Sentiment analysis of IMDb movie reviews

Machine learning (198:536) - Sring 2015
Rutgers University
Alejandro Pelaez, Talal Ahmed, Mohsen Ghassemi

1. Introduction

We competed in the Kaggle competition Bag of Words Meets Bags of Popcorn. It is basically a sentiment analysis challenge, where we have movie reviews labeled as positve or negative, and the challenge is to predict the sentiment of a new review. In the following writeup we explain several of our approaches, including a deep learning and information retrieval approach, which combined got us 2nd place in the competition at the time of writing.

2. Preprocessing

Before performing any algorithm, we have to make sure to clean up the data, in order to make it easier to process. Also, by finding and removing noise words in advance, we can increase greatly the accuracy of our algorithms.

2.1. Cleaning the data

The IMDb reviews contains html tags which do not serve any purpose for detecting sentiment, we also decided to remove punctuation whatsoever, even if this means that we get rid of emoticons (there are very few of them anyway), but it makes it easier for us to handle. Finally we lowercase everything.

We also apply Porter stemming algorithm, which helps us replace every word with it's root, and so words like cats and cat, or running and run, become the same. This has been shown to improve classification accuracy in sentiment analysis tasks.

2.2. Stemming

Another step that helps improve classification performance is what we call negation handling, intuitively words that are preceded by a negation (i.e no, not, hardly, etc.) means the opposite, and thus we replace every patter of the form *[negation] [word]* by *neg_[word]*. So for example, *hardly good* would be replaced by *neg_good*.

2.3. Negation handling

Finally we remove words that do not carry any sentiment, there is a useful set already available using the NLTK package in python. It include words like, and, or, then, etc.

3. Mutual information

One approach that we found to improve every single one of the algorithms accuracy, is to first compute the mutual information of the words with respect to the class label, and then keep only the top $X\%$ of the words, wher we chose X by cross validating, and we found that keeping between 40% to 50% gives the best results. This is intuitively obvious as we are basically doing feature selection and removing the noisy words that do not carry sentiment.

Let W_i be the indicator of word i. Let Y be the random variable that represents the class label. So $p(Y = 1)$ is the probability of observing a positive review, and $p(Y = 0)$ is the probability of observing a negative review. Booth quantities can be estimated using the empirical distribution.

The information of word i with respect to Y is then defined as

$$\sum_{(w,y)\in\{0,1\}^2} p(w,y)\log\left(\frac{p(w,y)}{p(w)p(y)}\right)$$

Where $p(w, y) = p(w|y)p(y)$, and $p(w|y)$ is estimated in the same way as $p(w)$ but just restricting ourselves to reviews with sentiment y. We can also perfomr the same analysis for bi-grams.

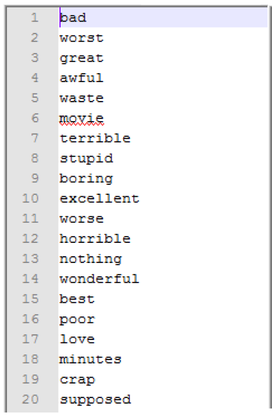
After we compute the mutual information for every word (bi-gram) and then sorting, we get the results showed in Figures 1 and 2. As expected, words like bad, worst, best are very useful for classification, or bigrams like waste time, really bad and must see. There is however one word that caught our attention, and that is the sixth word in the unigrams, namely *movie*. It shouldn't be expected to provide any good indication of whether the review is good or bad, as it should appear everywhere. We will explain this issue and how to deal with it in the next section.

4. Tf - Idf

Tf-Idf stands for term frequency, inverse document frequency. And it is a very useful technique used mainly in information retrieval in order to rank how important a keyword is to a given document in a corpus[1].

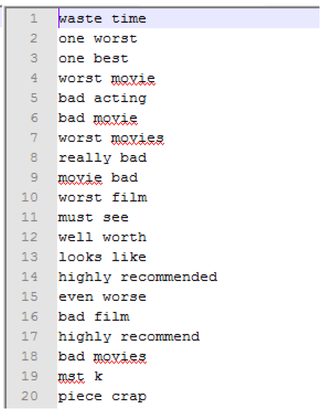
Returning to the word *movie*. This made us think that we still had garbage, and we had the idea that maybe a word can be very important to the whole set of reviews, but not to any single review by itself, and thus we needed a way of computing which words were important to which

[1]A corpus is a set of documents.



```
bad
worst
great
awful
waste
movie
terrible
stupid
boring
excellent
worse
horrible
nothing
wonderful
best
poor
love
minutes
crap
supposed
```

Figure 1. Top 20 unigrams



```
waste time
one worst
one best
worst movie
bad acting
bad movie
worst movies
really bad
movie bad
worst film
must see
well worth
looks like
highly recommended
even worse
bad film
highly recommend
bad movies
mst k
piece crap
```

Figure 2. Top 20 bigrams

reviews. After searching a bit we came with the concept of tf-idf, which solves precisely our problem. We found that this was actually a common approach for sentiment analysis [4] (although we couldn't find a paper were this was first proposed).

It works as follows, we want to compute the function $w(w, d)$ for every word w document d pair, this will give how important is the word for indexing the document. We have the two auxiliary functions $tf(w, d)$, which counts how important is the word for the document.

$$tf(w, d) = 1 + \log(\text{Number of occurrences of w in d})$$

This intuitively gives how important is that word in the document, but it doesn't take into account the possibility of a word belonging to every document (which will render it useless for indexing), and thus we need a balance term $idf(w, D)$, which is defined over the whole corpus D, and is given by

$$idf(w, D) = \log\left(\frac{|D|}{|\{d \in D : w \in D\}|}\right)$$

The more documents w is in, the lower it's idf score, which is what we want. So now, we get that

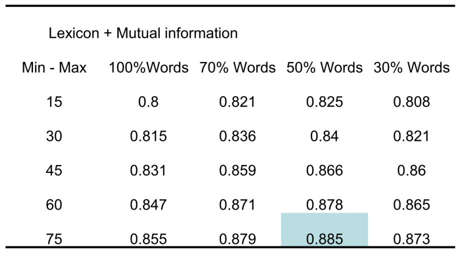
$$f(w, D) = tf(w, d) * idf(w, D)$$

And thus a word is important if it is frequent in the document, but if it is not to frequent among all other documents.

5. Lexicon classifier

The first approach, and maybe the most simple of them, is called the lexicon classifier. A lexicon is just an assignment of a score for each word, this score is positive for positive words, and negative for negative ones. We decided to build our own using the data, so the score of a word w, is just the number of positive reviews it appears in, minus the number of negative reviews it appears in.

Having built this, for a review r we get the following features: Average score, top k scores, bottom k scores. We applied logistic regressio to this features and got the results in table 8.



Lexicon + Mutual information

Min - Max	100%Words	70% Words	50% Words	30% Words
15	0.8	0.821	0.825	0.808
30	0.815	0.836	0.84	0.821
45	0.831	0.859	0.866	0.86
60	0.847	0.871	0.878	0.865
75	0.855	0.879	0.885	0.873

Figure 3. Lexicon classifier.

We see as expected that the bigger k is, the better. Also using the top 50% of the words sorted by mutual information words gave the best result. Unfortunately this approach had the biggest variance of all we tried, and when we submitted the results to Kaggle we got only a ROC score of 0.79972. A little better than we expected of this very simple approach.

6. Bayes classifier

We of course had to try a Bayes approach, to do this we estimated the following probability for every word w and sentiment y.

$$p(y|w) = \frac{p(w|y)p(y)}{p(w)}$$

All of $p(w|y)$, $p(y)$ and $p(w)$ can be estimated using the empirical distribution. Now Assuming independence of words in a review, we get that

$$p(y|r) = \prod_{w \in r} p(y|w)$$

And then the decision boundary becomes

$$\frac{p(y = 1|r)}{p(y = 0|r)} = \frac{p(y = 1)}{p(y = 0)} \prod_{w \in r} \frac{p(w|y = 1)}{p(w|y = 0)} = 1$$

Notice that here we don't require $p(w)$. The results are given in figure 4

Again, we can see that the best choice for the top mutual information words, was somewhere between 40% and 50%. And even though in the cross validation we got a lower score than the lexicon approach, on the actual Kaggle competition we got a ROC score of 0.86664, which is much better than we expected.

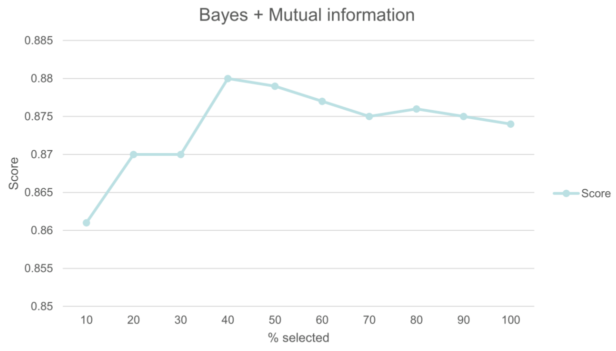


Figure 4. Binary Bayes classifier.

7. Bag-of-Words Model

The bag-of-words model is a representation of text used in natural language processing. This model represents a text as the set (bag) of its words and does not make use of the grammar or other linguistic structures that the words and sentences have.

Let's assume that after pre-processing of the data, we have N distinct words in the entire document. Also, suppose that the document contains R reviews. Each review, is represented by a N-dimensional feature vector. The entries of these vectors correspond to the N words in the document and the feature values reflect the frequency of their corresponding words in the review. Alternatively, we can construct a binary feature vector where 1 indicates the presence of the corresponding word in the review and zero represents otherwise.

However, we need to do "feature selection" in order to reduce the dimension of the data points, since the text document is very large and contains too many words. After pre-processing and cleaning the data, there are $N = 78,767$ words in the document. These words are sorted based on their importance which is measured using the mutual information criteria introduced in the previous sections and the top $d = 5000$ words are selected for constructing the feature vectors. Of course, larger number of the words would result in better accuracy results, however, we are limited by the memory size and the run-time of the algorithms.

Classification

In order to classify the reviews into positive and negative classes based on the bag of words model, we classify the feature vectors using three different supervised classification methods, namely, support vector machine, logistic regression, and random forests.

7.1. Support Vector Machines

Support vector machine (SVM) is one one of the most common models used for classification. In this method, we aim to find a separating hyperplane that maximizes the margin between the hyperplane and the data points and penalizes the points that do not satisfy the margin constraint.

Generally, an SVM classification problem can be written and solved in primal form or dual form. We solve the SVM problem using both methods and compare the empirical results for this problem. The dual form which was discussed in details in the class, makes it possible to use nonlinear kernels in order to find better classifiers for non-linearly separable data. However, it is not efficient when the size of data set is large because its complexity is $\mathcal{O}(n^3)$ where n is the size of the data set. However, solving the problem in primal form allows us to work with larger data sets in a more timely manner since its complexity is a function of dimension of the data d not n.

7.1.1 Built-in MATLAB SVM Package

The first method used for finding the SVM classifier is using MATLAB `fitcsvm` function which solves the dual form problem. Here, we train classifiers both with linear kernel and compare the results for the binary vector and the frequency vector.

The test accuracy results for these classifiers are shown in table 1.

Vector Type	Accuracy
Frequency Vector	0.79236
Binary Vector	0.79716

Table 1. SVM Accuracy Results Results for Bag of Words

7.1.2 Pegasos

This solver uses the basic stochastic gradient descent method for optimizing the primal objective functions of support vector machine (SVM) learning problems [5]. We remind that SVM problem with hinge loss function has the following form:

$$\min_{w\in\mathcal{R}^d} f(w) = \frac{\lambda}{2}\|w\|^2 + \frac{1}{m}\sum_{(x,y)\in S} \ell(w;(x,y)) \qquad (1)$$

where

$$\ell(w;(x,y)) = \max\{0, 1 - y\langle w, x\rangle\} \qquad (2)$$

Algorithm At every iteration, k samples out of m data points are chosen uniformly at random in order to find an unbiased estimate of the subgradient of the objective function $f(w)$. In other words, at each iteration the gradient of the following approximate objective function is computed:

$$f(w) = \frac{\lambda}{2}\|w\|^2 + \frac{1}{k}\sum_{(x,y)\in S_n} \max\{0, 1 - yw^T x\} \quad (3)$$

where $S_n \subset S$ with $|S_n| = k$. The computed gradient is the following:

$$\begin{aligned} g_n &= \nabla f_n(w_n) && (4)\\ &= \lambda w - \frac{1}{k}\sum_{(x,y)\in S_n} \mathbb{1}[yw^T x < 1]yx && (5)\\ &= \lambda w - \frac{1}{k}\sum_{(x,y)\in S_n^+} yx && (6) \end{aligned}$$

finally, a projection step is adopted to keep w inside the feasible domain $\mathcal{B}$ during the algorithm. The pseudo-code of this algorithm is the following:

```
Input $S, \lambda, N, k$
Initialize $w_1$ s.t. $\|w_1\| \le 1/\sqrt{\lambda}$
for $n$=1, 2, ..., $N$ do
    choose $S_n \subset S$ s.t. $|S_n| = k$
    set $\eta_n = \frac{1}{\lambda n}$
    set $w_{n+\frac{1}{2}} = (1 - \eta_n\lambda)w_n + \sum_{(x,y)\in S_n} yx$
    set $w_{n+1} = \min\left\{1, \frac{1/\sqrt{\lambda}}{\|w_{t+\frac{1}{2}}\|}\right\} w_{t+\frac{1}{2}}$
end
Output $w_{N+1}$
```

Algorithm 1: Pegasos

It can be shown that this method under some mild conditions has $\mathcal{O}(\frac{logN}{N})$ convergence rate.

Empirical results After cross validation, parameter λ is set to 0.1. The method is studied for different values of the subset size k, namely, $1, 1000$, and N. For the case of $k = 1$, one can see a very high variance in the results which can be partially addressed by mini-batching. Also, the performance deterministic (full batch) gradient descent is studied ny setting $k = N$. Note that training accuracy results and especially the variance is improved by using binary vectors instead of frequency vectors, however the test results are very bad for the binary case, which implies overfitting. The simulations were run for 10000 iterations.

7.2. Random Forest

Decision trees as classifiers have been discussed in the course lectures. However, they usually suffer from overfitting and high generalization error. In order to address this issue, many methods have been suggested in the literature including random forests.

Random forest method is an ensemble approach which makes use of the fact that a group of weak learners (trees) can form a strong learner if they come together.

In this method, b replicas of the training set are generated by randomly sampling N samples with replacement

Vector Type	K=1	k=1000	k=25000
Frequency Vector	0.73360	0.85024	0.83176
Binary Vector	0.53476	0.56708	0.56884

Table 2. Pegasos Test Accuracy Results for different subset sizes k Results for BoW

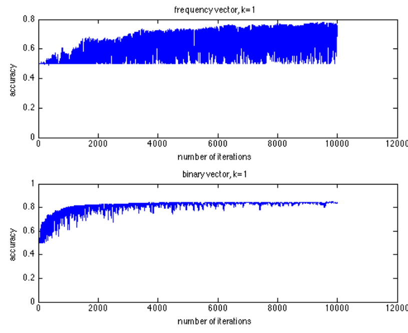


Figure 5. Pegasos with a single data point

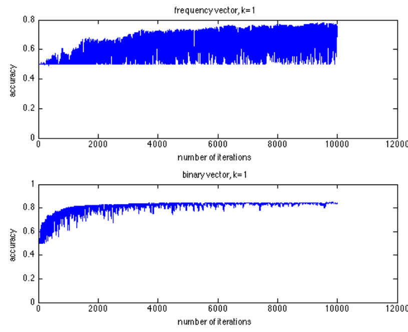


Figure 6. Pegasos with 1000 data point

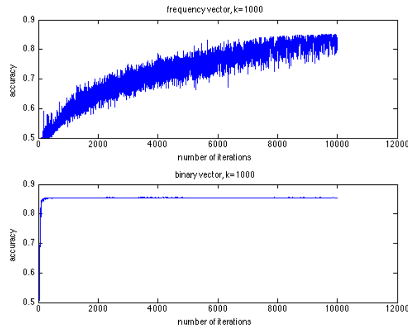


Figure 7. Pegasos with 1000 random data point

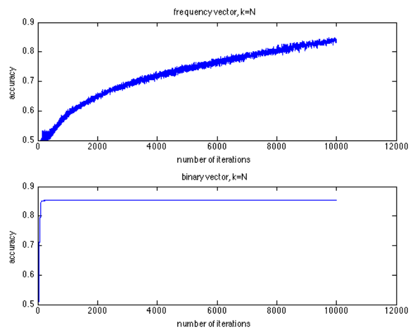


Figure 8. Pegasos with full batch (Deterministic)

from the original data set and each new data set is used for training a new decision tree.

For each decision tree, at every node a random subset of the features is selected and the best split on these m features (according to some objective function) is used to split the node and this procedure is repeated for the next node and so on.

Then, based on the average decision of the trees or the majority vote, the final decision on the label of a new data is made.

Here, we use the MATLAB `treebagger` function which uses $m = \sqrt{d}$ variables. About one third of the data is used as "out of bag" data to evaluate the performance of the trees. Cross validation results for $b = 100, 500$, and 1000 showed very small improvement in accuracy in case of using 1000 trees instead of 500 trees while it was much more time consuming, so we set $b = 500$.

The results for binary and frequency vectors are demonstrated in the following table.

Vector Type	Accuracy
Frequency Vector	0.84804
Binary Vector	0.50048

Table 3. Random Forest Accuracy Results for Bag of Words

8. Logistic regression

As mentioned in our presentation, we implemented the solvers for the logistic regression problem, namely, stochastic coordinate descent (SCD) and stochastic average gradient (SAG). The cross-validation results for the SCD method are disappointing (about 53%) which is probably because of high dimensionality of the data. Also, the SAG method deals with a very large gradient matrix that requires very large amount of memory, so we could not run our simulations for this method.

8.1. Tf-idf classifier

This can be seen as a modified bag of words approach, where instead of using raw frequencies to generate features, we use the tf-idf score to generate the features. Given any linear sorting of the reviews $r_1, \ldots, r_m$ and any linear ordering of the words $w_1, \ldots, w_n$, we can transform review r_i into vector v_i as follows

$$v_i = [f(w_1, r_i) \quad f(w_2, r_i) \ \ldots \ f(w_n, r_i)]$$

Were f is the tf-idf score of word w with repsect to review r. We fed this vectors to a logistic regression and bayes classifier and got the results given in figure 9

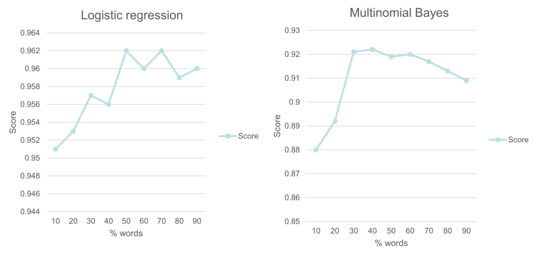
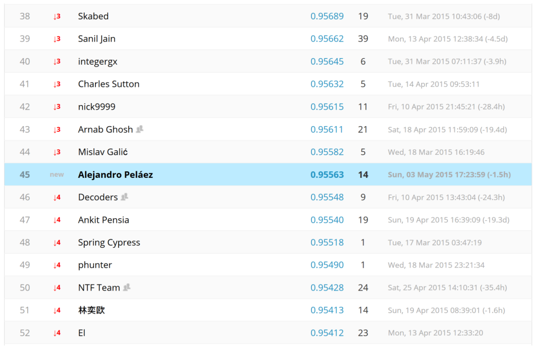


Figure 9. Logistic regression with tf-idf features.

The results are a incredible improvement over what we had before, and clearly logistic regression performed much better than the Bayes one. We got spot 45 using this approach as shown in figure 10



38	↓3	Skabed	0.95689	19	Tue, 31 Mar 2015 10:43:06 (-8d)
39	↓3	Sanil Jain	0.95662	39	Mon, 13 Apr 2015 12:38:34 (-4.5d)
40	↓3	integergx	0.95645	6	Tue, 31 Mar 2015 07:11:37 (-3.9h)
41	↓3	Charles Sutton	0.95632	5	Tue, 14 Apr 2015 09:53:11
42	↓3	nick9999	0.95615	11	Fri, 10 Apr 2015 21:45:21 (-28.4h)
43	↓3	Arnab Ghosh	0.95611	21	Sat, 18 Apr 2015 11:59:09 (-19.4d)
44	↓3	Mislav Galić	0.95582	5	Wed, 18 Mar 2015 16:19:46
45	new	**Alejandro Peláez**	**0.95563**	**14**	**Sun, 03 May 2015 17:23:59 (-1.5h)**
46	↓4	Decoders	0.95548	9	Fri, 10 Apr 2015 13:43:04 (-24.3h)
47	↓4	Ankit Pensia	0.95540	19	Sun, 19 Apr 2015 16:39:09 (-19.3d)
48	↓4	Spring Cypress	0.95518	1	Tue, 17 Mar 2015 03:47:19
49	↓4	phunter	0.95490	1	Wed, 18 Mar 2015 23:21:34
50	↓4	NTF Team	0.95428	24	Sat, 25 Apr 2015 14:10:31 (-35.4h)
51	↓4	林奕欧	0.95413	14	Sun, 19 Apr 2015 08:39:01 (-1.6h)
52	↓4	El	0.95412	23	Mon, 13 Apr 2015 12:33:20

Figure 10. Kaggle leaderboard.

9. Deep learning

Deep learning is an area of machine learning where multiple levels of abstractions are built on top of the data and then various machine learning tasks are performed by trying to recognize the pattern at a deeper level. Note that this is in contrast to the shallow learning approach where you learn a set of features from your data and then feed the set of features to a classifier. One can say that in deep learning you have multiple layers of feature extraction and the multiple levels form a hierarchy of concepts on top of the data;

whereas, in shallow learning you are usually working with a single layer of features. Some of the examples of deep learning approach are deep neural networks, convolutional deep neural networks, and deep belief networks.

9.1. Word2Vec

In recent research [1], it has been shown that the word embeddings in the hidden layer of the neural network actually capture some the semantic and syntactic regularities in the worlds. Words are semantically related if they frequently appear in the same context. Words are said to be syntactically related if they have the same syntax, for example words ending in the same way (-ies, -lly) can be said to be syntactically related. Such regularities indicate that word embeddings in the hidden layer of a neural network can be used to cluster similar words together which can be used as a handy tool for text classification. However, a problem is that the nonlinear feature extraction process in the hidden layers of a neural network makes the word embedding process computationally expensive. Thus, this puts a limit on the size of the corpus that can be used for training process. An idea is that instead of using the neural network, you learn the word embeddings using the first layer of the neural network and the resulting features can be used to train any classifier. Thus, you are not constrained to using a neural network for training purposes.

Word2vec is a recent package published by google that learns a distributed representations for words. Other deep or recurrent neural network architectures had been proposed for learning word representations prior to this, but the major problem with these architectures is the time required to train the models. If the network is given enough training data (tens of billions of words), it produces word vectors with intriguing characteristics. Words with similar meanings appear in clusters, and clusters are spaced such that some word relationships, such as analogies, can be reproduced using vector math. The famous example is that, with highly trained word vectors, "king - man + woman = queen."

The preprocessing process has already been explained earlier in the report. We use NLTK's punkt tokenizer for sentence splitting. One problem is that all reviews are of different lengths, and we need a way to ensure the feature extracted from each review is of the same length. One of the suggested ways on the Kaggle website is to find the average of the word vectors corresponding to all the words in a review. The average word vector of a review will represent the review in the feature space. We implemented this suggested word averaging procedure to get an accuracy of 83.4%. The code used to implement the procedure and the results (sentiment of each review in the test dataset) is attached.

However, an obvious problem with this procedure is that all the words in a review are used in the averaging process, ignoring the fact that words from some classes may be more important than words from some other classes. Words from classes like conjunctions and prepositions may not be relevant for predicting the sentiment of a review, so the presence of such words in the averaging process will act like noise in the sentiment estimation process. On the other hand, adjectives are more important than any other class of words for sentiment estimation. So, first we tried averaging the word vectors corresponding to the adjectives only in a review. However, a problem with this approach is that some of the reviews are short and thus may not contain any adjectives. We had many reviews in our training dataset without any adjectives. So, instead of averaging over adjectives only, we tried averaging over adjectives, verbs and adverbs and got an accuracy of 82.1%. A possible reason for this drop in the classification accuracy may be

Some future directions can be to train using words from more classes of words as it seems like the selected classes of words failed to capture the sentiment of reviews. Another approach is that instead of filtering out words of particular classes for the averaging process, you use the tf-idf algorithm to find the weightage of each word in a review in the averaging process.

9.2. Doc2Vec

Probably the most recent approach for sentiment analysis is given by the distributed representation of documents approach, which is a deep learning approach. Apparently only a preprint on arxiv is available [2]. This approach is an extension for the word2vec approach described here [3].

Basically it only modifies the word2vec in one way, when predicting a word w given it's context (neighoring words), that context is expanded to contain the whole paragraph. Again, every word is mapped to a vector, and every paragraph is mapped also to a vector, the classification is done with a neural network, and the vectors we are interested in are the intermediate weights of the network.

The model is defined as follows, given a sequence of words $w_1, \ldots, w_T$, and a matrix W containing the vector representations of the words and a matrix D containing the vector representations of the paragraphs, then the objective is to maximimize the average log probability

$$\frac{1}{T}\sum_{t=k}^{T-k} \log p(w_t|w_{t-k}, \ldots, w_{t+k})$$

And this is done usually by softmax where we have

$$p(w_t|w_{t-k}, \ldots, w_{t+k}) = \frac{\exp(yw_t)}{\sum \exp(y_i)}$$
$$y_i = b + Uh(w_{t-k}, \ldots, w_{t+k}; W, D)$$

Here b and U are parameters for the soft-max, h is a vector constructed using W and D, and W and D is what we are interested in.

We used *genism* implementation of doc2vec, and trained the model using vectors of length 400, and then fed them to a logistic regression algorithm. The results are given in 11.

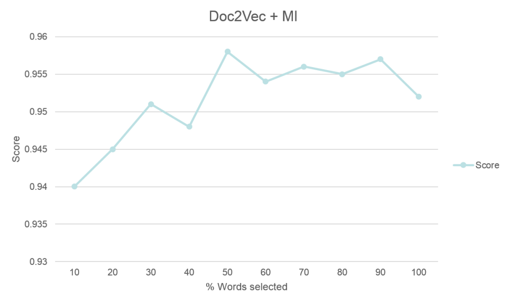


Figure 11. Doc2Vec classifier.

We can see the using around 50% wof the top mutual information words gave the bet results, but actually it didn't impact much, this was kind of expected, as the distributed vector representations already contains all semantic information. The Kaggle ROC score was 0.96170, yielding very good results, and achieving a little it more than what is claimed in the paper (probably due to the fact we are feeding the test data to build the representations).

We thought how we could merge our two best approaches so far, and we decided in the following, we stacked the vectors gotten for the tf-idf approach with the vectors gotten here. The hope was to be able to capture the semantic relationships with the doc2vec approach, and the structural relationships with the tf-idf approach. Again we used a logistic regression classifier and got the results showed in figure 12

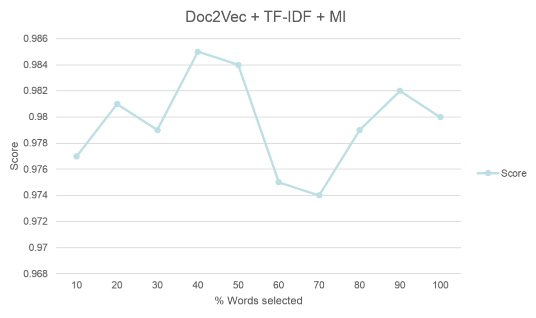
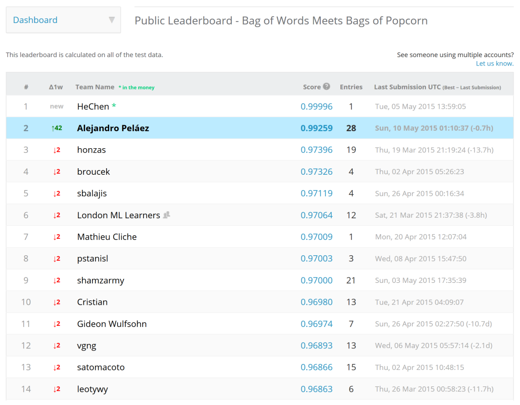


Figure 12. Doc2Vec + tf-idf classifier.

The results are definitely an improvement, and we can see that using around 40% of words sorted by mutual information seem to improve the results. Again, this is in part because we are using the test data to build bot the tf-idf vectors and to build the distributed document representation, as both approaches are unsupervised, and thus do not require labels (This is allowed in the competition rules, as long as they are used as unsupervised data, like we did).

This approach to our surprise had an ROC 0.99259 on the Kaggle competition, and we currently stand in 2nd place as shown in figure 13.


Dashboard

Public Leaderboard - Bag of Words Meets Bags of Popcorn

This leaderboard is calculated on all of the test data.

See someone using multiple accounts? Let us know.

#	Δ1w	Team Name * in the money	Score	Entries	Last Submission UTC (Best – Last Submission)
1	new	HeChen *	0.99996	1	Tue, 05 May 2015 13:59:05
2	↑42	**Alejandro Peláez**	**0.99259**	**28**	**Sun, 10 May 2015 01:10:37 (-0.7h)**
3	↓2	honzas	0.97396	19	Thu, 19 Mar 2015 21:19:24 (-13.7h)
4	↓2	broucek	0.97326	4	Thu, 02 Apr 2015 05:26:23
5	↓2	sbalajis	0.97119	4	Sun, 26 Apr 2015 00:16:34
6	↓2	London ML Learners	0.97064	12	Sat, 21 Mar 2015 21:37:38 (-3.8h)
7	↓2	Mathieu Cliche	0.97009	1	Mon, 20 Apr 2015 12:07:04
8	↓2	pstanisl	0.97003	3	Wed, 08 Apr 2015 15:47:50
9	↓2	shamzarmy	0.97000	21	Sun, 03 May 2015 17:35:39
10	↓2	Cristian	0.96980	13	Tue, 21 Apr 2015 04:09:07
11	↓2	Gideon Wulfsohn	0.96974	7	Sun, 26 Apr 2015 02:27:50 (-10.7d)
12	↓2	vgng	0.96893	13	Wed, 06 May 2015 05:57:14 (-2.1d)
13	↓2	satomacoto	0.96866	15	Thu, 02 Apr 2015 10:48:15
14	↓2	leotywy	0.96863	6	Thu, 26 Mar 2015 00:58:23 (-11.7h)

Figure 13. Kaggle leaderboard.

References

[1] R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM, 2008. 6

[2] Q. V. Le and T. Mikolov. Distributed representations of sentences and documents, 2014. 6

[3] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality, 2013. 6

[4] G. Paltoglou and M. Thelwall. A study of information retrieval weighting schemes for sentiment analysis. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, pages 1386–1395, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics. 2

[5] S. Shalev-Shwartz, Y. Singer, N. Srebro, and A. Cotter. Pegasos: Primal estimated sub-gradient solver for svm. *Mathematical programming*, 127(1):3–30, 2011. 3