COMP 551 Mini project 2

Group 21
Members:
Mona Naghashi, 260888871
Charles McCluskey, 260688016
Grégoire Moreau, 260874685

February 2019

1 Abstract

Sentiment analysis refers to the detection of attitudes and opinions of people about a subject, such as film reviews, restaurant customer opinions etc. Information overload and large numbers of reviews have increased the need for high-performance automatic tools.

In this machine learning project, the data consisted of IMDB movie reviews and was separated in two classes: positive and negative. Multiple approaches were used to classify the reviews, such as Naive Bayes classifiers and logistic regressions. A Naïve Bayes classifier is one of the baseline techniques for text analysis and its basic idea is assigning class probabilities to texts by using the joint probabilities of words and classes.

Our final model used stacking, a technique that uses multiple underlying classifiers to give a prediction. It can classify unseen IMDB movie reviews as positive or negative with an accuracy of almost 90%.

2 Introduction

The ultimate goal of mini project 2 is to use a dataset of positive and negative IMDB comments, each class set being 12500 reviews in size for a total of 25000, to automate the determination of the positivity or negativity of previously unseen comments. The training dataset and the testing dataset (25000 different comments) were provided by Kaggle, who also hosted an independent and private test set that our models would be tested on.

We implemented four independent approaches for our classifiers: Bernoulli Naive Bayes, Logistic Regression, Decision Trees, and a stacking configuration of classifiers. The Bernoulli Naïve Bayes classifier was implemented by ourselves, while the Logistic Regression and Decision Tree classifiers were provided by scikit-learn.

Our final approach, stacking, involved performing logistic regression on the probability predictions of multiple classifiers; more specifically, our stacking implementation regressed on five different classifiers: Naïve Bayes, Logistic Regression, Decision Trees, and Support Vector Machines all provided by scikit-learn, and NLTK's sentiment analysis tool. We discovered that the stacking approach created the best performing model, with a Kaggle score of almost 90%.

3 Related works

One of the first resources we looked at was a guide on sentiment analysis hosted by monkeylearn.com (https://monkeylearn.com/sentiment-analysis/). While the document did not go into extreme detail, nor did it describe any specific project, it did provide a significant amount of contextual and "broad stroke" information regarding the topic, as well as certain pitfalls and tricky cases to consider.

4 Dataset and setup

The dataset we were provided is a collection of 25000 training files, with 12500 of them expressing a positive sentiment and the other 12500 expressing a negative sentiment. We were also provided a testing set of 25000 files of unknown sentiment. Setup was made relatively simple by writing a pair of methods in a text processing file: the first method, getAllData, extracted the names of all the positive and negative files and stored them into their own respective lists. These lists were then stored in a dictionary with keys "pos" and "neg", which was then returned to the caller. The second method, openComment, would take the generated dictionary, a "pos" or "neg" string, and an index number. This method would open a specific text comment with the index number allowing for simple iteration through either lists.

5 Proposed approach

5.1 Implementation of Bernoulli Naive Bayes

The provided data contain a vocabulary size in excess of 86000 words. Since the number of training reviews (texts) is 25000, training the Bernoulli Naïve Bayes classifier with this number of words (features) is expected to give poor results.

Therefore, instead of considering the occurrence of each word in our corpus, a better approach should be taken. To decrease the number of words and vocabulary size, the most frequent words in training corpus have been found based on their occurrences in the IMBD training reviews. To do so, we must first preprocess the reviews.

Indeed, reviews written by people on social media contain lots of noise and misspellings, such as the use of abbreviations, poor punctuation, and misspelled words. As a result, this makes the feature space sparse and high dimensional. In order to alleviate such problems, various types of preprocessing methods are exploited in this work. For example, stop words and punctuation have been removed and words are converted to their lowercase versions.

After pre-processing the review texts, the 10000 most common words were selected to define our vocabulary words. Conditional probability of each feature (each of the 10000 most common words) in each category was estimated by dividing the frequency of that word's occurrence in one category by its overall frequency. These conditional probabilities related to one class were then multiplied with each other. The result was then further multiplied by class prior probability to approximate the corresponding posterior probability.

The class having the maximum posterior probability was selected as the predicted one. In testing, the words of the test document are checked one by one and, based on the existence of each word in our vocabulary, we assign a class conditional probability or one minus that class conditional probability. Those terms are then multiplied and the result gives the posterior for a specific class. Prediction relies on the maximum of the two calculated posteriors. Note that class priors are equal for this dataset and they can be neglected.

5.2 SciKit learn models

We experimented with 2 different Scikit learn classifiers: LogisticRegression and DecisionTreeClassifier. We used Scikit learn pipelines to make their testing more convenient. Both models use unigrams, bigrams and trigrams as features. Logistic regression was tested using binary counts of words and tfidf scores, and has a limit of 20,000 features as increasing that number didn't improve the classifier's accuracy on the validation set.

The decision tree classifier uses tfidf scores as features and has a limit of 75000 features. We also impose a maximum depth of 20 on the created decision tree to prevent it from overfitting to the training set.

We used CountVectorizer in binary mode to extract binary word counts as features and CountVectorizer followed by TfidfTransformer to get tfidf scores as features. The pipeline also includes a normalizer to regularize the features.

5.3 Stacking

Our final model uses stacking with logistic regression on the prediction 5 different classifiers to make decisions. The five classifiers are: the logistic regression and decision trees described above, Scikit learn svm.SVC and Naive Bayes classifiers, and nltk's SentimentIntensityAnalyzer.

It is trained in 3 phases. First the independent classifiers are trained on the first half of the training set and give prediction probabilities on the second half of it. Then the independent classifiers are trained on the second half of the training set and give their prediction probabilities on the first half of it. Finally, the stacking predictor uses logistic regression on the combined prediction probabilities of the first two phases to train itself. Our first iteration of the stacking classifier trained the independent classifiers on the whole training set, then made them give their prediction probabilities on that same set and used those probabilites to train the stacking predictor but that didn't give satisfying results as some classifiers such as the decision tree tended to overfit to the training set which means that classifier was given too much weight by the stacking predictor.

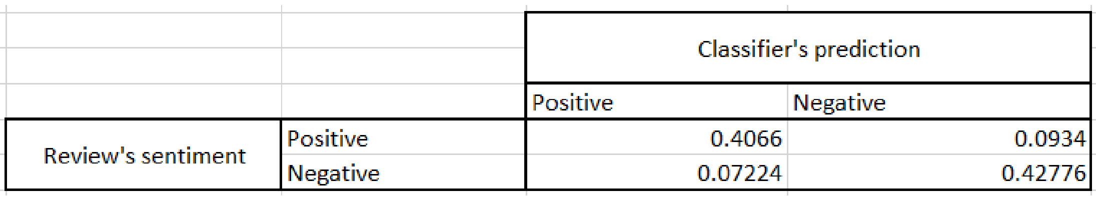
5.4 Validation

All of the models were tested using 5-fold cross validation. The comments were interleaved before every model training (one positive, one negative, one positive,...) to be sure to have equals weights for both classes during training.

6 Results

6.1 Bernoulli naive Bayes

The Bernoulli naive Bayes classifier, implemented ourselves, provided us with the following results: Our accuracy on the training set was : 0.87425, and our accuracy on the validation set was 0.83436. This produced the confusion matrix in figure 1:
Running 5-fold cross validation on this model takes an average of 42 minutes. When uploaded to kaggle, it provided us a score of 0.83640.



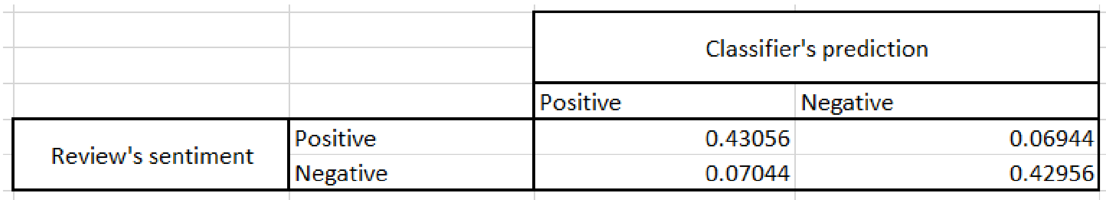
		Classifier's prediction	
		Positive	Negative
Review's sentiment	Positive	0.4066	0.0934
	Negative	0.07224	0.42776

Figure 1: Confusion matrix for Bernoulli Naive Bayes

6.2 Logistic regression

6.2.1 Using tfidf scores

Logistic regression provided by scikit learn trained with tfidf scores as features gave us the following results: Our accuracy on the training set was 0.93956, and our accuracy on the validation set was 0.86012. This produced the confusion matrix in figure 2:
Running 5-fold cross validation on this model takes an average of 2 minute and 50 seconds.



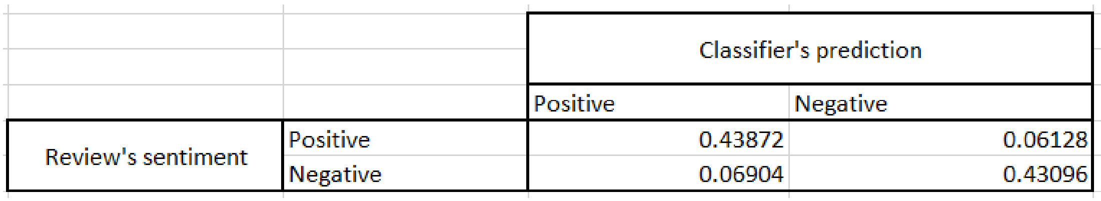
		Classifier's prediction	
		Positive	Negative
Review's sentiment	Positive	0.43056	0.06944
	Negative	0.07044	0.42956

Figure 2: Confusion matrix for logistic regression with tfidf scores

When uploaded to kaggle, it provides a score of 0.878.

6.2.2 Using binary word counts

Logistic regression provided by scikit learn trained with binary word counts as features gave us the following results: Our accuracy on the training set was 0.91777, and our accuracy on the validation set was 0.86968. This produced the confusion matrix in figure 3:
Running 5-fold cross validation on this model takes an average of 2 minute and 50 seconds.



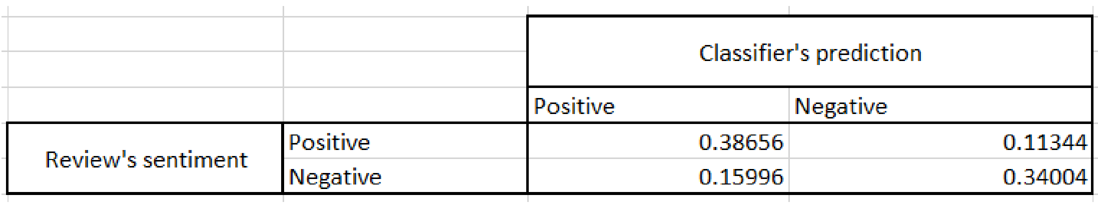
		Classifier's prediction	
		Positive	Negative
Review's sentiment	Positive	0.43872	0.06128
	Negative	0.06904	0.43096

Figure 3: Confusion matrix for logistic regression with binary word features

When uploaded to kaggle, it provides a score of 0.894.

6.3 Decision trees

Decision trees provided by scikit learn provided us with the following results: Our accuracy on the training set was 0.85172, and our accuracy on the validation set was 0.7266. This produced the confusion matrix in figure 4:
Running 5-fold cross validation on this model takes an average of 3 minute and 50 seconds. It was not tested on Kaggle.

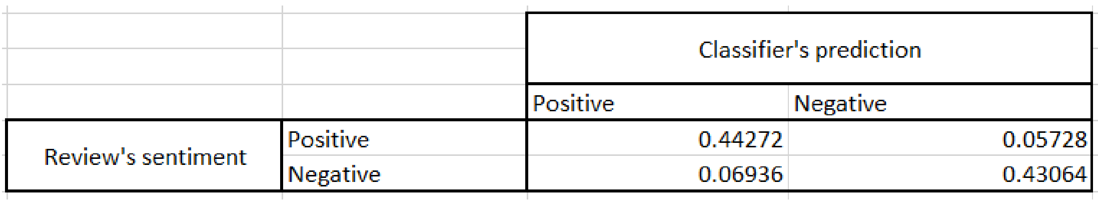


		Classifier's prediction	
		Positive	Negative
Review's sentiment	Positive	0.38656	0.11344
	Negative	0.15996	0.34004

Figure 4: Confusion matrix for the decision tree classifier

6.4 Stacking using logistic regression

Stacking with logistic regression provided us with the following results: Our accuracy on the training set was 0.92217, and our accuracy on the validation set was 0.87336. This produced the confusion matrix in figure 5:
Running 5-fold cross validation on this model takes a little more than an hour. When uploaded



		Classifier's prediction	
		Positive	Negative
Review's sentiment	Positive	0.44272	0.05728
	Negative	0.06936	0.43064

Figure 5: Confusion matrix for the stacking classifier

to kaggle, it provided us a score of 0.89586 which was our best score in this competition.

7 Discussion and Conclusion

Our first takeaway from this project is that the simple binary counts of words can give better models than models that use tfidf scores, as seen in the results of logistic regression. From the results described above, we can also see that the classifier that performs best is the one that uses stacking. This model could be improved further by improving each of its underlying classifiers by tuning their parameters more precisely for this specific problem, or by selecting classifiers that are make more distinct assumptions than our current classifiers.
Finally, we can notice that the runtime of the stacking classifier is around 20 times longer than the one using simple logistic regression despite only improving the accuracy by less than one percent. In the future, it would thus be important to optimize the training of such a model.

8 Statement of Contributions (1-3 sentences)

1) Charles implemented comment organization and text extraction, as well as error handling as not all comments contained unicode-valid characters.
2) Mona implemented a Bernoulli Naive Bayes model from scratch (i.e., without using any external libraries such as SciKit learn) and she also performed text preprocessing for that naive Bayes implementation.
3) Grégoire implemented the validation pipeline and the stacking classifier, and tested the different models.