University of Sheffield

Aspect-based Sentiment Analysis

Saif Haq

Supervisor: Prof. R.J. Gaizauskas

A report submitted in partial fulfilment of the requirements
for the degree of Computer Science

in the

Department of Computer Science

May 16, 2020

Declaration

All sentences or passages quoted in this document from other people's work have been specifically acknowledged by clear cross-referencing to author, work and page(s). Any illustrations that are not the work of the author of this report have been used with the explicit permission of the originator and are specifically acknowledged. I understand that failure to do this amounts to plagiarism and will be considered grounds for failure.

Saif Haq, May 2020.

Abstract

Aspect-based sentiment analysis (ABSA) is a subset of natural language processing with the aim of identifying opinions expressed by people in texts towards aspects of entities. ABSA is more important than ever in the age of big data where companies like Amazon store many millions of user reviews of products. These reviews are very important as they directly influence the number of sales of products. Extracting aspects of which opinions are expressed about; and their polarity can help revolutionise the product-lifecycle, innovating based on user feedback at a rapid pace. This should also in the long-run reduce customer service expenditure and improve corporate image, with the only requirement being training data. To create an ABSA system, three main tasks first need to be completed: labelling sentences with the aspects that opinions are expressed about (aspect category detection), separating sentences that are do not express an opinion (subjectivity classification) and finding polarities of opinions expressed for each aspect category identified (polarity classification). This project aims to implement sentence-level ASBA for these tasks using supervised machine learning techniques to analyse laptop reviews provided by SemEval 2016. Experiments will be conducted on the use of different input representations and a range of neural network architectures.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

E-commerce now being an industry with a revenue over $3.5 Trillion USD, and being expected to nearly double by 2023 to more than $6.5 Trillion Winkler (2019); analysis of client reviews is at the heart of growth within this industry. Taking a look at Amazon, a company who have amassed an e-commerce market share just shy of 50% in 2019 Statista (2019), analysing customer reviews has become an industry within its own right due to the impact that positive reviews have on sales conversion rates. In Amazon's CEO's own words:

'The most important single thing is to focus obsessively on the customer. Our goal is to be earth's most customer-centric company' – Bezos (2013)

A system that analyses client reviews is beneficial to both businesses and customers. Businesses can innovate the product-lifecycle from information gathered by the ABSA system, and customers get a better insight into what they are purchasing before they do, reducing the information failure within the market. With over thousands of microprocessors in some consumer-level GPUs in 2020[1], adoption of machine learning has become easy and necessary in most industries. Machine learning can be applied to ABSA, with the goal being to extract categories of which polar opinions are being expressed about in a text.

1.2 Aims and Objectives

The aim of this project is to design, implement and evaluate an ABSA system that makes use of supervised machine learning techniques applied to SemEval's 2016 Laptops dataset. Feature representation will be investigated, specifically by changing how text is represented in the input layer of a neural network via an embedding layer. Different architectures will also be investigated, including the popular CNN and LSTM

[1] `https://www.geforce.com/hardware/desktop-gpus/geforce-gtx-1080-ti/specifications`

architectures which have had many applications in image classification problems over recent years. This project will also implement a RandomSearch algorithm to explore the hyper-parameter search space of one of the models to see how it performs and whether the model sees an increase in performance as a result of tuned hyperparameters.
The models developed will assign labels to a sentence with the aspect categories an opinion is expressed about, and the polarity of these opinions, if one exists, will be determined. The tasks involved in this system are aspect category detection, subjectivity classification and polarity classification. As systems developed are not lexicon based, the model architectures implemented should theoretically be able to applied to any language domain.

1.3 Organisation

This report is structured as follows:

- **Chapter 2** begins with an overview of SemEval 2016 Task 5 before presenting several state-of-the-art projects in ABSA; explaining the underlying deep learning concepts and impacts of various architecture choices.
- **Chapter 3** formalizes the requirements of the project and how models will be evaluated. The SemEval laptops dataset is also further analysed.
- **Chapter 4** presents the design of the system for different tasks, providing specific details of choices made for each architecture and the reasoning behind these choices.
- **Chapter 5** presents the implemented architectures for different tasks, and some basic testing.
- **Chapter 6** provides a detailed report of the results for each of the implemented models, and evaluation including a discussion of the findings.
- **Chapter 7** concludes the report with a summary of the project and suggests possible future work.

Chapter 2

Literature Survey

2.1 Aspect-Based Sentiment Analysis

Liu (2012) published a paper studying aspect-based sentiment analysis. The following definitions are fundamental to this field.

Definition: Entity

An entity e is a product, service, person, event, organization, or topic. It is associated with a pair, $e : (T, W)$, where T is a hierarchy of components, sub-components, and so on, and W is a set of attributes of e. Each component or sub-component also has its own set of attributes.

Definition: Opinion

An opinion is a quintuple, $(e_i, a_{ij}, oo_{ijkl}, h_k, t_l)$, where e_i is the name of an entity, a_{ij} is an aspect of e_i, oo_{ijkl} is the sentiment on aspect a_{ij}, h_k is the opinion holder, and t_l is the time when the opinion is expressed by h_k. The sentiment oo_{ijkl} can have positive, negative or neutral polarity with different strength levels.

The goal of aspect-based sentiment analysis is to produce all opinion quintuples present in an opinionated document D.

2.2 SemEval

Semantic **Eval**uation is an ongoing series of evaluations of computational semantic analysis systems, that aims to explore different natural language processing technologies through conferences, workshops and competitions. Task 5 of SemEval's annual workshop, originally hosted in 2016 aims to tackle aspect based sentiment analysis, specifically in the domain of customer reviews. They provide data for two domains of customer reviews for the English language: laptop reviews and restaurant reviews. This project will focus on their laptop reviews datasets (Pontiki et al. 2016).

2.2.1 SemEval 2016 Task 5 Subtasks

Teams entering in the competition could opt to enter with the constrained condition, restricting data used for developing models to the SemEval datasets exclusively, comparative to the unconstrained condition where teams could use other resources such as publicly available lexica.

Subtask 1 is concerned with sentence level ABSA, with the task being to identify all opinion tuples about a target entity given an opinionated text. This is split into three slots.

- Slot 1: Aspect Category Detection: Return entity E and attribute A pairs towards which an opinion is expressed in the given text.
- Slot 2: Opinion Target Expression: Extract opinion target expressions, which is defined by starting and ending offsets of the linguistic expression used to describe the reviewed entity, for each E#A pair.
- Slot 3: Sentiment Polarity: Assigning each E#A pair a polarity label, being either neutral, positive or negative

Subtask 2 is to return all of the E#A pairs given a set of customer reviews about a target entity.

Subtask 3 is to perform aspect-based sentiment analysis with systems created on previous tasks on an unseen domain for which no training data will be made available.

For the laptops dataset, the data provided only allows completion of slots 1 and 2 of subtask 1, so these tasks will remain the focus of this project. Table 2.2 shows an overview of the laptops dataset for subtask 1.

	Sentences	Tuples
Training dataset	2500	2909
Testing dataset	808	801

Table 2.1: *Overview of SemEval 2016 Task 5, Subtask 1, English laptops datasets*

The datasets are made available publicly on METASHARE as XML files under an academic/non-commercial license.

2.2.2 Participation and Results

SemEval's task 5 attracted 245 submissions from 29 teams across all subtasks and languages, with 22 teams participating with subtask 1 using the laptops dataset. Each team participating could submit two runs per slot, with one being constrained and the other non-constrained.

	Slot 1 (F1)	Slot 3 (Acc)
Highest Score	51.937	82.772
Baseline Score	43.855	71.660

Table 2.2: *Results comparing performance of models for SemEval 2016 Task 5, Subtask 1, English laptops dataset*

2.3 Techniques for Aspect-Based Sentiment Analysis

2.3.1 Aspect Category Detection

Aspect category detection is concerned with extracting aspects from a sentence s which can be explicitly or implicitly defined.

"This computer is really fast " $\rightarrow$ (speed)
Is an example of explicit aspect expression as computer is explicitly defined.

"It is overpriced and then it dies quickly " $\rightarrow$ (price, battery life)
Is an example of an implicit aspect expression as the the laptop is not explicitly defined.

Aspect category detection becomes very difficult for opinions defined implicitly. Realizing what 'it' is referring to requires a lot of context that humans have through experience but a machine learning classifier would not. Opinions defined explicitly are much easier to classify, as less context is needed. A common method to classify explicitly defined opinions is to take n-grams of feature vectors. N-grams are a selection of parts of sequences of textual data or feature vectors, that can extract meaningful features from the original sequence (Fürnkranz 1998).

Toh & Su (2016) implemented a deep learning technique via a single feed-forward neural network and a convolutional neural network. They used word embeddings as feature representations for first layer of the network, and tested models with Head Word (stemming), conflating morphological variants of words to their base form, and

Word Cluster, which further processed word embeddings. The architecture of their system concatenates the different feature representations to be fed to a convolutional layer with max pooling then a softmax output layer, giving a probability distribution. Samples over some threshold t are considered correct. Their model with all listed features achieved the best results for Slot 1 of the SemEval 2016 Task 5 SB1 at the time of the competition.

2.3.2 Subjectivity Classification

As a precursor to polarity classification, it is often useful to perform subjectivity classification with the goal of distinguishing whether a sentence s expresses sentiment, or is objective. Sentiment and subjectivity are terms that can be used interchangeably.

2.3.3 Polarity Classification

The goal of polarity classification is to classify a subjective sample as either positive or negative given sentence text and the class labels.
Ruder et al. (2016) shows the application of a recurrent neural network applied to a 2-way polarity classification task. Their model uses two Bi-directional LSTMs, which propagate inputs over time between the 300-dimensional embedding layer with GloVe weights and the output layer, which has a softmax activation function. The network is regulated by a dropout layer of 0.5 directly after the LSTM layer (Section 2.4.8).

2.4 Machine Learning for Sentiment Analysis

Creating a sentiment analysis system will require the implementation of one or more machine learning learning algorithms. There are vast amount of choices when choosing a machine learning algorithm, but in their essence, algorithms boil down to one of four main categories. The four categories are supervised, semi-supervised, unsupervised and reinforcement learning. These techniques are described below (Burkov 2019):

Supervised Learning In supervised learning, the dataset is the collection of labelled examples $(x_i, y_i)_{i=1}^{N}$. For each element x_i among N is called a feature vector. If each example x represents a person, the first feature $x^{(1)}$ could represent their weight in kg, the second feature $x^{(2)}$ could represent their height in cm, etc. The label y_i can be an element belonging to a finite set of classes, Ie. for descriptions of a laptop, the classes could be $(BatteryPerformance, Usability, ...)$; or labels can be a more complex structure such as a vector, matrix or tree. The goal of a supervised learning algorithm is to use the dataset to produce a model that takes a feature vector x as input and ouputs the information that allows deducing the label for this feature vector.

Unsupervised Learning In unsupervised learning, the dataset is collection of unlabelled examples $(x_i)_{i=1}^N$ where x is a feature vector. The goal of a supervised learning algorithm is to use the dataset to produce a model that takes a feature vector x as input and ouputs a transformation of that feature vector that can be used to solve a problem. Clustering is an example that outputs the id of the cluster for each input x.

Semi-supervised Learning Semi-supervised learning contains both labelled and unlabelled examples, usually with a higher quantity of unlabelled examples. The goal of a semi-supervised learning algorithm is the same as supervised learning.

Reinforcement Learning In reinforcement learning, the machine "lives" in an environment and perceives the state of the environment as feature vectors. Learning is done through trial and error, and a reward mechanism is employed with the goal being to learn a policy. A policy is a function that takes a feature vector as an input and ouputs an optimal action to execute that state.

With use of SemEval datasets, this project will focus on supervised machine learning. This technique has seen a lot of success in the domain of aspect-based sentiment analysis, especially with the use of deep learning models for classification tasks such as aspect category detection and polarity classification.

2.4.1 Support Vector Machines

Support vector machines (SVMs) are a powerful machine learning technique that aims to find an optimal hyper-plane in a multi-dimensional feature space that separates classes of data using a soft margin classifier. This determines the optimal distance between observations; and and a threshold (Cortes & Vapnik 1995). A soft margin classifier works when data isn't completely linearly separable, ie. allows misclassification through the use of slack variables (Observations on the wrong side of the optimal hyper-plane). A hard margin classifier on the other hand does not work when data isn't completely linearly separable (Boser et al. 1992). The name support vector machine comes from the fact that observations on the edge and within the soft margin are called support vectors.

Support vector machines use kernel functions to systematically find support vector classifiers in higher dimensions; which have the function of calculating the relationship between every pair of observation points as if they are in the higher dimensions, but they don't actually perform the transformations. This is known as the 'kernel trick, and is the reason why SVMs are fairly computationally cheap (Awad & Khanna 2015). Two notable kernel functions are the polynomial function and radial based function. The polynomial kernel increases dimensions by setting a variable d, the degree of the polynomial; whereas the radial based function finds support vector classifiers in infinite

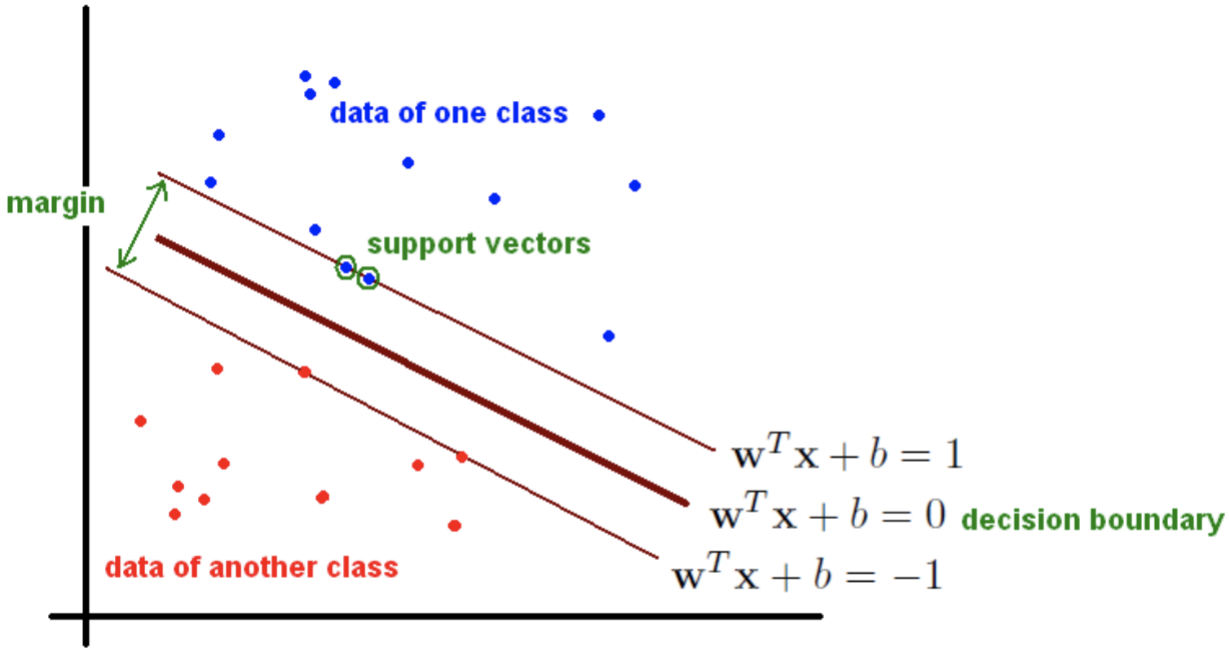


Figure 2.1: *Support Vector machine optimal margin and optimal hyper-plane (decision boundary) for a two-class classification problem in a 2-dimensional feature space*

Source: https://fderyckel.github.io/machinelearningwithr/svm.html

dimensions; behaving similar to a weighted nearest-neighbour classifier.

G & Chandrasekaran (2012) show that SVMs are one of the most competitive machine learning techniques for sentiment analysis, achieving higher performance than many other algorithms including Naïve Bayes, which have also been the findings from other comparative studies.

2.4.2 Neural Networks

The basic building block of a neural network is a neuron, commonly referred to as a node. A node is a functional unit with input U and output Y. Each node in a layer of a neural network can have a different activation function (Burkov 2019). An activation function of a node defines the output Y of that node given an input U. There are a vast variety of activation functions, each suited to different problems. Table 2.3 shows a few

Activation function	Equation
Sigmoid	$f(x) = \frac{1}{1+e^{-\beta x}}$
Softmax	$\mathrm{f}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$
TanH	$f(x) = \frac{2}{1+e^{-2x}} - 1$
ReLU	$f(x) = \begin{cases} 0, & \text{for x<0,} \\ \mathrm{x}, & \text{for x} \geq 0, \end{cases}$

Table 2.3: *Common activation functions*

common activation functions. Each connection between nodes has an arbitrary weight assigned to it. During training, these weights are constantly updated in attempts to reach their optimal value. The updating of these weights through evaluation of correct and incorrectly identified samples is called back propagation. In addition to a weight, nodes have a bias, which is a constant added to the input which helps move the output of each node up and down, similar to role b has in the linear equation 2.1.

$$y = ax + b \tag{2.1}$$

To train a neural network, a random permutation of training inputs are chosen from the training data, and they are fed into the network, calculating the neural activations and storing the result of the output layer. This is repeated until all training inputs have been exhausted, which is said to complete an epoch of training.

At the end of each epoch, a loss function is used to compute the errors for a prediction, and these are used by an optimisation algorithm to minimise loss, which improves model accuracy. The loss is the error between what the model is predicting for the training input, against the true label of the input.
Back propagation calculates the derivative of the activation functions starting at the end of the network, and pushes the error back through network towards the input, hence the name. This process updates the weights, which translates to a higher accuracy when classifying new samples. The whole process is repeated for either a fixed amount of epochs, or until the model starts converging and stops automatically through the use of early stopping.

Stochastic gradient descent (SGD) is a common optimisation algorithm. An optimisation algorithm has the aim of minimising a given loss function, by assigning the weights in a way to make this loss function as close to 0 as possible. It works by taking small steps in the direction of the steepest slope by calculating partial derivatives of the loss function; in an attempt to eventually reach the global minimum of said function (Nielsen 2015).

The most commonly used loss function is mean squared error, which calculates the square of difference between actual values and predicted values. In the following equations, y and $\hat{y}$ refer to the probability distributions given by output of the model, for actual and predicted samples.

$$MSE(y, \hat{y}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.2}$$

Mean-Squared Error isn't well suited for classification tasks, as the decision boundary (the difference in actual and predicted labels) is large and MSE does not punish misclassifications enough. It is better suited for regression tasks. For multi-class classification, Cross-Entropy loss is a better suited loss function.

$$CE(y, \hat{y}) = \sum_i y_i \log \frac{1}{\hat{y}_i} = -\sum_i y_i \log \hat{y}_i \tag{2.3}$$

As samples in SemEval's datasets for aspect category detection need to be multi-labelled, not assigned a single class out of multiple classes, an adapation of Cross-Entropy loss is required so classifications of one label do not affect others. Binary Cross-Entropy loss is the needed adaptation to Cross-Entropy to make it performant for use with multi-label classification (Ho & Wookey 2019).

$$BCE(y, \hat{y}) = \sum_i y_i \log \frac{1}{\hat{y}_i} + \frac{1}{(1 - y_i) \times \log(1 - \hat{y}_i)} \tag{2.4}$$

It is important to note what activation is in the output layer as the performance of the loss function of a model is mostly dependant on the output neuron's activation function. As cross entropy loss calculates the difference between two probability distributions, it is vital to have either a sigmoid or softmax activation in the output layer. Softmax is usually used for multi-class classification where each sample lies in one of many classes whereas sigmoid works well for multi-label classification where each sample lies in any number of the many classes.
Yang et al. (2017) describes softmax in NLP models to have a bottleneck and claims: 'Standard Softmax-based language models with distributed (output) word embeddings do not have enough capacity to model natural language.'. They introduce a technique called Mixture of Softmaxes (MoS) improving on the deficiencies of softmax by treating

natural language modeling as a matrix factorization problem.

A deep neural network can be defined as a neural network with an input layer, an output layer and at least one hidden layer. Figure: 2.2 shows a feed-forward architecture of a neural network, where each neuron feeds data in one direction, and every neuron is densely connected (connected to each neuron in the previous and following layers). Activations in one layer determine activations in the next layer. It is loosely analogous to how in biological networks of neurons, some groups of neurons firing cause other groups to fire (Hebbs Rule) (Hertz 2018). Neural networks are considered as universal function approximators meaning they can compute any function at all - activation functions allow non-linearity to exist within the network (Wang 2003).

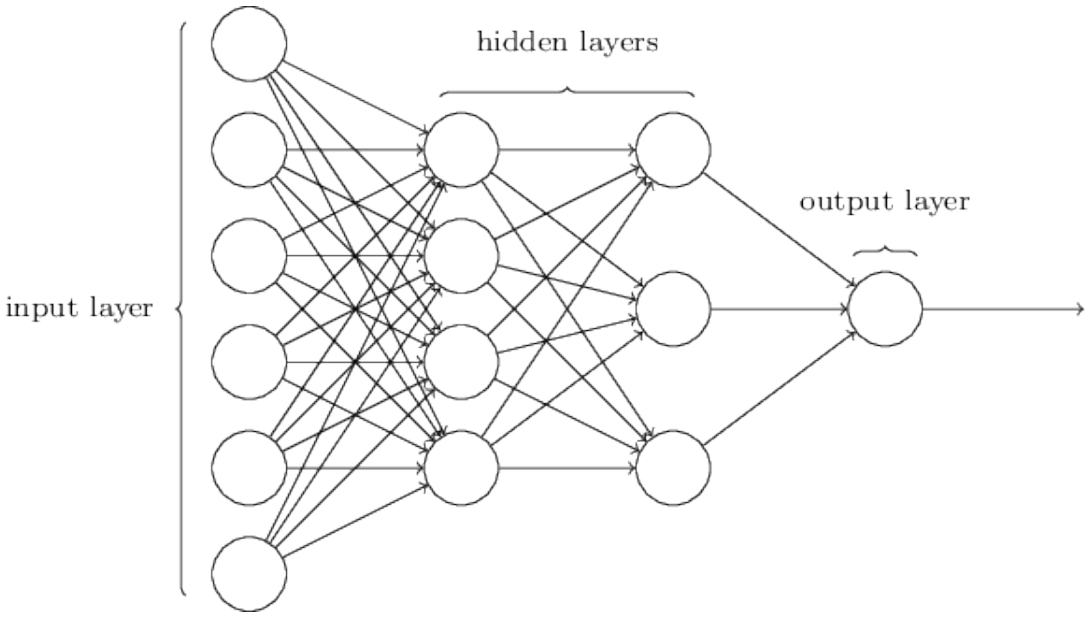


Figure 2.2: *A feedforward artificial neural network.*
Source: `http://neuralnetworksanddeeplearning.com/chap1.html`

2.4.3 Convolutional Neural Networks

Convolutional neural networks, (CNNs) have seen a lot of success in image recognition tasks in recent years. A CNN has convolutional layers that are useful where there is a translation invariance, ie. an entity in an image being able to be in different places or

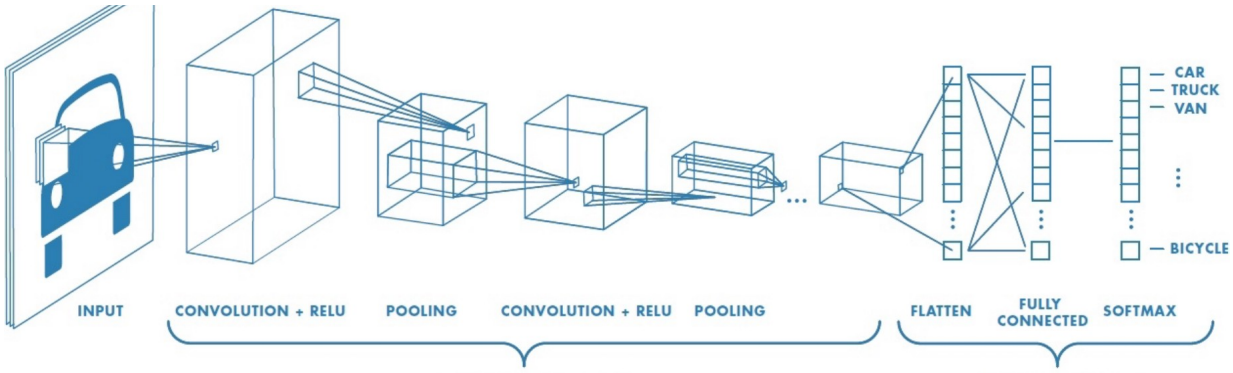


Figure 2.3: *Deep CNN visualisation for an image classification task*
Source: https://www.mathworks.com/solutions/deep-learning/convolutional-neural-network.html

words in a sentence being arranged differently. Despite being most known for image recognition, CNNs have proved powerful in other domains such as NLP. CNNs have filters which make them strong for pattern recognition. The filters detect presence of features, which could refer to edges in images near the beginning of the network, and possibly more sophisticated pattern recognition such as objects within the image; in the deeper of the convolutional layers. When a convolutional layer receives input, the filter will slide over the set of inputs, which is known as convolving. After the filter has convolved the entire input, what is left is a new representation of the input, made up of dot products of the filters with input matrices. Kim (2014) shows that CNNs perform well for sentence classification. Their model inspires ones of the models developed for this project.

2.4.4 Pooling

Pooling is an operator that is typically added to models following convolutional layers. It is possible to perform max pooling, or average pooling, with some variations including concatenations of both. A pooling operator simply performs a dimensionality reduction from the convolved input's generated representation. Max pooling takes the max of the pixel values within the filter size, whereas average pooling takes the average. A stride determines how the filter moves across the pixels until an edge is reached, then the same is repeated one stride distance below. Reducing feature vectors reduces some computational expense which can translate to noticeable time saving when training the model.

Global max pooling, max pooling applied without a stride, can be a very effective technique, even without a CNN in the prior layer. It is possible to apply global max pooling directly to an embedding layer, where it extracts the most salient features from every word embedding dimension. This is done by taking the maximum from

each dimension of the word embedding (Shen et al. 2018).

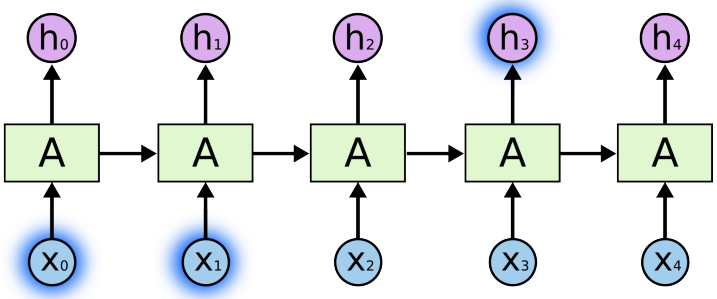


Figure 2.4: *Information flow in a Recurrent Neural Network*
Source: `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`

2.4.5 Recurrent Neural Networks

Recurrent neural networks, (RNNs) allow feedback loops between neurons, which is particularly useful for sequence data particularly temporal data, or any model where memory is desirable such as NLP tasks. The control flow of a RNN can be seen by Figure 2.4. Long short-term memory networks (LSTMs) and Gated Recurrent Units (GRUs) are popular types of RNNs that have been extensively used in the audio domain; making use of feedback loops to store memory within the network.

During back propagation, RNNs suffer from the vanishing gradient problem. This is when the gradient shrinks as it back propagates through time due to RNN's short-term memory. This happens as a result of gradient variables becoming small to the point where their contribution to learning is negligible. This occurs due to activations at different timestamps of the RNN, contributing the pushing features towards 0, which could occur in the case of a sigmoid function being the neural activation for the RNN. LSTMs have internal gates that regulate the flow of information, which combats RNNs traditional vanishing gradient problem. LSTMs can learn to keep track of important information and discard non-relevant data. This is what makes them a perfect for use in deep learning models for NLP tasks; with the LSTM being able to keep important features of word representations, and shrink features that do not have a large impact on classifying samples over a period of time stamps (Hochreiter & Schmidhuber 1997).

2.4.6 Long Short Term Memory Networks

An LSTM has the same control flow as a recurrent neural network that passes data sequentially. Figure 2.5 shows a single LSTM cell, that would be chained to other LSTMs that pass data along to the next from the main horizontal line at the top of the

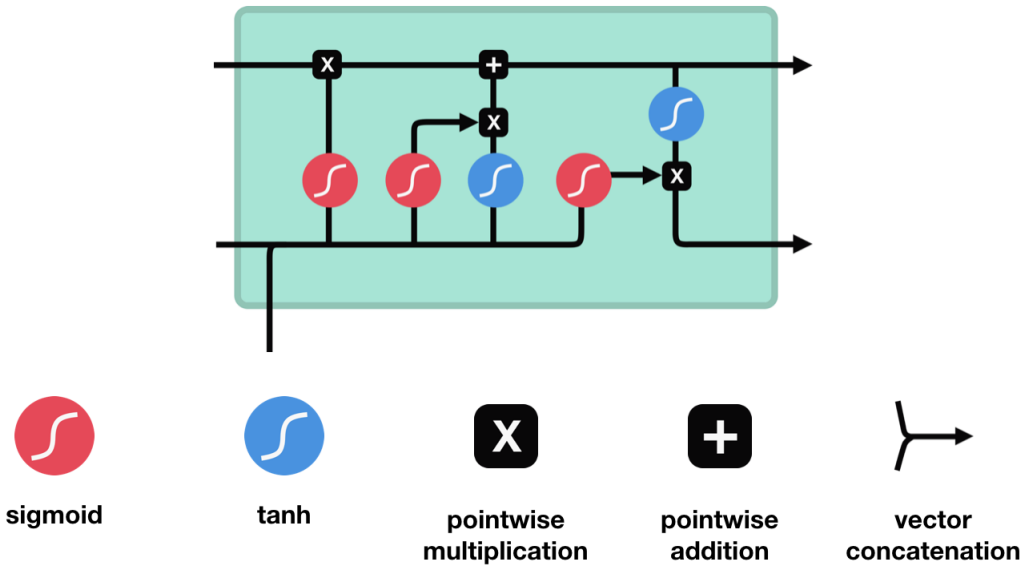
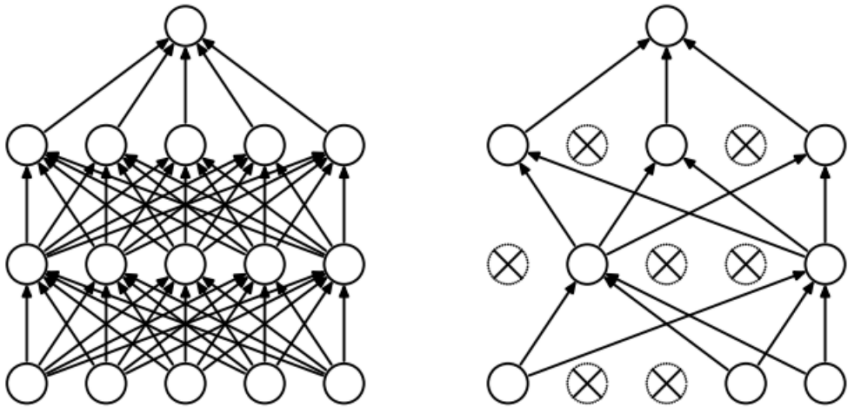


Figure 2.5: *A LSTM cell and it's operations.*
Source: `https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21`

diagram. This acts as a transport highway for relative information down the sequence chain. This is how LSTMs have a memory store. As more inputs are passed into the LSTM, information gets added or removed from the cell state via gates. Gates can learn what information to keep, and what information to discard during the training of the model. As the cell state can carry information throughout the sequence process, information can in theory be carried from the first time step to the last time step.

In a sequence of LSTM cells, a cell receives two inputs: a processed input vector (bottom), and a previous state vector (left). These two inputs are combined to make a single vector that is subsequently passed through a *tanh* function that squashes values between -1 and +1; to regulate the output of the network.
Gates within the LSTM contain *sigmoid* activation functions, which squashes data between 0 and 1. Data that is not relevant to the problem domain will converge towards 0, and data that is important will converge towards 1. Inputs closer to 0 diminish after multiple back propagations.



(a) Standard Neural Net (b) After applying dropout.

Figure 2.6: *Visualisation of dropout in the first three layers of a feed forward ANN*
Source: Srivastava et al. (2014)

2.4.7 Word Embeddings

In NLP, the representation of words in a model still remains one of the predominant focuses of the field. Mikolov et al. (2013)'s word2vec library helped word embeddings become the predominant approach for vectorizing textual data, as it has proven that it significantly outperforms other methods of vectorization such as TF-IDF which indexes words on the basis of their frequencies in documents. Word embeddings work on the assumption that words with similar meanings appear in similar contexts, and capture this relation between words in their dense vector representation.
Pennington et al. (2014)'s GloVe: Global Vectors for Word Representation, is a popular unsupervised learning algorithm, with trained embeddings packaged as a text file that provides vector representations of over 400,000 words that were trained on a 2014 Wikipedia dump. The version referenced for this project contains embeddings in the dimensions: {50, 100, 200, 300}.

2.4.8 Network Regulation

Overfitting in a neural network is a core problem in the training phase that occurs when the model fits the training data too closely, losing its ability to generalise to new data. A common prevention technique is early-stopping, whereby a percentage of the training data is used to monitor the network. Patience, p can be defined as the amount of epochs that are run with no improvement seen on the validation data when monitoring validation loss. Another prevention technique is called dropout (Hinton et al. 2012). Dropout layers are especially important when using a CNN, as convolutional layers increase the rate of convergence from feature vectors to labels. Dropout acts a

blocker to certain neurons as activations attempt to pass through. This can slow down the overfitting of the network, because not all features have an impact on the final classification for each sample; due to some neurons being blocked. Figure 2.6 shows a dropout of 0.4 in the first and penultimate layers, and a dropout of 0.6 in the second layer.

2.4.9 Hyperparameter Optimisation

A hyperparameter is an external characteristic of a model that cannot be estimated from training data. When implementing a machine learning model, hyperparameters will be encountered: such as the number of filters in a CNN, the amount of neurons in each hidden layer or even the number of layers the model has. The kernel of a SVM classifier is also a hyperparameter.

Grid search is an exhaustive method used to explore the search space of hyperparameters within a machine learning model, but is very computationally expensive as training phase of the model needs to be run once for every adjustment of each hyperparameter.
By contrast, random search works by randomly sampling parameters over a given set of values, so a computational budget can be chosen independent of the size of the search space. Figure 2.7 shows a comparison of grid search and random search optimizing a function $f(x, y) = g(x) + h(y)$ with 9 fixed trials. Random search is able explore all values of g whereas grid search is only able to test g in three distinct places (Bergstra & Bengio 2012).

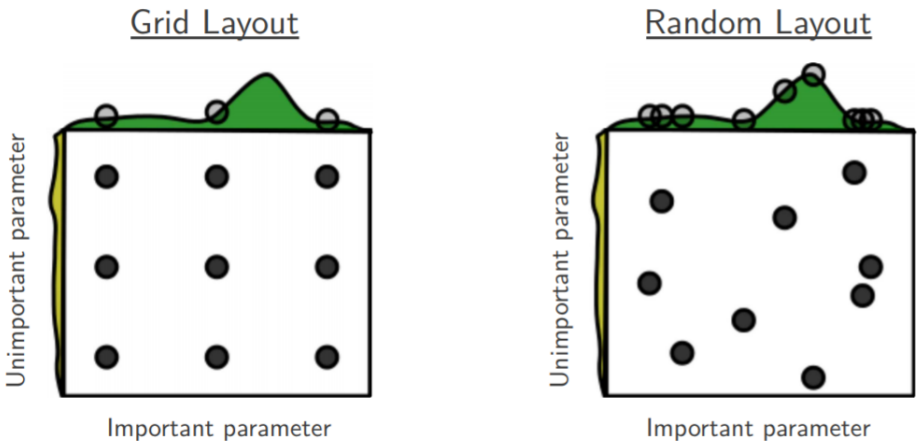


Figure 2.7: *Grid search vs Random search.*
Source: Bergstra & Bengio (2012)

2.5 Libraries and Frameworks

2.5.1 Pandas

Pandas[1] is an open source library for Python that allows data manipulation and analysis. It is well-suited to big data, and notably live data; hence its wide scale use in financial analysis applications.

2.5.2 Scikit-learn

Scikit-learn[2] is a popular open source library for Python that includes functions for various machine learning tasks, including data pre-processing and random splitting of training data into validation data. It has built in algorithms such as support vector machines, and an easy to use input pipeline. It also provides functions for model evaluation including metrics discussed in Section 3.1.2.

2.5.3 Deep learning framework

To implement a deep neural network, a deep learning framework is required. Deep learning frameworks operate on tensors and view any model as a directed acrylic graph, a finite directed graph with no directed circles.
There is much debate on which is the best deep learning framework, however, since the release of Tensorflow 2.0[3] in October 2019, it has become the clear favourite. TensorFlow is by far the most popular deep learning framework, using GitHub activity as a metric. Google created it to help scale all of its' services including Google Translate, and now it has been adopted by some of the world's largest software companies including Airbnb, Snapchat, and Uber. Having such a large community directly correlates to its' rich documentation and use in production environments. Tensorflow Serving is a high-performance, battle tested system for serving models to production, using the gRPC protocol[4] on HTTP/2 which is on its way to become the standard for microservice communication (Pai 2020). Tensorflow2.0 has since built Keras into the main distribution, which is a minimalist high-level API used to control the Tensorflow backend. TF also provides GPU support via NVidia's CUDA [5]; a parallel computing platform, allowing much faster training and evaluation of models.

[1]Pandas: `https://pandas.pydata.org/`
[2]Scikit-learn: `https://scikit-learn.org/stable/`
[3]TensorFlow: `https://www.tensorflow.org/`
[4]gRPC: `https://grpc.io/`
[5]NVidia CUDA: `https://developer.nvidia.com/cuda-toolkit`

Chapter 3

Requirements & Analysis

3.1 Requirements

SemEval 2016 Task 5 has been mentioned throughout this project, and is a starting point for the direction of the project. However, the requirements defined in Table 3.1 show the aims of this project, which mainly focus around investigating the impact of different deep learning algorithms and architectures to explore aspect based sentiment analysis. The laptops domain of SemEval is used, due to larger potential fiscal impact of developing an ABSA system for customer reviews, with the scope of the project easily being able to be extended to other groups of commercial products, whereas while a system for restaurant reviews may be interesting, it does not the same magnitude of commercial impact that a product reviews system would have.

The main components of an ABSA system are aspect category detection, subjectivity classification and polarity classification. In the context of SemEval, Slot 1 is aspect category detection, and Slot 3 of sentiment polarity has been split into subjectivity classification and polarity classification. The intention is for subjectivity classification to be a precursor to polarity classification, and if a complete end-to-end system is to be developed, neutral sentences will not reach the polarity classifier, as it only peforms two-class classification in the classes positive or negative for each sentence when given it's aspect category.
SemEval's Slot 2 regarding opinion target expression is not within the scope of this project due to the starting and ending offsets of liguistic expressions that describe entities; that are required to train an OTE system are not provided for laptops dataset.

Requirements **1-3** define the implementation and evaluation of baseline systems for the three main components. Requirements **4-6** cover aspect category detection, defining the requirement that three architectures will be implemented, and evaluated. These include a basic deep neural network, a convolutional neural network and recurrent neural network in the form of a long short-term memory network. These requirements in

addition to the requirements **8-9** will all be implemented using Keras's functional API, utilizing Tensorflow's backend.
To complete Requirement **7**, a RandomSearch algorithm will be implemented for the highest performing of the models for aspect category detection. To evaluate the generated models, the top ten models will be scrutinized to see if they show any concrete patterns of the best choice of hyperparameter.
Requirements **10-12** are concerned with investigating the impact of input representations on classifier performance, however this section of the project has the lowest priority for implementation.

	Requirement	Priority
1	Implementation & Evaluation of a baseline for aspect category detection	9
2	Implementation & Evaluation of a baseline for subjectivity classification	9
3	Implementation & Evaluation of a baseline for polarity classification	9
4	Implementation & Evaluation of a DNN model for aspect category detection	6
5	Implementation & Evaluation of a CNN model for aspect category detection	7
6	Implementation & Evaluation of a LSTM model for aspect category detection	7
7	Implementation & Testing of a hyper-parameter optimisation algorithm to improve the performance of a model	7
8	Implementation of a deep learning model for subjectivity classification	6
9	Implementation of a deep learning model for polarity classification	6
10	Investigating the impact of varying word embedding dimensionality on a classifier's performance	4
11	Investigating the impact of using a stoplist that removes stopwords on a classifier's performance	5
12	Investigating the impact of using a stoplist that replaces stopwords with an out of vocabulary token on a classifier's performance	5

Table 3.1: *Requirements of the project with priorities: 10 being high and 1 being low*

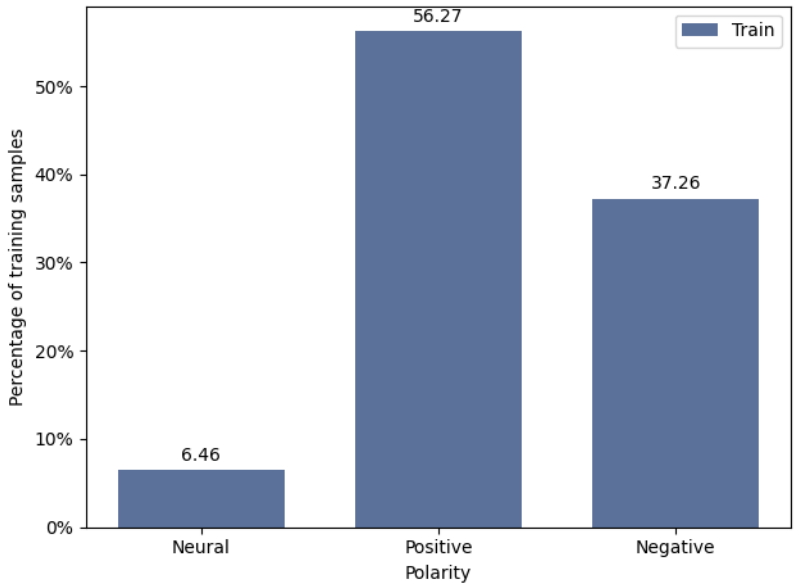


Figure 3.1: *Graph showing polarity distribution of opinions in training data*

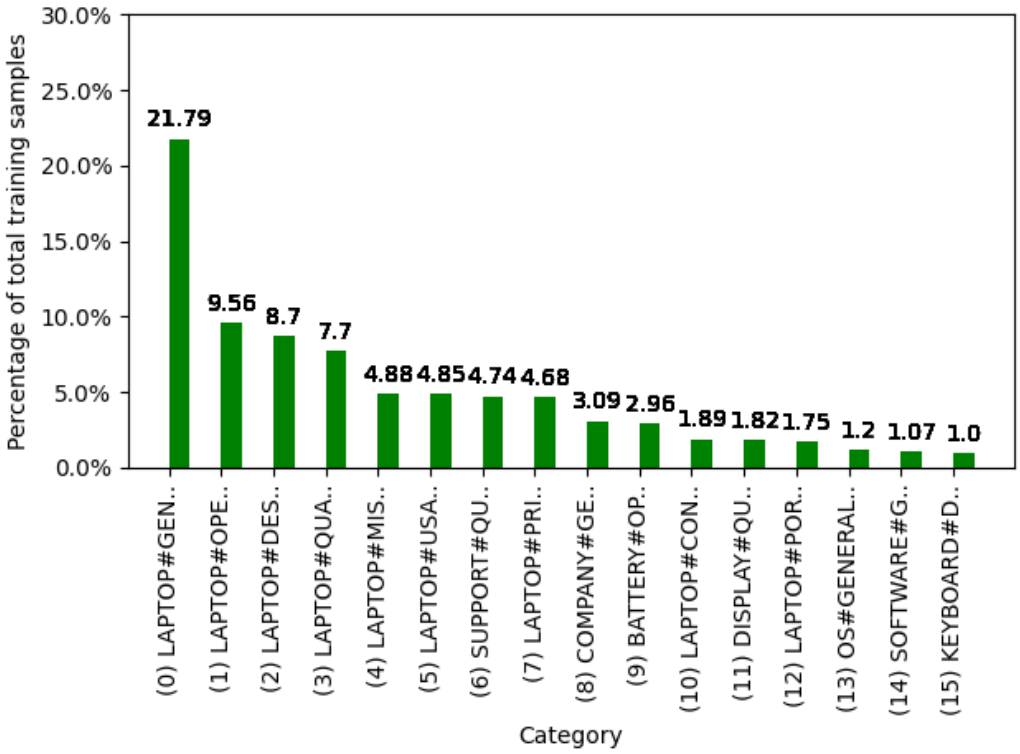


Figure 3.2: *Graph showing distribution of training sentences containing different aspect labels*

	Opinions	Count	Percentage
0	No Opinions	461	18.44%
1	One Opinion	1352	54.08%
2	Two Opinions	534	21.36%
3	Three Opinions	127	5.08%
4	Four or more opinions	26	1.04%

Table 3.2: *Table showing the distribution of number of opinions in sentences for training data*

3.1.1 Data Analysis

Figure 3.1, Figure 3.2 and Table 3.2 give insights into the SemEval laptops training dataset. These results can indicate possible pitfalls that could occur in the final models. It can be seen that the majority of sentences in the training data contain only one opinion at 54% of total sentences, however a large majority at 31.6% express multiple opinions. This is important to note because the aspect category detection system needs to be able to assign multiple labels to a single sentence, or it will not achieve very good performance at all.
Of the top 16 labels that have been chosen to be analysed, only four have a minimum of 5% of sentences being contained within them from training data. Table A.1 in the appendix shows the train counts for the visualized Figure 3.2. This shows that at least the top 13 aspects/labels have above 50 samples to train on. This is quite deplorable in comparison to the 634 sentences in the top label of LAPTOP#GENERAL, so it can be hypothesised that performance of correct classifications for each aspect will drop off significantly as the amount of training samples containing that label decreases.

The very low percentage of neutral sentences in the training data at 6.46% means the subjectivity classifier may not perform very well either, as this only contributes to 188 of the total 2909 opinion tuples. It is interesting to note that there exists slightly more positive sentences than negative, so the polarity classifier may have slight bias towards the positive class.

3.1.2 Evaluation Metrics

Aspect-category detection, being a multi-label classification problem, requires an evaluation metric that penalizes misclassifications. For this the micro-f1 is used, which is the harmonic mean of precision and recall. For subjectivity classification and polarity classification, the evaluation metric that will be used will be accuracy.
To define these evaluation metrics, first of all TP, TN, FP and FN need to be defined, which are the outcomes of classifying a single sample.

- TP: True Positive - A correctly identified positive sample

- TN: True Negative - A correctly identified negative sample
- FP: False Positive - An incorrectly identified positive sample
- FN False Negative - An incorrectly identified negative sample

Accuracy

Accuracy calculates the percentage of correct classifications over total

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.1}$$

Precision

Precision calculates the proportion of relevant retrieved samples

$$Precision = \frac{TP}{TP + FP} \tag{3.2}$$

Recall

Recall calculates the proportion of relevant relevant samples that are successfully retrieved

$$Recall = \frac{TP}{TP + FN} \tag{3.3}$$

Micro-F1

Micro-F1 uses the global precision and recall for all classes comparative to Macro-F1 which calculates the precision and recall for each class finds the average per class. F1 being a harmonic mean of both precision and recall penalizes a low score in any of the two metrics much more than the mean would, making it a good metric to measure performance by for aspect category detection.

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{3.4}$$

3.1.3 Ethics

The license of SemEval permits use for academic research. Although commercial applications of this project are discussed, this project is exclusively for academia. If hypothetically, a commercial ABSA system was to be developed, there exists services such as Amazon's Mechanical Turk[1] which allows participants to sign up and get paid for labelling training data. This would be an ethical use of human participants to get labelled training data. All of the libraries used for this project are open-source so freely

[1] `https://www.mturk.com/`

usable for any purpose, with the Apache License or equivalent (A permissive license whose main conditions require preservation of copyright and license notices).

Chapter 4

Design

4.1 General System Architecture

The laptops dataset that will be used for this project is available from SemEval in an XML format, so will need to be restructured so it can be utilised with Python. Each of the models implemented for this project will require their own pandas dataframe for training and testing data. For deep learning models, the training dataframe will have a random 20 percent taken to be the validation data. Preprocessed training data will be subject to feature extraction dependant on the task being tackled, and the training phase will halt at an optimal epoch given by the loss calculated from the validation data and the patience p.
The final model will be serialized for used with future data, including the test data. All of the deep learning models will use the adam optimisation algorithm, which is a modified version of stochastic gradient descent which is much computationally cheaper (Kingma & Ba 2014). It is the standard and recommended optimizer for use with Tensorflow.

The deep learning models will be developed using Keras's Functional API[1] which gives more flexibility than the standard sequential model; allowing models with non-linear topologies, allows shared layers between models, and most important allows multiple inputs or outputs; which is useful for some models developed in this project.

4.2 Standard Preprocessing

To improve accuracy of NLP models, textual data first needs to be preprocessed, with the goal being to leave only data that has a large impact on the classification of each sample.

[1] `https://www.tensorflow.org/guide/keras/functional`

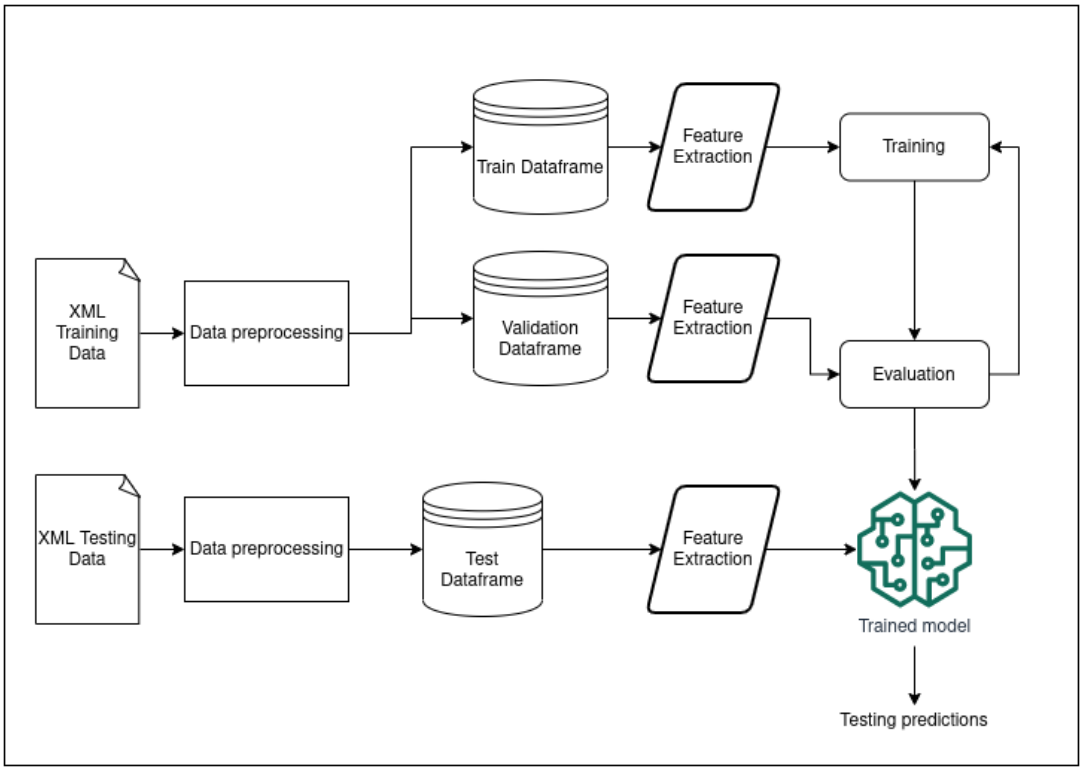


Figure 4.1: *System architecture diagram*

Text: *Suffice it to say, my MacBook Pro keeps me going with its long battery life and blazing speed.*

The raw text above from the training dataset's sentence:292:2 first needs to be needs to be normalised, converting all text to lower case. Next, the words need to be split from punctuation leaving the raw sentence text. Lastly, a stoplist is applied[2] removing non-content words from the sentences. A stoplist can be applied with two different methods. Firstly, the stopwords from the corpus can simply be removed, leaving the corpus with non-content words. The second method of applying a stoplist is replacing all stopwords with an out of vocabulary token, $< oov >$. This is done with the intention of keeping the sentence structure, as hypothetically, when a deep learning classifier fits the training data, it could use the distance between tokens of words to improve classification accuracy. The former stoplist application does not preserve token distance. When the first method of using a stoplist is applied to the training sample text above, it will become:

Text: *suffice macbook pro long battery life blazing speed*

[2] `https://github.com/igorbrigadir/stopwords`

If instead, the second method of stoplist application is used, the training sample text becomes:

Text: *suffice* <oov> <oov> <oov> <oov> *macbook pro* <oov> <oov> <oov> <oov> <oov> *long battery life* <oov> *blazing speed*

4.3 Embedding Layer

All of the deep learning models implemented for this project use Tensorflow's embedding layer to represent features in the input layer. For this layer to accept input data after standard preprocessing, the the text needs to be tokenized. A tokenizer object fits over training text and creates a dictionary of all words in the corpus, and maps them to an index. Sequences of input data are regarded as their index-dictionary mapping further down the input pipeline. Each word; now represented as a unique integer is replaced by it's d dimensional word embedding in the embedding layer. The indexes 0 and 1 are reserved for padding and out of vocabulary words respectively; which the tokenizer applies to the corpus samples, padding the words and changing unknown words to <oov>. The sentences that are shorter than the maximum sentence length, which is 29 words, are padded with trailing 0s.

When the tokenizer is initialized, a number of w max words is passed to it. The tokenizer counts the occurrences of each word in the training corpus and discards words that aren't in the top w words. The optimal w will be explored in this project.
Experiments will also be conducted on the best use of different weights for the Embedding layer, being either random weights; fixed GloVe weights, or the use of GloVe weights as a seed, allowing the network to adjust the weights through back propagation. The impact of the dimensionality of word vector representations will also be explored.

4.4 Aspect Category Detection

4.4.1 Baseline SVM Model

The first model developed for aspect category detection utilises a linear support vector classifier, as they have proven to perform effective in text classification tasks and make for a good baseline model. Sklearn's SGDClassifier is used in conjunction with the linear SVM, using the SVM to calculate a loss for each sample at a time; with the model being updated by performing stochastic gradient descent, maximising the performance of this linear classifier.

An effective sentiment analysis system needs to be able detect multiple opinions in a single sentence, especially as 27.48% of the training was found to contain more than one opinion per sentence. To allow the baseline to achieve this, a model is created for

each of the top 16 categories, with the output being the probability that a sample is in a given category or not; given by either a 1 or 0. Each of the sixteen models will perform separate classification on the data, each only concerned with classifying one label, and the results are joined at the end to analyse the performance of the models as a whole. The input of each model takes the standard preprocessed text described in Section 4.2, which feeds it into a Sklearn pipeline object. In the pipeline, a CountVectorizer then a TfidfTransformer is applied. TFIDF has been a popular information retrieval method for many years, which combines term frequency with inverse document frequency, to reflect how import a word is to a document in the corpus, with the idea being that words that appear often in a document are less relevant to to its retrieval and words that appear less frequent are more relevant to retrieval.

4.4.2 DNN Model

The deep neural network model employs a global max pooling layer attached directly to the features from the embedding layer, taking the maximum along each dimension of the word vectors (Section 2.4.4). This max pooling layer is fully connected to a 256-neuron dense layer with a ReLu activation function. The output layer has 16 neurons with sigmoid activations, calculating the probability a sample lies in each class.

The labels are a one-hot encoded matrix, with each index corresponding to whether the sample lies in the index's category. Each label has a index assigned based on its frequency in the train dataset: so the *LAPTOP#GENERAL* label has an index of 0, (as it has 634 samples in the dataset); and *KEYBOARD#DESIGN_FEATURES* has an index of 15 as it only has 29 samples in the train dataset. A sample containing exclusively these two labels is represented by the matrix:

Labels: *[1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1]*

4.4.3 CNN Model

Kim (2014) proposed a CNN be applied over multiple channels of feature vectors with different kernel sizes. The concatenation of the pooling of these layers showed to perform very well for various text categorization tasks. The CNN model developed for this project takes the feature vectors from the embedding layer and applies a Conv1D layer on each of the specified channels. Global max pooling is then applied to each of these channels before they are concatenated (See Figure 4.2). Each channel convolves over the input, extracting different n-gram features depending on the kernel size, which is different for each channel. Merged channels are then connected to an output layer of 16 neurons with a sigmoid activation function and dropout.

4.4.4 Bi-LSTM CNN Model

Zhou et al. (2015) proposed an advancement of the prior CNN model by applying an bidirectional LSTM to convolved feature vectors after concatenating the outputs from

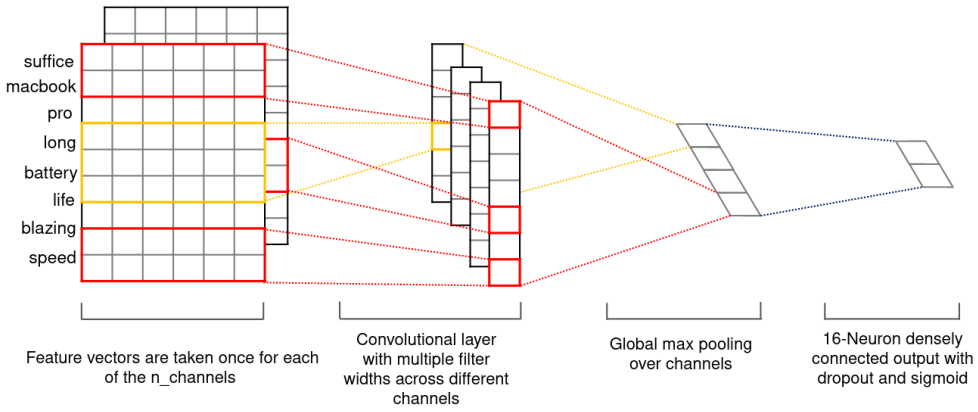


Figure 4.2: *CNN applied across multiple channels for feature vectors. Diagram adapted from Kim (2014)*

the different channels.
The Bi-LSTM model implemented for this project takes inspiration from this project, but applies the Bidirectional LSTM directly to the feature vectors after the embedding layer. Multiple channels of different kernel sizes are applied to the concatenation of the LSTM output and the original embedding layer output. The LSTM should learn important parts of the embedding feature representation, and store these over time stamps acting a memory unit within the network, using the gates within the LSTM unit. The rest of the model remains the same as the CNN model, with the convolved feature vectors being concatenated after max pooling, and then being connected to 16-neuron dense output layer with max pooling.

4.4.5 Hyperparameter tuning with Random Search

To explore the best hyper-parameters of this model, a RandomSearch algorithm is employed to explore the search space. As the Bi-LSTM is the main model developed for this project, it is the optimal model to use for testing hyper-parameter optimization. The library used for this is keras-tuner[3]. This library builds directly on top of Tensorflow, and can work with implemented Tensorflow architectures, with hyperparameters being defined as either an Int, Choice, or Float. For each hyperparameter, a *hp* object is created which has a set range of possible values which are defined on initialization, which the RandomSearch algorithm takes samples from. The search space for hyperparameters will need to include the number of filters, the number of channels and the

[3] `https://github.com/keras-team/keras-tuner`

value of dropout in the penultimate layer when being applied to the Bi-LSTM CNN model proposed.
Although it would also be ideal to add the number of convolutional layers per channel, or the number of neurons in the LSTM as parameters to be optimized, this would require more trials for the optimization to effectively explore the search space; and this is not within the computational budget of this project.

4.5 Subjectivity Classification

4.5.1 Baseline SVM Model

The subjectivity baseline takes sentences and fits a SVM classifier with SGD to TFIDF transformed word counts, similarly to the aspect category detection model, however all data is trained and tested with one model rather than a model for each aspect category. The sentences are labelled with a 0 if they are objective, and 1 if they contain a polarity. A sentence contains a polarity if the sentence contains at least one positive or negative opinion within it, else it is regarded as objective.

4.5.2 Basic Softmax CNN Model

After much experimentation with many different models, including re-using and modifying models from aspect category detection, the final deep learning model for subjectivity classification is a basic CNN with one convolutional layer, global max pooling and an output layer with two neurons with a softmax activation. The labels are a one-hot encoded matrix of the subjectivity of a sample, ie. [1, 0] represents objective and [0, 1] represents subjective. A sample can only be in one class. As this is a binary classification task, softmax is used instead of sigmoid in the output layer due to softmax being favoured for multi-class classification (where each sample is only in one of many classes) compared with multi-label classification (where each sample is in any number of many classes).
When using the same Bi-LSTM architecture from aspect category detection, the model did not even manage to perform better than the subjectivity baseline. This model was even modified to use softmax, yet still performed worse than the SVM and SGD classifier, so the final approach used is a much simpler one.

4.6 Polarity Classification

4.6.1 Baseline SVM Model

The SVM baseline for polarity classification is very similar to the aspect category detection architecture, with the difference being the labels, which are a 0 for negative

sentences and 1 for positive sentences.

4.6.2 Softmax Bi-LSTM CNN Model

The main model for polarity classification takes inspiration from both previous main models, but with some major changes. Firstly, the model takes two inputs, one being the sentence text and the second being a one-hot-encoded matrix of the label of the sentence wanting to be classified. Each sentence can be trained or tested multiple times, but with a different one-hot-encoded matrix for the class. Each matrix can only hold one class at a time. The general architecture takes the text, applies a Bi-LSTM to the embedding layer, where the concatenation of this and the original embedding layer is fed to the CNN channels. The output of the CNN channels is the merged poolings of the convolutions occurring on each layer. This is fed to a 16-neuron dense layer with a reLu activation, and this is finally merged with the original second input to the model, a 16-neuron reLu input layer representing the class of the sample. The amalgamation of these two layers are directly connected to an output layer of two neurons, with a softmax activation function. Labels are [1, 0] for a negative opinion expressed and [0, 1] for a positive opinion expressed.

Chapter 5

Implementation and Testing

5.1 Data Restructuring

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<Reviews>
...
<Review rid="292">
    ...
    <sentence id="292:2">
        <text>Suffice it to say, my MacBook Pro keeps me
            going with its long battery life and
            blazing speed.</text>
        <Opinions>
            <Opinion category="BATTERY#OPERATION_PERFORMANCE"
                polarity="positive"/>
            <Opinion category="LAPTOP#OPERATION_PERFORMANCE"
                polarity="positive"/>
        </Opinions>
    </sentence>
    ...
</Review>
...
</Reviews>
```

Figure 5.1: *Example review from train dataset in XML format*

SemEval's datasets are provided in XML format, so need to be put into a pandas dataframe for interaction with python functions.The XML tree hierarchy can be seen from Figure 5.1. Each node in the tree has some attributes, being id for sentences and

category or polarity for opinions within sentences.
To iterate through the XML tree, the xml.etree.ElementTree[1] module is used, with each sentence's text having standard preprocessing applied (See Section 4.2). Depending on the task at hand, a single sentence can be added multiple times with it's respective aspect/polarity tuple; or a single sentence can be added once, with relevant data regarding the task at hand. Before a sample is added to a dataframe, the sentence is normalised and split from punctuation. This is done with the following regular expression.

```
sentence_text = re.sub(r'[^\w\s]','',
    sentence_text.lower())
```

Pandas allows for serialization of data in a .pkl file format. A pickle file is a binary serialization format that can store a variety of datatypes and has built in interaction with other modules such as numpy for dealing with matrices. It is also considerably faster than storing data as a CSV or JSON.

5.2 Applying a stoplist

	Average sentence length	Max sentence length
Before applying a stoplist	13.37 words	73 words
After applying a stoplist	9.68 words	58 words

Table 5.1: *Average and max length lengths before and after a stoplist is applied to train data*

A stoplist which is loaded from a text file and stored in an array, can be applied in one of two ways, as described in Section 5.2. The first method is removing all stopwords from the text data, which is applied to the training dataframe with the following lambda function.

```
stopwords = stoplist()

train_df['text'] = train_df['text']
    .apply(lambda x: ' '
    .join([item for item in x.split()
    if item not in stopwords]))
```

Using the first implementation of a stoplist, table 5.1 shows that average sentence length decreases by 28% when a this method is applied to training data.

[1] https://docs.python.org/2/library/xml.etree.elementtree.html

The second method of applying a stoplist is replacing all stopwords with an out of vocabulary token, $< oov >$. The function to apply this to a dataframe can be seen below.

```
def apply_stoplist(df):
    stopwords = stoplist()
    for index, row in df.iterrows():
        split = row.text.split()
        sentence_words = []
        for item in split:
            if item in stopwords:
                item = '<oov>'
            sentence_words.append(item)
        row.text = sentence_words
    return df
```

5.3 Tokenizer

The tokenizer converts the text sequences to vectors from the training dataframe as follows:

```
tokenizer = tf.keras.preprocessing.text.Tokenizer(
    num_words=n_words,
    oov_token="<oov>")
tokenizer.fit_on_texts(train_df.text)
tokenizer.word_index['<pad>'] = 0
tokenizer.index_word[0] = '<pad>'
tokenizer.word_index['<oov>'] = 1
tokenizer.index_word[1] = '<oov>'

train_seqs = tokenizer.texts_to_sequences(train_df.text)
train_vector = tf.keras.preprocessing
    .sequence.pad_sequences(train_seqs, padding='post')
train_labels = np.stack(train_df.matrix, axis=0)

word_index = tokenizer.word_index
```

To explore the impact of the number of max words used by tokenizer, the DNN model from aspect category detection is used. For each fixed number of max words to be tested, *n_words*, an instance of the DNN model is created which is trained five times to try to mitigate randomness from weight initialization. The highest achieving of

the models for each *n_words* has it's test f1 score added to a pandas dataframe for later analysis. This process is completed three times to allow testing of the impact of different implementations of a stoplist; either not using one, having stopwords removed, or replacing stopwords with an out of vocabulary token.

5.4 Embedding Layer

The embedding layer is how feature vectors are fed into each deep learning model implemented for this project. The impact of varying embedding dimension will be explored.

```
glove_matrix = gloveEmbedding(300, word_index)

embedding_layer = layers.Embedding(len(word_index) + 1,
    300,
    weights=[glove_matrix],
    trainable=True)
```

The GloVe weights are loaded from a text file using a function that takes the parameters: d; the dimension of the embedding wanted, and the *word_index*; the word-index mapping that is generated from the tokenizer. The returned result is an array with length of the number the words in the corpus, and each index being d dimensions long, representing the GloVe feature vectors of each word. This is given when instantiating the Embedding layer. The length of *word_index* is also given when instantiating the Embedding layer, to indicate the number of unique words in the corpus. 1 is added to the number of words in word_index to account for the padding token.

5.5 Early Stopping

Early stopping is used by each of the deep learning models. The loss of the validation dataset is monitored, and if it does not see an increase after p epochs, the training phase will stop. The following snippet is used as a callback when fitting the training data, evaluating the validation data at the end of every epoch. It is also possible to monitor val_accuracy, however the models implemented use val_loss as the metric. The value of restore_best_weights was experimented with, and it was found that setting this to False gave a improvement in classification accuracy, so it is kept as False for all models implemented.

```
earlystop_callback = tf.keras.callbacks
    .EarlyStopping(monitor='val_loss',
    patience=p, restore_best_weights=False)
```

With manual testing, 50 was found to be a good early stopping p for aspect category detection, so this is the early stopping patience used throughout the project for all deep learning models.

5.6 Evaluation Metrics

When compiling a Tensorflow model, the metrics returned when evaluating need to be defined. The precision and recall can be used calculate the F1 Score, which is the main metric used to evaluate all of the models implemented.

```
METRICS = [
      tf.keras.metrics.BinaryAccuracy(name='accuracy'),
      tf.keras.metrics.Precision(name='precision'),
      tf.keras.metrics.Recall(name='recall'),
]
```

5.7 Aspect Category Detection

5.7.1 SVM Model

The SVM is a simple model to implement with the SKLearn pipeline object. The SGDClassifier works by using SVMs by default, to use as the classifier.

```
      text_clf = Pipeline([
          ('vect', CountVectorizer()),
          ('tfidf', TfidfTransformer()),
          ('svc', SGDClassifier()),
          ])
```

5.7.2 DNN Model

The implementation of the deep neural network with 256 neurons in a hidden layer after global max pooling of feature vectors can be seen below. As the top 16 aspects are being explored, the output layer has 16-neurons.

```
input_layer = layers.Input(shape=(input_length,))

embedding = embedding_layer(input_layer)
pooling = layers.GlobalMaxPooling1D()(embedding)
dense = layers.Dense(256, activation='relu')(pooling)
outputs = layers.Dense(n, activation='sigmoid')(dense)

model = tf.keras.Model(inputs=input_layer, outputs = outputs)
```

5.7.3 CNN Model

The CNN model uses three channels each with a *filter_size* of 256 to convolve over feature vectors. The GlobalMaxPooling operator is applied to the output of each of these layers, before they are concatenated. The concatenated result, *channels_output* is then connected to the output layer with dropout, which will be 16 for aspect category detection. *kernel_array* is an array of the kernel sizes used for each layer. The default array: [1, 2, 3] . The dropout in the penultimate layer of this model is 0.2.

```
n_channels = len(kernel_array)
convs = []
poolings = []

input_layer = layers.Input(shape=(input_length,))
embedding = embedding_layer(input_layer)

if (n_channels == 1):
    conv = layers.Conv1D(filters=filters, kernel_size=
        kernel_array[0], activation='relu')(embedding)
    channels_output = layers.GlobalMaxPooling1D()(conv)

else:
    for i in range(n_channels):
        conv1 = layers.Conv1D(filters=filters,
            kernel_size=kernel_array[i],
            activation='relu')(embedding)

        convs.append(conv1)
        poolings.append(layers.GlobalMaxPooling1D()(convs[i]))

    channels_output = layers.concatenate(poolings)

dropout = layers.Dropout(0.2)(channels_output)
outputs = layers.Dense(n, activation='sigmoid')(dropout)
model = tf.keras.Model(inputs=input_layer, outputs = outputs)
```

5.7.4 Bi-LSTM CNN Model

The architecture of the Bi-LSTM can be seen below. The model applies a bidirectional LSTM directly to the embedding layer, and concatenates this to the original embedding layer before it is fed into the CNN architecture from the model above. The default parameters are the same as the CNN model, with the only difference being the number of neurons in the LSTM layer, which is set to 128.

```
n_channels = len(kernel_array)
convs = []
poolings = []

input_layer = layers.Input(shape=(input_length,))
embedding = embedding_layer(input_layer)
bilstm = layers.Bidirectional(layers.LSTM(128,
    return_sequences=True))(embedding)
conc = tf.keras.layers.concatenate([embedding, bilstm])

if (n_channels == 1):
    conv = layers.Conv1D(filters=filters,
        kernel_size=kernel_array[0], activation='relu')(conc)
    channels_output = layers.GlobalMaxPooling1D()(conv)

else:
    for i in range(n_channels):
        conv1 = layers.Conv1D(filters=filters, kernel_size
            =kernel_array[i], activation='relu')(conc)

        convs.append(conv1)
        poolings.append(layers.GlobalMaxPooling1D()(convs[i]))

    channels_output = tf.keras.layers.concatenate(poolings)

dropout = layers.Dropout(0.2)(channels_output)
outputs = tf.keras.layers.Dense(n, activation='sigmoid')(dropout)
model = tf.keras.Model(inputs=input_layer, outputs = outputs)
```

5.7.5 Hyperparameter tuning with Random Search

To test hyperparameter tuning, a Random Search algorithm is employed to explore the search space of the Bi-LSTM CNN model. The chosen parameters to tune for this model are the number of filters, number of channels and dropout in the penultimate layer. In Keras Tuner, the parameters to be optimized are instantiated as a hp object which can be of type: Choice, Int or Float. The filters are a choice of [64, 128, 256]. The number of channels are between one and five, with the *kernel_array* being fixed, meaning if the number of channels is two, the kernel array will be [1,2]. Lastly, the dropout in the penultimate layer is between 0 and 0.5. The three parameters to be tuned are defined as hp objects below.

```
kernel_array = [1,2,3,4,5,6]
```

```
filters = hp.Choice(
        'filters',
        values=[64, 128, 256])

n_channels = hp.Int(
        'n_channels',
        min_value=1,
        max_value=5,
        )

dropout = rate=hp.Float(
        'dropout1',
        min_value=0.0,
        max_value=0.5,
        default=0.25,
        step=0.05)
        )
```

Exploring the hyperparameter search space after initialisation is done with the by the following:

```
tuner = RandomSearch(
build_model,
objective='val_accuracy',
max_trials=150,
executions_per_trial=6,
directory='keras_tuner',
project_name='acd_lstm_cnn_randomsearch')
```

5.8 Subjectivity Classification

5.8.1 Basic CNN with Softmax

```
input_layer = layers.Input(shape=(input_length,))
embedding = embedding_layer(input_layer)

conv = layers.Conv1D(filters=128, kernel_size=2,
    activation='relu')(embedding)
pooling = layers.GlobalMaxPooling1D()(conv)

outputs = layers.Dense(2, activation='softmax')(pooling)
model = tf.keras.Model(inputs=input_layer, outputs = outputs)
```

5.9 Polarity Classification

5.9.1 Softmax Bi-LSTM with Softmax

The model has a dropout of 0.3 after the CNN channels. The model also uses 3 channels with a kernel_array of [1,2,3], similar to that of aspect category detection's Bi-LSTM CNN model.

```
convs = []
poolings = []

input_layer = layers.Input(shape=(29,))
intput_layer_category = layers.Input(shape=(16,))

embedding = embedding_layer(input_layer)
bilstm = layers.Bidirectional(
    layers.LSTM(128, return_sequences=True))(embedding)
conc = tf.keras.layers.concatenate([embedding, bilstm])
for i in range(n_channels):
    conv1 = layers.Conv1D(filters=filters,
        kernel_size=kernel_array[i], activation='relu')(conc)
    convs.append(conv1)
    poolings.append(layers.GlobalMaxPooling1D()(convs[i]))
    channels_output = tf.keras.layers.concatenate(poolings)

dropout = layers.Dropout(0.3)(channels_output)

dropout_dense = tf.keras.layers.Dense(16,
    activation='relu')(dropout)
category_dense = layers.Dense(16,
    activation='relu')(intput_layer_category)
merged_inputs = layers.concatenate(
    [dropout_dense, category_dense])
outputs = tf.keras.layers.Dense(2,
    activation='softmax')(merged_inputs)

model = tf.keras.Model(inputs=
    [input_layer, intput_layer_category], outputs = outputs)
```

5.10 Testing

The main functions used throughout this project are all implemented from large open source projects, so it was assumed that they had already been diligently tested. The

functions developed for this project underwent manual testing.

Case	Test	Success
1	Does the aspect category detection training dataframe contain 2500 samples and the testing dataframe contain 808 samples with correct labels?	True
2	Does the subjectivity training dataframe contain 2500 samples and the testing dataframe contain 808 samples with correct labels?	True
3	Does the polarity training dataframe contain 2239 samples and the testing dataframe contain 581 samples with correct labels?	True
4	Does the text preprocessing lambda function remove all punctuation from text samples?	True
5	Does the first stoplist architecture of removing all stopwords correctly remove all words in the stoplist from text samples?	True
6	Does the second stoplist architecture of replacing all stopwords with an out of vocabulary token correctly replace all words in the stoplist from text samples?	True

Table 5.2: *Different tests implemented and their respective success*

Table 5.2 shows partial testing which covers all of the fundamental functions implemented directly for this project; testing in extreme cases where applicable.

Ten random samples are taken taken to ensure integrity of success claims for tests 1 through 3. Test 1 through 3 are validated by comparing the labels with the same sentence ID from the raw XML data. The ground truth for the number of samples in the training and testing datasets in these tests are generated using a Counter[2] object applied to a seperate dataframe, counting the number of sentences, and number of expressed opinions. The reason why Test 3 requires less samples than 1 and 2 is because a sample with no class or a neutral polarity is not considered to have a polarity by this system, so could not be represented in the one hot encoded matrix of two classes: [negative, positive].

For Test 4, the following sentence is passed to the preprocessing function:

Text: THIS IS A SENTENCE and needs to be normalized ??##%^()!

The given result is:

this is a sentence and needs to be normalized

Tests 5 and 6 are run with the following sentence:

this sentence has some stopwords

[2] `https://docs.python.org/2/library/collections.html`

The first stoplist removal technique gives the following result for Test 5:

['sentence', 'stopwords']

The second stoplist removal technique gives the following result for Test 6:

['<oov>', 'sentence', '<oov>', '<oov>', 'stopwords']

Chapter 6

Results and Discussion

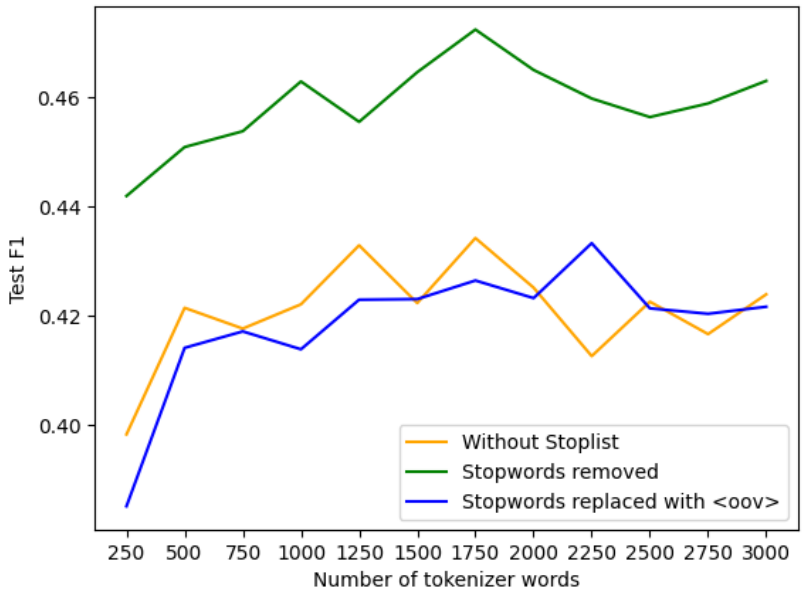


Figure 6.1: *Impact of max number of words used by tokenizer and method of applying stoplist stoplist on test F1-performance*

6.1 Representing Inputs

6.1.1 Stopwords & Tokenizer words

The impact of the maximum number of unique words to be used by the tokenizer can be seen by Figure 6.1. The results show that removing stopwords completely cause

models at each fixed number of max tokenizer words to perform significantly better than it's partner without. Analysing the data further, it can be seen that using a stoplist that removes all non-content words over not using one gives an average of a 9.03% increase in test F1 score over all models. As there are 12 models being used for each of the different tokenizer word counts, and each model is trained 5 times, there is 60 models in this sample size for using a stoplist, and 60 for not using one. For this reason, the validity of these claims is quite strong due to the variance of test accuracy from the random weight initialization being modulated from multiple runs.
The graph also shows a stoplist being applied that converts non-content words into out-of-vocabulary tokens. This has quite poor performance, with the F1-score showing a very similar performance to not using a stoplist at all. It is possible that keeping the semantic information of distance between tokens would be better used by a more advanced model such as the CNN or Bi-LSTM implemented later in this project, however to keep within the scope of this project's goals, all models discussed beyond this point will use a simple stoplist removal technique of removing non-content words completely. With samples taken at 250 word intervals, decreasing the tokenizer word

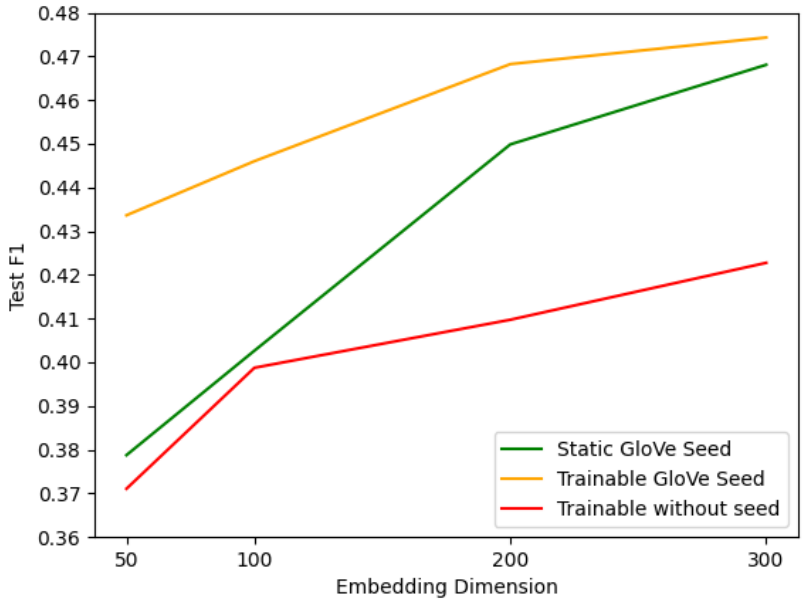


Figure 6.2: *Impact of varying the embedding dimension and type of weights on test performance*

count does not seem to show a strong pattern although it tends in a negative linear direction. Using less than 500 words as the word count gives a sharp decrease in test f1. The word count with the highest achieving performance on the test data for all models with the basic stoplist and no stoplist, is 1750; so this is the word count that

will be used for all future models. It is interesting to note that the advanced stoplist performs best at 2250 max words, but still doesn't come close to the simple stoplist application at any number of max tokenizer words.

6.1.2 Embedding Layer & GloVe Weights

Figure 6.2 shows that using a GloVe matrix as a seed for the weights in the embedding layer gives the best performance on test data. The lowest performance across all embedding dimensions is using the embedding layer with randomly initialized weights. This is the expected results from running these tests.
It is important to note that at an embedding dimension of 300, which is highest of provided Glove embeddings for this project: updating the GloVe seed through back propagation only gives a marginal performance benefit over keeping the weights static. For these reasons, all models discussed later in this project will use GloVe weights a seed for the embedding layer, but they are able to update in the training phase through back propagation within through the network.

6.2 Aspect Category Detection

Main Model Architecture	F1 Score
SVM with linear kernel and SGD	0.453
DNN with 256 neuron hidden layer	0.481
CNN with 3 channels	0.486
Bi-LSTM-CNN	0.493
Bi-LSTM-CNN with tuned hyperparameters	**0.494**

Table 6.1: *Comparison of Test F1 Scores achieved by different implemented models for aspect category detection*

Table 6.1 shows the best F1 scores for each system developed for aspect category detection. Table 6.2 summaries the results obtained by the different models implemented for aspect category detection, with the objective of each model being to maximise the F1-score for the test dataset. It can be seen, that as expected, the baseline SVM with linear kernel used with a SGD classifier achieves the lowest F1-score. The highest achieving model is the Bi-LSTM-CNN, only marginally beating the CNN without an LSTM however. The DNN implemented also achieves impressive results, with the F1 being only 0.005 less than the F1 of the CNN model.

SVM Model

The SVM used as a baseline achieves a higher than expected F1 score of 0.453. The system is a culmination of sixteen different models, one for each aspect category. Each

Aspect Category	Count	svm	dnn	cnn	lstm	lstm_tuned
LAPTOP#GENERAL	634	25.99%	39.86%	37.06%	37.06%	**42.66%**
LAPTOP#OPERAT...	278	**9.90%**	7.94%	7.94%	9.52%	9.52%
LAPTOP#DESIGN_FE...	253	9.16%	10.71%	12.50%	17.86%	**23.21%**
LAPTOP#QUALITY	224	6.93%	4.55%	4.55%	**9.09%**	**9.09%**
LAPTOP#MISC...	142	**5.57%**	0.00%	0.00%	0.00%	0.00%
LAPTOP#USABILITY	141	4.08%	0.00%	0.00%	**2.63%**	**2.63%**
SUPPORT#QUALITY	138	**4.58%**	0.00%	0.00%	0.00%	0.00%
LAPTOP#PRICE	136	3.59%	4.17%	4.17%	12.50%	**20.83%**
COMPANY#GENERAL	90	2.60%	8.11%	8.11%	**18.92%**	**18.92%**
BATTERY#OPERAT...	86	2.23%	**5.88%**	**5.88%**	**5.88%**	**5.88%**
LAPTOP#CONNECTIVITY	55	**1.24%**	0.00%	0.00%	0.00%	0.00%
DISPLAY#QUALITY	53	**1.86%**	0.00%	0.00%	0.00%	0.00%
LAPTOP#PORTABILITY	51	0.50%	0.00%	0.00%	**20.00%**	**20.00%**
OS#GENERAL	35	**2.23%**	0.00%	0.00%	0.00%	0.00%
SOFTWARE#GENERAL	31	**0.37%**	0.00%	0.00%	0.00%	0.00%
KEYBOARD#DESIGN_FE...	29	**1.36%**	0.00%	0.00%	0.00%	0.00%

Table 6.2: *Percentage of correct predictions when given samples only from the given aspect category for all of the models implemented for aspect category detection. Count is the number of samples in the training data with the aspect category label*

model determines the probability of whether a sample is in the model's single aspect category or not. As the number of training samples in the given aspect category decreases, so does the percentage of correct classifications. The SVM's performance is very linear comparative to the number of training samples, with the only exception being SUPPORT#QUALITY, achieving the highest correct classifications for this label out of all of the models. The SVM actually ties first place with the Bi-LSTM CNN for the most number of aspect categories with the best classification of all models; however three of these classes are the categories with the lowest training counts, with a combined number of training counts of 95.

It is interesting that all of the other classifiers correctly identify no samples from the LAPTOP#MISCELLANEOUS category, despite there being 143 training samplesm and the SVM managing to return more than 5% of test samples in this class. This is also true for SUPPORT#QUALITY with 138 training samples and only the SVM model again being the only classifier to achieve higher than 0% correct classifications. The SVM also achieves the highest percentage of correct classifications for LAPTOP#OPERATION_PERFORMANCE, which is again impressive as this is the label with the second highest number of training samples. Overall, the SVM model works very well, and is able to at least correctly identify one sample from each aspect category, something that cannot be said about any of the other models.

Deep learning models

The deep neural network and convolutional neural net models achieve a very similar F1-score, however the addition of a bidirectional LSTM to the CNN's embedding layer does help to improve performance significantly. The DNN performs worse or equal to the CNN in every aspect category except for LAPTOP#GENERAL, which it notably achieves better, especially as this label has by far the most training and testing samples at 21.79% and 19.73% respectively.
The Bi-LSTM implemented after the CNN manages to beat or match all of the CNN's correct classification scores for every aspect category. It seems that the bidirectional LSTM applied to the feature vectors, manages to learn and remember important features for correct identification of aspects. Although the results look good, of the sixteen top aspect categories, seven are never predicted by the lstm and tuned lstm, including LAPTOP#MISC which has 142 training samples.

n_channels	dropout	filters	test_acc	test_f1
3.0	0.30	256.0	0.955059	0.494343
2.0	0.30	256.0	0.954904	0.491718
2.0	0.50	256.0	0.954904	0.491718
4.0	0.40	256.0	0.954595	0.489121
3.0	0.15	256.0	0.953357	0.486809
3.0	0.50	256.0	0.953280	0.485520
3.0	0.50	256.0	0.953280	0.485520
1.0	0.00	256.0	0.955059	0.485385
4.0	0.50	256.0	0.956064	0.481752
4.0	0.50	256.0	0.956064	0.481752

Table 6.3: *Top 10 models returned from RandomSearch of tuning n_channels, dropout, and filters*

Hyperparameter Tuning with RandomSearch

Table 6.3 shows the results which give an overview of the tested hyper parameters by the RandomSearch algorithm. It was run for 100 trials with 3 executions per trial, which re-runs each trial multiple times to stop bias from the randomness brought forward by weight initialisation. The only hyperparameter that seems to show consistency amongst the top models is the number of filters for the CNN, which is 256 for all of the top 10 models. The number of channels appearing most in the top models is three, with the model with 3 channels also giving the highest test f1. The dropout value however, seems completely random, with no patterns being shown from the results.
Although the tuned model did achieve the highest f1 of all models for aspect category

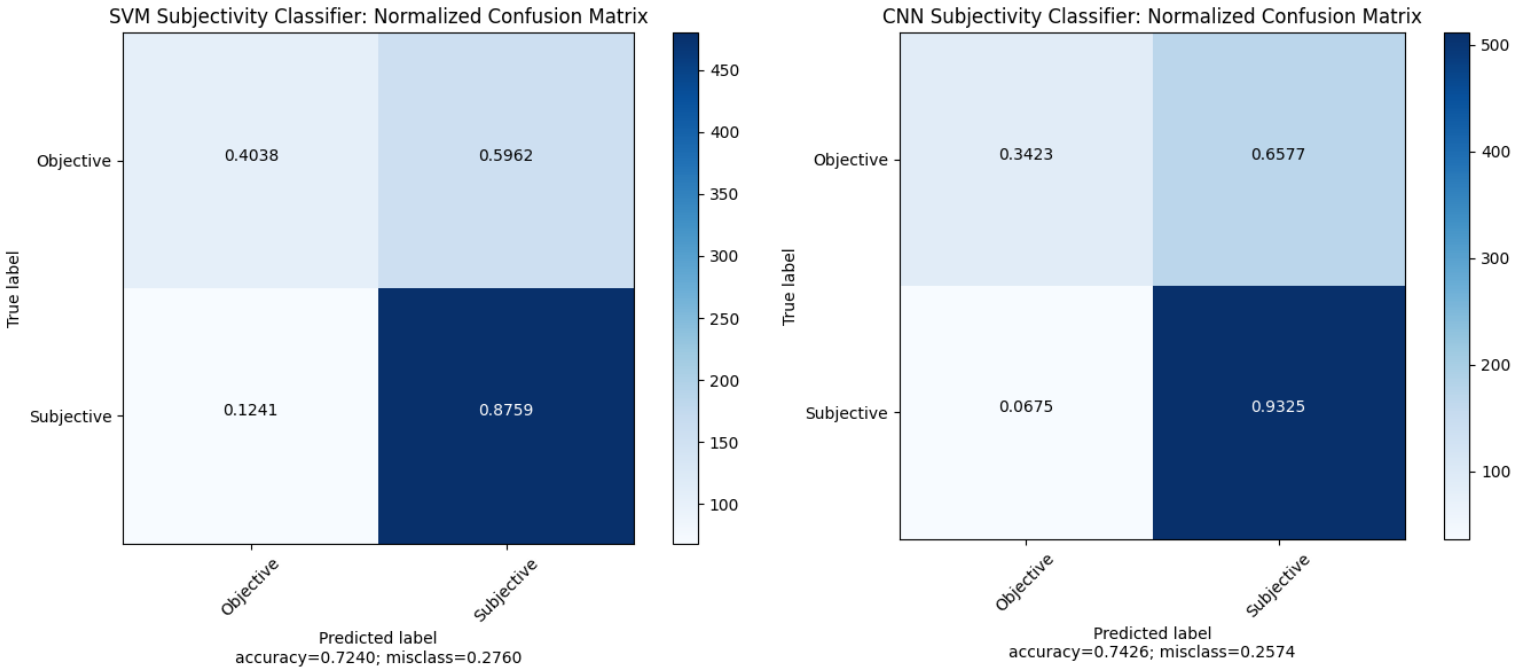


Figure 6.3: *Normalized confusion matrices for subjectivity classification*

detection, this could simply be due to convenient weight initialization, which is possible, as the tuner was had the highest computational budget of all of the models in this project. Perhaps to improve the consistency in the results, the number of trials needs to be increased. This is plausible, as the performance of a RandomSearch algorithm is simply the computational budget that it is given to it for it to explore the search space.

6.3 Subjectivity Classification

Main Model Architecture	Accuracy
SVM with linear kernel and SGD	0.724
Basic CNN Architecture	**0.743**

Table 6.4: *Comparison of Test F1 Scores achieved by different implemented models for subjectivity classification*

Table 6.4 shows the accuracy scores of the SVM and basic CNN architecture implemented for subjectivity classification. Although both models seem to achieve respectable accuracy scores of over 0.7, as expected, when analysing the training data, the models perform significantly worse for objective sentences, due to the very small

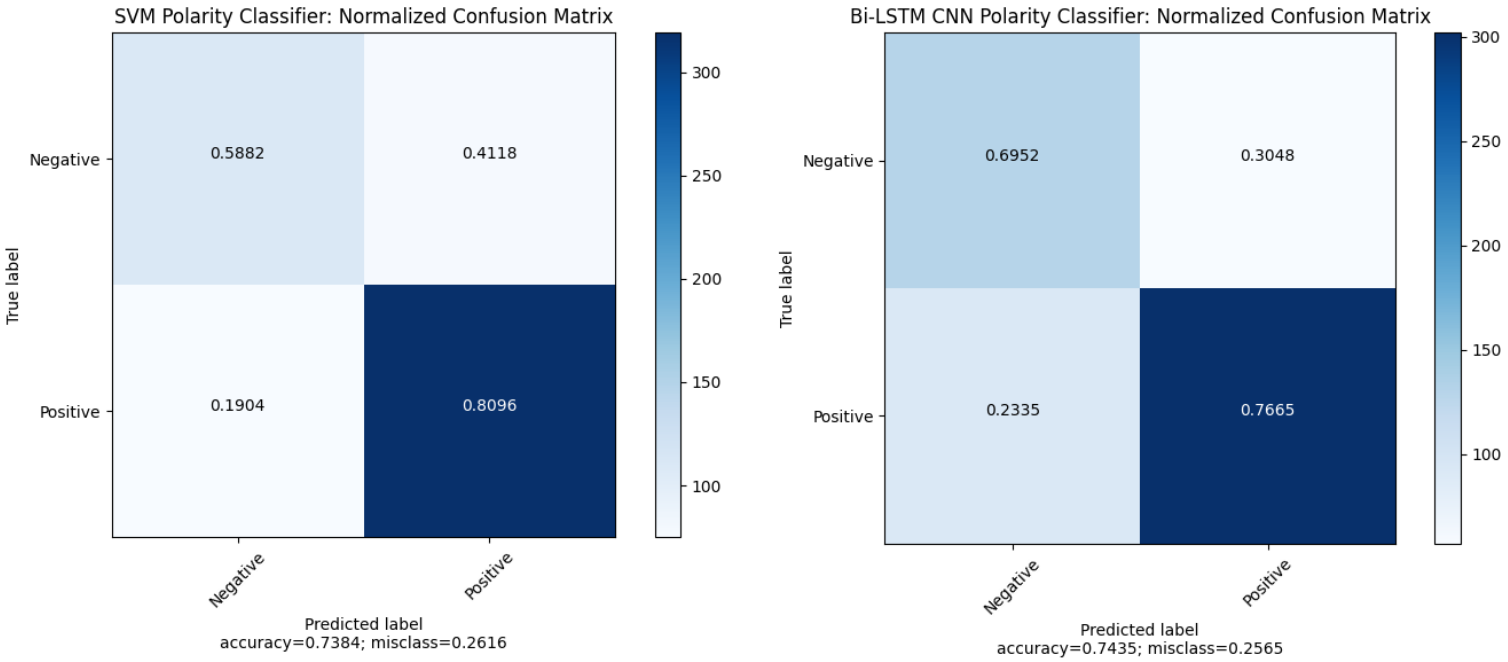


Figure 6.4: *Normalized confusion matrices for polarity classification*

amount of objective sentences the models encounter during the training phase. The CNN model improves on the SVM by increasing the amount of true positives for subjective statements. This comes at the cost of increased classifications of data that has a true objective label, but the CNN classifies as subjective. Overall, the increase of true positives for subjective labels from 0.876 with the SVM to 0.933 with the CNN is a welcome improvement. The confusion matrix showing this data can be seen by Figure 6.3.

Main Model Architecture	Accuracy
SVM with linear kernel and SGD	0.738
Bi-LSTM CNN with Softmax output	**0.744**

Table 6.5: *Comparison of Test F1 Scores achieved by polarity classification models*

6.4 Polarity Classification

Table 6.5 shows the accuracy of the SVM and Bi-LSTM implemented for polarity classification, and 6.4 shows the confusion matrix for both models. The models only take subjective statements, and perform 2-class classification returning either negative or

positive. The SVM model achieves a respective accuracy of 0.724, identifying positive sentences correctly more than 80% of the time. As 56.27% of subjective statements in the training data are positive, it is no surprise that the classifier is very good at detecting positive sentences. The Bi-LSTM improves on the SVM by 0.19 in accuracy score, however taking a look at the confusion matrix, the precision increases substantially over the SVM model. True negative predictions increase by over 20% with the Bi-LSTM, and the incorrectly classified negative samples are reduced by over 25%. The improved model does however reduce the amount of correctly identified positive samples from 0.81 to 0.77.

Chapter 7

Conclusion

7.1 Goals Achieved

Of the requirements set out in Table 3.1, all requirements were achieved to an above expected standard. For aspect category detection, a baseline was implemented in the form of an SVM with SGD, and the results of this were compared against a deep neural network, convolutional neural network, and a CNN with an LSTM applied to feature vectors. Each model improved on the accuracy of the last, and increased in model complexity. The Bi-LSTM CNN model was also used with a RandomSearch algorithm in attempts to find optimal hyper-parameters, finding that the a filter size of 256 works well for this problem domain, however more than expected trials should have been used for even the small search space of 3 hyperparameters with a large choice of options. An SVM similar to that of aspect category detection was implemented for subjectivity classification, and polarity classification. A deep learning model in the form a basic CNN model was implemented for subjectivity classification, testing with this model to prove claims of softmax performing better than sigmoid for multi-class classification, but sigmoid being better for multi-label classification, which is what was found by the implemented model. A deep learning model was also implemented for polarity classification, making use of complex techniques utilized in various models throughout the project to create a model that takes multiple inputs and uses this with a single model architecture to predict polarity of text, joining the two different inputs inside the model architecture with both playing a role of the final classification. The feature representation of text was also investigated, finding that the impact of word embedding dimensionality can actually be a greater impact on test results than even the choice of optimizer or architectural design, so is an imperative part of a model. Using pre-trained weights as a seed was also found to be optimal in the embedding layer, allowing seed weights to fit the problem domain proved to be of more use than a static embedding.

In conclusion, the use of Tensorflow to create various architectures of differing machine learning models to appreciate and understand the fundamentals of data science

in the application of supervised machine learning to investigate aspect-based sentiment analysis has definitely been completed with this project.

7.2 Further Work

An idea for further work for this project, could be the combination of the models created in this project to work in harmony in an input pipeline, with all of the models being served in a production environment with Docker. This would be exposed over a REST API so a scraper system could send requests to a production server with whole text reviews; splitting texts into sentences for them to be classified. The newly split sentences would be categorized, have subjectivity classification applied and polarity classification applied to subjective statements. This system could return all opinion tuples of sentences within the given text, and if the models were to be modified, intensity of opinions about aspects could be explored. This system could be sold to e-commerce sellers such as sellers on platforms such as Amazon's FBA; who would make use of analysing their product own reviews, and can use this to find flaws in their products and innovate their product lifestyle's, or analyse competitors products to find what is working well and what the customers like about the competitors products. Alternatively the system could be used more intuitively, to find patterns in products with similar aspect categories that generate a lot of revenue, to design and find new products to sell based on suggestions found by the ASBA system by information left in product reviews. With even better training data, the results achieved for the architectures implemented for this project could achieve such higher results and be used in a commercial application provided a Platform As a Service (PaaS) as a fully integrated cloud ABSA system.

Bibliography

Awad, M. & Khanna, R. (2015), *Support Vector Machines for Classification*, pp. 39–66.

Bergstra, J. & Bengio, Y. (2012), 'Random search for hyper-parameter optimization.', *J. Mach. Learn. Res.* **13**, 281–305.
URL: *http://dblp.uni-trier.de/db/journals/jmlr/jmlr13.htmlBergstraB12*

Bezos, J. (2013), 'Amazon ceo: Focus on customer is key'.
URL: *https://www.youtube.com/watch?v=56GFhr9r36Yfeature=emb$_t$itle*

Boser, B. E., Guyon, I. M. & Vapnik, V. N. (1992), A training algorithm for optimal margin classifiers, *in* 'Proceedings of the Fifth Annual Workshop on Computational Learning Theory', COLT '92, Association for Computing Machinery, New York, NY, USA, p. 144–152.
URL: *https://doi.org/10.1145/130385.130401*

Burkov, A. (2019), *The Hundred-Page Machine Learning Book*, Andriy Burkov.
URL: *https://books.google.co.uk/books?id=0jbxwQEACAAJ*

Cortes, C. & Vapnik, V. (1995), 'Support-vector networks', *Mach. Learn.* **20**(3), 273–297.
URL: *https://doi.org/10.1023/A:1022627411411*

Fürnkranz, J. (1998), 'A study using n-gram features for text categorization', *Austrian Research Institute for Artifical Intelligence* **3**(1998), 1–10.

G, V. & Chandrasekaran, D. (2012), 'Sentiment analysis and opinion mining: A survey', *Int J Adv Res Comput Sci Technol* **2**.

Hertz, J. A. (2018), *Introduction to the theory of neural computation*, CRC Press.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. R. (2012), 'Improving neural networks by preventing co-adaptation of feature detectors'.

Ho, Y. & Wookey, S. (2019), 'The real-world-weight cross-entropy loss function: Modeling the costs of mislabeling', *IEEE Access* **PP**, 1–1.

Hochreiter, S. & Schmidhuber, J. (1997), 'Long short-term memory', *Neural computation* **9**(8), 1735–1780.

Kim, Y. (2014), Convolutional neural networks for sentence classification, *in* 'Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)', Association for Computational Linguistics, Doha, Qatar, pp. 1746–1751.
URL: *https://www.aclweb.org/anthology/D14-1181*

Kingma, D. P. & Ba, J. (2014), 'Adam: A method for stochastic optimization'.

Liu, B. (2012), *Sentiment Analysis and Opinion Mining*, Morgan and Claypool Publishers.

Mikolov, T., Chen, K., Corrado, G. & Dean, J. (2013), 'Efficient estimation of word representations in vector space', *arXiv preprint arXiv:1301.3781* .

Nielsen, M. A. (2015), *Neural networks and deep learning*, Vol. 2018, Determination press San Francisco, CA, USA:.

Pai, V. (2020), 'gRPC Design and Implementation'.
URL: *https://platformlab.stanford.edu/Seminar%20Talks/gRPC.pdf*

Pennington, J., Socher, R. & Manning, C. D. (2014), Glove: Global vectors for word representation, *in* 'Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)', pp. 1532–1543.

Pontiki, M., Galanis, D., Papageorgiou, H., Androutsopoulos, I., Manandhar, S., AL-Smadi, M., Al-Ayyoub, M., Zhao, Y., Qin, B., De Clercq, O., Hoste, V., Apidianaki, M., Tannier, X., Loukachevitch, N., Kotelnikov, E., Bel, N., Jiménez-Zafra, S. M. & Eryiğit, G. (2016), SemEval-2016 task 5: Aspect based sentiment analysis, *in* 'Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)', Association for Computational Linguistics, San Diego, California, pp. 19–30.
URL: *https://www.aclweb.org/anthology/S16-1002*

Ruder, S., Ghaffari, P. & Breslin, J. G. (2016), 'A hierarchical model of reviews for aspect-based sentiment analysis', *CoRR* **abs/1609.02745**.
URL: *http://arxiv.org/abs/1609.02745*

Shen, D., Wang, G., Wang, W., Renqiang, M., Su, M. Q., Zhang, Y. & Chunyuan Li Ricardo Henao1, L. C. (2018), *Baseline Needs More Love: On Simple Word-Embedding-Based Models and Associated Pooling Mechanisms*, Association for Computational Linguistics.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. & Salakhutdinov, R. (2014), 'Dropout: A simple way to prevent neural networks from overfitting', *J. Mach. Learn. Res.* **15**(1), 1929–1958.

Statista (2019), 'U.s. amazon market share in e-commerce and retail 2020'.
URL: *https://www.statista.com/statistics/788109/amazon-retail-market-share-usa/*

Toh, Z. & Su, J. (2016), NLANGP at SemEval-2016 task 5: Improving aspect based sentiment analysis using neural network features, *in* 'Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)', Association for Computational Linguistics, San Diego, California, pp. 282–288.
URL: *https://www.aclweb.org/anthology/S16-1045*

Wang, S.-C. (2003), *Artificial Neural Network*, Springer US, Boston, MA.
URL: *https://doi.org/10.1007/978-1-4615-0377-4*$_5$

Winkler, N. (2019), 'What is the future of ecommerce?'.
URL: *https://www.shopify.com/enterprise/the-future-of-ecommerce*

Yang, Z., Dai, Z., Salakhutdinov, R. & Cohen, W. W. (2017), 'Breaking the softmax bottleneck: A high-rank RNN language model', *CoRR* **abs/1711.03953**.
URL: *http://arxiv.org/abs/1711.03953*

Zhou, C., Sun, C., Liu, Z. & Lau, F. C. M. (2015), 'A C-LSTM neural network for text classification', *CoRR* **abs/1511.08630**.
URL: *http://arxiv.org/abs/1511.08630*

Appendices

Appendix A

Data Analysis Appendix

	Aspect Category	Train Counts	Percentage of train
0	LAPTOP#GENERAL	634	21.79%
1	LAPTOP#OPERATION_PERFORMANCE	278	9.56%
2	LAPTOP#DESIGN_FEATURES	253	8.70%
3	LAPTOP#QUALITY	224	7.70%
4	LAPTOP#MISCELLANEOUS	142	4.88%
5	LAPTOP#USABILITY	141	4.85%
6	SUPPORT#QUALITY	138	4.74%
7	LAPTOP#PRICE	136	4.68%
8	COMPANY#GENERAL	90	3.09%
9	BATTERY#OPERATION_PERFORMANCE	86	2.96%
10	LAPTOP#CONNECTIVITY	55	1.89%
11	DISPLAY#QUALITY	53	1.82%
12	LAPTOP#PORTABILITY	51	1.75%
13	OS#GENERAL	35	1.20%
14	SOFTWARE#GENERAL	31	1.07%
15	KEYBOARD#DESIGN_FEATURES	29	1.00%

Table A.1: *Distribution of labels in the training dataset*