Sentiment Classification for Social Media

CS918: Assignment II

Student ID: 2024991
University of Warwick
December 2020

1 Introduction

This report outlines the approach taken to the sentiment classification of twitter messages challenge in SemEval-2016 Task 4. The second section of this report discusses the preprocessing techniques adapted to clean twitter messages and then moves to the third section which elaborates on the features extracted from the provided datasets. The three models used to classify tweet sentiment are introduced in section four and an evaluation of the models' accuracy is carried out in the last section.

2 Corpus Preprocessing

The *preprocess* function is called at the start of the notebook execution which extracts the tab-separated tweets into the following lists:

- Tweet ID list
- Tweet Sentiment list (positive/negative/neutral)
- Unprocessed Tweet message list

The function iterates through the individual datasets, namely the training and test sets. For the purpose of parameter tuning, the development set was also included during development.

Regex functions are called during preprocessing which clean the corpus messages. Table 1 shows the regex substitutions carried out during this step together with corresponding transformation examples. The function ends by tokenising each tweet in a concatenated list, and returns the tokenised preprocessed content, the tweet sentiment and the tweet id.

The *lemmatize_content* function takes the tokenised preprocessed content as input and starts by assigning Part-Of-Speech (POS) tags to each word (such as 'NN' for nouns and 'JJ' for adjectives). The function calls the *treebank_pos* function which abbreviates the treebank tags (e.g. tags starting with 'V' transformed to 'v') and lemmatizes input words to their root form. *Stopwords* are removed after lemmatisation is done. A tokenised and concatenated preprocessing lemmatised list of tweets is returned.

A comparison was done between the WordNetLemmatizer with integrated POS tagging against a PorterStemmer as shown in Table 2. Whilst PorterStemmer is faster to compute, it can be concluded that WordNetLemmatizer better normalises words to their root form.

Transformation Description	Example	
	Transformation From	Transformation To
@user	@NLPUser	USERNAME
http link	http://nlpisfun.com	URLLINK
Happy faces and expressions	:) or lol	HEMOTICON
Sad faces and expressions	:(	SEMOTICON
Repeated letters in a word	heeello	hello
White space characterss	Tabs	Empty Space
End of sentence symbols	. or !	END
Non-alphanumeric characters except spaces	+	Empty Space
Pure digits	10	Empty Space
Single letter words	a	Empty Space

Table 1: Preprocessing transformations description coupled with transformation examples for illustration purposes

Original Tweet Message	PorterStemmer	WordNetLemmatizer
Felt privileged to play Foo Fighters songs on guitar today with one of the plectrums from the gig on Saturday.	felt privileg play foo fighter song guitar today one plectrum gig saturday end	felt privilege play foo fighter song guitar today one plectrum gig saturday END
Happy Birthday to the coolest golfer in Bali! @tjvictoriacnd !! :) may you become cooler and cooler everyday! Stay humble little sister! Xx	happi birthday coolest golfer bali end usernam end hemoticon may becom cooler cooler everyday end stay humbl littl sister end xx	happy birthday cool golfer bali END USERNAME END HEMOTICON may become cooler cooler everyday END stay humble little sister END xx

Table 2: Comparison of lematisation and stemming of two tweets using Porter Stemmer and WordNet Lemmatizer. It can be concluded that the WordNet Lemmatizer better transforms words to their root form.

3 Feature Extraction

This section describes the different features extracted from input datasets. The evaluation and contribution of each feature to the models' performance is later discussed in the Evaluation section.

3.1 Bag of Words

The *bag_of_word* function returns the frequency of tokens in a tweet as a matrix. The matrix is limited to the top 500 most occurring words in the corpus.

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

The TF-IDF is a term weight word embedding approach which firstly computes the normalised term frequency and then the inverse document frequency. The former assigns a weight to each word based on its occurrence in a corpus whilst the latter diminishes weights for frequent occurring words and increases the weights for rarely occurring words.

TF-IDF feature matrices are constructed at two different levels:

1. *Word level TF-IDF* which returns a matrix having TF-IDF scores for every term in the corpus. A 2000 maximum features term frequency is capped within the vectoriser.
2. *N-Gram level TF-IDF* which computes a TF-IDF matrix over bigram and trigrams. A maximum feature set of 5000 is used in this case.

3.3 Gensim word-to-vector (word2vec)

Contrary to TF-IDF which relies on the count of each occurrence in a corpus, the Gensim word2vec approach returns a weight matrix based on how probable a word is to occur near another word. A maximum number of 500 features is used for this prediction based approach and a skip-gram model is adapted.

3.4 Glove Embebbing

Glove embedding incorporates a combination of the previously explored word embedding features by combining the count-based approach (similarly to the TF-IDF) as well as the prediction based approach (similarly to word2vec).

Given the assignment requirement specification, the embedding matrix is limited to a maximum number of 5000 tokens and an embedding dimensionality of 100. The *glove_LSTM_model* function starts by converting the Glove file into its vectorised form and stores the vector for each word in the dictionary *embedding_dict*. It continues by creating a zero padded NumPy array of size (5000,100). The function loops through the most common words and stores the respective word index and vector representations in *embedding_matrix*.

3.5 Sentiment by Lexicon

The *sentiment_by_lexicon* function iterates through words in a tweet and compares them to predefined positive and negative words in *positive-words.txt* and *negative-words.*txt files. A count of positive and negative word occurrence is recorded for each tweet and an array of results is returned as a feature set.

3.6 Sentiment by Emoticon

Similarly to the sentiment by lexicon, the *sentiment_by_emoticon* function iterates and count the frequency of happy emoticons (denoted by 'HEMOTICON' in the preprocessed tweets) and the frequency of sad emoticons (denoted by 'SEMOTICON').

4 Classification

4.1 Naïve Bayes Classifier

Two variants of the *Naïve Bayes* classifier were explored: *Multinomial* and *Gaussian Naïve Bayes*. The former was tested against the TF-IDF feature set (given that it cannot handle negative probability values returned by the word2vec matrix) whilst the latter was tested with the Gensim word2vec features respectively. Sentiment lexicon and emoticon features were also added and

their contribution on the development F1-score was analyzed. An exhaustive *GridSearchCV* cross-validation was also computed, to obtain the optimal model hyperparameters.

Table 3 shows the results obtained by both Multinomial and Gaussian Naïve Bayes classifiers using a combination of features. The bag of words feature returns poor results as expected. The word2vec embedding returns less accuracy than the TF-IDF concatenated word and n-gram level embedding. It can be concluded that the Multinomial Naïve Bayes with the word and n-gram level TF-IDF, lexicon and emoticon feature set and hyperparameters of alpha=1 and fit_prior set as True yields the best score.

Features	Model	Development Set F1-Score
Word level TF-IDF	Multinomial Naïve Bayes	0.523
N-gram level TF-IDF	Multinomial Naïve Bayes	0.426
Word and N-gram level TF-IDF	Multinomial Naïve Bayes	0.583
Word and N-gram level TF-IDF, Lexicon	Multinomial Naïve Bayes	0.622
Word and N-gram level TF-IDF, Lexicon and Emoticon	**Multinomial Naïve Bayes**	**0.625**
Bag of words	Multinomial Naïve Bayes	0.260
Bag of words, Lexicon and Emoticon	Multinomial Naïve Bayes	0.345
Word2Vec	Gaussian Naïve Bayes	0.517
Word2Vec, Lexicon	Gaussian Naïve Bayes	0.522
Word2Vec, Lexicon and Emoticon	Gaussian Naïve Bayes	0.523

Table 3: Naïve Bayes feature set comparison. The combination of word and n-gram level TF-IDF, lexicon and emoticon feature sets return the optimal F1-score when the Naïve Bayes classifier is tested on the development dataset

4.2 Maximum Entropy (Logistic Regression) Classifier

A similar approach was followed for the Maximum Entropy classifier. Table 4 shows the F1-scores obtained for different feature set. The same conclusion can be drawn out, that is, the Maximum Entropy model reaches the optimal F1-score when the word and n-gram level TF-IDF, lexicon and emoticon feature set is used. Hyperparameters of C=1 and penalty L2 yield the optimal results.

Features	Development Set F1-Score
Word level TF-IDF	0.598
N-gram level TF-IDF	0.443
Word and N-gram level TF-IDF	0.603
Word and N-gram level TF-IDF, Lexicon	0.620
Word and N-gram level TF-IDF, Lexicon and Emoticon	**0.622**
Bag of words	0.251
Bag of words, Lexicon and Emoticon	0.389
Word2Vec	0.524
Word2Vec, Lexicon	0.561
Word2Vec, Lexicon and Emotico	0.564

Table 4: Maximum Entropy feature set comparison against the development set F1-score obtained. The word and n-gram level TF-IDF, lexicon and emoticon feature sets return the optimal score.

4.3 Long-Short Term Memory (LSTM) Classifier

The LSTM model is trained in the *glove_LSTM_model* function. It takes the glove embedding matrix as input feature set. The maximum tweet length based on the tweet having the longest number of words in the training set is determined and consecutively, shorter tweets are padded and longer tweets are truncated. The model shown in the Figure 1 yields an optimal F1-score on the development set. The embedding layer was provided with the *embedding_matrix* as input weights, and it was frozen to avoid the weights from being modified during training.

Two cascaded LSTMs with an intermediate dropout of 0.2 and an 'rmsprop' optimiser is applied during model compilation. A batch size of 100, validation split of 20% and a total of 10 epochs are enough for the model to minimises the loss function. Figure 2 shows that the model converges to an optimal validation set accuracy at around 10 epoch. The model starts overfitting on the training set and the validation set's loss starts increases beyond the 10 epochs. An F1-score of 0.608 is achieved when the LSTM model is tested against the development set.

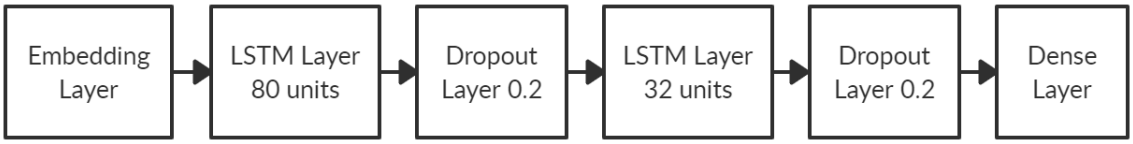

Figure 1: The LSTM model with an Embedding Layer, 2 LSTM layers having integrated dropouts and a Dense layer at the output.

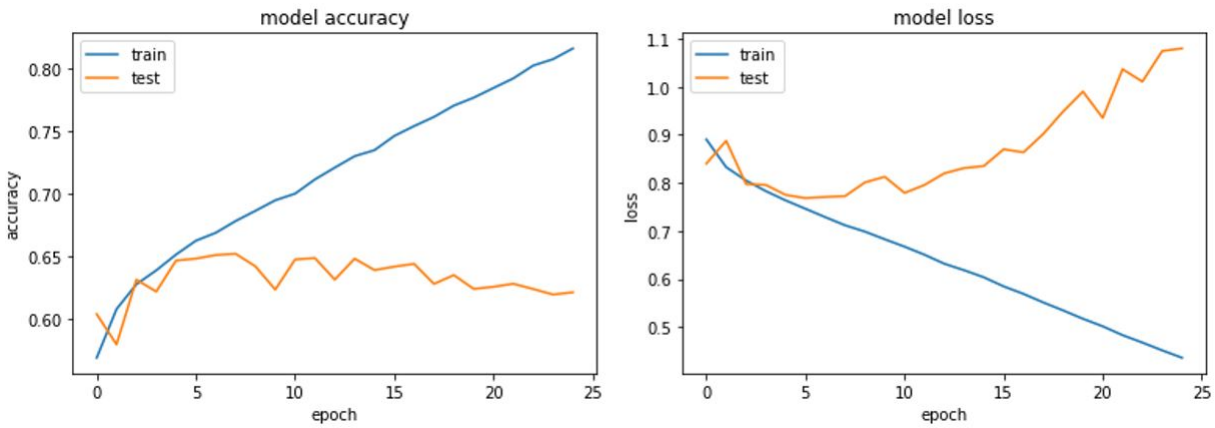

Figure 2: The optimal validation set accuracy is achieved at 10 epochs. The model start overfitting on the training set beyond 10 epochs and the validation set's loss starts increasing.

5 Evaluation

The models' accuracy was improved by performing error analysis on incorrectly classified tweets. It was analytically identified that mis-classified tweets had lexicon expression words such as 'frustrated' or 'enjoyable' which contributed towards significantly increasing the model's accuracy. The *sentiment by lexicon* feature set was added after error analysis was carried out.

The models' accuracy was also fine-tuned by analytically identifying additional happy and sad emoticons, such as 'lol' from incorrectly classified tweets, and incrementally adding them to the 'HEMOTICON' and 'SEMOTICON' regex substitutions respectively. Table 5 shows the final F-scores obtained by each model.

Model	F1-Score			
	Development Set	Test Set 1	Test Set 2	Test Set 3
Naïve Bayes	0.625	0.565	0.593	0.567
Maximum Entropy	0.622	0.571	0.597	0.551
LSTM	0.608	**0.614**	**0.641**	**0.578**

Table 5: Model F1-score performance comparison against different sets used. The LSTM model generalises better and yields a higher score for the test sets.

Although a higher F1-score is achieved using the Naïve Bayes and the Maximum Entropy models when tested on the development set, the LSTM model generalises better and returns higher scores when tested on unseen test datasets.

6 Conclusion

This report showed the approach taken in modelling the SemEval-2016 Task 4 challenge. The methodology of preprocessing twitter messages was firstly analysed. This step was critical for the sections that followed, as high model accuracy cannot be achieved if the tweet messages are not cleaned and normalised.

A variety of features were also explored. The bag of words embedding proved to be ineffective in increasing the models' accuracy. The Gensim word2vec returned less accuracy the that concatenated word and n-gram level TF-IDF embedding. Finally, the sentiment by lexicon and the sentiment by emoticon feature set further enhanced the model's accuracy when combined with the TF-IDF features.

Three classifiers were explored in this assignment. The Naïve Bayes and Maximum Entropy models returned similar scores. They performed better than the LSTM when tested on the development set. However, the LSTM returned higher scores when tested on unseen test datasets. This means that the LSTM generalised better than the former two models.