CS918 Assignment 2: Sentiment Classification

u5525549

March 20, 2024

Abstract

Two types of features and a total of four classifiers are used. The bag-of-word features are used with traditional machine learning methods MaxEnt and Naïve Bayes, while the GloVe word embeddings are used with LSTM. Another attempt was to use BERT Transformer for feature extraction and adding a fully connected layer for the classification task.

1 Preprocessing

Each tweet is preprocessed in the following order:

1. Remove "RT" (retweet);

2. Replace consecutive dots (ellipsis, "...") with one single white space (this is done before removing URLs, so that cases like "tomorrow...please" will not be recognized as a URL;

3. Covert to lower case;

4. Remove URLs, user Mentions and hashtags;

7. Remove non-alphanumeric symbols (but keep ".", "'", "-", ":" and "," at this stage);

8. Remove ".", "'" and "-" if they are not inside a word (between letters), this is to keep abbreviations like "i'm", "you're", "u.s", "u.s.a", and compound words like "anti-immigration";

9. Remove ":" and "," if they are not inside a number (between digits), this is to keep time like 7:30, and large numbers written in the form of 100,000. These two steps are done to map the tokens better with the GloVe word embeddings, as the GloVe vocabulary supports (includes) abbreviations ("'m", "'re", which will be separated from "i", "you" later using the NLTK tonkenizer, as well as time and large numbers using the thousands comma separator;

10. Replace multiple consecutive white spaces with one and remove spaces at the ends.

2 Feature Extraction

2.1 Bag-of-word

For bag-of-words, we remove the stop words and perform lemmatization after tokenizing the preprocessed tweets. Apart from the occurrence of words in each tweet, we also count the total occurrence of words in the training dataset, and words that occurred less than 10 times are discarded. At this step, the size of our vocabulary is reduced from 34,337 to 5,055. This is to improve the performance and efficiency of the model while reducing the impact of noise, the computational complexity and memory consumption (by cutting down the feature dimensionality). The bag-of-words representation for each tweet is a vector (list) with the length of 5,055. Each element of the vector represents the number of occurrence of the corresponding word in this tweet.

2.2 Word Embedding

For word embedding features, we keep the stop words and don't perform lemmatization, so that each tweet is tokenized into a list of "raw" words. This is because GloVe supports word in different forms (play, played, playing), so we may just keep the words in their original forms. We then calculate the average length of tweet from each dataset, which is around 18-19. In general, the length of most tweets should not exceed twice this average value. After experimentation, we can see the majority of tweets have a length of 30 or less. Therefore, we can take this as the max length, and perform truncation for any tweets longer than that without worrying about losing much information. Here the "raw" vocabulary has a size of 39,497, which is, naturally, larger than the number of lemmas. We then load the GloVe word embedding vectors and filter out the words that can be matched in GloVe's dictionary, and end up with 31,308 of them. An index mapping

to the vocabulary is built, similar to bag-of-word, with only the words that occurred more than 10 times in the training dataset, and two special tokens "<unk>" and "<pad>" for unkown tokens and padding. The word embedding feature representation for each tweet is a vector (list) with the length of 30, mapping (truncated) list of tokens to their reference word index.

3 Sentiment Classifiers

3.1 MaxEnt

For MaxEnt we use the LogisticRegression model from sklearn, the default max_iter=100 failed to converge so we had to set the maximum number of iterations to 1000.

3.2 Support Vector Machine (discarded)

All types of SVM models (linear, polynomial, RBF) took somewhere between 5 to 7 hours to train (fit and converge), and the trained model would take another 5 to 10 minutes to predict a validation set with only around 2,000 examples. Judging from the validation performance, the linear model is probably "good enough" (compared to MaxEnt and Naïve Bayes), while the two non-linear ones could be overfitting. But even the linear model is too slow to train and use, besides, the pickle file ended up larger than 1GB, so I decided not to use SVM for my final testing or to submit it.

3.3 Naïve Bayes

For Naïve Bayes we use the MultinomialNB model from sklearn.

3.4 LSTM PyTorch

We defined a customized dataset reader for PyTorch that maps labels (neutral, negative, positive) to integer values (0, 1, 2) and pad the index representation of tweets to length 30 with (the index for) "<pad>", which is loaded to PyTorch's data loader with batch size 32. The model has three layers, embedding, LSTM and fully connected. The embedding layer loaded from GloVe for our LSTM model has the shape 5890 * 100, which means 100-dimensional embedding vectors for 5,890 tokens, and is frozen during training. We also add some dropout for more robustness and better generalization. The model has a total of 403,803 trainable parameters. For linear layers, the weight matrix is initialized to follow the Xavier normal distribution, and the bias term is initialized to zero. For LSTM layers, we iterate through all parameters, during which the bias term is initialized to zero, and the weight matrix is initialized to be orthogonal. This initialization approach aims to facilitate faster convergence and better performance of the model.

We set the learning rate to 0.001, using Adam optimizer and cross entropy loss, and train the LSTM-based classifier for 50 epochs. For each epoch, we calculate the average training and validation loss as well as the macroaveraged F1 score (our target evaluation metric) and keep track of how these metrics change over time. We can see from the convergence plot that though the loss and macro F1 score on the training set keeps improving steadily in the 50 epochs, on validation set they stopped improving after around 20 epochs and began to fluctuate. We save model weights throughout iteration and load the ones with the best validation F1 for our final model.

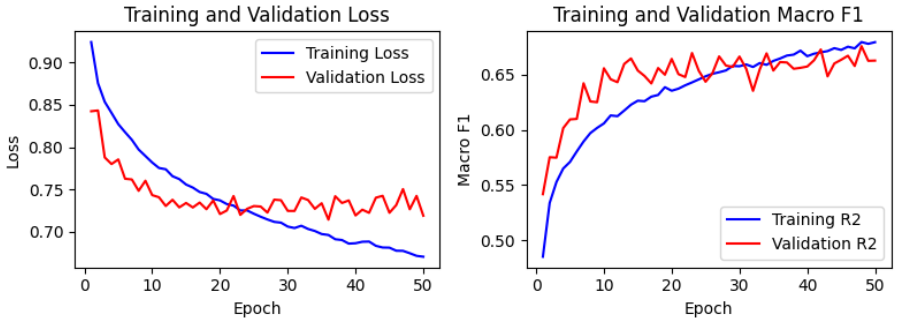

Figure 1: LSTM-based Sentiment Classifier

3.5 BERT Transformer

We use the pretrained "bert-base-uncased" Transformer (and its tokenizer on our preprocessed tweets). Here the average sentence length is 20, and we choose to use 40 as the max length. The customized dataset reader returns the attention mask along with input ids and label since the input ids are padded. For the classification task, only one fully connected layer is added to the pre-trained Transformer, so unlike the LSTM model where we kept the embedding layer frozen, the transformer layer is not frozen, and the total number of trainable parameters is 109,484,547. Adam optimizer and cross entropy loss are still used, but the learning rate must be smaller (0.00001) than the previous one in order for the model to train (for the loss to drop). We can see from the convergence plot that the performance on the training set kept improving steadily in the 50 epochs, on validation set the f1 score did not change much, and the loss even dropped. There may be danger of overfitting, but we choose the one with the best validation f1 to prevent it.

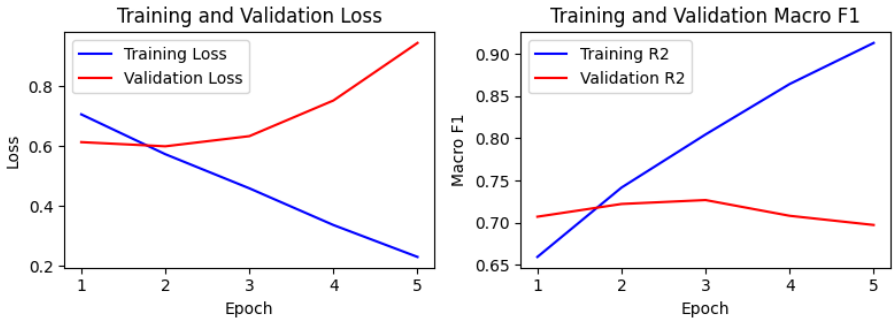

Figure 2: BERT-based Sentiment Classifier

4 Evaluation

We evaluate the four classifiers on validation set with three metrics, accuracy, average recall, and macroaveraged F1 score, as mentioned in the Semeval 2017 task paper. The performance is BERT Transformer > LSTM > Naïve Bayes > MaxEnt (the last two are quite close to each other).

Model + Feature	Accuracy	Average Recall	Macro F1
MaxEnt + bow	0.640	0.607	0.599
NaïveBayes + bow	0.607	0.611	0.602
LSTM + embed	0.691	0.667	0.660
BERT Transformer	**0.731**	**0.725**	**0.730**

Table 1: Multiple Metrics on Validation Set

We then evaluate the classifiers across all test sets using the macroaveraged F1 score only, and calculate the mean and standard deviation. It can be inferred from the results that test set 2 is probably the most similar to the training and validation set (so that it generalized well), while test set 3 is the most different. Here MaxEnt is slightly better than Naïve Bayes, the LSTM-based model clearly outperforms the former wo, while the BERT-Transformed-based is not only much more precise but also more robust (the highest mean and the lowest standard deviation).

Model + Feature	Test 1	Test 2	Test 3	Mean	StdDev
MaxEnt + bow	0.554	0.569	0.536	0.553	0.017
NaïveBayes + bow	0.546	0.548	0.533	0.542	0.008
LSTM + embed	0.595	0.620	0.562	0.592	0.029
BERT Transformer	**0.709**	**0.709**	**0.704**	**0.707**	**0.003**

Table 2: Macro F1 Across Test Sets