Università degli studi di
Milano-Bicocca

Text Mining & Search

Project Report

Training a Sentiment Classifier on the Amazon Reviews Dataset

Authors:
Khaled Hechmi – 793085 – k.hechmi@campus.unimib.it
Gian Carlo Milanese – 848629 – g.milanese1@campus.unimib.it

January 10, 2020

Contents

Introduction

The goal of this project is to use the *Amazon Reviews Dataset* to train a classifier that performs sentiment analysis. The task is treated as a binary classification problem, with *positive* and *negative* (or *critical*) being the two classes to predict. Several algorithms for classification are compared.

Each classifier is trained on a subset of the Amazon Reviews Dataset and evaluated on different sets:

- A portion (33%) of the starting dataset, which was not used in the training phase;
- Three set of tweets and replies to tweets from different sources.

Large and varied datasets of labelled tweets are difficult to find or laborious to create, especially if one is interested on tweets concerning a specific topic. On the other hand, the Amazon Reviews Dataset provides, thanks to the e-commerce platform popularity, a large collection of opinions on several different categories of products, each labelled with a score that is helpful for assigning a basic sentiment. This project aims to understand if it is possible to train a classifier on a dataset of reviews that is

also efficient and effective in performing sentiment analysis on tweets. Solid performances on this dataset would mean that, for instance, a company could be able to quickly understand how its new products are perceived by a subset of the population, even though it will be important to filter genuine users appraisals and concerns from internet trolls or bots messages. While only Twitter is considered for this project, a good classifier can be used in different domains where, like Twitter, it is too expensive to build a valid training set: some examples are finding the sentiments of text coming from other social media, such as YouTube comments or discussions in specialized online communities.

This report is structured as follows:

Data sources Brief description of the Amazon Reviews Dataset and of the Twitter data.

Text pre-processing Pre-processing and normalization of the Amazon reviews and the Twitter data.

Text representation Representation of the text to be passed to the classifier and motivation.

Classification Algorithms used for classification.

Evaluation Evaluation of the classification algorithms on the Amazon Review Dataset (training set and held-out test set) and on the Twitter datasets.

1 Data sources

The Amazon Reviews Dataset [1] is a large collection of user reviews for products sold on Amazon. The dataset features 233.1 million reviews for products of different categories, such as *Fashion*, *Books* and *Digital Music*.

For this project the subset of the dataset regarding the categories

- *Automotive*,
- *Cell Phones and Accessories*,
- *Grocery and Gourmet Food*, and
- *Video Games*

was chosen as the training data for the classifier that performs sentiment analysis.

The features of interest in this dataset are *reviewText* and *overall*, the score associated with the review, which is an integer value ranging from 1 to 5. After removing missing values from *reviewText* and deleting non-verified reviews the dataset counts 3942028 observations.

For the purpose of this project a review with score strictly greater than 3 is considered having positive sentiment, and negative sentiment otherwise. This choice can be justified by noting that the *All positive/All critical* filter on the Amazon website follows this same convention. Another reason behind this decision is the fact that the classifier will be tested on tweets, where a *positive/negative* sentiment label is more appropriate than a score in the 1 to 5 range.

The Twitter data consists of:

- *CESTweets*: 2000 tweets containing the hashtag *CES2020* regarding the CES 2020 technology conference, downloaded with the *tweepy* Python library.
- *SanTweets*: the twitter sentiment corpus created by Sanders Analytics [8], consisting of 5113 hand-classified tweets, each regarding a topic among *Apple*, *Google*, *Microsoft* and *Twitter*.
- *SMTweets*: 289 replies to a tweet by *Samsung Mobile* presenting the Samsung Galaxy Note10 and 10+ [9], collected with the *tweepy* Python library and manually labelled as either *positive*, *negative* or *neutral*;

2 Text pre-processing

2.1 Pre-processing Amazon reviews

The reviews needed to be pre-processed in order to achieve a better representation and be intelligible by machines.

The first group of operations involved *cleaning* the reviews and some initial *normalization*. This phase is essential in order to reconcile some differences in the text representation. Different operations were carried out to this end:

- Lowercase every word, as otherwise a computer would treat two identical words as different even if only a pair of their letters has different case.
- Delete URLs and HTML text related to the presence of a video in the review.
- Replace emoticons with positive or negative sentiments: reviews can be written without any constraints therefore it is highly improbable that users will use

them. Emoticons that express a positive sentiment are replaced with the word *good*, while the negative ones are replaced with the word *bad*. Before applying this translation, for each emoticon that usually conveys a positive sentiment it was checked whether the proportion

$$p_1 = \frac{\sharp(\text{positive reviews containing the emoticon})}{\sharp(\text{reviews containing the emoticon})}$$

was significantly larger than the proportion

$$p_2 = \frac{\sharp(\text{positive reviews not containing the emoticon})}{\sharp(\text{reviews not containing the emoticon})}$$

and similarly for emoticons associated with negative sentiments. The rationale for this test is that if p_1 is in fact the same as p_2, then the presence of the emoticon in a review is not informative about its sentiment. Emoticons not satisfying this test were discarded.

- Slang and abbreviations: for reasons similar to the emoticon case, slang and abbreviations are present in the reviews. A subset of them has been replaced with their common equivalent (e.g., "gr8" $\rightarrow$ "great").
- Contracted forms: words having a contracted form (e.g. "wouldn't") are expanded in their "standard" form (e.g. "would not"). This is done because the keyword *not* has a useful meaning and representations such as bigrams can benefit from its presence.
- Special characters and punctuation: these characters are removed, as they are not useful for finding the sentiment of a given review. This operation also helps reduce the dataset dimensionality, leading to an increase in efficiency. Characters that might affect tokenization are not deleted at this step.

The second operation that was performed is the so-called *tokenization*. This step is necessary for splitting each review, which is stored as a string, in multiple tokens made of single words and punctuation characters. This operation was implemented using the *word_tokenize* function inside the *nltk* library [5], which employs the pre-trained Punkt Sentence Tokenizer [6].

After tokenization, *spelling correction* was performed as an additional normalization operation: when writing reviews or tweets users may spell some words incorrectly for a variety of reasons, such as keyboards layouts, and important words might lose their meaning as a consequence. The spelling correction is achieved using the

pyspellchecker library, which implements the spell checking algorithm described by Peter Norvig [7]. For performance reasons only permutations having a Levenshtein distance of 1 were evaluated by the algorithm.

The next step involved the *removal of stop words*. The benefit of this task is to largely reduce the dimensionality of the dataset, while keeping intact its informativeness: words such as articles and pronouns are really frequent in sentences and their removal greatly contributes to reducing the dataset dimensionality without altering their meaning or sentiment. From the list of stop words made available in the *nltk* libraries, words that represents a negation of the following term are not replaced for the same reasons that led to the expansion of contracted forms.

The last pre-processing step that was performed is *stemming*, in order to remove suffix from words and bring them to their root form: in this way singular and plural forms of the same word will be replaced with the same term. The stemming algorithms implementations that were employed are the Porter and Lancaster stemmer from the *nltk* Python library, and the Porter stemmer from *pystemmer*, a Python interface to the stemming algorithms from the Snowball project [2].

2.2 Pre-processing Tweets

Besides the operations from the previous subsection, the following additional preprocessing was applied to Twitter data:

- Removing mentions from tweets, i.e. the @ sign followed by a Twitter handle.
- Removing the *RT* acronym at the beginning of retweets.

3 Text representation

Once the reviews corpus was correctly pre-processed, text representation techniques were applied.

The first text representation that was chosen is the computation of the TF-IDF matrix associated with the corpus. Because of the heterogeneity of the reviews corpus and its high dimensionality, the number of unique words is really high: therefore it was essential to identify features that help discriminating the different reviews.

The TF-IDF matrix was built using only the reviews training set: in a production use case it is certain that at a certain point new words will emerge, therefore this scenario was simulated by finding the most general features associated with positive and negative sentiments. Because of the high number of reviews it was decided to ignore really rare features: only words appearing at least in 5 documents were kept

in order to focus only on general words. After building a TF-IDF matrix each word has an associated weight: only the 50 000 features with the highest weight were kept in order to reduce dimensionality.

Both unigrams and bigrams were taken into consideration in the construction of the TF-IDF matrix. The bigram representation was chosen in order to allow predictive models to have a wider understanding of the meaning of the reviews: in this way words such as "not + adjective" will be seen as unique feature, whose expressive power is greater than that of the two single words.

The TF-IDF representation was applied to the pre-processed data

- Without stemming;
- After stemming with NLTK's Porter stemmer;
- After stemming with NLTK's Lancaster stemmer;
- After stemming with PyStemmer's Porter stemmer;

in order to compare the performances of the classifiers trained on these different representations.

Doc2Vec was considered as another method of obtaining a representation of the reviews, but was discarded due to poor performance and long training time.

4 Classification

As specified before the goal of the project is to build a predictive tool for labeling the sentiment of a given text. Different algorithms were evaluated against each other in order to evaluate their efficiency and efficacy; some of them were only applied using the output of the Truncated SVD algorithm.

Truncated SVD was used for reducing quite considerably the number of columns in the dataset: the linear dimensionality reduction was performed using the Python library *scikit-learn*, obtaining 200 new features that replaced the starting 50 000 ones.

- Multinomial Naive Bayes (MNB).
- Random Forest (RF).
- Truncated SVD + Random Forest (RF SVD).
- Truncated SVD + Linear SVC (LSVC SVD)
- Truncated SVD + SVC (SVC SVD)
- Truncated SVD + Adaboost (AB SVD)

As mentioned in Section 1, the positive class to predict consists of reviews with score 4 or higher, while the negative class is made up of reviews with score 3 or lower. The positive and negative classes are unbalanced, with the former making up 84% of the observations. In order to simplify the classification task, since each classifier is also evaluated on a different dataset consisting of tweets – where the positive and negative classes might show a different proportion –, the data was sampled in order to achieve a balance between the positive and negative classes. After this operation the dataset counts 500 000 observations for each class, and each classifier was trained on 67% of this data.

5 Evaluation

5.1 Evaluating the classifiers on the reviews test set

The results of the evaluation of the classifiers on the reviews test set can be found in Table 1 (the tables can be found after the references). A PC with an Intel i5-7500 @3.40 GHz CPU and 16 GB of RAM was used for training the different classifiers.

5.2 Evaluating the classifiers on tweets

A different strategy was adopted in order to evaluate the performance of the classifiers on the Twitter data, which consists of 2000 tweets containing the hashtag CES2020 (CESTweets), 5113 labelled tweets from the Sanders Analytics twitter sentiment corpus (SanTweets), and 289 labelled replies to a tweet by Samsung Mobile (SMTweets).

Besides having positive or negative sentiment, tweets can also be neutral: this is the case, for instance, for the replies to Samsung Mobile's tweet that simply consist of questions regarding the expected release date of Samsung's new smartphone. Because the classifiers trained on the Amazon Reviews Dataset can only predict two classes, namely *positive* and *negative*, the neutral tweets needed to be filtered out. This was achieved in two ways:

- Discarding tweets labelled as *neutral* from the SanTweets and SMTweets data and evaluating the positive/negative classifier on the rest of the tweets.
- Using an additional classifier that labels each tweet as either being neutral or expressing some sentiment before employing and evaluating the positive/negative classifier. This additional classifier was built by resorting to SentiWordNet [3], a lexical resource that assigns to each synset of WordNet three sentiment

scores: positivity, negativity, and objectivity. This classifier (henceforth referred to as SWNClassifier) takes in input the pre-processed tweets, performs *Part of Speech tagging* and computes the positivity and negativity scores of a tweet as the sum of the SentiWordNet positivity and negativity scores of each token/tag pair. Based on these scores, each tweet is assigned either a positive, negative or neutral label (this approach roughly follows [4]).

The performance of the positive/negative classifier depends heavily on whether the data provided as its input is correctly classified as expressing some sentiment or not. Therefore, the scores of this classifier on the manually filtered data might be optimistic, as this method is not realistic in a production scenario, while its scores on the data filtered by the SWNClassifier might be pessimistic, due to the simplicity of the algorithm. Indeed, this classifier's capability at detecting sentiment in a tweet was tested on SanTweets and SMTweets: the results are shown in Table 2. A better approach could consist in training a more reliable classifier that detects the presence or absence of sentiment in a text in order to perform this filtering before feeding the data to the positive/negative classifier.

Among the trained classifiers, Multinomial NB was by far the most efficient one, having train and test times orders of magnitude lower than the others, and the not stemmed dataset was the one that achieved the best values on accuracy, precision and recall. For these reasons it was decided to use only this classifier for evaluating the following sets of tweets:

- CESTweets labelled as either positive or negative by the SWNClassifier.
- SanTweets with positive and negative labels.
- SanTweets labelled as either positive or negative by the SWNClassifier.
- SMTweets with positive and negative labels.
- SMTweets labelled as either positive or negative by the SWNClassifier.

The results can be found in Table 3

Conclusion

Building a classifier that is able to correctly label the sentiment in a given text is not an easy task. The Amazon Review Data is a good starting point for building a predictive model for accomplishing this task, however its large dimensionality made the pre-processing tasks computationally demanding. The Spell checking correction

phase, for example, was hampered by the dataset high dimensionality and only the lightest possible modification was performed for ensuring acceptable execution times.

Regarding the stemming procedure, it was somehow surprising that its use did not lead, in any classification algorithm, to an in improvement in accuracy or recall: only for the Random Forest algorithm using the Lancaster or Porter stemming algorithms led to a slight improvement in the precision value.

The Multinomial NB algorithm has proven to be the best algorithm among those tested thanks to its efficiency that did not impact its predictive capabilities. Another of its advantages is that it was trained on the entire dataset in feasible times, while the other ones, apart from Random Forest, were trained exclusively on the dataset after the application of SVD. These results also allow to deploy this classifier in scenarios where hardware resources are limited, further increasing its future applications.

The evaluation on Twitter data shows that the approach of training a sentiment classifier on reviews is also promising in the scenario of an application to sentiment analysis on Tweets, but this classifier should be better tested after neutral tweets have been filtered out by a reliable *Sentiment* Vs. *NoSentiment* classifier, instead of manually filtering out neutral tweets or relying on a simple algorithm like the one behind the SWNClassifier.

References

[1] Amazon Review Data. Retrieved from `https://nijianmo.github.io/amazon/index.html` (January 2020)

[2] Github. *PyStemmer*. Retrieved from `https://github.com/snowballstem/pystemmer` (January 2020)

[3] Github. *SentiWordNet*. Retrieved from `https://github.com/aesuli/sentiwordnet` (January 2020)

[4] Medium, Towards Data Science, August 2019. *Sentiment Analysis on Swachh Bharat using Twitter*. Retrieved from `https://towardsdatascience.com/sentiment-analysis-on-swachh-bharat-using-twitter-216369cfa534` (January 2020)

[5] NLTK 3.4.5 Documentation. *nltk.tokenize package*. Retrieved from `https://www.nltk.org/api/nltk.tokenize.html` (January 2020)

[6] NLTK 3.4.5 Documentation. *Source code for nltk.tokenize.punkt*. Retrieved from `https://www.nltk.org/_modules/nltk/tokenize/punkt.html` (January 2020)

[7] Norvig, Peter. *How to Write a Spelling Corrector*. Retrieved from `https://norvig.com/spell-correct.html` (January 2020)

[8] Sanders Analytics Twitter Corpus. Retrieved from `https://github.com/zfz/twitter_corpus` (January 2020)

[9] Twitter. *Replies to Samsung Mobile's Tweet*. Retrieved from `https://twitter.com/SamsungMobile/status/1159826737770856448` (January 2020)

Table 1: Results on Amazon test data

Algorithm	**Stemming**	**Accuracy**	**Precision**	**Recall**	**Train time**	**Test time**
MNB	No Stemmer	0.84922	0.86423	0.82880	219 ms	47 ms
	Porter	0.84598	0.86120	0.82508	271 ms	78 ms
	Lancaster	0.84457	0.86029	0.82293	203 ms	63 ms
	PyStemmer	0.84679	0.86221	0.82568	203 ms	47 ms
RF	No Stemmer	0.84545	0.87829	0.80220	41 m 58 s	6 s
	Porter	0.84542	0.88152	0.79828	42 m 27 s	6 s
	Lancaster	0.84452	0.88293	0.79453	42 m 38 s	6 s
	PyStemmer	0.84536	0.88044	0.79941	41 m 35 s	6 s
RF SVD	No Stemmer	0.80865	0.82979	0.77681	3 m 5 s	2 s
	Porter	0.80826	0.83311	0.77119	3 m 9 s	2 s
	Lancaster	0.80651	0.83221	0.76806	3 m 5 s	2 s
	PyStemmer	0.80728	0.83103	0.77163	3 m 6 s	2 s
LSVC SVD	No Stemmer	0.81442	0.82533	0.79786	5 m 29 s	125 ms
	Porter	0.81379	0.82199	0.80129	4 m 17 s	125 ms
	Lancaster	0.81085	0.81992	0.79688	4 m 34s	125 ms
	Pystemmer	0.81131	0.82105	0.79637	3 m 59 s	125 ms
SVC SVD	No Stemmer	0.50385	0.50208	0.98761	2 m 56 s	1 m 9 s
	Porter	0.52300	0.51241	0.95858	2 m 56 s	1 m 9 s
	Lancaster	0.49975	0.49999	0.95143	2 m 56 s	1 m 9 s
	PyStemmer	0.52013	0.51060	0.98046	2 m 55 s	1 m 9 s
AB SVD	No Stemmer	0.74446	0.74800	0.73768	6 m 56 s	2 s
	Porter	0.74038	0.75414	0.71368	6 m 59 s	2 s
	Lancaster	0.73868	0.74714	0.72194	6 m 59 s	2 s
	PyStemmer	0.74495	0.75732	0.72128	7 m	2 s

Table 2: SWNClassifier's performance in classifying *sentiment* against *no sentiment*

Dataset	**Accuracy**	**Precision**	**Recall**	**Test time**
SanTweets	0.48680	0.93148	0.37519	8.11 s
SMTweets	0.57093	0.75000	0.15217	499 ms

Table 3: Multinomial Naive Bayes performance on Twitter data

Dataset	**Labelling**	**Accuracy**	**Precision**	**Recall**	**Test time**
CESTweets	SWN	0.56080	0.81130	0.51424	998 μs
SanTweets	Manual	0.71402	0.78202	0.55299	998 μs
	SWN	0.58689	0.73879	0.51684	978 μs
SMTweets	Manual	0.95364	0.83333	0.78947	997 μs
	SWN	0.62835	0.78846	0.32283	0 ns