Plaint-or-Tweet

Tommaso Battistini, Edoardo De Matteis, Leonardo Emili,
Mirko Giacchini, Alessio Luciani

December 25, 2020

Introduction

For our final project we decided to implement some sentiment analysis techniques on social media data, and in particular on tweets from the social network Twitter. We used a dataset from Kaggle [4] which contains 1.6 million tweets labelled as positive or negative (the dataset is balanced: there are 800k positive tweets and 800k negative tweets). We chose this topic because it is of course interesting from a research point of view, but it has also many applications in industry: for example we discovered some companies that provide sentiment analisys on social media as one of their main services [1] , these services are especially useful to other companies, for example to understand what their customers think about the release of new products (and in many similar scenarios).

Models

As we have seen during coursework, the naive bayes classifier has proved to be a good model for spam classification, and it is therefore reasonable to expect good results also in sentiment analysis. We decided to implement many variations of naive bayes to have a deeper understanding on how the representations of a tweet (and in general of a text) affects the performances of this class of models.

Classic Naive Bayes

We implemented the multivariate bernoulli event model, in which each tweet T is associated to a boolean vector V and $V[i] = 1$ if and only if the i-th word of the dictionary is present in tweet T.
Another classical model we implemented is the multinomial event model: for each tweet we create a vector where the i-th value is the number of times the i-th word of the dictionary appears inside the tweet; note that this representation is a bit different from the one seen during coursework but we get equivalent formulas for the parameters and this representation makes the implementation of the model easier (in the representation from coursework the i-th value of the vector is the index, in the dictionary, of the i-th word in the tweet). In the multinomial event model we also tried to associate to each word its tf-idf score instead of the number of occurrences: the tf-idf value is a score associated to a word inside a tweet, that increases the more the word is repeated inside the tweet and decreases the more the word is repeated in the whole training set. This is an improper way of using the multinomial event model because we are not using vectors of positive integers, but we are using positive real values; however this variation has been used a lot in practice and it seems to perform well.

Embeddings

A relatively new idea that has been really successful in natural language processing, is the one of word embeddings: we associate to each word a vector of real values, learned via deep learning techniques. The two most common algorithms to create word embeddings are FastText and Word2Vec, we tried both of them to get the embeddings of words, and to get the embedding of a tweet we average the embeddings of the words it contains. To make predictions from the tweet embeddings we used three versions of naive bayes. We tried the multinomial event model on the raw values of the embedding, the only difficulty here is that the embedding might contain negative values and this could result in negative probabilities in the multinomial event model: to fix this, we translate all the values to make them positive (we do this looking at the minimum value in the training set, for each feature), the intuition behind this model is to consider the values of the embeddings as score (as it is done with tf-idf), but of course this might not be true in general, since the main property of word embeddings is that similar words will be close to each other. We also tried the multinomial event

model discretizing each features in k buckets (and in this case, the features will not "mix up", ie: each feature will be treated indipendently), this is something often done in the multinomial event model when dealing with continuous values, so we expected some decent results from this model. The last model we used is a gaussian naive bayes: assuming that the feature are gaussian is another common way to deal with continuous values in naive bayes, however in this case the values are artificial, and we didn't expect them to be gaussian, so from this model we expected bad results.
Word embeddings are generally used as input for complex models such as neural networks, we thought it was interesting to see if also simpler models such as naive bayes can take profit from these successful representations.
Note that there are many other ways to create the embedding of a tweet from the word embeddings, for example we could do a weighted average (weighting the words according to some score, for example tf-idf), or we could concatenate the word embeddings (truncating too long tweets, and padding too short tweets).

Notes on implementation

The core of the models has been implemented from scratch, but for the data preparation we used some scikit-learn function: for example we implemented the multinomial event model assuming in input vectors of positive values, and we used a scikit-learn function to transform a tweet to a vector (of frequencies or of tf-idf scores). Since the dataset is really large, the efficiency of the learning and testing procedures has been a critical factor to take into account. We managed to get reasonably efficient algorithms leveraging numpy and scipy functions: speaking at a real high level, the key idea has been to compute most of the needed values across all the training set (or all the test set) instead of iterating over the train sample (or test sample) and compute such values in a trivial way.

Preprocessing

Tests and metrics

...

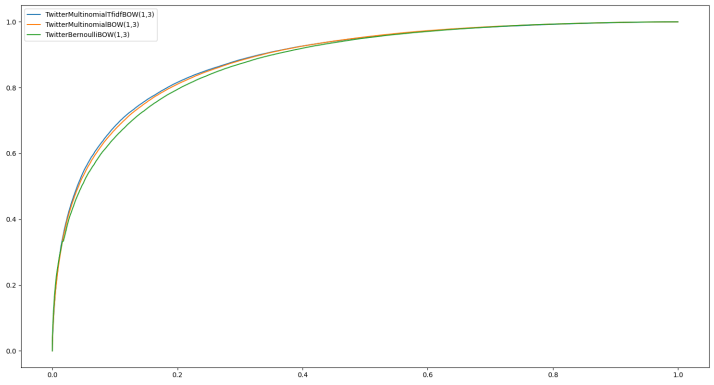

Figure 1: ROC curve for the Naïve Bayes approach with tf-idf.

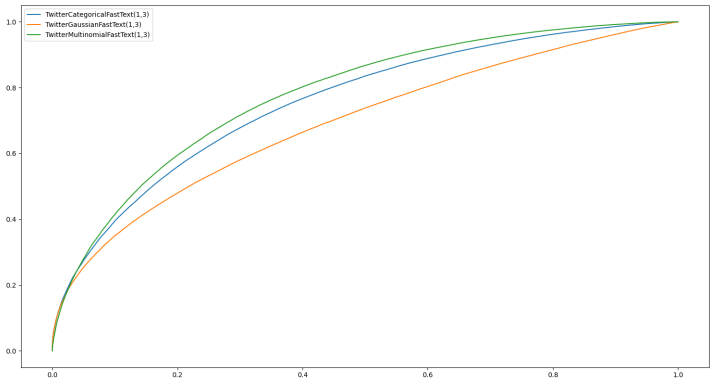

Figure 2: ROC curve for the FastText embedding.

Kaggle notebooks

Further tests

We trained our model with a dataset and tested it against a different one, to this aim were used am IMDb dataset of cinematographic reviews [3] and a Reddit

one about various NFL games [2], results for the Bag-Of-Words Naïve Bayes (1,3)-gram model can be seen in figures 3 and 4.

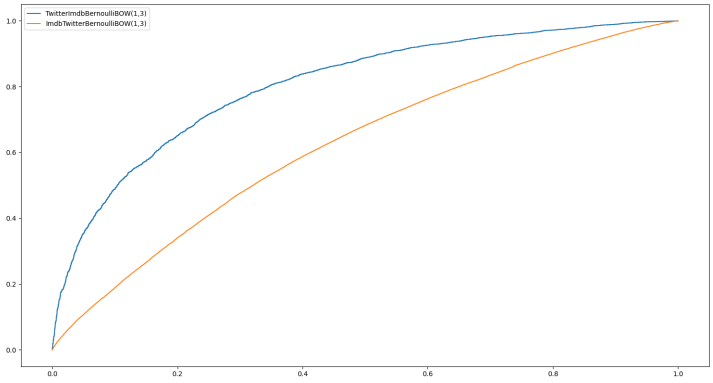

Figure 3: Comparison between Imdb-Twitter and Twitter-Imdb.

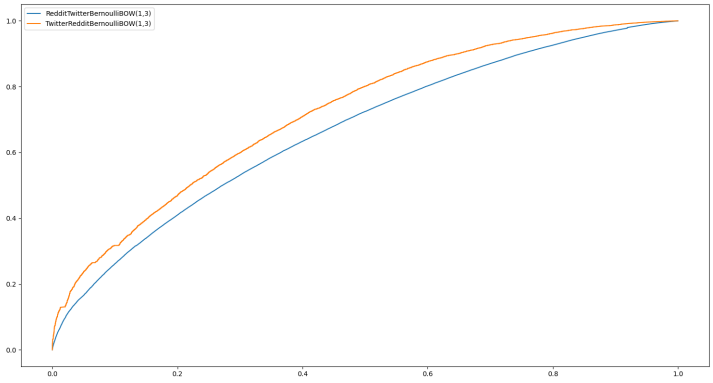

Figure 4: Comparison betwen Reddit-Twitter and Twitter-Reddit.

This is something usually not done since different datasets will have different distributions hence we expect bad results, strangely enough training our model with the Twitter dataset and testing it with the IMDb one gives rather good metrics that can be seen in table 1, is exposed only the comparision betwen Twitter and IMDb since results are more interesting. This could happen since Twitter's dataset has a very large number of examples, other datasets other than being smaller are also very specific: the Reddit one is only about a specific

set of american football games and the IMDb one is about cinematographic reviews, reasonably in this case lexicon is also more specific. Sentiment140 on the other hand covers a larger class of human language and is less prone to be biased.

Table 1: Comparing metrics for some train test combinations.

Train-Test	Accuracy	F1	AUROC
Twitter-IMDb	0.732	0.723	0.806
IMDb-IMDb	0.891	0.887	0.963
IMDb-Twitter	0.550	0.330	0.625
Twitter-Twitter	0.797	0.800	0.880

References

[1] Spiketrap company. URL: https://www.spiketrap.io/.

[2] Caio Brighenti. Nfl draft reddit comments. URL: https://www.kaggle.com/caiobrighenti/nfl-draft-reddit-comments?select=picks.csv.

[3] IMDb. Imdb datasets. URL: https://www.imdb.com/interfaces/.

[4] Marios Mikaelides Kazanova. Sentiment140, 2017. URL: https://www.kaggle.com/kazanova/sentiment140.