Warsaw University of Technology

Faculty of Electronics and Information Technology

Dimitrios Georgousis

Andrea Amato

Group 12

Introduction to Artificial Intelligence

Project 7: Sentiment analysis

Midterm report

Description

Sentiment analysis (or opinion mining) involves identifying and categorizing opinions expressed in a text to determine whether the attitude is positive, negative, or neutral. The task will involve classifying customer reviews from the Amazon Reviews dataset available at Kaggle, so we consider it to be a classification task and, more specifically, since our dataset does not include neutral reviews, it reduces to a binary classification task. This project aims to develop, compare and improve multiple machine learning models for sentiment analysis and determine the best-performing approach.

Dataset Description

The dataset is of the form: `__label__<X> <Title>: <Text>` and this pattern repeats. Labels can only be `__label__1` (negative) and `__label__2` (positive). Most of the reviews are in English, but there are a few in other languages, like Spanish.

Dataset for Midterm submission

We test various algorithms and preprocessing techniques and to do that in reasonable time, we use a small slice of the complete dataset (100.000 reviews). We use 80.000 reviews for training and 20.000 for testing.

Label Balancing

Our first approach did not attempt any forced label balancing on the training set. Even then the positive and negative reviews are not that much removed from each other.

Result of just taking the first 100.000 reviews:

```
label
2    51267
1    48733
Name: count, dtype: int64
```

In our later approach we forced the dataset to be completely balanced by placing a limit of 50.000 reviews on each type.

Data Preprocessing

Our first approach was a `tfidf` preprocessing methodoly, where:

- The title and review texts are processing together, as one piece of text
- Html tags were removed
- Special characters and digits where removed
- The text was converted to lower case
- There were no stopwords
- And the words were lemmatized
- Only English reviews were kept

This type of preprocessing was fed to the Multinomial Naive Bayes classifier.

The second approach uses the BERT tokenizer embeddings, but also handles the dataset a bit more differently.

- The title and review text are still processed together
- Emojis are more useful to our algorithm now by being replaced with descriptive text
- Words do not need to be lemmatized
- Only English reviews were kept

Texts are tokenized to a fixed length, to achieve truncation or padding is performed. The BERT tokenizer takes into account a word's context when generating its vector of values, thus managing to better capture the general sentiment of one's review. It is a more robust system that handles misspellings, complex word forms and rare words.

These embeddings are used in training Logistic Regression, SVC and Random Forest classifiers.

Performance Evaluation

Metrics

The task is binary classification and appropriate evaluation methods will be used. Metrics such as:

- Accuracy: describing the number of correct predictions over all predictions
- Precision: measure of how many of the positive predictions are true positives
- Recall: measure of how many of the positive cases the classifier predicted correctly
- F1-Score: harmonic mean of precision and recall

The confusion matrix supplies useful information to us too.

Comments on Results

tf-idf and Naive Bayes (`data_preprocessing.ipynb` file)

This is the simplest approach we attempted and, also, the fastest in running time. It does not achieve good results with an f1 score of 43% for negative reviews and 60% for positive reviews and a general accuracy of 53% on our testing set.

We present the classification report and the confusion matrix for this classifier:

```
Naive Bayes:
              precision    recall  f1-score   support

         1.0       0.52      0.36      0.43      9769
         2.0       0.53      0.69      0.60     10156

    accuracy                           0.53     19925
   macro avg       0.52      0.52      0.51     19925
weighted avg       0.52      0.53      0.51     19925

[[3508 6261]
 [3199 6957]]
```

We may note the following reasonings for this behaviour:

- Class Imbalance: as mentioned the classes were not balanced in this approach, but still the difference between the label counts for the two labels was not severe enough to warrant such an underwhelming result.
- Feature Representation: TF-IDF does not really consider the order of words, their semantics or the context of their usage. Thus TF-IDF might be a reason behind this rather poor performance.
- Naive Assumption: TF-IDF extracts frequencies of words, which are not really that independent from each other, since the usage of words is heavily related to context. Thus the naive assumption is probably not helping our classifier in this specific text classification task.
- Text preprocessing: our text cleaning algorithm in this approach was much simpler. It did not handle emojis, which carry a lot of sentiment. It also doesn't remove URLs or hashtags.

It is evident that TF-IDF does not manage to extract a lot of meaning from our review texts, especially when we compare its runtime to that of the BERT tokenizer and following embeddings, which take considerable time to be processed.

Classification with BERT embeddings (`main.ipynb` and `main.py` files)

After producing the embeddings, we trained a Logistic Regression model, an SVC model and a Random Forest. We present the classification report and the confusion matrix for these classifier:

```
Logistic Regression:
              precision    recall  f1-score   support

           1       0.82      0.84      0.83      9879
           2       0.84      0.82      0.83     10084

    accuracy                           0.83     19963
   macro avg       0.83      0.83      0.83     19963
weighted avg       0.83      0.83      0.83     19963

[[8300 1579]
 [1799 8285]]
SVM:
              precision    recall  f1-score   support

           1       0.83      0.85      0.84      9879
           2       0.85      0.82      0.84     10084

    accuracy                           0.84     19963
   macro avg       0.84      0.84      0.84     19963
weighted avg       0.84      0.84      0.84     19963

[[8398 1481]
 [1776 8308]]
Random Forest:
              precision    recall  f1-score   support

           1       0.76      0.81      0.78      9879
           2       0.80      0.76      0.78     10084

    accuracy                           0.78     19963
   macro avg       0.78      0.78      0.78     19963
weighted avg       0.78      0.78      0.78     19963

[[7970 1909]
 [2467 7617]]
```

We got rather promising results, considering these classifier are rather simple and they are boosted by the information we managed to encode into the word embeddings we performed previously. The f1 scores and accuracies are rather high for both labels with the Random Forest classifier underperforming when compared to the other two. This behaviour was rather expected, because as discussed in the previous report this classifier is considered the least suitable for language processing.

Comments on BERT embeddings

The runtime of their calculation/production is quite long and the computation was moved to the GPU on Google Colab in order to speed up the process. In general, they require high processing power and a lot of time. It was required to process the tokens in batches because of memory restrictions and we opted to save the results on local files, because recalculation was prohibitive to performance, especially, when doing tests and trying different coding methodologies.

Comments on Python Notebooks

Python notebooks helped us a lot in optimizing our code and fixing errors without much runtime overhead, that is why they appear in this stage of our project. The final submission will, of course, be a python file, but the intermediary steps are not harmed, but rather aided by their usage. Nonetheless, to meet the requirement for strictly python files, we have a `main.py` file with our most promising ideas.

Discussion

Challenges

- Runtime is our main challenge, when implementing more sophisticated classification algorithms (such as LSTM or BERT which are included in our planned classifiers) the runtime will be even larger. We already experimented a bit with the LSTM algorithm but decided to not go further with those results for now. This situation is rather problematic for our project, since we used a very small portion of the original dataset (100.000 reviews, from which only 80.000 were actually used in training) and training on the whole dataset will need significant resources. Lastly, the large runtime complicates attempts of hyperparameter tuning, which is a usually important part of model training. In general, the dataset is quite large, the embedding generation is very demanding and the training of more complex models is demanding on its own too.
- Data cleaning: language detection didn't always perform properly (e.g. when a lot of uppercase words were used, detection was inclined to classify it as German and other such observations we made). Also, upon inspecting the dataset, misspellings were quite common in the review texts. Because of the nature of this classification problem and the discussed issue of runtime, it is hard for us to test different data cleaning/text preprocessing methods and pick a more consistent methodoly.

Continuation – Plans

- Implementing the rest of our algorithms
- Using a more robust hyperparameter tuning scheme on training of our classifiers
- Training with a slice of our dataset

- Attempting data scaling on the embeddings
- Try different tokenizers (maybe one suitable for LSTM training)
- Perhaps train Naive Bayes with Count Vectorizers instead of TF-IDF

Results shown on the final report

We will show and comment on the classification report, which bundles most performance metrics together, and the confusion matrix, as we already do for our classifiers.