CS918 Natural Language Processing Assignment 2

Andrew Bell

March 18, 2023

1 Methodology

1.1 Development

Throughout this project, I followed a simple iterative design and development process:

1. Research and implement a process, feature or model and apply it to training data
2. Validate results with the development data
3. Adjust or change according to validation results

In step 1, I mainly looked at previous approaches and general implementation tutorials for various library functions.

In step 2, I mainly considered the SemEval F1 score, which is the macro average over the positive and negative class F1 scores. I also considered the confusion matrices and some error samples. Error samples often were ambiguous even to me, so I was largely unsuccessful in finding further linguistic features to capture sentiment.

In step 3, I iteratively made changes to increase the SemEval metric, or decided against steps, such as stop word removal (see section 2.3).

1.2 Evaluation

The final evaluation was performed once all training was complete. I saved the models, disabled the training scripts and loaded the models in evaluation mode.

At this point I used the three test sets and evaluated each model, producing scores for each training set for each neural model, and the best linear one. For the feature/classifier combinations, I used a mean over all three test sets.

2 Preprocessing

2.1 Emojis

The first step was to replace emojis, which are fairly common in social media text. In the task of sentiment classification, it is useful to transform these symbols into their meaning. In my implementation, I use an emoji dictionary[1] which maps descriptions to emojis. This is used to convert any emojis found into their descriptions, and extract only 'sentiment' words from this description. This means an emoji with description :smiling_face: simply becomes "smiling". To do this I utilised the NLTK opinion lexicon, and consider any word in the positive or negative list to be a 'sentiment' word. This avoids adding unnecessary tokens (noise) that don't indicate sentiment.

2.2 Slang

Another common inconsistency between formal texts and social media text is slang, which are words, acronyms or abbreviations that are not typically used in formal writing. To help normalise the text

[1]https://www.kaggle.com/datasets/divyansh22/emoji-dictionary-1?resource=download

somewhat, I employed a simple slang dictionary[2], which was parsed, before checking words against it. Slang words (according to this dictionary) were replaced by their “regular” English equivalent. This list covers only a small subset, but does regularise many common slang words.

2.3 Cleaning

The following list gives the cleaning steps, implemented using regular expressions. This exact order was necessary to prevent one step from impacting the next.

1. Replace positive emoticons :) $\longrightarrow$ “happy”
2. Replace negative emoticons :($\longrightarrow$ “sad”
3. Remove twitter mentions e.g @user
4. Remove websites [3]
5. Remove non-alphanumeric characters
6. Remove one character words
7. Remove numeric words
8. Replace consecutive repeated characters with two of the repeated character e.g yayyyy to yayy.[4]

Simple stop-word removal was not used, as this has been shown to reduce sentiment classification performance [1]. My experiments concur with this study.

Hashtags were not removed as the symbol ’#’ is removed in step 5 and the content can often provide sentiment information e.g #amazing.

2.4 Lemmatisation

To lemmatise, I used NLTK, with a part of speech (POS) tagger, to make the lemmatisation more accurate. The POS tagger gives results in Penn Treebank format, which were converted into WordNet standard and passed to the lemmatiser.

3 Linear Classification

3.1 Features

The following features were utilised to transform the text into a meaningful representation for use with linear classifiers.

- TF-IDF Features
- Opinion Lexicon Features
- Word Embedding Features (GloVe)

TF-IDF Features represent each tweet by a ∼43000 long vector which is the size of the training set’s vocabulary. Each element in the vector represents a token, and the element is the frequency of that token in the current tweet, multiplied by the inverse frequency of that token in all tweets. This was an effective way to normalise the vector such that the large values were not dominated by very common tokens (with little meaning). The vectors were computed using scikit-learn without performing any prior tokenization.

[2]https://www.kaggle.com/code/nmaguette/up-to-date-list-of-slangs-for-text-preprocessing/notebook
[3]Websites of the form“name.domain” will get through unless it’s a common domain
[4]Doesn’t always change to correct word, but does normalise

Lexicon Features represent each tweet T by a vector of size 2. The first entry is given by $lf(T)$ of the positive list, and the second entry is given by $lf(T)$ of the negative list. Tokenisation is performed prior to extracting this feature, using NLTK's word_tokenize().

$$lf(T) = \frac{1}{N} \sum_{i=1}^{N} presence(t_i)$$

where $presence(t_i) = \begin{cases} 1, & \text{if } t_i \text{ in list} \\ 0, & \text{otherwise} \end{cases}$
and $T = t_1, t_2, ..., t_N$

Word Embeddings Features represent each tweet as a 100 long vector. I used the GloVe 6B pre-trained embeddings [2], which have a dimensionality of 100. Again using NLTK's word_tokenize() I generated a list of tokens for each tweet, and took the mean of the word embedding vector over all tokens in a tweet. If a token was not in the GloVe pre-trained embeddings, I used the mean value of all existing embeddings.

3.2 Classifiers

The following linear classifiers were utilised for this task:

- Gaussian Naive Bayes (GNB)
- Support Vector Machines (SVM)
- Maximum Entropy (MaxEnt)
- Random Forest (RF)

Each classifier was imported from scikit-learn and trained using the training data, encoded according to each feature vector defined in section 3.1. Some features are incompatible with certain classifiers. For example the TF-IDF features were too large and sparse to be used effectively with GNB, and as such were not combined.

3.3 Feature & Classifier Results

I tested each combination of feature and classifier (where possible) on all three test sets. Each combination gets a score which is the mean of the SemEval F1 scores over the three test sets. The results can be seen in table 1. The bold entries indicate the best classifier for each feature. GloVe word embeddings were used most effectively by the SVM classifier, whereas the low dimensionality lexicon features were used most effectively by random forest. Naive Bayes was the best classifier for the combination feature of word embeddings and lexicon features. The TF-IDF features were used most effectively by the maximum entropy classifier.

The main observation from this experiment is that each feature performs best with a different classifier. This is very important to consider when designing models, and suitable classifier(s) must be used for the chosen feature(s).

Table 1: Mean SemEeval Macro F1 scores over three test sets

	Classifier			
Features	MaxEnt	SVM	GNB	RF
GloVe	0.450	**0.521**	0.474	0.339
Lexicon	0.400	0.296	0.390	**0.463**
GloVe+Lexicon	0.492	0.483	**0.530**	0.452
TF-IDF	<u>**0.562**</u>	0.505	N/A	N/A

4 Neural Models

The three models were trained according to their respective hyperparameters given in table 2. Weighted cross entropy refers to a multi class cross entropy loss that is biased towards the postive and negative classes. Specifically, I used weights [positive= 1, neutral= 0.5, negative= 1]. I found this to have a small negative effect on accuracy, but a significant increase in the SemEval f1-score. This is because the SemEval score is taken over the positive and negative classes, whose F1 scores are impacted more by their own respective assignments than by neutral assignments.

Training of the final models was performed on the lab computer with an Nvidia GTX 3060 GPU. While the LSTM models can be trained with a CPU for fewer epochs, the BERT model requires a graphics card to train. Evaluation was also performed using the GPU, but can be done using a CPU.

Table 2: Training hyperparameters of the three models

	Model		
Parameter	Base-LSTM	Bi-LSTM	BERT
Epochs	30	30	3
Training Time (mins)	< 1	< 1	~6
Optimiser	Adam		
Loss Function	Weighted Cross Entropy		
Learning Rate	0.001		
Training Device	GPU		

4.1 Base LSTM

I implemented the base LSTM model according to the given specification. This included a fixed word-embedding layer, generated from the same glove vectors that were previously parsed. Then I used a 2-layer single directional LSTM with a hidden dimensionality of 32. The next layer is a Tanh activation function, followed by a linear layer which outputs a vector of size 3. I used a dropout ratio of 0.4 to reduce the chance of overfitting.

To prepare data for the LSTM, I wrote a tokenizer, which can be fit to training data and then used to transform all data. I ran tokenizer.fit() on only the training data, giving an embeddings matrix. Then I ran tokenizer.transform() on all data (training, dev, test) to convert the data into indexes in said embeddings matrix.

In figure 1, a graph of a previous training loss and F1 scores for this model can be seen over 50 epochs. The training and validation loss converge around 30 epochs, and it was therefore I retrained with 30 epochs, to ensure the models generalise.

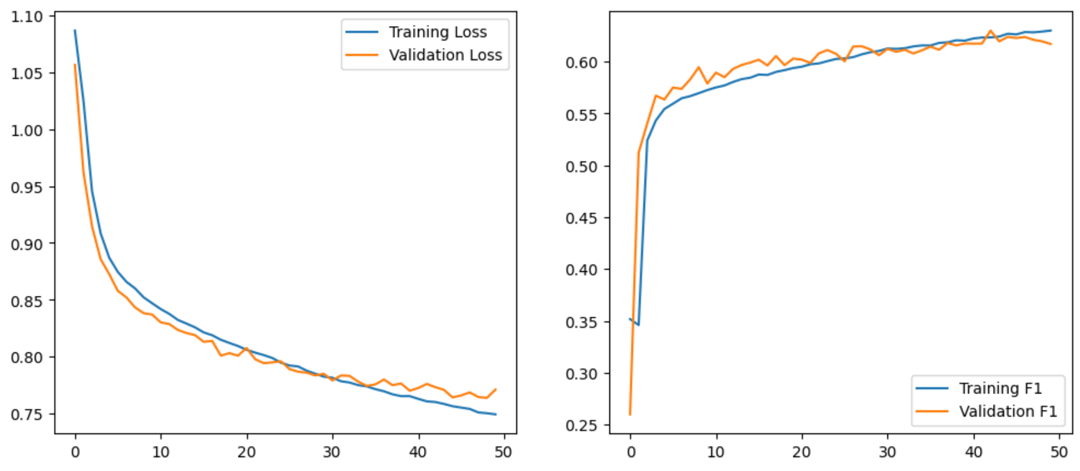

Figure 1: Training graph for LSTM-base, with loss and F1 scores over 50 epochs

4.2 Bidirectional LSTM

The best performing team in the 2017 Semeval task 4-a utilised an ensemble model, using CCNs and LSTMs [3]. The LSTM model was a bidirectional LSTM stack of size 2. Taking inspiration from this, I utilised a 2 layer bidirectional LSTM, with hidden dimension of 32 to match that of my base model. The other difference to the base model is the input to the linear layer, which has a doubled input size to account for the doubled outputs of the Bi-LSTM.

The other components such as the fixed word embeddings and dropout were identical to the base LSTM.

4.3 BERT classifier

To achieve better performance on this task, I utilised Transformer based models. Since Transformers require large amounts of training time and computational power, I utilised transfer learning, by taking a pre-trained transformer encoder and applying a linear layer on its output.

The pre-trained model I utilised was BERT (Bidirectional Encoder Representations from Transformers) [4]. This is a popular choice for research given its manageable size but effective power in many tasks. Because it has been pre-trained on a huge dataset, I was able to leverage the model by applying a simple linear layer to classify our task. I utilised the Transformers library from HuggingFace to retrieve and use this model.

Specifically I used BERT-base, whose parameters were frozen so they were not updated. I trained the linear layer on top of it, which converged at 3 epochs without further improvement. This is because BERT is pre-trained and contains enough meaningful parameters to facilitate very minimal fine tuning to the task. Inputs were processed using the the transformers library's BERT tokenizer, before feeding them into this model.

5 Results

I tested the three neural models on each test set and report the results in table 3 alongside the best performing linear classifier. All three neural models all outperform the best linear one, and the best performing sentiment classifier was the BERT-based model.

My experiments only show a marginal increase in score for the bidirectional LSTM model, over the single directional one. This suggests that future context does not encode as much useful information in this task as we expect. Nevertheless, the testing data may be too limited to conclude this, and further tests on final two test sets may reveal more.

Comparing the original results from the 2017 SemEval task 4-A [5], I achieve lower accuracy in my LSTM models than the more sophisticated LSTM-based systems such as BB_twtr who make use of an additional convolutional model [3] and DataStories who make use of attention mechanisms [6].

However my BERT model, as expected, outperforms all the systems from 2017 in the SemEval macro averaged F1 score. To compare this to a more recent system, consider TwitterBERT, which is an ensemble learning framework making use of BERT as well as LSTMs and CNNs [7]. TwitterBERT achieves an F1 score of 0.719 on the SemEval 2017 dataset. My BERT model has far less complexity but was able to achieve a slightly lower mean SemEval F1 of 0.689. This demonstrates the power of large pre-trained transformer based models.

Table 3: SemEval Macro-F1 results of the four main models

Model	Test set 1	Test set 2	Test set 3	Mean
TF-IDF+MaxEnt	0.570	0.572	0.545	0.562
GloVe+Base-LSTM	0.599	0.575	0.578	0.584
GloVe+Bi-LSTM	0.598	0.575	0.594	0.589
BERT+Linear	**0.707**	0.698	0.662	0.689

5.1 Reproducibility

Using the provided Jupyter notebook, my results are reproducible by taking the steps outlined in readme.txt. However, the exceptions are the SVM and RF classifiers, which despite seeding still produce different random results, but are always close to the reported results. Neural model performance can only be reproduced if training is performed on a GPU, otherwise performance may vary slightly. However, if using the provided pre-trained models, evaluation should be deterministic and give the same results (up to floating point errors) on a CPU.

References

[1] H. Saif, M. Fernandez, Y. He, and H. Alani, "On Stopwords, Filtering and Data Sparsity for Sentiment Analysis of Twitter," in *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*. Reykjavik, Iceland: European Language Resources Association (ELRA), May 2014, pp. 810–817.

[2] J. Pennington, R. Socher, and C. D. Manning, "GloVe: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543.

[3] M. Cliche, "BB_twtr at SemEval-2017 Task 4: Twitter Sentiment Analysis with CNNs and LSTMs," in *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*. Vancouver, Canada: Association for Computational Linguistics, Aug. 2017, pp. 573–580.

[4] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186.

[5] S. Rosenthal, N. Farra, and P. Nakov, "SemEval-2017 Task 4: Sentiment Analysis in Twitter," in *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*. Vancouver, Canada: Association for Computational Linguistics, Aug. 2017, pp. 502–518.

[6] C. Baziotis, N. Pelekis, and C. Doulkeridis, "DataStories at SemEval-2017 Task 4: Deep LSTM with Attention for Message-level and Topic-based Sentiment Analysis," in *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*. Vancouver, Canada: Association for Computational Linguistics, Aug. 2017, pp. 747–754.

[7] N. Azzouza, K. Akli-Astouati, and R. Ibrahim, "TwitterBERT: Framework for Twitter Sentiment Analysis Based on Pre-trained Language Model Representations," in *Emerging Trends in Intelligent Computing and Informatics*, ser. Advances in Intelligent Systems and Computing, F. Saeed, F. Mohammed, and N. Gazem, Eds. Cham: Springer International Publishing, 2020, pp. 428–437.