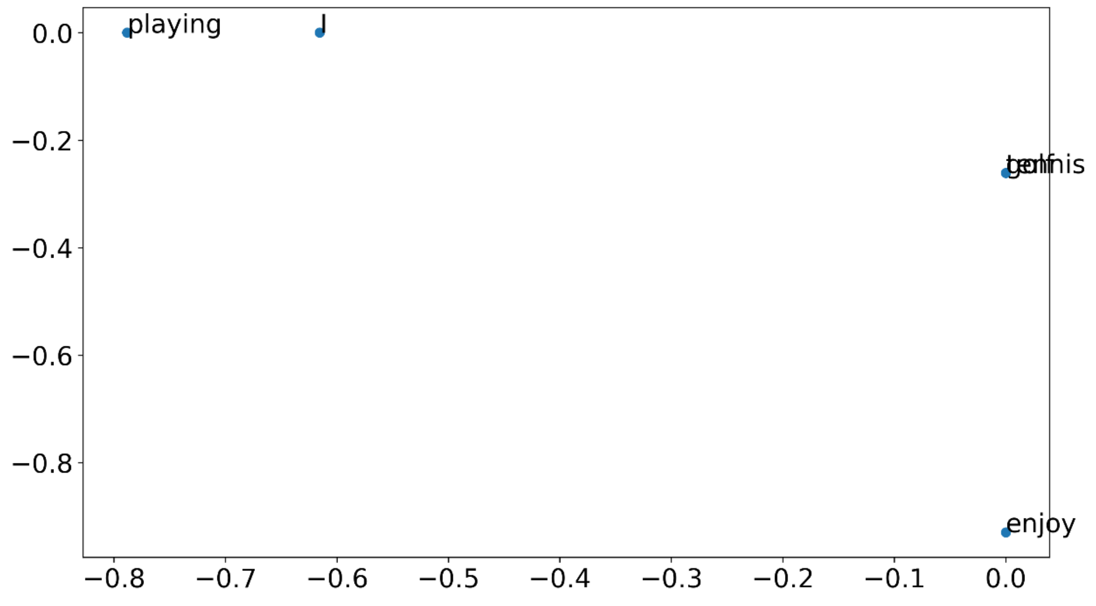


Figure 12 *Plotting the first two columns of U (the two largest singular values)*

We can see that the words $\{I, enjoy, playing\}$ are frequently occurring words and so are further away, and we can see that the words $\{I, playing\}$ are nearest neighbours (in our corpus). From this very simple example we can get a very intuitive feel for what word vectors can capture.

Rohde et. al.[23] used the SVD method and further refined it to produce co-occurrence matrices to show the numerous relationships between words. It is common to have 100 and even 300 dimensional vectors for words. To represent this visually Rohde et. al.[23] chose a few words and graphed them in a hierarchical cluster showing the nearest neighbours, *Figure 13*.

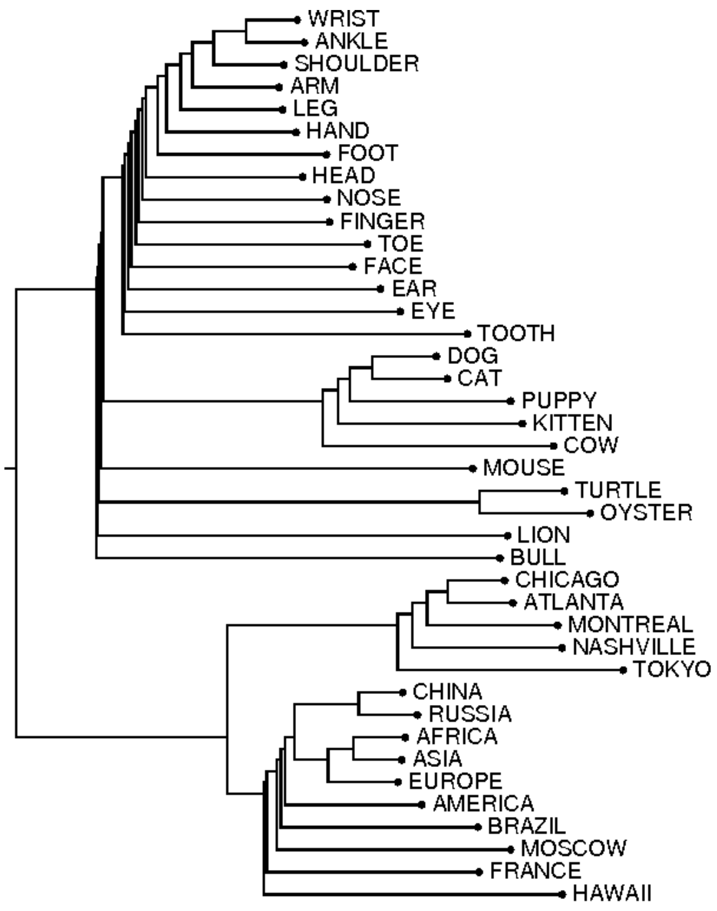


Figure 13 *Hierarchical clustering using distances based on vector correlations[23]*

From *Figure 13*, we can see that $\{WRIST\}$ is very close to $\{ANKLE\}$ as are other extremities on the human body, and that cities are also clustered together, and American cities are closer together then cities in other countries. Rohde et. al.[23] also noticed and graphed the syntactical patterns that emerged in the vectors, *Figure 14*.

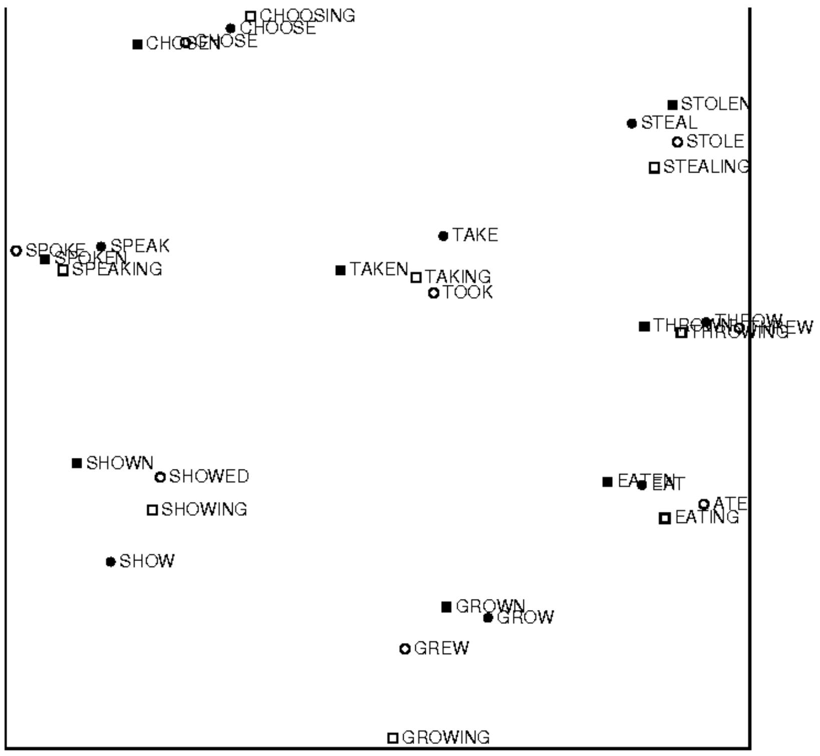


Figure 14 *Scaling of present, past, progressive, and past participle forms for eight verb families[23]*

The issue with using SVD is in terms of computational cost, which scales quadratically for an $n * m$ matrix giving $O(mn^2)$, this does not scale well when we have a corpus with millions of words. Each time a new word is added, we would need to re-run these calculations to add them to our co-occurrence matrix and vector space.

GloVe[4] addresses the shortcomings of the SVD method, and provides a more efficient, scalable, and high performant method that works on small and large corpuses and vector sizes.

For this project we used pre-trained GloVe word embeddings to train the NLP model. The embedding space created by GloVe likely contains all the words we will encounter in our movie reviews, so we can use these vector representations instead of creating our own from a much more limited vocabulary set.

5.3 Sentiment Analysis Model Training

The steps taken to train the sentiment analysis model were:

1. Dataset Loading & Pre-processing
2. Train and Test Data Split
3. Tokenization
4. Word Embedding
5. Model Training
6. Model Test & Evaluation

5.3.1 Dataset Loading & Pre-processing

In this project we will use the IMDB Movie Reviews dataset, see *Section 5.1* above. The reviews are evenly split evenly between positive and negative sentiments as shown in *Figure 15*.

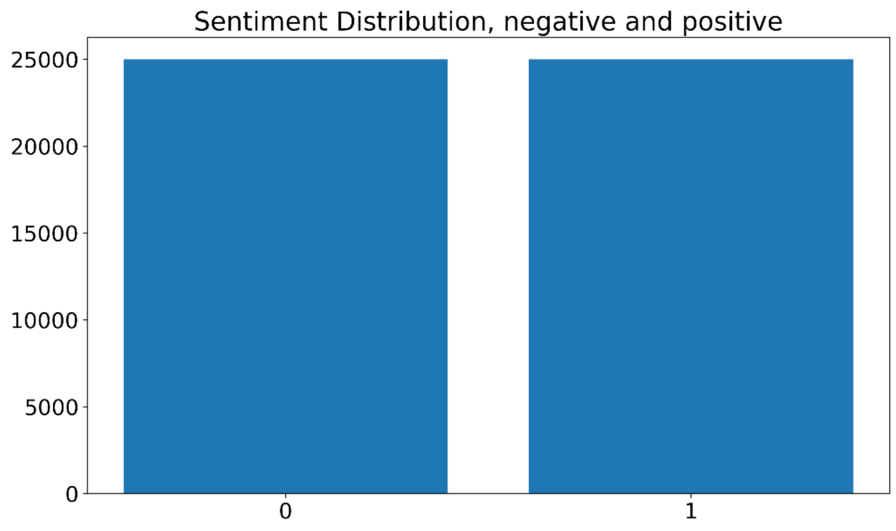
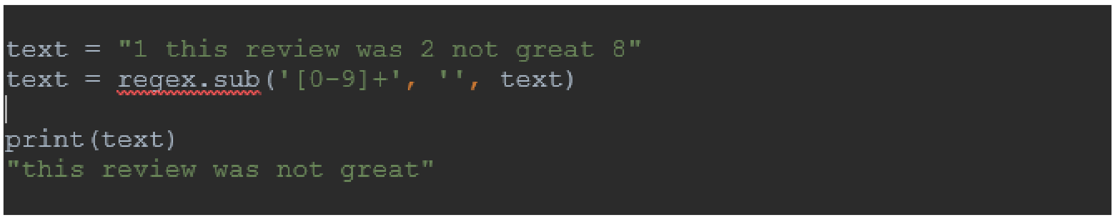


Figure 15 *Data distribution between positive(1) and negative(0) sentiments*

The dataset is loaded from disk and then pre-processed to normalize the data. This involves removing punctuation, URLs, special characters, and any other characters that would not add any value to our model. This was done by defining regular expressions and then using them against the text of each movie review to remove the matching pattern. *Figure 16* shows a simple regular expression example in Python to remove all digits found in a string.



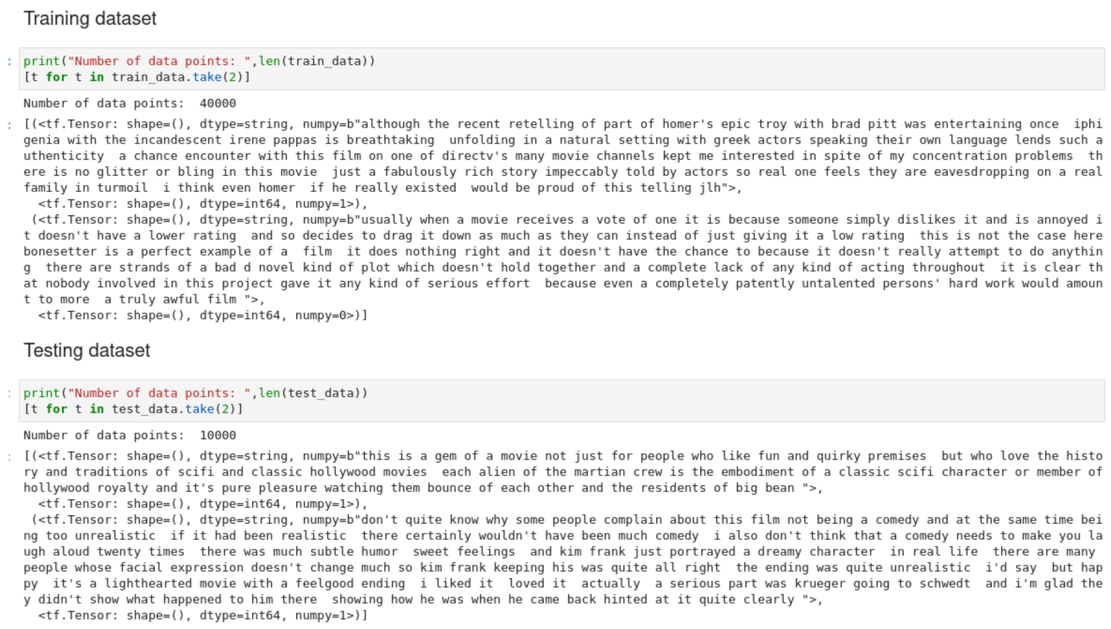
```
text = "1 this review was 2 not great 8"
text = regex.sub('[0-9]+', '', text)

print(text)
"this review was not great"
```

Figure 16 *Simple regular expression example*

After normalisation of the data, we have a clean dataset that contains only the text we will require for our model training, and we move onto splitting the data into a training set and a test set.

5.3.2 Train and Test Data Split

Prior to splitting the dataset into a training and test set. We combine the positive and negative reviews into one dataset, we then shuffle and randomise the data. Once randomised the dataset is split into 80% (40,000 data points) for training and 20% (10,000 data points) for testing, as shown in *Figure 17*.



Training dataset

```
print("Number of data points: ",len(train_data))
[t for t in train_data.take(2)]
```

```
Number of data points:  40000
```

```
[(<tf.Tensor: shape=(), dtype=string, numpy=b"although the recent retelling of part of homer's epic troy with brad pitt was entertaining once  iphi
genia with the incandescent irene pappas is breathtaking  unfolding in a natural setting with greek actors speaking their own language lends such a
uthenticity  a chance encounter with this film on one of directv's many movie channels kept me interested in spite of my concentration problems  th
ere is no glitter or bling in this movie  just a fabulously rich story impeccably told by actors so real one feels they are eavesdropping on a real
family in turmoil  i think even homer  if he really existed  would be proud of this telling jlh">,
  <tf.Tensor: shape=(), dtype=int64, numpy=1>),
 (<tf.Tensor: shape=(), dtype=string, numpy=b"usually when a movie receives a vote of one it is because someone simply dislikes it and is annoyed i
t doesn't have a lower rating  and so decides to drag it down as much as they can instead of just giving it a low rating  this is not the case here
bonesetter is a perfect example of a  film  it does nothing right and it doesn't have the chance to because it doesn't really attempt to do anythin
g  there are strands of a bad d novel kind of plot which doesn't hold together and a complete lack of any kind of acting throughout  it is clear th
at nobody involved in this project gave it any kind of serious effort  because even a completely patently untalented persons' hard work would amoun
t to more  a truly awful film ">,
  <tf.Tensor: shape=(), dtype=int64, numpy=0>)]
```

Testing dataset

```
print("Number of data points: ",len(test_data))
[t for t in test_data.take(2)]
```

```
Number of data points:  10000
```

```
[(<tf.Tensor: shape=(), dtype=string, numpy=b"this is a gem of a movie not just for people who like fun and quirky premises  but who love the histo
ry and traditions of scifi and classic hollywood movies  each alien of the martian crew is the embodiment of a classic scifi character or member of
hollywood royalty and it's pure pleasure watching them bounce of each other and the residents of big bean ">,
  <tf.Tensor: shape=(), dtype=int64, numpy=1>),
 (<tf.Tensor: shape=(), dtype=string, numpy=b"don't quite know why some people complain about this film not being a comedy and at the same time bei
ng too unrealistic  if it had been realistic  there certainly wouldn't have been much comedy  i also don't think that a comedy needs to make you la
ugh aloud twenty times  there was much subtle humor  sweet feelings  and kim frank just portrayed a dreamy character  in real life  there are many
people whose facial expression doesn't change much so kim frank keeping his was quite all right  the ending was quite unrealistic  i'd say  but hap
py  it's a lighthearted movie with a feelgood ending  i liked it  loved it  actually  a serious part was krueger going to schwedt  and i'm glad the
y didn't show what happened to him there  showing how he was when he came back hinted at it quite clearly ">,
  <tf.Tensor: shape=(), dtype=int64, numpy=1>)]
```

Figure 17 *Data split between training and test samples*

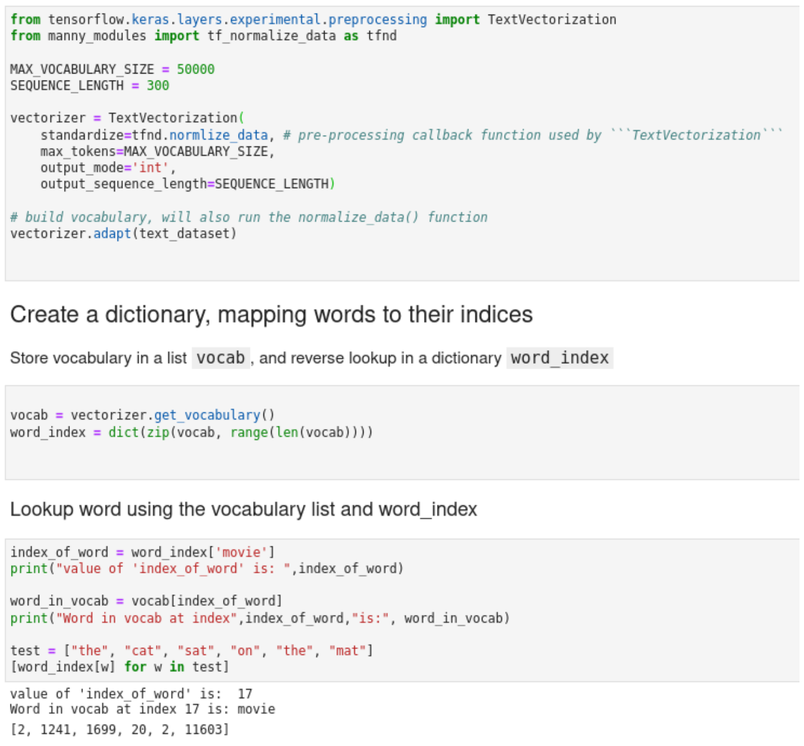
5.3.3 Tokenization

Tokenization is the breaking down of the raw text of each movie review into smaller chunks. At this step we split the string into individual words called tokens. These tokens will aid in understanding the context and eventually in developing our sentiment analysis model. As stated in *Section 5.2*, text must be represented as real values for the data to be used in a ML model, this mapping is done at this tokenization step. This mandatory step is required prior to training any NLP type model.

These tokens are then used to create a vocabulary. A vocabulary can be thought of as the set of all unique tokens in our corpus. In our vocabulary we limit it to the first k frequently occurring words, where we set $k = 50{,}000$.

In this project we split the text string on white space. This works well because our language of interest in English, which uses white space to break apart sentences into meaningful words. We will use the TensorFlow *TextVectorization* layer to produce both our tokens and vocabulary file. This layer performs pre-processing of our raw movie reviews, text normalization, tokenization and vocabulary creation and indexing.

We designed a function to take as input each review and to process the string, firstly to lowercase the entire string and then to remove all special characters and punctuation, this function was then used as a call-back function in the *TextVectorization* layer, *Figure 18*.



```
from tensorflow.keras.layers.experimental.preprocessing import TextVectorization
from manny_modules import tf_normalize_data as tfnd

MAX_VOCABULARY_SIZE = 50000
SEQUENCE_LENGTH = 300

vectorizer = TextVectorization(
    standardize=tfnd.normlize_data, # pre-processing callback function used by ```TextVectorization```
    max_tokens=MAX_VOCABULARY_SIZE,
    output_mode='int',
    output_sequence_length=SEQUENCE_LENGTH)

# build vocabulary, will also run the normalize_data() function
vectorizer.adapt(text_dataset)
```

Create a dictionary, mapping words to their indices

Store vocabulary in a list `vocab`, and reverse lookup in a dictionary `word_index`

```
vocab = vectorizer.get_vocabulary()
word_index = dict(zip(vocab, range(len(vocab))))
```

Lookup word using the vocabulary list and word_index

```
index_of_word = word_index['movie']
print("value of 'index_of_word' is: ",index_of_word)

word_in_vocab = vocab[index_of_word]
print("Word in vocab at index",index_of_word,"is:", word_in_vocab)

test = ["the", "cat", "sat", "on", "the", "mat"]
[word_index[w] for w in test]
```

```
value of 'index_of_word' is:  17
Word in vocab at index 17 is: movie
[2, 1241, 1699, 20, 2, 11603]
```

Figure 18 *TextVectorization layer definition*

We also limited the size of the string to the first 300 tokens for each review, this ensures all the sequences are of the same length and will speed up training of our model. We require all sequences to be of the same length because the learning libraries in TensorFlow assume a vectorized representation of our training data. This standard vectorized representation allows the code to perform efficient matrix operations on the data.

5.3.4 Word Embedding

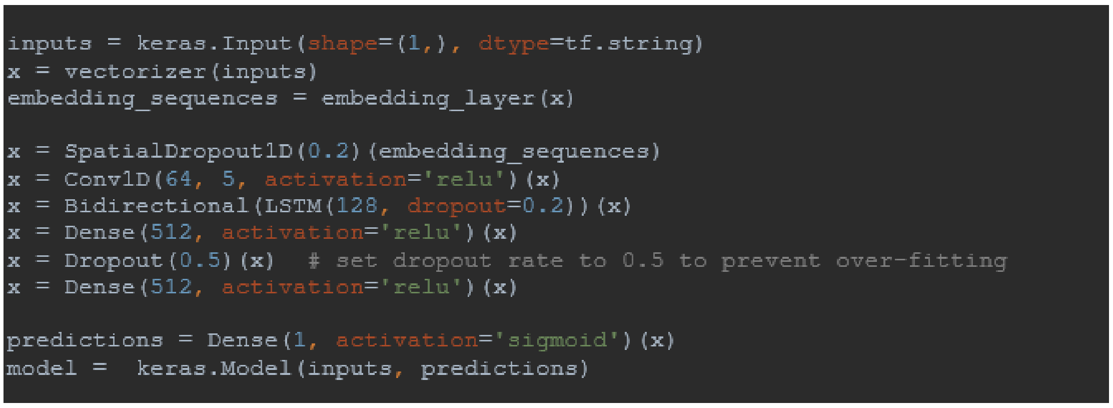
As stated in *Section 5.2*, word embeddings encode similarities between words, as learned for the specific dataset the model was trained on.

In this project we use the GloVe pre-trained word embeddings. The word embeddings layer is a vector representation of each word. In the case of the pre-trained GloVe embedding, this is nothing more than a lookup table, in our case consisting of 50,000 rows, representing each word in our vocabulary and 300 columns (dimensions). These vectors are used to encode the similarity of words and are used to train our sentiment analysis model. Using pre-trained embeddings means that we do not need to train this layer in our model, and indeed set the parameter Trainable=False because we do not want this layer to be updated during our model training, instead we want the model to use the pre-trained weights.

Word embeddings are dependent upon the corpus they are trained on. This can mean that they capture inherent biases within the corpus used. As noted by Brunet et. al.[26] GloVe word embeddings could acquire stereotypical human biases through the text data it is trained on. Deploying these word embeddings in certain ML tasks could amplify these biases, however our project does not produce output that is consumed by any other ML model. We use the embeddings purely for categorising of reviews. If the data were to be used for translation or for hiring aids, this could become an issue and should be kept in mind as noted by Angwin et. al[27] and Caliskan et. al.[28].

5.3.5 Model Training

We will use the Functional API in TensorFlow to create our model, as shown in *Figure 19*.



```
inputs = keras.Input(shape=(1,), dtype=tf.string)
x = vectorizer(inputs)
embedding_sequences = embedding_layer(x)

x = SpatialDropout1D(0.2)(embedding_sequences)
x = Conv1D(64, 5, activation='relu')(x)
x = Bidirectional(LSTM(128, dropout=0.2))(x)
x = Dense(512, activation='relu')(x)
x = Dropout(0.5)(x)  # set dropout rate to 0.5 to prevent over-fitting
x = Dense(512, activation='relu')(x)

predictions = Dense(1, activation='sigmoid')(x)
model =  keras.Model(inputs, predictions)
```

Figure 19 *Model definition in TensorFlow*

- First, we create an Input object. This specifies that our model will accept a single string as input.
- Next, we pass this input to our *text_vectorization* layer, which will clean and normalize the string according to the set of normalization steps we defined in the call-back function *tf_normalize_data.py*, shown in *Appendix A*. The input will also be set to a fixed size of 300 tokens, and converted to an array of integer values, where each word has been replaced by a unique integer value corresponding to a unique integer value from our vocabulary list.
- Once normalized, we pass this array to our pre-trained GloVe embeddings layer. Since we are using a pre-trained embeddings layer, we do not want these values to update as we train our model. So, we initialise this layer with our GloVe embeddings and set the parameter *trainable=False*. These values are then used as the weights in our model.
- The next layer, SpatialDropout1D is used for regularization. This is a process to reduce the number of coefficients in our model. This has the effect of discouraging the learning of an overly complex or too flexible a model. The aim being to prevent overfitting, and produce a generalized model, that works well on unseen data.
- Next, we have the convolution layer Conv1D. In this layer we define a filter (feature detector) of height (kernel size) 5. We define 64 of these filters to enable learning of 64 different features on this layer of our model. The output of this layer is a matrix of size 296 x 64. Each column of the output matrix holds weights of one single filter. With the defined height and considering the length of the input matrix, each filter will contain 296 weights.
- The next layer is the Long Short-Term Memory (LSTM)[29]. LSTM networks were first introduced by Hochreiter et. al. [29]. They are used to solve issues that RNNs have with short term memory. If a sequence of text is too long, the DNN will have problems with propagating information from earlier time steps to later ones. In practice this means that if the RNN is processing some unstructured text as input, the RNN may leave out entirely some important pieces of information. Hence, we add this layer here to counter this effect.

- Next, we add a Dense layer, this is simply a layer where each neuron is connected to each neuron of the next layer. We specify an output size for the layer as 512 and the activation function to be the rectified linear activation function (ReLU).
- We also add a Dropout layer. This layer will randomly assign 0 weights to the units in the network. We chose a dropout rate of 0.5 i.e. 50% of the units will receive a zero weight. By doing this, the RNN becomes less sensitive to small variations in our dataset. This will further increase our accuracy on unseen data and aid in generalizing our final model.
- The final layer is a fully connected layer with Sigmoid activation. This final layer will reduce our vector from height 512 to a vector of one. This is because we will use the sigmoid activation to return a "*probability*" of the input being positive. The Sigmoid function is used to force the returned value to be between 0 and 1, so that we can treat this as a probability value.

Our model uses the Adam(***Ada****ptive* ***m****oment* estimation) algorithm[30] for optimization. Adam optimization is an extension of Stochastic gradient descent, it updates the weights more efficiently, leading to much faster convergence times.

The layers of our final model, including the input and output vectors at each layer can be seen in *Figure 20*.

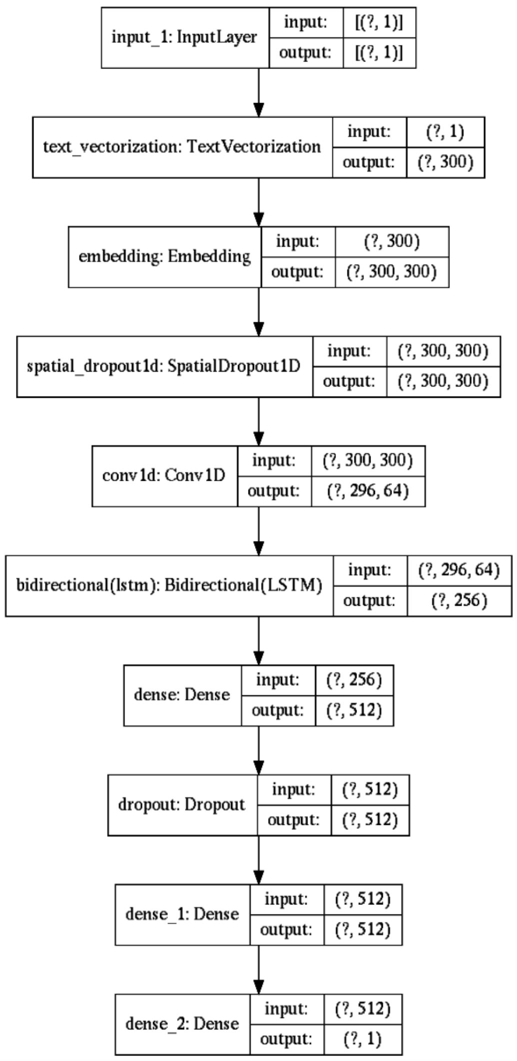

input_1: InputLayer | input: [(?, 1)] | output: [(?, 1)]
text_vectorization: TextVectorization | input: (?, 1) | output: (?, 300)
embedding: Embedding | input: (?, 300) | output: (?, 300, 300)
spatial_dropout1d: SpatialDropout1D | input: (?, 300, 300) | output: (?, 300, 300)
conv1d: Conv1D | input: (?, 300, 300) | output: (?, 296, 64)
bidirectional(lstm): Bidirectional(LSTM) | input: (?, 296, 64) | output: (?, 256)
dense: Dense | input: (?, 256) | output: (?, 512)
dropout: Dropout | input: (?, 512) | output: (?, 512)
dense_1: Dense | input: (?, 512) | output: (?, 512)
dense_2: Dense | input: (?, 512) | output: (?, 1)

Figure 20 *Visualization of sentiment analysis model layers*

5.3.6 Model Test & Evaluation

The model was trained over 40 epochs with a learning rate set to $1 * 10^{-4}$. The history of the training and validation loss was plotted in *Figure 21* and the training and validation accuracy was plotted in *Figure 22*, to track the improving accuracy of our model after each epoch.

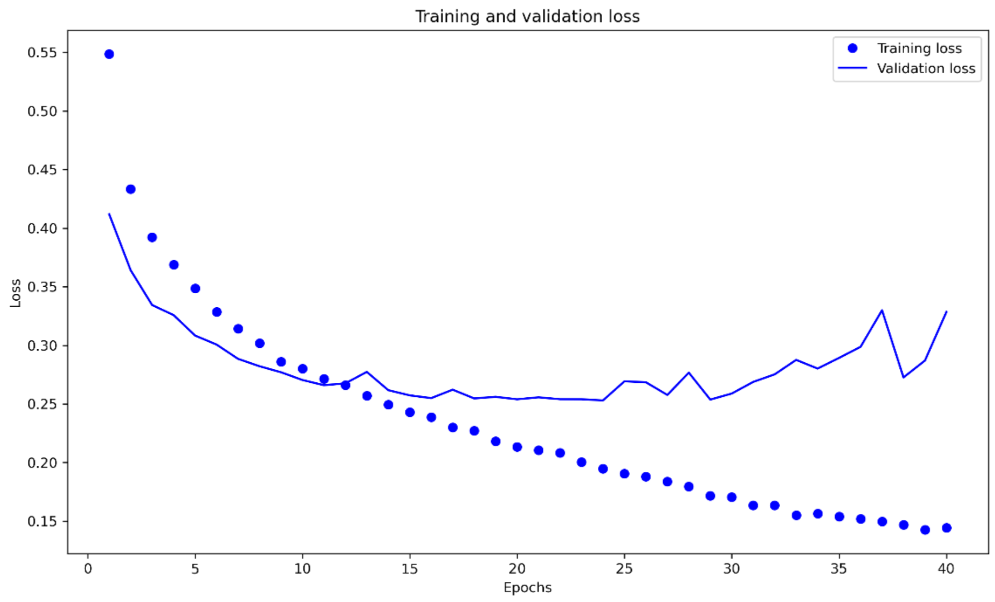


Figure 21 *Training and validation loss over 40 epochs*

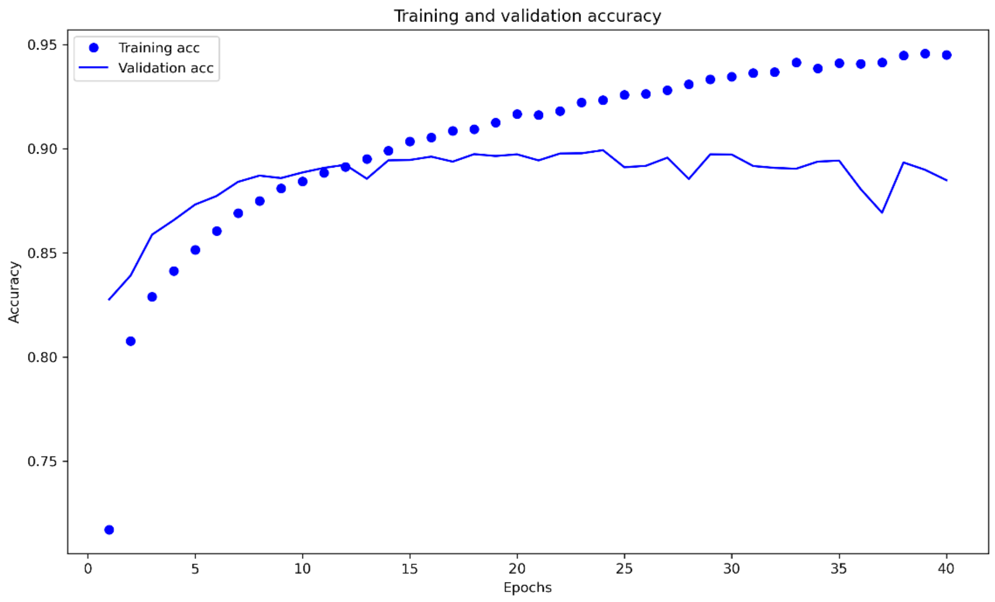
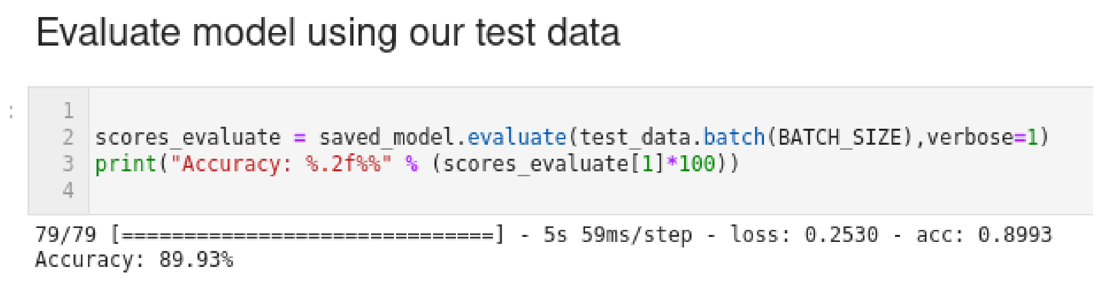


Figure 22 *Training and validation accuracy over 40 epochs*

As can be noted from the *Figure 21* and *Figure 22*, the validation loss plateaus around 20 epochs, as does validation accuracy. We could have set the epochs to 20, to speed up the training as after this point there is negligible improvement in the model's performance.

Once trained the model was evaluated on the test dataset, as shown in *Figure 23*. We recorded an accuracy of 89.93%. In *Figure 24* we have the ROC (Receiver Operating

Characteristic) curve for our model, we have a threshold of 0.899, which is close to 1 and hence considered an excellent model.



Evaluate model using our test data

```
scores_evaluate = saved_model.evaluate(test_data.batch(BATCH_SIZE),verbose=1)
print("Accuracy: %.2f%%" % (scores_evaluate[1]*100))
```

```
79/79 [==============================] - 5s 59ms/step - loss: 0.2530 - acc: 0.8993
Accuracy: 89.93%
```

Figure 23 *Model evaluation on test dataset*

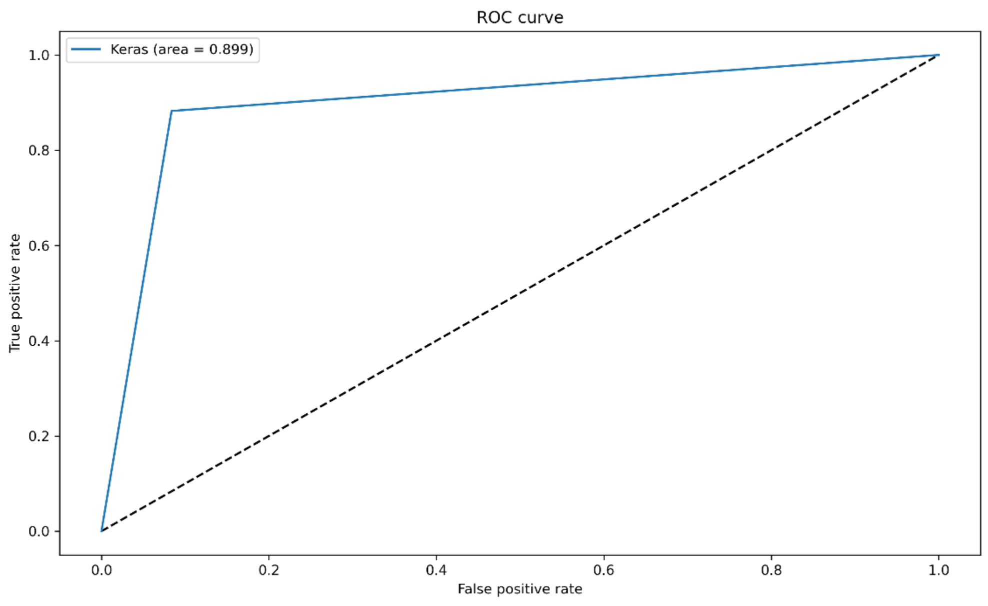
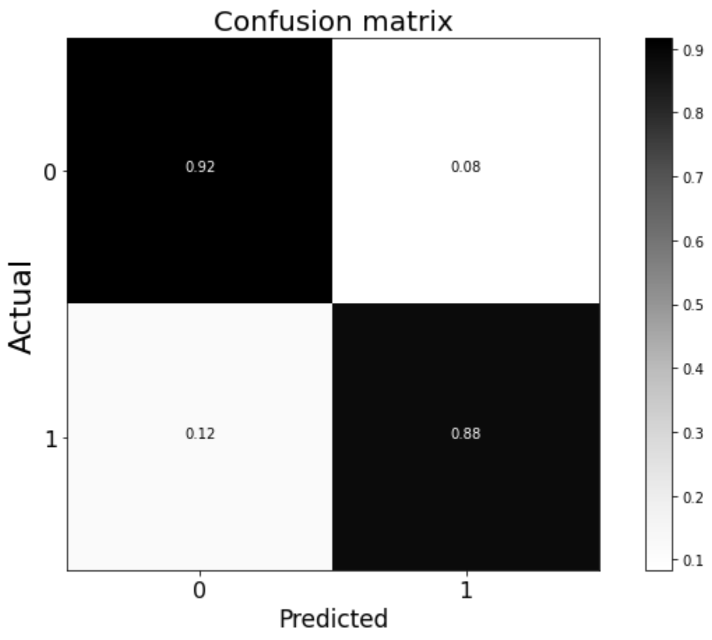


Figure 24 *ROC Curve showing the performance of our model at all thresholds*

The results for our confusion matrix are shown in ***Figure 25***.



```
              precision    recall  f1-score   support

           0       0.89      0.92      0.90      4993
           1       0.91      0.88      0.90      5007

    accuracy                           0.90     10000
   macro avg       0.90      0.90      0.90     10000
weighted avg       0.90      0.90      0.90     10000
```

Figure 25 *Confusion matrix for model performance on the test set of 10000 data items (0 = negative sentiment, 1 = positive sentiment)*

From *Figure 25*, we see that our true-negative rate is 0.92 and our true positive is 0.88. We have a recall of 0.92 for negative reviews and 0.88 for positive reviews. Our F-Score, the harmonic mean of recall and accuracy of our model, is 0.9, which on our binary classification task is an excellent indication that our model has generalised well and is highly likely to make good predictions on unseen data.

6 Development of Attack Algorithm

6.1 Solution Overview

As mentioned above in *Section 3*, we will use a black-box approach for generating adversarial examples. We assume we do not have access to the internals of the sentiment analysis model. Our aim is to generate the highest scoring solution, that is the solution with the least number of modifications to our original text, and one that maintains semantic similarity after modifications.

With this aim in mind, we developed a genetic algorithm that iteratively evolves a population of possible solutions that take us toward our optimal solution. Each population that we generate is called a *generation*. In each generation we evaluate population members against a *fitness function*. Our aim is to minimise the number of changes to each population; therefore, we select the two highest scoring population members of each generation and use them to *breed* the next generation. The next generation is generated through a *crossover* and *mutation* function. The crossover function takes the two best population members to create a *child solution*. We use mutation to introduce *diversity* and thereby enable wider exploration of our search space. Using this genetic algorithm approach, we hope to find adversarial examples that have the least number of modifications, and a solution that maintains semantic correctness.

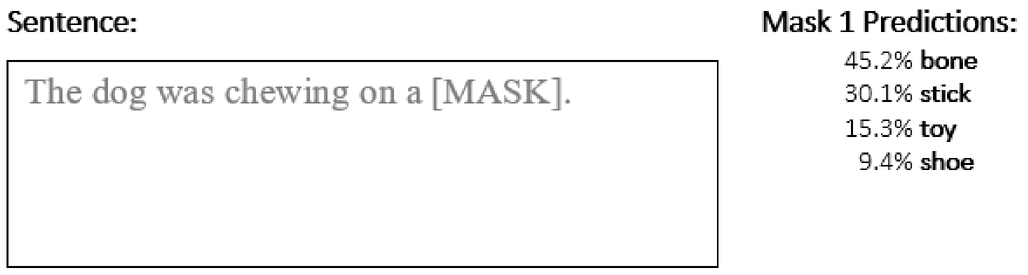
6.1.1 Generate Initial Population and mutation

Expanding upon our discussion of initialisation in *Section 3.1.1*. We first generate an initial population P_0 of size S by calling a function S times to create a set of unique modifications to one of the original training texts.

This is done by randomly selecting a word w in the text and generating *replacements* for that word. Stop-words (e.g., *the*, *and*, *to*, *a* …) are excluded from being selected. The generation of replacement words is done via a mutation function. This mutation function is also applied to offspring after the crossover operation (see 6.1.4). In our project we have created *three separate methods*, *A*, *B* and *C*, for generating *replacement* words for our sentence:

A. In the first method we return a set of synonyms for w. The number of synonyms in the set is limited to $max\_neighbours$, we then swap each synonym with the original word and query our sentiment model, the final selection of which synonym to swap with w is the one that takes us furthest toward our $target$ label.

B. In our second method we use a language representation model called *BERT* (**B**idirectional **E**ncoder **R**epresentations from **T**ransformers)[34]. The first method described above returns a set of synonyms for our selected word, w, however the synonyms generated do not consider the context of our selected word w. In this second method we, again, randomly select a word w, however this time we select three words before and three words after w to give our selected word context, as shown in ***Figure 26***. The words returned, limited to the top 5 only, are given a probability, which is dependent upon the context given by the surrounding words. The selection of the word to swap for w is the one that takes us furthest toward our $target$ label.


Sentence:

The dog was chewing on a [MASK].

Mask 1 Predictions:

45.2%	bone
30.1%	stick
15.3%	toy
9.4%	shoe

Figure 26 *Example of using the [MASK] token in BERT[35]*

C. In our third method we firstly return a set of synonyms, as outlined in method A. Each synonym is swapped for w in turn and a partial sentence, three words before and three words after w, is passed to the BERT language model to score the sentence for grammatical correctness. A lower score indicates a more grammatically correct sentence[36]. The synonym that gives a more grammatically correct sentence is the one that is swapped for w.

For all three methods the number of modifications made to each sentence is limited to $max\_perturbations$. By defining $max\_perturbations$ for each sentence, we can *guide* the genetic algorithm toward a solution with the least number of modifications. In *Section 6.1.2* we will define $target$ label and $fitness\ function$. After $max\_perturbations$ have been made to the original text the mutation function will then return the mutated text string and add it to our population list.

6.1.2 Target label and fitness function

Our sentiment analysis model accepts text in the format of a string and returns a probability of that text being a positive sentiment.

We define the following thresholds:
A return of > 0.5 is a positive sentiment and so labelled 1, and sentiment ≤ 0.5 is a negative sentiment and labelled 0.

If our original text is labelled as being a positive sentiment, i.e., 1, then we define our $target$ label to be 0, and vice versa.

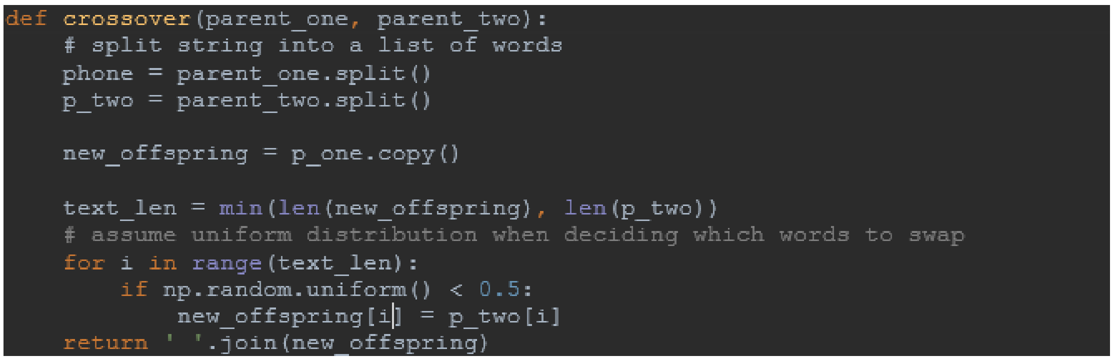
As outlined in *Section 3.1.2*, the fitness of each text is defined by the probability returned by the sentiment analysis model, and how close this probability is to the $target$ label.

6.1.3 Selection

After we have generated our population. We need to choose which two solutions will act as parents, being passed to the crossover and mutation stages. For each population member we query the sentiment model and store the returned probability. If at this point, we have a population member whose probability is equal to the $target$ label, the solution is adversarial and we have reached our goal, so we stop (see 6.1.5). Otherwise, we select the two population members that take us closest to our $target$ label and pass these two $parents$ to our $crossover$ function to generate a new $child$. The $crossover$ function is outlined next in *Section 6.1.4*.

6.1.4 Crossover

When the two $parents$ are passed into the $crossover$ function we randomly select words from the parents to generate a $child$, as shown in the code in *Figure 27*. The new $child$ is created from a combination of the two parents where each word has the same probability of being selected.



```python
def crossover(parent_one, parent_two):
    # split string into a list of words
    phone = parent_one.split()
    p_two = parent_two.split()

    new_offspring = p_one.copy()

    text_len = min(len(new_offspring), len(p_two))
    # assume uniform distribution when deciding which words to swap
    for i in range(text_len):
        if np.random.uniform() < 0.5:
            new_offspring[i] = p_two[i]
    return ' '.join(new_offspring)
```

Figure 27 *crossover function, to generate a new child*

Once we have generated a new $child$ we then repeat the steps from *Section 6.1.1*.

6.1.5 Exit condition

We have two possible exit conditions for the algorithm. The first being that we have a population member that, after querying the sentiment model, is equal to the $target$ label and so we have our optimal solution and can exit. The other condition being that we have run for a set number of generations.

The number of generations to run the algorithm was decided after experimentation, where we initially set the number of generations at 100, and 50 and finally settled upon 20. We noted that after increasing generations to over 20 there was no improvement in the adversarial examples generated. The lower number was also decided upon due to one of the aims of the project, which is to generate an adversarial example that has the lowest number of modifications.

7 Experiments and Results

Having trained the sentiment analysis model, outlined in *Section 5.3*, and coded the genetic algorithm, outlined in *Section 6*, we will now detail the adversarial attacks and subsequent results. In *Section 7.1* we will describe the sample dataset used. In *Section 7.2* we describe the test setup, results and outline the parameter settings used for the algorithm, in Section 7.3 we will present the results from the three different mutation algorithms, A, B and C used, as described in *Section 6.1.1*.

7.1 Sample Dataset

From *Figure 23*, we see that our sentiment analysis model has an accuracy of 89.93% i.e., the model will likely wrongly classify samples 10.07% of the time. For our test dataset we randomly sampled 250 examples from the dataset we used to train our sentiment analysis model (attack model). We queried our sentiment model for each sample and retained only the samples our model correctly classified. This was done to ensure that the accuracy of our sentiment model does not interfere with our results.

The same dataset was used for all three methods outlined in *Section 6.1.1*. This is to ensure that we can compare the performance for each method against a common set of data points.

7.2 Test Set-Up and Results

The configuration settings in *Table 1* are common to all the three methods used. These configuration settings were kept consistent across all the test runs. The settings used were selected through trial and error; these were found to be the optimal settings for minimising the total number of words swapped and giving the shortest run time for the algorithms.

Common Configuration	Generations	Population Size	Dataset size	Maximum perturbations
All Runs	**20**	**20**	**250**	**5**

Table 1 *Common configuration settings, to give adversarial examples with minimum words swapped and best run time for all three methods used*

Maximum perturbations: *is the maximum number of swaps of words allowed per review when generating a new population member for a new population set*.

Table 2 shows the Smallest, largest, and mean number of words in the reviews in our test dataset.

Length of reviews in our dataset (number of words)		
Smallest Review	**Largest review**	**Mean review**
37 words	945 words	236 words

Table 2 *Smallest, largest, and mean number of words in the reviews in our test dataset*

The run times are difficult to measure accurately, being subject to hardware, operating system, and random variations due to background processes. To mitigate this, each of the methods was run on the same machine and repeated 4 times for each data point in the test set. The same test set was used for each of the three methods outlined in *Section 6.1.1*. Ideally more than 4 repeats would be used but owing to long run times for each algorithm this was not possible in the time available. (a total of 2 days per run of Method A and B and up to 3 days for Method C). The results for run time of each algorithms data point (in seconds) was averaged across all these runs and are shown in *Figure 28* for all reviews regardless of whether the sentiment was successfully changed or not. *Figure 29* is the box plot with only the times for only successful adversarial examples for the same 4 runs of the methods. *Figure 30* shows run times for successful reviews less than or equal to the mean review size of 236 words.

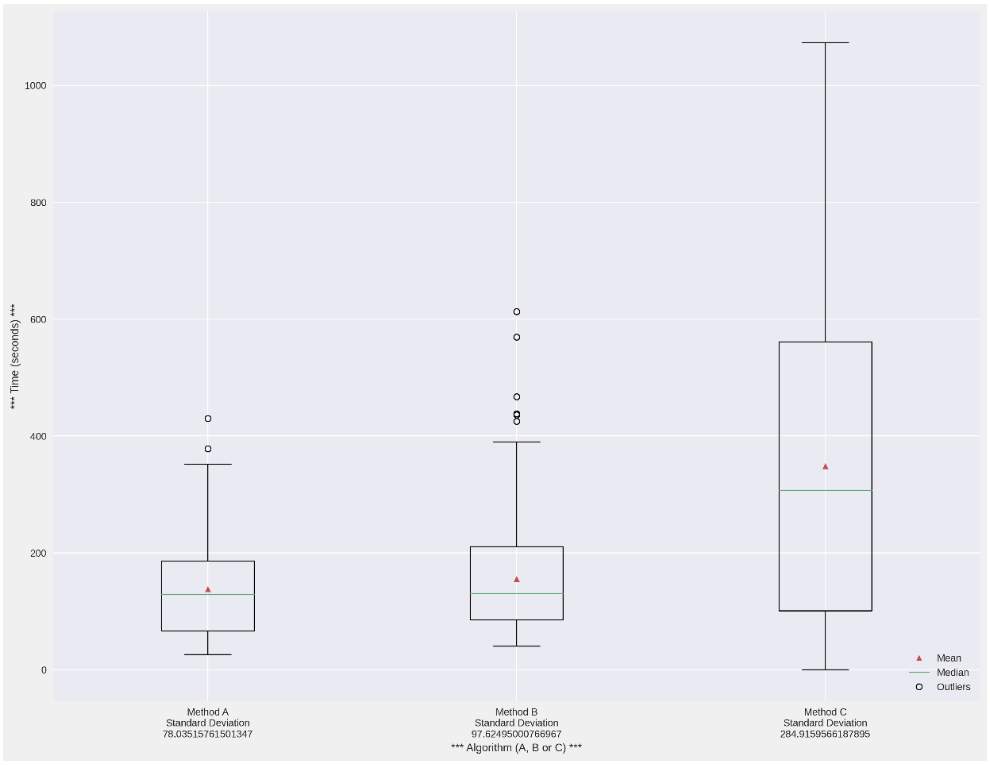


Figure 28 *Boxplot for time taken to generate adversarial example for each method, regardless of success, each box plot represents the average time (in seconds) of the 4 runs of each method* ($\frac{t_1+t_2+t_3+t_4}{4}$)

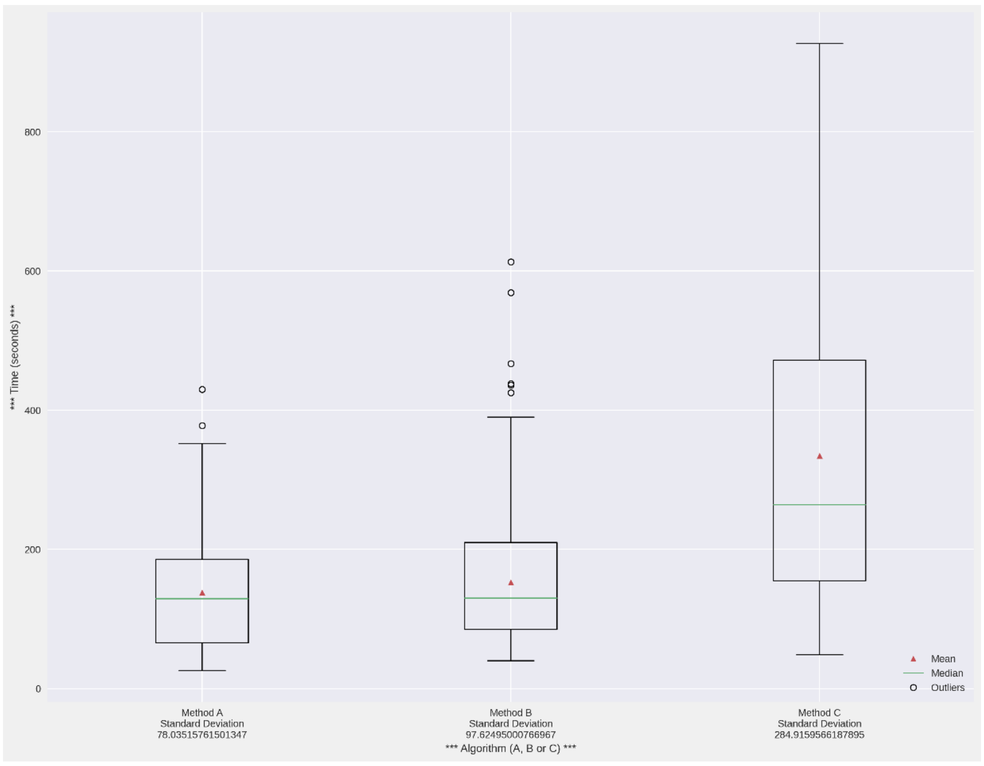


Figure 29 *Boxplot for time taken to generate adversarial example for each method for successful results only, each box plot represents the average time (in seconds) of the 4 runs of method* $(\frac{t_1+t_2+t_3+t_4}{4})$

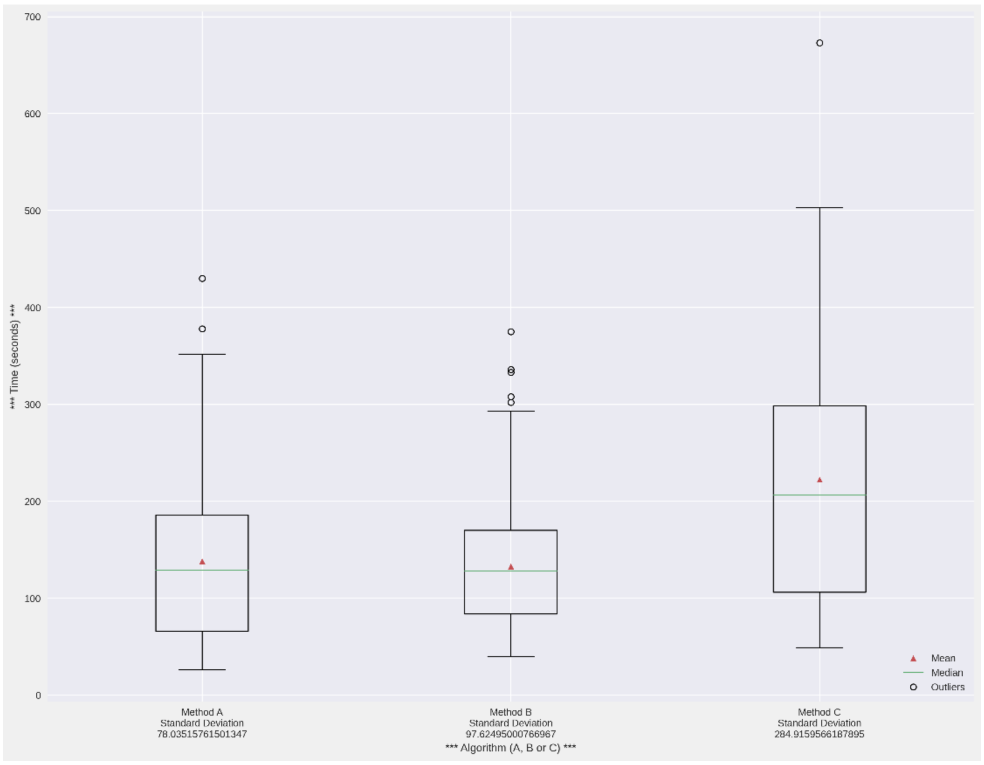


Figure 30 *Boxplot for time taken to generate adversarial example for each method for successful results only, each box plot represents the average time (in seconds) of the 4 runs of method* $(\frac{t_1+t_2+t_3+t_4}{4})$*. Restricted to reviews less than or equal to mean review size of 236 words*

7.2.1 Discussion

We can see from both *Figure 29* and *30* that the time taken to generate a successful adversarial example is more widely spread for Method C i.e., a wider range of values. Method A has range from 26 to 430 seconds, Method B has a range from 40 to 613 seconds and Method C has a range of 49 to 927 seconds. The adversarial examples generated corresponding to the lowest time values (26, 40 and 49 seconds) are different for each of the methods and consist of reviews with less than 150 words. The most time taken, as expected, is for longer reviews, in all cases reviews with over 500 words.

We can also note that the mean and median values for methods A and B are significantly lower than for method C. There is also significantly less variance in time taken for Methods A and B, 78 and 97 seconds respectively compared to 284 seconds for Method C. This was expected, because Method C checks not only for synonyms but also for grammatical correctness of the partial sentence when swapping for synonyms. The large range is also partly due to the significant difference in length of the reviews. (ranging from 37 to 945 words). In *Figure 30,* we have plotted only reviews less than or equal to the mean review size (236 word), and as can be noted the range for Methods B and C is significantly reduced, i.e., the time taken (in seconds) is significantly less if we reduce the length of each review in our data set.

Further results and graphs for each method are shown in *Sections 7.3*.

7.3 Results for each algorithm

In *Sections 7.3.1, 7.3.2* and *7.3.3* we will present graphically the results from each of the methods we outlined in *Section 6.1.1*. In *Section 7.3.4* we discuss these results in more detail. These sections meet objective 4 (*Section 1.2*, analysis of our solution).

7.3.1 Method A

Figure 31 shows the number of words swapped per successful adversarial example for each review using Method A. This is for the 235 successful adversarial examples generated for the dataset of 250 reviews; no adversarial examples were found for the remaining 15 reviews, so they are excluded from this plot.

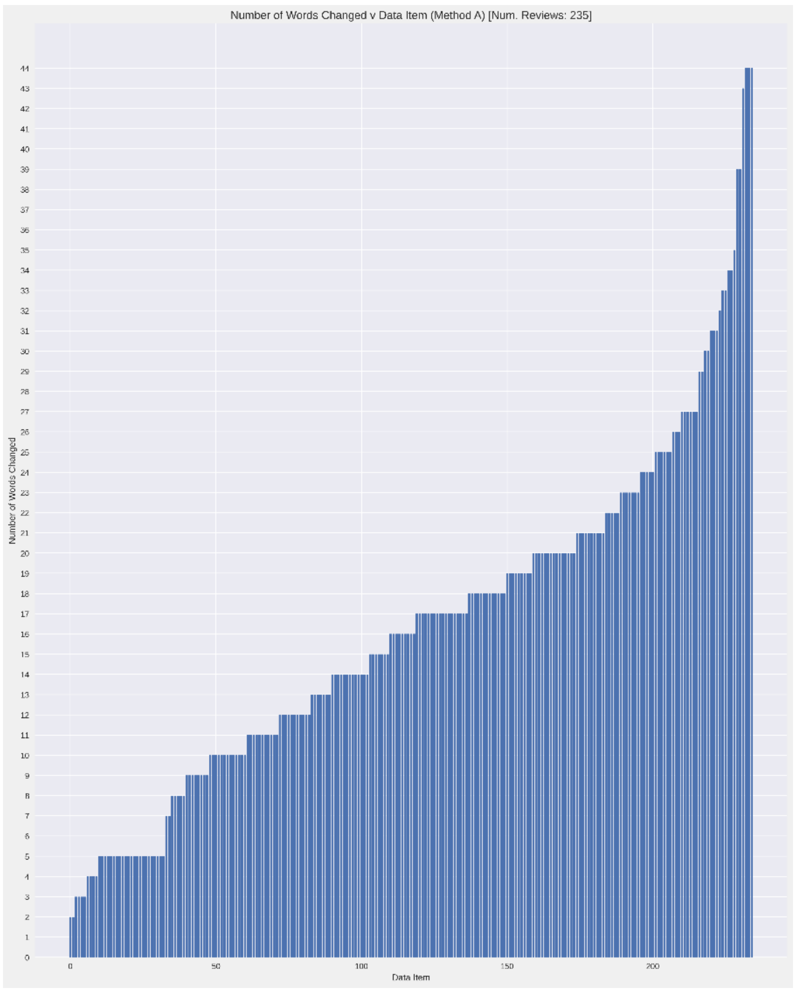


Figure 31 *Number of words swapped per review for Method A for successfully generated adversarial examples*

Figure 32 shows the number of words swapped against the length of the review text for successful examples for Method A.

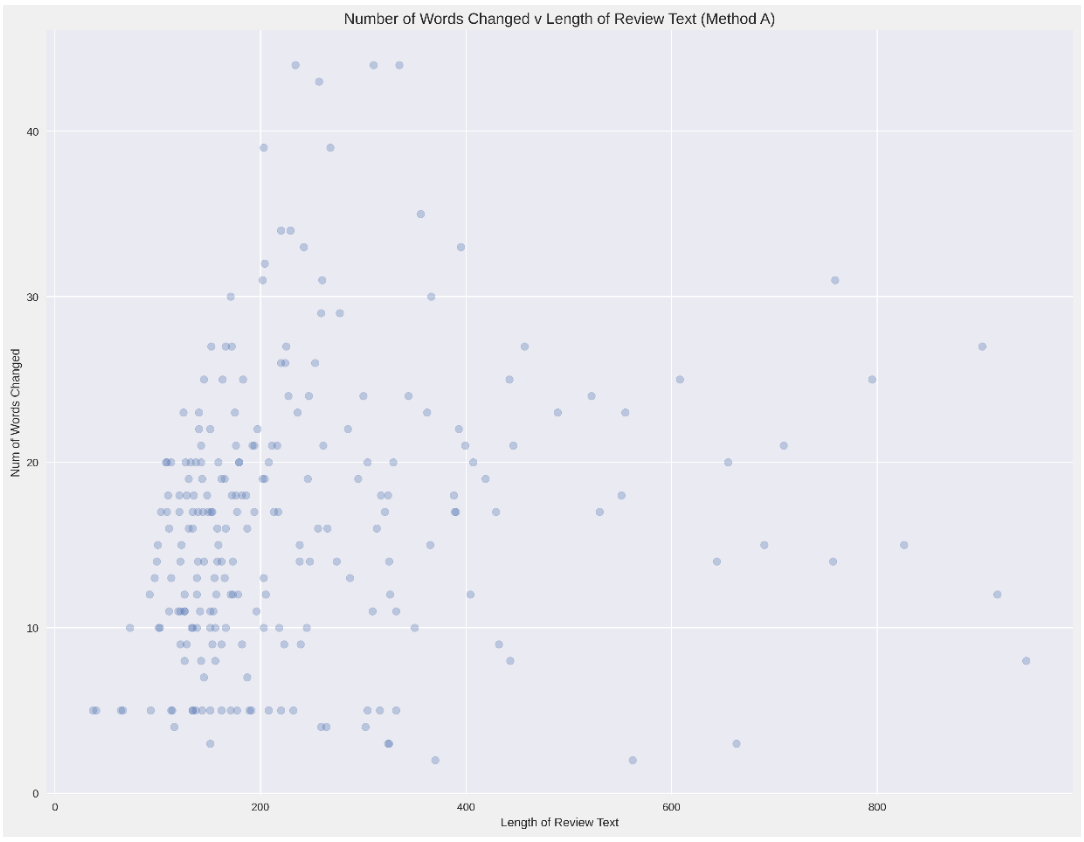


Figure 32 *Number of words swapped against the length of the review text for successful examples (Method A)*

Figure 33 shows the number of words swapped against generation for Method A. This plot includes only those reviews that had their sentiment successfully changed during the attack.

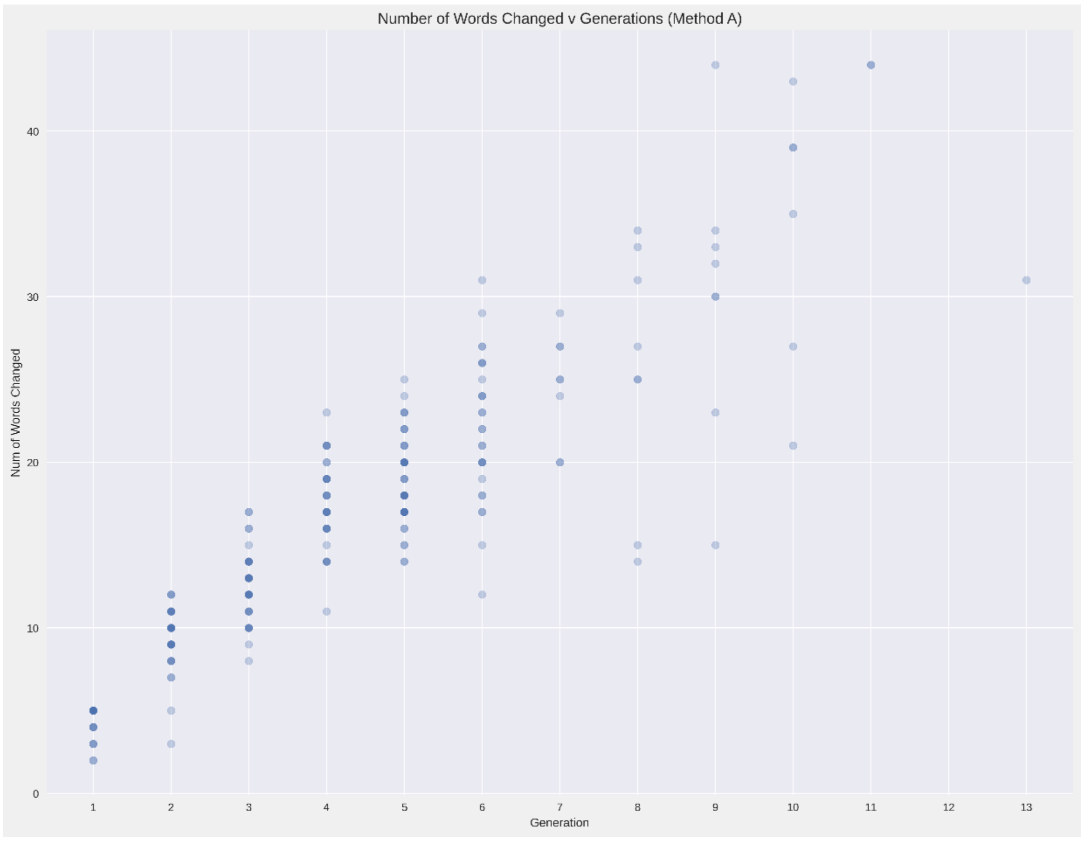


Figure 33 *Number of words swapped against generation, for successful adversarial examples (Method A)*

Figure 31 is to be expected: the longer the algorithm runs for, the greater the number of words that are swapped. *Figure 32* shows that there is a disproportionate number of shorter reviews in the dataset i.e., less than 250 words. Future tests could be run on a dataset that had an equal split of reviews with less than 250 words and reviews with greater than 300 words, to get a better representation of the performance of this Method, in terms of words swapped and success or failure of the algorithm to generate adversarial examples. *Figure 32* also seems to indicate that there is not a strong relationship between length of review and number of words swapped, which seem counterintuitive, however this would need to be further tested by running the algorithm against a dataset of reviews with larger word counts.

7.3.2 Method B

Figure 34 shows the number of words swapped per successful adversarial example for each review using Method B. This is for the 243 successful adversarial examples generated for the dataset of 250 reviews; no adversarial examples were found for the remaining 7 reviews, so they are excluded from this plot.

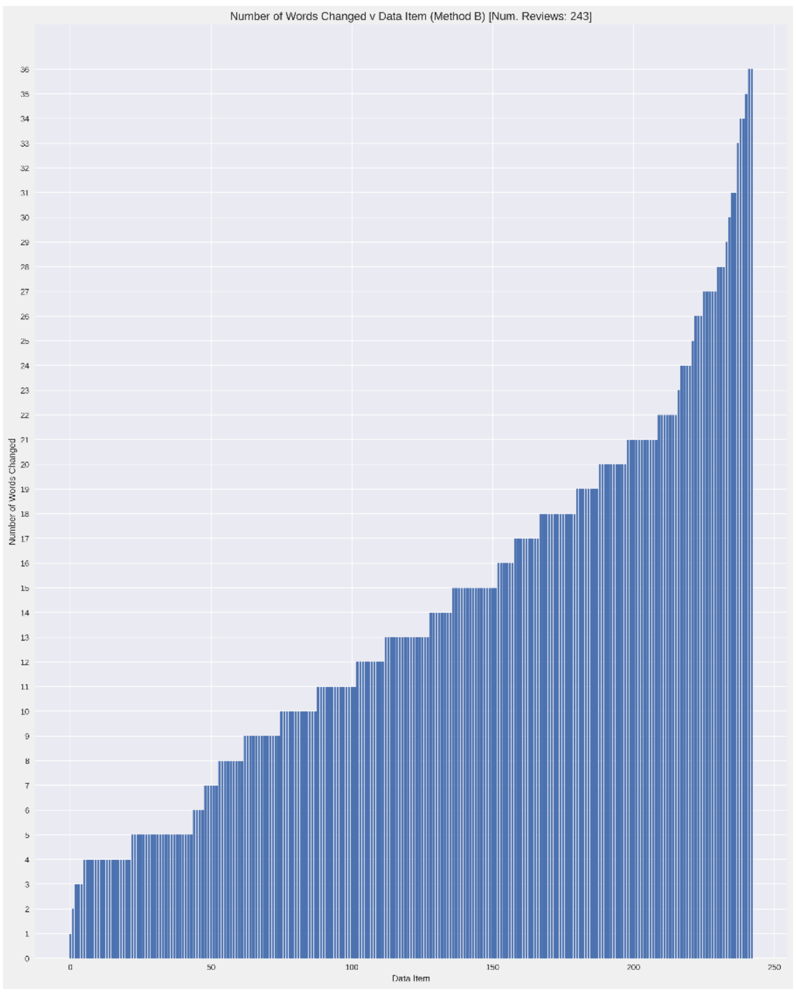


Figure 34 *Number of words swapped per review for Method B for successfully generated adversarial examples*

We can see that this graph is very similar to *Figure 31*, there is little significant difference between the number of words swapped for each data item between the Method A and B.

Figure 35 shows the number of words swapped against the length of the review text for successful examples for Method B.

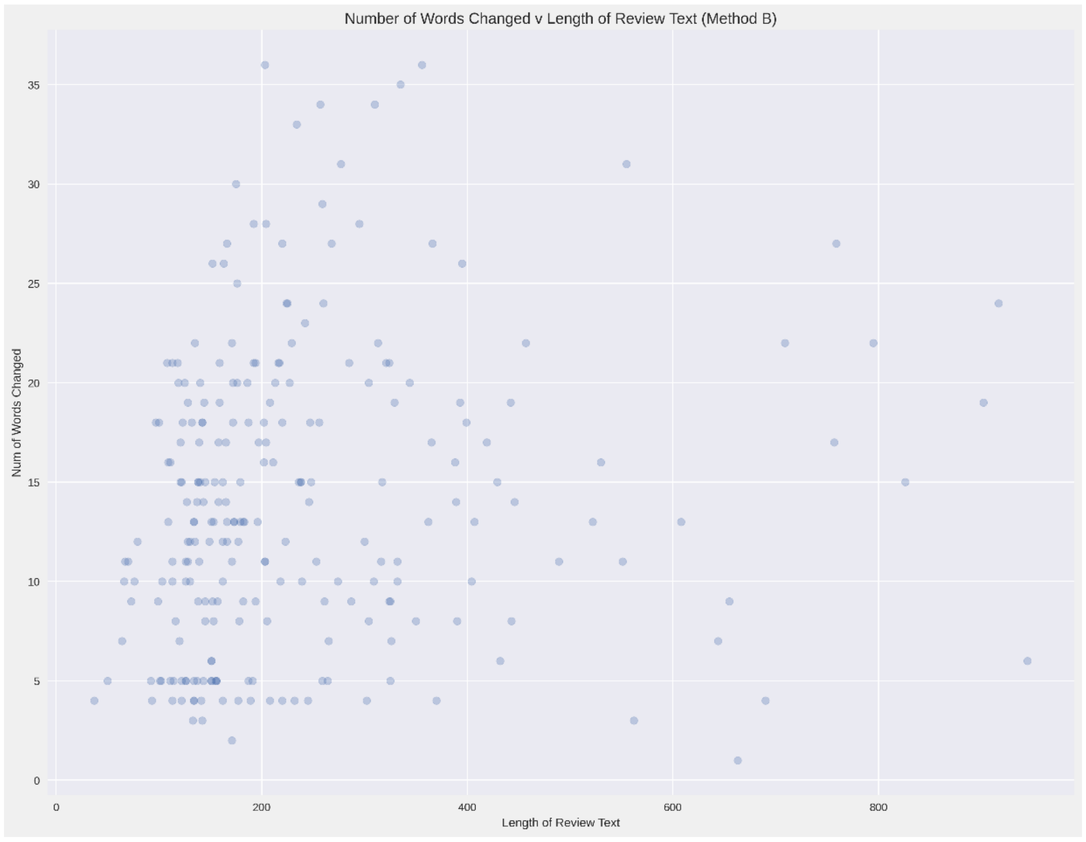


Figure 35 *Number of words swapped against the length of the review text for successful examples (Method B)*

Figure 36 shows the number of words swapped against generation for Method B. This plot includes only those reviews that had their sentiment successfully changed during the attack.

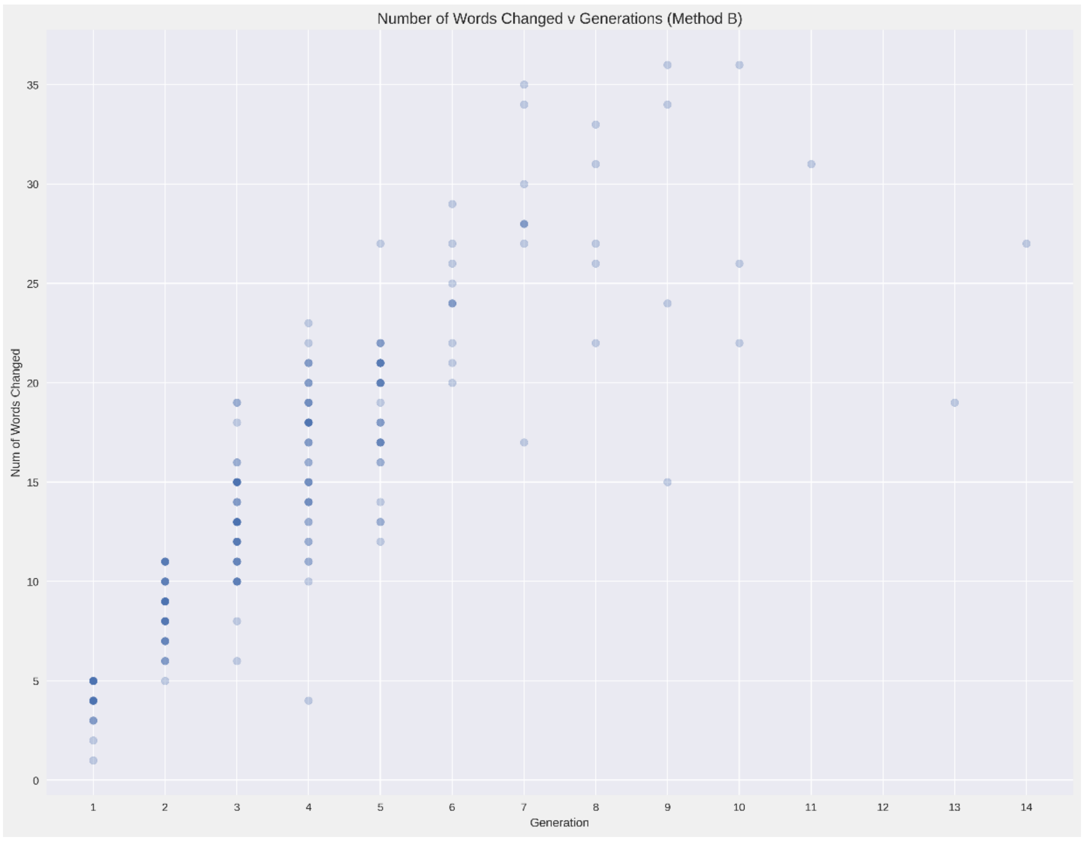


Figure 36 *Number of words swapped against generation, for successful adversarial examples (Method B)*

7.3.3 Method C

Figure 37 shows the number of words swapped per successful adversarial example for each review using Method C. This is for the 131 successful adversarial examples generated for the dataset of 250 reviews; no adversarial examples were found for the remaining 119 reviews, so they are excluded from this plot.

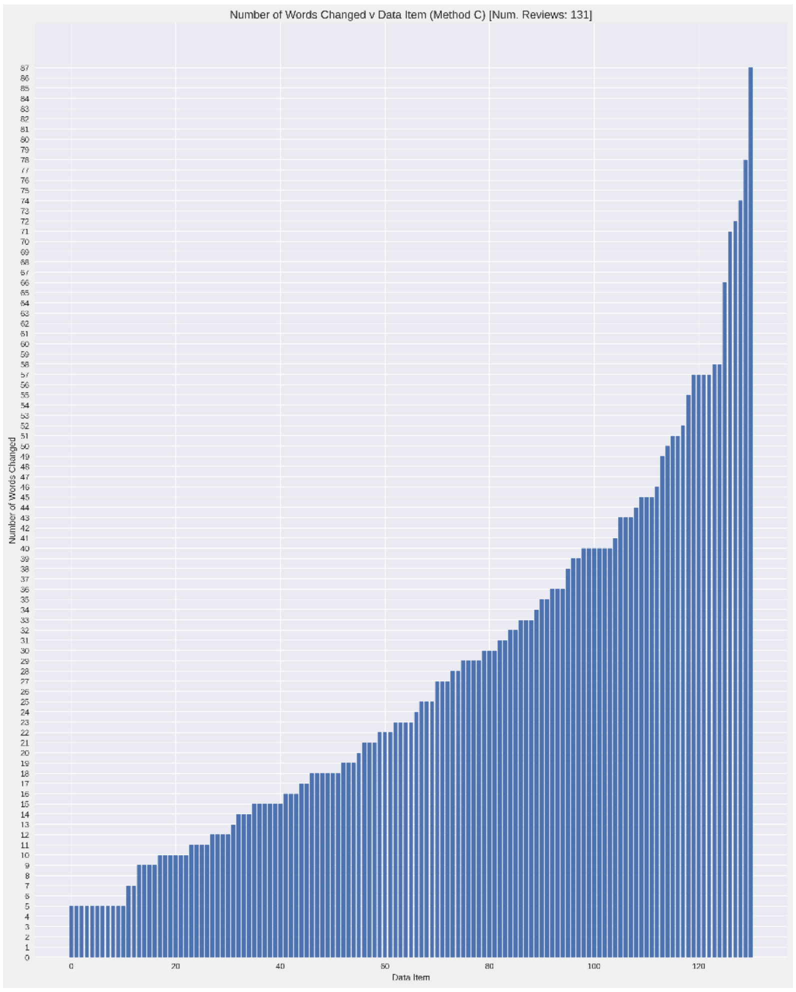


Figure 37 *Number of words swapped per review for Method B for successfully generated adversarial examples*

Figure 38 shows the number of words swapped against the length of the review text, for Method C.

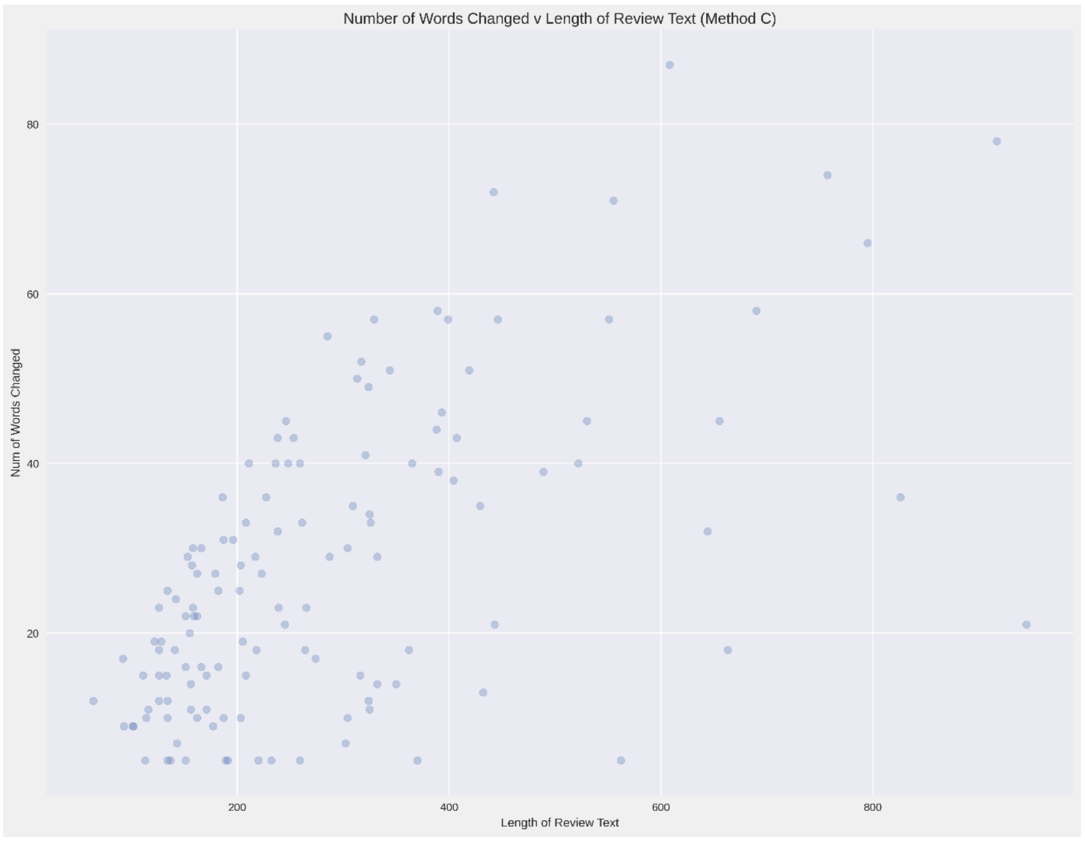


Figure 38 *Number of words swapped against the length of the review text (Method C)*

Figure 39 shows the number of words swapped against generation for Method C. This plot includes only those reviews that had their sentiment successfully changed during the attack.

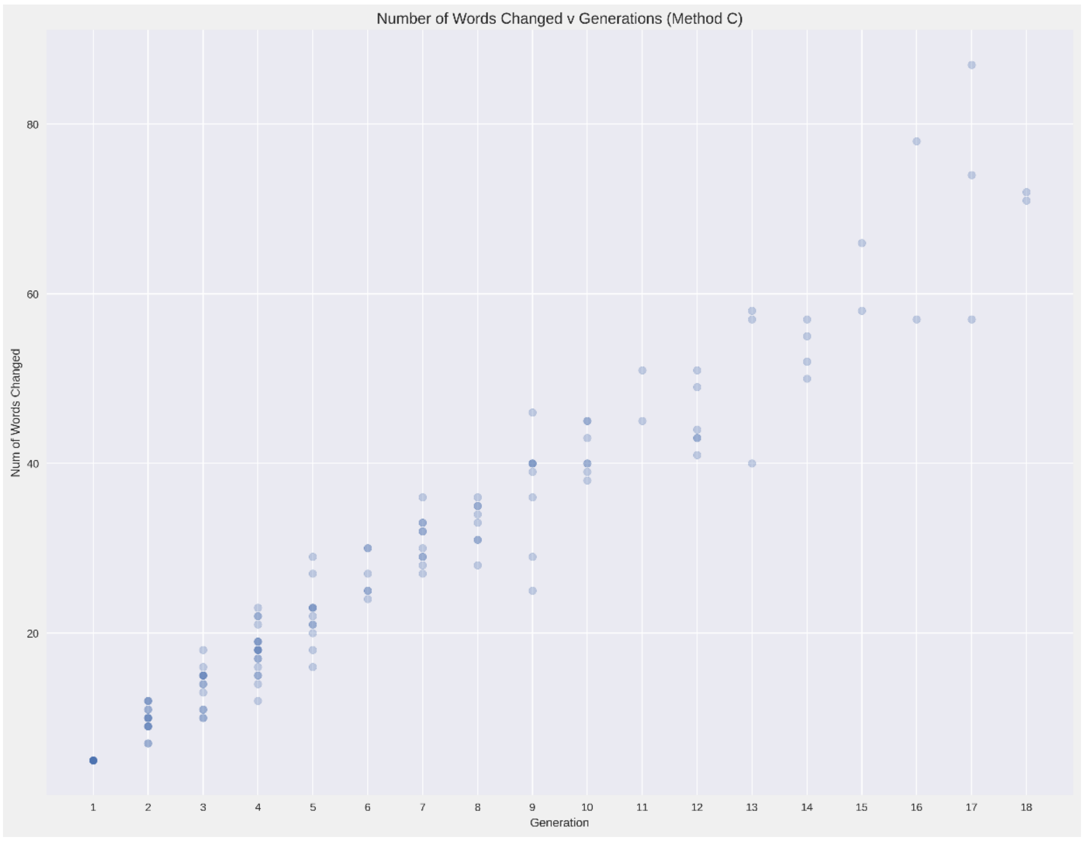


Figure 39 *Number of words swapped against generation, for successful adversarial examples (Method C)*

7.3.4 Discussion

We can see for *Figures 32*, *35* and *38* that not only do shorter texts, proportionally to text length, tend to have more words swapped than longer reviews they also make up most reviews that have had their sentiment changed successfully. However, the reviews with more text require fewer word swaps, proportionally to length of text, to change their sentiment. There does not however appear to be a relationship between the number of words swapped and length of text. This could be further tested by splitting the reviews into two equal groups, e.g., those with reviews of text length less than 500 words and those with text consisting of reviews much longer than 500 words.

For *Figures 33*, *36* and *39* we can see that all successful adversarial examples were generated before reaching the 20th and final generation. For Method A and B, we see that most of the successful adversarial examples were generated before we reached generation 8. Method C however has a more even spread of successful adversarial examples across all generations up to generation 15. Based on these results Method B most closely meets the requirements we set out for this project, i.e., it generates the most grammatically correct adversarial example with the least number of words swapped.

To further test all the methods, we would in future run each of the algorithms again against a dataset with shorter reviews, i.e., review length of less than 250 words, and against a separate dataset with reviews with significantly more than 250 words, this would give a better indication as to the number of words swapped and run time for each of the above algorithms. The dataset used, had mostly shorter reviews (167 reviews with less than or equal to 236 words out of 250 total reviews) and significantly less reviews with word counts over 300 words (60 reviews with more than or equal to 300 words out of 250 total reviews).

7.4 Adversarial Examples Generated

In this section we present some sample adversarial examples generated by each of the methods from *Section 6.1.1*, and in *Section 7.4.1* we discuss these results.

In *Figure 40* and *41* we have selected at random two reviews. For each review we show the original text and then the examples generated by each of the 3 methods. We highlight the words swapped by each method in red and state the number of words swapped for each method.

Original Review: Probability = **Positive** with confidence of **99.86%**

As the superb ‘Prime Suspect’ series reaches part four there is no loss of momentum at all, this in itself a considerable achievement. 'Prime Suspect IV: The Lost Child' has the solid supporting cast that we take for granted in these British dramas but of course the beautiful Helen Mirren easily dominates; our eyes never leave her while she's on-screen.

The search for the lost child of the title leads Superintendent Jane Tennison's CID team to a prime suspect who turns out to be a convicted pedophile now living with a single mother and her two young daughters. The insight we are given into the workings of his mind is one of the emotional highlights of this mini-series but it may be too strong for many stomachs.

The action sequences are brilliantly handled with the hand-held camera thrusting us right into the middle of the excitement and there's gripping tension during the climactic siege.

Altogether this is another magnificent police procedural drama.

Method A (Adversarial Example): Probability = **Negative** with confidence **62.33% (25 words swapped)**

As the superb ‘**Prima** Suspect’ **number** reaches **parties oven** there is no **loosing** of momentum at all, this in itself a considerable achievement. **'First** Suspect **d:** The Lost Child' has the solid supporting cast that we take for granted in these British dramas but of **courses** the **handsome Jolene** Mirren easily dominates; our **eyeballs** never leave her while she's on-screen.

The search for the lost child of the title leads **Headmaster** Jane Tennison's CID **machine** to a prime suspect who turns out to be a convicted **pedophilia** now living with a single mother and her **trois** young **female**. The insight we are given into the workings of his mind is **somebody** of the emotional **emphasize** of this mini-series but it may be too strong for many stomachs.

The action **sequencing** are **magnificently processed** with the hand-held **cameras** thrusting us right into the middle of the **agitation** and there's gripping **voltage** during the climactic siege.

Altogether this is another **marvellous** police procedural drama.

Method B (Adversarial Example): Probability = **Negative** with confidence **64.82% (26 words swapped)**

As the superb ‘Prime Suspect' **night** reaches part **ii** there is no loss of momentum at all, this in itself a considerable achievement. ‘**Of S** Suspect IV: The Lost Child' has the **biggest acting** cast that we take for **free** in these **Holy isles** but of course the beautiful Helen Mirren **now** dominates; our eyes **not** leave her while she's **running**.

The **charity** for the lost child of the title leads Superintendent Jane **To** CID team to a prime suspect who turns out to be a convicted **adult** now living with a single mother and her two young daughters. The insight we are given into the workings of his **brain** is **capable** of the emotional **consequences** of this mini-series but it **would** be too **hot** for many stomachs.

The action sequences are **generally associated** with the **movie** camera thrusting us right into the middle of the excitement and there's **no silence** during the climactic **stage**.

Altogether this is another magnificent police procedural drama.

Method C (Adversarial Example): Probability = **Negative** with confidence 5**4.13% (52 words swapped)**

As the superb ‘**Prima** Suspect' **instalment obtains parties** four there is no **loosing** of **dynamic** at all, this in itself a **sizable successes**. '**First** Suspect IV: The **Outof** Child' has the solid supporting **thrown** that we **picked** for granted in these British **opera** but of **paths** the beautiful **Jolene** Mirren **easy** dominates; our **eyelids not letting** her while she's **internet**.

The **searches** for the **outof infantile** of the **designation directors Governing** Jane Tennison's CID **machine** to a **first** suspect who **rotates** out to be a convicted **perverts** now living with a **sole mom** and her **ii youngsters girls**. The insight we are **handed** into the **exmrs** of his mind is **anybody** of the **psychological emphasize** of this mini-series but it **perhaps** be too **influential** for **numerous** stomachs.

The action **timeline** are **magnificently** handled with the **laptop camcorder** thrusting **ourselves ok** into the middle of the excitement and there's **fascinating pressure** during the climactic **seating**.

Altogether this is another magnificent police procedural drama.

Figure 40 *Comparison of original text to Method A, B and C generated adversarial examples*

Original Review: Probability = **Positive** with confidence of **99.13%**
There's some very clever humour in this film, which is both a parody of and a tribute to actors. However, after a while it just seems an exercise in style (notwithstanding great gags such as Balasko continuing the part of Dussolier, and very good acting by all involved) and I was wondering why Blier made this film. All is revealed in the ending, when Blier, directing Claude Brasseur, gets a phone call from his dad (Bernard Blier) - from heaven, and gets the chance to say how much he misses him. An effective emotional capper and obviously heartfelt. But there isn't really sufficient dramatic tension or emotional involvement to keep the rest of the film interesting throughout it's entire running time. Some really nice scenes and sequences, however, and anyone who likes these 'mosntres sacrés' of the French cinema should get a fair amount of enjoyment out of this film.

Method A (Adversarial Example): Probability = **Negative** with confidence **60.66% (17 words swapped)**
There's some very clever **mood** in this film, which is both a **travesty** of and a tribute to **agents**. However, after a while it just seems an exercise in style (notwithstanding **grand jaws** such as Balasko continuing the **parties** of Dussolier, and very **alright behaving** by all involved) and I was wondering why Blier made this film. All is revealed in the **ceases**, when Blier directing Claude Brasseur, gets a phone call from his dad (**Barnard** Blier) - from heaven, and gets the chance to say how much he misses him. An effective **sentimental** capper and obviously heartfelt. But there isn't really sufficient dramatic **voltage** or emotional involvement to keep the rest of the film interesting **during** it's entire **run** time. Some really **handsome image** and sequences, however, and anyone who likes these 'mosntres sacrés' of the French cinema should get a fair **amounts** of enjoyment out of this film.

Method B (Adversarial Example): Probability = **Negative** with confidence **61.17% (12 words swapped)**
There's some very **bad** humour in this **book**, which is both a **member** of and a tribute to actors. However, after a while it just seems an exercise in style (notwithstanding **some** gags such as Balasko continuing the part of Dussolier and very good acting by all involved) and I was wondering why Blier made this film. All is revealed in the ending, when Blier, directing **La** Brasseur, gets a phone call from his dad (Bernard Blier) - from heaven and gets the chance to say how much he misses him. An effective **show** capper and obviously **not**. But there isn't really **no** dramatic tension or emotional involvement to keep the rest of the film interesting throughout it's entire running time. Some **rather** nice scenes and sequences, **anything**, and anyone who likes these 'mosntres sacrés' of the French cinema should **make** a **reasonable** amount of enjoyment out of this film.

Method C (Adversarial Example): Probability = **Negative** with confidence **53.01% (37 words swapped)**
There's some very **smart mood** in this film, which is both a parody of and a **compliments** to **actresses**. However, after a while it just **occurs** an **exercising** in style (**although** great **jaws** such as Balasko continuing the part of Dussolier, and very **ok behaving** by all involved) and I was **asks** why Blier made this **films**. All is **disclosed** in the ending, when Blier, directing Claude Brasseur, gets a **telephones** call from his **pappy** (**Barnard** Blier) - from heaven, and gets the **chances** to **says** how much he **lack** him. An **efficient** emotional **evilness** and **clearly candid**. But there isn't **frankly suffice whopping pressure** or **thrills involved** to keep the **remainder** of the **movie** interesting **in** it's entire running **timeframe**. Some really **pleasant imagery** and sequences, however, and anyone who likes these 'mosntres sacrés' of the **Frenchmen** cinema should get a fair amount of **realisation** out of this film.

Figure 41 *Comparison of original text against adversarial examples generated for Methods A, B and C*

Qualitatively, Method B seems to produce the best English in both examples. From *Figure 41*, comparing one sentence for all three methods, we see that Method B, gives the sentence with least number of words swapped and most grammatically correct (5 words swapped for Method B, 6 for Method A and 14 for Method C):

Original Sentence:
"But there isn't really sufficient dramatic tension or emotional involvement to keep the rest of the film interesting throughout it's entire running time. Some really nice scenes and sequences, however, and anyone who likes these 'mosntres sacrés' of the French cinema should get a fair amount of enjoyment out of this film."

Method A:
"But there isn't really sufficient dramatic ***voltage*** *or emotional involvement to keep the rest of the film interesting* ***during*** *it's entire* ***run*** *time. Some really* ***handsome image*** *and sequences, however, and anyone who likes these 'mosntres sacrés' of the French cinema should get a fair* ***amounts*** *of enjoyment out of this film."*

Method B:
"But there isn't really ***no*** *dramatic tension or emotional involvement to keep the rest of the film interesting throughout it's entire running time. Some* ***rather*** *nice scenes and sequences,* ***anything****, and anyone who likes these 'mosntres sacrés' of the French cinema should* ***make*** *a* ***reasonable*** *amount of enjoyment out of this film."*

Method C:
"But there isn't ***frankly suffice whopping pressure*** *or* ***thrills involved*** *to keep the* ***remainder*** *of the* ***movie*** *interesting* ***in*** *it's entire running* ***timeframe****. Some really* ***pleasant imagery*** *and sequences, however, and anyone who likes these 'mosntres sacrés' of the* ***Frenchmen*** *cinema should get a fair amount of* ***realisation*** *out of this film."*

In **Table 3**, we summarise the success rate for each of the three methods along with the mean percentage of swapped words for each method.

Method A		Method B		Method C	
Success (%)	Mean Words Swapped (%)	Success (%)	Mean Words Swapped (%)	Success (%)	Mean Words Swapped (%)
94.00%	6.55%	97.20%	5.42%	52.40%	9.44%

Table 3 *comparison of the three methods for percentage success and mean number of words swapped for all successful attacks generated by each algorithm*

7.4.1 Discussion

As observed from *Figure 40*, *41* and *Table 3*, Method B is the most successful at meeting our original goal of least number of words swapped and most grammatically correct adversarial example, we had a success rate of 97.20% and the mean number of words swapped is 5.42%. The least successful is Method C with a success rate of 52.40% and mean number of words swapped of 9.44%.

As we can see when comparing a sample sentence between the three methods, above, Method B gives the most grammatically correct sentence with the fewest words swapped. The use of context when deciding which word to swap improves the grammar of the adversarial example produced. We used the *bert-large-uncased* pretrained model provided by Hugging Face[37]. This is a self-supervised transformer model trained on a large corpus of English data. For our project we used this model on partial sentences, i.e., we selected 3 words before and 3 words after our target word to provide context with the aim of selecting the most appropriate word to swap. This could be improved, and possibly provide better results if we included more words before and after the target word to improve the context for the entire sentence rather than partial sentences.

8 Conclusion

8.1 Summary

During this project, a sentiment analysis model was trained on the IMDB movie review dataset[22], the training and evaluation of the model is detailed in *Section 5.3*. This model was then used as our victim model.

Once the sentiment analysis model was trained and saved, we created three genetic algorithms, as detailed in *Section 6.1.1*. The algorithms were labelled Method A, B and C. The algorithms all function correctly and can generate adversarial examples that successfully flipped the original sentiment.

The goals of the project were to create a genetic algorithm that could not only flip the sentiment of the original text, but to do so with the least number of words swapped and for the adversarial example to be grammatically correct. The project was successful in both regards. Method B, which involved swapping of words based on context, using surrounding words, was the most successful of the three algorithms.

The adversarial examples produced by this project could be further analysed through use of human evaluation and feedback regarding the adversarial examples and the perception of sentiment in the text output by each of the algorithms.

8.2 Evaluation

The aims of this project, as set out in *Section 1.2* will now be discussed in turn. We will then discuss additional findings and work in *Section 8.2.3* and overall project management of work in *Section 8.2.4*, finally moving onto a critical comparison of this project to similar work in this area in *Section 8.2.5*.

8.2.1 Create and test a supervised NLP sentiment analysis model

For us to generate adversarial examples against a model, we first needed to train the victim model. We first selected a dataset to use for training of the model. We selected the IMDB Movie Review dataset[22]. This dataset consists of 50,000 movie reviews which are equally split between positive and negative reviews. The model training is detailed in *Section 5.3*. The final model trained had an accuracy of 0.899% and an area under the curve(AUC), as given by the ROC curve, of 0.899. This met objectives 1. and 2. (from *Section 1.2*, creating the model and testing it).

8.2.2 Generate adversarial examples with the smallest words swapped and that are grammatically correct

The algorithms coded are detailed in *Section 6.1.1*, following a genetic algorithm framework. Design and implementation of these met objective 3. (*Section 1.2*). We initially coded only one algorithm (Method A), which involved simple swapping words with their synonyms. This was done by firstly creating a distance matrix, as outlined in *Section 5.2*, and a synonym was selected by looking at the closest words, spatially, to the selected word. This worked well in terms of flipping the sentiment, however the resultant adversarial example was not well formed grammatically.

To improve upon this initial algorithm, it was decided that we would use context as a deciding criterion when selecting a replacement word. To this end we imported a pretrained language model, BERT[34], as detailed in *Section 6.1.1* and created a second algorithm (Method B). This improved not only the grammar of the adversarial example but also reduced the number of words we had to swap. See *Table 3* for a comparison of all three algorithms.

To test if we could further improve upon the results from the second algorithm (Method B), we decided to add an additional test after selecting a word to swap. So, took the initial algorithm (Method A) and add a test for sentence structure, again using the pretrained BERT language model[34]. We created our third algorithm (Method C), which was a combination of Method A with the addition of a test of sentence structure. We swapped words with their synonyms and then tested a partial sentence for correctness, the synonym that gave the most accurate sentence structure was swapped with the original word. This was the least successful of the three algorithms, see *Table 3* for comparison of performance.

The core aims of this project were to generate adversarial examples that were grammatically correct with the least number of changes between the original and the adversarial example, we were successful in achieving these aims.

8.2.3 Additional findings and work

The aims of the project listed in *Section 1.2* were the main goals at the start of this project. During this project, we identified improvements that could be made. The main one was the decision to use a pretrained language model, BERT[34], this was not in the original outline for the project, however it has proven to be the solution that provided the best outcome in terms of generating adversarial examples with the least number of changes that are also grammatically correct.

We did not meet objective 5 (*Section 1.2*, potentially ask a human subject to classify the adversarial examples produced by the system and compare against the NLP model classification). This is work that could be conducted to extend and possibly improve upon the adversarial examples generated in this project. This would aid in generating, potentially, more grammatically correct examples. This could be done through an iterative process of generating examples, then submit them to a human subject for review and based on the feedback adjusting the algorithms and repeating these steps to further refine each of the algorithms. This would also highlight any algorithms that do not improve and hence we could focus on the best performing algorithm to refine it further.

One of the bottlenecks in this project was the time taken to generate adversarial examples (2 – 3 days for a sample size of 250 data points). This could be an issue in a production setting. One solution for this issue could be to limit the size of the reviews to only the first N number of words. More research would need to be done to find the optimal value for N, through experimentation and comparison of success rates with gradually reducing the number of words for each review. Further extensions to this project are discussed in *Section 8.3*.

8.2.4 Project management

To organise and manage this project we used the CRISP (Cross-Industry Standard Process) methodology when creating the sentiment analysis model. This ensures that we follow steps that will allow our project to be replicated in the future. In accordance with this methodology, we first selected a dataset and designed the process of cleaning and preparing the data, in this regard we were fortunate that the IMDB dataset used in training our model was well prepared and split into two distinct parts, i.e., positive, and negative reviews. This required minimal cleaning; however, we coded all the necessary steps, with the reasoning that if we switched datasets, we had all the functionality in place. All steps are fully documented in the code, and the final model structure is documented and again commented extensively in the code.

GitHub was used for version control, and provided a backup of the project, any files too large to upload, such as the trained model, can all be recreated from the code on GitHub[42]. Both the code and this document was added to version control and pushed to GitHub after each change, to ensure we always had the most recent version available. This enables us to pull the full project code and this document to any machine, in case of damage to the main computer used for the project.

8.2.5 Comparison to similar work in this area

Most of the research for generating adversarial examples against a model were initially focused on image classifiers due to the continuous nature of image data, which enables generation of adversarial examples with relative ease[11]. Recently due to big data and access to much larger data sets, there has been more focus on generating adversarial examples against text data [3,5,7,12,16]. Though most of the methods rely upon access to the underlying model, i.e., white-box attacks,[38,39], in this project we assume no access to the model training data, hence the use of a black-box attack. This assumption is the reason for the use of a genetic algorithm, because without access to the model data, we needed an efficient algorithm to find an optimal solution in a large search space. This project complements other research conducted using black-box attacks[7,18]. However, most of the current research in this area does not focus on generating grammatically correct adversarial text, which was one of the aims of this project.

8.3 Future Work

As discussed in *Section 7.4.1*, the adversarial examples could be improved, in terms of grammar, through increasing the number of words before and after the target word, to improve context for the target word. It is possible that future work could take the finding of this project and combine with white-box attacks such as[38,39] and build more robust NLP models from the outset. This could be done by building the model and generating adversarial examples against it using the approach outlined in this project in combination with the white-box attacked outlined in [38,39] and then feeding the adversarial examples back into the training set for the NLP model, thereby hardening the model against these types of attacks in the real world.

In *Section 1.2*, the feeding back of adversarial examples into the original model was mentioned as was checking of the examples by a human for the correctness of the grammar and their perception of the adversarial example however due to lack of time this was not possible.

Section 7.2 showed that there was a significant improvement in run time for shorter reviews, in future work we could restrict all reviews to a predefined word length, and test whether there is a significant impact on success rate. This could be one possible way of shortening the algorithm run times. We could gradually reduce text length (number of words) to find the optimal balance between text size and run time and success rate.

Also mentioned in *Section 1.2* was an extension to this research of combining it with a time series model and development of a predictive model for stock-market buy/sell decisions or predicting future share value based on time series data and sentiment analysis of pre-selected groupings of organisations or even currencies. This could be done by graphing sentiment for a stock, over a time, based on historic sentiment data and overlaying this onto a stock price movement graph to evaluate any correlation between the two models. Of particular interest would be the cryptocurrency markets, which are more susceptible to the prevailing sentiment expressed on social media[40,41].

References

[1] M. Ángeles López-Cabarcos, Ada M. Pérez-Pico, Paula Vázquez-Rodríguez & M. Luisa López-Pérez (2020) Investor sentiment in the theoretical field of behavioural finance, Economic Research-Ekonomska Istraživanja, 33:1, 2101-2119.

[2] Lucey, B. M., & Dowling, M. (2005). The role of feelings in investor decision-making. Journal of Economic Surveys, *19*(2), 211–237.

[3] Go, A., Bhayani, R. and Huang, L., 2009. Twitter sentiment classification using distant supervision. *CS224N Project Report, Stanford, 1(2009), p.12*.

[4] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In Empirical Methods in Natural Language Processing (EMNLP), pages 1532–1543.

[5] Mrkšić, Nikola & Séaghdha, Diarmuid & Thomson, Blaise & Gašić, Milica & Rojas-Barahona, Lina & Su, Pei-Hao & Vandyke, David & Wen, Tsung Hsien & Young, Steve. (2016). Counter-fitting Word Vectors to Linguistic Constraints. 142-148. 10.18653/v1/N16-1018.

[6] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. Intriguing properties of neural networks. arXiv preprint arXiv:1312.6199, 2013.

[7] Ji Gao, Jack Lanchantin, Mary Lou Soffa, and Yanjun Qi. 2018. Black-box generation of adversarial text sequences to evade deep learning classifiers. CoRR, abs/1801.04354.

[8] V. Sze, Y. Chen, T. Yang and J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," in Proceedings of the IEEE, vol. 105, no. 12, pp. 2295-2329, Dec. 2017, doi: 10.1109/JPROC.2017.2761740.

[9] Li, Z., Ding, Q., Zhang, W. (2011). A Comparative Study of Different Distances for Similarity Estimation. In Intelligent computing and Information Science; Chen, R, Ed.; Communications in Computer and Information Science. Springer, Berlin/Heidelberg (GE), Volume 134.

[10] Alshemali, Basemah & Kalita, Jugal. (2020). Improving the reliability of deep neural networks in NLP: A review. Knowledge-Based Systems. 191. 10.1016/j.knosys.2019.105210.

[11] Goodfellow, Ian & Shlens, Jonathon & Szegedy, Christian. (2014). Explaining and Harnessing Adversarial Examples. arXiv 1412.6572.

[12] Liang, Bin & Li, Hongcheng & Su, Miaoqiang & Bian, Pan & Li, Xirong & Shi, Wenchang. (2018). Deep Text Classification Can be Fooled. 10.24963/ijcai.2018/585.

[13] Samanta S., Mehta S. (2018) Generating Adversarial Text Samples. In: Pasi G., Piwowarski B., Azzopardi L., Hanbury A. (eds) Advances in Information Retrieval. ECIR 2018. Lecture Notes in Computer Science, vol 10772. Springer, Cham. https://doi.org/10.1007/978-3-319-76941-7_71

[14] N. Papernot, P. D. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, “The limitations of deep learning in adversarial settings,” in IEEE European Symposium on Security and Privacy (EuroS&P’16), 2016, pp. 372–387

[15] Nicolas Papernot, Patrick McDaniel, Ananthram Swami, and Richard Harang. 2016. Crafting Adversarial Input Sequences for Recurrent Neural Networks. In Military Communications Conference, MILCOM 2016-2016 IEEE. IEEE, 49–54.

[16] Zhao, Zhengli & Dua, Dheeru & Singh, Sameer. (2017). Generating Natural Adversarial Examples.

[17] Goodfellow, Ian & Pouget-Abadie, Jean & Mirza, Mehdi & Xu, Bing & Warde-Farley, David & Ozair, Sherjil & Courville, Aaron & Bengio, Y.. (2014). Generative Adversarial Nets. ArXiv.

[18] Hu, Weiwei & Tan, Ying. (2017). Black-Box Attacks against RNN based Malware Detection Algorithms.

[19] Bungum, L. and Björn Gambäck. “Evolutionary Algorithms in Natural Language Processing.” (2010).

[20] Ye, Z. et al. “Encoding Sentiment Information into Word Vectors for Sentiment Analysis.” *COLING* (2018).

[21] Go, A., Bhayani, R. and Huang, L., 2009. Twitter sentiment classification using distant supervision. CS224N Project Report, Stanford, 1(2009), p.12.

[22] Maas, Andrew & Daly, Raymond & Pham, Peter & Huang, Dan & Ng, Andrew & Potts, Christopher. (2011). Learning Word Vectors for Sentiment Analysis. 142-150.

[23] Rohde, Douglas L. T., and D. Plaut. “An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence.” (2005).

[24] TensorFlow: https://www.tensorflow.org/

[25] Keras: https://keras.io/

[26] Brunet, M., Alkalay-Houlihan, C., Anderson, A. & Zemel, R.. (2019). Understanding the Origins of Bias in Word Embeddings. Proceedings of the 36th International Conference on Machine Learning, in PMLR 97:803-811

[27] Angwin, J., Larson, J., Mattu, S., and Kirchner, L. Machine bias: Theres software used across the country to predict future criminals. and its biased against blacks. ProPublica, 2016.

[28] Caliskan, A., Bryson, J. J., and Narayanan, A. Semantics derived automatically from language corpora contain human-like biases. Science, 356(6334):183–186, 2017.

[29] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. Neural Comput. 9, 8 (November 15, 1997), 1735–1780. DOI:https://doi.org/10.1162/neco.1997.9.8.1735

[30] Kingma, Diederik & Ba, Jimmy. (2014). Adam: A Method for Stochastic Optimization. International Conference on Learning Representations.

[31] Anderson, E.J., Ferris, M.C. (1994). Genetic algorithms for combinatorial optimization: the assemble line balancing problem. ORSA Journal on Computing, 6(2), 161-173. https://doi.org/10.1287/ijoc.6.2.161

[32] Gebru T, Krause J, Wang Y, Chen D, Deng J, Aiden EL, Fei-Fei L. Using deep learning and Google Street View to estimate the demographic makeup of neighborhoods across the United States. Proc Natl Acad Sci U S A. 2017 Dec 12;114(50):13108-13113. doi: 10.1073/pnas.1700035114. Epub 2017 Nov 28. PMID: 29183967; PMCID: PMC5740675.

[33] S. Dodge and L. Karam, "A Study and Comparison of Human and Deep Learning Recognition Performance under Visual Distortions," 2017 26th International Conference on Computer Communication and Networks (ICCCN), Vancouver, BC, 2017, pp. 1-7, doi: 10.1109/ICCCN.2017.8038465.

[34] Devlin, J.; Chang, M.-W.; Lee, K. & Toutanova, K. (2018), 'BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding' , cite arxiv:1810.04805Comment: 13 pages .

[35] https://www.gainchanger.com/nlp-techniques-bert-algorithm/

[36] https://www.scribendi.ai/can-we-use-bert-as-a-language-model-to-assign-score-of-a-sentence/

[37] https://huggingface.co/bert-large-uncased

[38] Dai, Jiazhu, Chuanshuai Chen, and Yufeng Li. "A backdoor attack against LSTM-based text classification systems." *IEEE Access* 7 (2019): 138872-138878

[39] Tsai, Yi-Ting, Min-Chu Yang, and Han-Yu Chen. "Adversarial attack on sentiment classification." In Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural

[40] Iulia Cioroianu, Shaen Corbet, Charles Larkin, “The differential impact of corporate blockchain-development as conditioned by sentiment and financial desperation.” Journal of Corporate Finance, Volume 66, 2021, 101814, https://doi.org/10.1016/j.jcorpfin.2020.101814.

[41] Kapar, B, Olmo, J. Analysis of Bitcoin prices using market and sentiment variables. *World Econ*. 2021; 44: 45– 63. https://doi.org/10.1111/twec.13020

[42] M. Singh, “Vulnerability of Natural Language Classifiers to Evolutionary Generated Adversarial Text”, GitHub repository, https://github.com/masamiweb/GeneticAlgorithmAdversarialNLP