Subjectivity and Polarity Classification with Deep models

Final Project of the Natural Language Understanding Course

Matteo Guglielmi (232088)

University of Trento

matteo.guglielmi@studenti.unitn.it

Abstract

This work focuses on explaining the salient steps taken to implement a Sentiment Analysis algorithm that consists in *Subjectivity Detection* and *Polarity Classification*.
More in detail, a custom model is compared to a baseline in order to highlight the improvements achieved. To carry out this project, three different datasets were used: the "`MovieReviews`" and "`Subjectivity`" datasets and the public "`IMDB`" dataset dowloaded from Kaggle [1].

***Index Terms* —** BertModel, BertTokenizer, BertForSequenceClassification, polarity classification, subjectivity detection

1. Introduction

Sentiment analysis consists in performing sentence classification with the final goal to predict whether a sentence/review/comment expresses a positive or negative sentiment. It is a very important task in the field of Natural Language Processing (NLP) and has been studied for many years being a very challenging task due to the presence of negation, sarcasm, irony, and other linguistic phenomena that make the job difficult.
In this work I'll be comparing two different models to perform sentiment and polarity classification.

1.1. Baseline

The baseline model consists of:

- a `CountVectorizer` [2], which acts as an encoder, to extract the features from the text;
- a `Naïve Bayes` [3] classifier to perform the final classification.

It is worth mentioning that both datasets were pre-processed applying double negative marking to consider double negations (double negations corresponds to a neutral effect).
More in detail, the step-by-step procedure used to perform the classification is the following:

1. Pre-processing: the text is pre-processed by collapsing the several sentences in documents and the double negations are marked;
2. Feature extraction: text from the Subjectivity dataset is encoded using a `CountVectorizer`;
3. Classification: the encoded text from the previous point is classified using a `Naïve Bayes` classifier;
4. Evaluation: the classification is evaluated using a 10-fold cross-validation procedure exploiting the `accuracy` metric to address the overall performances;
5. Filtering: the trained model is used to filter the objective sentences from the Movie Review dataset;
6. Feature extraction: the text belonging to the Movie Review dataset, after being processed as point (1), is encoded using a `CountVectorizer` as point (2);
7. Classification: the encoded text is classified using a `Naïve Bayes` classifier as point (3);

1.2. Proposed model

The proposed model follows the same logic of the baseline but with a completely different approach. Very briefly, the proposed model consists in using :

- a `Bert Tokenizer` [4] to extract the features from the text and a `BertModel` [5] to derive the embeddings;
- a small `BiLSTM` network used to filter objective movie reviews from the Movie Review dataset, accepting the output of the previous point as input;
- a `BertForSequenceClassification` to perform the final classification on the MovieReviews dataset as a whole.

2. Task formalization

Sentiment analysis is deployed in different fields and, with the rise of deep learning, it is possible to deal with very complex scenarios, e.g. when the opinions are not explicitly expressed.
In the specific case of this assignment, the goal is to build a machine learning/deep learning model capable of performing subjectivity and polarity classification comparing the achieved results to a baseline model.
More precisely, the aforementioned model needs to perform:

- sentiment classification: the automated process of identifying opinions in text and labeling them as positive, negative, or neutral, based on the emotions expressed within them. In this specific case, the polarity dataset (**Section 3.1**) presents only the positive and negative tags reducing the problem to a binary classification task;
- subjectivity detection: consists in labeling a text as subjective or objective.

3. Data description and analysis

3.1. Movie reviews dataset

The movie reviews dataset used in this work is directly deployed by the `nltk.corpus` python package whose characteristics are the following :

- number of words : 1583820;
- lexicon size : 39768;

- categories : 'neg', 'pos'.

Each movie review has a file id associated with it used for its identification containing all the relative information. This dataset is used just for evaluation, since the amount of reviews is too small to perform a proper fine-tuning operation of the BertForSequenceClassification (see **Section 4.3**).

3.2. Subjectivity dataset

The subjectivity dataset is provided by the `nltk.corpus` python module as well.
The main characteristics of this dataset are :

- vocabuluary size : 240576;
- lexicon size (without considering repetitions of words) : 23906;
- classes : 'obj', 'subj';
- total number of sentences consiering both objective and subjective sentences : 10000.

In this specific case, there are just two unique file ids corresponding to the documents containing the objective and subjective sentences. The target datastructure used consists of the union of those two files.

3.3. IMDB dataset

The IMDB Dataset has $50'000$ movie reviews for natural language processing or text analytics [1]. This dataset can be used for binary sentiment classfication and it provides $25'000$ highly polar samples for training and $25'000$ samples for testing. More information reguarding this dataset can be found at [6].

4. Model

For the proposed task, I have decided to use a simple `Naïve Bayes classifier` as baseline, following the logical procedure explained in the 11^{th} laboratory [7].
Once the baseline performances were addressed, I tried to devise a completely different procedure by making use of more complex deep architectures.

4.1. Architectures

Baseline and custom model follow the same logical flow leading to both subjectivity and polarity classification, although the single components are complitely different. The logical salient steps taken in both cases are :

1. extract the encodings of each sentence leading to a vector representation;
2. train a ML/DL model upon the `subjectivity dataset` and save the weights;
3. use the pre-trained model from the previous point to filter out the objective sentences;
4. perform polarity classification upon the sentences predicted as subjective and address the performances through the `accuracy` metric.

4.2. Naïve Bayes classifier baseline

Concerning the baseline, the followed workflow is well expressed by `Algorithm 1`:

Algorithm 1: Steps performed in baseline model.

Input: subj_dt, mr_dt
Output: subj_acc, pol_acc

```
/* subj_dt: list of
   (sentences,label)
mr_dt: list of (sentences,label),
subj_acc: subjectivity accuracy,
pol_acc: polarity accuracy        */
subj_features ← FitVectorizerOn(subj_dt)
subj_scores ← 10-fold-cross-val(NB_clf, subj_features,
 subj_labels)
/* considering best estimator      */
best_clf ← argmax_clf (subj_scores)
subj_sents ← filter_objectivness(best_clf(pol_dt))
/* new vectorizer and clf          */
pol_features ← FitNewVectorizerOn(subj_sents)
pol_scores ← 10-fold-cross-val(New_NB_clf,
 pol_features, pol_labels)
```

4.3. Custom model

On the other side, my proposed implementation adopts only deep models to carry out this assignment. In particular:

- the `CountVectorizer` used as embedder in the baseline has been substituted with a small pipeline composed of:
  - a pre-trained BertTokenizer [4] based on WordPiece [8] used to break text into tokens, pad sentences to a maximum of 512 letters allowed by BERT, truncate to `max_length` longer phrases and convert text into float tensors;
  - a pre-trained BertModel [5] to extract text encondings to be used later on in the pipeline. These encodings are obtained by extracting the output of the transformer last hidden state as done in [9].
- the Naïve Bayes classifier has been changed with:
  - a simple `BiLSTM` network consisting of :
    1. a `BiLSTM` layer which accepts an output with a 768 long feature vector (output of the last hidden layer of BertModel) with an hidden dimension of 128;
    2. a `Dense` layer with an output size of 64 and a ReLU activation function to flatten the feature maps;
    3. a `Dropout` layer to reduce at the minimum possible the computational load with a probability of 0.35;
    4. and finally a `Linear` layer to produce a binary out.

    to perform subjectivity classification;
  - a `BertForSequenceClassification` [10] model fine-tuned for one epoch (to shorten training time) (results in **Figure 3**) upon the IMDB Dataset (**Section 3.3**) for performing polarity classification upon the `MovieReviews` dataset (**Section 3.1**).

Table 1: *Effects of filtering upon polarity dataset.*

Custom Model	Accuracy
Without filtering	0.9283
With filtering	0.8642

4.4. Optimizers

In this work, two different optimizers have been used:

1. `Adam` optimizer has been used to update the BiLSTM model parameters. It's been chosen among the other options due to its ability to converge quickly;
2. `AdamW` optimizer has been used to train the BertForSequenceClassification model. This particular variant of the classic Adam optimizer has been used because it's the default training option when using the `Trainer()` interface [11]

4.5. Experiments

Several experiments were run using the proposed model (**Section 4.3**). In particular, I initially tried to use the output of the pre-trained BertModel instance to predict the polarity scores after filtering out the objective sentences.
In this case the issue was that this model [5] outputs by definition 5 possible labels, ranging from $0 \rightarrow 5$. As a consequence, I tried mapping the two lowest scores $(0, 1)$ to the 'negative' label and the upper scores $(4, 5)$ to the 'positive' one, leaving score 3 as an indicator of uncertainty and misclassification. After analysing the predictions, the model turned out as heavily undecided, predicting mostly 3.
With these results as a reference, I tried to "flat" the 5 predicted labels to a binary score appending a linear layer with a Sigmoid activation function in combination with a `BCELoss()` [12].
After some research, I found out that a "C-class version" for the canonical BertModel already existed (`BertForSequenceClassification`).
Thanks to that, I was able to exploit a pre-trained very deep model to perform binary ($C = 2$) text classification combined with a binary cross-entropy loss used during the fine tuning procedure (it is worth mentioning that the Trainer class is responsible for computing some sort of loss and returning it as output if no `compute_loss()` method is specified, as discussed in [13]).
Further experiments were conduced attempting to reduce the amount of encoded tokens in the MovieReviews dataset by clearing the source text from elements such as stopwords, numbers, HTML special entities, hyperlinks, hashtags, punctuation (besides exclamation marks), words with less than 2 letters, usernames (@USERNAME) and tickers. It turned out that both the BiLSTM model and the BertForSequenceClassification achieved better results without filtering (see **Table 1**), proving these elements as relevant to the context.

5. Evaluation

This section will showcase results from different runs. In particular, it is worth mentioning that some default hyper-parameters were used, such as:

- *batch size* set to 128 samples both for training the BiSTM and for the final polarity evaluation procedure and to 64 samples for the binary classifier;
- concerning the *dataset split* dimensions:
  - the subjectivity dataset was split into training set (80%) and test set (20%). The training set has been further splitted with a randomly sampled 10% for evaluation pruposes;
  - the IMDB dataset has been splitted following the built-in format of the dataset itself ($25'000$ samples for training and $25'000$ samples for evaluation)
- during the training procedure an *early stopper* has been used to implement early stopping. To achieve it, the difference between the accuracy of the previous run and the current one is investigated. If the latter is grater than $MIN\_DELTA = 0.075$, the training procedure is interrupted and the last model saved;
- the BiLSTM model has been trained for 10 *epochs* and the binary classifier just for one (for time reasons);
- regarding the BiLSTM model, the multiple runs have been performed using randomly selected seeds: $[91, 11, 57, 822, 19]$;
- for the AdamW and Adam optimizers, the learning rates used are 0.001 and 0.0002 respectively.

5.1. Evaluation metric

Across the different models, several evaluation metrics have been used.

5.1.1. Baseline

For the baseline, a simple `accuracy` metric has been used across a 10-fold cross validation procedure.

5.1.2. Custom model

Regarding my proposed model, different evaluation metrics have been used for the different components:

- for the BiLSTM model a simple cumulative accuracy metric, which is the sum of all batches' agreements over the number of samples in the training set, has been used;
- for the `BertForSequenceClassification` instance an ensamble of metrics has been computed, exploiting the `accuracy_score` and `precision_recall_fscore_support` methods provided by the `sklearn` python module. In particular, the function used to compute these metrics can be analysed in **Algorithm 2**
- as for the BiLSTM model, in the final polarity classification step a cumulative accuracy has been used, where the agreements between the predictions given by the pre-fine-tuned binary classifier and the ground truth are investigated and quantified.

Algorithm 2: Metric computation function used in Trainer interface.

Input: pred
Output: accuracy, f1, precision, recall

```
precision, recall, f1, _ ←
  precision_recall_fscore_support(labels, preds,
  average='binary')
acc ← accuracy_score(labels, preds)
return{ 'accuracy': acc,
        'f1': f1,
        'precision': precision,
        'recall': recall }
```

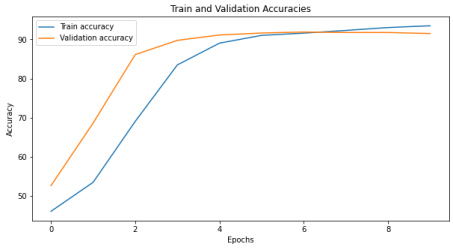

Figure 1: *BiLSTM model training and evaluation accuracies on a single run.*

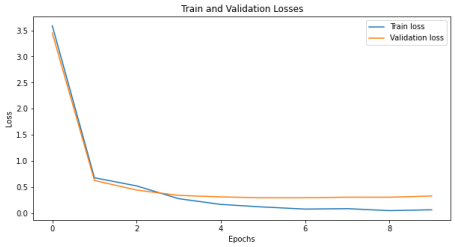

Figure 2: *BiLSTM model training and evaluation losses on a single run.*

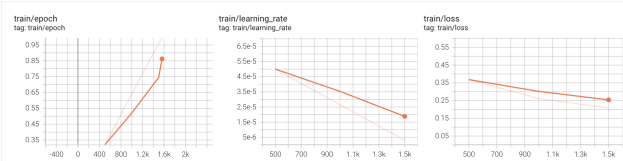

Figure 3: *Binary classifier statistics.*

Table 2: *5-fold cross validation subjectivity results using BiLSTM model*

Seeds	Accuracy
91	89.75
11	90.7
57	90.6
822	89.25
19	88.85
Avg. Acc.	89.83
Acc. Var.	0.73

Of course, the overall objective is to maximise the accuracy on both subjectivity and polarity datasets by minimizing the losses used during the training procedure and finding the best possible set of weights.

5.2. Results

In this particular section, the results obtained across different runs and experiments are analysed.

In **Figure 1** and **Figure 2** it is possible to appreciate the training and evaluation accuracies and losses respectively, on a single run for the BiLSTM model (discussed in **Section 4.3**).
Since a single run is not expressive and reliable enough, I decided to perform a 5-fold cross validation addressing both the average accuracy as well as the variance across the different runs (for details refer to **Table 2**).

The main reason behind using a BiLSTM model is to try to better catch the dependencies between the words in a sentence. Surprisingly, the results turned out to be worse than the ones obtained with the baseline both in terms of cumulative accuracy and stability (i.e. variance), as shown in **Table 3**. Despite this, the BiLSTM model is still used just to give an alternative to the baseline also considering that the difference in accuracy is minimal.

Table 3: *Comparison between the baseline and the BiLSTM model*

Model	Accuracy	Variance
Baseline	90.75%	0.13765%
Custom Model	89.83%	0.73%

Table 4: *Comparison between the baseline and the proposed model*

Model	Accuracy	Variance
Baseline	83.2%	0.00112
Custom Model	92.83%	x

Investigating the final results on the polarity dataset, we can notice that the proposed model reaches a better accuracy than the baseline, as shown in **Table 4**.
It is worth mentioning that the "x" on the `accuracy` column of the table means that the model has not been tested multiple times on the polarity dataset since the latter is used just for evaluation as discussed in **Section 3.1**.

5.3. Error Analysis

In this section, the errors of the proposed model will be analised. Although error analysis for polarity prediction is extremely complex, it is possible to identify possible limitations and critical points by analysing the pipeline building elements. In particular:

- the BiLSTM model performances may be influenced by the lack of an attention mechanism, which is a common technique used to improve the ability of a model when dealing with sequences and time series. In this particular case, the sentences within the subjectivity dataset are not very long (100% of the sentences are 66 words long or shorter), so the lack of an attention mechanism shoudn't represent a big issue even though it may bring a possible improvement to experiment;
- another possible source of error lies in the BERT model use since it truncates sentences to a maximum length of 512 words. This means that the model may not able to take into account the whole sentence when long dependencies are involved. As previously anticipated, this doesn't represent a problem when dealing with the subjectivity dataset since the sentences are not very long, although it may be when dealing with the polarity dataset since each document contains far more than 512 words. For this reason different experiments were conducted with the final goal of reducing the maximum length of the sentences to be fed to the BERT model (e.g. filtering the text as mentioned in Section 4.5), to no avail.
  One possible solution to this problem is to use a different embedder such as GloVe [14] or Word2Vec [15] which should't suffer the same problem.

To further understand the reasons why the model obtained certain performances, I tried to understand whether the misclassification of a sentence and its length are related. In particular, I tried to understand whether the model is more likely to make errors when dealing with long sentences. It turned out that the model commits errors when dealing with sequences longer than the 512 words limit, as proof of what has been previously

pointed out.

One elegant alternative to the proposed model would be to exploit a multi-task learning approach, i.e. to train the model on both the subjectivity and the polarity datasets and perform classification simultaneously. This approach would allow the model to learn the dependencies between the words in a sentence as well as the polarity and subjectivity of a document. However, the disparity between size of the datasets would make this method difficult to apply since the datasets would need to be of the same size. One possible solution would be shrinking the subjectivity dataset to the lenght of the polarity dataset. However, this approach would require a lot of computational resources and time, which are not available in this project. As a reference for this work, please refer to [9].

6. Conclusion

To summarise, in this work I've presented an alternative method for performing subjectiviy detection and polarity classification explaining the most salient steps taken to carry out this task as well as possible limitations and improvements.
As pointed out during the report, the baseline model used for this task is a simple shallow model, which is in most of the cases not the best choice for such complex problems. In response to this I also tested a deep learning model able to perform the task with a better accuracy (**Section 5.2**).
As final remarks, I would like to point out that the results obtained in this work are not intended as optimal, but they are a good starting point for further improvements. In particular, for future improvements I would focus on dealing with the limitation of using the Bert model as encoder trying to use other variants that are not limited to a maximum sequence length.

7. References

[1] "IMDB Dataset of 50K Movie Reviews," https://www.kaggle.com/datasets/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews.

[2] "sklearn.feature_extraction.text.CountVectorizer," https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html.

[3] "sklearn.naive_bayes.MultinomialNB," https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html#sklearn.naive_bayes.MultinomialNB.

[4] "BertTokenizer," https://huggingface.co/docs/transformers/model_doc/bert#transformers.BertTokenizer.

[5] "BertModel," https://huggingface.co/docs/transformers/v4.21.1/en/model_doc/bert#transformers.BertModel.

[6] "Large Movie Review Dataset," https://ai.stanford.edu/~amaas/data/sentiment/.

[7] E. A. Stepanov. (2021, Oct) Sentiment analysis/opinion mining. https://github.com/esrel/UNITN.NLU.Lab/blob/master/notebooks/11_sentiment_analysis.ipynb.

[8] W. Yonghui, M. Schuster, Z. Chen, Q. V. Le, W. Norouzi, M. Macherey, M. Krikun, Y. Cao, Q. Gao, Q. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, L. Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean, "Google's neural machine translation system: Bridging the gap between human and machine translation," https://arxiv.org/abs/1609.08144v2, Oct 2016, [Online] Submitted on 26 Sep 2016 (v1), last revised 8 Oct 2016 (v2).

[9] R. Satapathy, S. Pardeshi, and E. Cambria, "Polarity and subjectivity detection with multitask learning and bert embedding," https://doi.org/10.3390/fi14070191, Jun 2022, [Online] Submitted: on 6 May 2022/Revised: 8 June 2022/Accepted: 16 June 2022/Published: 22 June 2022.

[10] "BertForSequenceClassification," https://huggingface.co/docs/transformers/v4.21.1/en/model_doc/bert#transformers.BertForSequenceClassification.

[11] huggingface.co, "Trainer," https://huggingface.co/transformers/v3.0.2/main_classes/trainer.html.

[12] PyTorch, "Bceloss," https://pytorch.org/docs/stable/generated/torch.nn.BCELoss.html.

[13] Stack Overflow, "What is the loss function used in trainer from the transformers library of hugging face?" https://stackoverflow.com/questions/71581197/what-is-the-loss-function-used-in-trainer-from-the-transformers-library-of-huggi.

[14] Jeffrey Pennington, Richard Socher, Christopher D. Manning, "Glove: Global vectors for word representation," https://nlp.stanford.edu/projects/glove/.

[15] Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean, "Efficient estimation of word representations in vector space," https://arxiv.org/abs/1301.3781, Sep 2013.