University of Paris Sud, France
Master 2 Machine learning, Information and Content
Search and retrieval of information in texts

Title

Semi-supervised recursive deep auto-encoders for Sentiment prediction

Supervised by : M. Xavier Tannier and M. Thomas Lavergne
Presented by :

M. Ahmed Mazari
Ms. Miha Sorostinean
Mr. Hafed Rhouma

Promotion 2016/2017

Abstract

Recently, Deep learning has enjoyed remarkable success in Natural Processing Language , establishing new state-of-the-art performance in language modeling, machine translation, document classification, and sentiment information retrieval. This breakthrough in artificial intelligence is associated with the availability of massive amount datasets and the high computing power. In this study, we address the problem of sentiment prediction with zero prior knowledge using a state-of-the-art deep learning techniques : autoencoders and denoising autoencoders. We first survey the latest deep learning models for learning semantic representation. Then, we develop a theoretical and experimental study of these techniques that cover sentiment analysis and semantic representation.

Keywords : manifold learning, computational semantic, learning representation, Dimensionality reduction, autoencoders, cross-entropy, semi-supervised learning.

Contents

List of Figures

1 General Introduction

Information retrieval (IR) is the activity of obtaining information resources relevant to an information need from a collection of information resources. The collection is generally a set of documents from a single or several databases, described by their content or the metadata associated. The first part will describe the search engine that we implemented which is capable of quering strings or words provided by the user in a collection of text files.
The explosion of amount of data through the 21st century paved the way to new technics and models to extract relevant information from them. It applies to sentiment analysis field which refers to the use of natural language processing, text analysis and computational linguistics to identify and extract subjective information in source materials. Sentiment analysis is widely applied to reviews and social media for a variety of applications, ranging from marketing to customer services.
Autoencoders have been introduced to solve the aformentionned tasks. They learn a representation (encoding) for a set of data, typically for the purpose of dimensionality reduction with the aim to facilitate the learning task. We will present them more in detail in the second part and present how they are used to learn sentiments such as positive or negative reviews on movies [16] based on [2].
Nevertheless, works by the scientific community for autoencoders improvement are still running to get better performances, this is how stacked denoising autoencoders appear by introducing some noise to the data in order to make the learning task stronger.
The Goal of this report is to present some tools and state of the art methods about information retrieval and computational linguistic for new students in machine learning.

2 Search engine

2.1 Introduction

One of the tasks that scientists working on text processing are often faced with, is building search engines that satisfy the needs of the users. Due to the complexity of a natural language, as well as the complexity of defining some metrics that can be used to evaluate the satisfaction of an user with respect to the results of a search engine, the task of building a pertinent search engine is not an easy one. In the first part of our project we implemented a system capable of searching in a collection of text documents, the ones that where relevant for a search string provided by the user. We then performed an evaluation of our system with respect to the pertinence of the search as well as an evaluation of its performance.
Section 1 of this chapter provides a brief description of the proposed architecture, Section 2 details the evaluation of the pertinence of our system to a set of search strings while Section 3 provides the details of the performance evaluation in terms of speed of processing and memory space used. Finally, a conclusion about the challenges we were faced with, when building such a system, is provided.

2.2 Architecture of the system

Step1. Parsing the subindex

The subindex directory contains the XML files which describe the text files, so the first step of our implementation consists in parsing each file from the subindex and save the id (which is later used to build the path to each text file) and the title of the file. For reading and parsing each XML file we utilize the class *ReadXMLFile.java*. Since the name and path to each text file is big and will impact the total size of our index, we chose to assign an integer id to each file. The structure *TreeMap<Integer, ArrayList<String»*, makes the mapping between an Integer id for a file and an ArrayList of type String which contains two strings:

- The path to the text file
- The title of the article

Step2. Build the index

In this step we read all the text files and build the index file containing all the words from these files, using one of the indexation methods provided as parameter when running the program. Two indexation methods are implemented:

- Indexation through simple segmentation
- Indexation through stemming

If the program is run with the option build_index then this step is performed. The method to create the index is relatively simple, we parse all the text files and build a structure *TreeMap<String, TreeSet<Integer>*, where the key is a word and the value is a set of integers representing the files ids we assigned before. Finally this index is saved on a file, with each word on a different row. If the program is run with the option use_index then this step consists only in reading the index form the text file and saving it in a data structure.

Step3. Perform the search

In this step we use the index that we previously built in order to perform the search of pertinent files for the expression provided as the third input argument of the program. First, the same indexation method which was used on the text files is applied to the search string as well. Then, an intersection of all the files containing the words from our search string is computed. If the result is an empty set (there is no file which contains all the words in the search string) the search ends and a text is displayed indicating that the search was not successful. If the intersection set is not empty, the weight for all the words in the files is computed and stored. Then, the distance between the search string and each file is computed and stored. Finally, the set of files is ordered based on the their distance to the search string, so as to select the most pertinent documents for our search.

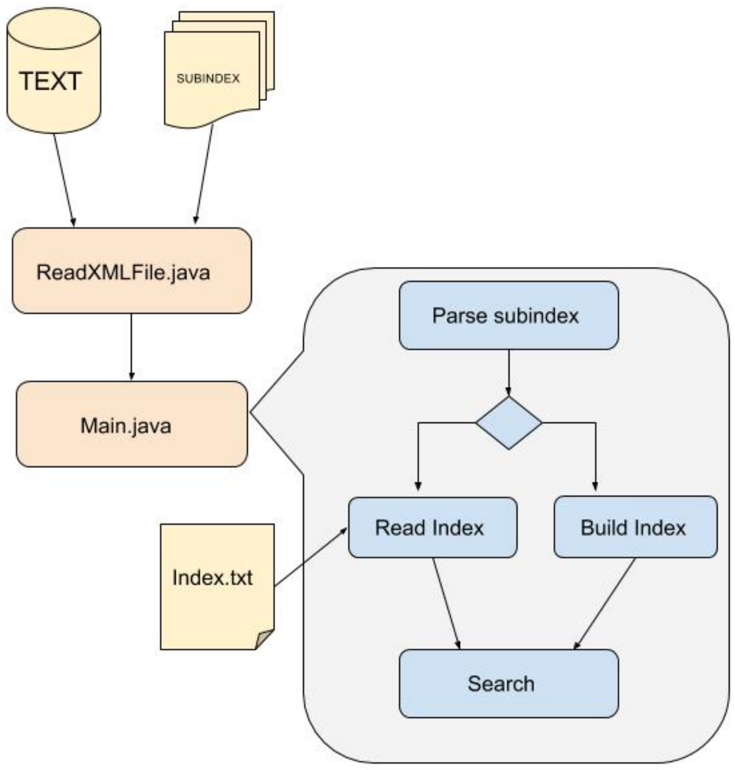
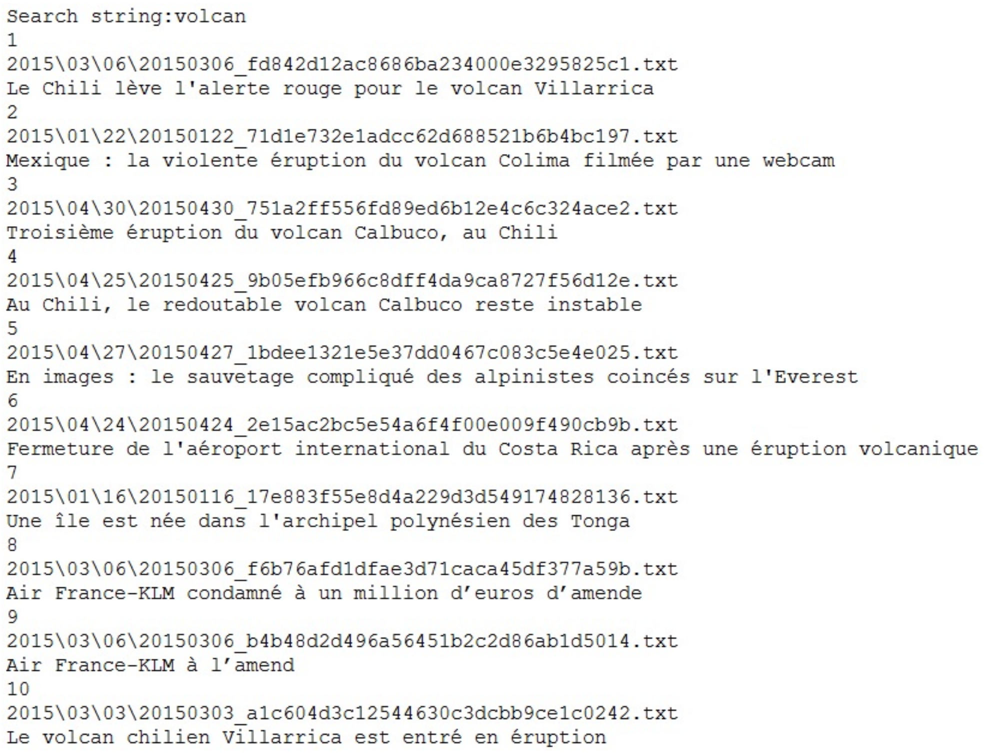


Figure 1: Architecture of the Program

2.3 Evaluation of search results

Once the system was built, an evaluation of the pertinence of the search results was required. Performing such an evaluation on a search engine is not an easy task, mainly because it strongly depends on the subjectivity of the user. What an user might consider acceptable in terms of time of response, can be seen as unacceptable for another user. Moreover, while a search request might return the documents which fulfill the need of an user, these documents might be not what another user was searching for.
For the evaluation of our program, we ran search tests with 10 different search strings for each indexation method, and performed an analysis of pertinence of the first 20 documents returned by each search.
When using the indexation method based on simple segmentation we notice a major drawback even from the first search. More precisely, since our system searches for the intersection of documents which contain each word in the search string, for the case of simple segmentation, the search string “Charlie Hedbo”, which is misspelled, yields no results. This is of course not a desired outcome of a search engine, since typing mistakes are very common for such a system.

Figure 2 illustrates the first 10 documents provided by our system as a result to the input string ’volcan’, when simple segmentation was used as indexation method. While we can observe that all the documents returned by the system contain references to volcanoes, it is nevertheless difficult to asses whether an user of our system would consider the list of documents pertinent. It is possible that the user was interested in a different volcano, and no file referencing this volcano has been reported.
From a first study of the best 20 results returned by the system for each of the search strings, we can observe that the documents returned by the system when stemming indexation was performed, seem more relevant relative to the search string. Moreover, the results returned are less susceptible to typing mistakes and seem more strongly correlated with the subject of the search string. But what we would actually like to compute is a value for the relevance of each search result, and as explained before, that is not really easy to compute. A good way to measure the performance of a system is to compute the precision and recall of the result returned. But in order to compute the precision and recall we need to know which are the pertinent documents with respect to a search string. To



```
Search string:volcan
1
2015\03\06\20150306_fd842d12ac8686ba234000e3295825c1.txt
Le Chili lève l'alerte rouge pour le volcan Villarrica
2
2015\01\22\20150122_71d1e732e1adcc62d688521b6b4bc197.txt
Mexique : la violente éruption du volcan Colima filmée par une webcam
3
2015\04\30\20150430_751a2ff556fd89ed6b12e4c6c324ace2.txt
Troisième éruption du volcan Calbuco, au Chili
4
2015\04\25\20150425_9b05efb966c8dff4da9ca8727f56d12e.txt
Au Chili, le redoutable volcan Calbuco reste instable
5
2015\04\27\20150427_1bdee1321e5e37dd0467c083c5e4e025.txt
En images : le sauvetage compliqué des alpinistes coincés sur l'Everest
6
2015\04\24\20150424_2e15ac2bc5e54a6f4f00e009f490cb9b.txt
Fermeture de l'aéroport international du Costa Rica après une éruption volcanique
7
2015\01\16\20150116_17e883f55e8d4a229d3d549174828136.txt
Une île est née dans l'archipel polynésien des Tonga
8
2015\03\06\20150306_f6b76afd1dfae3d71caca45df377a59b.txt
Air France-KLM condamné à un million d’euros d’amende
9
2015\03\06\20150306_b4b48d2d496a56451b2c2d86ab1d5014.txt
Air France-KLM à l’amend
10
2015\03\03\20150303_a1c604d3c12544630c3dcbb9ce1c0242.txt
Le volcan chilien Villarrica est entré en éruption
```

Figure 2: Example of a search result - results for the string ’volcan’ on simple segmentation index

find all the pertinent documents we have to manually parse the whole data set and select these documents, and beside being a highly tedious task, it is also based on the subjectivity of the evaluator. Nevertheless, in order to compute the values for precision and recall for both indexation methods we made the assumption that the relevant documents for each search string are the represented by the union of the first 20 results obtained with both indexation methods. Therefore, we merged the best 20 results obtained with both indexation methods and then we manually evaluated each result in order to evaluate its pertinence regarding the search string. Thereafter, we considered only the documents selected in this process as pertinent while all others were assumed not pertinent. Finally, we computed the values for precision and recall on the first 100 results obtained for each search string through both indexation methods. Table 1 illustrates the results of our computations for each search string. It is important to mention that for some search strings, such as ’Charlie Hedbo’ and ’Sepp Blatter’ the search based on simple segmentation returned no results. Moreover, while we set the maximum number of resulting documents to 100, for some of the search strings a lower number of documents were returned as result, meaning that the specific words were found just in those documents.

Analyzing the values obtained for precision and recall for our experiments we can conclude that stemming indexation obtained better results throughout our search. While we did not always obtain high values for precision, the value for recall is in many cases 1.0, which means that our system successfully selected all relevant documents in regard to our search string. But this does not mean the results are satisfactory form the user’s perspective, since the user will have to select the relevant documents out of a large collection of documents reported as pertinent. Moreover, we need to consider that some assumptions were made in selecting the set of pertinent documents, while the process also involved the subjectivity of the evaluator.

2.4 Performance evalution

In addition to the pertinence of the search task, another evaluation which needs to be considered for a search engine is the performance in terms of computing time and memory occupied. In the context of an exponential grow of available data especially on the web, and considering that even a "small" search engine for a site has to deal with a large, if not very large amount of data, computing time and required memory are important criteria to be considered when building a search engine. Moreover, the speed of the system in returning the results of a search is a very important factor in its acceptance by the user. Therefore, in the current chapter we perform

Search String	**Indexation Method**	**Precision**	**Recall**
Charile Hedbo	Segmentation	-	0.0
Charile Hedbo	Stemming	0.0311	1.0
volcan	Segmentation	0.4838	0.9375
volcan	Stemming	0.3404	1.0
playoffs NBA	Segmentation	0.625	1.0
playoffs NBA	Stemming	1.0	1.0
laïcité	Segmentation	0.2272	1.0
laïcité	Stemming	0.2272	1.0
Élections législatives	Segmentation	0.1223	0.7666
Élections législatives	Stemming	0.1470	1.0
Sepp Blatter	Segmentation	-	0.0
Sepp Blatter	Stemming	0.7692	1.0
budget de la défense	Segmentation	0.2363	1.0
budget de la défense	Stemming	0.2241	1.0
Galaxy S6	Segmentation	0.0065	1.0
Galaxy S6	Stemming	1.0	1.0
Kurdes	Segmentation	0.4615	1.0
Kurdes	Stemming	0.4285	1.0

Table 1: Precision and recall using both segmentation methods for 10 search strings

an evaluation of our system, at first in regard to the memory consumption and secondly regarding the computation time. All of our experiments have been conducted on a computer with an Intel i7 processor and 8 GB of RAM.

Memory Consumption

In order to evaluate the memory consumption of our system we used the JAVA utility JConsole. Figure 3 and 4 show the heap memory consumption for indexation and search, using simple segmentation or stemming respectively. For both indexation methods we ran the JConsole utility during the indexation phase and for all ten search strings. Comparing the two figures, we can see that the memory consumption is more elevated in the case of simple segmentation, which is to be expected considering the fact that through simple segmentation the number of words obtained by splitting the corpus is larger. Nevertheless, the difference between the two indexation methods is not too big and in general the consumption of memory in the indexation phase as well as in the search phase is considerably bellow the limit of 1GB.

Regarding the disk space required by the index files, there is as well a difference between the index generated through simple segmentation and the index generated through stemming. In the first case, the disk space required by the index file is 13,968 KB while in the second case, for stemming indexation, the disk space required by the index file is just 12,681 KB. So there is about 1000 KB difference in the disk space between the two indexation methods. Considering the size of the set of documents is around 40 MB, both indexes fall below the size constraint of less than 60% of the whole data size. Since in the index file we keep the list of words and for each word the documents in which we can find this word, we assigned an integer value as a key for each file, since keeping the entire name of the file or the path to it has proven to increase too much the size of our index file.

Computation Time

As mentioned before, the time required for the system to build the index and perform a search is a very important factor in the general acceptance of the search engine. If the time required for the system to return a result to a search query is too long, it is to be expected that even if the results provided are pertinent, the user will be unsatisfied with the experience. Therefor, we need to build a system that is at the same time fast and offers pertinent results, or try to find the best compromise between them.
Considering the large amount of documents from our data set, the indexation phase is relatively time consuming for both indexation methods. In this case, as it was to be expected, the simple segmentation performs better, taking around 10.95 seconds to build the index. On the other hand, the indentation through stemming is considerably

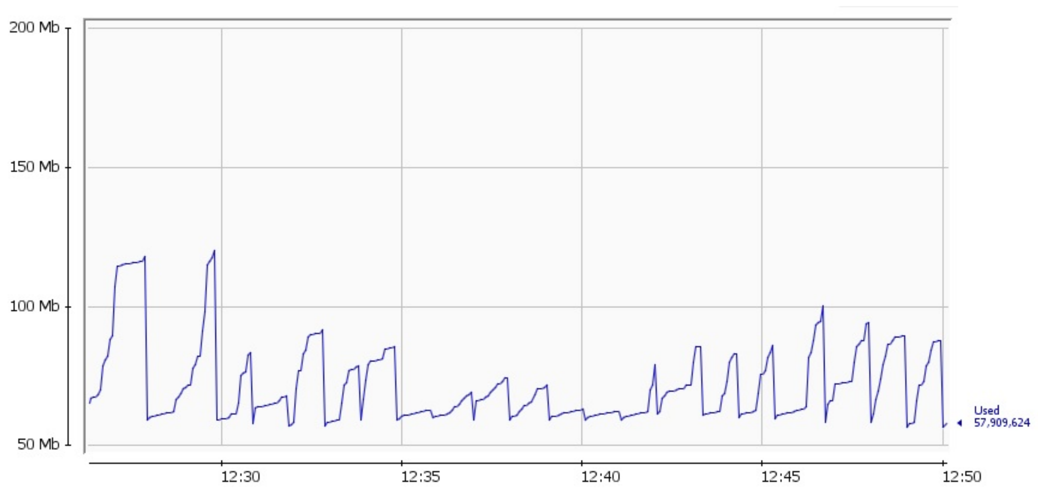


Figure 3: Consumption of heap memory during indexation and search for simple segmentation

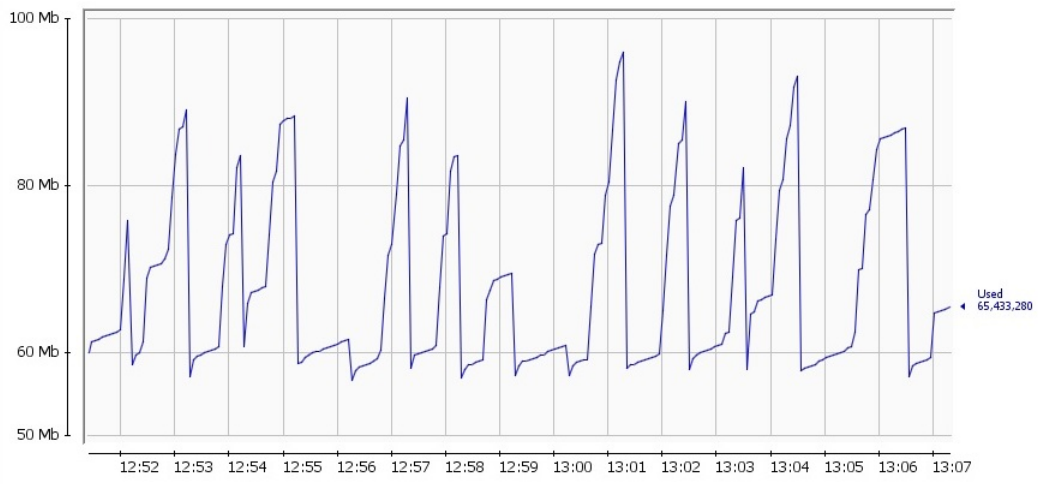


Figure 4: Consumption of heap memory during indexation and search for stemming

slower and takes around 15.89 seconds. Since both indexation methods are quite time consuming, which might render the search experience as too slow for the user, we have the possibility to run the program with the option 'use_index', in which case we just load into memory the index file which we have previously built. In this case, loading the index file takes about 4 to 5 seconds, while the search is a lot faster, taking around 1 second.

2.5 Conclusion

The purpose of the first part of our project was to build a search engine which based on a set of data of around 10,000 files is able to select the most relevant documents in response to a search string provided by the user. We used two indexation methods in the construction of this search engine and compared the results obtained with both of this methods. Besides the constraints related to the implementation, computation time and memory consumption, a big challenge was represented by the evaluation of the system's pertinence regarding a search string. We observed that results obtained when using stemming indexation are more relevant and less susceptible to typing mistakes than the results of simple segmentation, but at the same time the indexation phase is more time consuming.

In the search engine which we built, the cosinus similarity was the metrics which we used to order the documents in according to their similarity to the search string. Nevertheless, we consider that in order to build a system that returns the most pertinent results which can fulfill the need of the user, other criteria should be taken into consideration. For instance, the date of an article could be an important criterion, since we should provide to the user the most recent information from our dataset.

3 Sentiment analysis and prediction

3.1 Introduction

In this section, we are going to introduce the recent advances in natural processing language. We focus on the sub area of sentiment prediction. The idea behind sentiment prediction is to come up with an algorithm which is able to distinguish different sentiments according to their context, for instance : good is a positive sentiment, bad is a negative sentiment. From that, a set of questions occur since the semantic of a word varies from a context to another :

1. What are the properties behind positive and negative sentiments ?
2. How to learn semantic representation of a sentiment in a given context?
3. How to build efficiently a semantic word vector representation ?

3.2 State of art

Natural processing language is one of the most challenging domain. The difficulty is associated with the fact that each word has a different meaning according to context. The word is in a close relationship with its precedents and next words(sentence). These two reasons make learning semantic representation so challenging. Moreover, it is needed to find a smart trick to write the words in a vector representation. Mainly, the popular idea is to do a sampling from a gaussian distribution.
The first contributions on sentiment analysis go back to the early 21th century. It is tackled as a classification problem using Naive Bayes to build a bag of words [6]. These Techniques work well for document classification but fail to predict sentiment. After that , [5] have made some improvement by analyzing the contextual polarity of phrases to distinguish neutral and polar sentences but still failed to come up with an approach that aggregates context and polarity words simultaneously. Furthermore, compositional semantic were discussed [4], lexica and rules for polar and negated words have been built but manually. Recently in [3] a novel approach that learns features and tree structure using POS (Part of speech tagging), dependency parsers and sentiment lexica have been introduced. It's based on conditional random forests with hidden Variables. In [2], these features and tree structure are learned independently of POS, parsers and sentiment lexica.
The anterior techniques used to tackle sentiments prediction failed to come up with a semantic representation due to the approaches used such as Naive bayes, SVMs. These models are efficient for document classification but fails to capture a semantic representation. For that reason, the emergence of deep neural networks allowed to make complex models which are able to capture complex features and regularities by reducing drastically the dimensionality of data and learning invariants as we go further in the deep architecture. In [1] showed how to reduce the dimensionality of data by training a multi layer neural networks and reconstruct the inputs. The model is called autoencoders which are neural networks that learn a reduced Dimensional representation of a fixed size input such as bag-of-word representation of text document. Recently, [7] introduced a framework for parsing using recursive neural networks which requires a labeled tree structure. In contrast, [2] applies a Recursive autoencoders to learn the tree structure and semantic word vectors.

3.3 Definitions

3.3.1 Supervised learning

Supervised learning problems are categorized into "regression" and "classification" problems. In a regression problem, we are trying to predict results within a continuous output. In a classification problem, we are instead trying to predict results in a discrete output. In these learning tasks we know already what our correct output should look like.

3.3.2 Unsupervised learning

Unsupervised learning is a task where we have no idea of how results should look like because the data are unlabeled. The idea is to derive structure from data by clustering the data and find out the correlation between the variables.

3.3.3 Semi supervised learning

Semi supervised learning is an hybrid approach. We start with unsupervised learning then go ahead with supervised learning.

3.3.4 PCA to autoencoders

Principal component analysis takes N-Dimensional data and finds M orthogonal directions in which the data have the most variance. These M principal directions form a lower-dimensional subspace. We reconstruct the N-dimensional data by using the mean value on the N-M directions that are not represented. But PCA learns a linear representation by compressing the input vector representation. To tackle non linear data, we need to add a non linear layers to efficiently represent data that lies on a non-linear manifold. The first non linear layer(related to the input layer) is called the encoder where it converts coordinates in the input space to coordinates on the manifold. The last non linear hidden layer (related to the output) called decoder does the inverse mapping. The encoder and decoder form an autoencoder.

3.4 Deep autoencoders Architecture

Autoencoders achieved state of art performance on non linear dimensionality reduction in a linear time. They provide flexible mappings both ways . But it's very difficult to fine tune the weights using backpropagation. For that reason, th most common used technique is to do an unsupervised layer by layer pre-training [8].
The idea behind the algorithm of autoencoder is the reconstruction of inputs, with the manifold learning we explore data relationship to come up with the appropriate manifold structure. First of all, each instance x_i is used to reconstruct a set of instances x_j. The reconstruction error of each instance $\|x_j - x_i^{'}\|^2$ is weighted by a relational function of x_i and x_j as defined in the learned manifold. The structure of data space is captured by minimizing the weighted distance between reconstructed instances and the original one. The first version of autoencoder, is only used to reconstruct itself and the error reconstruction is $\|x_i - x_i^{'}\|^2$ (it doesn't explore data relationship). The figure 5 below illustrates the difference between the original autoencoder and the improved one [9].

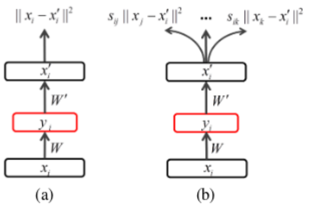


Figure 5: Structure of autoencoders
(a) original version (b) improved version

3.5 Autoencoders domain of applications

An autoencoder is used to learn compact representation of data. It's widely used for computer vision tasks to learn a high level features representation and for speech recognition.
In our work, we introduce autoencoders in the context of natural language processing to make sentiment prediction from a set of sentences.

4 Autoencoders for sentiment analysis and prediction

4.1 Introduction

In this section, we introduce the idea of how to apply Autoencoders for Predicting sentiment label distributions. This brand new technique is to train recursively a semi supervised autoencoders to learn semantic representation. We start by presenting its architecture. Then, we explain the Prepocessing step and finally how to train the autoencoder.

4.2 Architecture of recursive autoencoders for sentiment labeling

The idea developed by [2] consists of inducing semantic representation and predicting multidimensional distribution over several complex interconnected sentiments. This architecture takes advantages from hierarchical structure and compositional semantic to learn semantic representation. It doesn't need any prior knowledge such as sentiment lexica parsers .The figure 6 gives an overview of the recursive autoencoder that predicts sentiment distribution. The inputs represent a sentence divided into a set of words. Each word is a variable-sized from which the model aim to find a vector representation. The hidden layer consists of the part where the neural networks learn semantic representation. The outputs present the predicted sentiment distribution. Recursive autoencoders learn a good semantic representation comparing to Brown clustering (hierarchical agglomerative clustering) which is not able to distinguish good and bad, they make them in the same cluster.

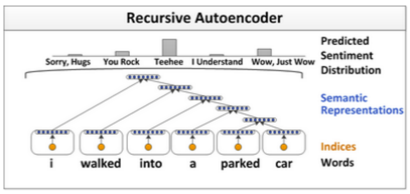


Figure 6: Recursive autoencoder for learning semantic vector representation

4.3 Word representation

The words are represented as a continuous vector of parameters. We have two techniques to do that :

- initialize each word vector $x \in R^n$ by sampling it from a zero mean gaussian distribution $x \approx N(0, \sigma^2)$
- Pre-train the word vector with an unsupervised neural language model [10]

These two techniques predict the the behavior of a word in a give context. After the training ,the word vectors capture syntactic and semantic information thanks to their co-occurence statistics. From a linguistic point of view co-occurence statistics can be seen as an indicator of semantic proximity.

4.4 Feedforward neural network training

As an input we have an $n - gram$ in an ordered list of vectors $(x_1, ..., x_m)$. Each vector is composed of a fixed-size continuous values. This representation fits the autoencoders because the sigmoid function is not linear and continuous. From these vector representation for sentences the autoencoders have as a goal to learn a representation of the inputs. To do so, we define three strategies

- Tree structure is given a priori : the tree is applied to pair of adjacent children and builds a triplets ((y_1->x_1x_2),(y_2->x_3x_4)...). Now, we can compute the parent representation as follow:
  $p = f(W^{(1)}[c_1; c_2] + b^{(}1))$ where W is the matrix of parameters ,b is the bias term and f is the activation function. In this context, we use $tanh$ because it's defined on $[-1; 1]$. It's is more appropriate to distinguish positive and negative sentiments.
  One way to assess our model is the ability to reconstruct the the children as follow :
  $[c_{1'}; c_{2'}] = w^{(2)}p + b^{(2)}$
  The overall assessment is to to minimize the reconstruction error of this input pair. We would like to minimize the euclidian distance between the original input and its reconstruction as follow :
  $E_{rec}([c_1; c_2]) = \frac{1}{2}||[c_1; c_2] - [c_1^{'}; c_2^{'}]||^2$
- No tree structure is given a priori : in this case we are in an unsupervised learning task where we want to learn a tree structure. The goal is to minimize the reconstruction error of all vector pairs of children in a tree. We put $A(x)$ as set of all possible trees that can be built from an input sentence x and $T(y)$ a function that returns the triplets of a tree indexed by s (of all non terminal node). The reconstruction error becomes:
  $RAE_\theta(x) = argmin_{y \in A(x)} \sum_{s \in T(y} E_{rec}([c_1; c_2]_s)$

- Semi supervised tree structure : In this case, each node of tree is associated with it distributed vector representation as the objective is to predict a sentence target distribution rather than general semantic representation. To do so, we add at each parent node a simple $softmax$ layer to predict class distribution as follow :
  $d(p;\theta) = softmax(W^{label}p)$
  For k labels, t_k is the kth element of the multinomial target label t for one entry, $d \in K$ is a k-multinomial distribution such that $\sum_{k=1} d_k = 1$, we can compute the cross entropy error as follow :
  $E_{ce}(p,t,\theta) = \sum_{k=1}^{K} t_k log d_k(p;\theta)$
  The figure 7 depicts how cross entropy and reconstruction error behave at a nonterminal tree node.

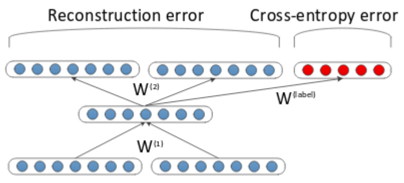


Figure 7: Cross entropy and reconstruction error at a nonterminal tree node

Now, we can define the objective function J over a pair of sentence label pairs noted (x,t) as follow:

$J = \frac{1}{N}\sum_{(x,t)} E(x,t,\theta) + \frac{\lambda}{2}\|\Theta\|^2$
such that the sum over all the errors in the nodes $E(x,t;\theta)$ is computed as follow:

$E(x,t;\theta) = \sum_{s\in T(RAE_\theta(x))} E([c_1;c_2]_s, p_s, t, \theta)$
The overall error at each nonterminal node is the weighed sum of reconstruction and cross-entropy errors:

$E([c_1;c_2]_s, p_s, t, \theta) = \alpha E_{rec}([c_1;c_2]_s;\theta) + (1-\alpha)E_{cE}(p_s,t;\theta)$

Now we write the exhaustive formula of our objective function :

$J = \frac{1}{N}\sum_{(x,t)} \sum_{s\in T(RAE_\theta)(x)} \alpha E_{rec}([c_1;c_2]_s;\theta) + (1-\alpha)E_{cE}(p_s,t;\theta) + \frac{\lambda}{2}\|\Theta\|^2$
The hyperparameter θ weights reconstruction and cross entropy error. When we start training the model and looking for minimizing tge cross entropy error of the softmax layer , the error will backpropagate and influence both RAE parameters and the word representations [2].

After learning the vector representation of the top tree node we train a simple logistic regression classifier on it to predict sentiment distribution.

4.5 Discussion and analysis

The semi-Supervised recursive autoencoders outperforms the anterior architecture thanks to its complex structure that allows to capture complex features and regularities. Its power go back to reconstruction error and cross entropy error . This two measures regulate and come up after a certain number of operations with a stable representation for classification since the error backpropagates and influences the parameters of the autoencoders and the word representation.
The first setting (reconstruction error) is used to reduce the dimensionality of the input then reconstruct the original input. The second setting (cross-entropy) is used to denote the difference between the predicted probability distribution and the actual probability distribution. Hence, we adjust and regulate gradually the hyperparameter until convergence. We noticed that in the final hidden layer softmax function is used as a direct mapping to the output. The efficiency of softmax is in term of differentiability and universal approximation property which are helpful in gradient based learning setting.
Another advantage of this architecture is that the number of hidden nodes is smaller than the number of input

nodes. So the activation function $tanh$ of each hidden node captures most of the information (high density)f rom the input nodes.

5 Implementation and execution

5.1 Dataset presentation

The data set we've used is a collection of movies reviews [16] from people who watched them.The data are split as follows :

- 4798 train examples for negatives reviews: $data/mov/neg.txt$
- 4798 train examples for positives reviews: $data/mov/pos.txt$
- 533 test examples for negatives reviews: $data/mov/test-neg.txt$
- 533 test examples for positive reviews: $data/mov/test-pos.txt$

The size of the vocabulary is 4134. (Only word that occurs more than 5 five times are taken into account).

5.2 Code presentation

The code is split into 7 packages :

- Classify : Contains all the classes and methods related to classification techniques. Here are a description of some main classes :
    1. Accuracy class print the results for the 4 different metrics used to asses the prediction : Precision, Recall, Accuracy, F1 Score.
    2. Classifier theta instantiate the W matrix and b vector with different constructors.
    3. SoftmaxClassifier contains what is related to the softmax function. At each node of the built recursive tree, it contains the methods to compute the class distribution, the accuracy of the result, the train and test score. Also contains the method to get the label vectors.
- Io : Contains all the classes related to vocabulary parsing, data loading. It's the package dedicated to deal with the data.We can find the methods to count word, create vocabulary..etc
- Maths : All the classes related to the maths. QNMinimize for example contains the gradient descent function. The function used for the reconstruction error can be found there too : tanh, sigmoid.
- Parallel : Contains all the classes and methods that deals with the user's computer performance. Checks if the number or cores is enough, if the processors ended the currently running thread and warn the user if not. It's a kind of master in the map-reduce model, it allocates tasks to the different cores to complete tasks.
- RAE : for the Recursive autoencoder construction (RAE). Also contains the feature extraction (W, b) methods for a specific node, getting subtrees, value of RAE cos tat a specific node. Can also compute the classification error for a given tree (using methods from classify package). The methods for backpropagating the cost function through the tree are also here.

5.3 Code processing

1. Running in terminal :
   the first thing to do, is to set the correct configuration for launching the project. Download the file $jblas-1.2.4.jar$ on $http://jblas.org/download.html$. Replace the $jblas-1.2.0.jar$ file by the one downloaded in the lib directory. You can now execute the script $run.sh$ through $./run.sh$ command in the terminal.
2. Using Eclipse :

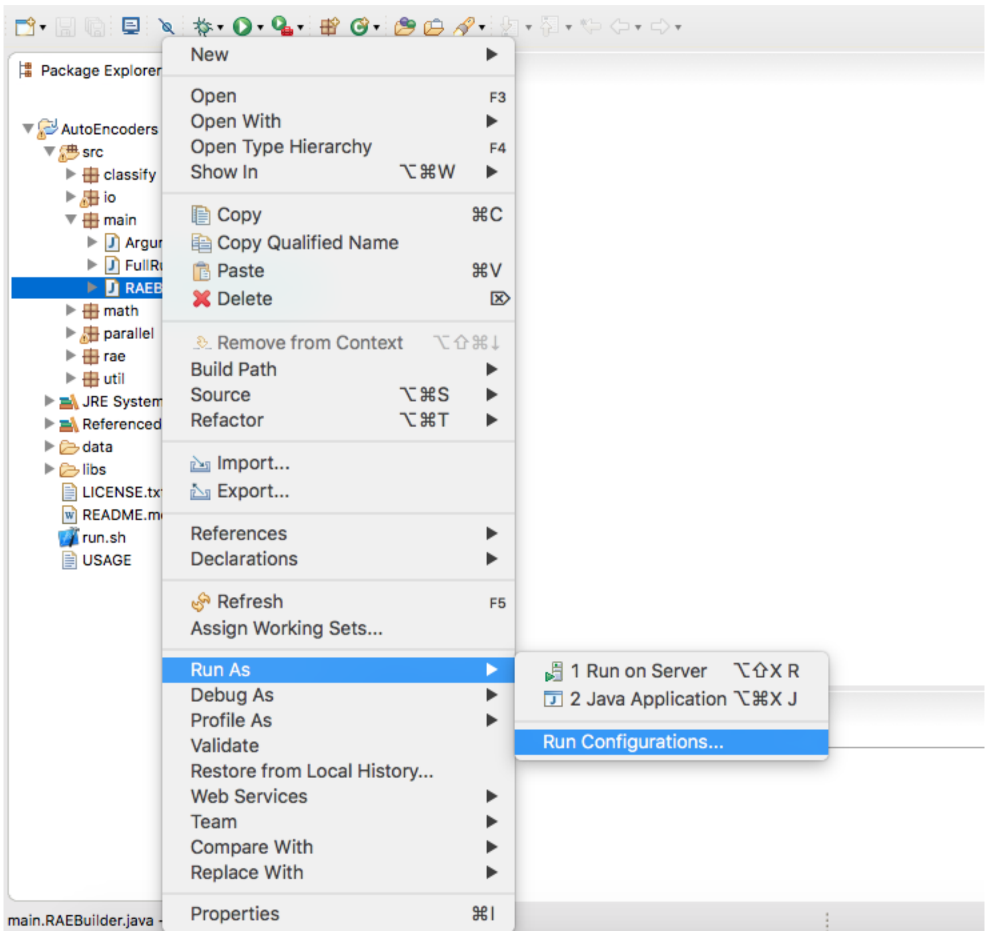
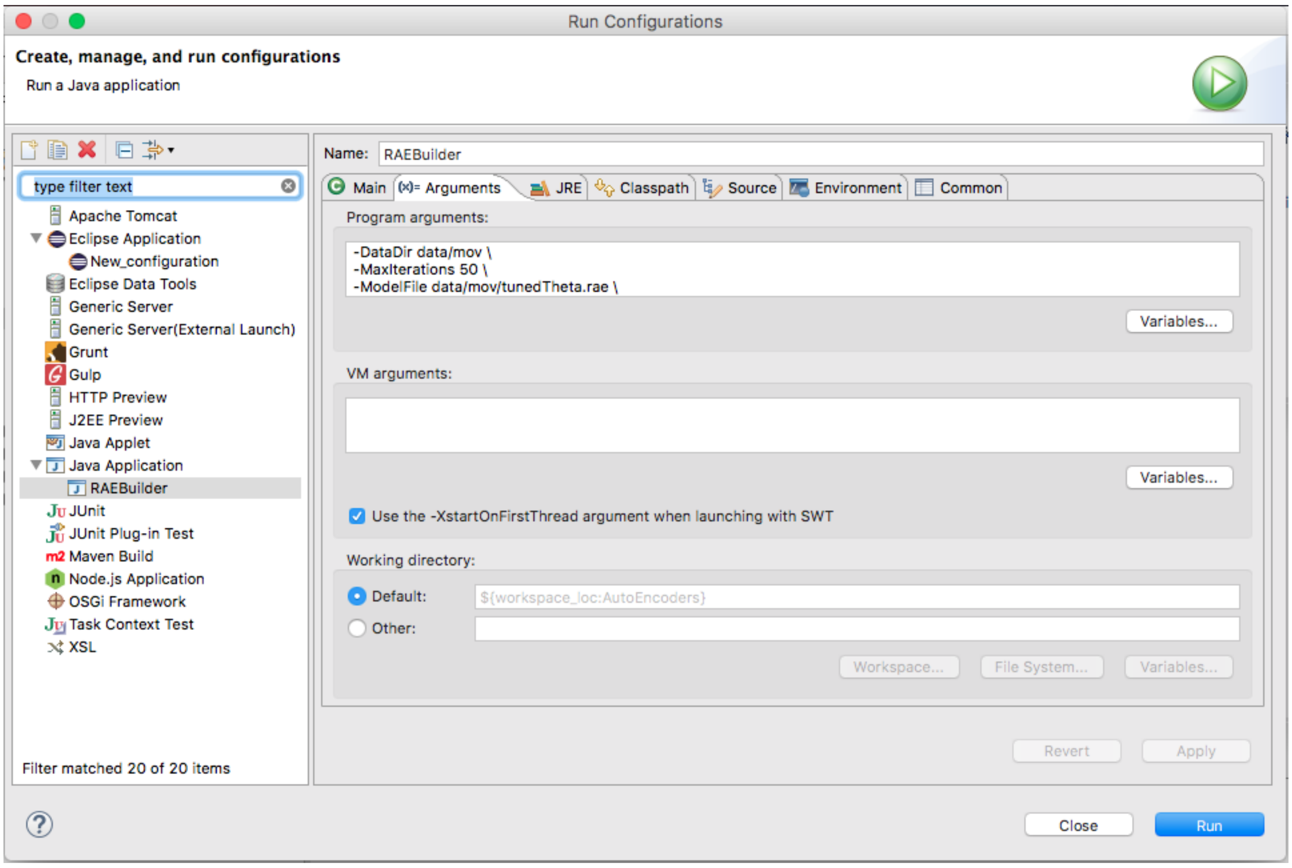


Figure 8: Eclipse environment to lunch RAEBuilder

In the project that you set, do as shown in the picture above.

3. This is the window that will be opened :


Run Configurations

Create, manage, and run configurations

Run a Java application

Name: RAEBuilder

Program arguments:

```
-DataDir data/mov \
-MaxIterations 50 \
-ModelFile data/mov/tunedTheta.rae \
```

VM arguments:

☑ Use the -XstartOnFirstThread argument when launching with SWT

Working directory:

● Default: ${workspace_loc:AutoEncoders}

○ Other:

Filter matched 20 of 20 items

Figure 9: Run configuration parameters

4. run.sh file :

```
-DataDir data/mov \
-MaxIterations 50 \
-ModelFile data/mov/tunedTheta.rae \
-ClassifierFile data/mov/Softmax.clf \
-NumCores 16 \
-TrainModel True \
-ProbabilitiesOutputFile data/mov/prob.out \
-TreeDumpDir data/mov/trees
```

Figure 10: Parameters needed to execute the program

You have to copy this part from the run.sh file in the argument window shown in the picture above.
Your project Is ready to start.
To launch the code, one have to go in the main package.
The RAEBuilder is the central class.
Here is what it does :

1. Load the data and Create the vocabulary
2. Train the RAE.
3. Instantiate a new SoftmaxClassifier that will be trained on the built RAE
4. Computes the accuracy of the classification

5.4 hardware and software constraints

The performance of the user's computer impacts the running time and the convergence speed. On a 4-Core machine, training time for the smaller corpora such as the movie reviews takes around 3hours, and as regarding the EP dataset, it is 12 hours.

5.5 Performance evaluation and results interpretation

There are four metrics used to give the accuracy of the classification:

- Precision and recall :
  Precision here computes the division of correctly classified positive review over (correctly classified positive + correctly classified negative).
  Recall computes the division of correctly classified positive over all the positive review (correctly classified or not).

- F1 Score :

$$F_1 = 2 \cdot \frac{1}{\frac{1}{\mathrm{recall}} + \frac{1}{\mathrm{precision}}} = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}} .$$

Figure 11: F1 score formula

- Accuracy: simply computes the division of correctly classified positive over all the reviews.

The four metrics used give the same results, all around 76% correct classification.There are 2 interpretation to that result :

First 76% is not an excellent score, a model is said to classify correctly when its error is less than 5% in general.
Secondly, the fact that we have the same result for the different metrics proves that the model is relatively robust, as with different performance measures it gives the same classification.

5.6 Conclusion

The code provided here makes the job, gives a reasonable classification rate but is still far from the optimized one. Further, we give an overview of what can be suggested to have improvement : Denoised stacked autoencoders.

6 Critics and suggestions

6.1 Introduction

After the study of the article [2] and its code we noticed how autoencoders look like a really nice way to do a non linear dimensionality reduction. However some problems occur when we put the autoencoders in a real system.
In this section we are going to underline some limits of the autoencoders and suggest some paths to circumvent the related anomalies.

6.2 Critics of autoencoders

The first thing to consider when setting an autoencoder is to make sure that you it doesn't reproduce the identity function. The problem is called bottleneck constraint, to circumvent it we have to make the number of hidden layer much smaller than the input layer or forcing many hidden units tp be 0 or near 0 (sparse coding).
Autoencoders are sensitive to noise but the reality is noisy. What would be the performance of autoencoders if we add to data a noise with ϵ order ?
For all these considerations, we introduce a stacked denoised autoencoders which remedy to the aforementioned anomalies.

6.3 Presentation of Stacked denoising autoencoders

The stacked denoising autoencoders add some randomness to the input and to the hidden values by creating some random noise ϵ [15]. After that, we compute $\widehat{x} = f(x + \epsilon)$ and see whether $\widehat{x} \approx x$. The advantage of noise is that it forces the hidden layers to learn more robust features. It also adds robustness against overfitting to the model. The general overview of stacked denoising autoencoders is depicted in figure 12 [15].

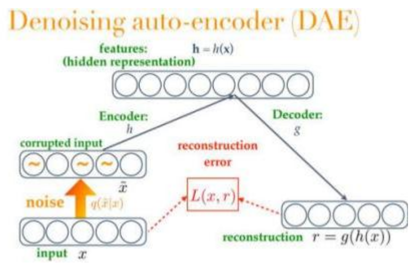


Figure 12: Stacked denoising autoencoders architecture

In the stacked denoising autoencoders, layers are wisely trained one by one until convergence, so all the hidden layers are fine tuned,the outputs of each hidden layer after pre-training is the input of next hidden layer. At the end of training we have two setting to explore in order to fine tune all the architecture as it is depicted in figure 13 .The advantages of pre-training is that it makes optimization easier and reduces overfitting.

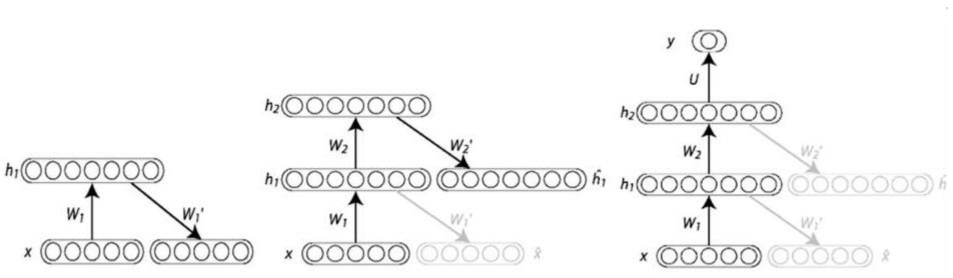


Figure 13: Layer by layer pre-training denoising autoencoders

We have two ways of fine-tuning :

1. Unsupervised fine-tuning by applying backpropagate to the complete autoencoders and compute the error function.
2. Supervised fine-tuning by using a softmax layer where we do backpropagation only on this layer and not on all the architecture.

Pre-training can be avoided if only if we choose the right activation function and parameters initialization [8]. But we don't know a priori the good initialization [14] and activation function

7 General conclusion

The general purpose of our project was to discover, study and understand various methods and algorithms used in the domain of text processing and information extraction from documents. Our project was divided in two main parts. The first part consisted in building a search engine, which is able to select the most relevant documents form a collection of about 10000, in respect to a search string provided by an user of the system. We built the system and analyzed it in terms of pertinence of the results obtained as well as the performance regarding the computation time and memory required by the system. We reported on the results of our system, its limitations and the challenges we faced. The second part of our project consisted in understanding and implementing the work done by Richard Socher on Semi-Supervised Recursive Autoencoders for Predicting Sentiment Distributions [2]. Autoencoders are powerful tools to reduce the dimensionality of word vectors and learn semantic representation. They have shown a 76% accuracy in the sentiment prediction for movies, but there are still some improvement to be done. Stacked denoising autoencoders are one of the solutions that can be envisaged to help such a purpose.

8 Individual and group contributions to the project

Initially, we started all working together on the two parts of the project. Each of us finished the fifth lab sessions required to make the search engine and give it to the student that centralizes this task. Meanwhile, we did a group reading about the article to get a better understanding of the article. Then, we've shared tasks as it is mentioned below. We also talked about the general structure ,organization and table content of the final report. Everyone worked on the two parts of the project but each of us has a specific task to centralize.

Ahmed MAZARI

I finished my five lab sessions and sent it to my colleague . Then, my main task was to get a deep mathematical and algorithmic understanding of the article. Looked at the state of art and make a survey. Analyzed (study and critics) deep autoencoder model and suggested alternative. Explained how to use autoencoder for sentiment prediction. I was in charge of the final report organization and structure.I also wrote the corresponding part of the report which is :

- Abstract
- Section 3 :Sentiment analysis and prediction
- Section 4 :Autoencoders for sentiment analysis and prediction
- Section 6 : Critics and suggestions

Hafed RHOUMA

I finished my five lab sessions and sent it to my colleague Mihaela who was in charge of the first part of the project. Secondly, my main task was to get a deep understanding of the code, the article and how to use autoencoders for sentiment analysis. I want through th code package by package and class by class to get a deep understanding. I analyzed the code and interpret the results. I also wrote the corresponding part of the report which is :

- Section 5 implementation and execution
- General Introduction
- General conclusion

Mihaela SOROSTINEAN

I was in charge with the first part of the project, building the search engine. Based on my work from the practical part of the class, but also using the work of my two colleagues, I built the search engine and did the evaluation of it's performance.I also give a general overview and understanding about the article and wrote my corresponding part which is :

- Section 1 Search engine
- General conclusion

References

[1] G. Hinton, R. Salakhutdinov. Reducing the Dimensionality of Data with Neural Networks. 2006

[2] R.Socher, J. Pennington, E.Huang,A. Ng, C.Manning. Semi-Supervised Recursive Autoencoders for Predicting Sentiment Distributions. 2011

[3] T.Nakagawa, K.Inui, S.Kurohashi. Dependency Tree-based Sentiment Classification using CRFs with Hidden Variables.2010

[4] Y.Choi, C.Cardie. Learning with Compositional Semantics as Structural Inference for Subsentential Sentiment Analysis.2008

[5] T.Wilson, J.Wiebe, P.Hoffmann .Recognizing contextual polarity in phrase-level sentiment analysis.2005

[6] B.Pang, L.Lee, S.Vaithyanathan. Thumbs up? Sentiment Classification using Machine Learning Techniques. 2002

[7] R.Socher, C.Chiung-Yu, A.Ng, C.Manning. Parsing Natural Scenes and Natural Language with Recursive Neural Networks. 2011

[8] X.glorot, Y.Bengio. Understanding the difficulty of training deep feedforward neural networks. 2010

[9] W.Wang and al.Gerealized Autoencoders : A Neural Network Framework for Dimensionality Reduction. 2014

[10] R.Collobert, J.Weston.A Unified Architecture for Natural Language Processing : Deep Neural Networks with Multitask Learning. 2008

[11] Lisa Lab, deep learning tutorial. University of Montreal

[12] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Lecun and Rob Fergus. Regularization of Neural Networks using DropConnect. ICML 2013

[13] G.E.Hinton, N. Srivastava, A.Krizhevsky, I. Sutskever, and R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. CoRR, abs/1207.0580, 2012.

[14] Dmytro Mishkin and Jiri Matas.All you need is a good init. ICLR 2016.

[15] Pascal Vincent and al.Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion. 2008

[16] Movie-review dataset. http://www.cs.cornell.edu/people/pabo/movie