Cardiff University

Conversational explanations for deep learning systems

Final Report

Oliver Schwertfeger
5-11-2018

Abstract

The application of machine learning and deep learning models in recent years has become ubiquitous across multiple data-intensive domains. From e-commerce product recommendation to object recognition in images, these models are becoming increasingly present in our day to day lives, but can their classification predictions truly be trusted? In this paper, we aim to compare two models used for text classification, a Naïve Bayes based approach and an LSTM based approach. We then seek explanations from these models using LIME [1] predictions to determine if their predictions are accurate. The project aims to provide a useful insight into the data for the user, allowing them to decide if they should place their trust in predictions given by the classification systems. Our results indicate that using a machine learning-based approach provides better metric performance results over a deep learning based implementation. However, it is the deep learning based approached which provides superior results when classifications are explained.

Acknowledgements

I would like to thank Professor Alun Preece, Andrew Dawson, David Rogers and Daniel Harborne for their contributions towards this project.

Contents

Table of Figures

Table of code snippets

Introduction

In recent years, there has been an exponentially increasing amount of generated and available data. It is predicted that in 2018, 30 zettabytes of data will be created, a tenfold increase since 2010 [2]. This increase of data creation has led to traditional methods of data analysis becoming ineffective due to increasing labour and computational costs. This has sparked commercial and academic interest in the use of data-hungry deep learning-based models (AI) to solve big data issues. Coupled with the widespread adoption of graphics processing unit(GPU) powered computing [3] [4], deep-learning based models have had great success in solving computational intensive real-world problems. However, as the popularity of deep learning models has grown, little has been done in tackling one of the leading issues surrounding deep learning. The issue of transparency.

The motivation for this project comes from the lack of transparency of these deep learning systems. How do they come to their conclusions? And can we trust them? As the word deep suggests, these systems are formed by connecting multiple layers of neurons (which is what makes them deep) to create a network. As each layer can vary significantly in its size and function, numerous connected layers can often be complicated and difficult to understand without in-depth technical knowledge. This is referred to as the 'black box' problem of deep learning [1]. For a general user of a system, this can be an issue, as few classifiers are able to give reasoning to their classification. As more tasks become autotomonised, we, as users of these systems, are required to put more trust into them. But can we be confident that a system is making its decisions for the correct reasons? An example of this increased requirement of trust in the use of autonomous vehicles [5].

The core focus of the project is to develop and compare two text-based classification methods, which will classify social media data (tweets) into two categories and provide an explanation for classification into a category. The first is 'of interest' and the second is 'uninteresting' in the context of Crime and Security to a data analyst. The first classifier will use a machine learning based implementation; the other will use a deep-learning based implementation. Both of which are further described in the research and method sections of this report.

Supplementary to the need to improve transparency in deep learning systems, on 28th May 2018, the European Union's updated General Data Protection Regulation (GDPR) comes into force. One of the critical points of this regulation is Art. 15 - Right of access by the data subject. This states that "The data subject shall have the right to obtain from the controller confirmation as to whether or not personal data concerning him or her are being processed, and, where that is the case, access to the personal data and the following information:" [6]. This updated legislation now puts a legal obligation on companies which collect and use personal data to explain how and where an individual's data is used. The need for greater transparency of deep learning-based systems is now not just a performance improvement but, is a financial incentive to companies who use them. As companies who operate within the European Union and do not comply with the updated GDPR can be subject to hefty fines.

Research

Inspection of data:

A tweet is of 'interest' to a crime and security analyst when it has been manually labelled as such. The interesting tweets within this dataset have been collected by human analysts who have manually deemed that the tweet contains useful information and should, therefore, be stored and analysed.

The complete dataset for this project contains 20,772 individual labelled tweets. 973 of those are labelled as 'interesting' and 19,799 are labelled as 'uninteresting', giving a ratio of around 20:1. The timeline of the dataset starts on the 22nd March 2017 and concludes on the 3rd of July 2017. Figure 1 shows the count of tweets collected per day across the dataset. As we can see, there are four distinct spikes in the timeline that correspond to four events studied by the Cardiff University Crime and Research Centre.

1. 22nd March 2017 - Westminster attack, London
2. 23rd May 2017 – Manchester Arena attack, Manchester
3. 3rd June 2017 – Champions League final, Cardiff
4. 19th June 2017 – Finsbury Park attack, London

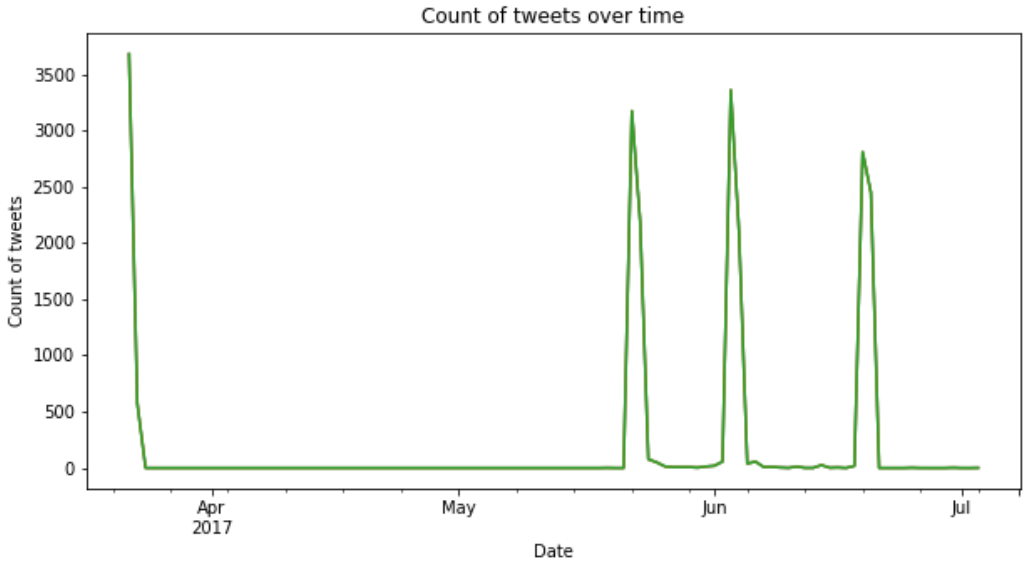


Figure 1. *Count of tweets over period of dataset.*

'RT @CNN: This message is greeting commuters at #FinsburyPark tube station as Londoners awake to another tragedyâ€¦ '

Tweet 1. *Uninteresting Tweet: Created on 19-Jun-17 @ 11:21:31*

'After weapons caches were stolen in several countries, the Saudi King (who sponsors terrorism) won't go to G20. Really makes you think... pic.twitter.com/4yuipQmabU'

Tweet 2*. Interesting Tweet: Created on 03-Jul-17 @ 06:33:00*

Figure 2 describes the count per day of interesting labelled tweets. Comparing to Figure 1, we can see that the interesting dataset does not include any tweets from before May 2017. Therefore the interesting dataset is unlikely to contain any tweets related to the Westminster attack. Figure 2 also shows the occurrence of an outlier, with one single tweet occurring in July (Tweet sample 1), which is unrelated to any of the previous events.

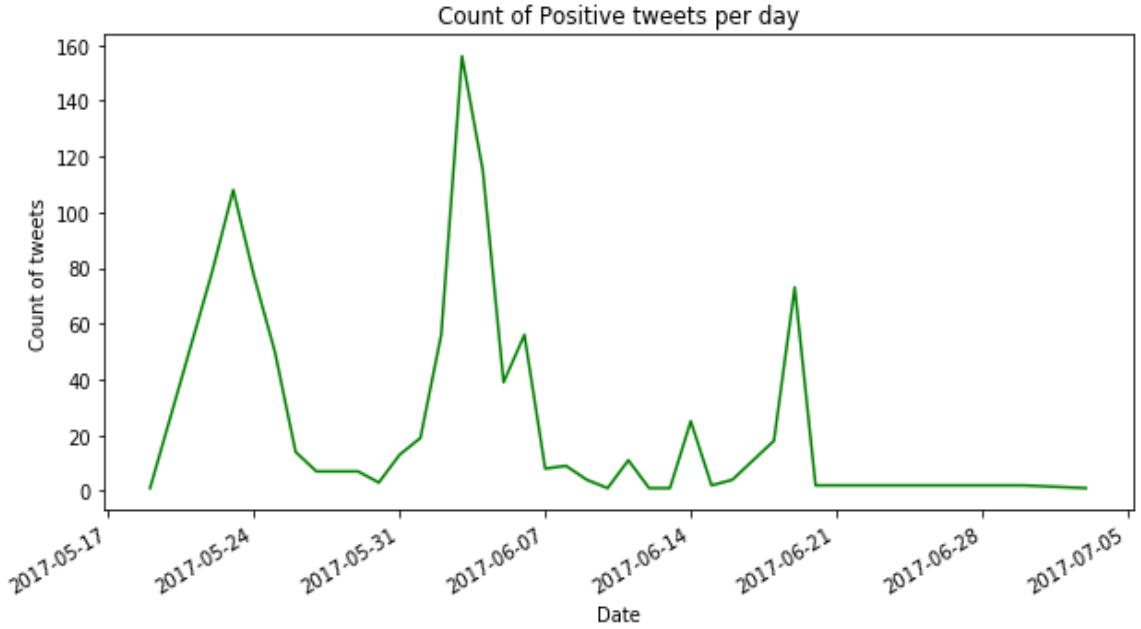


***Figure 2**. Count of interesting tweets per day.*

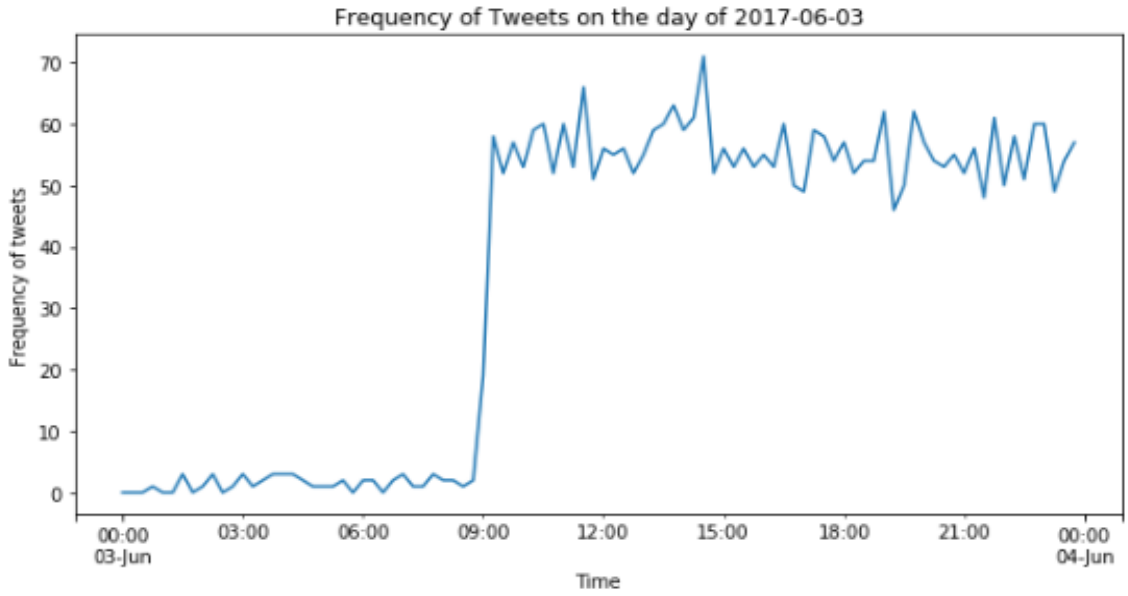


***Figure 3.** Count of tweets per 15 minutes on 3rd June 2017.*

Under and overfitting a model

During the training of a model, it is essential to be able to give the model sufficient data to train on. If there is insufficient training data, the model will not be able to learn the desired function of the data(underfitting), therefore, perform poorly. If a model is given too much data, it will learn the noise of the training data well, but when a test set is used, will perform poorly(overfitting). To prevent overfitting, a validation set can be of used in this scenario. Used during the training of a model, a validation set allows metrics to be computed which enable the model to adjust, reducing the chance for overfitting of the data.

Machine Learning models

To understand deep learning, one must first be familiar with the principles of machine learning, as deep learning is a sub-topic of machine learning. Machine learning is essentially the application of an algorithm that can learn from data. An algorithm is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E. [7]

To determine which algorithm is most applicable to our dataset, we must decide what category our dataset belongs to. Machine learning tasks are split into two main groups. Supervised learning (when the input data has in addition to itself, the output attribute that we want to predict or labelled data) and unsupervised learning (in which we only have the input data or unlabelled data).

As this project contains labelled data, a supervised learning approach is suitable. We then subdivide this into two types of problem, a classification or a regression problem. A classification problem requires one out of two or more classes to be predicted. Regression problems do not require a single class output. Instead, regression problems expect a continuous variable to be predicted as output. Our dataset is then most suited to use a *supervised* classification algorithm, in particular, an algorithm that is suited for binary classification problem as we are only trying to predict two classes.

Many algorithms can be used in this case; typical examples include support vector machines(SVM), decision trees and Naïve Bayes. However, for this project, we have chosen to implement the Naïve Bayes model as our machine learning algorithm. Even though it is not the highest scoring method for text classification (the Bayes method is thought to be a benchmark in machine learning, as its strengths come from the algorithm' high efficiency. This allows for models to be designed, trained and tested quickly, as the algorithm only requires one pass over the training data to fit and test while delivering good results.

Naïve Bayes

The Naïve Bayes is a method of probabilistic classification method based on Bayes' Theorem. *Bayes' theorem describes the conditional probability of an event. Based on prior knowledge of conditions that might be related to the event.* [8]

A Naïve Bayes classifier assumes that the presence of a feature in a class is unrelated to the presence of another feature. Meaning all features associated with a class are treated independently to another when contributing to the probability that a data sample belongs to a specific class. This assumption is where the 'naïve' connotation of Bayes occurs, as the assumption is often incorrect. When probabilities are calculated, it is rare that a sample has a 100% probability of belonging to a single class. Therefore, the class with the highest probability is outputted as the predicted class of the sample.

$$P(x \mid y) = \frac{P(x)\, P(y \mid x)}{P(y)}$$

Multinomial Naïve Bayes

This implementation is used for multinomially distributed data. In this model, features are treated as event probabilities so that a word can have a probability value in the range of *0 – 1.* It is the most widely used implementation of Bayes as it has been shown to outperform other Bayes classifiers in the field of text classification. [9]

$$\hat{\theta}_{yi} = \frac{N_{yi} + a}{N_y + an}$$

Bernoulli Naïve Bayes

This implementation is used for multivariate Bernoulli distributed data. Unlike Multinomial Naïve Bayes, features are treated as binary. It is either present with a value of *1,* or it is absent, which gives a value of *0.* This implementation only works on datasets with two classes (binary classification).

$$P(x_i \mid y) = P(i \mid y)x_i + (1 - P(i \mid y))(1 - x_i)$$

Parameter optimisation

The multinomial Naïve Bayes method supports the use of Laplace smoothing. Laplace smoothing is a method of smoothing categorical data. If tuned correctly the application of this can lead to an improved performance compared to the Bernoulli Naïve Bayes.

Deep learning-based models

Why deep learning?

When attempting to predict complex patterns, deep learning models are preferred over machine learning models due to their ability to learn complex non-linear functions. Unlike machine learning algorithms, neural networks have many complex additional functions that can be applied to adjust its predictions. Neural networks are typically trained by using iterative, gradient-based optimisers which require multiple passes over the training dataset to train.

Primary flow of a neural network

The most basic of deep learning models is that of a feedforward neural network or a multilayer perceptron (MLPs). Their function is to map an input 'x' to an output category 'y'. A layer is the definition of a function that is to be applied to data. As a layer can vary significantly in size and function, it is useful to categorise layers into three types; an input layer, a hidden layer(s) and an output layer (as shown in figure 3). The chaining of multiple layers defines a model.

The input layer takes the input data and feeds it to the second layer of the network. The second layer then applies a function to the input data and passes it onto the 3rd layer. The 3rd layer then applies another function and passes the input to the output layer. The output layer then outputs a predicted value. The second and third layers are classed as hidden, as the input data only specifies the input dimension and output dimension of the network, not what should occur between these layers. In this example, our model has two hidden layers, but in theory, a network could have an infinite number of them. However, this can vary due to implementation. The network is feedforward if connections between layers do not form a cycle.

As a network can comprise many layers, the overall number of layers gives us the depth of the model, which is where the term 'deep learning' arises, as more layers means a deeper model.

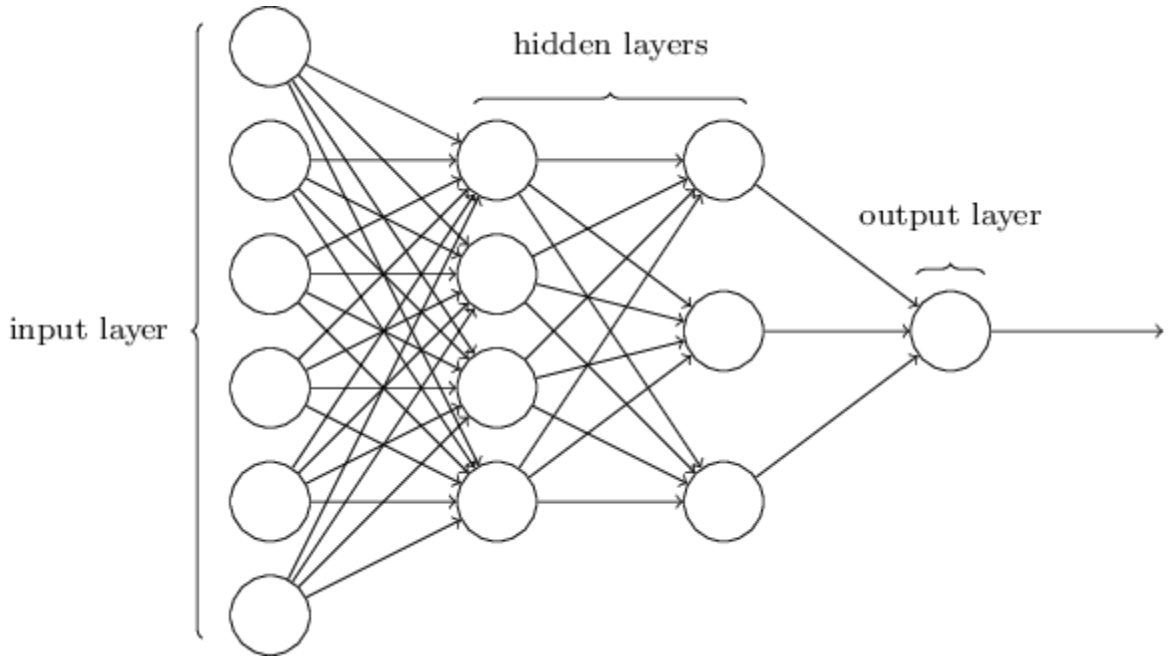


***Figure 4**. A diagram of a fully connected MLP[1] [10]*

Training of a neural network

Each layer of a network is comprised of at least one neuron. A neuron's function is to sum all the data passed to it, multiply each input by a weight and then add a bias to that sum (equation below). The resulting calculation is then passed onto the next layer for further computation.

$$N = \sum(inputs * weights) + bias$$

A neural network is trained by using a loss function to determine the error between a model's predictions and the targeted output of the training data. This value of error is then given to an optimiser, whose job is to adjust the weights of the neurons in such a way to minimise loss. The optimiser achieves this by using a method called back propagation, which 'sends back' the value of loss to the previous layer, the optimizer then uses this value of loss to adjust the weights of the neurons.

Activations functions

An activation function is a used to apply either a linear or non-linear functional mapping between the output values from a neuron to a defined output range. The activation function is required as a neuron cannot control the size of the values it receives. Therefore, once it has completed its computation, the resulting (*N*) value can be infinitely large, making it potentially unsuitable for input into another layer. This is particularly the case of output layers, where the output value is required to be within a certain range. (i.e. *0 -1*).

[1] - https://github.com/ledell/sldm4-h2o/blob/master/sldm4-deeplearning-h2o.Rmd

Non-linear functions have an advantage over linear functions as the output range of the function can be specified, whereas a linear function as a range of -infinity to infinity. Below are some non-linear functions.

- Sigmoid – An S-shaped function, which maps between 0 and 1.
- Tanh (hyperbolic tangent) – An S-shaped function, which maps between -1 and 1.

Both Sigmoid and Tanh functions suffer from a problem called vanishing gradients. This occurs when the input values to a function map to the limits of the function, causing a loss in gradient. This is a problem as optimiser functions use gradient values to adjust the weights of a neuron to improve model performance. [11]

- Rectified Linear Unit (ReLU) – A linear-shaped function when the input is greater than 0, otherwise flat. Maps from 0 to infinity.
- Softmax – Similar to Sigmoid but, used for multi-classification tasks.

Loss functions

Calculated during the training of a network, loss functions are used to measure the error between a model's predicted output and the actual output. The higher the error, the worse the model tends to perform when tested. The loss function that will be used for this project is binary cross-entropy, which is used to measure the loss for input with the probability value between 0 and 1. In this project, a data sample is either interesting (*1*) or uninteresting (*0*).

Optimiser functions

In this report, due to time constraints, we shall focus on only one type of optimisation algorithm, gradient descent. These algorithms minimise or maximise the loss function by using its gradient values and alter the weights of neurons to adjust the model. The examples provided are improvements based on previous optimiser functions such as Momentum [12] and AdaGrad [13].

Examples of optimiser functions

Adaptive moment estimation (ADAM) [14] is a method which computes adaptive learning rates for each parameter.

Adadelta [15] is an extension of AdaGrad, which adapts the learning rate over time and has minimal computational overhead compared to standard gradient descent algorithms.

Both functions are vast improvements than their predecessors, addressing the problems of the vanishing learning rate (discussed in the Recurrent neural network section of this report) and slow performance.

Types of neural networks

An essential aspect of a neural network is in its architecture, which refers to the overall structure of the network.

Convolutional neural network

A convolutional neural network (CNN) is a neural network which specialises in processing data with a grid-like topology; this type of network is not applicable for this project, as the dataset does not follow this format.

Recurrent neural network

A recurrent neural network (RNN) is a type of artificial neural network with the addition of feedback loops in its network topology. RNN's specialise in the processing of sequential data, such as sequences of vectorised text or pixels in an image. The addition of recurrent loops in the network allows the model to 'remember' what sequences it has been shown before, permitting previous computations to influence making new decisions.

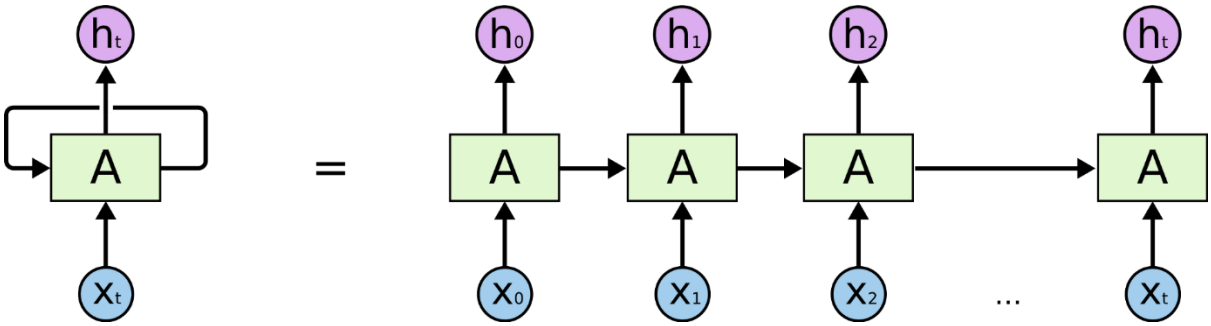


***Figure 4**. A 'rolled' RNN cell (left) and it's 'unrolled' structure.[2] [16]*

The ability of RNN cell to 'remember' is provided by a single tanh or ReLU activation function within the cells' internal structure (Figure 5).

The problem of long-term dependencies

In deep learning, a task displays long-term dependencies if computation of the current output depends on input presented at an earlier time than the current input.

It is challenging to train RNNs to capture long-term dependencies because their gradients tend to either vanish (most of the time) or explode (rarely, but with severe effects). This makes gradient-based optimization methods struggle, not just because of the variations in gradient magnitudes, but because of the effect of long-term dependencies is hidden [17] (being exponentially smaller with respect to sequence length) by the effect of short-term dependencies. [18]

[2] http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long term short memory

Introduced in 1997, long-term short memory (LSTM) [19] is a variant of an RNN that is capable of learning long-term dependencies. Since its introduction, it has gained popularity in the fields of natural language processing (NLP) tasks, outperforming previous models and has set new benchmarks for performance.

The internal structure of an LSTM cell consists of three gates (input, forget, output), and a cell unit. Gates use a sigmoid activation, while the input and cell state uses a tanh transformation

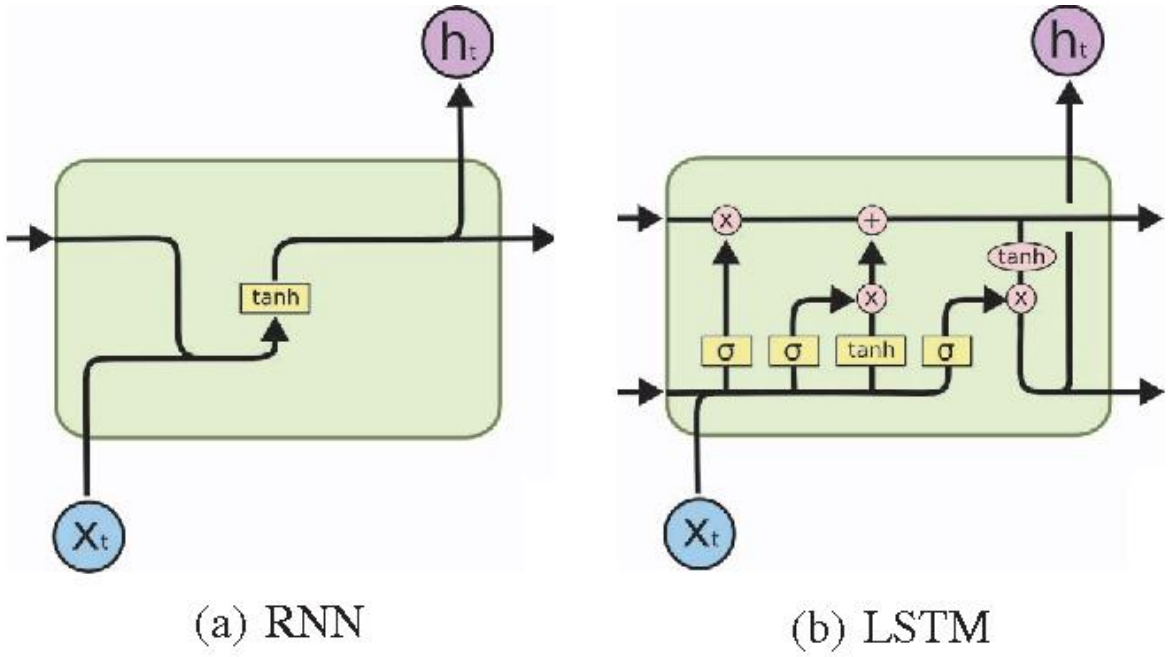


***Figure 5**. The internal structure of standard RNN unit (left) and LSTM (right)[3] [16]*

Figure 5 (b) shows the internal structure of the LSTM cell. The line running straight through the top of the cell represents the cells 'memory', allowing information to 'flow' between cells. The X_t entry into the cell represents new input data, the arrow in the bottom left corner represents the h_t input from the previous cell (which was ignored by the previous cell).

The first gate sigmoid (σ) is the input gate, it decides if an input value should update the cells state or should it be disregarded. The gate computes a value between 0 and 1, the lower the value, the more likely the new input will be used to update the cell state. If the cell determines its state should be updated, it passes the input through another sigmoid layer and then through a tanh function which is used to map the output of the sigmoid function to a range of -1 to 1. The cell then multiplies the tanh output by the output of the sigmoid gate; this removes the features that the cell did not require, allowing the cell state to be correctly updated. [16]

[3] http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

Training a neural network

Batching & Epochs

Batching is a technique used to specify how much training data is fed into the network before weights are adjusted. Batching also affects the training time of the model, as the smaller the batch size, longer the training time. The value of epoch states the number of times the model will be shown the entire training dataset. Large epoch values should be used for larger batching, as this will give the model more time to minimise loss.

Early stopping

During the training of models, model performance can sometimes plateau after a certain number of epochs. After this point, the continuation of training provides no significant improvement to the performance of the model, it instead merely increases the total time for the model to train. To determine when a model should stop training monitoring of the validation error is required. When the error of this metric does not improve after a given number of epochs, the model should stop training.

Dropout

Dropout is computationally inexpensive, and useful method of preventing over-fitting. It works by randomly dropping out neurons from the network during training. In a fully connected layer, neurons are prone to develop a co-dependency amongst one another. This co-dependency reduces the impact of neurons not connected to this dependency. Therefore, the random removal of neurons during training is supposed to stop the formation of co-dependencies.

CPU training vs GPU training

The central processing unit (CPU) and GPU are two essential parts of computer hardware. The CPU is the central unit in computer hardware which conducts arithmetic required for programs to run. GPU's are also used for computation but have typically been specialised to processing inputs in the form of vectors (initially images). GPU's are capable of performing thousands of more calculations a second more with a CPU's. As training a deep learning model is a computationally expensive task to do, it makes sense for these computations to be run on a GPU instead of a CPU. The result of this is a significant improvement in reducing the time taken for training.

Methods for text vectorisation

The need for pre-processing of text

Text in its raw format is not processable by computer. To allow computations to be performed using text, it must be converted into a vector. The process of text vectorisation is main up to three parts. Text tokenisation, text 'cleaning' and vectorising.

Text tokenisation is when a sentence or list of phrases are split up into individual words. These words are then individually inspected by a text 'cleaner', whose job is to remove 'noise' in the dataset. In text classification, noise is defined as a character or word that does not add value when the probability of a sentence belonging to a category is calculated.

The removing of items such as punctuation, digits and common English words (also known as stop words) is the standard practice of a text cleaning function. The removal of URL's is also recommended when processing tweets, as a large proportion of tweets use these, but rarely add value to a classification unless a URL is repeated shared which is unlikely.

Bag of words model

The bag of words (BoW) model is a simple and computationally cheap method used to extract features(words) from a corpus (collection of documents/words). BoW is a list representation of the number of individual words within a corpus and the occurrence of each word without considering their order. This list can then be used to convert sentences into vectors by comparing the sentence against the BoW model, ready for input into a machine learning/deep learning model.

One hot vectors

One hot vectors is a method of vectorising a sequence of words. It is the simplest method suitable for input into a model. The length of the vector is the total size of the BoW. When a text is encoding using one hot, its length is the total length of the vocab. If a word is present, there is a 1 value. Otherwise, it is 0. This approach gives a 'sparse' matrix, which is largely filled with 0 values.

```
Vocab = [one,two,three,four,numbers]

Sentence = [one,three]

One-hot encoding = [1,0,1,0,0]
```

***Figure 6**. One hot encoding example.*

TF-IDF

TF-IDF (term frequency-inverse document frequency) is a statistical method that determines how important a word is in a corpus. Typically used in large documents, a word is weighted based on its rate of recurrence within a document, compared to its frequency within the corpus. This weighting helps to normalise frequently occurring words.

Performance metrics

To accurately evaluate a model, we need to compute a variety of performance metrics that give a holistic view of the model; these metrics can then be directly compared against other models to determine which method is superior.

Accuracy

Accuracy is calculated as the number of correct predictions / total number of samples in the dataset. In unbalanced datasets, this metric can be misleading and should not be solely relied on.

$$accuracy = \frac{number\ of\ correctly\ predicted\ samples}{number\ of\ samples}$$

Confusion matrix

In (binary) classification tasks, four base figures can be calculated that determine how successful a model has been at predicting the defined classes.

- True positive – This score is generated by the count of correctly predicted samples in the positive category
- True negative – This score is generated by the count of correctly predicted samples in the negative category
- False positive – This score is generated by the count of incorrectly predicted samples in the positive category
- False negative – This score is generated by the count of incorrectly predicted samples in the negative category

These four figures are usually plotted in a confusion matrix, which gives an easy to interpret matrix or graph that provides greater insight into how a classifier works compared to accuracy alone.

Precision

Precision is used to determine the proportion of positive classifications were correct.

$$precision = \frac{tp}{tp + fp}$$

Recall

Recall is used to determine the proportion of actual positives that have been correctly identified. In binary classification, recall is known as sensitivity.

$$recall = \frac{tp}{tp + fn}$$

F1-Score

This is a weighted average of recall and precision. It provides a better representation of a model's accuracy when using unbalanced datasets.

$$F_\beta = (1 + \beta^2)\frac{precision\ \times recall}{\beta^2 \times precision + recall}$$

Cross-validation

Cross validation is a model evaluation method that is better than training and testing a model a singular time. The problem with singular evaluations is that they do not give an indication of how well the model will do when it is asked to make new predictions for data it has not already seen. One way to overcome this problem is to not use the entire data set when training a model. Some of the data is removed before training begins. Then when training is done, the data that was removed can be used to test the performance of the learned model on ``new'' data. [20]

LIME

LIME [1] can explain how each word contributes towards a class prediction. When used in text classification, LIME takes a single sample as input (a sentence or sequence of words) and replaces and removes each word. Creating 5000 combinations of the sequence of words. It then runs these combinations through a trained model, recording the scores of each combination. As words are removed, the probability towards a class is expected to change. Therefore, when a word is removed, the larger the fall in probability towards a class, the more likely that the word positivity contributes towards that class prediction.

```
                ##  LIME  ##
Original sentence
['The women asked the man, what is your name?']

Combination #1
['. women asked the man, what is your name?']

Combination #2
['The . asked the man, what is your name?']

...

Combination #5000

['The women asked the man, what is your .?']
```

***Figure 7**- Lime example*

Tools and libraries used

Below are listed the installed libraries used for this project.

Anaconda 3 & Jupyter notebooks

Anaconda 3 is a Python library distribution package. It has allowed for fast and straightforward installation of multiple vital packages such as Pandas, NumPy and Matplotlib which are used throughout the implementation. Jupyter notebooks [20] are a particularly important part of this distribution, as they provide a python-based editor through a web browser. Using Jupyter notebooks has allowed for quick prototyping of models and easy sharing and showcasing of code.

NLTK

An imported Python library that is used in the pre-processing of text. The library contains a corpse of stop-words which are used to filter commonly used words from the dataset.

GloVe (pre-trained word vectors)

In the implementation of the LSTM, a GloVe [21] pre-trained vector file to in an embedding layer to improve performance. The data used contains 400,000 unique words.

Scikit-learn

As a part of the Anaconda distribution, Scikit-learn [22] is machine learning library in Python. It provides simple and efficient tools for data mining and data analysis tasks. It has been used to create the machine learning models and Keras wrapper in this project.

LIME

Lime [1] is a library which can be used to explain predictions behind a black box classifier. It also supports sci-kit's pipeline function which allows for easy integration into a sci-kit model.

TensorFlow

TensorFlow [23] is a popular deep learning library. Predominantly written for use in Python, it is a well-supported standalone library itself, using NVIDIA's CUDA / CuDN software for GPU training.

Keras

Keras [24] is a high-level neural networks API written in Python. For this project, Keras was used to design and implement the LSTM using TensorFlow backend, as Keras provides multiple easy to use text-pre-processing tools.

NVIDIA

In order to speed up the training time of the neural network, training on GPU compared to a CPU became a necessity. [25] To do this TensorFlow required two pieces of NVIDIA software. The training performance comparison can be viewed in the implementation section of this report.

1. CUDA Toolkit9.0 [3] – allows for GPU training and parallel processing using
2. CuDNN v7.0 [26]– GPU accelerated library for deep neural networks.

Experimental work

Choice of dataset

To validate and develop the skills required to create a machine learning based classifier, we decided first to implement a Bayes classifier using a well-known and researched dataset. After some extensive research, we decided to use the 20Newsgroup dataset as a benchmark. [27] As it was found to be supported by multiple research papers which had given comparable metrics to achieve. LIME paper also cites the 20Newsgroup as an example of its application. [1]

Bayes implementation

Much of the work from this implementation has been built on to create the features used in both machine learning and deep learning implementations. As an excellent introduction to machine learning, the Sci-kit learn library contains the 20newsgroup dataset ready for input into a classifier. The module allows importation of all 20 newsgroups, but Atheism and Christianity have been chosen as categories to enable comparison against performance scores in well cited research papers. [1] [27].

After the importation of the relevant libraries, Code Snippet 1 defines the required categories for training and testing. The function fetch_20newsgroup then splits the data into a train and test set. It takes in parameters to specify what parts of the dataset should be loaded. In this example, the headers and footers from each sample are removed, this is used to improve the reliability of the model during prediction. [28]

```
# choosen categories to be trained
categories = ['alt.atheism', 'soc.religion.christian']

# Gathers training and testing data, removing headers and footers from data samples
newsgroups_train = fetch_20newsgroups(categories =categories, remove = ('headers','footers'))
newsgroups_test = fetch_20newsgroups(subset='test', categories = categories, remove = ('headers','footers'))
```

***Code Snippet 1**. Loading in data for Newsgroup classifier.*

A pipeline function is then created. The pipeline function is a sequential list of transforms, followed by the definition of the classifier model. The transforms steps are used to manipulate the input data to the pipeline (dataset) into the required input for the model. In this case, we use the CountVectorizer function to create a BoW model from our text corpus. The stop_words parameter is used to compare the corpus against common English words and stops them from entering the BoW.

The pipeline is then ‘fed’ the training data, which is given as a list of strings (newsgroups_train.data) and the actual class of the string (newsgroup_train.target).

This can, however, be simplified by using a TFIDFVectorizer instead.

```
#Create pipeline with transform functions
pipeline = Pipeline([('vect', CountVectorizer(stop_words='english')),
                     ('tfidf', TfidfTransformer()),
                    ('clf', MultinomialNB(alpha =0.1)),])

# train model
pipeline.fit(newsgroups_train.data, newsgroups_train.target)
```

Code Snippet 2. *Pipeline creation and model fit for Newsgroup classifier.*

Testing of the classifier occurs when the predict function is called with the input of the testing data

```
#test model
predicted = pipeline.predict(newsgroups_test.data)
accuracy = np.mean(predicted == newsgroups_test.target)
print("Accuracy of the model is %d" %(accuracy*100)+ "%")
```

```
Accuracy of the model is 89%
```

Code Snippet 3*. Testing of Naïve Bayes implementation for Newsgroup dataset.*

This high level of model accuracy was consistent with the ones published in well cited research papers which was between 83-91%.

Method

Data cleaning and pre-processing

The first task of this project was to process the data in such a way that it could be easily used as input into both models. Once loaded, a pre-processing function was defined that took the raw text as input and outputted a text sequence that would be ready for input into the models.

Once the texts had been pre-processed, we then needed to generate the required datasets used for model training and testing. To do this, a train/test split function was used to pseudo-randomly create the training and testing sets while maintaining the ratio of interesting to uninteresting tweets. After consultation with various professionals in the field of machine learning, it was suggested that an 80:20 training/test split should be used for the machine learning model and a 70:10:20 train/validation/test split for the deep learning model. The validation set used in the deep learning model is created later during the initialisation of model training.

As the collected dataset has roughly a ratio of 20:1 (uninteresting to interesting), if time permitted, it was decided to investigate how the change in ratio changed the performance of a model. Typically training on an equal split ratio of classes is recommended, but as this split does not replicate the real-world ratio, it was considered insightful to explore. To allow a fair comparison between all models, all random functions used the same starting seed, so that all models would have the same randomly generated sequence of training and testing data.

Design of the machine learning model

Given the research into which Bayes approach should be used for this project, it was still not clear which implementation should be favoured. From an academic view, most of the papers that researched Bayes preferred the Multinomial approach [9] [27]. However, the description of the Bernoulli approach seemed to be better suited to this binary classification problem. It was therefore decided to create two Bayes models for comparison. The best performing method would then be used to compare against the deep learning implementation. The BoW approach was chosen to learn the vocabulary ready for input into the models, as it is simple and easy to implement for beginners to machine learning.

The models were implemented using the sci-kit learn library. This was due to the familiarity with available functions in the library, gained from prototyping the 20newsgroup dataset. The high-quality documentation of the libraries. API's and source code also made implementing and learning new concepts easier. Unlike the 20newsgroup dataset, the TF-IDF was not used in implementing the final Bayes classifier, as during prototyping it leads to a reduction in overall performance compared to the standard BoW model.

Design of the deep learning model

Out of the available versions of deep learning models, it was decided that a model with the ability to 'remember' certain words which caused a probability to either category should be used. This naturally led to the introduction to RNN's, and subsequently to the LSTM variant. Due to its ability to handle long-term dependencies, leading to improved performance over RNN's. [29]

The implementation of the model was initially written using TensorFlow. During the research stage of the project, it seemed that TensorFlow was the most widely used and supported deep learning framework, meaning there was more likelihood for support to be available should any problems occur during development of the implementation. TensorFlow also supports the training of neural networks using a GPU. However, we encountered a lot of issues in using the TensorFlow method with LIME, so implementation was switched to Keras.

Model parameters

Unlike the Bayes model, deep learning models have a plethora of functions that can be fine-tuned to improve model performance. The majority of these were decided during the laborious stage of training the model using trial and error. During our research, it seemed a common theme that using the ADAM optimiser would provide the best results for the project. [30]

An additional design decision for this model was the introduction of an embedded layer. Due to the reliable computational power of modern computers, and small dataset size, the one-hot encoding of vectors used in the machine learning approach had little effect on the training time. Therefore, there was no need for further runtime optimisation. However, in a neural network, the vectors(tweets) have calculations frequently performed on them as weights are adjusted during training. The sparseness of one hot-encoding means that many redundant calculations occur, contributing to a significant increase in training time of a deep learning model to perform the task of text classification tasks.

Performance comparison

During the initial design stages of this project, it was determined that if a classifier could not achieve good results, then it would be preferable for it to classify negatively. As the system could then be used to omit a significant amount of uninteresting twitter data reducing the amount of data an analyst would have to look through.

To determine how successful a model was, various performance metrics needed to be computed. When the models were first being tested, the output metrics given as the precision score, recall and F1 score. As knowledge grew about how functions worked, confusion matrixes were introduced to provide more insight into how changing model parameters affected model performance. When it came to compare the models, 10-fold cross-validation was used to provide assurance that the model's score was reliable and wasn't an anomaly caused by a distorted training and testing set. The value of 10-folds was chosen as it is a good balance between the time taken and validity of results [31]

LIME predictions

The application of LIME predictions is well supported by its creators. However, during this project, there was no function which could provide multiple explanations at the same time. It was therefore required to design a function which would generate explanations for the entire testing set. This would allow the analyst to gain an insight into if the classifications are correct. To compare the explanation results between each model, the top 25 contributing words for a positive (interesting) and negative (uninteresting) classification were shown to an analyst. To avoid bias, the analysts were not told the name of each model but asked to decide which model they trusted the most and therefore believed would be the most useful when applied to real-world setting.

Implementation

There is frequent code re-usage between all implemented models. Especially between both Bayes implementations, (it is almost identical). To provide a more concise description to the reader, the explanations of model implementation will not be repeated. Full implementation details can be found in the code submission. It is also worth noting the change in naming convention used in this implementation. To provide greater clarity during prototyping, the naming convention of 'x' as data and 'y' as target have been changed to 'data' and 'target'.

Data formatting

The dataset for this project was collected from two different systems at the Cardiff University Crime and Research Centre. The interesting tweet data were sourced from a social media database, which had been populated by range of social media platforms. The other was sourced from the output of a web scrapper called Sentinel, which provided the uninteresting tweets. The two systems outputted different file types, a txt file and a json file. Therefore, it made sense to convert both files to the same file format so that they could be processed more easily.

```
## Opens up interesting tweets file, grabs the text data of the tweet ONLY. Prints to console
f = open('Data/edittedTwitterdata.txt', 'r')
string1 = "as post date and has "
string2 = " as post text"
for line in f:
    pos1 = (line.index(string1)) #tweet id index
    pos2 = (line.index(string2)) #tweet data index
    print(line[pos1+21:pos2])
f.close()
```

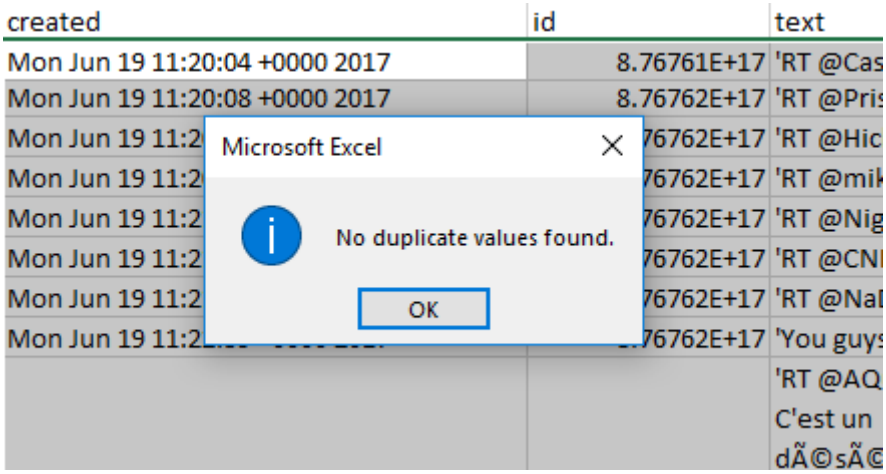
```
'@Robinhughes66 @GarethEnticott @cardiffcouncil Add this to the facial recognition that will be in place. It&#39;s all a bit we
ird. Could understand it for NATO, but not a football match. &mdash; Cardiff By Bike (@cardiffbybike) May 22, 2017'
'Police responded to reports of an incident at Manchester Arena. Please stay away from the area. More details to follow....'
'They are people that are bleeding , crying , legit injured at Manchester show at the moment. What is happening . I was there I
heard a blast'
'All crew of police came with guns outside the arena . Something big is up. #DWTManchester'
'Victoria station near the #Manchester Arena has been evacuated, all lines at the station are suspended now.. pic.twitter.com/5
AI0j9UNIm'
'A GUN SHOT WENT OFF IN THE ARENA AND THERES BLOOD ON THE FLOOR'
'Terrifying tweets coming from an Ariana Grande concert in Manchester pic.twitter.com/8sAUy5G2ZS'
```

***Code snippet 4**. Index search text data in the interesting tweets data output file.*

On inspection of the files, they both contained extra information related to each tweet that was not going to be useful for the classifier. For this project, we are only interested in the text of a tweet and its creation date. Code snippet 4 gives an example of the type of function that was used to extract the text data from the interesting data file. Similar functions where used to retrieve information from the json file.

Once the required data had been extracted from both datasets, they were exported to tabula format(csv) to conduct further processing. The first task was to check to see if there are where duplicated entries in the set, as repeats of the same value could cause a distortion in the data. Only when it came to view the files in file explorer was it noticed that the CSV data files take up more space than a standard excel workbook. The csv files were then saved a .xlsx instead for better transportability between computers.

Code snippet 5. shows Excels' remove duplicates function in which found no duplicates.



***Code snippet 5**. Remove duplicate function in Excel*

Data pre-processing

Loading in of data

Across all models, the *get_data()* function has been used to load both datasets and return pandas data frame data types. As dataset is unbalanced, uninteresting dataset is randomly sampled to match the count of interesting tweets. The random state value is the starting seed for this random selection and is kept the same to ensure all models are fed the same, pseudo randomly generated dataset.

```
def get_data():
    #Read in excel files. Add to dataframe.
    uninteresting = pd.read_excel('Data/jsonOutput.xlsx')
    interesting = pd.read_excel('Data/InterestingTweets_dates.xlsx')

    sampled_unintestingdata =uninteresting.sample(n=len(interesting),random_state = 999)
    uninteresting_data = sampled_unintestingdata[["text",'target']]
    interesting_data = interesting[["text","target"]]

    return(uninteresting_data,interesting_data)
```

***Code snippet 6.** Load in excel files and split into two files.*

Data cleaning and train/test split

```
def text_preprocessor(tweet_text):
    import re
    lower = tweet_text.lower() #converts to lower
    removeMdash = re.sub("mdash","",lower) #removes mdash
    removeJune = re.sub("june","",removeMdash) #removes mdash
    removeRT = re.sub("RT","",removeJune) #removes RT
    removedURL = re.sub(r"http\S+", "", removeRT) #removes URL
    removedDigits = re.sub(r"\d+", "", removedURL) #remove digits
    removeSmallWords = re.sub("\b[a-zA-Z]{1,3}\b", "", removedDigits)
    removepunc = [char for char in removeSmallWords if char not in string.punctuation] #removes puncuation
    removepunc = ''.join(removepunc)
    #converts to lowercase and removes stop words.
    return ' '.join([word for word in removepunc.split() if word not in stopwords.words('english') and len(word)>3])
```

***Code snippet 7**. Pre-processing function.*

The *text_preprocessor ()* function takes in a tweet as an argument, it converts the words in the tweet to lower case, removes a selection of specialist stop words, digits, URLs and common English words. The function then returns a list of processed(cleaned) words which are larger than 3 characters long.

The *split_data()* function takes both loaded datasets as parameters, each dataset is then split into training and testing files with a split of 80:20 training to testing data. The two files are subsequently split up into text (containing the tweets) and target files. The *text_preprocessor ()* function is then applied, and four files are returned.

```
def split_data(uninteresting_data,interesting_data):
    #generate train/test
    interesting_train, interesting_test = train_test_split(interesting_data, test_size=0.2,random_state = 10)
    uninteresting_train, uninteresting_test = train_test_split(uninteresting_data, test_size=0.2,random_state = 10)

    training_file = interesting_train.append(uninteresting_train)
    testing_file = interesting_test.append(uninteresting_test)

    #apply text pre-processing
    training_data =  training_file['text'].map(lambda x: text_preprocessor(x))
    training_target = training_file['target']
    testing_data = testing_file['text'].map(lambda x: text_preprocessor(x))
    testing_target = testing_file['target']

    return(training_data, training_target,testing_data,testing_target)
```

***Code snippet 8**. Generation of datasets*

During the initial stages of development, this pre-processing step was performed by passing the *text_preprocessor ()* to the model. However, this lead to an increased computational overhead, as the *text_preprocessor ()* results were not saved on exit of model training and testing and therefore had to be recomputed every time the model was run.

Code snippet 9 showcases the input and output from the *text_preprocessor ()* function. Index 16,106 shows how the majority of 'filler' words like 'Pls' are removed. One exception

```
Raw tweet data
7937      'vou chorar ate o final https://t.co/RxoHJts0n9'
12889    'RT @NaDialna: Un terroriste fonce sur des mus...
16106    'RT @NHSMillion: Pls RT to thank all the emerg...
11276    'Thoughts &amp; prayers for Westminster. I'm s...
10771    'RT @jonyterter: Ca se dit fidÃ¨le d'Allah mai...
Name: text, dtype: object

Processed tweet data
7937                                          chorar final
12889    nadialna terroriste fonce musulmans sortie dun...
16106    nhsmillion thank emergency services acted brav...
11276    thoughts prayers westminster sorry hear attack...
10771    jonyterter fidã¨le dallah mais poignarde gens ...
Name: text, dtype: object
```

Code snippet 9. *Comparison between raw and processed tweet data.*

to note is that this function does not remove twitter handles or hashtags used in tweet data. Instead, it merely removes the punctuation surrounding it. Therefore, in 12,889 indexed sample, one can see how an individual has retweets the @NaDialna handle, which has subsequently been converted into nadialna. This is a 'stupid' feature of the pre-processor and does lead incorrect categorisation later.

Machine Learning implementation

Multinomial definition

```
#Beginning of Pipeline
# MultinomialNB implementation
pipeline = make_pipeline(CountVectorizer(stop_words='english'),(MultinomialNB(alpha =0.1)),)
```

Code snippet 10*. Multinomial model and CountVectoriser definition.*

The CountVectoriser function is used to take the input of tweets and return a matrix of token counts. (BoW) Use of smoothing set at 0.1. As described in the 20newsgroup implementation, the pipeline function is a list of transforms and functions which are grouped together for ease of model implementation and showcasing of code.

Bernoulli definition

```
#Beginning of Pipeline
#BernoulliNB implementation
pipeline = make_pipeline(CountVectorizer(stop_words='english'),(BernoulliNB()),)
```

Code snippet 11*. Bernoulli model and CountVectoriser definition.*

In the Bernoulli implementation it is worth noting that there is no Laplace smoothing, as shown in the description of the model, there is no α in its equation.

Training and testing

```
#Train and generate predictions
predictions = pipeline.fit(training_data, training_target).predict(testing_data)
```

***Code snippet 12**. Bayes model training and testing.*

This pipeline function is an optimised method for training and testing a model. '. fit' performs the training of the model, with the supplied training_data and training_target. '.predict' then tests the model using the testing data.

Visualisation of performance metrics

The *plot_confusion_matrix()* function constructs a graphical presentation of the confusion matrix. It can display either the count or the ratio of each class. During the trial and error stage of model training, sci-kit learns classification report was used to provide an quick overview of precision, recall and f1-score for each category.

Cross Validation

The cross-validation splitter function requires one data input and one target input. As the current implementation already defined the train/test split, the *get_cv_data()* function was created to act as wrapper function for the other *get_data()* functions.

```
#get CV data
def get_cv_data():
    data_int, data_unint = get_data()
    training_data, training_target,testing_data,testing_target = split_data(data_int, data_unint)

    cv_data = training_data.append(testing_data)
    cv_target = training_target.append(testing_target)

    return(cv_data,cv_target)
```

***Code snippet 13**. Cross validation load dataset function.*

StratifedShuffleSplit is the method used to perform the splitting of cv_text, unlike a standard splitter it maintains the ratio between the categories based on their corresponding target values given in cv_target. The metrics that we want to keep track of during each cross-validation are defined in the scoring_metrics dictionary, these are precision, accuracy, recall and the values that make up a confusion matrix

```
#Define CV values
CV_function =StratifiedShuffleSplit(n_splits=10,random_state = 1)

scoring_metrics = {'precison': make_scorer(precision_score),'acc': make_scorer(accuracy_score),
                   'recall' :make_scorer(recall_score),
                   'tp' : make_scorer(tp), 'tn' : make_scorer(tn),
                   'fp' : make_scorer(fp), 'fn' : make_scorer(fn), }

cv_text,cv_target = get_cv_data()
```

```
#Computes CV results for classifer
cv_results = cross_validate(pipeline, cv_text, cv_target,cv=10, scoring=scoring_metrics)
```

***Code snippet 14**. Bayes cross-validator.*

Lime

The prototyping LIME code has been omitted from this report, but it can be found in the code submission files. Further to how LIME was described in section research, the LIME library is still in its infancy stages and does not have much functionality. During the implementation of this project, it was only possible to gain one explanation at a time from LIME. Generating the explanations for the testing dataset was more difficult than expected. As when even when using a loop on the explain instance, the explainer would crash at around 150 instances. Therefore, the *get_class_predictions()* function was created(Code snippet 15). It can adapt to a varying input of testing data, making it useable across ratio splits. It then returns the count of each explanation and it's total sum.

The sum of a LIME prediction is the total amount that

In order to reduce bias towards which frequently occurred in a class (but had a low scoring) the score values where divide by the number of occurrences it had been used to generate a prediction.

The lime explainer crashes if more than 200 instances are passed through it in this way.

```
for batch_count in range(1,number_of_loops+1):
    index_addition = 0
    for index in range(100):
        idx = index+index_addition
        explainer = LimeTextExplainer(class_names=categories)
        exp = explainer.explain_instance(testing_data[idx], pipeline.predict_proba, num_features=5)
        output =exp.as_list()
        for line in output:
            for char in line[:1]:
                key = str(char)
                vocab[key.lower()]+=1
            for number in line[:2]:
                value = number

            if key in lime_output_dict:
                lime_output_dict[key] = lime_output_dict[key] + value
            else:
                lime_output_dict[key] = value
    print("Explanations stages completed : %3.d  /%3.d"%(batch_count, number_of_loops))
    index_addition = index_addition +100

if index_extra > 0:
    print("Finishing explanations")
    for index in range(index_extra):
        idx = index+(data_length-index_extra)
        explainer = LimeTextExplainer(class_names=categories)
        exp = explainer.explain_instance(testing_data[idx], pipeline.predict_proba, num_features=5)
        output =exp.as_list()
        for line in output:
            for char in line[:1]:
                key = str(char)
                vocab[key.lower()]+=1
            for number in line[:2]:
                value = number

            if key in lime_output_dict:
                lime_output_dict[key] = lime_output_dict[key] + value
            else:
                lime_output_dict[key] = value
print("Explanations finished!")
```

***Code snippet 15**. Generate LIME predictions across dataset function.*

Deep learning implementation

TensorFlow

At the time implementing the deep learning model, TensorFlow had little to no tutorials covering text classification. Any text-related tutorial only implemented the word2vec method for sentence sequencing. So, it was a struggle but to start work on text classification. However, after much perseverance, I was able to create a working mid-level API implementation. (The TensorFlow code can be found in the other code section of this report). The problems then came with the trying to add the LIME explanations to the model, as LIME didn't work when using tensors and I did not have enough experience in using TensorFlow to adapt the program accordingly. Therefore, it was decided that, a new approach was to be found. After further research I came across Keras, which was much more suited for text classification and provided clearer documentation.

Therefore, I had lost a considerable of time with no useful results to show for it.

Keras

The Keras implementation used the same data input functions as described in the machine

Model definition

```
#Model definition
def LSTM_model(max_features):
    model = Sequential()
    model.add(Embedding(input_dim =total_words+1,weights=[embedding_matrix],input_length =maxlen, output_dim=100))
    model.add(CuDNNLSTM(100))
    model.add(Dropout(0.2))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss='binary_crossentropy', optimizer='Nadam', metrics=['accuracy'])

    return model
```

***Code snippet 17**. Current LSTM architecture.*

```
model = KerasClassifier(build_fn=LSTM_model,callbacks=[history_logging,early_stop],
                        epochs=40, batch_size=100,validation_split=0.1,max_features=max_features)
```

***Code snippet 167**. Keras - Sci learn wrapper*

Keras provides a high-level Sequential model, which allows for neural network layers to be stacked linearly. The input of the model is an embedding layer, which feeds into the LSTM. The LSTM then has a dropout of 20%. Finally, the dense layer is the output layer of this model, a sigmoid function is used to map the output. (code snippet 16). This LSTM_model is then fed into a KerasClassifier function which acts a wrapper ready for input into sci-kit learn in order to generate predictions.

This is an example out of how a tweet would be represented before it enters the embedding layer(a reduction in the number of 0's compared to a one hot-coding approach)

```
[[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
    0   0   0   0   0   0   0   0 806  29 246 247 105   1 472  25  85 807
  473 333  20  52 474 105  25  79 475 116   1 334 808   3]]
```

***Code snippet 18**. Example of embedded tweet with sequence padding.*

```
Train on 1244 samples, validate on 312 samples
Epoch 1/100
1244/1244 [==============================] - 12s 9ms/step
Epoch 2/100
1244/1244 [==============================] - 8s 7ms/step
Epoch 3/100
1244/1244 [==============================] - 8s 7ms/step
Epoch 4/100
1244/1244 [==============================] - 8s 7ms/step
Epoch 5/100
1244/1244 [==============================] - 8s 7ms/step
```

```
Train on 1244 samples, validate on 312 samples
Epoch 1/100
1244/1244 [==============================] - 67s 54ms/step
0e+00
Epoch 2/100
1244/1244 [==============================] - 63s 51ms/step
Epoch 3/100
1244/1244 [==============================] - 67s 54ms/step
Epoch 4/100
1244/1244 [==============================] - 68s 55ms/step
Epoch 5/100
1244/1244 [==============================] - 65s 52ms/step
```

***Code snippet 19.** GPU (left) vs CPU (right) training performance.*

Code snippet 19 gives the training time for each cycle of the LSTM. The left uses a CuDNN-variation which runs its calculations on the GPU compared to a standard LSTM cell which is run on a CPU

Problems I had to overcome:

Installation of TensorFlow caused more hassle than expected

Keras model cross validation. During the final stages of implementation, when it came down to evaluate the model, there was a frequent error with trying to get the sci-kit learn cross validation function to run, so instead I wrote my own function to compute the scores.

Results

Confusion Matrices

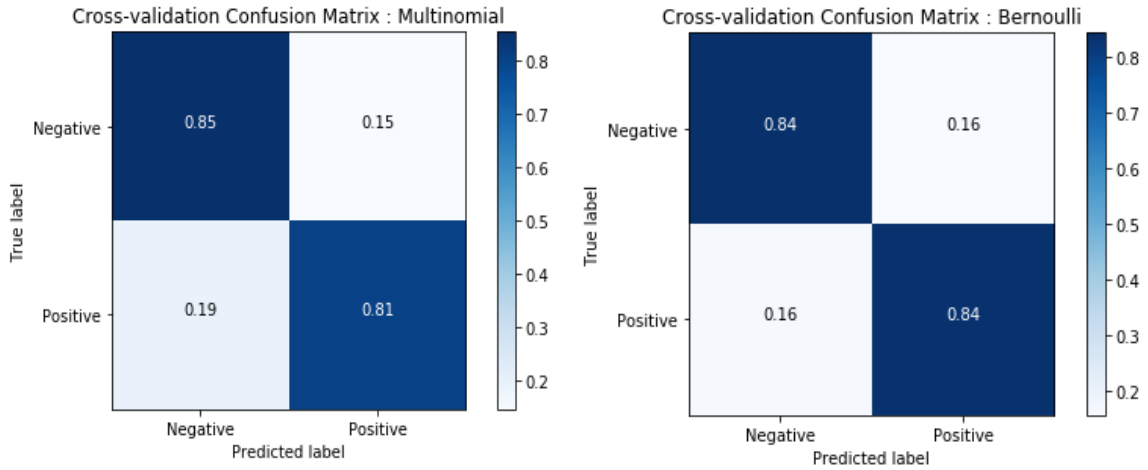


Figure 8. *Multinomial(left) & Bernoulli(right) confusion matrix's.*

(Figure 8) Showcases the cross-validated confusion matrixes of the Bayes methods. As we can see there is little difference between the two implementations and are within +/- 3% of each other. Therefore, from these results alone we cannot determine which Bayes method is preferable. Both methods when comparing against the LSTM implementation (figure 9), provide preferable results, with a improvement of around 2%. I am however, sceptical of this. I found that during the training of the LSTM, that the performance values would fluctuate around +/-5% of the values shown. Which, I am not currently able to explain why, and recommend that this needs to be investigated further, as this subsequently has a large impact on every time the LIME predictions are generated.

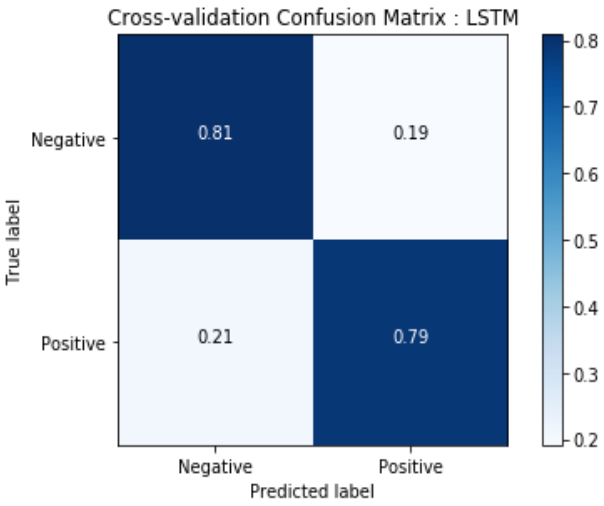


Figure 9. *LSTM confusion matrix*

Overall metric score

Model Name	Accuracy (%)	Precision (%)	Recall (%)
Multinomial Bayes	82.2	80.0	85.6
Bernoulli Bayes	83.2	82.9	83.8
LSTM	79.7	81.5	79.1

The overall scores produced from this set of cross-validated models. With each Bayes model outperforming the LSTM implementation, which was an expected outcome given the confusion matrix results.

LIME results

Here below are the outputted results from LIME. Further graphical models are available the notebooks of each module in the submission of code.

word	score
response	0.621570
preachers	0.582133
post	0.478609
shouted	0.467942
juventusfc	0.466946
filled	0.462070
chose	0.447855
human	0.427572
totally	0.415460
package	0.410254
finsburyparkattack	0.404440
osborne	0.393717
channels	0.388762
kind	0.299190
available	0.294359
assembly	0.288907
jihadis	0.281886
protect	0.278407
nonmuslims	0.275021
murder	0.274485
barriers	0.268495
bombers	0.267573
update	0.267065
throat	0.233255
fucking	0.228522

word	score
channels	0.387385
human	0.347720
osborne	0.330677
juventusfc	0.325568
bomb	0.319899
abedi	0.295703
response	0.291147
jihadis	0.276745
fucking	0.274714
salman	0.258092
update	0.241998
minutes	0.237158
fake	0.234777
preachers	0.230289
threat	0.226785
crowd	0.220926
londonattacks	0.210999
anti	0.210066
filled	0.206740
person	0.205664
look	0.204121
kind	0.202188
known	0.198406
attacker	0.198074
follow	0.196750

word	score
islamic	0.512617
pray	0.451559
mourinho	0.431842
antifascist	0.430499
ever	0.423358
response	0.422135
happened	0.412953
known	0.396819
victims	0.385180
tears	0.383984
police	0.361589
world	0.356434
sarcasticscot	0.344775
ultras	0.344302
level	0.344263
arson	0.339466
links	0.318279
pictwittercomozdmhulny	0.315143
reported	0.303142
fucking	0.302583
another	0.296246
breaking	0.293684
abedi	0.293345
unfortunately	0.293290
theresa	0.291627

Figure 10. *Words most likely to cause an interesting classification. Multinomial Bayes (left), Bernoulli Bayes (middle), LSTM (right)*

word	score
shared	-0.503478
meeting	-0.500258
responded	-0.485739
estã	-0.468588
united	-0.464917
arrest	-0.417753
dã	-0.381376
protected	-0.379393
evil	-0.355545
strategy	-0.354720
clean	-0.352605
cheap	-0.329237
perpetrator	-0.324357
hole	-0.315376
frost	-0.314830
music	-0.311446
club	-0.305349
stabbed	-0.301097
missed	-0.300960
latest	-0.265309
whitehall	-0.251304
httâ	-0.226482
imam	-0.221197
ohnikkers	-0.220019
message	-0.217958

word	score
united	-0.499441
imam	-0.280034
httâ	-0.279363
estã	-0.261777
prisonplanet	-0.254706
arrest	-0.251276
latest	-0.249576
victim	-0.223216
message	-0.220089
protected	-0.220006
mentally	-0.202774
club	-0.198489
evil	-0.197090
hero	-0.196936
shared	-0.194647
weeks	-0.193663
westminster	-0.190808
dã	-0.188915
hell	-0.185167
bbcbreaking	-0.175654
video	-0.175627
love	-0.175468
perpetrator	-0.163256
meeting	-0.163031
music	-0.162832

word	score
westminster	-0.512273
dont	-0.444591
heading	-0.349068
affected	-0.320403
leftists	-0.280707
sure	-0.265491
hearing	-0.261183
feed	-0.255284
tommy	-0.240438
town	-0.240026
views	-0.239450
corbyn	-0.234365
person	-0.231397
following	-0.226921
jkrowling	-0.222134
carry	-0.219511
hero	-0.212288
know	-0.208217
accident	-0.205215
tragedy	-0.188934
different	-0.187065
ðÿ	-0.187057
sources	-0.186818
newcastle	-0.185776
today	-0.179680

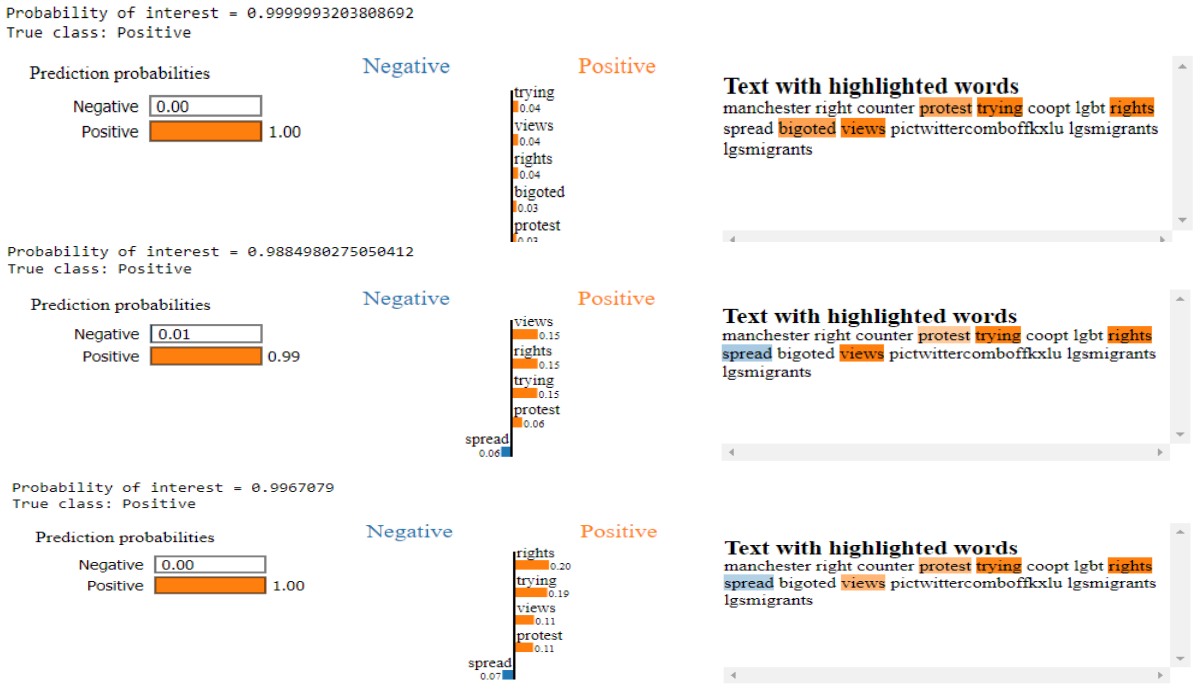
***Figure 112**. Words least likely to cause an interesting classification. Multinomial Bayes (left), Bernoulli Bayes (middle), LSTM (right)*

After showcasing these predictions to security analysts at the Cardiff University Crime and Research Centre, the consensus was that the LTSM showed better positive results, as it had not learnt the 'event names' for an interesting classification like the Bayes classifiers had. As both Bayes classifier have a high scoring of 'juventusfc' relating to the football match that occurred in Cardiff. It was also notable to see that positive classification of the pictwitter link, signifying there is more work to be done on the pre-processing of the text before model training.

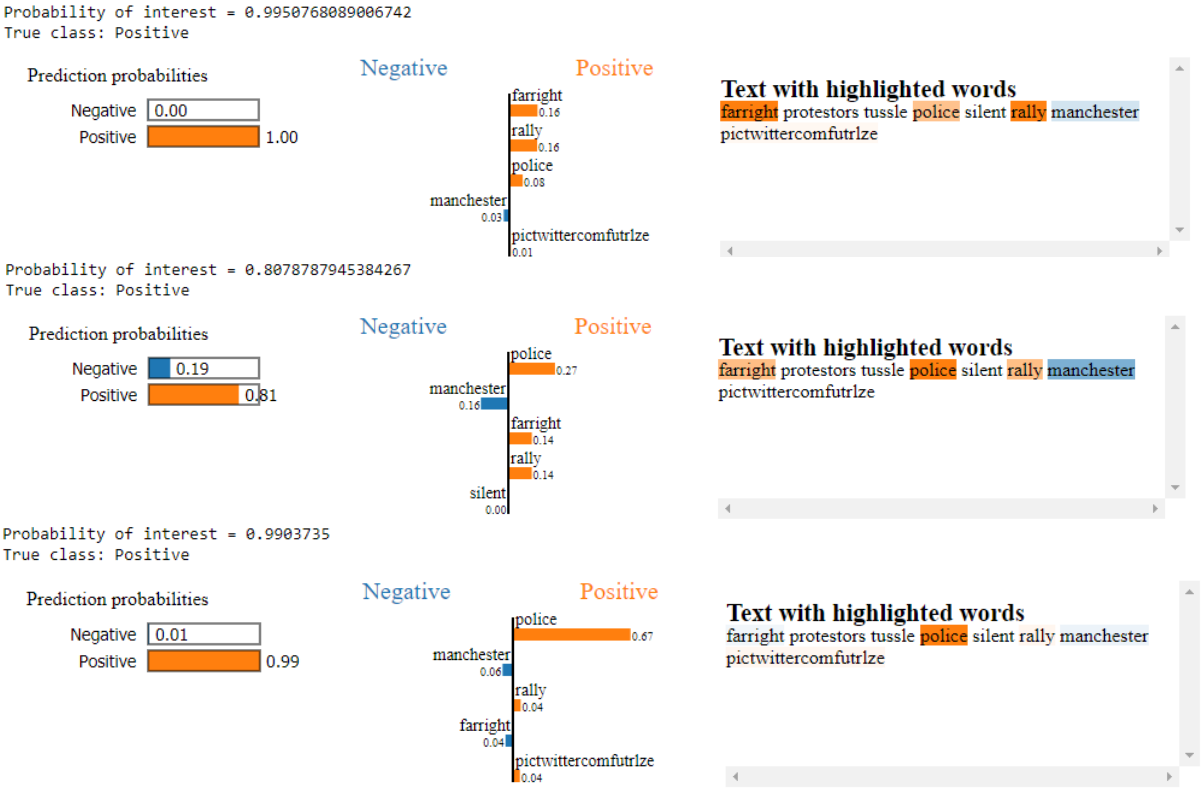
For the negative set, it was the Bernoulli implementation that was preferred. Showing that a possible combination of models might be suitable for the best in world results.

The most uninteresting word lists contain a lot of words that we might consider to be interesting, given the general context of a conversation. I believe the reason for this is that many of the uninteresting tweets are related to the events that surround the interesting tweets. Meaning that they maybe about the same topic, but are not marked as interesting as this information may already be captured by other tweets or sources of social media.

The two explanation figures below showcase a side by side comparison of how each classifier has classified the tweet. Along with how much each word was adding to that prediction.



***Explanation 1**. Multinomial (top), Bernoulli (middle), LSTM (bottom). For index 14.*



***Explanation 2.** Multinomial (top), Bernoulli (middle), LSTM (bottom). For index 112*

Future work

From this project there are several avenues in which we believe have value in exploring. In the future, we would be interested in looking at the addition of a conversational interface to with the LIME explanations function, which was originally proposed as the third aim in this project. This conversational interface could have a noticeable impact on operations at the Cardiff University Crime and Research Centre, as this could be set up during the planned monitoring of an event. Allowing an analyst to verbally question a classification while performing other tasks.

Measures which would improve the current classifier performance are also considered valuable. The improvement of the text processing function so that it can more reliably remove unwanted objects from the dataset. The addition of a twitter accounts username when making a classification could lead to greater insights and better classifications. Throughout this project, many sources of interesting data samples have come from twitter accounts of high profile individuals or government agencies. Both of which could lead to a noticeable improvement in classifiers performance matrix and explanation output. It would be useful to try and experiment with the process of model checkpointing. It was achieved too late on in the project to be able test it out fully, but it may be a solution to the LSTM's fluctuating performance metric problems.

We also believe that further research into other forms of neural networks to implement could provide value. As during the ongoing research involved with this project led to the discovery of CNN implementations claim to have improved on the performance of a basic LSTM without the costs of high computational time. [18] [32]

Conclusion

The aim of this project was to develop two text-classifiers which would be able to explain reasoning behind a tweet's classification into one of two categories through an Alexa based, conversational interface. This aim however, was not fully achieved. Two out of the three aims initially set out in the project plan have been achieved. The first is the successful creation of two classifiers which can classify tweets into one of two categories with a good level of success. The second, is the ability for these classifiers to explain the reasoning behind each classification, demonstrating to an analyst if the classifier can be trusted to make correct future predictions. The third aim of the Alexa based interface has not been achieved in this project but is considered in the future works section.

The methodology in which the results have been collected in this project is to be relied upon. However, the fluctuation in performance of the LSTM means that comparison between the two models gives a good oversight but, should not be considered accurate without further investigation into the cause of the fluctuating metrics.

Reflective points

In reflection, I wish that I had allowed more time to write this report. Even though I had stuck to my initial plan and given myself three weeks to write this, I still do not believe it has been enough. I have grossly underestimated the amount of time it would take to write, I therefore feel that I have not been able to fully showcase my knowledge surrounding this project. Particularly in the results and conclusion section of this report, as I had generated a variety of data visualisations that would enhance the reading of the report, but I unfortunately have run out of time.

I would also like to reflect on the knowledge that I have gained, before starting this project I had no previous knowledge of any of the topics it has covered, everything in this report has been self-taught which I am very proud of. The formation of knowledge in topics related to machine learning and deep learning will provide invaluable to me in my future career. The challenge of learning a difficult topic and one which can be related to state of the art technology has been a real pleasure.

I have also learnt that things will inevitability go wrong in projects like these, and that I should plan in case failure does occur.

Table of Abbreviations

Abbreviation	Meaning
GPU	Graphics Processing Unit
GDPR	General Data Protection Regulation
SVM	Support Vector Machines
MLP	Multilayer Perceptron
ReLU	Rectified Linear Unit
ADAM	Adaptive Moment Estimation
RNN	Recurrent Neural Network
CNN	Convolutional Neural Network
LSTM	Long-term Short Memory
NLP	Natural Language Processing
CPU	Central Processing Unit
BoW	Bag of Words
TF-IDF	Term frequency-inverse document frequency

References

[1] S. S. C. G. Marco Tulio Ribeiro, “"Why Should I Trust You?": Explaining the Predictions of Any Classifier,” *{Proceedings of the 22nd {ACM} {SIGKDD} International Conference on Knowledge Discovery and Data Mining,* pp. 1135-1144, 2016.

[2] D. . Reinsel, J. . Gantz and J. . Rydning, “Data Age 2025: The Evolution of Data to Life-Critical,” , . [Online]. Available: https://www.seagate.com/files/www-content/our-story/trends/files/Seagate-WP-DataAge2025-March-2017.pdf. [Accessed 5 5 2018].

[3] NVIDIA, “NVIDIA Accerlerated Computing,” [Online]. Available: https://developer.nvidia.com/cuda-downloads. [Accessed 01 May 2018].

[4] K. Krewell, “What’s the Difference Between a CPU and a GPU?,” NVIDIA, [Online]. Available: https://blogs.nvidia.com/blog/2009/12/16/whats-the-difference-between-a-cpu-and-a-gpu/. [Accessed 01 May 2018].

[5] L. . Collingwood, ““Trust in the machine: the case of Autonomous vehicles”,” , 2018. [Online]. Available: https://journals.winchesteruniversitypress.org/index.php/jirpp/article/view/43. [Accessed 8 5 2018].

[6] “GDPR - Art.15,” [Online]. Available: http://www.privacy-regulation.eu/en/article-15-right-of-access-by-the-data-subject-GDPR.htm. [Accessed 5 5 2018].

[7] T. M. Mitchell, Machine Learning, ed., vol. , , : The Mc-Graw-Hill Companies, Inc., 1997, p. .

[8] “Stanford Library,” [Online]. Available: https://stanford.library.sydney.edu.au/archives/win2015/entries/bayes-theorem/. [Accessed 9 May 2018].

[9] A. M. a. K. Nigam, “A comparison of event models for naive Bayes text classification.,” *Proc. AAAI/ICML-98 Workshop on Learning for Text Categorization,* pp. 41-48, 1998.

[10] E. LeDell, “Github,” [Online]. Available: https://github.com/ledell/sldm4-h2o/blob/master/sldm4-deeplearning-h2o.Rmd. [Accessed 4 May 2018].

[11] A. S. V, “Understanding Activation Functions in Neural Networks,” 2017.

[12] N. Qian, “On the momentum term in gradient descent learning algorithms,” *Neural Networks : The Official Journal of the International Neural Network Society,,* p. 145–151, 1999.

[13] J. H. E. &. S. Y. Duchi, “Adaptive Subgradient Methods for Online Learning and Stochastic Optimization,” *Journal of Machine Learning Research,* p. 2121–2159, 2012.

[14] D. P. Kingma and J. . Ba, “Adam: A Method for Stochastic Optimization,” *arXiv: Learning,* vol. , no. , p. , 2015.

[15] M. D. Zeiler, “ADADELTA: An Adaptive Learning Rate Method,” 2012.

[16] C. Olah, “http://colah.github.io,” [Online]. Available: http://colah.github.io/posts/2015-08-Understanding-LSTMs/. [Accessed 01 May 2018].

[17] P. S. a. P. F. Y. Bengio, “Learning long-term dependencies with gradient descent is difficult.,” *IEEE Transactions on Neural Networks,* vol. 5, no. 2, pp. 157-166, 1994.

[18] C. G. K. C. Y. B. Junyoung Chung, “Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling,” *NIPS 2014 Deep Learning and Representation Learning Workshop,* 2014.

[19] S. . Hochreiter and J. . Schmidhuber, “Long short-term memory,” *Neural Computation,* vol. 9, no. 8, p. , 1997.

[20] Jupyter.org, “Machine Learning and Jupyter,” [Online]. Available: https://nbviewer.jupyter.org/github/rhiever/Data-Analysis-and-Machine-Learning-Projects/blob/master/example-data-science-notebook/Example%2520Machine%2520Learning%2520Notebook.ipynb. [Accessed 13 03 2018].

[21] J. . Pennington, R. . Socher and C. D. Manning, “GloVe: Global Vectors for Word Representation,” , . [Online]. Available: http://www-nlp.stanford.edu/pubs/glove.pdf. [Accessed 4 5 2018].

[22] F. . Pedregosa, G. . Varoquaux, A. . Gramfort, V. . Michel, B. . Thirion, O. . Grisel, M. . Blondel, P. . Prettenhofer, R. . Weiss, V. . Dubourg, J. . Vanderplas, A. . Passos and D. . Cournapeau, “Scikit-learn: Machine Learning in Python,” *Journal of Machine Learning Research,* vol. 12, no. , p. 2825–2830, 2011.

[23] J. . Dean, R. . Monga and S. . Ghemawat, “TensorFlow: Large-scale machine learning on heterogeneous systems,” , . [Online]. Available: http://download.tensorflow.org/paper/whitepaper2015.pdf. [Accessed 9 5 2018].

[24] F. a. o. Chollet, *Keras,* 2015.

[25] NVIDIA, “NVIDIA Data Science,” [Online]. Available: http://www.nvidia.com/object/data-science-analytics-database.html. [Accessed 01 May 2018].

[26] NVIDIA cuDNN, “NVIDIA DEVELOPER,” [Online]. Available: https://developer.nvidia.com/cudnn. [Accessed 01 May 2018].

[27] T. Joachims, “A Probabilistic Analysis of the Rocchio Algorithm with TFIDF for Text Categorization,” 1996.

[28] scikit-learn developers , “scikit-learn.org,” [Online]. Available: http://scikit-learn.org/stable/datasets/twenty_newsgroups.html. [Accessed 06 May 2017].

[29] X. Q. X. H. Pengfei Liu, “Recurrent Neural Network for Text Classification with Multi-Task Learning,” 2016.

[30] J. L. B. Diederik P. Kingma, “Adam: A Method for Stochastic Optimization,” in *3rd International Conference for Learning Representations*, San Diego, 2015.

[31] K. H. Esbensen, “Principles of Proper Validation: use and abuse of re-sampling for validation,” *Special Issue: Conferentia Chemometrica,* vol. 24, no. 3-4, pp. 168-187, 2010.

[32] A. Graves, Supervised Sequence Labelling with Recurrent Neural Networks, Springer, 2012.

[33] Z. C. Lipton, “The Mythos of Model Interpretability,” 6 March 2017. [Online]. Available: https://arxiv.org/pdf/1606.03490.pdf. [Accessed 03 02 2018].

[34] Y. B. A. C. Ian Goodfellow, Deep Learning, MIT Press, 2016.

[35] S. Ruder, “An overview of gradient descent optimisation algorithms.,” 2016.

[36] “Towards Data Science,” [Online]. Available: https://towardsdatascience.com/. [Accessed 2018 May 01].

[37] J. Brownlee, “Machine Learning Mastery,” [Online]. Available: https://machinelearningmastery.com/use-word-embedding-layers-deep-learning-keras/. [Accessed 25 April 2018].