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* 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 both downloaded from the NLTK python module and the public "`IMDB`" dataset dowloaded from Kaggle [1].

Index Terms — BertModel, BertTokenizer, BertForSequenceClassification, polarity classification, subjectivity detection, transformer, pre-trained

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.
Sentiment analysis is a very important task in the field of Natural Language Processing (NLP) and has been studied for many years. It is a very challenging task because of the presence of negation, sarcasm, irony, and other linguistic phenomena that make the task very difficult.
From a commercial point of view, sentiment analysis is a very important task because it can be used to automatically analyze the opinion of the customers about a product or a service. For the latter reason, many industries are interested in this specific applicatio to understand the degree of appreciation a specific product has.

As briefly mentioned before, in this work I'll be comparing two different models to perform Sentiment and polarity classification.

1.1. Baseline

The baseline model consists in :

- 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 followed to perform the classification is the following:

1. Pre-processing: the text is pre-processed by collapsing the severale sentences in documents and the double negations are marked
2. Feature extraction: the text belonging to the Subjectivity dataset is encoded using a `CountVectorizer` which counts the number of occurrences of each word in the text;
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 precedure exloiting 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 is encoded using a `CountVectorizer` as point (2);
7. Classification: the encoded text from the previous point is classified using a `Naïve Bayes` classifier as point (3);

1.2. Proposed model

The proposed model follows the same logical workflow as 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 shallow `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 the use of natural language processing and text analysis to identify, extract and quantify affective states and subjective information.
As previously mentioned, sentiment analysis is deployed in different fields to give voice to the customers and for the companies to receive a feedbacks through the analysis of forums or reviews.
With the raise of deep learning, it is possible to deal with very difficult 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 Subjectivity dataset presentes only the positive and negative tag reducing the problem to a binary classification task;
- polarity classification which consists in labeling a text as subjective or objective.

3. Data description and analysis

For this project, I've used three different datasets to perform different steps.

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 which is the identification factor of movie review.
The dataset presents itself in a peculiar structure. As just mentioned, each review has a specific file asociated to it so in order to ease the following work, everything has been flattened into a single list containing different documents. Each of the list element has an associated label.
From an operational perspective, this dataset is used just as an "evaluation set". The motivation behind this is that the total number of reviews (elements in the list) are just 2000. Since the procedure deployed implies to perform a fine-tuning operation of a Bert transformer model, dividing the already small dataset into train and test sets would result in an insufficient amount of data.
For this reason, the whole dataset has been used just to address the final polarity classifiaction accuracy.

3.2. Subjectivity dataset

As the previously discussed dataset, also in this case the Subjectivity dataset comes from the nltk.corpus python module.
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 documentis containing the objective and subjective sentences. The taraget datastructure used consists in the union of those two files.

3.3. IMDB dataset

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

4. Model

In order to address the proposed sentiment analysis task, I have decided to use as a baseline model a simple `Naïve Bayes classifier` following the logical procedure explained in the 11^{th} laboratory [7].
Once the baseline performances were addresses, I tried to devise a completely different pipline despite using the same coneptual workflow by making use of more complex and deep architectures (i.e. BiLSTMs and different transformer models).

4.1. Architectures

As hinted before, both models follow the same logical flow leading to both subjectivity and polarity classification but, on the other side, the single components are complitely different. In particular, the baseline approach (**4.2**) consists in a pure shallow machine learning procedure; on the other hand, for the custom model, I decided to opt for a deep architecuture-based techinque.
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` metrix.

4.2. Naïve Bayes classifier baseline

As briefly mentioned in Section 4, the baseline models uses the same ingredients presented in the 11^{th} laboratory of the Natural Language Course. In particular, the followed workflow is well expressed by `Algorithm 1`:

4.3. Custom model

On the other side, my proposed implementation adopts only deep models to carry out this assignment. Throughout the following lines, I'll be entering more in details about implementation choices and roles of the different architectures; 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 broke text into tokens, pad shorter sentences to the maximum length admissible by BERT (512), truncate to `max_length` longer phrases and convert text into float tensors;

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 the 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)
```

  - 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 with reference to [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 a ReLU activation function to flatten the feature maps;
    3. a `Dropout` layer to reduce at the minimum possible the computational load;
    4. and finally a `Linear` layer to produce a binary out.

    to perform subjectivity classification;
  - an instance of a `BertForSequenceClassification` model [10] fine-tuned for one epoch (to shorten training times) upon the IMDB Dataset (Section 3.3) for performing polarity classification upon the `MovieReviews` dataset (Section 3.1).

4.4. Optimizers

In this work, two different optimiers have been used:

1. `Adam` optimizer has been used to update the BiLSTM model parameters. It's been chosen among the other options because of 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 also because set as defualt option when using the `Trainer()` interface [11]

4.5. Experiments

Several experiments were run using the Custom model (Section 4.3). In particular, at the beginning I tried to use the output of the pre-trained BertModel instance to predict the polarity scores after having filtered out the objective sentences.
The problem with this approach was that [5] outputs by definition 5 possible labels, ranging from $0 \rightarrow 5$. As a consequence, I tried to map the two lower scores $(0, 1)$ to the 'negative' label and the upper scores $(4, 5)$ to the 'positive' label leaving the score 3 as an indicator of uncertainty and misclassification. After a brief analysis of the predictions, the model turned out to be heavily undecided, predicting for the majority of the times the neutral label (3).
In light of this, 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 finding, I was able to exploit a pre-trained very deep model to perform binary ($C = 2$) text classification in combination 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 in output if no `compute_loss()` method is specified as discussed in [13]).
Further experiments were conduced about reducing the amount of encoded tokens by deleting from the original text 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 BERT models achieved better results without filtering. They appear to be able to better catch contexts inside the different sentences.

5. Evaluation

The purpose of this section is to explain and show the obtained results across different runs. In particular, it is worth mentioning that lots of default valued hyper-parameters have been used such as:

- *batch size*, for both the training of the BiSTM (Section 3.2) and the final polarity evaluation procedure, of 128 samples. In the case of the binary classifier ([10]), a batch size of 64 has been used;
- concerning the *dataset split* dimensions:
  - the subjectivity dataset into training set (80%) and test set (20%). The training set has been further splitted considering a randomly sampled 10% for evaluation pruposes;
  - the IMDB dataset has been splitted following the built-in splitting format of the dataset itself ($25K$ samples for training and $25K$ sampels for evaluation)
- during the training procedure an *early stopper* has been used to implement an early stopping techinque. To do so, the difference between the accuracy at the previous run and the current one is investiagate. If the latter is grater than $MIN\_DELTA = 0.075$, then 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 a randomly selected seeds: $[91, 11, 57, 822, 19]$;
- for the AdamW and Adam optimiers, 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

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

5.1.2. Custom model

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

- for the BiLSTM model, as well as for the final evaluation on the polarity dataset, to detect subjectivity labels, a simple cumulative accuracy metric, defined as the sum of all the agreeements across all the batches 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, also for the final polarity classification step a cumulative accuracy has been used, where the agreeements between the predictions given by the pre-fine-tuned binary classifier and the ground truth are investigated and quantified.

Algorithm 2: Metric computation function for 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
}
```

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 as application of the maximisation of the posterior probabilities.

5.2. Results

In this particular section, the results obtained across different runs and experiments are analysised as much in detail as possible.

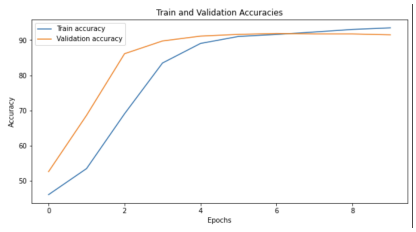

Figure 1: *BiLSTM model training and evaluation accuracies on a single run.*

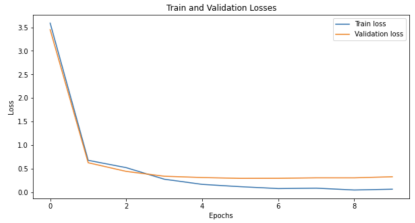

Figure 2: *BiLSTM model training and evaluation losses on a single run.*

In **Figure 1** and **Figure 2** it is possible to respectively appreciate the training and evaluation accuracies and losses, on a single run for the BiLSTM model (discussed in Section 4.3). Since a single run is not very expressive and fully reliable, 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 1).

Table 1: *5-fold cross validation subjectivity results using BiLSTM model*

Seeds	Accuracy
91	89.5
11	90.8
57	91.05
822	90.25
19	89.1
Avg. Acc.	90.14
Acc. Var.	0.74

The main reason behind using a BiLSTM model is to try to better catch the dependencies between the words in a sentence. The results turned out to be worse than the ones obtained with the baseline from the cumulative accuracy perspective but better in terms of consistency and stableness. Indeed, using the same evaluation metric, the average statistics, listed in Table 2, shows that the baseline reaches a better percentual average accuracy but a higher variance across the different runs, i.e. seems to be more unstable and inconsistent.

Table 2: *Comparison between the baseline and the BiLSTM model*

Model	Accuracy	Variance
Baseline	90.75%	0.00013765
Custom Model	90.14%	$5.534000000000004e-05$

Investigating the final results on the polarity dataset, it is possible to appreciate that the proposed model reaches a better accuracy than the baseline, as shown in Table 3.
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 purposes as discussed in Section 3.1.

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

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

6. Conclusion

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.