Student ID: 2031077, Text Analytics Coursework

Task 1

1.1

For my classifier, I have chosen multinomial logistic regression (MLR), implemented via scikitlearn [1]. MLR does not rely on the conditional independence assumption, so should work well on features that are highly correlated with each other. For this reason, I thought it would be more suitable than Naive Bayes. MLR attempts to predict the probabilities of different classes by using the sigmoid function. As this is a multiclass problem, the softmax function is utilised to compute probabilities across functions, with the model selecting the class with the highest probability. Furthermore, MLR is easy to interpret, so I can later see what the model classifies well and what it struggles on.

1.1.2 Preprocessing Steps and Features

Preprocessing Steps:

- **Emoji Normalisation:** Given the dataset is that of tweets, there will be many emoji's used throughout. The classification models cannot take these as a useful input. I used the `emoji.demojize` method to convert these emoji's into textual descriptions of themselves, meaning emotional content contained by an emoji is not lost.
- **Expand Contractions:** I expanded contractions (e.g., "won't" to "will not") using the `contractions` library for better clarity [2].
- **Noise reduction:** The tweets contain URLs, emails, usernames, phone numbers and more. I replace all of these with placeholder regular expressions to reduce noise but retain the fact they are featured in the text. This ensures meaning isn't lost.
- **Punctuation Replacement:** I replaced significant punctuation marks like "!" and "?" with tokens to maintain emotional intensity.
- **Tokenisation:** I split the text into individual tokens using `nltk.word_tokenize` [2].
- **Slang Tagging:** As tweets will employ a lot of vernacular and slang, I created a small dictionary of common slang abbreviations such as 'lol', which are frequently used to convey meaning and replaced them with standardised tokens, making informal language easier to analyse.
- **Negation Handling:** I combined negation terms ("not" and "n't") with the following word for context, allowing the model to differentiate negated sentiments.
- **Lemmatisation with Part-of-Speech (POS) tagging:** I used the WordNet Lemmatizer alongside `pos_tag` [2] to firstly assign each word a tag based on its grammatical category, such as "noun" or "verb"; and then parse this to the lemmatiser to reduce words to their base form. The POS tagger improves the accuracy of the lemmatiser.
- **Stopword Removal:** I removed stopwords (words like "the," "a") using a custom list excluding "no" and "not", which are important for negation.

Feature Extraction:

- **TF-IDF Vectorisation:** Instead of using CountVectoriser and a traditional bag-of-words representation, I experimented with TF-IDF. This method calculates the importance of terms by computing their Term Frequency-Inverse Document Frequency score, which is helpful in distinguishing text classes. I.e it measures how important a term is in the document, relative to the whole vocabulary.
- **Sentiment Scores:** Using the VADER package [3], I also computed sentiment scores for the vocabulary. This adds a further layer of emotional context to the dataset.

1.1.3 Rationale and Hypothesised Effects

Many of the preprocessing steps, such as stop-word removal, converting all text to lower case, tokenising impoortant punctuation and expanding conractions, are common to text classification tasks. They clean the corpus and ensure the data is represented in a better way for classification. The steps I have added, including emoji tokenisation, lemmatisation with POS tagging and slang tokenisation, give another layer of emotional complexity to the data. This should ensure more meaningful relationships are located in the text by the classifier. Using TF-IDF with sentiment scores, should mean that particular emotions are successfully distnguished in the corpus. For example, unique terms associated with 'joy' would hopefully be assigned positive scores.

1.2

I opted for an LSTM-based classifier using pre-trained GloVe word embeddings [4]. LSTMs, a type of recurrent neural network (RNN), are particularly good at understanding long-term dependencies in sequential data. They use memory cells to analyze data in both directions, which helps in recognizing patterns in the emotional sequences of tweets. GloVe, which stands for Global Vectors for Word Representation, is a word embedding model pre-trained on Twitter data. It captures word relationships through co-occurrence statistics, making it a perfect match for this Twitter-based task.

1.2.1 Model Architecture

The data underwent the same preprocessing as it did for MLR. This is because I made purposeful preprocessing choices to impact the data and it would be an unfair comparison

between models if the data was not the same. The LSTM model architecture includes:

- **Embedding Layer:** Pre-trained GloVe embeddings with 50 dimensions, this was found to be more optimal than 25 dimensions as it better captured the corpus features.
- **LSTM Layer:** A bidirectional LSTM with 2 layers, each containing 35 hidden units. Increasing the number of the hidden layers to two, meant the LSTM could learn deeper hierarchical representations. Furthermore, each hidden layer having size of 35 also allows the model to learn more complex features. I found that increasing these dimensions any further, led to increased overfitting.
- **Fully Connected Layer:** Linear layer with 2×35 input features for the final classification.
- **Activation:** ReLU (Rectified Linear Unit) activation, to introduce non-linearity and to prevent the vanishing gradient problem.

1.2.2 Transfer Learning

The approach leverages the pre-trained twitter GloVe embeddings, which are learned from large corpora of tweets. This provides a strong initial understanding of word semantics, reducing training time and improving generalisation.

1.2.3 Preprocessing

On top of the preprocessing steps for MLR I padded or truncated each sequence to a fixed length of 20 tokens. After an exploratory histogram of sequence lengths, this was found to be a good length.

1.2.4 Training and Evaluation

The model was trained for 20 epochs (any more saw a convergence in validation accuracy and increase in loss), using cross-entropy loss and the Adam optimizer with a learning rate of 0.0005. The training batch size was set to 64.

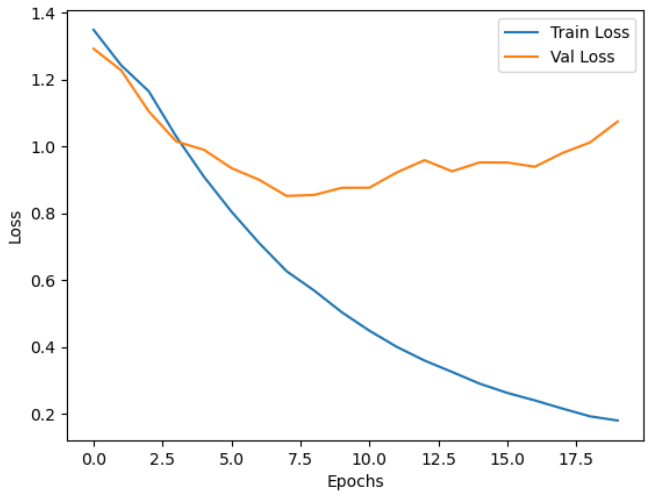

Fig. 1. Learning Curves for the LSTM model

Learning Curves

Figure 3 illustrates the training and validation loss over 20 epochs. The training loss decreases steadily, showing that the model fits well to the training data. The validation loss decreases to a trough at epoch 7 and then starts increasing slowly, indicating overfitting. The highest validation accuracy was achieved at epoch 17 (66.84%), which suggests that the model's optimal performance is at that point. This discrepancy between epochs could be explained by the fact that whilst the validation loss starts increasing due to overfitting, the model may still maintain or increase accuracy for a few epochs because it's getting more confident about correctly classified samples. The loss function is influenced by the certainty of predictions, unlike accuracy, which remains unaffected by it. For example, a prediction with high confidence that turns out to be incorrect could have a substantial impact on the loss function while only minimally altering accuracy.

This information can be used to add in early stopping, such that the model stops training once the loss/accuracy does not continue decreasing. We could also save the model with the best overall performance and use that to make predictions.

I did experiment with adding dropout for regularisation and early stopping, in an attempt to prevent over-fitting but this decreased the results significantly, even with increased epochs for convergence. This suggests the model is simple enough as is, and adding regularisation inhibits the training procedure from learning meaningful patterns.

1.3

To measure performance I will be using accuracy, precision, recall and f1-score.

1.3.1 Performance Metrics

- **Accuracy:** The proportion of correctly classified instances over the total instances.

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

- **Precision:** Of instances predicted positive, how many are really positive.

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

- **Recall:** Of instances actually positive, how many were correctly predicted positive.

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

- **F1-Score:** Harmonic mean of precision and recall.

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

Where:

- TP = True Positive
- TN = True Negative
- FP = False Positive
- FN = False Negative

TABLE I
TOP INFLUENTIAL FEATURES FOR EACH CLASS

Class	Positive Features	Negative Features
Anger	angry, f*ck, rag, fume, rage	sad, sadness, depression, worry, depress
Joy	hilarious, slang_lol, shake, horror, face_with_tears_of_joy	nightmare, optimism, never, right, like
Optimism	worry, fear, optimism, start, quote	user, questionmark, sad, like, face_with_tears_of_joy
Sadness	sad, depression, depress, sadness, sadly	f*ck, angry, anger, fume, furious

Accuracy can be misleading in imbalanced datasets, as a high score may simply reflect the model's ability to predict the dominant class while ignoring minority classes. Hence, precision and recall are used to contrast and verify the accuracy. However, they do not come without flaws. Precision can lead to overconfidence if not balanced with recall, especially in cases where false negatives are common. Recall, on the other hand, can lead to an inflated view of performance if many false positives occur. Thus, we evaluate both metrics together through their harmonic mean: F1-score. I also provide confusion matrices to see where the classifier is going wrong.

1.3.2 Testing Procedure

I split the dataset into training, validation, and testing sets. The training set was used to train both models. The validation set was used to tune hyperparameters. This meant I could adjust the models without causing the model to learn on the test set. All processing steps were applied to each training, validation and test sets equally; to ensure a fair test when classifying. Applying the processing to the split sets individually instead of the combined dataset, also ensured no data leakage.

1.3.3 Results

The MLR outperformed the LSTM model. The results for both are shown in Table's II & III, respectively. Whilst MLR performed well overall, the recall score for class 2 (optimism) stands out. The score of 0.14 suggest just over 1/10 true positives is actually identified. This is very poor performance, that I attribute to the fact that class 2 is the most underrepresented in the training data; with the following split: 'optimism': 294, 'anger': 1400, 'joy': 708, 'sadness': 855. This shows a clear class imbalance.

Table I, shows the most important features for distnguishing the classes, as designated by the MLR model. What is very clear is that for classes 0,1 and 3, the main positive and negative features clearly distinguish between the emotions. For class 2, the words do not tend to be synonymous with optimism, with postitve features including "worry", and "fear". This gives clear insight into where the classifiifier tripped up.

TABLE II
MLR CLASSIFICATION REPORT

	Precision	Recall	F1-Score
0	0.74	0.89	0.81
1	0.73	0.77	0.75
2	0.67	0.14	0.24
3	0.74	0.61	0.67
Accuracy			0.74

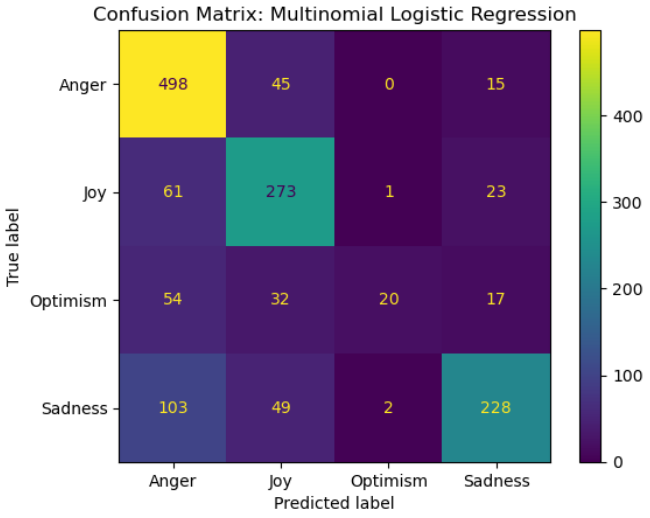

Fig. 2. MLR confusion matrix, showing that classes 'Anger', 'Joy' and 'Sadness' performed similarly well, but 'Optimism' performed poorly.

The LSTM model did not perform as well as the MLR. It also struggled to predict class 2, with low precision suggesting the model over-predicts positive cases. To delve deeper into where the models struggle, beyond the obvious class imbalance, I looked at examples of misclassified tweets.

TABLE III
LSTM MODEL CLASSIFICATION REPORT

	Precision	Recall	F1-Score
0	0.80	0.76	0.78
1	0.75	0.57	0.65
2	0.34	0.51	0.41
3	0.65	0.73	0.69
Accuracy			0.68

Misclassified Tweets

On analysing the misclassified tweets from both classifiers, there are some common trends and useful observations that could be used to refine the training process.

1) **Ambiguous Sentiments**: Many sentences express a mix of emotions, such as sadness with optimism or joy with fear. For instance, phrases like "sweet merciful crap bad singing" may contain sarcasm that can be interpreted differently. As the classifier will become confused by the use of "sweet", "merciful", and "bad" all together. This can explain why this tweet labelled as "joy" was classified as "anger".
2) **Contextual Errors**: Some errors occur due to a lack of external context. Tweets or sentences referencing

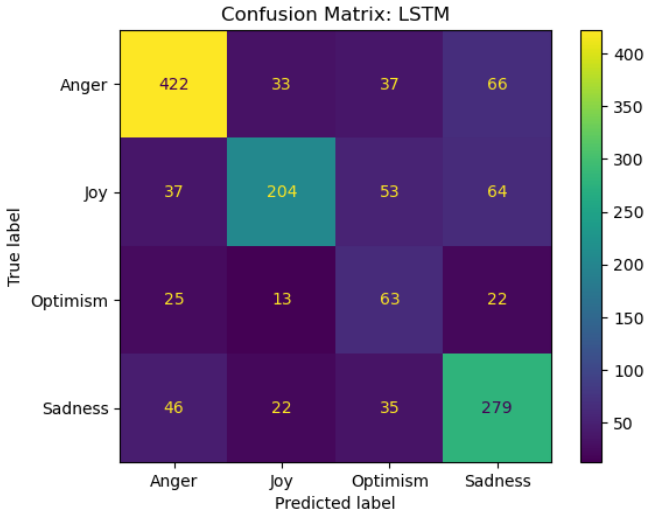

Fig. 3. LSTM model confusion matrix

specific events or cultural phenomena can lead to misclassification without additional context. For instance, the tweet, "@user @user Christy is a bigger piece of sh*t then trump is and a bully", is reffering to Donald Trump. However, "trump" being used in the lowercase, gives the words an actual definition, completely seperate to that of the proper noun, 'Trump'. This lead the classifier to predict 'sadness', when in fact the tweet is displaying 'anger'.

3) **Sarcasm**: Sarcastic or ironic statements are often misclassified because sarcasm detection is a complex task requiring nuanced understanding and very specific preprocessing steps.
4) **Examples of Optimisim misclassification**: extending on my earlier analysis, 'optimism' was the most misclassifed class. For example, the classifier labelled this tweet as optimism: "Would love to stop crying sometime today, on my tenth cry (emoji) deep sadness on top of food poisoning do not mix. miserable". As shown by Table I, where it clearly exhibits 'sadness'.

Improvements

1) **Language models**:Sarcasm detection is a pressing issue in text-classification. Whilst simpler classification models have been shown to struggle, pre-trained language models such as BERT have shown promising results for sarcasm identification [5].
2) **More Training Data**: More training data could be colleceted to give richer examples. This would particularly benefit the 'optimism' class.
3) **More classes:** "Why is an alarm clock going 'off' when it actually turns on? #alarm #alarmclock #ThursdayThoughts", is an example of a tweet displaying no strong emotions. This rhetorical question could indicate inquisitiveness or simple neutrality, yet it's labelled as anger and was predicted as sadness. When the tweet istelf does not seem to fit fully into its labelled category, this makes the task much harder for the classfier, as the discerning features for that category are not there. Therefore, the data should include more emotional categories, or simply a category for neutrality.
4) **Emoji's**: Whilst I hypothesised that tokenising emoji's would lead to better emotional sentiment, it seemingly did the opposite. From analysing many tweets I found that emoji's are often used ironically, or rather in a tone not in keeping with the rest of the text. For example, "Every day I dread doing an 8 hour shift in retail, (smile emoji)", is rightfully labelled as sadness. However, the ironic use of the smiling emoji, confuses the classifier to label it as 'optimism'. Thus, more care needs to be taken in handling emoji's.

One interesting thing to note is the difference in embedding steps. Whilst both the MLR model and the LSTM model had the same preprocessing, the MLR was embedded using TF-IDF and the LSTM with GloVe-50. If we take a common word such as 'anger' and calculate the cosine similarity between it and other words, we can find the words that the embedding technqiues find to be most similar. As shown in Table IV, the two models yield different sets of similar words to "anger". The GloVe model associates "anger" with other emotions and related feelings. In contrast, the MLR model provides words that seem more contextually diverse, including terms related to emotions, nationalities, and colloquial expressions. This could explain the difference in classification abilities, as MLR has better contextual similarities.

TABLE IV
COSINE SIMILARITIES OF 'ANGER' FOR GLOVE AND TF-IDF

GloVe Word	GloVe Score	MLR Word	MLR Score
frustration	0.8830	book	0.9998
emotions	0.8775	racism	0.9994
hatred	0.8408	pakistan	0.9949
sadness	0.8311	stupid	0.9945
jealousy	0.8253	resent	0.9936
fear	0.8237	shoot	0.9933
emotion	0.8184	fan	0.9924
feelings	0.8085	rise	0.9923
guilt	0.8039	answer	0.9912
grief	0.8014	fury	0.9901

1 1.4 Topic Modelling

1.4.1 Preprocessing

I did a run through with little preprocessing, in order to see any flaws in the topics, so I could eventually optimise the output. This led to adding custom stop words ("user" and "amp") to a comprehensive stop word dictionary. Each text is tokenized and lemmatized, excluding stop words, to generate preprocessed tokens. A model is trained to detect and form bigrams, which are then used to build a dictionary mapping words to unique IDs. Finally, the bigrammed texts are converted into a bag-of-words corpus for LDA.

I also set up another dataset, whereby the data was first filtered to only contain tweets labelled as 'optimistic' or 'joyful', to see if this would achieve clearer topics.

Methodology

I implemented both Latent Dirichlet Allocation (LDA) and Hierarchical Dirichlet Process (HDP). As LDA requires input for the number of topics to model, I thought a good starting place would be to employ HDP. HDP allows the number of topics to grow, to fit the dataset. Hence, no input for number of topics is required. I set up the HDP model and found that it fit over 140 topics. This was far too many to analyse. Furthermore, the distribution of topics throughout the dataset was almost constant. I adjusted the alpha and gamma parameters and still found over 140 topics, with a constant distribution. I ,therefore, turned my attention to LDA.

Starting with four topics as per the four emotion labels, I did not find meaningful topics. There were no clear themes and lots of words between topics overlapped. I, therefore, implemented coherence score, to find the optimal number of topics with the highest coherence score. I used C_V coherence score, which measures the semantic similarity of top words in each topic using cosine similarity to compute the co-occurrence probabilities between word pairs. The scores fall between 0 and 1, with scores closer to 1, indicating that words within a topic are more likely to co-occur, suggesting a more meaningful and interpretable topic [6].

This was achieved through looping over each number of topics. I looped up until 100 to obseverve whether HDP's etsimate was reasonable.

I found that the coherence score increased to a peak at 40 topics(score = 0.5896932684313372) before a plateau until 70 topics occurred. After this point the coherence score began to decrease. I then implemented LDA with 30 topics, as although the coherence score was lower (score = 0.5784703164014809) the difference was marginal and in keeping with the plateau, shown in Figure 4. Less topics also made for easier analysis.

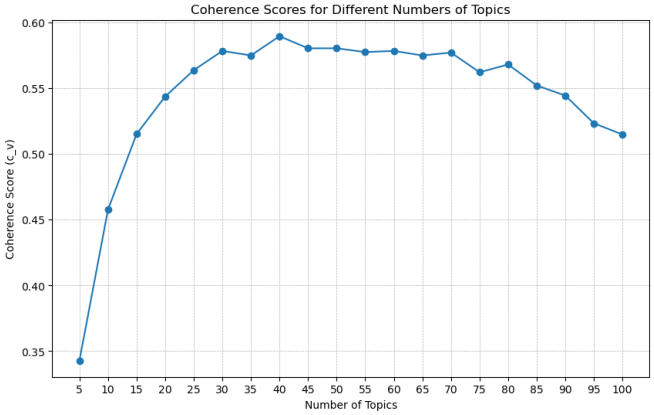

Fig. 4. LSTM model confusion matrix

Of the 30 topics it was next prudent to identify which ones were 'optimistic' or 'joyful'. I did this in three ways. Firstly, I examined the topics myself and used my own judgement to discern between positive and negative topics. Secondly, I analysed the sentiments of the topics, scoring them positively or negatively using the TextBlob package [7]. Thirdly, I created a custom lexicon of the words: "happy", "joy", "love", "cheer", "optimistic", "great", "good", "celebrate", "smile", "excited", and graded topics positive or negative based on these words featuring in them.

These three approaches gave me a common group of topics, seen to be positive. Topics of note are as follows:

1) **Social Interaction**:
   - **Theme**: Words like *like*, *look*, *cheer*, *love*, *people*, hint at a theme surrounding social interactions and positive emotions.
2) **Media and Entertainment**:
   - **Theme**: These topics have a lot of overlap, including words like *lol*, *broadcast_musically*, *live_ly*, *music*, *media*, *omg*, *watch_amaze*, indicating a positive outlook towards entertainment and media.
3) **Birthday Celebration**:
   - **Theme**: With words like *shock*, *laughter*, *love*, and *birthday*, this topic blends surprise and joyous celebrations.
4) **Social Media (Snapchat)**:
   - **Theme**: Keywords like *think*, *snap*, *best*,*snapchat*, *night* and *experience* suggest people are optimistic about their nights and interactions on social media.
5) **Social Change**
   - **Theme**: Contains terms like *state*, *mind*, *change*, *feel*, *good* and *unga*, suggesting optimism about social improvements, possibly at the UN general assembly.
6) **Overcoming Challenges**
   - **Theme**: Words like *smile*, *anxiety*, and *start* imply positivity and hope despite challenges.

Using multidimensional scaling to visualise the topics, we can see which topics the dimensionality reduction finds to be distinct and which ones share attributes. Figure 5, shows that whilst there are distinct categories, the majority of Topics have considerable crossover.

Shortcomings of approach

1) **Preprocessing**: The preprocessing also has a major effect on LDA. For example, I have removed common stop words and further stopwords such as "user". Whilst this was done to reduce noise, some underlying relationships might be lost with this cleaning.
2) **Bag-of-Words**: LDA requires a bag-of-words representation. This means only the frequency of the word is encoded and powerful techniques like TF-IDF and word2vec are not utilised.

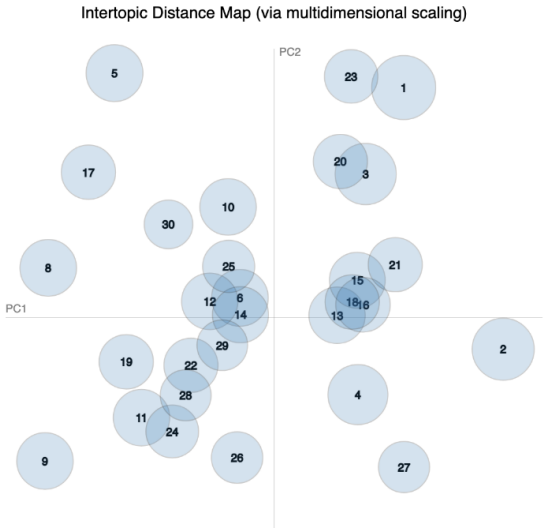

Fig. 5. Multidimensional Scaling of LDA Topics

3) **Difficulty in Interpreting Topics**: While LDA identifies topics, interpreting these topics can be subjective and even the techniques I have enlisted to help, such as sentiment scoring and a custom lexicon, still leave a lot of room for improvement; as they go off individual words and not whole semantic phrases.
4) **Limited to Unsupervised Learning**: Finally, LDA is an unsupervised learning algorithm, meaning it does not leverage labeled data for training. For this reason I extended the approach to filter the dataset to only those tweets labelled joyful or optimistic.

After filtering the dataset to tweets only labelled as joyful or optimistic, I conducted the same LDA analysis. Whilst many of the topics included generic words associated with joy and optimism, and did not have discernible themes, themes that again emerged included those expressing gratitude (”thank”), excitement about watching live broadcasts (”watch_amaze,” ”live_ly”), and joy in starting something new (”smile,” ”start”).

Task 2

I did not want to pass up the opportunity to experiment with transformers, such as BERT. Therefore, to build my named entity recognition tool, I have implemented a mini-BERT model. To evaluate the effectiveness of the mini-BERT model, I have also implemented traditional CRF with POS-tagging.

2.1.1

The method involves using a pre-trained model checkpoint, ”huawei-noah/TinyBERT_General_4L_312D”) [8] to fine-tune for Named Entity Recognition (NER). Specifically, the approach uses the AutoModelForTokenClassification from the Transformers library to perform token classification. This is then trained with the Trainer class, where metrics like F1-score are computed. One advantage of using the BERT tranformer, is the fact it is pretrained on a large-scale and can, therefore, identify complex language patterns. CRF is highly dependent on the initial pre-processing and feature engineering, and hence may perform worse than BERT, if steps have been missed in feature engineering. Furthermore, BERT captures bidirectional context in a text, through masked language modelling. This means, understanding the meaning of each word in relation to both the words that come before it and those that come after it. The main downfall of BERT are the computational demands. It takes very long to run and is very computationally expensive; and given the limited power of my machine, I can’t fully utilise its full potential, hence the use of the more compact tiny-BERT.

2.1.2 BERT tokenizer

BERT uses its own tokenizer, converting the original tokens, which can be words, into ’sub-words’. To maintain the relationship between the original token and the sub-word, BERT retains a mapping between the two, through ’word_ids’. In the alignment phase, each sub-word receives a label corresponding to its parent word. The assignment of these labels is controlled by the label_all_tokens flag. If set to True, all sub-words inherit the original word’s tag; otherwise, only the first sub-word receives the tag while the remaining sub-words are given the label ’-100’ to be ignored during the loss calculation. I experimented with these settings and found setting the flag to false, decreased the computational complexity and did not affect the results of entity recognition, as the dataset has a sufficient number of tokens.

Entity Spans

The model is trained to predict the entity spans. These are represented using a BIO scheme, whereby tokens that represent the beginning of an entity are given a ”B-” prefix followed by the entity type, such as ”B-Chemical” for the beginning of a chemical entity. Tokens that continue inside the entity span receive the ”I-” prefix, like I-Disease for an entity related to a disease. Tokens that don’t belong to any entity are tagged as O for ”Outside.”

2.1.3 Feature Selection

BERT: as aforementioned, BERT is bidirectional, so it considers the words before and after a token, when generating a specific token. Furthermore, to ensure that the model can distinguish the order of tokens in a sequence, BERT introduces positional embedding that add information about the relative or absolute positions of tokens. I also implemented the Optuna library to conduct a comprehensive hyperparameter search [9]. This was a very lengthy process so I only searched for two key hyperparameters, learning rate and batch size. The search with 5 trials returned learning rate = $5.71x10^{-}5e-05$ and batch size = 8.

CRF+POS: Firstly, I employ capitalisation do distinguish between capitalised and lower-case words (tokens), as proper nouns are often capitalised. I also use HAS_NUM to identify whether a number is present in a token. Furthermore, like BERT I employ a bidirectional approach, whereby the words

neighbouring a token are considered as features, an attempt to give more context. Finally, punctuation patterns and suffixes are extracted, this helps to identify abbreviations and special language use-cases. On-top of this, I use POS-tagging. This helps the CRF model infer better language structure, by tagging different parts of speech, such as nouns and verbs.

2.2.1 Evaluation

To evaluate the different methods I will be comparing their precision, recall and f-1 score. I will not be using accuracy. Accuracy is not ideal for evaluating tasks like NER because most tokens are often non-entities (O in BIO tagging), leading to a class imbalance. This can cause a model to achieve a high accuracy score by predicting non-entities correctly while missing actual entities. Furthermore, NER requires sequence-wide evaluation rather than evaluating individual tokens in isolation. Precision, recall, and F1 scores are calculated for complete spans, whereas accuracy is not.

Much like the previous task, the data was split into train, validation and test sets. This ensured I could train the models and then tweak performance, using the validation set to test changes. This prevents the model from seeing the test set and over fitting.

The results in Table V, show that whilst BERT in conjunction with CRF and POS-tagging, improved the results, this improvement was very minimal, only improving the recall for "Disease". This, therefore, suggests that the BERT model is already complex enough, already capturing contextual relationships well, which may overlap with the information POS tagging provides. Therefore, adding POS tags as features may not significantly contribute. I also compared this to a baseline of CRF without a transformer. My results show that simple CRF using default parameters is outperformed by a CRF with the additional features and POS-tagging (Table VI). However, BERT outperforms this quite significantly, indicating its implementation is worthwhile.

TABLE V
BERT VS. BERT+CRF+POS RESULTS

Model	Precision	Recall	F1 Score	Support
BERT				
Chemical	0.97	0.97	0.97	103,747
Disease	0.78	0.80	0.79	7,121
BERT+CRF+POS				
Chemical	0.97	0.97	0.97	103,747
Disease	0.86	0.88	0.87	7,121

From all sets of models it is clear to see that 'Chemicals' are labelled more accurately than 'Diseases'. To find out why this might be the case, I looked at examples from the CRF-with-additional-features-and-POS-tagging model, of mislabelled instances.

1. Under-prediction of Disease Entities

A common trend amongst the mislabbeled data was the under-prediction of Disease labels. For example:

TABLE VI
CRF VS CRF+FEATURES+POS RESULTS

Model	F1 Score	Macro-average F1
CRF+Features+POS		
Chemical	0.84	
Disease	0.69	0.76
CRF		
Chemical	0.83	
Disease	0.65	0.74

1) **Gold Standard Labels:**

   [('Tricuspid', 'B-Disease'), ('valve', 'I-Disease'), ('regurgitation', 'I-Disease')

   Predicted Labels:

   [('Tricuspid', 'O'), ('valve', 'O'), ('regurgitation', 'O')

2) **Gold Standard labels:**

   ('congenital', 'B-Disease')

   Predicted labels

   ('congenital', 'O')

From these examples, I believe that one problem the model is having, is when diseases are comprised of words with standard meanings, outside-of being used for diseases. For example, "tricuspid valve regurgitation", does not immediately seem like a disease but instead a bodily function. Furthermore, 'congenital', simply means a trait from birth. In this context it was followed by "heart disease", which the model labelled correctly. Further examples of this includes the word "ulcers" being misclassified, as without specific context it is not necessarily a disease. It could also be that the feature dealing with multi-word entities, is not performing correctly and may need refining.

Further Observations

The model struggles with acronyms, as shown in the following examples. "IDM" should be B-Chemical but was labeled as O. "PRA" was misclassified as B-Chemical instead of O. "SRC" was incorrectly classified as O rather than B-Disease. "SSc" was misclassified as O instead of B-Disease. Acronyms are ambiguous without domain specific knowledge, therefore they are prone to mis-classification as they rely on broader context.

One way of solving these issues is to add in more features to the CRF model tuning. For example, a lexicon could be added of common acronyms, such that they can be processed and split into their root words. However, this would require domain specific knowledge. One could also apply heuristic rules as a post-processing step for entities that frequently exhibit confusion.

2.3

Now, I compare two techniques for measuring similarity between diseases using "sepsis" as the query term: Word2Vec with cosine similarity [10], and Levenshtein distance. The re-

TABLE VII
MOST AND LEAST SIMILAR DISEASES TO 'SEPSIS' (WORD2VEC AND LEVENSHTEIN)

Rank	Word2Vec (Disease)	Cosine Similarity	Levenshtein (Disease)	Edit Distance
Most Similar:				
1	rosacea	0.9872	emesis	3
2	lethargy	0.9864	stenosis	3
3	hemiparesis	0.9863	Nephrosis	4
4	psychoses	0.9861	cysts	4
5	hypomania	0.9846	gliosis	4
Least Similar:				
-5	cancer	0.6881	Syndrome of inappropriate secretion of antidiuretic hormone	55
-4	nephropathy	0.6696	syndrome of inappropriate secretion of anti-diuretic hormone	55
-3	hyperactivity	0.6596	Antineutrophil cytoplasmic antibody (ANCA)–associated vasculitis	60
-2	seizures	0.6538	abnormalities in the cortex, hippocampus, white matter, and ventricles	64
-1	catalepsy	0.5927	impairments in learning, attention, inhibitory control, and arousal regulation	74

sults highlight the strengths and differences between semantic and string-based comparisons.

Word2Vec learns dense vector representations of words through the Skip-gram model, enabling words with similar contexts to have similar vector embeddings. The Cosine similarity between these two vector embeddings, $\mathbf{A}$ and $\mathbf{B}$ is calculated as:

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|},$$

Levenshtein distance is the minimum number of single-character edits (insertions, deletions, or substitutions) required to change one string into another.

Analysis

Table VII, shows the 5 most similar and disimilar diseases to 'sepsis', excluding itself. Word2Vec finds words like "rosacea," "lethargy," and "hemiparesis" to be most similar to "sepsis" due to their shared context in medical conditions. Whereas, diseases like "cancer," "seizures," and "catalepsy" share little semantic similarity with "sepsis," possibly due to different medical associations.

Levenshtein distance finds diseases like "emesis" and "stenosis" most similar to "sepsis" due to low edit distances. Whilst complex conditions like "Antineutrophil cytoplasmic antibody (ANCA)–associated vasculitis" are the least similar due to their length and distinct character sequences.

Overall, this shows that Word2Vec embeddings capture far greater contextual meaning from the text, compared to Levenshtein. Using edit distance as a similarity metric is a crude approach, as similar diseases will not necessarily share similar names and spellings.

References

[1] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[2] Steven Bird, Ewan Klein, and Edward Loper. Natural language toolkit, 2009. Version 3.5.

[3] C.J. Hutto and Eric Gilbert. Vader: A parsimonious rule-based model for sentiment analysis of social media text. In *Eighth International Conference on Weblogs and Social Media (ICWSM-14)*, 2014.

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

[5] Christopher Ifeanyi Eke, Azah Anir Norman, and Liyana Shuib. Context-based feature technique for sarcasm identification in benchmark datasets using deep learning and bert model. *IEEE Access*, 9:48501–48518, 2021.

[6] Emily Rijken. C topic coherence explained, 2023. Accessed: 2024-05-08.

[7] Steven Loria. Textblob: Simplified text processing, 2018. Version 0.15.3.

[8] Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xinxiong Chen, Linlin Li, Fang Wang, and Qun Liu. Tinybert: Distilling bert for natural language understanding. *arXiv preprint arXiv:1909.10351*, 2019.

[9] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2623–2631, 2019.

[10] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2013.