ECS8054 Final Assignment

Hugo Collins
Student Number: 40446925
Email: hcollins05@qb.ac.uk
MSc in Artificial Intelligence 2024

Word Count (Excluding Front Page, References, Table of Contents, Figure Titles, Titles and Tables): 3282

Abstract

This report analyses three different aspects of natural language processing - Text processing and linguistic insights, Text Classification and Sentiment Analysis. In this exploratory report, I experiment with numerous approaches to these tasks, analysing the results in great detail. The data used for this project is a set of 601 short stories created by OpenAI's GPT-4 model [7]. The report highlights the effectiveness of BERT for text classification, comparing performance to the static word vectors from Word2Vec. It also explores dependency parsing, revealing that more complex sentence structures lead to longer dependency spans, while poetic and journalistic styles tend to have shorter spans despite their complexity. Additionally, I analyse part-of-speech (POS) tags, showing distinct distributions across themes. Finally, the report addresses the limitations of averaging emotion vectors for sentiment analysis, which may oversimplify emotional content and looks at BERT for sentiment analysis.

Section 1.1

The first question I researched was whether there are differences in linguistic structure for the five categories of stories based on their writing style. As humans, we naturally are able to create different sentence structures and word choices based on our audience or intended purpose but for our large langauge models they rely solely on lingustic patterns. To better understand these patterns I wanted to analsye the five styles of the AI generated stories which are described in the brief as seen below:

- *"descriptive"*: Long sentences, rich modifiers.
- *"concise"*: Shorter sentences, fewer modifiers.
- *"poetic"*: Complex sentence structures, metaphorical language.
- *"journalistic"*: Neutral, declarative sentences with medium complexity.
- *"for children"*: Simple sentence structures, simple vocabulary.

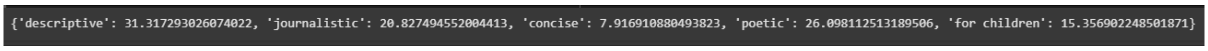
My main function was the longest_dependency_span, which took a single sentence as input and computed the longest distance between any token and its syntactic head (longest dependency span). I was able to do this using SpaCy's token.i (the index of the token) and token.head.i (the index of the token's syntactic head).[4]

I subtracted the token index from the index of its parent "head" and then took the longest span to represent that sentence. For each story, I got the average of these maximum spans, grouping them by writing style. The final results were stored in a dictionary, mapping each style to its average dependency span.

I plotted the longest depency span per story for each style using a raincloud plot to show their distribution by style.[6]

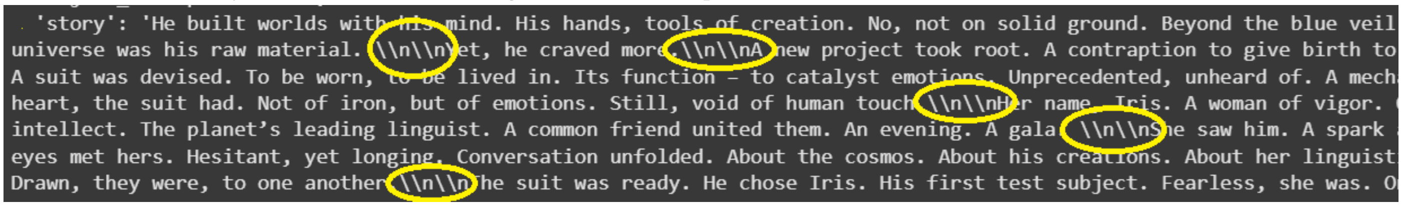
Data Preprocesing:

Originally, I inputted the data in its current form and got the following results:



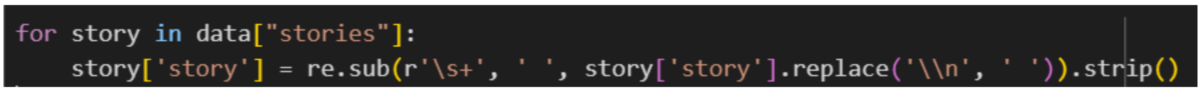
```
{'descriptive': 31.317293026074022, 'journalistic': 20.827494552004413, 'concise': 7.916910880493823, 'poetic': 26.098112513189506, 'for children': 15.356902248501871}
```

These figures seemed high, so I examined the parser more closely. I noticed lots of occurrences of "//n" at the end of many sentences in the "story" text, which was causing issues. For example:



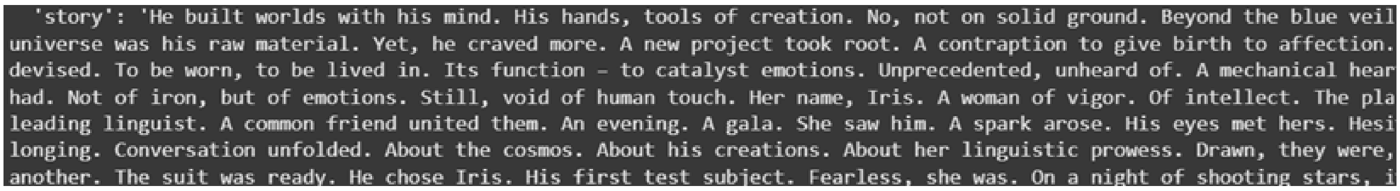
```
 'story': 'He built worlds with his mind. His hands, tools of creation. No, not on solid ground. Beyond the blue veil
universe was his raw material.\\n\\nYet, he craved more.\\n\\nA new project took root. A contraption to give birth to
A suit was devised. To be worn, to be lived in. Its function – to catalyst emotions. Unprecedented, unheard of. A mech
heart, the suit had. Not of iron, but of emotions. Still, void of human touch.\\n\\nHer name, Iris. A woman of vigor.
intellect. The planet's leading linguist. A common friend united them. An evening. A gala.\\n\\nShe saw him. A spark
eyes met hers. Hesitant, yet longing. Conversation unfolded. About the cosmos. About his creations. About her linguist
Drawn, they were, to one another.\\n\\nThe suit was ready. He chose Iris. His first test subject. Fearless, she was. O
```

To deal with this I used .replace() to replace with a whitespace. I then had the new issue of having a double whitespace so I used:



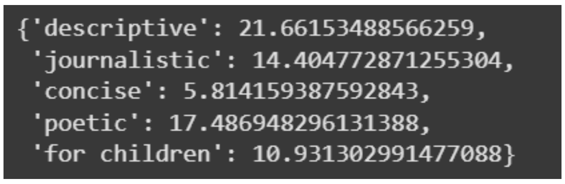
```python
for story in data["stories"]:
    story['story'] = re.sub(r'\s+', ' ', story['story'].replace('\\n', ' ')).strip()
```

This cleaned all the stories entries and allowed for accurate calculations of the longest dependency spans. The cleaned extract can be seen below:



```
  'story': 'He built worlds with his mind. His hands, tools of creation. No, not on solid ground. Beyond the blue veil
universe was his raw material. Yet, he craved more. A new project took root. A contraption to give birth to affection.
devised. To be worn, to be lived in. Its function – to catalyst emotions. Unprecedented, unheard of. A mechanical hear
had. Not of iron, but of emotions. Still, void of human touch. Her name, Iris. A woman of vigor. Of intellect. The pla
leading linguist. A common friend united them. An evening. A gala. She saw him. A spark arose. His eyes met hers. Hesi
longing. Conversation unfolded. About the cosmos. About his creations. About her linguistic prowess. Drawn, they were,
another. The suit was ready. He chose Iris. His first test subject. Fearless, she was. On a night of shooting stars, i
```

The resulting average longest dependencies by style were now:



```
{'descriptive': 21.66153488566259,
 'journalistic': 14.404772871255304,
 'concise': 5.814159387592843,
 'poetic': 17.486948296131388,
 'for children': 10.931302991477088}
```

The raincloud plot of the max sentence dependency span per style can also be seen below to get an understanding of the distribution.

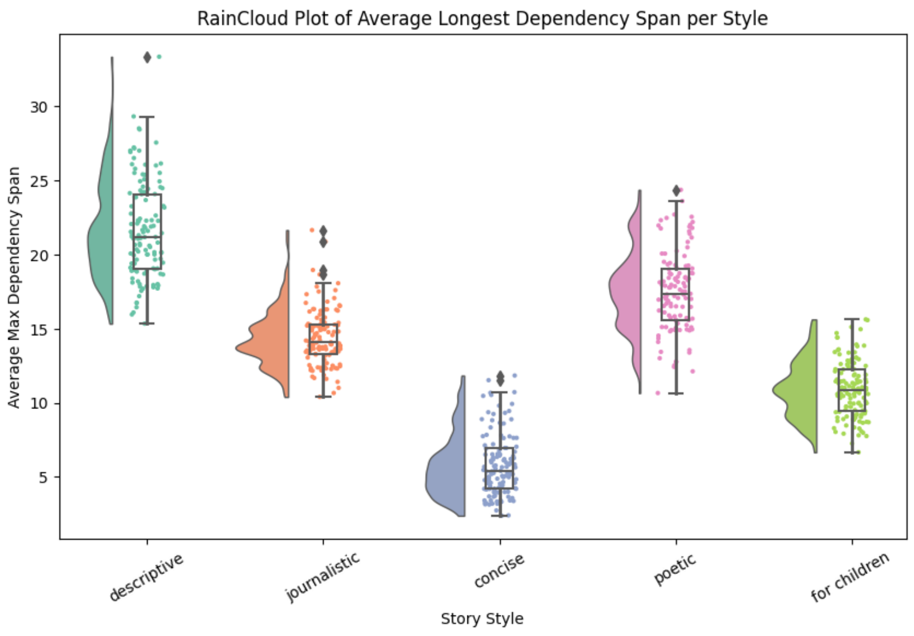


Fig. 1 *Raincloud plot of Average Longest Dependency Span per style*

Observations:

The overall takeaway from this analysis is that the longest dependency span is heavily correlated to the writing style, specifically the lingustic structure. The styles with more complex syntax such as “decriptive”, have longer average dependency spans than those of a “consise” or “for children style” which have simpler syntaxes. I wanted to examine why this is happening.

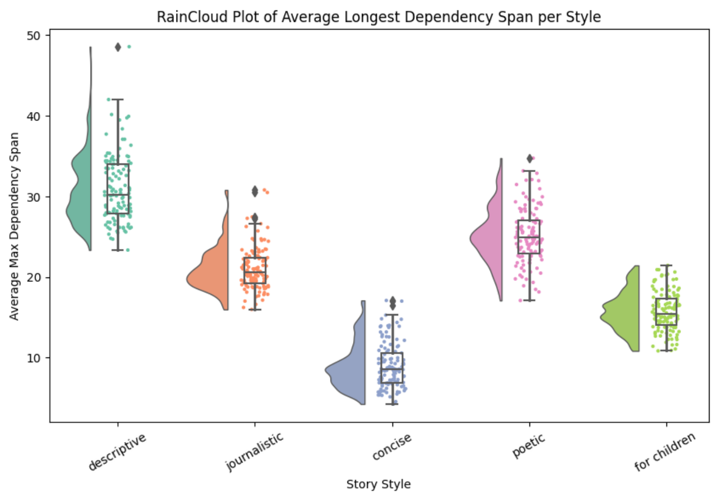
Descriptive vs Concise

Descriptive stories tend to have longer sentences with more nouns, verbs, and modifiers, which naturally leads to longer dependency spans. The more details and complexity in a sentence, the more relationships and dependencies it needs, making the structure more intricate. On the other hand, concise writing keeps things direct, avoiding long noun phrases and complex wording. With shorter, simpler sentences, there are fewer dependencies, which explains the lower average span (5.81).

Poetic Style (Complex Structure) vs Richer Modifiers

The results mostly aligned with expectations, but one surprise was the drop in average dependency spans from descriptive to poetic style. I initially expected poetic stories to have spans similar to descriptive ones, given their rich language and complex structures. However, the poetic style was actually closer to journalistic writing in terms of dependency span. While poetic language uses intricate vocabulary and structure, it seems it doesn’t always result in longer sentences or more dependencies. This suggested to me that adding rich modifiers has a bigger impact on length of dependency spans than sentence structure alone, explaining why the poetic style average span remained moderate (17.49).

Correlation between Sentence Length and Dependency Span



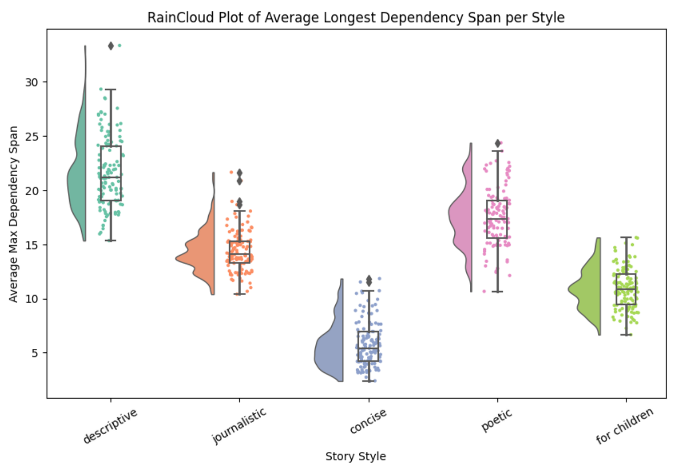


Fig 2 *Average Sentence Span by Style* **Fig 3** *Longest Dependency by style*

Looking at the correlation between sentence length and dependency span, a clear relationship emerges: longer sentences tend to have longer dependency spans. Also looking at the plots above, the both follow very similar distributions, with similar spreads and interquartile ranges. Across all the styles, the longest average dependencey span is roughly 65/70% of the average sentence length which is a constant theme regardless of style which I found interesting.

Challenges:

Originally I had been calculating the longest dependency incorrectly. I had been using spacy's parser and then using .head to check the syntactic parent of each token and then track each longest span manually.[4] However, this was way too complex and was actually adding 1 to the length of the actual span (e.g 25 instead of 24).

Section 1.2

For this section I focused on POS tagging. POS tagging is a very important aspect of NLP. It is the process which assigns grammatical labels to words and is very useful for understanding relationships between words and is important in semantic analysis. named entity recognition etc. [9] I decided to use nltk as it incorporated the Penn Treebank Tagset directly compared to SpaCy, which uses a slightly different tagset.[8] I performed the same preprossecing as above but before performing any tagging I considered a hypothesis.

My hypothesis: The frequency of each type of POS tag will differ based on the type of story

Reasoning:

I strongly believe that the frequency of POS tags will differ based on story types. Language—specifically, the choice of words and grammatical structures — plays a crucial role in shaping the tone, pacing, and meaning of a narrative. Different genres or themes rely on specific parts of speech more than others to convey their intended theme. The way a story is told directly influences the distribution of POS tags.

Action-packed stories tend to use more verbs (VB, VBD, VBG) and adverbs (RB, RBR) to create urgency, while love and descriptive stories focus more on adjectives (JJ, JJR) and symbolic language. Romance stories may have more personal pronouns (PRP), while discovery-based ones use more proper nouns (NNP). Children's stories likely feature simpler pronouns (PRP) and proper nouns (NNP), while adult stories use more complex nouns (NN), adjectives (JJ), and modal verbs (MD). Overall, the frequency of each POS tag should vary depending on the story type in my opinion.

To get the actual POS tags, I created a count_pos_tags_by_story() class, which used nltk.pos_tag to get the POS tags for each token in every story. Since there were too many tags to analyse directly, I saved the counts in a CSV file, which I attached in the submission. I then created a POStag() class to visualise the counts by story type, as shown below.

Actual splits by story:

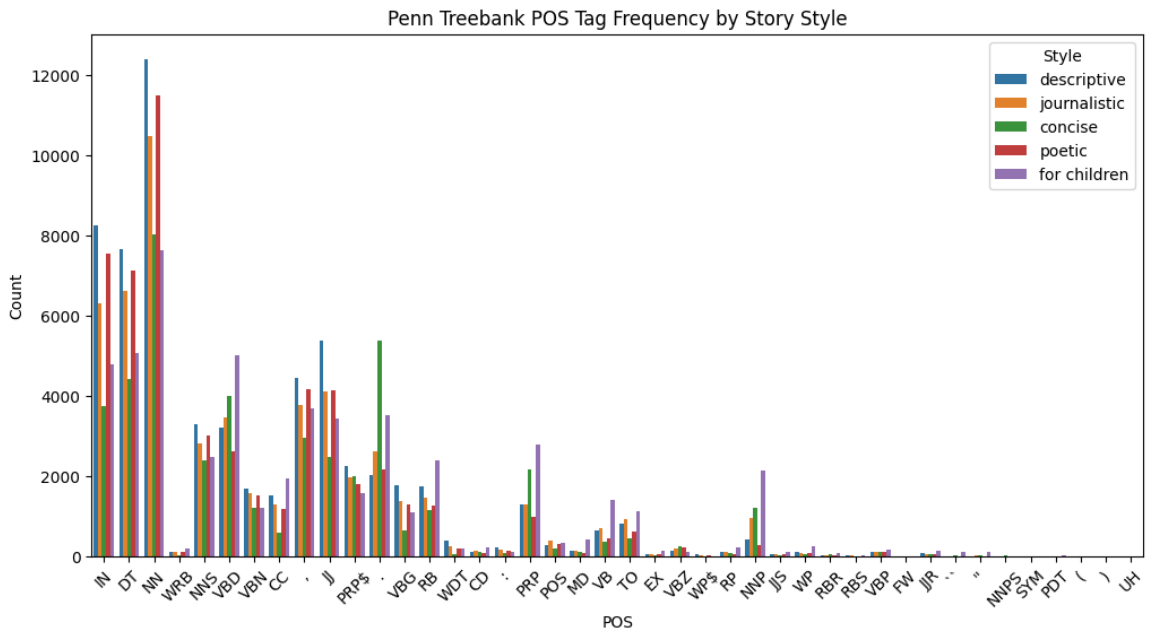


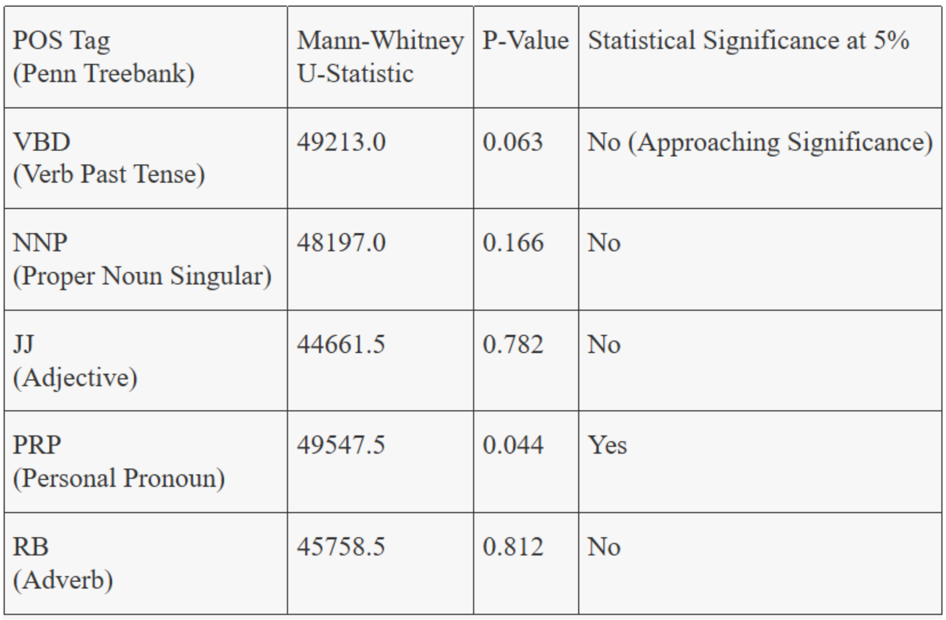
Fig. 4 *POS Tags count by Story Style*

Here we can see that there clearly is a fluctuaton in POS tags based on story type and that my above hypothesis stands.

The next step was to select five tags which I believed would likely show a difference between the“victory” and “defeat” categories. The five tags I chose can be seen below:

- **VBD (Verb, Past tense)** : I would expect more VBD in defeat stories to show actions leading to failure and more reflection whereas sucess might look to the future more etc.
- **RB (Adverbs)** Perhaps defeat stories may include more adverbs to convey sense of defeat.
- **VBZ (Verb, 3rd person singular present)** Victory may focus on more present verbs comapred to defeat
- **NNP (Proper noun, singular)** May change based on the focus of the story.
- **PRP (Personal pronoun)** Maybe one story type will focus more on the protagonist

Due to the count-based nature of my data, I found that a t-test might not meet all the necessary assumptions. I performed a normality check using the Shapiro-Wilk test and found that all but one variable were non-normal. Therefore, I chose the Mann-Whitney U test, a non-parametric method that doesn’t require the assumption of normality, making it more appropriate for my data.[2] Although this test doesn’t directly compare means, it detects differences in the distributions—which, for count data, effectively shows whether there is a significant difference between groups. The table below displays the results.



POS Tag (Penn Treebank)	Mann-Whitney U-Statistic	P-Value	Statistical Significance at 5%
VBD (Verb Past Tense)	49213.0	0.063	No (Approaching Significance)
NNP (Proper Noun Singular)	48197.0	0.166	No
JJ (Adjective)	44661.5	0.782	No
PRP (Personal Pronoun)	49547.5	0.044	Yes
RB (Adverb)	45758.5	0.812	No

Fig. 5 *Results of Mann-Whitney Test to check for Statistcial Difference Between Victory and Defeats for 5 POS Tags*

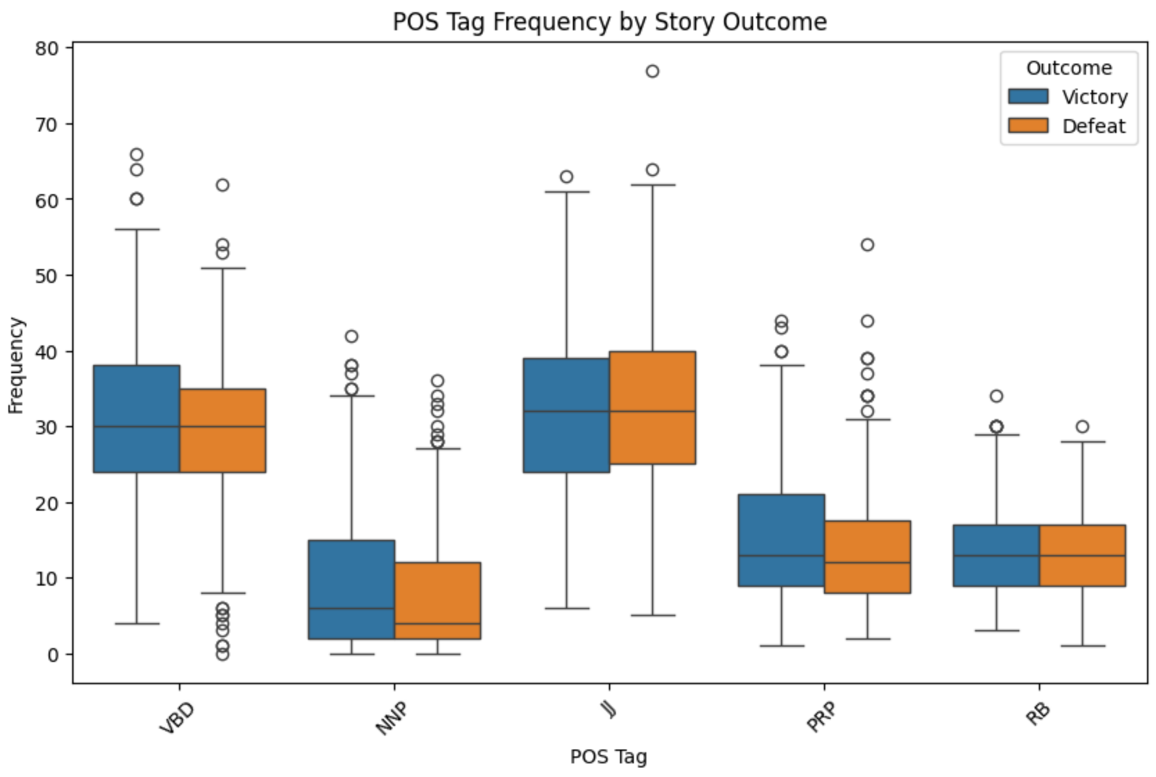


Fig. 6 *Box Plots of distributions of Counts*

These results showed that PRP (Pronouns) was the only POS tag with a statistically significant difference between victory and defeat outcomes. The boxplot suggested PRP had a slightly higher mean in victory stories but more outliers lead to the distributional difference. VBD was close to significance (p = 0.063), showing a possible difference in verb usage between outcomes. While not statistically confirmed, the boxplot suggested to me that it could be a solid indicator between victory and defeat labels. The other tags (NNP, JJ, RB) showed no meaningful differences. Adjectives appeared nearly identical across groups, with a high p-value, suggesting adjective usage was not a strong indicator.

The results for PRP suggest victory stories reference characters more frequently, while VBD's trend contradicted my oriinal thought, with defeat stories using fewer past-tense verbs than expected. This could indicate that stories of defeat describe events in a more immediate or reflective manner rather than relying heavily on past actions.

Section 2.1

I transformed the data into a pandas dataframe again and same as above I removed the "//n" from the text. I then loaded in the pre-trained model from gensim using *word2vec_model = gensim.downloader.load('word2vec-google-news-300').* I downloaded the stopwords library from nltk and used this to remove stopwords to allow my model to generate more meaningful and informative word embeddings rather than focusing on words that lack relevent content. I used nltk tokeniser to split the text and then used my Word2Vec model to generate the emeddings which I stored in a list. For my train/test split I went with an 80% training and 20% testing split and stratified y to ensure an even split of label types to account for the slight class imbalance shown below.

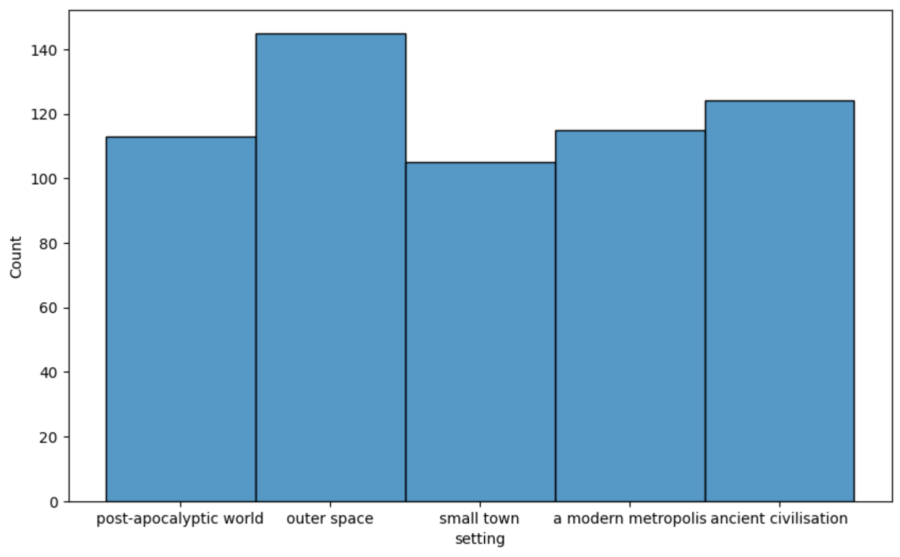


Fig. 7 *Distribution of Setting Type Counts*

To prepare the text input to my model I fed the stories to a function I created which first tokenised the text, then removed tokens that were considered stopwords and finally got the embeddings for from the pretrained Word2Vec model using .get_vector and .index_to_key. I averaged the word embedding vectors of all the words in each story. The y values used were in the setting column.

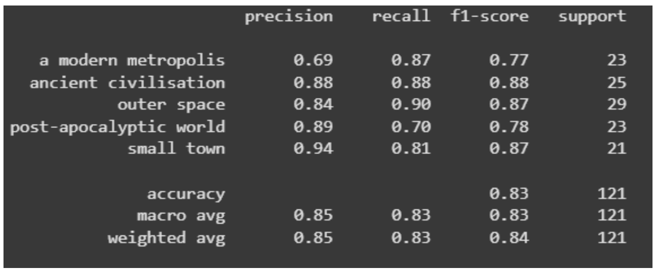
I used three different model types, all of which were used in ECS8051. I started with a simple train/test split, but to obtain more reliable results, I went for 5-fold cross-validation. This approach was necessary due to the limited data available, ensuring that the train/test split wasn't overly favorable and inflating model performance.

Model	Cross-Validation (5 Fold)	Accuracy Score	Weighted F1 Score
Logistic Regression	No	83.47%	83.54
Random Forest	No	80.17%	80.04
SVC	No	85.12%	85.32
Logistic Regression	Yes	76.74%	76.42
Random Forest	Yes	74.25%	73.97
SVC	Yes	79.40%	79.44

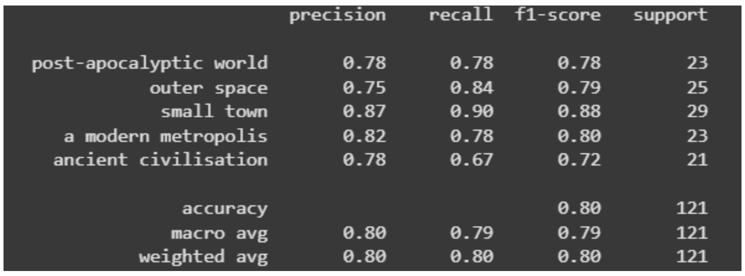
Fig. 8 *Results of Models using Word2Vec Embeddings*

Interestingly, the weakest performer across the board was Random Forest, with 80% accuracy for the train/test split and only 74% for the cross-validation model. In contrast, SVC clearly outperformed the other two models in both scenarios, achieving the highest accuracy of 85.12% in the train/test split and 79.40% with cross-validation. The fact that there was a consistent drop in performance metrics when using cross-validation suggests that it introduced a more robust evaluation. This highlights the potential issue with me just using a simple train/test split, which had a more favorable (and less challenging) split due to the relatively small size of the test set. By using cross-validation, the evaluation was more reliable and accounted for variations in the data distribution.

However, this relatively small drop in performance between the train/test split and cross-validation shows that the models are quite robust, and the word embeddings I used are effective for capturing the setting of the stories. Even with limited data, my models demonstrated their ability to generalise well to new, unseen data.

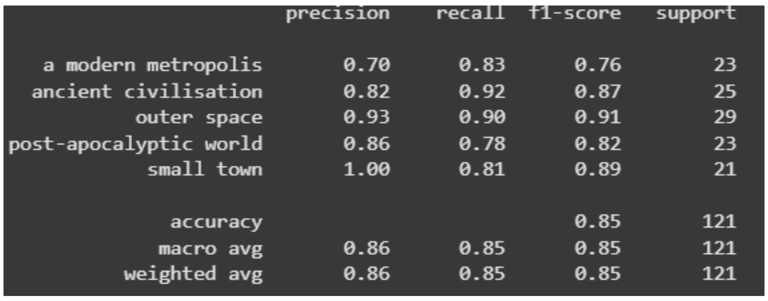


```
                       precision    recall  f1-score   support

  a modern metropolis       0.69      0.87      0.77        23
 ancient civilisation       0.88      0.88      0.88        25
          outer space       0.84      0.90      0.87        29
post-apocalyptic world       0.89      0.70      0.78        23
           small town       0.94      0.81      0.87        21

             accuracy                           0.83       121
            macro avg       0.85      0.83      0.83       121
         weighted avg       0.85      0.83      0.84       121
```



```
                        precision    recall  f1-score   support

post-apocalyptic world       0.78      0.78      0.78        23
           outer space       0.75      0.84      0.79        25
            small town       0.87      0.90      0.88        29
   a modern metropolis       0.82      0.78      0.80        23
  ancient civilisation       0.78      0.67      0.72        21

              accuracy                           0.80       121
             macro avg       0.80      0.79      0.79       121
          weighted avg       0.80      0.80      0.80       121
```

Fig.9&10 *Logistic and Random Forest Model Results*



```
                        precision    recall  f1-score   support

   a modern metropolis       0.70      0.83      0.76        23
  ancient civilisation       0.82      0.92      0.87        25
           outer space       0.93      0.90      0.91        29
post-apocalyptic world       0.86      0.78      0.82        23
            small town       1.00      0.81      0.89        21

              accuracy                           0.85       121
             macro avg       0.86      0.85      0.85       121
          weighted avg       0.86      0.85      0.85       121
```

Fig. 11 *SVC Model Results*

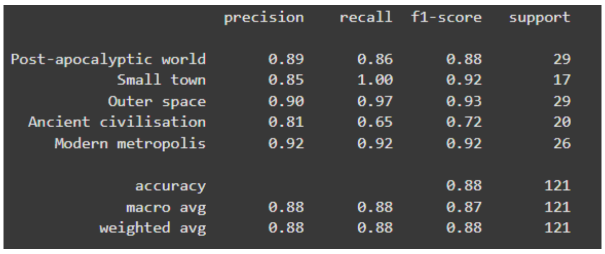
Quickly looking at the classification report, despite having the lowest accuracy, Random Forest actually has very similar precison and recall across the classes showing a stable model. Logistic Regression has very god metrics but is slightly prone to defaulting to predicting “a modern metropolis” leading to that low precision score. SVC had similar tendencies. It also performed best on the dominating class “outer space” which shows a powerful model.

I suspect SVC outperformed the other models because of its linear kernel. As word embeddings transform the data into a dense, continuous vector space this makes my task suited to linear decision boundaries therefore making SVCs a useful model to use.[11] Random Forest, being a more complex ensemble model, may have overfitted the data in the simpler train/test scenario, leading to poorer generalisation performance in cross-validation and explains why logistic and SVC outperfomed it. My key takeaway is that SVC is a good balance between a complex enough model to capture the key information but not overly comlex that it struggles with the small training set.

Section 2.2

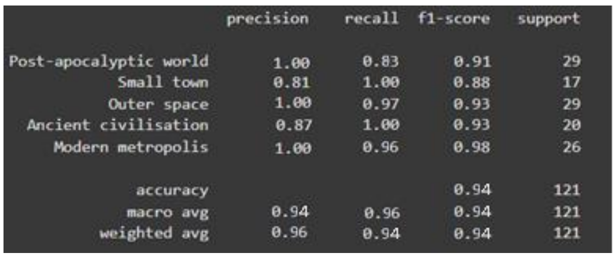
This section I leveraged BERTs predictive power to predict the setting category same as above. I split the data as described above. I mapped the labels from 0-4 to comply with BERT. I used the BERT model "bert-base-uncased" and applied its tokeniser to my data, with padding and truncation set to True and max_length set to 512. This ensured that any stories exceeding the allotted size were truncated. For training, I used the Hugging Face Trainer API to fine-tune my model.[5] I experimented with different training epochs and found that due to the power of BERT, the training loss converged rapidly. As a result, 3-5 epochs appeared to be the optimal range to prevent significant overfitting. Below is a table showing the classification results for 3 and 4 epochs.

At 3 Epochs:

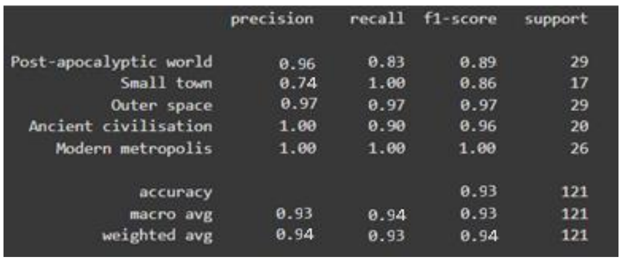


```
                        precision    recall  f1-score   support

Post-apocalyptic world       0.89      0.86      0.88        29
            Small town       0.85      1.00      0.92        17
           Outer space       0.90      0.97      0.93        29
  Ancient civilisation       0.81      0.65      0.72        20
     Modern metropolis       0.92      0.92      0.92        26

              accuracy                           0.88       121
             macro avg       0.88      0.88      0.87       121
          weighted avg       0.88      0.88      0.88       121
```

At 4 and 5 epochs



```
                       precision    recall  f1-score   support

Post-apocalyptic world      1.00      0.83      0.91        29
            Small town      0.81      1.00      0.88        17
           Outer space      1.00      0.97      0.93        29
   Ancient civilisation     0.87      1.00      0.93        20
     Modern metropolis      1.00      0.96      0.98        26

              accuracy                          0.94       121
             macro avg      0.94      0.96      0.94       121
          weighted avg      0.96      0.94      0.94       121
```



```
                       precision    recall  f1-score   support

Post-apocalyptic world      0.96      0.83      0.89        29
            Small town      0.74      1.00      0.86        17
           Outer space      0.97      0.97      0.97        29
   Ancient civilisation     1.00      0.90      0.96        20
     Modern metropolis      1.00      1.00      1.00        26

              accuracy                          0.93       121
             macro avg      0.93      0.94      0.93       121
          weighted avg      0.94      0.93      0.94       121
```

These results provide key insights into the fine-tuning process. BERT clearly outperforms Word2Vec embeddings combined with a classifier in text classification tasks. While higher epochs (4 and 5) have better accuracy scores than 3 epochs, accuracy alone did not determine the best-performing model. Examining the validation and training loss trends, it was evident that the model began to overfit quickly. This is expected given that my training set had only 480 examples, while BERT has 12 transformer layers, 12 attention heads, and 110 million parameters—a massive size compared to the dataset size. [3]

At 3 epochs, precision and recall scores remained consistent across all classes at around 90%, indicating stable learning. However, at 4 and 5 epochs, the model started favoring majority classes, leading to instability. For example, "Modern Metropolis" and "Post-Apocalyptic World" achieved 100% precision and recall, but other classes declined, showing that the model was memorising dominant patterns rather than generalising well. While that inflated the accuracy due to the train/test split, it ultimately harmed overall performance and stability.

To prevent overfitting, I used early stopping and selected 3 epochs as the optimal point. This provided a model that performed consistently well across all metrics, achieving 88% overall accuracy while maintaining stable class-wise precision and recall.

Comparing Word2Vec and BERT:

Through this process, I gained a deeper understanding of how Word2Vec and BERT differ in their approach to text classification and why BERT performs better. Word2Vec, creates static word representations, meaning that a word has the same meaning regardless of the sentence it appears in. BERT, on the other hand, produces contextual embeddings, allowing the same word to have different meanings depending on the surrounding words. As I was trying to predict story themes with a large body of text this proved invaluable in improving classification accuracy, as BERT was able to capture contxt that Word2Vec simply could not. This additional context can be reflected in the difference in metrics, however the gap is not as wide due to the smaller training set size and the issue of overfitting for BERT.

Section 3.1

In this section I used the last sentence of each story to predict the either “defeat” or “victory” uisng BERT. I was interested to see if one sentence is enough to make an accurate prediction. The first step I took was to ensure that there wasn’t a major class imbalance to avoid bias. There was no significant imbalance as seen below:

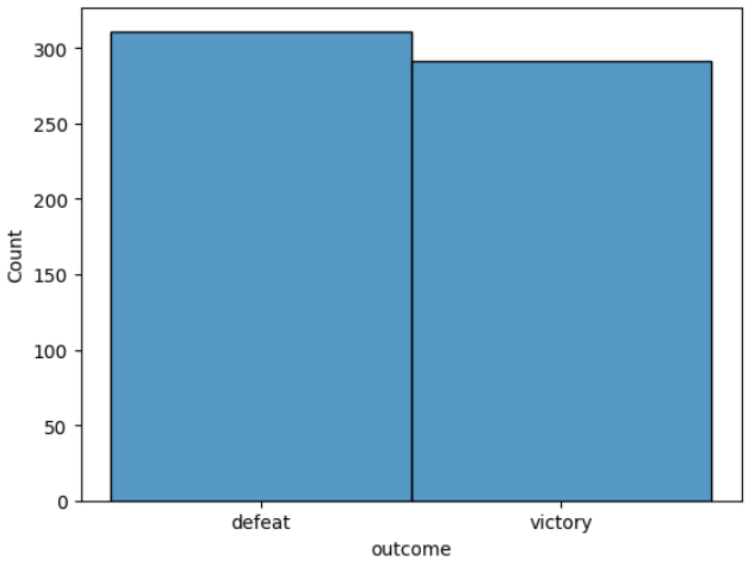
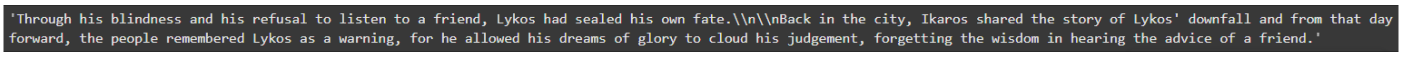


Fig. 12 *Class Distribution for Outcome column*

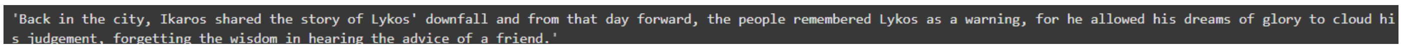
I then loaded in the “bert-base-uncased" model from huggingface and loaded in its tokenizer. I transformed the data into a panadas dataframe and tried splitting the story and extracting the final sentence using:

data_df["last_sentence"] = data_df["story"].str.split(r'(?<=[.!?])\s+').str[-1]

This did work but there was an issue caused by the “//n” as seen below so I removed them the same as above and re-ran and got the correct sentence.



```
'Through his blindness and his refusal to listen to a friend, Lykos had sealed his own fate.\\n\\nBack in the city, Ikaros shared the story of Lykos' downfall and from that day
forward, the people remembered Lykos as a warning, for he allowed his dreams of glory to cloud his judgement, forgetting the wisdom in hearing the advice of a friend.'
```



```
'Back in the city, Ikaros shared the story of Lykos' downfall and from that day forward, the people remembered Lykos as a warning, for he allowed his dreams of glory to cloud hi
s judgement, forgetting the wisdom in hearing the advice of a friend.'
```

Fig. 13&14 *Before and After Data Splitting for Last Sentence Example*

There were a number of initalisations that needed to be made for compatibility with BERT. First I needed to find the max length for token input to BERT. To do this I plotted the lengths of the last sentences in each story (the input to BERT).

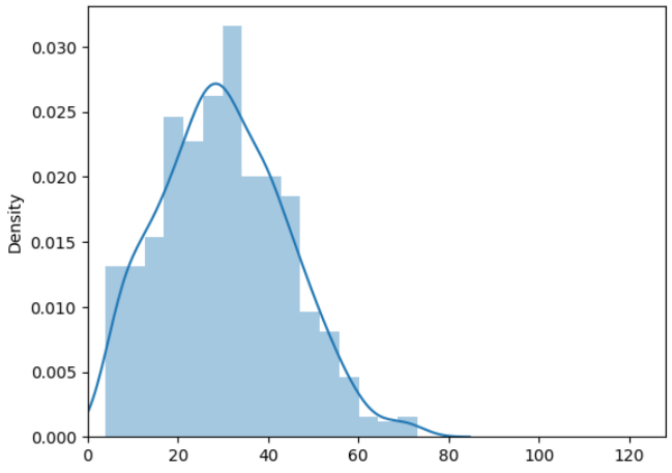
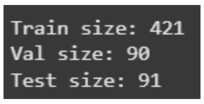


Fig. 15 *Number of tokens per sentence to help chose length of Max_Length*

I settled for a max_length of 80, meaning that any inputs under that length would be padded with zeros to ensure uniform length. I also mapped the ouput tags of *outcome* and *defeat* to 0 and 1 to make them numeric for BERT.

The train/val/test split I then used sklearns build in split function and split training and validation 70/30. I then split the validation into 50/50 to create a test set. All these were performed with random_seed set to 40446925.



```
Train size: 421
Val size: 90
Test size: 91
```

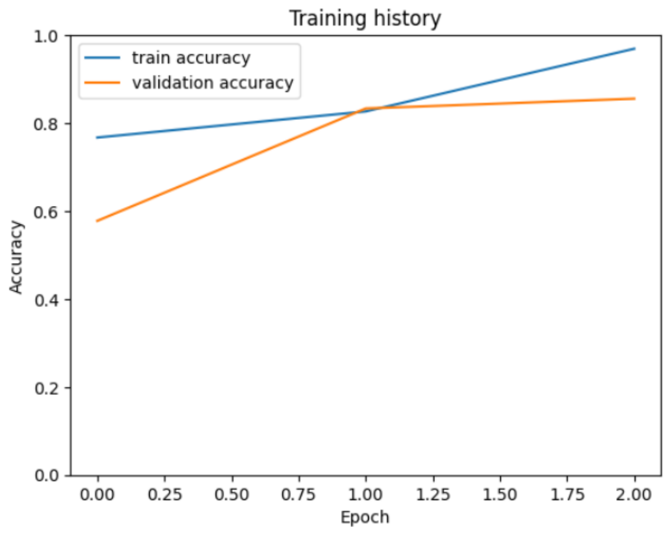
The data split is shown above. Given the small training set, a smaller split would have made it difficult to ensure a representative test and validation set. This lack of data is a limitation of the model.

I created the StoriesDataset class to handle the stories and their sentiment labels for BERT. The class takes in the stories, sentiment labels, a tokenizer, and the maximum token length. The len method returns the number of stories, while the getitem method tokenizes each story, handles padding and truncation, and returns the input IDs, attention masks, and sentiment labels in a BERT-compatible format.

While passing this class into the DataLoaders, I encountered shape issues. The problem was that I needed to switch from .flatten() to .squeeze(0) to avoid merging everything into a single vector, which caused indexing errors. [10] This took some time to resolve.

Next, I passed the data into DataLoaders with batch size 16 and used BertForSequenceClassification.from_pretrained(). I used the Adam optimiser with a learning rate of 2e-5 and set total steps as len(train_loader) * epochs. For the training and evaluation, I implemented the loop using PyTorch and tutorial code as a guide. The training loop used a linear learning rate scheduler and computed the loss with the cross-entropy loss function, applying gradient clipping to prevent exploding gradients.

Results:



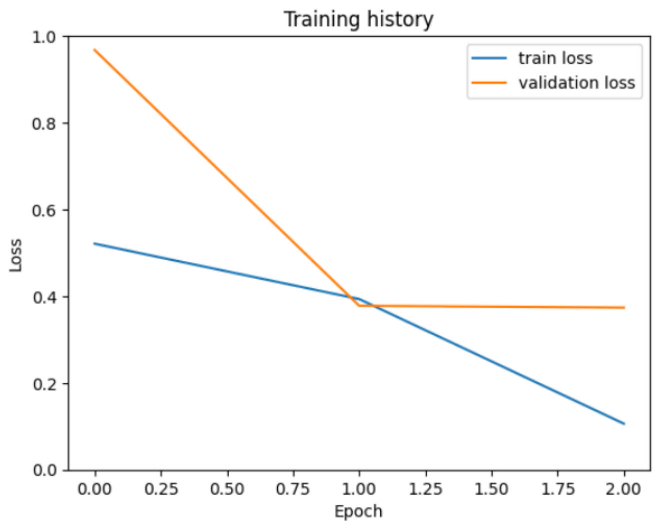
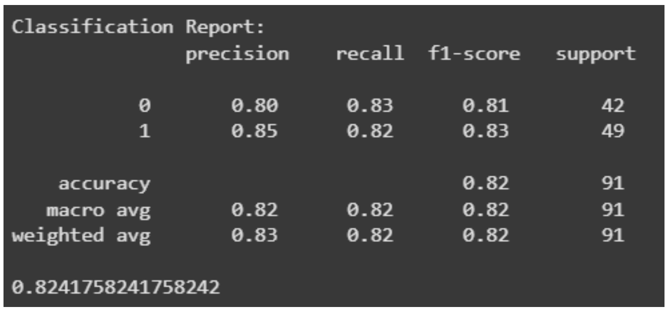


Fig. 16&17 *Train and Loss Accuracy and Loss Plots*

The training and validation accuracy plot shows that BERT's strong learning capability allows it to achieve high accuracy quickly. However, the loss curves exhibit an unusual shape, likely due to the small dataset size. BERT, being a powerful pre-trained model, learns from the limited data and began to overfit after just one epoch. Initially, both training and validation losses decreased, but the validation loss soon plateaued while the training loss continued to drop. This indicated that the model memorised the training data but struggled to generalise on unseen validation samples. Cross validation didn't improve performance either.

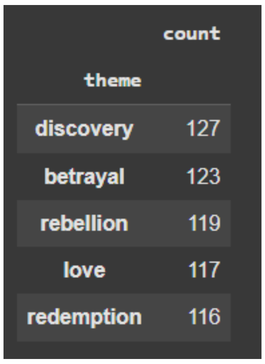
Despite the model converging quickly, it performed well on the test set with an 82% accuracy and balanced precision and recall across both victory and defeat. The test results closely aligned with validation accuracy, showing a representative data split and the absence of bias. While adding more data would help refine validation tuning and further improve performance, the current results are impressive, especially considering that only the final sentence from each story is used. This highlights BERT's strong capability for sentiment analysis, even with limited input.



```
Classification Report:
              precision    recall  f1-score   support

           0       0.80      0.83      0.81        42
           1       0.85      0.82      0.83        49

    accuracy                           0.82        91
   macro avg       0.82      0.82      0.82        91
weighted avg       0.83      0.82      0.82        91

0.8241758241758242
```

Section 3.2

In this section I used a the pretrained emotional model *twitter-roberta-base-emotion-multilabel-latest* to get a probability distribution for each sentence.[1] When looking at the theme field I actually found five themes. The themes and their counts are shown below:



theme	count
discovery	127
betrayal	123
rebellion	119
love	117
redemption	116

There was a roughly even spread of the themes which was good. In order to analyse if the the emotional content differs across the themes, I averaged the probabilty vectors together per story and grouped by theme.

Data Preprocessing:

Similar to all the steps above I removed the "//n" cases and any leading whitespaces. I also decided to convert the data into a pandas dataframe for this section as I was more comfortable working with that format for this and felt it was the best approach.

Issues Loading in model:

Initially, I tried to import by pip installing built in library tweetnlp as shown on the huggingface website.[1] However, this was leading to an import error so I decided to take the traditioan approach of loadingh in the model the same way as in Section 2.

Process:

I created three functions for the process: the first obtained emotion scores for each sentence, the second processed the entire story by splitting it into sentences and applying the first function, and the third, get_average_emotion_vector, calculated the average emotion vector for each story. This added two new columns to my dataframe: one for the emotional vectors per story and another for the overall emotion vector.

Next, I grouped the data by theme and extracted the corresponding vectors from data_df["average_emotion_vector"]. I calculated the average for each theme and plotted the emotional labels with their values.

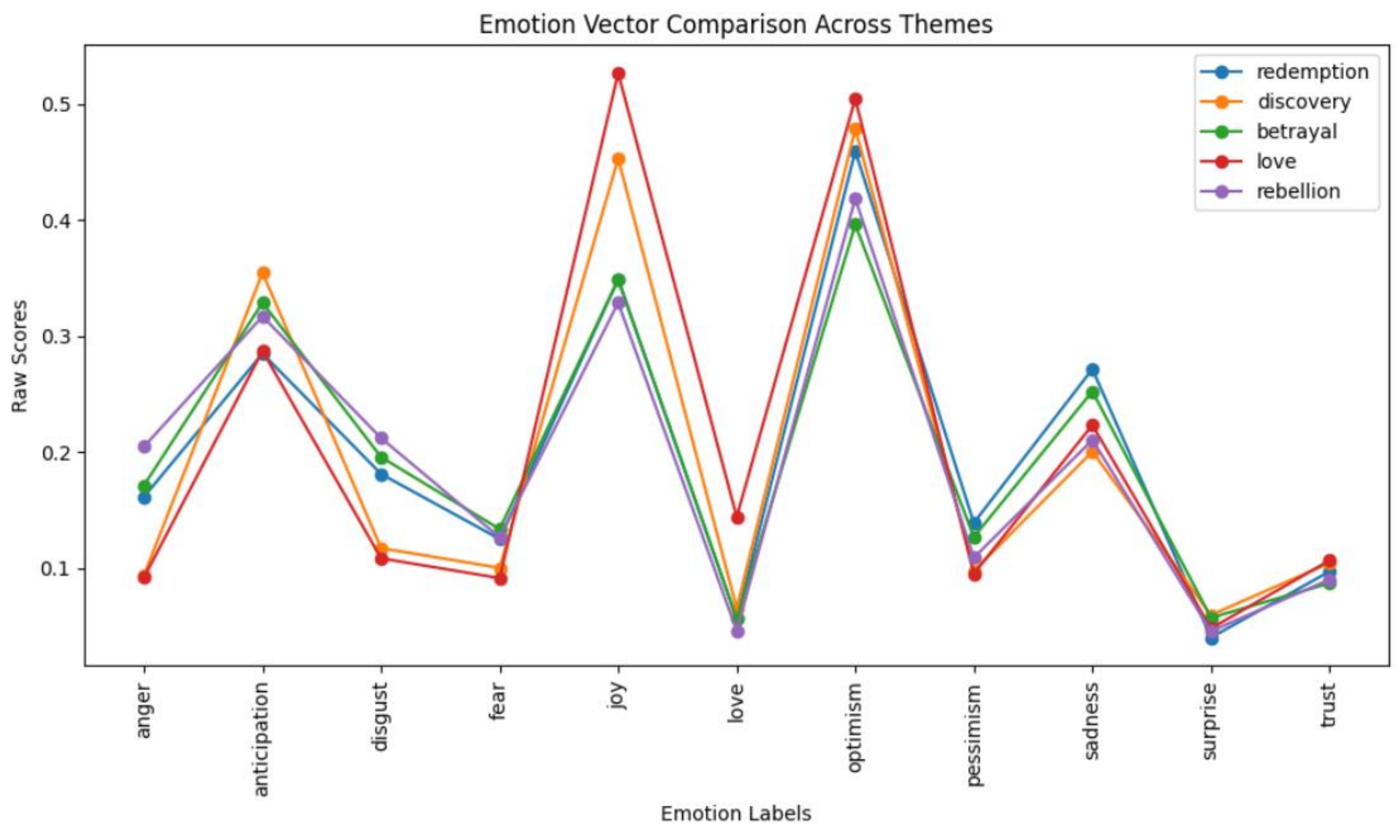


Fig. 18 *Emotion Vector Comaprison Across Themes*

Observations:

I was initially surprised by the plot's consistency across themes, which seemed too uniform. I manually calculated the average emotion vectors for the "love" category, expecting higher values, but after confirming the accuracy of my computations, I realised the results were correct.

The dominating emotions—joy, optimism, anticipation, and sadness—appear across all themes, with optimism consistently high. This makes sense, as even in negative themes like rebellion, there's often an underlying sense of hope. These broad emotions are common in the early parts of a story and don't vary much by theme, making them easier for the emotion detection model to capture.

Broad emotions like joy and optimism are common early in a story, while more specific emotions like surprise or trust emerge later. These universal emotions are easier for the model to detect and can appear across various contexts, leading to the pitfall of averaging sentence vectors. For example, in a betrayal story, joyful sentences at the start are treated the same as those describing betrayal, skewing the results. Since emotions like betrayal are harder to detect, the model often defaults to broader emotions. This highlights the weakness of averaging emotion vectors as a method for predicting emotional content in stories.

References

1. “Cardiffnlp/Twitter-Roberta-Base-Emotion-Multilabel-Latest · Hugging Face.” Huggingface.co, huggingface.co/cardiffnlp/twitter-roberta-base-emotion-multilabel-latest.
2. “Conduct and Interpret a Mann-Whitney U-Test.” Statistics Solutions, www.statisticssolutions.com/free-resources/directory-of-statistical-analyses/mann-whitney-u-test-2/.“Demystifying
3. BERT: The Groundbreaking NLP Framework.” Analytics Vidhya, 25 Sept. 2019, www.analyticsvidhya.com/blog/2019/09/demystifying-bert-groundbreaking-nlp-framework/.
4. “DependencyParser · SpaCy API Documentation.” *DependencyParser*, spacy.io/api/dependencyparser.
5. “Fine-Tuning a Model with the Trainer API - Hugging Face NLP Course.” Huggingface.co, 2025, huggingface.co/learn/nlp-course/en/chapter3/3.
6. Holtz, Yan. “Raincloud Plot in Python with Matplotlib and PtitPrince.” The Python Graph Gallery, 2024, python-graph-gallery.com/raincloud-plot-with-matplotlib-and-ptitprince/.
7. OpenAI. “OpenAI API.” Platform.openai.com, 2024, platform.openai.com/docs/models.
8. “Penn Treebank P.O.S. Tags.” *Www.ling.upenn.edu*, www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html
9. Qasim Al-Ma'arif. “Language Basics in Natural Language Processing (NLP).” Medium, 15 Jan. 2024, https://medium.com/@datailm/language-basics-in-natural-language-processing-nlp-f478ecf8f900
10. Shaikh, Mohsin. “Numpy Ravel V/S Numpy Flatten V/S Numpy Squeeze - Mohsin Shaikh - Medium.” Medium, 18 July 2023, medium.com/@mohsin.shaikh324/numpy-ravel-v-s-numpy-flatten-v-s-numpy-squeeze-25d215a5ccbd.
11. Truşcă, Maria Mihaela. “Efficiency of SVM Classifier with Word2Vec and Doc2Vec Models.” Proceedings of the International Conference on Applied Statistics, vol. 1, no. 1, 1 Oct. 2019, pp. 496–503, https://doi.org/10.2478/icas-2019-0043.