Opinion mining project : Sentiment Analysis on the IMDB movie review dataset

Hugo Vallet, Ivaylo Petkanchin and Wiem Gharbi

***Abstract*— This report describes our "Sentiment Analysis and Text Mining" python project. It was part of the "Advanced big data analytics" (MAP670B) course of the Ecole Polytechnique led by M. Vazirgiannis.**

I. INTRODUCTION

The area of text analytics (or text mining) includes techniques from multitude scientific areas (A.I., statistics, linguistics), and it has a wide range of applications (security, marketing, information retrieval). One of them is that of sentiment analysis and opinion mining. Opinion mining uses a subset of text analytics techniques with the goal to categorize opinions/reviews within a pre-specified range of non-favorable to favorable rankings.

In the scope of this project we deployed Sentiment Analysis techniques using supervised learning algorithms on the IMDB movie review dataset to create a model able to predict the sentiment of a given review (positive or negative).

II. TASK OVERWIEW

We are given two sets of movie reviews:

- A training set of 25,000 documents containing an equal number of positive and negative reviews.
- A test set of 25,000 documents containing unlabeled reviews to be classified as either positive or negative.

In the IMDB dataset, every user was required to include a rating for each movie in the range of 1 to 10. In this collection, no more than 30 reviews were allowed for every movie in order to avoid an overpopulation of reviews with similar ratings. In the labeled dataset:

- negative reviews were considered as such if they had a rating less than or equal to 4.
- Positive reviews were considered as such if they had a rating higher than or equal to 7.

With this approach, neutral reviews are eliminated as they usually introduce noise to the dataset.

The data is imported as a list of strings, i.e. every review is considered as a unique string containing spaces. To be able to learn a model, we first need to extract descriptive features from the raw text. Then we can learn predictive model from these features. Finally, we can use the learned model to predict the sentiment of the test reviews. The following graph summarizes this process :

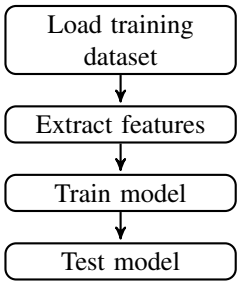


Fig. 1. The process to be implemented

A. Feature extraction

In this step we need, first, to tokenize our reviews. That is to say : cut the documents into "real" words. In our process we decided that words are separated by spaces. After this step, we can apply a lot of transformations to the words extracted : remove stop-words (highly frequent words which bring no information about the text itself), apply stemming (i.e. keep only the root of each word), etc. We describe the transformation we used in the next sections.

B. Model Training

Multiple models can be trained for sentiment analysis. Some recent studies showed that, for that task, simple models like linear SVM and Multinomial Naive Bayes can outperform state-of-the-art text mining algorithms such as Word Representation Restricted Boltzmann Machine[1].

C. Model testing

The final goal of the project is to have a classifier trained to separate positive reviews from negative reviews. As far as it is a binary classification problem, to assess the performances of the classifiers we used ROC curves. Nevertheless, in the scope of this problem, false positives and false negatives have the same weights on the final error. Consequently, the accuracy was the criterion retained to rank the classifiers we trained.

III. FIRST STEP : TFIDF AND LINEAR MODELS

A. Analysis of the data's characteristics

To have a better understanding of the data, we retrieved some insights on the training data such as the most frequent words in each class and the average length of the reviews. We have also considered the number of unique words in our training set. We obtain a corpus of the size 281158 unique words.

Fig. 2. WordCloud of the training dataset after removal of the html tags and removal of the stopwords.

We observed that some of the reviews contained html tags. In order to remove them from the original documents, we used the BeautifulSoup Python library. The next step was to tokenize, remove stop words, remove punctuation and stem the corpus. Finally, using sklearn FeatureExtraction library we were able to get the bag-of-word representation of our set of reviews and its TF-IDF matrix.

$$\begin{array}{c} \\ review_1 \\ \vdots \\ review_N \end{array} \begin{array}{c} \begin{array}{ccc} word_1 & \dots & word_P \end{array} \\ \begin{pmatrix} f(1,1) & \dots & f(1,P) \\ \vdots & & \vdots \\ f(N,1) & \dots & f(N,P) \end{pmatrix} \end{array}$$

Fig. 3. Tokenized representation of the documents corpus. We have N reviews from which we extract P different words. The f function is the value associated to a word in a document. It could be, for example, the tf-idf of the word in that document.

As a first approach, we used for the function f, the tf-idf. That is to say :

$$f(i,j) = tfidf(word_i, doc_j)$$

In order to compute the tfidf matrix of our documents, we have used the class tfidf vectorizer from the the feature extraction library of sickit-learn. The tfidf vectorizer uses the following formula to compute the tfidf function:

$$f(i,j) = TF(i,j) * (IDF(i) + 1)$$

- TF(t) = Number of times term t appears in a document / Total number of terms in the document
- IDF(t) = log(Total number of documents / Number of documents with term t in it)

Using this formula, terms with zero IDF, i.e. terms that appear in every document, will not be completely ignored compared to the formula TF(t)*IDF(t)

Classically, using this function, the matrix described above becomes really sparse since there are a lot of words that appear in only few documents of the corpus. This allows to decrease tremendously the memory needed to store it because we can store only the indexes and values of the non-null coefficients. This kind of storage is implemented in numpy.sparse library and permits basic cost-efficient matrix operations.

Words extracted	n-grams per review	dictionary size	sparsity
1-grams	223	280k	82%
2-grams	453	1522k	98%
3-grams	666	5183k	99%

TABLE I

SPARSITY OF THE DATA MATRIX ON THE TRAIN DATA SET. HTML TAGS REMOVED, STOP WORDS NOT REMOVED, NO STEMMING APPLIED

B. *Training our first models*

Since our feature space is of a considerable size, we used, as a start, very simple learning algorithms : Linear SVM, Logistic Regression (LR) and Multinomial Naive Bayes (MNB). Stop-word removing and vocabulary stemming are classical text mining pre-processing techniques. We checked their impact on our final accuracy.

	1-grams	2-grams	3-grams
MNB/NoSW/Stem.	0.83 (+/- 0.02)	0.86 (+/- 0.01)	0.86 (+/- 0.01)
MNB/SW/Nostem.	**0.84** (+/- 0.01)	**0.88** (+/- 0.01)	**0.89** (+/- 0.01)
SVM/NoSW/Stem.	0.88 (+/- 0.01)	0.88 (+/- 0.01)	0.88 (+/- 0.01)
SVM/SW/Nostem.	**0.88** (+/- 0.01)	**0.89** (+/- 0.01)	**0.89** (+/- 0.01)
LR/NoSW/Stem.	0.87 (+/- 0.01)	0.87 (+/- 0.01)	0.86 (+/- 0.01)
LR/SW/Nostem.	0.87 (+/- 0.01)	0.87 (+/- 0.01)	0.86 (+/- 0.01)

TABLE II

SW : KEEPING STOP-WORDS IN THE INPUT DATA. STEM : APPLY STEMMING TO THE INPUT DATA. THE SVM'S C PARAMETER WAS ARBITRARILY FIXED TO 1 IN EVERY CASE. THE ACCURACY PRESENTED WAS AVERAGED ON A 5-FOLD CV USING 10K EXAMPLES (40% OF DATA) FOR TESTING

From the previous results we come to the following conclusions:

- Stop-word removing and stemming is not really a good option on this dataset. This is quite understandable : some words that are considered as stop words in the nltk.stopwords library that we have used are meaningful to the sentiment analysis task such as "not" "very" "nor".
- Taking n-grams of larger size increases the accuracy because we are working on long reviews with, possibly, repeating long patterns. Nevertheless increasing the length of n-grams generates an additional computational cost.
- Among classical linear classifiers, SVM seems to be the most robust learner. This is coherent with Wang and Manning's work[1][2].

We can also look for the best C penalty for the SVM using a grid search with, again, a 5-fold CV and 40% of data for testing :

	10^{-2}	10^{-1}	1	10	100
Mean accuracy	81.9%	87.3%	89.4%	89.4%	89.4%
Std	0.007	0.007	0.004	0.004	0.004

C = 1 gives a good accuracy in cross-validation and allows to improve a little our results.

C. First approach conclusion

In the light of the previous results, we adopted the following approach:

- We do not use stemming on our tokenized data-set. As in the previous section, the obtained accuracy with previous linear models decays when using stemming. Stemming tries to cut off details like exact form of a word and produces word bases as features for classification. Sometimes, however, these details play an important role by themselves and should be thus considered as meaningful features.
- When building our bag of words model, we limit ourselves to the use of bigrams as trigrams does not significantly improve the performance of the models (compared to bigrams) and is computationally expensive.

IV. 2ND STEP : FEATURE ENGINEERING AND NB-SVM

A. TW-IDF

Semantically speaking, the TF-IDF method does not take into account nor the word order neither the word dependence. To counter this, we have implemented another weighted method which has been proposed in Rousseau and Vazirgiannis (2013) TW-IDF[3]. The basic concept of this approach is to consider each document as a unweighted directed graph of words. We have used the simplified version of the term weighting scheme given by the following formula:

$$TW - IDF(t,d) = TW(t,d) * IDF(t)$$

- TF(t,d) is the weight of the vertex t (term t) in the graph of word representation of the document d
- IDF(t) = log(Total number of documents / Number of documents with term t in it)

We obtain the following results using the cross validation score of various models (Linear SVM, Logistic Regression and Multinational Naive Bayes). We compared the cross validation score on the training dataset to the cross validation score given by the same models on the same datasets using TF-IDF. The following comparison was made using 137805 features. The feature space was designed using unigrams. We have used a sliding window set to 2. We have tried several values for the sliding window parameter for the TW-IDF implementation, the value 2 gave the best performance.

	TF-IDF	TWIDF
SVC	0.88(+/-0.01)	**0.89 (+/-0.01)**
Logistic Regression	0.88(+/-0.01)	0.88 (+/-0.01)
MNB	0.84(+/-0.01)	0.84 (+/-0.01)

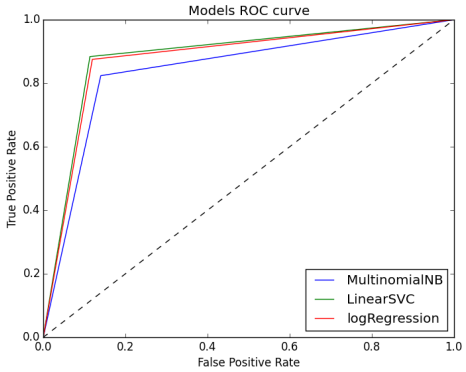


Fig. 4. Models ROC curve using TW-IDF as the feature matrix

To conclude, using the TW-IDF representation hardly improves the performance of our basic linear models compared to the TF-IDF representation. Additionally, our TW-IDF approach is computationally expensive compared to TF-IDF as we are to construct a graph for each of the 25000 documents and compute the term weight for each word in the graph. TW-IDF efficiently improves the performance of prediction compared to TF-IDF when the documents are quite lengthy (>1000 unique words)[3] which is not the case in our dataset.

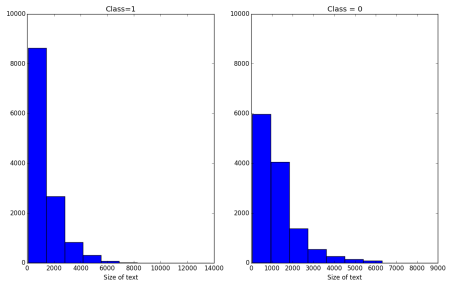


Fig. 5. Length of document per class (O: negative, 1:positive)

B. Feature engineering considerations

To improve the performance of our learning algorithms, we tried to add new meaningful features. Typically we considered new features such as the length of the reviews, the type of numbers they contain, the number of uppercase letters, the number of exclamation points, etc.

When considering only the matrix of the new features, we managed to get over 0.60 of accuracy using a GradientBoosting Classifier. However the important part was to combine this new information with the regular TF-IDF Matrix to create a better classifier.

C. Using NB-SVM instead of the TF-IDF

To improve our model, we also tried to change the TF-IDF coefficients to more meaningful coefficients. As describe in Wang and Manning, for the sentiment analysis task, one way to make the SVM classification more efficient is to use, as input data, the NB coefficients of our dictionary instead

of the classical TF-IDF. The NB coefficients are obtained through the following procedure :

- let "pos" and "neg" be the ensembles of the positive and negative training reviews, we compute the p and q vectors defined by $\forall i \in [1, ., P]$,
  - $p_i = \alpha + \sum_{doc_j \in pos} count(word_i, doc_j)$
  - $q_i = \alpha + \sum_{doc_j \in neg} count(word_i, doc_j)$

  where alpha is a smoothing parameter.
- we compute the NB vector "r" given by $\forall i \in [1, ., P]$, $r_i = \ln \dfrac{p_i/||p||_1}{q_i/||q||_1}$
- then we take the binarized word count matrix $\hat{f}$ (which is exactly the matrix describe in fig. 2 with $f(i,j) = 1$ if the $word_i$ appears in $review_j$, 0 if not) and multiply coefficient-wise its lines by the NB vector r to get the NB matrix $\tilde{f} = \hat{f} \odot r$.

Interpretation of the ratio : if $word_i$ appears more often in positive reviews than in negative reviews then we have $r_i > 0$, otherwise $r_i < 0$. This ratio is just a way to quantify how "positive" or "negative" a word is using the Bayes rule to compute the probability of being in a negative or positive document.

At this step we have the NB matrix instead of the TF-IDF matrix as an input. Like before, we can apply a linear SVM on it. We have, then, the NB-SVM classifier described by Wang and Manning. We estimated the best couple of (α,C) through a 5-fold cross-validation using 40% of the training set for cross-val testing.

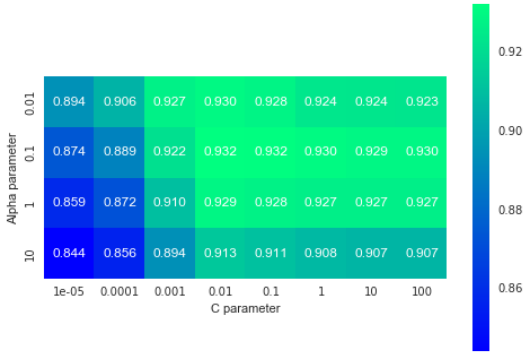
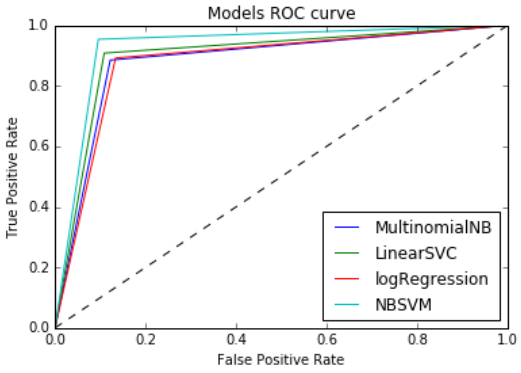


Fig. 6. Heatmap of the CV test error generated with different couples (α,C). We see that, in fact, the parameters don't have a huge impact on the final prediction quality. Alphas ranging from 0.01 to 1 and C ranging from 0.001 to 100 all give an accuracy on test set higher than 90%.

The next graph recaps the performances of all the models previously learned, parameters fixed through grid search (NBSVM with alpha = 0.1 and C = 0.1, SVM with C = 1).



D. Ensemble methods

As stated in a previous section we wanted to combine the TF-IDF and the information contained in the engineered features. To do so we decided to build a multi-layer classifier. The first layer is composed of multiple basic classifiers such as MNB or Logistic Regression fitted on the TF-IDF Matrix, a GradientBoosting Classifier fitted on the matrix containing the new features and the NBSVM model. Using these models, we would compute the probability vectors for the test set and store them in a probability array. As this probability array contains highly correlated data, we observe that performing a dimensionality reduction is a good idea.

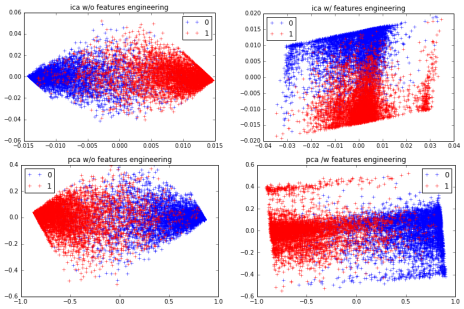


Fig. 7. 2 dimension probability array after PCA and ICA distribution in function of labels 0 and 1 and with/without features engineering.

	Basic model	Ensbmod w/o feat-eng	Ensbmod w/ feat-eng
Accuracy	0.89 (+/-0.004)	0.91 (+-0.003)	0.91 (+/-0.003)

Basic model = Logistic Regression, Ensbmod = ensemble method

The Values in the previous table are achieved with a random split of 40% for test. We perform 5 different random split and average the scores. We can see that in the end the feature engineering does not add any precision. It is the combination of the NBSVM and the TF-IDF that improves the accuracy up to 0.91. The drawback of this model is the necessity to split the train data in order to create the dataset of probabilities needed to train the second Layer. For this second Layer, a GradientBoosting algorithm seems to give the best results in average even if we are in the margin of error with the other models (RandomForest, SVC,..).

V. CONCLUSIONS

In this project we tried multiple approaches combining different pre-processing methods with different learning algorithms. We came to the conclusion that over the number of models that we have used, ranging from complex models (such as ensemble methods or TW-IDF feature mapping) to more simple ones (such as linear models with basic TF-IDF), the trade off between complexity and interpretability was best met with the NB matrix representation of features and Linear SVM as a classification model.

As our feature space is very sparse, we did not deem it necessary to use spark. Instead we have used sparse matrix representation (scipy.sparse) which is supported by sickit-learn classification algorithms.

REFERENCES

[1] Sida Wang and Christopher D. Manning, "Baselines and Bigrams: Simple, Good Sentiment and Topic Classification", Stanford University, 2012

[2] Mesnil, Mikolov, Ranzato, and Bengio, "Ensemble of generative and discriminative techniques for sentiment analysis of movie reviews", 2015, arXiv:1412.5335v7 [cs.CL] 27 May 2015.

[3] Rousseau, Vazirgiannis "Graph-of-word and TW-IDF: New Approach to Ad Hoc IR", 2013, pages 59-68 of: Proceedings of the 22nd ACM international conference on Conference on information and knowledge management. ACM