Master of Computer Science

Machine Learning for Data Science

Tutored Project Report

Unsupervised Sentiment Analysis

by

Lucas Rodrigues Pereira and **Mathieu Pont**

directed by

Mickael Febrissy and **Lazhar Labiod**

Paris University, France

May 2019

Abstract

Unsupervised machine learning consists of grouping observations of a dataset into clusters (classes) based on its features. As the amount of data produced and available increases rapidly, so increases the difficulty to organize and learn insights from it. Sentiment analysis is a field of machine learning that aims to cluster a set of documents based on the predicted sentiment (positive/negative). The Scale Movie Review Data is a dataset by the Cornell University containing thousands of movie ratings and reviews for sentiment analysis. This article compares the results of unsupervised machine learning methods to perform sentiment analysis on movie reviews. In the end, some supervised machine learning methods are implemented in order to compare their accuracy as regards the unsupervised methods.

Contents

List of Figures

List of Tables

1 Introduction

Sentiment analysis is the process of extracting subjective information out of data, regardless of its format. In its simpler form, the sentiment can be binary (positive/negative, aggressive/friendly, ...). However, recent advances in artificial intelligence have been providing more overwhelming and complex results. This process can be either supervised (when there is a labeled dataset for training) or unsupervised, which is the focus of this article.

In a unsupervised machine learning scenario, sentiment analysis can tricky due to the difficulty to assess results (and the amount of possible approaches), and the interference of other subjective aspects. For instance, when comparing two different authors, the difference in their styles can be greater than the polarity difference of the sentiment. Therefore, the unsupervised approaches at times become semi-supervised, as they tend to incorporate external data into the analysis.

This paper will test and compare some approaches to the unsupervised sentiment analysis based on the **Scale Movie Review Dataset v1.0**. Although this dataset contains the labels for each review (the score to the movie by the critic), they have not been used as input to the clustering methods. These labels have only been used to assess the results and to compare each approach tested.

	Author	**Reviews**
1	Dennis Schwartz	1027
2	James Berardinelli	1307
3	Scott Renshaw	902
4	Steve Rhodes	1770
	Total	**5006**

Table 1: Number of reviews per author.

This dataset contains over 5 thousand movie reviews written by 4 different professional film critics: Dennis Schwartz, James Berardinelli, Scott Renshaw and Steve Rhodes. As described below, the distribution is not well balanced among reviewers. Important to notice that, even though not being the focus of the analysis, the dataset did not provide a name for each film reviewed. The operations are described below. All words were previously converted to lowercase.

	Author	**Vocabulary**
1	Dennis Schwartz	20053 (36.2%)
2	James Berardinelli	31764 (57.3%)
3	Scott Renshaw	26378 (47.6%)
4	Steve Rhodes	26868 (48.4%)
	Total	**55450** (100.0%)

Table 2: Number of words per author relative to the complete vocab before pre-processing.

Before pre-processing the data, each author uses only approximately 50% of the total number of words in the vocabulary. This matters, as the differences among authors' style can generate noise to the sentiment analysis. The pre-processing is expected to increase this percentage, as described in the following section.

2 Problem Formulation

The main goal in this research is to assess the available unsupervised methods' clustering of movie reviews. As the Movie Review Dataset has a rating (label) for each review (document), these labels can be used to measure the quality of the results.

The input model is an array of words and the output must be its sentiment polarity. In order to decide how many categories (or clusters) will be set as goal, we must beforehand look into the labels for the dataset. This will give us a glimpse of how difficult the clustering will be. In the original dataset, the ratings range from 0 to 1. The following histogram shows the frequency of reach rating.

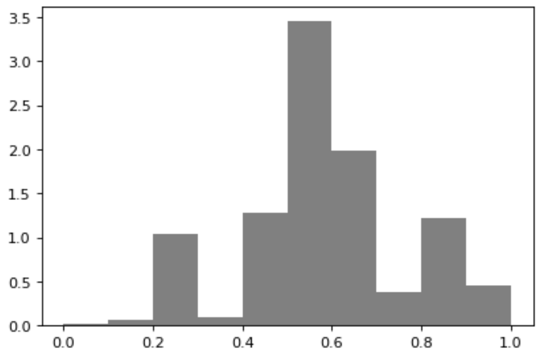


Figure 1: Distribution of ratings for the entire dataset.

The distribution is similar if we analyze it by author, as shown in the histograms below:

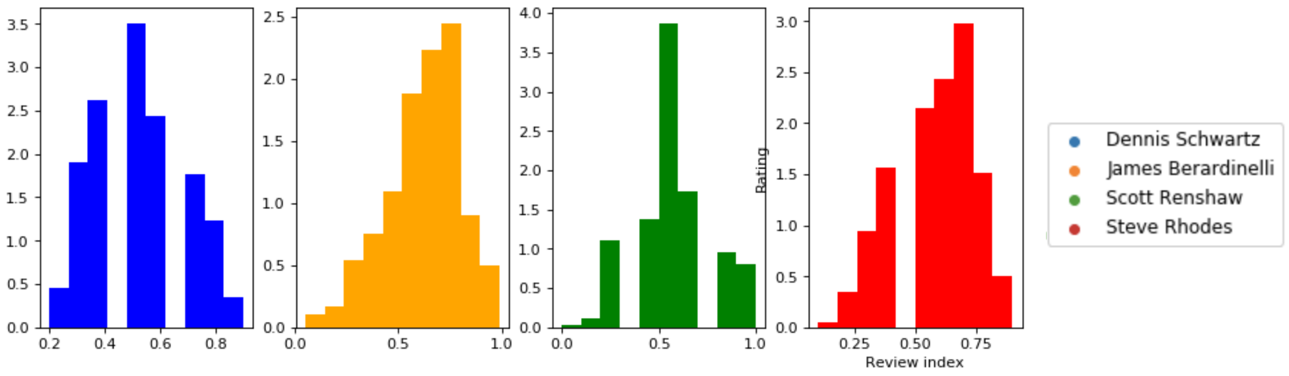


Figure 2: Ratings histogram by author.

This distribution shows that the sentiment for the reviews are more concentrated around 0.5. This implies that it should be very hard to distinguish between 0.4 and 0.8 rating points, range which contains over 60% of all ratings. Thus, further analysis has to be done in order to decide how many clusters the machine learning should aim for. On the one hand, the bigger the number of clusters, the harder will be for the model to classify each document according to the rating given originally. On the other hand, fewer clusters will tend to concentrate all documents around 0.5, which does not add much to the sentiment analysis.

The original rating for each document by every author is described in the three graphs below, considering 3, 4 and 5 different clusters.

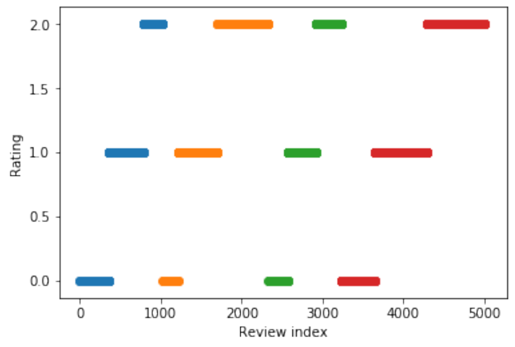


Figure 3: Goal clusters (three classes)

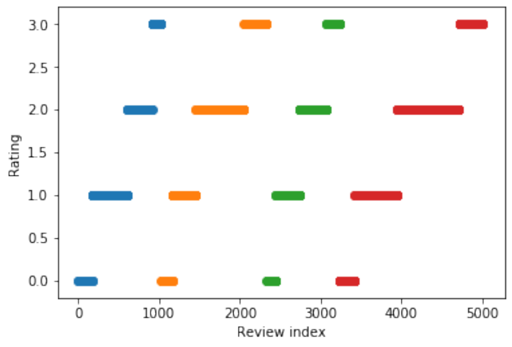


Figure 4: Goal clusters (four classes)

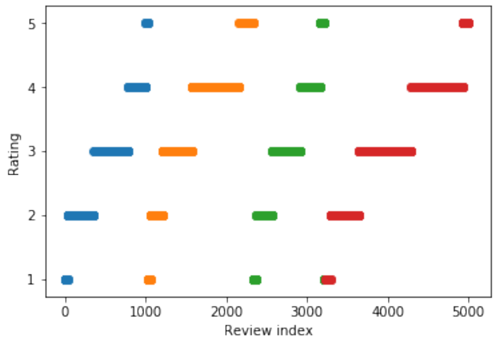


Figure 5: Goal clusters (five classes)

3 Methodology

3.1 Pre-processing documents

In order to achieve more accurate results, a robust pre-processing is required. The goal is to keep only the relevant data and remove all the noise. The operations are described below. All words were previously converted to lowercase.

Removing special characters. Special characters, such as punctuation and line-breaks, do not add information to the model, for they are very frequent and are contained in all documents regardless their sentiment. Therefore, all of them can be safely removed from the documents.

Numbers, infrequent words and stop-words. Following the same principle, numbers, infrequent and stop-words words do not add to the model. Stop-words are a list of english words that carry no meaning alone. As no method used retains the order of the words in sentences, they were also removed.

Lemmatization. Lemmatization consists of returning all words to its primitive form, thus reducing the total count of words. For example, the word 'to walk' can appear in different forms, such as 'walk', 'walked', 'walks', 'walking'. This is particularly important for the model to capture more correlation among words.

Finally, the vocabulary has shrunk to 9749 words. More importantly, the number of words that appears only in few specific author's reviews has decreased substantially, as demonstrated in the table below.

	Author	**Vocabulary**
1	Dennis Schwartz	8060(82.7%)
2	James Berardinelli	9374(96.2%)
3	Scott Renshaw	9049(92.8%)
4	Steve Rhodes	9127(93.6%)
	Total	**9749** (100.0%)

Table 3: Number of words per author relative to the complete vocabulary after pre-processing.

3.2 Encoding documents in vector format

The first step to do sentiment analysis is to encode all documents in a vector format that will be used as input for a classification algorithm for example.

Bag of words. One of the simplest encoding is the *bag of words*, it's simply an histogram of the words present in each document. The output is a matrix $\mathbf{X} \in \mathbb{R}_{+}^{n \times t}$ where n is the number of documents and t the total number of words within all documents. x_{ij} is the number of times the word corresponding to the column j appears in the document i. We call this matrix the document-word matrix.

TF-IDF score. Another method, based on the *bag of words*, is the *TF-IDF score* that stands for *Term Frequency and Inverse Document Frequency.* This score is computed for each word in each document. It gives us in output a matrix $\mathbf{X}$ with the same properties than with the *bag of words* except that we multiply the frequency of the word in the document by a factor depending on the frequency of the word across all the documents like in equation 1.

$$x_{ij} = tf_{ij} \times \log \frac{N}{df_j} \tag{1}$$

Where tf_{ij} is the frequency of word j in document i. N the total number of documents. df_j is the number of documents containing the word j. Multiplying the second factor to the frequency of the word allows us to give more importance to word that are rare across all documents. Indeed, words that appear frequently across documents do not add much information to the model and therefore do not contribute to clustering them. Finally we can apply L2 normalization to make each vector unit length, i.e. its norm equals 1.

Doc2Vec. Doc2Vec, an unsupervised machine learning method presented by Mikilov and Lee in 2014, is an extension of the Word2Vec, and cannot be explained independently. Therefore, the latter should be briefly discussed before actually tackling the embedding of documents into a vector.

In a nutshell, Word2Vec is the representation of words as vectors in space, encapsulating their relations to each other, such as synonyms, antonyms or analogies. This representation is created using either Continuous Bag-of-Words (CBOW) or Skip-Gram. Both methods have different approach to the same issue: they create search to predict the missing word given a context (usually called window). As the model is trained using a neural network, the weights of the hidden layer encapsulate the relation between the context and the output word.

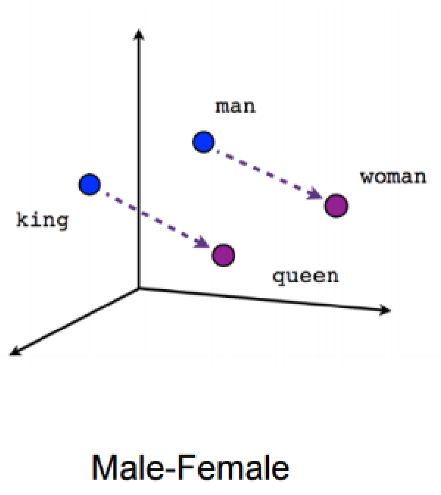


Figure 6: Example of relationship encapsulated in Word2Vec embedding.

As [Le et al., 2014] proposed, the Doc2Vec extends this method by adding an *id*, which is unique for each document, to be embedded along with the words. This gives each document a single feature vector regardless of its length (number of words in the document). This result can be used for clustering using traditional unsupervised methods, such as K-Means.

Density Matrix Representation. This method was first introduced by [Zhang et al., 2018] based on the work of [Gonçalves et al., 2013]. It uses quantum probability theory to represent a document as a density matrix. In quantum theory this matrix describes the state of a system, it can be understood as a probability distribution over the different possible states of the system.

A quantum state $\boldsymbol{u}$ is a vector and can be represented as a ket $|u\rangle$ and $\boldsymbol{u}^T$ as a bra $\langle u|$ (quantum bra-ket notation). Hence the density matrix is expressed as the following:

$$\rho = \sum_i \phi_i |u_i\rangle\langle u_i| \tag{2}$$

where ϕ_i is the probability to be in state $|u_i\rangle$.

The product $|u\rangle\langle u|$ is called an event and is represented by an orthogonal projector $\Pi = |u\rangle\langle u|$. Notice that $|u\rangle$ is a column vector therefore its corresponding event Π is a matrix.

Here, a document is seen as a system and each word of the document as a state. The quantum state vector of each word correspond to its one-hot encoding. We can then define the projector for each word and hence the sequence of projectors for each documents.

The density matrix of each document is estimated with the Globally convergence based Quantum Language Model (GQLM) algorithm presented in [Gonçalves et al., 2013]. They use Maximum Likelihood estimation to train density matrix with gradient descent by maximizing the product of the probability of each event (or projector).

3.3 K-Means and Spherical K-Means

The idea of these algorithms is to create k clusters where each of them gather documents by their similarities measured by a distance. K-Means uses Euclidean distance while Spherical K-Means uses cosine distance.

The main difference is that Euclidean distance will regroup data points having similar coordinates whereas cosine distance will regroup data points having their vector pointing in the same direction (it measures the cosine of the angle between the vectors).

These algorithms search to minimize the distance among documents inside each cluster (within-cluster inertia) and the center of this cluster (centroid). The goal is to minimize:

$$\sum_{i=1}^{k} \sum_{x \in S_i} d(x, p_i) \tag{3}$$

where S_i is the set of all the points corresponding to the cluster i and p_i the centroid.

For K-Means:

$$d(x, p) = \|x - p\|^2 \tag{4}$$

For Spherical K-Means:

$$d(x, p) = 1 - cos(x, p) = 1 - \frac{\langle x, p \rangle}{\|x\| \|p\|} \tag{5}$$

The algorithm consists of two steps:

- **assignment step**: each data point is assigned to its closest centroid based on the distance metric used.
- **update step**: the position of each centroid is updated to the center of the data points assigned to it.

3.4 Hierarchical Clustering

Hierarchical clustering algorithm is widely used and is very easy to understand. This iterative model follows a few basic steps: at the beginning, each data point is considered to be a different cluster. Then, the closest clusters merge. This process repeats until all data points have been merged into one single cluster.

There are many different approaches to measuring distance (similarity) between clusters, such as minimum, maximum, centroid linkage criteria, or Ward's method.

For exemple, using Euclidian distance ($\|c_k - c_t\|$), we merge each cluster with its closets neighbor. This is the Centroid linkage clustering.

3.5 Autoencoder

Autoencoders are a widely used method for dimentionality reduction. The main idea behind it is to use a feedforward neural network to compress the input to a lower dimension, then try to reconstruct the original input given the compressed input.

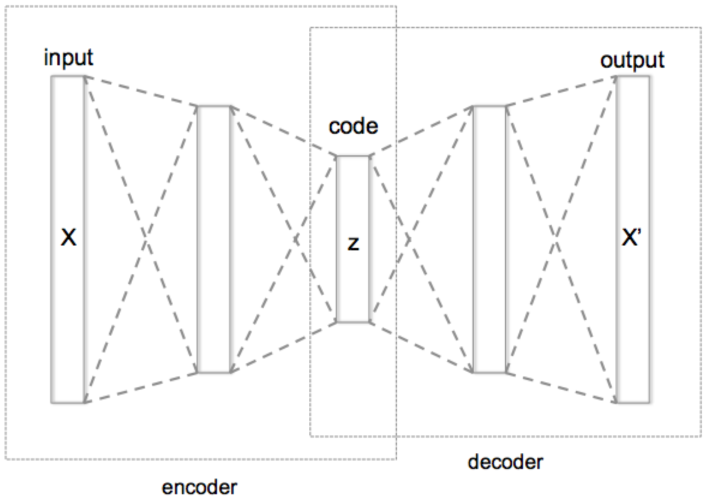


Figure 7: Schematic structure of an Autoencoder with 3 fully connected hidden layers. (Source: Wikipedia)

3.6 Word Co-Occurrence Non-Negative Matrix Tri-Factorization (WC-NMTF)

This method is based on Non-Negative Matrix Tri-Factorization (NMTF) [Long et al., 2005] and [Ding et al., 2006] which factorizes an input matrix $\mathbf{X} \in \mathbb{R}_+^{n \times t}$ into three matrices $\mathbf{Z} \in \mathbb{R}_+^{n \times g}$, $\mathbf{S} \in \mathbb{R}_+^{g \times m}$ and $\mathbf{W} \in \mathbb{R}_+^{t \times m}$ such that all matrices do not have any negative values and:

$$\mathbf{X} \approx \mathbf{ZSW}^T \tag{6}$$

$\mathbf{Z}$ plays the role of the cluster membership of each document if we have g clusters. The Word Co-Occurrence Non-Negative Matrix Tri-Factorization (WC-NMTF) [Salah et al., 2018] uses the same factorization but also factorizes another matrix $\mathbf{M} \in \mathbb{R}_+^{t \times t}$ into the matrices $\mathbf{W} \in \mathbb{R}_+^{t \times m}$ and $\mathbf{Q} \in \mathbb{R}_+^{t \times m}$ such that:

$$\mathbf{M} \approx \mathbf{WQ}^T \tag{7}$$

WC-NMTF factorizes both $\mathbf{X}$ and $\mathbf{M}$. Note that the same $\mathbf{W}$ matrix is used in equation 6 and 7.

In our case $\mathbf{X}$ is the document-word matrix. $\mathbf{M}$ is a word-word matrix, also called the context matrix. It is made from a co-occurrence matrix, which tells the number of times each word occur in the same context (i.e. document) than an other word. Then they use the Point-wise Mutual Information (PMI) on the co-occurrence matrix to make $\mathbf{M}$ (they actually use a variant of PMI, the Sparse Shifted Positive PMI or SPPMI).

The method minimizes the difference between $\mathbf{X}$ and $\mathbf{ZSW}^T$ and between $\mathbf{M}$ and $\mathbf{WQ}^T$, so we have to minimize the loss function:

$$F = \frac{1}{2}\|\mathbf{X} - \mathbf{ZSW}^T\|^2 + \frac{\lambda}{2}\|\mathbf{M} - \mathbf{WQ}^T\|^2 \tag{8}$$

λ is the regularization parameter. When $\lambda = 0$ in equation 8 we get back to the original NMTF.

After rewrite and derive equation 8 they arrived at four update rules, one for each matrix (**Z**, **W**, **S**, **Q**). Then we just need to iterate and apply these update rules many times to minimize the loss function.

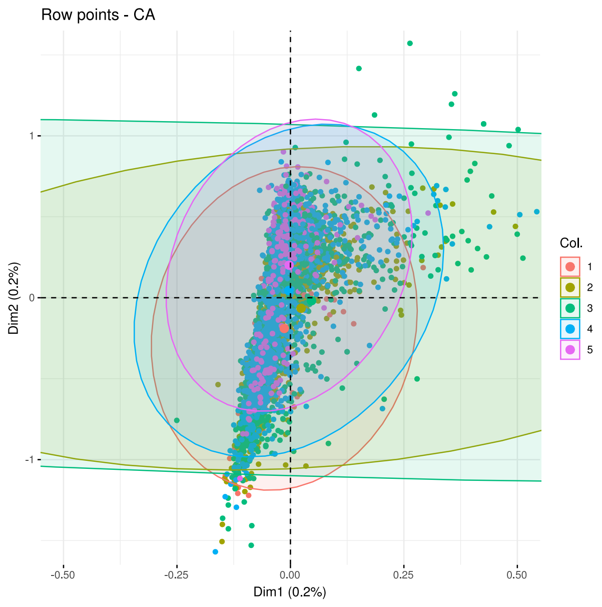
4 Experiments and Results

Here we use many unsupervised algorithms trying to find the correct class of each document. For our first experiments we used the *TF-IDF* matrix with L2 normalization.

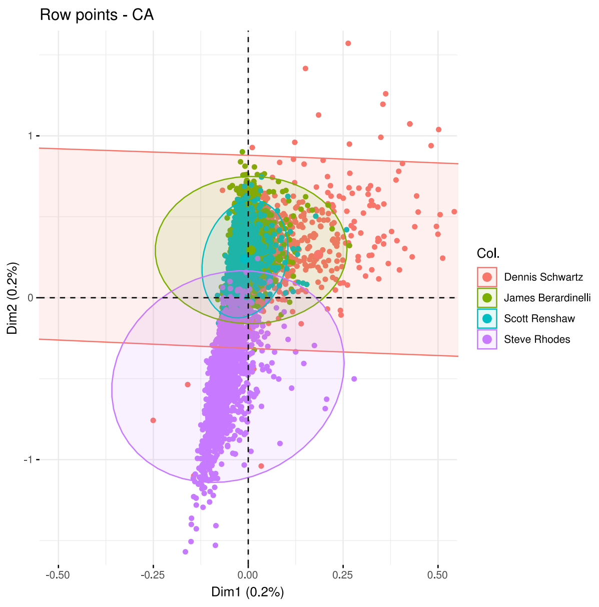
To evaluate the clustering of each algorithm we use as metrics the Normalized Mutual Information (NMI) and Adjusted Rand Index (ARI) between the true distribution and the one found by the algorithms.

4.1 Correspondence Analysis

At first we have done a correspondence analysis to plot our data in 2 dimensions, to see if we can already found some clusters just by looking at them in a visuable way.



(a) Rating clusters.



(b) Authors clusters.

Figure 8: Correspondence analysis.

We see in figure 8a that the clusters overlap each other. We must notice the variance explained by the first two components is very low (0.4%).

In figure 8b we see that even if clusters seems mingled they are less than with the rating. Indeed the reviews from Steve Rhodes are more or less quite separated from the other ones. However, the cluster of the reviews of Scott Renshaw seems ”inside” the one of James Berardinelli.

4.2 K-Means and Spherical K-Means

Elbow criterion. To find the value of k (number of clusters) we can use the elbow criterion. It implies to compute the algorithm for many values of k and measure the percentage of variance explained by the clusters. For each k we need to run the algorithm many times because the result depends on the initialization of the clusters which changes for each execution.

For a certain k adding another cluster will not really help us in the data modeling, it will not add much to the variance explained. The behaviour expected is to see the first clusters adding a lot to the variance explained then the gain for each cluster added will drop and ideally we should see an angle in the graph.

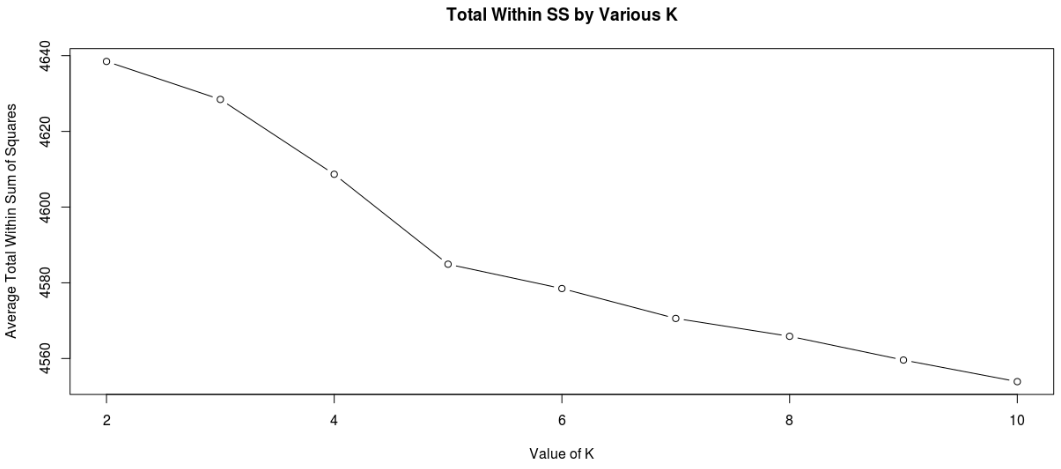


Figure 9: Elbow criterion for K-Means algorithm.

Ac figure 9 the elbow criterion for K-Means with the total within sum of squares as a measure. We can easily see an angle in the curve for the value $k = 5$.

Documents clustering. Then we use the K-Means and Spherical K-Means to separate our documents in clusters.

We see at the top of figure 10 the real clusters, then the clusters found by K-Means and finally those found by Spherical K-Means. In the real clusters graph we can see four ”staircase” composed of five ”steps” each. Each staircase correspond to an author and each step to the reviews having a same rating.

The clusters found by K-Means and Spherical K-Means are more or less the same. They both regroup the reviews of a same author in a same cluster. Instead of partitioning the documents according to their rating they were partitioned depending on their author.

This result can be explained by the fact that the writing styles within authors have more in common than the writing styles for different authors giving the same rating. In other words all the reviews of an author are more similar than the reviews having the same rating (according to the Euclidean and cosinus distance).

Moreover, it corroborates with the correspondence analysis result where we saw that the clusters of the authors were more separated than the ones of the rating.

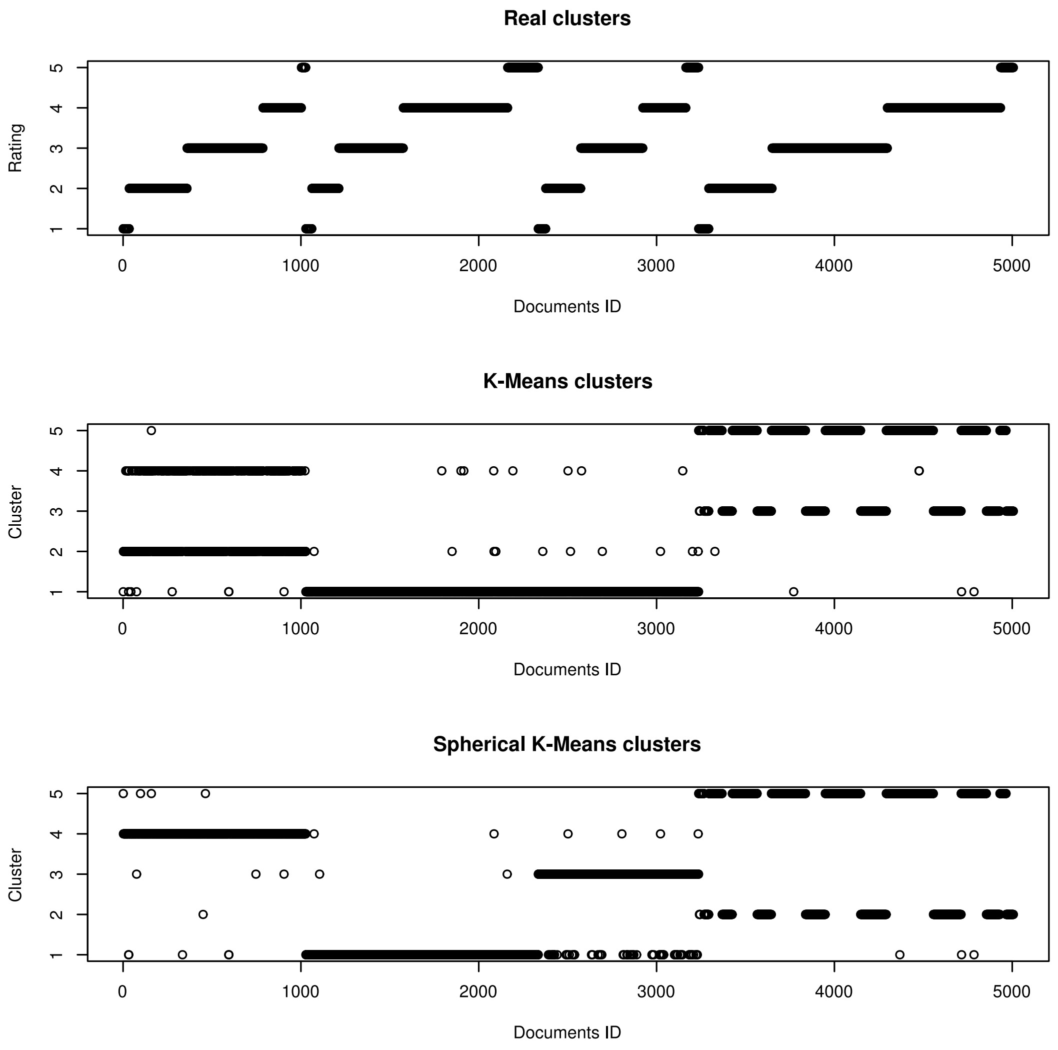


Figure 10: Cluster of each document with K-Means and Spherical K-Means.

We must still notice that Spherical K-Means discriminate better authors than (standard) K-Means, especially for the author 2 and 3. K-Means regroup the reviews of these authors in the same cluster whereas Spherical K-Means manages to separate them.

It make sense since the cosine distance is more adapted for textual data than the Euclidean distance. Indeed the cosine distance will be small for documents containing the same words, no matter if one of this word appears more times in a document than in the other one. If documents contain the same words (i.e. talk about more a less the same thing) but not in the same proportion the Euclidean distance will be high despite the fact that they talk about the same thing.

	NMI	ARI
K-Means	0.020	0.004
Spherical K-Means	0.026	0.020

Table 4: NMI and ARI for K-Means and Spherical K-Means.

We have also tried to increase the number of clusters to see if the algorithms can find by this way the rating. We have tried 10 and 50 clusters, however for both values the author was found instead of the rating.

4.3 Hierarchical Clustering

This algorithm seems to also find the authors as clusters but in a more noisy way than K-Means and Spherical K-Means like show in figure 11.

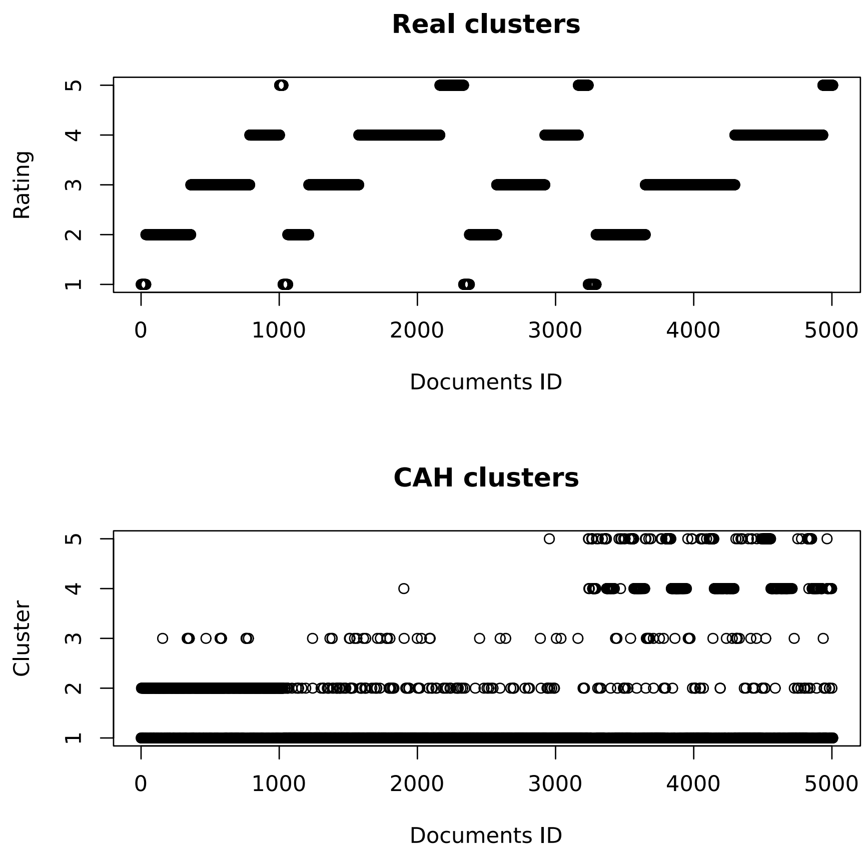


Figure 11: Cluster of each document with Hierarchical Clustering.

The cluster number 4 and 5 regroup reviews from the fourth author. The cluster 2 seems to regroup reviews from the first author even if it have some reviews from the other authors.

Hierarchical Clustering	NMI	ARI
Euclidean distance	0.0081	-0.0007
Cosine distance	0.0045	0.0024

Table 5: NMI and ARI for Hierarchical Clustering.

We saw that the NMI and ARI are very low compared to those of the K-Means and Spherical K-Means (almost 10 times lower).

4.4 Autoencoder

We try to use a Deep Autoencoder to encode the 9749 dimensions vector (a document) to a 5 dimensions vector (one for each cluster) through many layers then we decode the 5 dimensions vector to reconstruct the original 9749 dimensions vector. The results were very noisy, which it make sense, since reconstruct the 9749 dimensions vector from the 5 dimensions vector should be complicated because the information is too much compressed.

Another way could be to encode document in a larger vector (let say 50 dimensions for example) and then use the encoded vector as input to another algorithm like K-Means or Spherical K-Means.

4.5 Other ways to represent documents

We also try to represent the documents with other methods such as Doc2Vec [Le et al., 2014] and the Density Matrix Representation [Zhang et al., 2018] and then re-run the unsupervised algorithms on the dataset produced by these methods.

K-Means and Spherical K-Means produced the same results than with the *TF-IDF* matrix with L2 normalization, the clusters corresponded to the authors of the documents was found instead of the rating.

	NMI	ARI
K-Means	0.046	0.036
Spherical K-Means	0.082	0.059

Table 6: NMI and ARI with Doc2Vec representation.

The Doc2Vec representation improve the evaluation metrics for both algorithms even if this representation doesn't find the rating. This can show us that the Doc2Vec representation seems to be better than the TF-IDF one.

	NMI	ARI
K-Means	0.023	0.016
Spherical K-Means	0.026	0.020

Table 7: NMI and ARI with Density Matrix representation.

We've used a variant of the GQLM, instead of only use the term frequency for the computation we used the *TF-IDF* score normalized with L2 since this score seems to be better than the simple term frequency score.

The Density Matrix representation (computed with the TF-IDF score and the one-hot encoding) seems to not enhance the result compared to the TF-IDF representation for K-Means and Spherical K-Means. However we still notice a slight improvement for the K-Means metrics.

4.6 Use of external information: SentiWordNet

After these results we decided to add external information to help algorithms to correctly extract the sentiment of the reviews.

We found SentiWordNet [Baccianella et al., 2010], this is a dictionary composed of a lot of words (more than 150 000) and each of them is associated with two scores, each of them indicates how much the word is positive or negative. SentiWordNet founds 78% (7604 in 9749 words) of the vocabulary of our documents.

Then, with the help of the SentiWordNet dictionary, we made the matrix $\mathbf{S} \in \mathbb{R}_+^{t \times 3}$ with t the total number of words within all documents. Each word is associated with 3 scores, how much is a positive, negative and neutral word. A word is neutral if it doesn't have a positive and a negative score (they are equal to 0) or if it's not found in the dictionary. We call this matrix the word-sentiment matrix.

A word in SentiWordNet can be positive but also negative in the same time (for example the word "wrong" as a positive score of 0.125 and a negative one of 0.75). So we have two possibilities, either keep both scores as they are in the dictionary or put the higher value at 1 and the lower at 0 (with this method the word "wrong" would have a positive score of 0 and a negative one of 1). Thus we have two different $\mathbf{S}$ matrices.

Another thing to consider is that a word can have multiple meanings depending on the context and therefore has different sets of value for the positive and negative score in the dictionary. We took the maximum between all its positive scores and same with the negative ones. A better way to handle this would be to see manually the different contexts of each word and take the one that is the most frequent in our every day life.

Lexicon-based sentiment analysis. We tried a lexicon-based approach with this dictionary by finding the global sentiment of a text through the sentiment of each word constituting it.

Basically, we just sum the sentiment of each word of the text to find its sentiment. Therefore, with $\mathbf{S}$ we can make another matrix $\mathbf{T} \in \mathbb{R}_+^{n \times 3}$ (with n the number of documents).

We tried to run K-Means and Spherical K-Means on $\mathbf{T}$, but the clusters were also corresponding to the authors but in a more noisy way.

4.7 Word Co-Occurrence Non-Negative Matrix Tri-Factorization (WC-NMTF)

As said above WC-NMTF needs two matrices as input. $\mathbf{X}$ the document-word matrix and $\mathbf{M}$ a word-word matrix.

Like explained in section 3.6 in [Salah et al., 2018] they use the SPPMI on the co-occurence matrix in order to make $\mathbf{M}$. Here we try two variants by using the word-sentiment matrix of the preceding part in order to add information about the sentiment of each word.

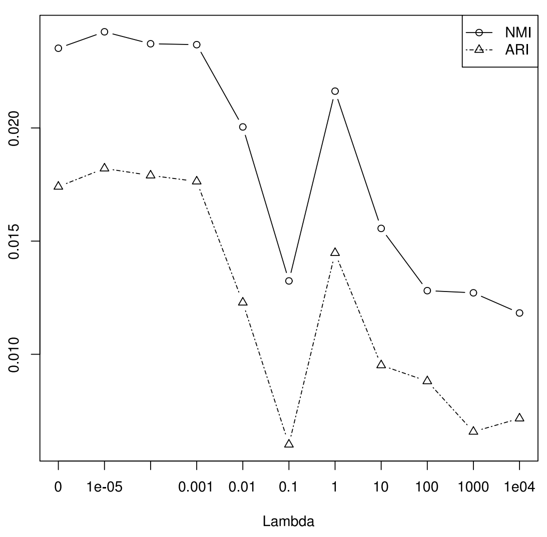
These two variants are based on the fact that we want to represent $\mathbf{M}$ such that m_{ij} tells us how much the word i and j share the same sentiment. We have the purpose to help the algorithm

According to the matrix product, multiplying $\mathbf{S}$ by its transpose gives us this property. Hence: $\mathbf{M} = \mathbf{S}\mathbf{S}^T$.

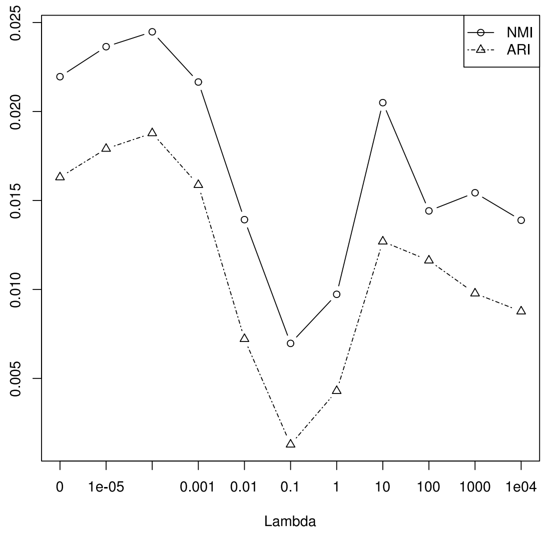
Another way is to compute the cosine similarity between the sentiments of two words, so $m_{ij} = cos(\mathbf{s}_i, \mathbf{s}_j)$ where:

$$cos(A, B) = \frac{AB}{\|A\|\|B\|} \tag{9}$$

To see how much the information added by $\mathbf{M}$ help in the clustering we computed the WC-NMTF for many value of λ. We perform 10 runs for each value because the initialization can affect the result of the algorithm.



(a) WC-NMTF with $\mathbf{M} = \mathbf{S}\mathbf{S}^T$.



(b) WC-NMTF with $\mathbf{M}$ made with cosine similarity.

Figure 12: NMI and ARI for WC-NMTF with both ways to make $\mathbf{M}$ matrix.

The information added with the sentiments (found by SentiWordNet) seems to not increase the NMI and ARI very much. It even looks like adding too much information (i.e. increasing lambda) decrease the metrics, for example for $\lambda = 0.1$ we can see a huge drop in the metrics.

This is a quite surprising result hence the sentiments should have helped the algorithm to better find the rating of each review. It tells us that the sentiment of each word is not as important as we thought to be to find the rating of each review. Otherwise, maybe the way WC-NMTF uses this external information it's the wrong one.

Compared to Spherical K-Means the NMI and ARI values are more or less the same (by taking the highest values). Therefore, the addition of external information did not help with WC-NMTF.

4.8 Supervised Learning

Finally, due to the preceding results we can say that the dataset seems to not be well suitable for unsupervised learning. Therefore, we've decided to use supervised algorithms to see if they can handle this dataset. We've used the TF-IDF representation as input for the different following algorithms.

4.8.1 Multiple Linear Regression (MLR)

To stay in our context of classifying 5 classes but to also use a regression algorithm we make the regression model using the raw (continuous) rating (from 0 to 1) then convert it to 5 classes (value from 0 to 0.2 is converted to class 1, 0.2 to 0.4 to class 2 and so on). It allows us to compare the performance of the model with the previous algorithms.

	0	1	2	3	4	**Mean**
Accuracy	0.52	0.52	0.51	0.48	0.49	**0.50**
NMI	0.19	0.21	0.18	0.19	0.19	**0.19**
ARI	0.13	0.16	0.12	0.13	0.13	**0.13**

Table 8: Metrics with MLR for each fold with k-fold cross-validation.

We see in table 8 that the metrics are much higher than with the preceding algorithms. However, the metrics are still not very high. The accuracy of 50% tells us that one in two samples are misclassified, it's not very good but still better than a random classifier having an accuracy of 20% (because we work with 5 classes).

Since our dataset is higly unbalanced we've computed the confusion matrix for each fold to have a better idea of the classification.

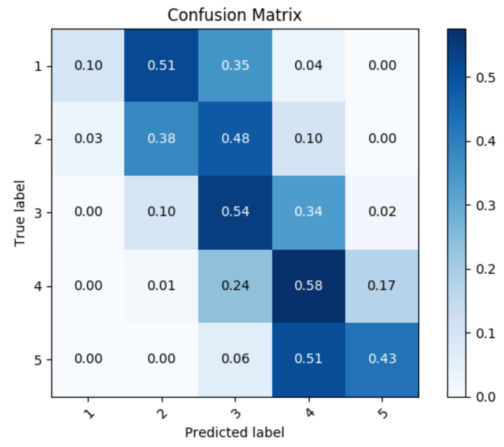


Figure 13: Normalized confusion matrix heat map with MLR.

	1	2	3	4	5
1	**16**	86	59	6	0
2	31	**394**	498	106	1
3	6	175	**971**	599	35
4	0	22	398	**968**	294
5	0	0	21	174	**146**

Table 9: Confusion matrix with MLR.

	1	2	3	4	5	**Mean**
F1	0.15	0.46	0.52	0.55	0.35	**0.41**

Table 10: F1 score with MLR.

In table 9 we see the confusion matrix for the classification made by MLR. Actually, it is the sum of the 5 confusion matrices (one for each fold), as for table 10, it is the mean of the F1 score of each fold for each class.

Even with supervised learning the ratings are hard to find according to MLR. The confusion matrix tells us that MLR seems to have difficulty to differentiate reviews with close rating, for example, many reviews with a rating of 3 is classified as 4 and conversely.

MLR failed to classify correctly review with a rating of 1 by assigning to more than 80% of them the class 2 or 3.

This can be explained by the fact that reviews with close rating are quite similar and therefore hard to discriminate. Moreover, according to the confusion matrix, reviews with distant rating are easier to differentiate since they are probably not similar.

4.8.2 Linear Discriminant Analysis (LDA)

	0	1	2	3	4	**Mean**
Accuracy	0.55	0.58	0.56	0.58	0.58	**0.57**
NMI	0.22	0.24	0.21	0.22	0.23	**0.23**
ARI	0.17	0.21	0.18	0.20	0.20	**0.19**

Table 11: Metrics with LDA for each fold with k-fold cross-validation.

We see that the NMI and ARI metrics with LDA are 10 times higher than those with K-Means or Spherical K-Means. They are also a little higher than those with MLR.

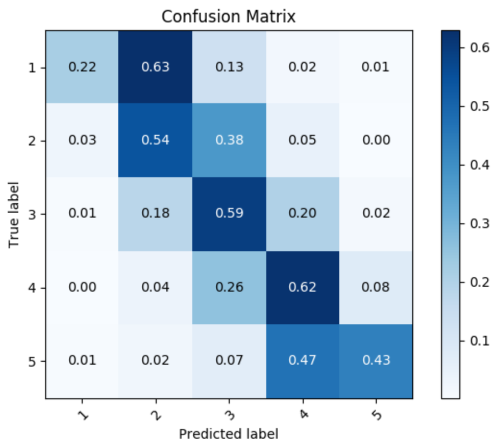


Figure 14: Normalized confusion matrix heat map with LDA.

	1	2	3	4	5
1	**36**	105	22	3	1
2	32	**558**	388	48	4
3	11	324	**1061**	359	30
4	4	62	436	**1045**	135
5	2	8	23	160	**148**

Table 12: Confusion matrix with LDA.

	1	2	3	4	5	**Mean**
F1	0.28	0.53	0.57	0.63	0.44	**0.49**

Table 13: F1 score with LDA.

LDA seems to suffer the same problem than MLR, both struggle to differentiate correctly reviews with close rating. But it is less flagrant with LDA, for example the majority of class 2 is correctly classified unlike MLR. The F1 score of class 1 (in table 13) shows us that the classification of this class is way better than with MLR whereas the other class have more or less the same score (even if LDA have always a higher score).

However, according to table 14, LDA have better performance than MLR.

	Accuracy	NMI	ARI	F1
MLR	0.50	0.19	0.13	0.41
LDA	**0.57**	**0.23**	**0.19**	**0.49**

Table 14: Metrics comparison between MLR and LDA.

5 Conclusion and future work

5.1 Conclusion

The Scale Movie Review Dataset has some characteristics that makes their documents hard to cluster. Unbalanced classes, style difference between authors, number of reviews (5006) are among the peculiarities of the dataset.

The different unsupervised methods grouped the reviews by their authors and not by their rating. Therefore, we can say that the writing style of each author is predominant on the one of each rating.

Another thing to take into account, that was already noticed in [Pang et al., 2005], is that the notation of each author could not represent the same thing. For example a rating of 2 for an author could not express the same feeling than the rating of 2 of another author. So it can make harder the clustering if the same rating can express different things.

Therefore, we have also tried to run different algorithms on only reviews from one author at a time. The first results was a little bit more convincing. Some clusters were more associated with reviews with lower rating and other clusters with reviews with high rating. However it was very noisy and not still very discriminant for each rating but better than with the clustering on the whole dataset. It makes sense since the writing style of each author no longer influence the result.

We must notice that we have tried without the pre-processing step to see if it would have remove too much information. We've also tried to cluster not only in 5 classes, but also in 3 and 4 classes, and even 2 (just positive or negative). In all cases the authors were found instead of the rating.

Moreover, even the supervised algorithms used had trouble to correctly classify the reviews.

Finally, our github repository is open source and free to use[1].

5.2 Future Work

Some other unsupervised algorithms could be used (like Self-Organizing Map) to see if those used was not the right ones for our problem.

The Doc2Vec and the Density Matrix representation did not work well either. An interesting possible variant in the computation of the latter would be to use Word2Vec to encode each word instead of the one-hot encoding.

Due to the poor results, maybe it is not the representaThe Scale Movie Review Dataset has some characteristics that makes their documents hard to cluster. Unbalanced classes, style difference between authors, number of reviews (5006) are among the peculiarities of the dataset.

[1]https://github.com/lucarp/ter-sentiment-analysis

tion that is lacking but the dataset itself. Therefore, the addition of information like we did with SentiWordNet should be a good path to explore.

It would have been also interesting to use transfer learning to see if our dataset could be correctly classified by a model having learned on another dataset.

References

Baccianella et al., 2010 - Stefano Baccianella, Andra Esuli,and Fabrizio Sebastiani. SENTI-WORDNET 3.0: An Enhanced Lexical Resource for Sentiment Analysis and Opinion Mining. 2010.

Ding et al., 2006 - Chris Ding, Tao Li, Wei Peng Haesun Park. Orthogonal Nonnegative Matrix Tri-Factorizations for Clustering. 2006.

Gonçalves et al., 2013 - D. S. Gonçalves, M. A. Gomes-Ruggiero and C. Lavor. Global convergence of diluted iterations in maximum-likelihood quantum tomography. 2013.

Hornik et al., 2012 - Kurt Hornik, Ingo Feinerer, Martin Kober and Christian Buchta. Spherical k-Means Clustering. 2012.

Le et al., 2014 - Quoc V. Le and Tomas Mikolov. Distributed Representations of Sentences and Documents. 2014.

Long et al., 2005 - Bo Long, Zhongfei (Mark) Zhang and Philip S. Yu. Co-clustering by Block Value Decomposition. 2005.

Pang et al., 2005 - Bo Pang and Lillian Lee. Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales. 2005.

Salah et al., 2018 - Aghiles Salah, Melissa Ailem and Mohamed Nadif. Word Co-Occurrence Regularized Non-Negative Matrix Tri-Factorization for Text Data Co-Clustering. 2018.

Zhang et al., 2018 - Yazhou Zhang, Dawei Song, Xiang Li and Peng Zhang. Unsupervised Sentiment Analysis of Twitter Posts Using Density Matrix Representation. 2018.