CS918 Natural Language Processing
Assignment 2 - Sentiment Classification for Social Media
Report

5652781

Abstract

This report describes how we build classifiers for Twitter/X and analyse the classifiers. We train four classifiers based on traditional machine learning methods on different sets of features, and a classifier based on LSTM using PyTorch.

1 Introduction

1.1 Aim & Objectives

Given a training set of tweets, we build and compare classifiers for Twitter/X which predicts the sentiment of a tweet. We train several classifiers based on traditional machine learning methods on different sets on features, and a neutral classifier based on LSTM using PyTorch. Then we adjust our classifiers with our development set, and finally apply on test sets and give the macro F1 score of classifiers on the test sets.

1.2 Methodology

We first observe manually given sets of tweets and identify the data structure, significant features of tweets, and noises such as URLs (as in Section 2). Then we pre-process and lemmatise our tweets (as in Section 3). After that, we build our classifiers (as in Section 5) with chosen features (as in Section 4), which we describe what and how parameters are chosen. Finally, we give and discuss the macro F1 scores of each pair of classifier and feature (as in Section 7).

2 Tweets data observation

We observe that tweets include URLs, @user mentions, #hashtags, emoticons (e.g. :-), :(, etc.), emojis, abbreviated words and foreign words. It is too difficult to extract sentiment information from URLs and @user mentions. For example, the user mention @RealDonaldTrump may appear to be positive for a Trump supporter but negative for democrats, and neutral for the rest. Hence, we will remove them before sentiment analysis. However, the rest may contain sentiment information. We will discuss these in Section 3.

3 Pre-processing & Lemmatisation

Before we extract features from the tweets, we need to pre-process the tweets so that only words carrying sentiment information remain. After removing URLs and @user mentions, the order is as follows:

3.1 #hashtags

While #hashtags contain sentiment information, we encounter the following problems:

- Training data contain a lot of noise. For example, #FantasticBeasts contains a positive word fantastic, but Fantastic Beasts is a movie and is unrelated to the tweet's sentiment.
- It is difficult to extract words if the #hashtag is uppercase/lowercase only.

Therefore, we decide to remove all #hashtags.

3.2 Emoticons

We use a dictionary from [1], with key being the emoticon and value being its meaning. We write a function that on a text input, return the text with all emoticons converted to their meanings in text.

3.3 Emojis

Emojis contain sentiment information for the tweets. Hence, we want to extract the meaning of each emoji. This is done by using the package `emoji` and its function `emoji.demojize`, which returns a string of the form in regular expression `:[A-Za-z0-9_]*:`. We further pre-process the string into a phrase without colons or underscores.

3.4 Abbreviated words

Notable abbreviated words include "RT" (retweet), suffices "n't", "'re", "'ve" and "'m". If "RT" appears before a @user mention, we remove "RT"; else we turn "RT" into "retweet". For "'m", it mainly appear as "I'm", so we change "I'm" into "i am" (lowercase).

3.5 End-of-sentence (E-O-S) punctuations

It is easier to deal with lemmatisation and negation after we separate E-O-S punctuations from the sentence. For example, we cannot tag a word with an E-O-S punctuation as the tokeniser cannot recognise the word. (See Figure 1)

$$\texttt{['better.']} \rightarrow \texttt{['NN']}$$
$$\texttt{['better', '.']} \rightarrow \texttt{['adj', '.']}$$

Figure 1: POS tagging with PennWordTree with / without separate E-O-S punctuations.

3.6 Lemmatisation

We first apply POS tagging on each word of a tweet using Penn Tree Bank. Then we convert the tag into WordNet POS tag by whether the word is a noun, verb, adjective or adverb. Finally, we lemmatise the tweet based on the WordNet POS tagging.

3.7 Dealing with negation

It is difficult to deal with negation after pre-processing each tweet. For example, we consider the sentence "I don't like apples. I like oranges." If we pre-processing the sentence and add "not_" after negation, it becomes "i don't not_like not_apples not_i not_like not_oranges", the latter part contradicts the meaning of the original sentence because we fail to identify the punctuation and stop adding "not_" afterwards. Therefore, we add negation prefix before removing non-alphanumeric characters.

If a negation word like "not" appears, we start adding `not_` prefix to word after the negation word until a stop word or an E-O-S punctuation. We built a list of stop words using Penn Tree Bank and NLTK stop words with manual adjustment. This includes

- "CC" - Co-ordinating conjunction (e.g., "and", "or", "but"),
- tags that starts with "W" - "wh"-words (e.g., "who", "what", "where"),
- and a customised list of stop words.

3.8 Stopword

We experimented the impact of stopword removal and notice that the macro F1 score for sentiment analysis drops in general. A possible reason is that we also take care of tweets with neutral sentiment. Stopword removal helps determine if a tweet is positive or negative, but not neutral. Therefore, we keep stopwords for sentiment analysis.

3.9 The rest

Finally, we remove non-alphanumeric characters, number-only characters, and single characters, most of which do not provide any sentiment information. We note that due to the negation prefix, we need to take care of unneeded words with the `not_` prefix.

3.10 Example

Consider the following text:

“ *RT nltk your #nlp is not bad at all... :(http://t.co/PNwNBanIvr* ”.

After pre-processing and lemmatising, the text becomes

“ *your be not not_bad not_at not_all sad* ”.

After pre-processing and lemmatising, we train our classifiers with features with the training data. Below, we describe what features and classifiers have been chosen and their expected results.

4 Features

We choose the following features to be paired with classifiers, ranging from simple ones like BoW to more complicated one like TF-IDF.

4.1 Bag-of-Words (BoW)

We form a *bag of words* by concatenating all tweets into a single document and count the number of each word in the big document. Then, for each tweet, each feature in the resulting vector reflects the frequency of a word within the tweet. Since BoW is a naïve (but fast) feature, we do not expect high for its performance.

4.1.1 Binary BoW

Instead of frequency count, each feature in the resulting vector reflects the presence of a word within the tweet, i.e. 1 if the word is present and 0 else.

4.1.2 n-grams

The BoW also contains n-grams apart from unigrams. This is useful since tweets contain phrases like slangs that unigrams cannot capture.

4.2 Lexicon

Instead of BoW, we count the frequency or presence (if binary) of a word within the tweet if the word appears in a given lexicon, such as a collection of positive and negative words.

This feature is well considered with a lexicon of positive and negative words. However, we expect this feature does not perform well since we also need to identify neutral sentiment.

4.3 Term Frequency-Inverse Document Frequency (TF-IDF)

Term Frequency-Inverse Document Frequency (TF-IDF) weighs each word based on how unique it is across all documents. It scales the raw frequency of words in a document by the inverse frequency of those words across the dataset. This can diminish the impact of ubiquitous terms like “the” and “is” while giving more weight to words that carry distinctive meaning. With a complex classifier, we expect TF-IDF works better than BoW.

Similar to BoW, TF-IDF can be extended to binary or with n-grams.

5 Traditional Classifiers

We start with simpler classifier like Naïve Bayes to estimate the lower bound of the macro-F1 score, then use more complex classifiers and features.

5.1 Naïve Bayes (NB)

We first introduce a simple classifier called *Naïve Bayes classifier*. For a (pre-processed and lemmatised) test tweet $T = w_1...w_k$, we assign predicted sentiment s^* to T if

$$s^* = \underset{s}{\text{argmax}} P(s^*|T) = \underset{s}{\text{argmax}} \frac{P(s)P(T|s)}{P(T)} = \underset{s}{\text{argmax}} P(s)P(T|s).$$

Since computing $P(T|s) = P(w_1...w_k|s)$ is time-consuming, we make the naïve assumption that

$$P(w_1...w_k|s) \approx P(w_1|s)...P(w_k|s).$$

Moreover, to avoid zero probability, we apply additive-α smoothing such that

$$P(w_i|s) = \frac{\text{count}(w_i, s) + \alpha}{\sum_{w \in V} \text{count}(w, s) + \alpha|V|}.$$

where V is the vocabulary.

5.1.1 Complement NB (CNB)

Since our training data are imbalanced, such as containing slangs and abbreviated words, we use statistics from the complement of each sentiment class to compute the model's weights. We expect Complement NB performs better than Binary NB.

5.1.2 Choosing α

We find α for additive-α smoothing which maximises the macro F1 score of sentiment analysis on the development set of tweets for each combination of Naïve Bayes classifier and feature. This is done by running sentiment analysis on various values of α. Table 1 shows the choice of α for each pair of Naïve Bayes classifiers. (See Appendix B for performances of NB classifiers under various α.)

Features / **Classifier**	BoW	Binary BoW	Binary BoW + n-gram	Lexicon
NB	0.26	0.35	0.15	0.01
CNB	0.635	0.62	0.33	0.01

Features / **Classifier**	TF-IDF	Binary TF-IDF	TF-IDF + n-gram	Binary TF-IDF + n-gram
NB	0.05	0.05	0.01	0.01
CNB	0.49	0.485	0.1	0.2

Table 1: Chosen α value for each pair of Naïve Bayes classifiers and feature.

5.1.3 Expectation

Due to its naïve mechanism, Naïve Bayes classifiers do not have high performance (but they are fast). They should also be well paired with BoW feature and its variations.

5.2 Logistic regression

Given a feature vector $\mathbf{v}$, a weight matrix $\mathbf{W}$ and a bias vector $\mathbf{b}$, the expected sentiment $\hat{y}$ is computed by

$$\hat{y} = \text{softmax}(\mathbf{W}\mathbf{v} + \mathbf{b}),$$

where the softmax function on a vector $\mathbf{z} = (z_1, ..., z_k)$ is defined by

$$\text{softmax}(\mathbf{z}) = \left(\frac{\exp(z_1)}{\sum_{j=1}^{k} \exp(z_j)}, ..., \frac{\exp(z_k)}{\sum_{j=1}^{k} \exp(z_j)} \right).$$

Logistic regression iteratively updates the values of weights $\mathbf{W}$ and biases $\mathbf{b}$ so that the expected sentiment $\hat{y}$ is close to the true sentiment y.

5.2.1 Regularisation

To avoid overfitting, we use *regularisation* to penalise large weights. There are two types of regularisations - L1 regularisation (using L1 norm) and L2 regularisation (using L2 norm). While L1 regularisation is theoretically more suited for sparse input, it has much lower efficiency than L2 regularisation. We tested that it takes more than 2 minutes to train a logistic regression classifier with L1 regularisation for BoW feature. Therefore, we chose L2 regularisation for our logistic regression classifier.

We also determine the regularisation constant λ which control the strength of the regularisation. To do this, we do a grid search on λ based on the performance of the classifier on the development tweets set. Table 2 shows our choice of λ for each feature. (See Appendix C for logistic regression performance under various λ.)

Features	BoW	Binary BoW	Binary BoW + n-gram	Lexicon
λ	$10^{0.5}$	$10^{0.5}$	10^{0}	$10^{-0.5}$
Features	TF-IDF	Binary TF-IDF	TF-IDF + n-gram	Binary TF-IDF + n-gram
λ	$10^{-0.5}$	$10^{-0.5}$	$10^{-0.5}$	$10^{-1.5}$

Table 2: Chosen λ value for each feature.

5.2.2 Expectation

We expect logistic regression performs better with TF-IDF and its variations than Naïve Bayes classifiers due to a more complex structure.

5.3 Support vector machine (SVM)

Support vector machine (SVM) takes feature vectors as input and iteratively finds the best hyperplane that separates the sentiment classes.

5.3.1 Regularisation

Similar to Logistic Regression, regularisation is applied to avoid overfitting. We have chosen the regularisation constant λ for each feature as shown in Table 3. (See Appendix C for logistic regression performance under various λ.)

Features	BoW	Binary BoW	Binary BoW + n-gram	Lexicon
λ	10^{1}	10^{1}	$10^{0.5}$	$10^{-1.5}$
Features	TF-IDF	Binary TF-IDF	TF-IDF + n-gram	Binary TF-IDF + n-gram
λ	10^{0}	10^{0}	10^{0}	10^{-1}

Table 3: Chosen λ value for each feature.

5.3.2 Expectation

We expect SVM performs better with TF-IDF and its variations than Naïve Bayes classifiers due to a more complex structure.

5.4 General expectation

We expect traditional classifiers are more efficient but perform not as good as a well-trained neural classifier, which we discuss below.

6 Neural Classifier based on Long Short-Term Memory (LSTM)

We train a neural classifier based on LSTM with an 'Adam' optimiser and pre-trained GloVe word embedding. As required by the assignment, we set the maximum number of tokens as 5,000 and embedding dimensionality as 100.

6.1 Word choice for embedding matrix

Since the maximum number of tokens are 5,000 as required, we must choose most relevant words related to tweets. We observe that the GloVe word embedding contains characters that were removed during pre-processing, such as non-alphanumeric characters and single characters, we remove these words from the given word embedding. Then, we choose the top 5,000 most frequent words that appear in the training set as our embedding matrix.

6.2 Maximum length for each tweet token vector

We note that if we set a high maximum length for each tweet token vector, then the vectors contain a lot of zero entries for short tweets, which diminishes the sentiment reflected in the tweets, hence, lowers the accuracy of the classifier. On the other hand, if the maximum length is too low, then the vector cannot capture the full sentence for long tweets.

We observe that the longest tweet (after pre-processing) has 43 words. Hence, we set the maximum length as 43.

6.3 Batch size

In general, a smaller batch size has higher improvement on sentiment analysis per training but runs very slowly, while a larger batch size has smaller improvement on sentiment analysis per training but runs fast. Since we are training our classifier with more than 40,000 training data, we set batch size as 512, and we use more epochs to compensate the low improvement issue. By training slowly, we can also identify overfitting early and revert to previous model.

6.4 Deriving hidden states

For each loop, the LSTM layer returns a sequence of hidden states. We compare choices of deriving the hidden state as follows:

- *Final hidden state*: We obtain the final hidden state and feed it to the linear layer. In theory, the final hidden state should contain information of previous hidden states.
- *Mean Pooling*: We obtain a mean of all hidden states and feed it to the linear layer. This smooths out noise but may diminish strong signals of sentiment information.
- *Max Pooling*: We obtain the maximum of all hidden states and feed it to the linear layer. This highlights the strongest signal but may miss out other sentiment information.

6.5 Dropout

To avoid overfitting, we introduce dropout layer to drop half of the trained data per loop.

6.6 Training epochs & learning rate

To determine the number of epochs for training, we run a sentiment analysis on the development set after every training. In particular, we check if the classifier improves after each training. If the classifier fails to improve, we use and re-train the previous model with a lower learning rate. We stop training once the classifier is unable to improve after few epochs, and we obtain the number of epochs used.

We set our tolerance level as 6, i.e. if the model fails to improve based on the development set after 6 consecutive epochs, then we terminate the training. While higher tolerance may find a better model, it is too time-consuming since each training takes about 2 minutes and training is ineffective with very low training rate.

6.7 Expectation

Since neural classifiers consider the order of a sentence, they should perform better than traditional classifiers.

7 Results & Discussion

After adjusting parameters of every classifier, we obtain the following results.

7.1 Traditional classifiers

Table 4 shows the average macro F1 score of each pair of traditional classifier and feature across given three test sets. We highlighted the highest average macro F1 score in the table and note that no pair of traditional classifier and feature has average macro F1 score exceeding 0.6. (See Appendix A for plots of maximum, minimum and average macro F1 scores of each pair of tradition classifier and feature across given three test sets.)

Features / **Classifier**	BoW	Binary BoW	Binary BoW + n-gram	Lexicon
NB	0.548	0.548	0.545	0.483
CNB	0.576	0.577	0.553	0.471
Logistic regression	0.562	0.563	0.519	0.516
SVM	0.578	0.581	0.565	0.533
Features / **Classifier**	TF-IDF	Binary TF-IDF	TF-IDF + n-gram	Binary TF-IDF + n-gram
NB	0.446	0.45	0.475	0.478
CNB	0.566	0.526	0.49	0.524
Logistic regression	0.577	0.579	0.587	0.593
SVM	0.573	0.47	0.599	0.596

Table 4: Average macro F1 score of each pair of classifier and feature across three test sets (rounded to 3 d.p.).

As expected, NB and CNB performs better with BoW and its variations, while logistic regression and SVM performs better with TF-IDF and its variations. No traditional classifiers with lexicon feature perform well due to the neutral sentiment class as discussed.

Naïve Bayes classifiers have the highest macro F1 scores below 0.58, while logistic regression classifier has 0.593 and SVM has 0.599. While it suggests both logistic regression and SVM classifiers perform better than the NBs, their best performances do not differ much.

7.2 Neural classifiers

Table 5 shows the average macro F1 score of the best model and respective epochs taken under different methods of deriving the hidden states.

Methods	Final hidden state	Mean Pooling	Max Pooling
Macro F1 score	0.224	0.0.632	0.62
Epochs	6	13	9

Table 5: Average macro F1 score of the best LSTM model across test sets. Note that the best model is determined based on the development set. It is possible that the model is not the best with regards to test sets.

Figure 2 shows maximum, minimum and average macro F1 score of the neural classifier after varying number of epochs.

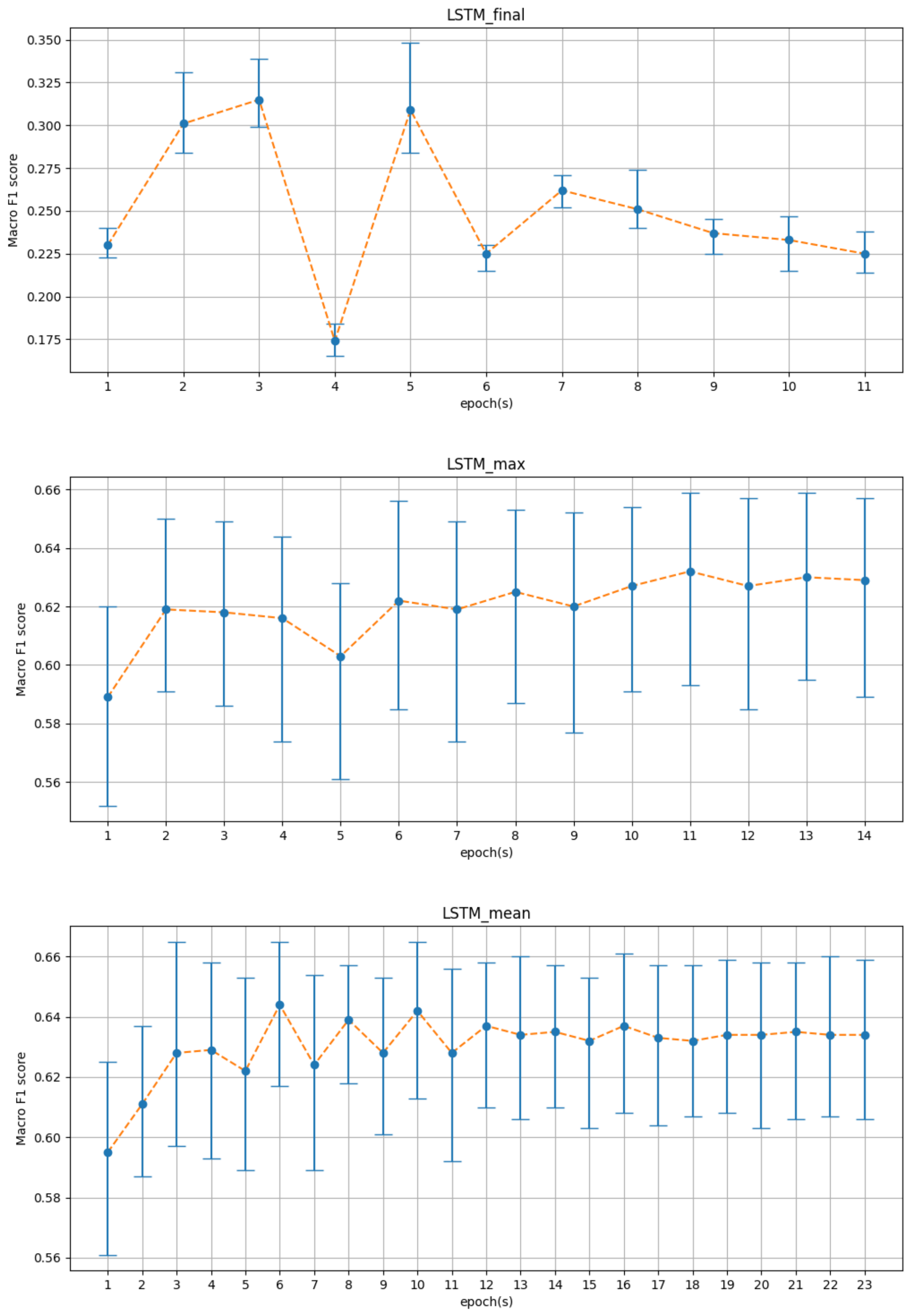


Figure 2: Average macro F1 score across three test sets after n epochs.

We observe that the best model with final hidden state has a maximum macro F1 score below 0.30. This suggests that deriving hidden states by their final one is not suitable for sentiment analysis. The range, i.e. the difference of maximum and minimum macro F1 score, fluctuates over epochs. Based on the plot, there are better models than 'best' model based on test sets. This suggests inconsistent performance of the model.

The best model with max pooling has an average macro F1 score of 0.62, which is much higher than using final hidden state. The range is stable at 0.06 level, which suggests a more consistent performance of the model.

The best model with mean pooling has an average macro F1 score of 0.632, which is the highest amongst three. The range is also stable at 0.06 level. The score bar also tends to be constant after epochs of training. This suggests a more consistent performance of the model.

While it takes more epochs to find the best model with mean pooling, we re-ran the training with new model for a few times and observe that this is due to randomness of the dropout.

7.3 Potential improvements & Future directions

7.3.1 Sarcasm

We note that we did not deal with potential sarcasm in tweets, which, as a result, can confuse the classifiers. For example, the sentence

"*Congratulations, Liz! You have broken the world record of being the shortest-serving prime minister in British history!*"

has positive words like "congratulations" but the whole sentence is negative. Sarcasm detection can become a good future direction.

7.3.2 Rough pre-processing

As mentioned, we note that stopword removal leads to a worse performance, possibly due to the neutral sentiment class. However, we can fine-tune to remove only a subset of stopwords. This may improve classifiers' performance.

Instead of removing #hashtags, we can deal with the potential noise and obtain sentiment information.

8 Conclusion

While a well-trained neural classifier has higher performance than traditional classifiers, training also takes much longer. If efficiency is prioritised, we can consider SVM classifier with TF-IDF with n-gram feature. For future works, we can consider sarcasm detection and fine-tuning pre-processing.

References

[1] C. Baziotis, "emoticons.py," GitHub repository, 5 December 2017. Accessed: 25 March 2025. [Online]. Available: `https://github.com/cbaziotis/ekphrasis/blob/master/ekphrasis/dicts/emoticons.py`.

Appendix

A Plots of macro F1 score

Figures 3, 4, 5 and 6 show plots of maximum, minimum and average macro F1 scores of each pair of traditional classifier and feature across given test sets.

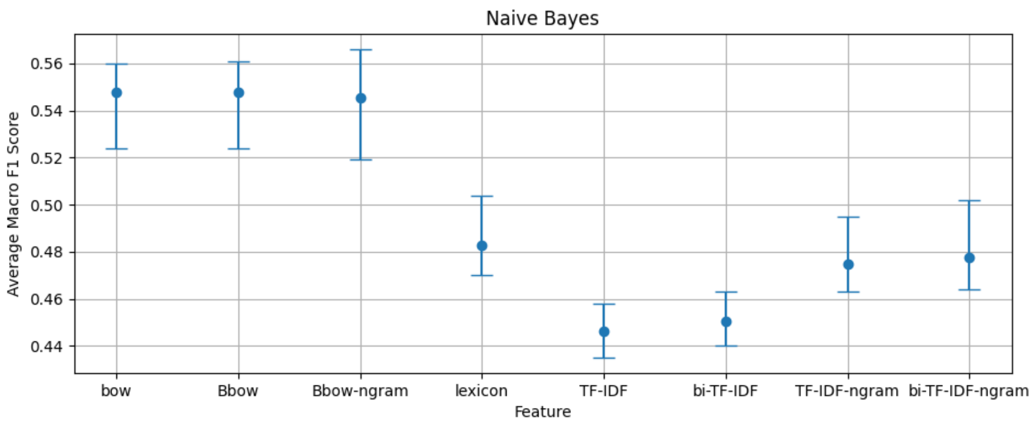


Figure 3: Maximum, minimum and average macro F1 scores of Naïve Bayes classifier with varying features (produced using `matplotlib.pyplot`).

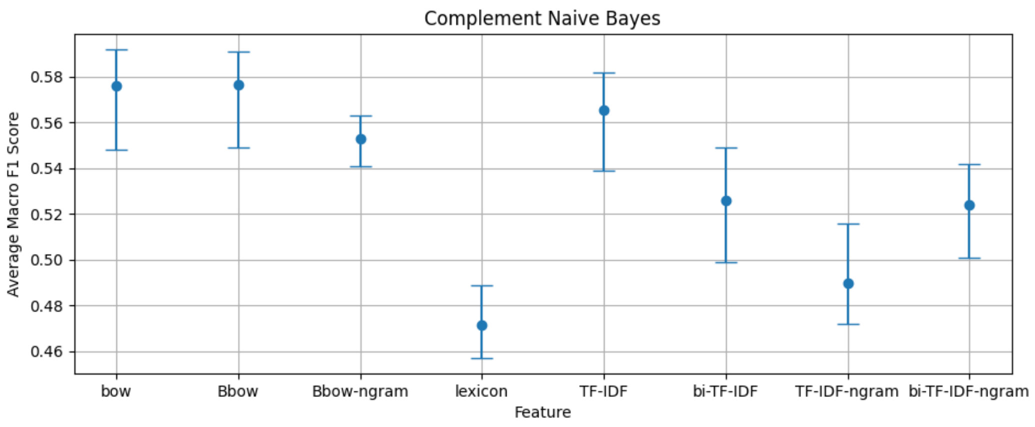


Figure 4: Maximum, minimum and average macro F1 scores of Complement Naïve Bayes classifier with varying features (produced using `matplotlib.pyplot`).

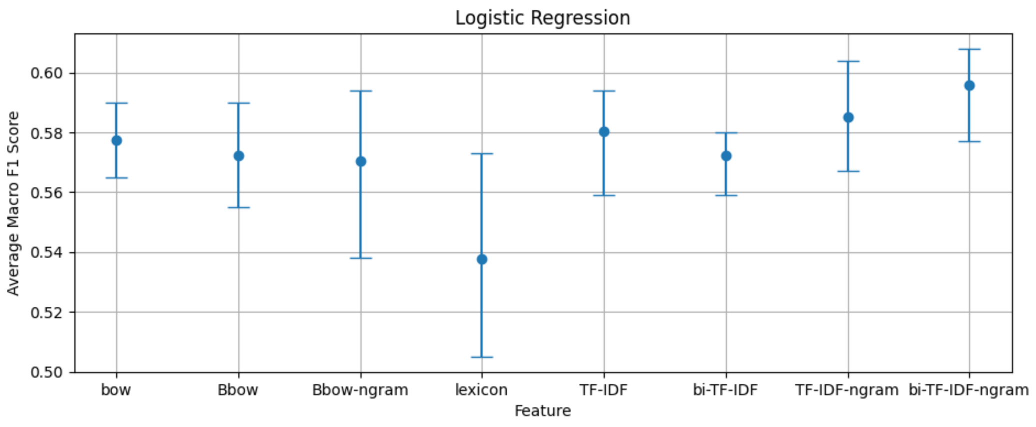


Figure 5: Maximum, minimum and average macro F1 scores of Logistic Regression classifier with varying features (produced using `matplotlib.pyplot`).

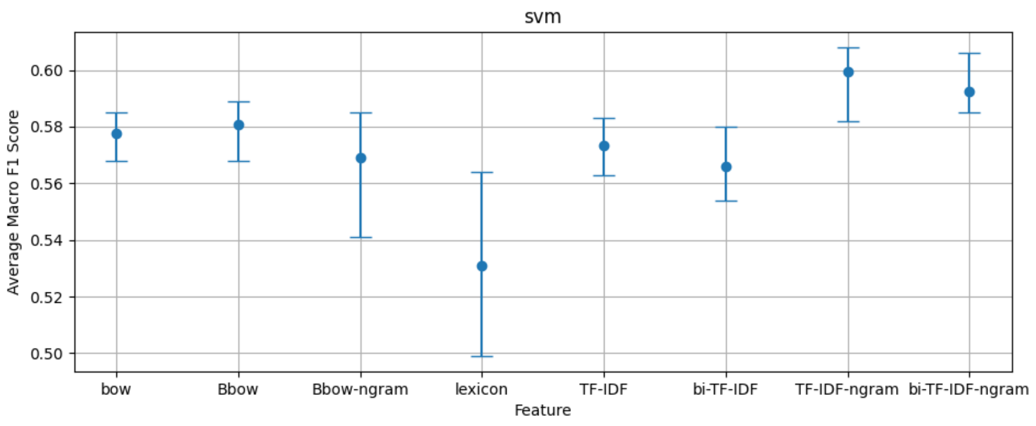
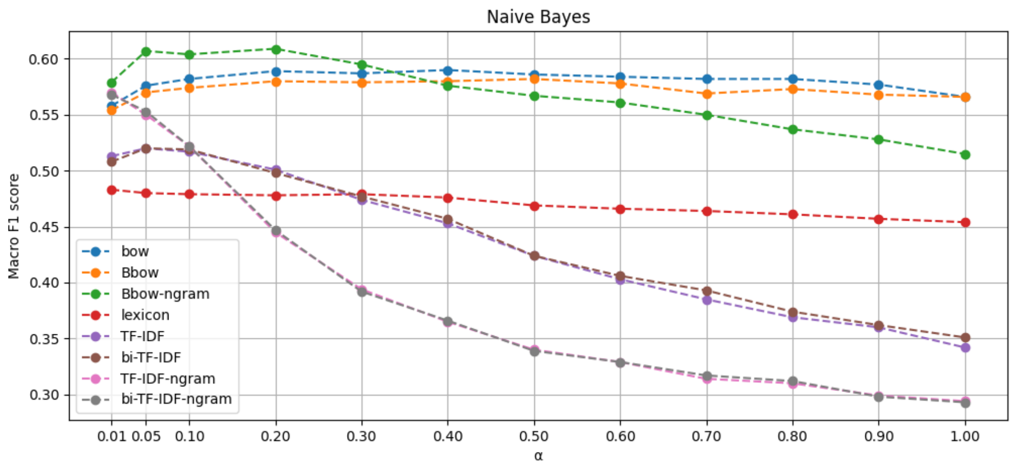


Figure 6: Maximum, minimum and average macro F1 scores of SVM classifier with varying features (produced using `matplotlib.pyplot`).

B Choosing α for additive-α smoothing for Naïve Bayes classifiers

For each feature and $\alpha \in [0.01, 1]$, we obtain the macro F1 score of both Naïve Bayes classifiers on tweet development set as in Figure 7. For NB classifier, we note that BoW and BoW with n-gram performs better in general. Therefore, we only zoom in the ideal α values for these two features and obtain Figure 8. For CNB classifier, we note that lexicon and TF-IDF with n-gram perform poorer in any α values, so we only zoom in the ideal α values for other features and obtain Figure 9.



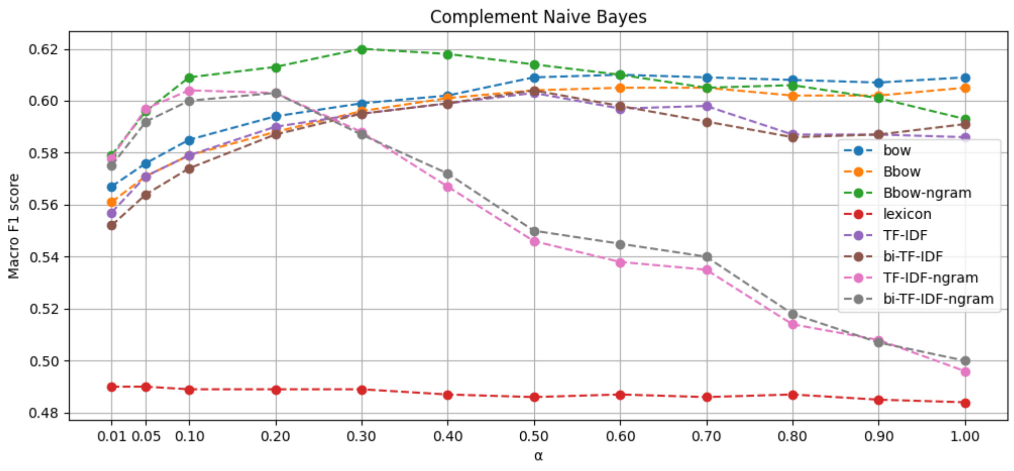


Figure 7: Plots of macro F1 scores of two Naïve classifiers (produced using `matplotlib.pyplot`).

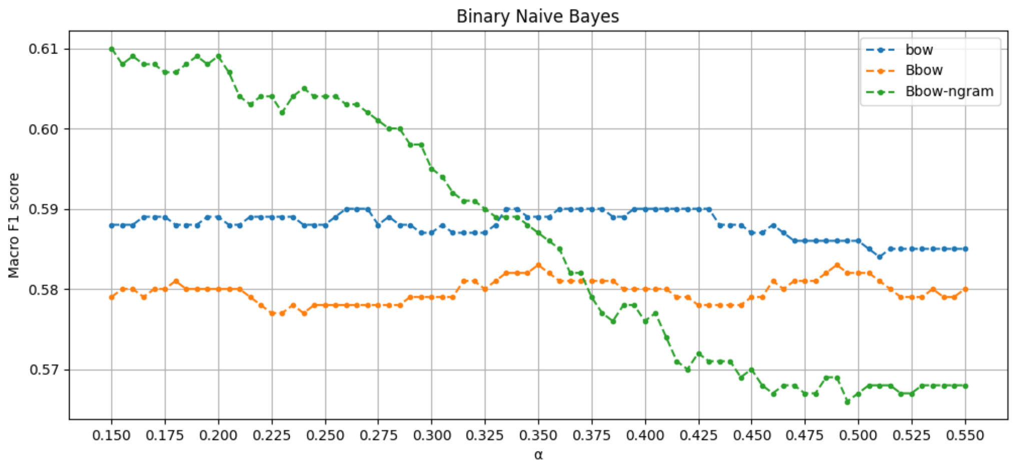


Figure 8: Plot of macro f1 score of NB classifier (produced using `matplotlib.pyplot`).

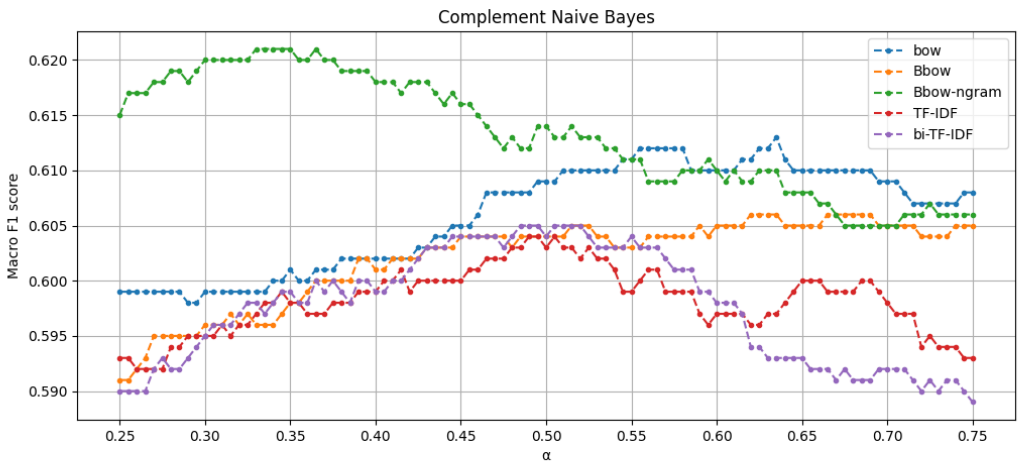
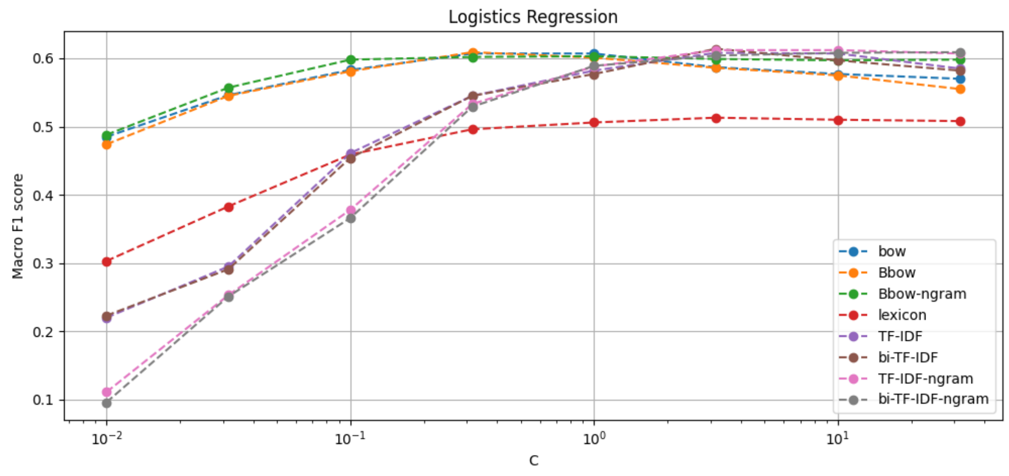


Figure 9: Plot of macro f1 score of CNB classifier (produced using `matplotlib.pyplot`).

Finally, we obtain the ideal α values in Table 1.

C Choosing λ for regularisation

For each feature and for a number $C = \frac{1}{\lambda}$ values, we obtain the macro F1 scores of both Logistic Regression and SVM on tweet development set as in Figure 10.



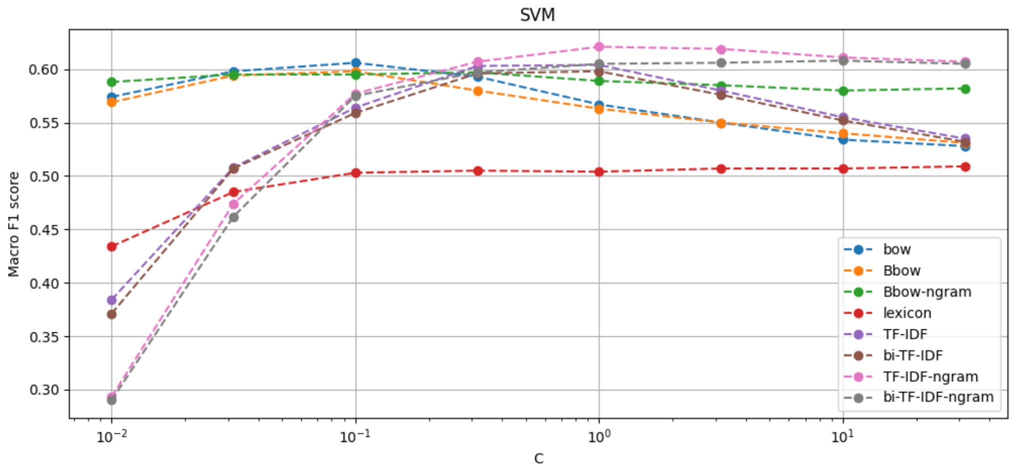


Figure 10: Plots of macro F1 scores of Logistic Regression and SVM (produced using `matplotlib.pyplot`.