Politecnico di Milano
Scuola di Ingegneria Industriale e dell'Informazione
Corso di Laurea Magistrale in Ingegneria Informatica

ANALYSING MARKETING TRENDS FROM PRODUCT REVIEWS

Relatore: Prof. Pier Luca Lanzi

Correlatore: Dott. Andrea Lui

Tesi di Laurea di:
Paolo Polimeno Camastra
Matricola 874845

Anno Accademico 2017/2018

Acknowledgments

I would like to thank my Advisor, Professor Pier Luca Lanzi, for his support and guidance. I would like to thank Professor Barbara Di Eugenio from UIC, my colleagues Mohammed Elshendy and Andrea Lui for their help and feedback on this work. I would like to thank my family, especially my sister for her help in revision, my friends and course mates, especially Gabriele who inspired the first page design, and my colleagues, especially Fedez.

P.P.C.

Contents

List of Figures

List of Tables

Sommario

Oggigiorno l'e-commerce è più importante di quanto sia mai stato: Amazon.com detiene il 5% della quota del mercato retail statunitense e Alibaba Group ha generato 25 Miliardi di dollari di ricavi nel Single Day in Cina. Le recensioni online sono sempre più cruciali per i consumatori che cercano consigli su cosa comprare o cosa fare e sono di innegabile importanza per un gran numero di business: retail, alberghi, ristoranti, strutture mediche...

Questo lavoro analizza alcune sfide presentate dalla vasta mole di dati costituita dalle recensioni di Amazon.com, in particolare Document Clustering, Opinion Polarity Analysis e Forecasting dell'Impatto Economico delle Recensioni. Proporremo soluzioni per i casi d'uso identificati, che riteniamo interessanti per le aziende. I casi d'uso identificati sono: Document Clustering nel campo delle rencesioni online, Transfer Learning per classification del Sentiment e previsione dell'indice di ricerca di Google Trends per alcuni prodotti.

Il seguente lavoro è diviso in tre parti principali, ciascuna delle quali si inquadra in una delle tre macro aree menzionate. Inoltre, in ciascuna parte verrà utilizzato principalmente un particolare sottoinsieme di tecniche di Machine Learning, rispettivamente: Unsupervised Learning, Supervised Learning e Time Series Forecasting. Dopo ciascuna fase valuteremo i risultati secondo metriche matematiche di performance.

Abstract

Nowadays e-commerce is more relevant than it has ever been before, with Amazon having 5% percent of USA retail and Alibaba Group being able to generate $25 Billion of revenues on China's Singles' Day. Online reviews are more and more crucial for customers trying to choose what to buy or what to do and they are of undeniable importance for a wide variety of businesses: retail, accomodations, restaurants, healthcare...

This study looks at some challenges offered by the huge amount of data constituted by Amazon.com reviews, in particular Document Clustering, Opinion Polarity Analysis and Forecasting of Economic Impact of Reviews. We will propose solutions to identified use cases, which are deemed interesting for companies. The use cases selected are: Document Clustering in the field of the online reviews, Transfer Learning for MultiClass Sentiment classification and Google Trends index prediction for some products.

The study is divided in three main parts, following the three above mentioned macro areas. Each of the steps will make use of a particular subset of Machine Learning techniques, respectively: Unsupervised Learning, Supervised Learning and Time Series Forecasting. After each phase we will evaluate results according to mathematical performances metrics.

Chapter 1

Introduction

Product reviews are ubiquitous nowadays: the dot-com bubble of 1999 saw the birth of a number of companies which allowed their users to express their opinion on anything from sports teams to movies. Their importance has been steadily increasing: according to a survey from BrighLocal.com, Brightlocal, 2018 in 2017 97% of consumers used the internet to find a local business, up from 95% in 2016, and 85% of consumers trust online reviews as much as personal recommendations. The significant role played by online reviews nowadays prompted us to choose them as object of our study. Below we highlight the purpose of our work as well as the methodology we adopted.

1.1 Purpose

1.1.1 Economic Impact of Reviews

It is clear that, given the undeniable relevance online reviews have assumed in recent years, it becomes critical for a business to be able to estimate the "economic impact" of online reviews on the sales of their products. Specifically, we decided to focus on Amazon.com, given its importance as online marketplace. What we mean by "economic impact" is a series of indicators which relates Amazon reviews to product sales: of course, we can imagine that very bad reviews reflect a product which is not appreciated at all and that will likely sell less in the future and, viceversa, product which have wide and clear consensus will probably undergo an upward trend. What we hope to extract is quantitative insights such as: do the volume of published reviews affect product sales? how long after a trend in reviews reflects on actual sales? how much does good/bad review polarity matter? how

much do competitors' product reviews affect our sales? In order to answer these and more questions we need two kinds of data: Amazon reviews, which we happen to have, and sales data form various companies, which, on the contrary, are not so easy to collect, given their proprietary nature. Confronted with this problem, we came up with an interesting solution: as shown in various publications (e.g.[2]) Google Trends is strongly correlated with sales volume and can therefore be considered a reliable proxy for them.

Google Trends

Google Trends is a website which provides query volumes for the majority of terms searched on Google.com. As we can see in Figure 1 the scale of Google Trends results ranges from 0 to 100. The y-axis is the relative volume of search with respect to the highest volume in the specified timeframe (y=100).

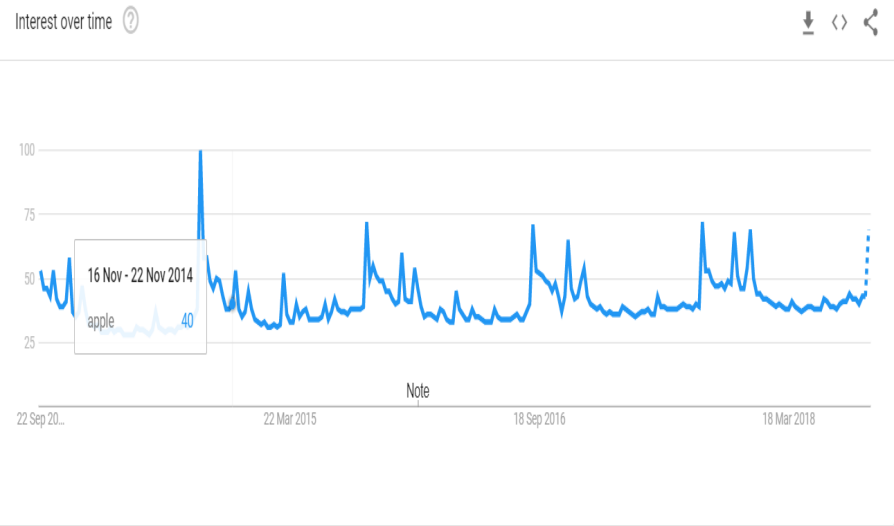


Figure 1.1: Sample Google Trends Time Series

Google Correlate

Google Correlates is another tool by Google which enables you to search for search volume trends which are strongly correlated to a time series of your choice. We will use it to look for other features which can help us in economic value prediction.

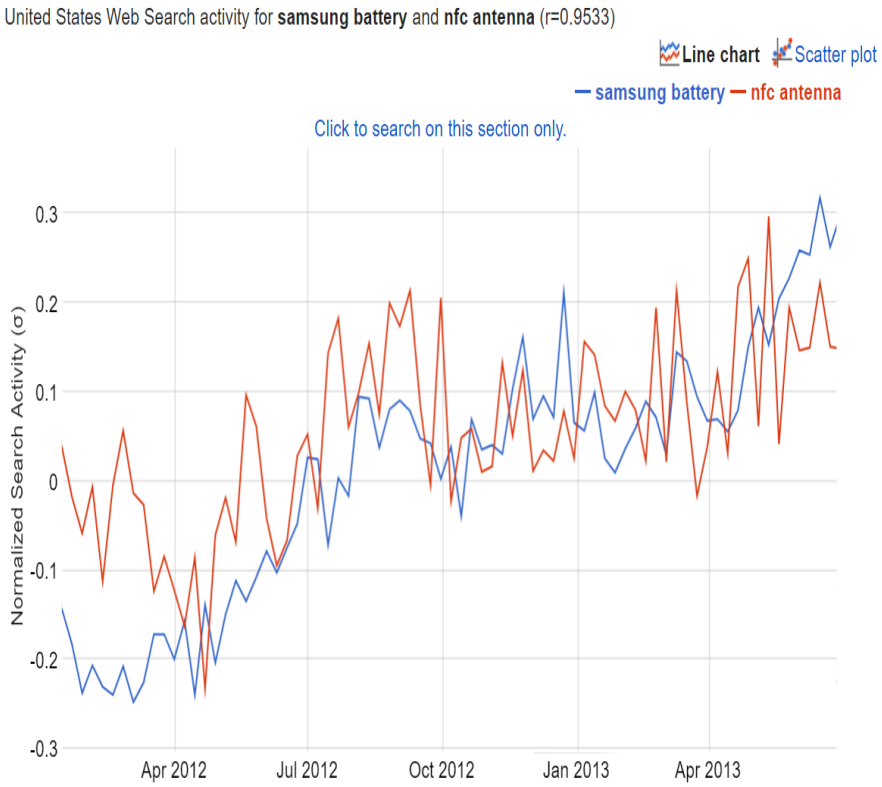


Figure 1.2: Sample of correlated Time Series from Google Correlate

1.1.2 Other tasks

While working on Time Series prediction for Amazon.com reviews, we noticed how two important tasks related to online reviews and which can also be of help when dealing with less structure environments such as Social Networks, online news or blogs.

1.2 Proposed Approach

In this section we will detail the approach we will be following for each of the three tasks we intend to address. In particular in each of the three tasks we will make use of tools from a specific area of Machine Learning, respectively: Unsupervised Learning for the Text Clustering part, Supervised Learning for the Opinion Polarity part and Time Series prediction for the Economic Impact Review part.

1.2.1 Text Clustering

Before looking at text clustering, we had envisioned a system which would have performed detection of "noise" in reviews, where with noise we meant reviews which are written in order to deceive customers into thinking a certain product is much better than it actually is (hyper spam) or much worse than it actually is (defaming spam).
However, we soon noticed that it is really difficult to identify "spam" reviews, because, if they are well-crafted they can appear exactly like a legitimate one and even a human would not be able to detect them. In order to divide products into groups and limit the application of "spam" detection procedures, we had already started working on text clustering. We were getting interesting results and, therefore, we decided to go on working in this field.
We divide the process into two steps: first we focus on clusterization of product titles, subsequently we consider product reviews.

Product Name Clustering

Before performing clustering of the products name, we need to transform them into their vector-representation. We will follow various vectorization flows, in particular: tfidf+SVD, LDA and Word2vec. After we have obtained the desired vectorization we will perform K-means clusterings and we will try to select the optimal number of clusters.
Furthermore we are interested in looking at how different vectorization flows can give rise to different clusterizations of product titles and whether these clusterizations are similar or not. A good similarity between them will make us confident that clustering is robust.

Product Reviews Clustering

In the second steps, we inquire further whether different vectorization flows give rise to similar clusterings. We perform hierarchical clusterings at different granularities and we evaluate Adjusted Mutual Information Score.

1.2.2 Opinion Polarity Analysis

This part of the work is focused on Supervised Learning. We have a double intent: we want to explore models for Sentiment classification which perform well

on a vast and imbalanced dataset such as ours. Then we want to be able to transfer what we learnt in the starting domain to another given domain. This is known in the literature as Transfer Learning. For the second domain we choose a small dataset in order to simulate the situation where we have handlabelled a dataset and where pre-existing knowledge would be very much needed, given that training set is small.
Our expectation is that a model pretrained on a vast dataset should yield better results than the one trained only on the small training set.

1.2.3 Economic Impact of Reviews

This part of the work is in the field of Time Series Prediction and it is an attempt at modeling Google Trends time series for a certain number of products. In order to do so we will consider different approaches. At first we will try to model time series with traditional approaches like ARIMA, after testing for stationarity and differencing if necessary. Then we will implement other models which take input, in particular the features we will try are extracted from Amazon reviews of the product taken into consideration. This last step will allow us to evaluate whether reviews can give us an edge in predicting Google Trends.

Outline

Thesis text is thus structured:

In the first chapter we introduce our work, we discuss its purpose and the various tasks we are going to work on.

In the second chapter we will present NLP and ML tools we will be using.

In the third chapter we will present the previous work in the fields which we are going to work in.

In the fourth chapter we will describe the dataset.

In the fifth chapter we will deal with Text Clustering.

In the sixth chapter we will deal with Opinion Polarity Analysis.

In the seventh chapter we will deal with Economic Value Prediction.

In the eighth chapter we will discuss conclusions and future works.

Chapter 2

Text processing technologies

In the previous chapter we discussed the problems we are going to address in our work. In both problems we will be dealing with plain English text and we will be using the tools which have been developed in the field of NLP since its beginnings in the 50s to these days with the advent of Deep Learning.

2.1 Vectorial representation of text

In order for algorithms to be able to process text, various representations have been developed. The ones we take into consideration map either the whole document or single words or n-grams to a Vectorial Space. Below we illustrate the representations which will be used in our work.

2.1.1 Bag of Words model

This model can be found as early as 1954 in Harris, 1954. If we are working on a set of documents D, we will be able, after a preprocessing phase, to extract N unique tokens. Each document is represented as a vector, whose length is the same as the number of words of the vocabulary extracted from the corpus, that is to say: N. The entries of the vector represent the frequency of each vocabulary term in the document.
For example, if we assume the following ordered vocabulary:

$$\{'I','you','like','hate','pizza','pasta','and'\}$$

and we consider the following document:

$$I\ like\ pizza\ and\ I\ like\ pasta$$

we can represent the document as the vector:

$$[1, 0, 2, 0, 1, 1, 1]$$

This representation does not capture word order in the document, but it keeps all the information about words and their frequencies. However, for certain applications, e.g. documents classification, it would be useful to have a measure of how relevant a term is.

2.1.2 Tf-idf representation

The tf-idf representation was introduced by Karen Jones in Sparck Jones, 1972. A document is defined as a vector of length equals to vocabulary length, just like in BOW model. What changes is that the values in the vector are not the frequency of each term in the document, but they are given by the following function which takes as input the document d, the term t and the set of all documents D:

$$tfidf(t, d, D) = tf(t, d) * idf(t, D)$$

The first product term represents the term frequency used in BOW model. In tf-idf representation it is weighted by the Inverse Document Frequency term, which tells us whether a term is present in a lot of other documents of the corpus and thus is not very relevant or if it is in just a few/in only the one considered and thus is very relevant. Idf is defined as follows:

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

2.1.3 Word Embeddings

In the BOW representation each word (feature) has its own dimension and therefore what we have learned for a specific word, e.g. "orange", is not transferred to semantically close words, e.g. "lemon".
It would be better if we managed to build word representations which captured

semantic similarity, so that our models could generalize more easily and would need a smaller training set. What we would like to get is something similar to 5.7, that

	Man	Woman	King	Queen	Apple	Orange
Gender	0.99	-0.99	0.99	-0.99	0	0
Royal	0	0.01	0.93	0.95	0	0
Age	0.01	0.01	0.7	0.72	0	0
Food	0.04	0.04	0.02	0.03	0.98	0.90
...						

Figure 2.1: Semantic representation of words

is to say a vector representation where the dimensions do not correspond to single words in the vocabulary, but rather to "concepts". Even if we cannot decide which concepts our model learns, we would like to have vectors whose cosine distance is smaller the more they represent semantically similar words.
It turns out that performing tasks such as the one depicted in 2.2, that is to say predicting a word given a context, is really useful for obtaining vectors with the above mentioned characteristics.

Figure 2.2: Predict word from context

In particular in 2013 Mikolov et al., 2013 Mikolov and his team at Google proposed two algorithms that are able to create word embeddings efficiently and drawing from very large text dataset. Their approach is known as Word2vec. Word2vec embeddings have been one of the most popular in research, but other embeddings have emerged such as FastText Joulin et al., 2016 or GloVe Pennington et al., 2014.

In 2.3 we can see examples of word2vec-like representations of different relationships

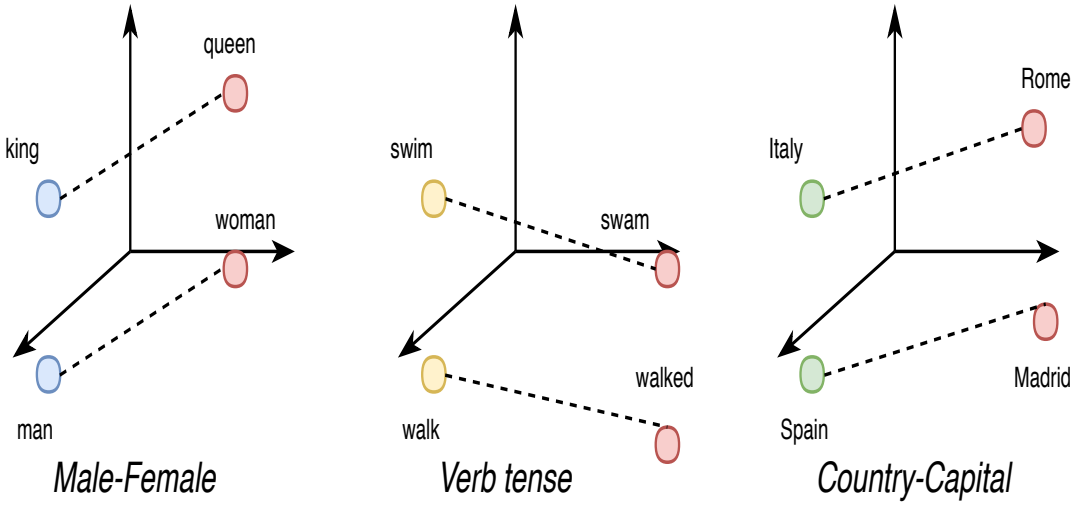


Figure 2.3: Semantic relationships between word2Vec vectors

which cannot be captured by classic vectorial representations, such as BOW, but are indeed very well captured by word2vec and similar word embeddings.

Main Algorithms

Below we give a brief description of three algorithms which are extensively used in statistical NLP and that we will use as baselines in our experiments.

Naive Bayes: Naive Bayes, as the name suggests is based on Bayes' Theorem (Bayes) and on the assumption of independence of features (Naive).

$$\mathbf{x} = (x_1, ..., x_n) \qquad \text{feature vector}$$
$$p(C_k|\mathbf{x}) \qquad \text{for each possible class } C_k$$

The above probabilities are difficult to calculate from a corpus, therefore we use Bayes' Theorem to obtain a better formulation:

$$p(C_k|\mathbf{x}) = \frac{p(C_k)p(\mathbf{x}|C_k)}{p(\mathbf{x})}$$

We notice that the numerator is equal to the joint probability

$$p(x_1, ..., x_n, C_k)$$

which, with chain rule and independence assumption, can be rewritten as

$$p(C_{\mathrm{k}}) \prod_{i=1}^{n} p(x_{\mathrm{i}}|C_{\mathrm{k}})$$

We can estimate both factors from the corpus and we can classify the sample by finding the class that maximizes the above expression.

SVM: Given a training dataset:

$$(\mathbf{x}_1, y_1), ..., (\mathbf{x}_{\mathrm{n}}, y_{\mathrm{n}})$$

SVM is a linear classifier which tries to find the "maximum-margin hyperplane" and then classify the new points based on which sides of the hyperplane they fall. A hyperplane is the set of points which satisfies:

$$\mathbf{x} * \mathbf{w} - b = 0$$

where w and b are weights which must be trained. If the classes are linearly separable, it is possible to draw two parallel hyperplanes that separate the points of the two classes and are as far as possible. The parallel hyperplane which is halfway in between them is the "maximum-margin". If the points are not linearly separable we can still classify them with SVM, but before we need to project them onto a higher dimension hyperplane, a procedure known as "kernel trick".

Random Forest

Random Forest is an ensemble method which is based on decision trees, in particular on averaging a number of unpruned regression trees. Given that it is an ensemble method, in order to be able to exploit its potential we need to generate a series of different weak learners. We start from the same dataset so we need a strategy to generate the above mentioned different learners. Random Forest makes use of bootstrap aggragation, that is to say it generates a variety of new training sets by uniformly sampling the starting dataset with replacement. Furthermore it adds another source of randomness by splitting each trees on only a part of the available featyres. Random Forest improves the variance reduction of bootstrap aggregation by creating little correlated trees.

XGBoost

XGboost is the short for Extreme Gradient Boosting and it is an implementation of Gradient Boosting Machines. GBM is a family of model which have yielded good results in various practical applications. XGBoost has a parellel implementation which makes it faster compared to similar algorithms. Just like Random Forest XGBoost is an ensemble method. In general GBMs method works by trying to iteratively improve the weak learners.
Starting from a model F we can calculate the loss function L which for ecample could be MSE. At each iteration the learner F is updated in such a way $F_{m+1}(x) = F_m(x) + d(x)$ where our objective is to make F equal to the target function. The way we select $d(x)$ is by going in the direction opposite of the gradient of the loss function L.

2.2 Neural Network

Neural Network is family of models which has been developed taking inspiration from the human neural system. The perceptron, presented in 1957, is the first example of NN which is able to perform supervised learning, in particular linear classification.
Perceptron seemed promising at first, but the fact that a single perceptron was not able to produce XOR function, underlined also by the famous book Perceptron by Minsky and Papert, caused a decline of its popularity and of NN research in general. However, it was soon discovered that more perceptrons stacked one after another were able to generate the XOR function and perform non-linear classification.

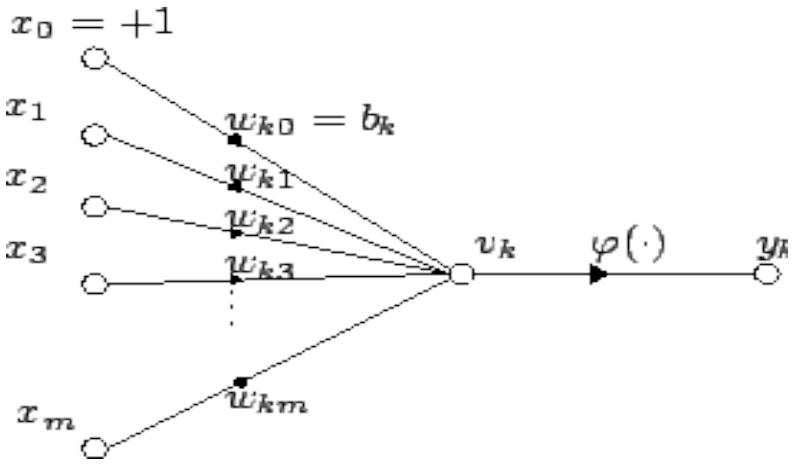


Figure 2.4: Artificial Neuron

Artificial Neural Network was born and it is exactly that: a series of layers formed by artificial neurons, which are like a perceptron which outputs continous values between 0 and 1 instead of only 0 or 1. Before showing the structure of a typical ANN let's see exactly how an Artificial Neuron works.

2.2.1 Artificial Neuron

Artificial neurons are inspired by biological one, in the sense that they receive inputs from other Artificial neurons and their output is a function of the inputs received.
Let's see the mathematical expression which defines an Artificial Neuron:

$$y = \varphi(\sum_{j=0}^{m} w_j x_j)$$

Specifically:

- m is the number of inputs
- x_js are the different inputs
- w_js are the coefficients (weights) of the linear combination of the inputs which is fed to φ
- x_0 is a special input because it is always equal to 1 and its corresponding weight w_0 is also indicated as b_0 and is named bias
- φ is a nonlinear function of the above mentioned linear combination of inputs, $\varphi : \mathbb{R} \to \mathbb{R}$
- y is the output of the neuron

Some examples of the most common nonlinear functions φ used for the AN are given below:

- Sigmoid: $f(x) = \frac{1}{1+e^{-x}}$
- Hyperbolic Tangent: $\frac{2}{1+e^{-2x}} - 1$
- Rectified Linear Unit: $f(x) = max(0, x)$

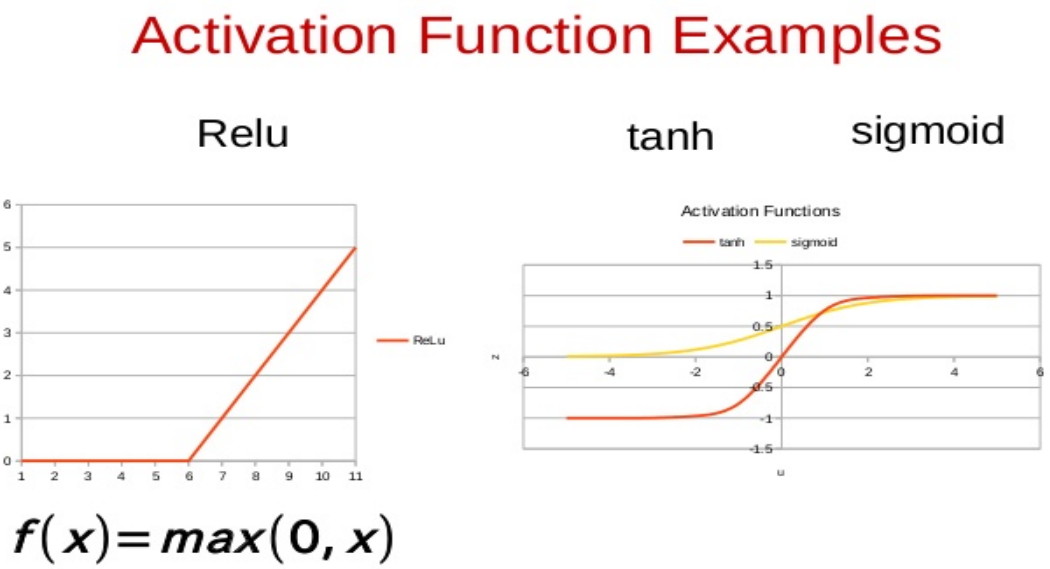


Figure 2.5: Sigmoid function, Hyperbolic Tangent and ReLU

Multi-label classification When we find ourselves with a multi-label classification problem, the Neural Network is usually structured so that the last layer has as many neurons as the number of class we are taking into consideration. The idea is that the output of the i-th neuron should represent the probability of the input being in the i-th class and therefore it should take a value in $[0, 1]$, with the summation of the outputs being equal to 1. In order to do this another kind of nonlinear function is introduced, the *softmax* function: $o_j = \frac{e^{x^T w_j + b_j}}{\sum_{k=1}^{K} e^{x^T w_k + b_k}}$

2.2.2 Training phase: optimization

The objective of the training phase is to find parameters which minimize the selected error function (Loss) calculated over all the samples of the training set.

$$\mathbf{w^*} = \sum_{i=0}^{N\_training\_samples} (y - (\hat{y}))$$

where $\hat{(y)}$ is the prediction output by NN.

Algorithm 1: Random Forest Training Algorithm

```
1 Random initialization of network weight;
2 for all s : TrainFeatures and t : TrainTargets do
3     ŷ= compute_output(network, s);w = w  α(w^T y)y;
```

Stochastic gradient descent

The function L which has to be minimized is a nonlinear and very complex one, therefore we cannot use exact methods. The standard method for NN optimization is Stochastic Gradient Descent which is a variation of standard Gradient Descent. Gradient Descent is a very simple iterative : we start from a random vector of weights w_0, we calculate the gradient of the loss function L with respect to the vector of weights and we substract it to the current vector of weights.

As we have seen, SGD is an iterative method and it updates the weight vectors each time it is presented with a sample from training set. Multiple passes, called epochs, are made on the training set. Until convergence is reached the error on the training set continues to decrease, since we still have not reached a minimum. Up to a certain point the increase in performances will be reflected on the data outside the training set, while if we train for too long the model will probably overfit the train data and we will see performances on outside training data decrease.
In order to avoid that, we can use Early Stopping. Early Stopping is a method of regularization, that is to say a modification of the algorithm which try to prevent overfitting. The standard way Early Stopping is implemented is this: we train model on the Training Set and at the end of each epoch we evaluate its performances on the Validation Set. If performances on the Validation Set have decreased at least by a given selected amount, we keep on training, otherwise it means that the model has probably started to overfit and therefore we stop the training phase. The final weights will be the ones obtained during the last training epoch. In [] we can see a plot of RMSE loss on both Training and Validation set. The point where they start to diverge is where the model starts to overfit and where Early Stopping halts training phase.

2.2.3 Dropout

Deep Neural Networks present a very large number of parameters and it would take a lot of time to train a large number of different networks and combine their prediction in order to prevent overfitting. One way to do so is by using the Dropout

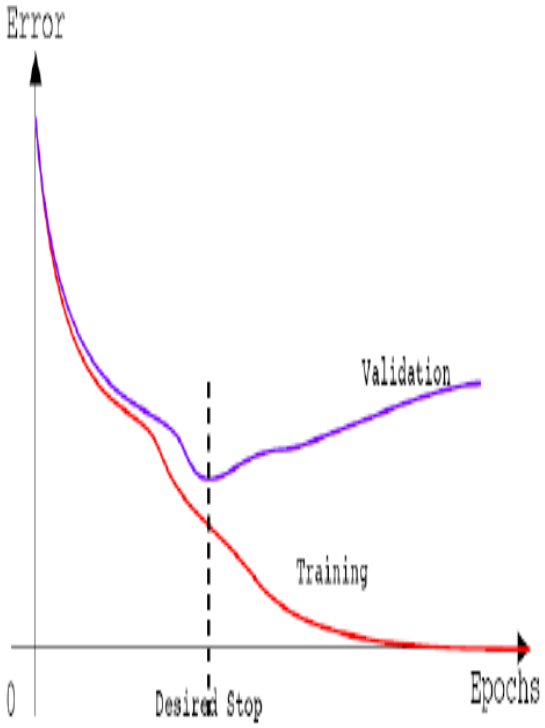


Figure 2.6: Sample graph of error on Training and Validation set showing optimal point

technique. Dropout takes its name from the way training phase is performed: at each iteration p neurons and their connections are dropped. The result is that at each iteration we are training one of the 2^n, where n is the number of neurons, possible networks. During test phase we use the full, not "thinned", network with the weights divided by the factor p, so as to account for the "thinned" factor of training phase. By applying dropout neurons becomes more independent, since they cannot rely on each other and makes the NN more robust, preventing overfitting. In [] we can see a depiction of a full NN vs its Dropout version.

2.3 Deep Learning Architectures for NLP

We start by describing the main DL architectures which are currently used in the NLP field.

2.3.1 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are a particular kind of Neural Network (NN) which was introduced in the 80s and they are particularly suited for modeling

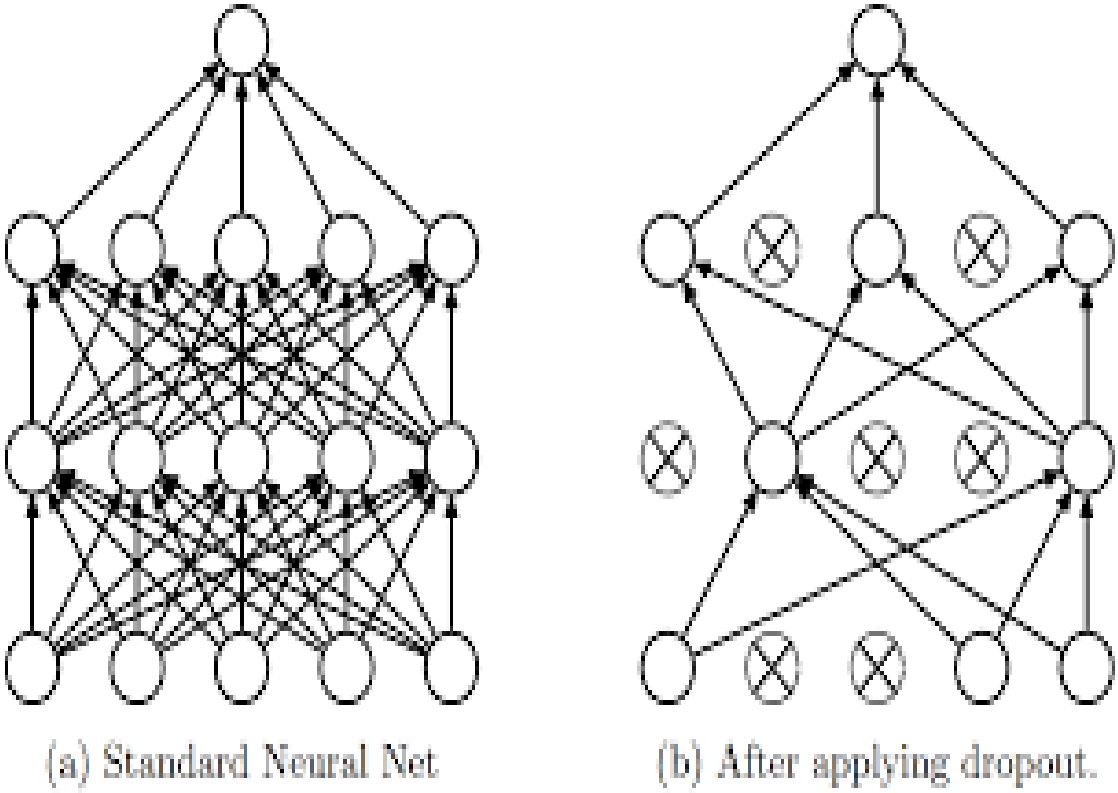


Figure 2.7: Sample neural network with and without dropout

temporal dependencies. RNNs are not so different from feed-forward NN. In both cases we have a Tx input vectors and Ty output vectors and we assume $Tx = Ty$. While the standard NN takes in input all the word vectors, which we can assume to be word in a document, at the same time, the RNN is basically a succession of NNs which takes in input only two vectors: the hidden state vector of the previous time step $a < t - 1 >$ and the word vector of current time step $x < t >$.

We list the two main advantages of using RNN over NN:

- RNN uses the same set of weights at every time step, therefore we can use very few weights with respect to a NN, which must have weights for each word of the longest input sequence.

- With RNN we can have sequences as long as we desire, without having to fix length in advance.

- With an NN the first word and e.g. the fourth word in a sequence are two completely different feature and NN cannot transfer something it learnt for the first word to the fourth word in a sequence. On the contrary RNN, by sharing weights, is able to capture pattern also if they appear in different

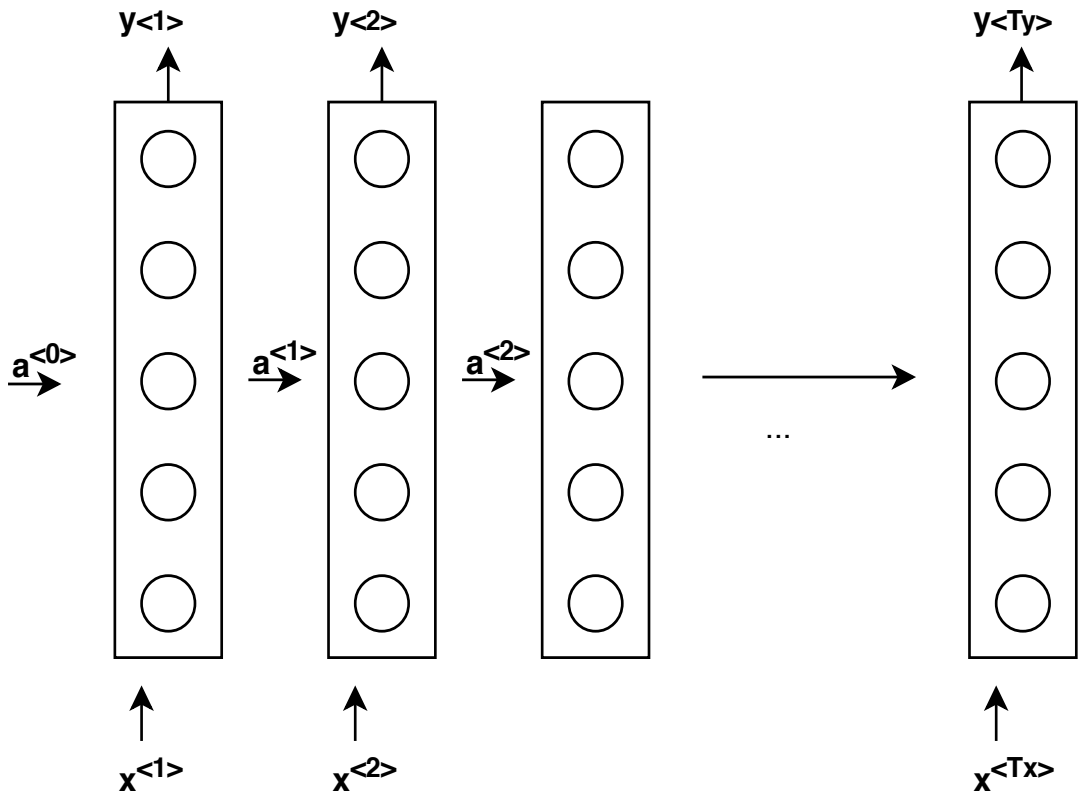


Figure 2.8: A sample multioutput RNN

positions.

As we have seen RNN is nothing more than a series of NNs and therefore we can train it with standard backpropagation, after having unrolled it for the number of time-steps of the input sequence. Unfortunately, an unfolded RNN can be a rather deep NN and as in all deep NN we encounter the exploding and vanishing gradients problem, that is to say a numerical impossibility of propagating the weight update needed to the first layers of the network during backpropagation.
This weak point in the implementation has been addressed with architectural changes. In particular, we will look at two of the variations which have been developed by researchers: Gated Recurrent Unit (GRU) and Long Short-Term Memory (LSTM) unit.

2.3.2 GRUs and LSTMs

In this section we give an overview of a slightly simplified version of a GRU. Both GRUs and LSTMs are based on the same principle of adding so-called "gates". Gates are functions which are in charge of changing the way the hidden status is updated. In this way we allow some information which is in the hidden state, i.e.

some of the values of the vector, to be partially or even completely unmodified with respect to what would have happened without it.
By doing so we are able to mitigate the problem of vanishing gradients and we are able to obtain a network which captures long-term dependencies.
As we said, in 2.10 we can see a simplified GRU, which has a single gate. LSTM was the first of this kind of architecture to be proposed Hochreiter and Schmidhuber, 1997 and we may be interested in discussing its structure or also the structure of the more recent GRU. Actually, there is no need to go into such details, since all these architectures are variations which have been developed by researchers in order to avoid vanishing gradients problem and have emerged as the best one. Still, the principle of filtering the hidden state with gate function is the main idea and it underlies all of them. An interesting point is made in Khandelwal et al., 2018 as to how much context LSTMs are really able to capture.
From the analysis it emerges that increases in window size after 200 time steps affect very little LSTMs performance and, furthermore, words order is only captured in the most 50 recent tokens.

2.3.3 Convolutional Neural Networks in NLP

Convolutional Neural Networks (CNNs) are well known in Machine Learning community. In particular, the Imagenet challenge results accomplished by Krizhevsky, Sutskever and Hinton Krizhevsky et al., 2012 are considered to be the start of Deep Learning revolution. CNNs are widely used in Image Recognition and they are based on the concept of Convolution: a filter, 2.11, slides over the image matrix and takes the dot product between the network's ordered weights and the ordered pixels of the image portion it is on at time t. Filter weights are shared, i.e. they do not change when the filter slides over the image, so that each filter is able to learn a specific feature and it can recognize it in every section of the image.
Each layer of the CNN comprises several filters and each of those is able to learn a specific feature. In CNNs we speak of "compositionality": going from one layer to the next one, features become more complex since they are obtained as a combination of the previous layer's features. As an example, given an image, the first layer of the network takes as input raw pixels. Pixels get combined by the filters: during the training phase, each filter learns parameters which enable it to detect simple patterns, e.g. vertical lines, horizontal lines, objects' edges. The second layer of the network takes as input the patterns learned by the first layer and combines them,

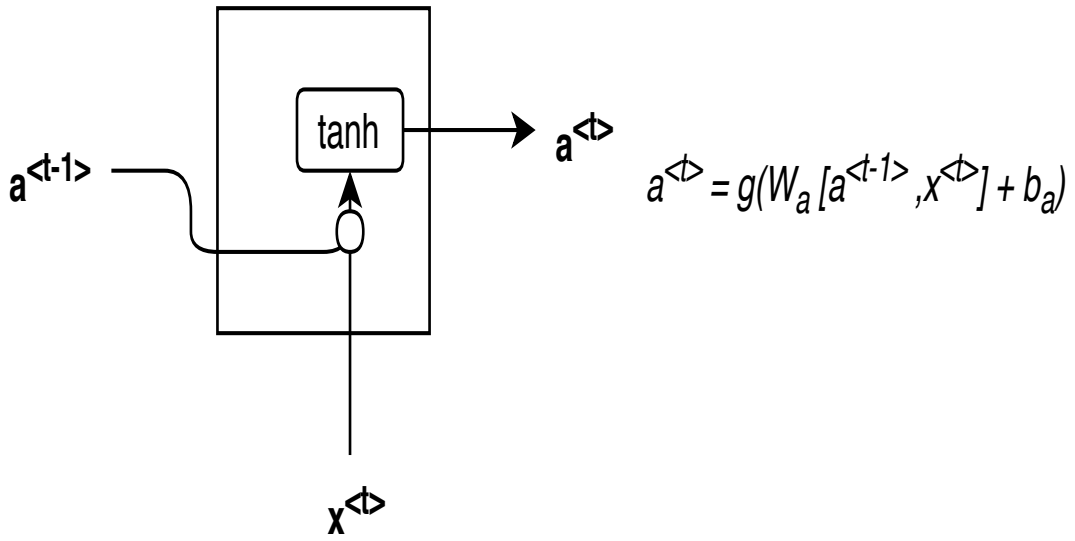


Figure 2.9: A sample RNN cell

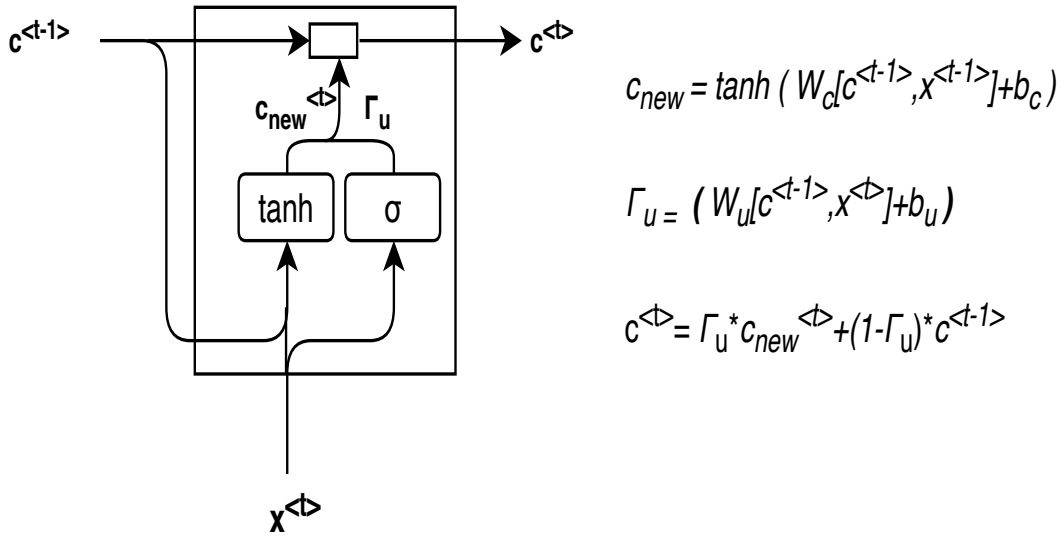


Figure 2.10: Simple GRU

yielding more complex patterns such as lines meeting at specific angles, specific shapes, etc. The third layer aggregates second layer's features and detects details in images: if, as an example, the image depicts a car, each filter of the third layer will detect either wheels or windows or fenders, etc.

CNNs also presents another kind of layer, the Pooling Layer, which makes the network robust with respect to translation and rotation. It achieves this by down-sampling the image, that is to say by extracting a single value (usually average or maximum) from every group of adjacent pixels.

In NLP, 2.12, instead of considering images, we consider the matrix obtained by juxtaposing the word vectors. Filters are a way to capture combinations of words, syntax and semantics.

2.4 Summary

In this section we looked at the various techniques which we are going to employ in the rest of the work. We gave both theoretical foundations and practical use cases. We started from the most important tool, which are the various kinds of documents vectorization, both the classical ones and the more recent ones based on word embeddings.
Subsequently we went to machine learning algorithms which are commonly used in the field of NLP. Even in this case we analyzed both the more traditional approaches such as Naive Bayes, SVM, etc. and the Neural Network-based ones. We presented main concepts of NN training and we described the main architectures which have been proven useful in the field of NLP.

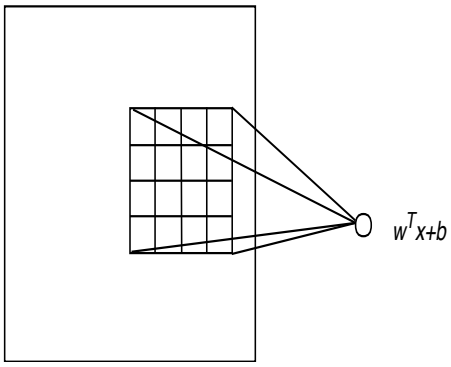


Figure 2.11: Example of CNN filter

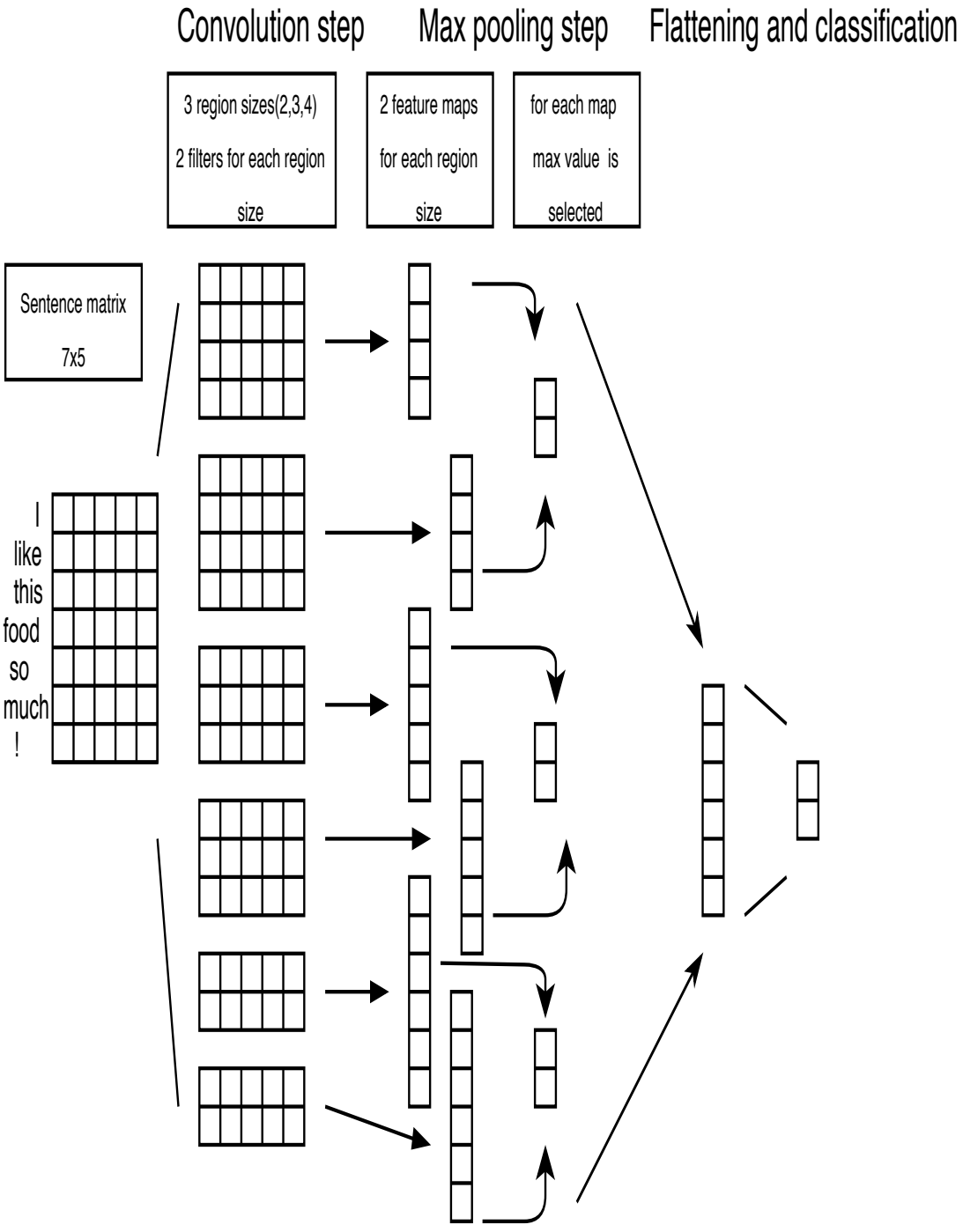


Figure 2.12: Example of CNN applied to NLP

Chapter 3

Literature Review

3.1 Text clustering

In this section we introduce techniques which we will apply in the Text Clustering section of our work, as well as some use cases which can be found in the literature.

3.1.1 Clustering

Clustering is an *unsupervised* technique which aims to discover patterns in data, in particular to assign each data point to a group, i.e. a "cluster". It is *unsupervised* because data has no label to learn from and to be used for evaluation purposes. Clusters should represent a group of points which are more similar to each other than to the points which are not in their cluster, based on some similarity measure of choice.

3.1.2 Singular Value Decomposition (SVD)

SVD gives us a specific k-dimensional subspace which is contained in the original vectorial space. Specifically, SVD gives us the k-dimensional subspace which minimizes the mean projection square error and explains the most variance with respect to all the other k-dimensional subspaces.

3.1.3 Latent Dirichlet Allocation (LDA)

LDA Blei et al., 2003 is a generative model, which is used for topic modeling, that is to say to "discover" which latent topics are hidden in a collection of documents. LDA assumes a fixed number of latent topics and describes each document as a

distribution over topics and each topic as a distribution over words. Topics are not predefined but are discovered through inference. By using Bayesian inference we are able to see which words are most associated to a topic and which topics are most present in each document.

3.1.4 Text Clusterings Examples

In Xu et al., 2015 the authors compare various text clustering approaches: K-means, Spectral Clustering and Convolutional Neural Networks. They cluster 2 labeled datasets of around 10^4 samples and they evaluate obtained clusterings in terms of Accuracy and Normalized Mutual Information. Both metrics are calculated with respect to the true labels.
In T. Liu et al., 2003 the authors evaluate clusterings obtained after a process of feature selection with Precision. Precision labels a cluster with the label of the class which is found to be more present and gives the ratio between the number of samples from majority class and cluster's cardinality.
Again, in Kuang et al., 2015 authors consider Accuracy and NMI with respect to true labels as measures for Clustering Quality. They also evaluate the consistency of the clusterings, by running each method 30 times and giving a measure of how much assignment varies: they consider each couple of runs. For each couple they perform 1-1 matching of clusters from run 1 to clusters from run 2, by similarity, and then they calculate how many samples are not in the same clusters.

3.2 Opinion Mining

Opinion Mining does not focus on the topic of the document, instead, it focuses on the opinion expressed in it.

3.2.1 Opinion base components

According to Esuli and Sebastiani, 2005, main components of an opinion are:

- Opinion holder: the person/organization which holds a specific opinion on a specific object
- Opinion object: what the opinion is about
- Opinion: a statement regarding an object coming from the opinion holder

3.2.2 Main tasks

Again, according to Esuli and Sebastiani, 2005, these are the main tasks which are performed in Opinion Mining:

- Determining subjectivity: it consists in determining whether an opinion is objective or it expresses a personal view. It takes the form of a binary classification task.
- Determining the orientation (polarity): it consists of determining whether a *subjective* text expresses a positive or negative opinion.
- Determining the strength of orientation: it consists in determining how much a *positive* or *negative* opinion is strong.

Among the above mentioned three tasks, we decided to focus on task two and task three, which are known as Opinion Polarity Analysis.

3.2.3 Opinion Polarity Analysis

Opinion Polarity Analysis, also known in the industry as Sentiment Analysis, is usually framed as a classification problem. In its basic version, it is a binary classification (Positive and Negative), while in our case it becomes a multiclass classification problem since we have 5 different possible scores.
We will deal with Document Level Classification: this type of classification considers a document as a basic unit and tries to classify it. Below we give an overview of main approaches which have been tried by researchers.

3.2.4 Lexicon-based Approach

This rather simple approach is based on counting how many positive and negative words are present in a text and classifying the document accordingly.

Dictionary-based Approach

This approach is adopted, for example in Hu and B. Liu, 2004. The idea is very simple: compile a list of words with positive and negative orientation. Since positive or negative connotation is not something we can usually find in dictionaries, we need a strategy to collect the needed list. One idea, which is detailed in Turney, 2002 is to use mutual information: you perform a web search on a term and you

calculate the mutual information with a very positive term such as "excellent" and a very negative term such as "poor". The disadvantage lies in the number of web searches and operations to be performed.
A simpler idea, which is the one described in Hu and B. Liu, 2004 is to exploit the similarity/antonymy information contained in a synonym/antonym dictionary(e.g. WordNet): you start from a list of seeds for which orientation is given and you iterate over all adjectives. If they have a synonym in positive/negative seed list you add them to that list, while if they have an antonym in positive/negative seed list you add them to the opposite list. Of course, some of the words will not fall in either one or the other category, but if we run the procedure again they may, given that the size of the lists has increased. You keep running iteratively until the lists remain the same. Dictionary approach is not able to catch context-dependent orientation of words.

Corpus-based Approach

In the corpus-based approach, starting from a seed list we want to extract other sentiment words from a specific corpus. One idea is explained in "Predicting the semantic orientation of adjectives": they designed a series of rules to extend orientation from one word to another based on connectives. One example is with the 'and' conjunction: if a word is linked to another by 'and' their polarities are assumed to be the same.

3.3 Machine learning-based approach

3.3.1 Supervised Learning

The first paper to use the supervised machine learning approach to classify reviews was Pang et al., 2002. Researchers did not rely on prior knowledge and they let models free of selecting the most discriminant features for the task. They worked with movie reviews, which they selected because of the large availability and the lack of need to hand-label data. The authors show how the results of a classification based on words count from positive and negative lists vary considerably depending on which words are put in the lists. They then resorted to classic supervised learning algorithm for topic classification and they obtained good results using unigrams as features.

Example features

As we can find in B. Liu, 2012, different kind of features are commonly used in sentiment classification:

1. **Words and Word Frequencies**: these are the most important features and are captured by the BOW model.TF-IDF can also be applied.

2. **POS and Syntactic Features**: these features, although more generic, can help in weighing more important words (e.g. adjectives) or in capturing peculiar sentences structure.

3. **Sentiment shifters**: whether a word comes after a negation can completely reverse the meaning of a sentence.

3.4 Transfer Learning

Transfer Learning in Machine Learning context refers to the problem of reusing knowledge gained while solving a certain problem to solve a different but related problem. In Daume III and Marcu, 2006 Domain Adaptation for multiclass classification, a form of Transfer Learning, is detailed: while in standard classification task we assume both Training and Test data $D = (x_i, y_i) \in X \times Y$ to be drawn from the same distribution p, in Domain Adaptation setting, we are presented with training data drawn from two different distributions $p1$ and $p2$ and we need to classify data extracted from $p2$, having $N2 << N1$, with $N1$ and $N2$ being the number of training examples we have for $p1$ and $p2$ respectively.
In our case we deal with a specific kind of Transfer Learning, known as "Cross-Domain" Sentiment Classification. According to B. Liu, 2012 Sentiment Classification has been shown to be highly sensitive to the domain of the training data, both because in different domains there are different ways to express opinions and because some words have different meaning and connotation in different domains. In Aue and Gamon, 2005 Aue and Gamon, from Microsoft Research, focus on transfer learning for two-class sentiment classification. They make use of SVMs in all experiments but one, where they use Naive Bayes, and they experiment with various configurations of training sets. They work with four domains: Books, Movies and two user feedbacks datasets.
In 3.1 we see the number of samples used in the paper we are discussing and we can see that dataset sizes are of the order 10^4.

	Positive	Negative	Total
Books	1000	1000	2000
Movies	1000	1000	2000
Survey_1	2564	2371	4935
Survey_2	6035	6285	12320

Table 3.1: Labels and Classes Distribution

	Experiment 2	Experiment 1	Domain Exp. 1
Movies	72.89	72.08	Books
Book	64.58	70.29	Movie
Survey_1	63.92	70.48	Survey_2
Survey_2	74.88	81.42	Survey_1

Table 3.2: Results of 1st and 2nd Experiment

In experiment 1 the authors train 4 classifiers for each of the domains and then test them on all domains. In experiment 2 the authors also train 4 classifiers: each classifier is trained on a training set comprising equal number of samples from 3 domains and is tested on the remaining domain.
It is interesting to compare performances obtained in Experiment 2 with best performances obtained in 1.
3.2 shows us that best training domain (different from original), is better or almost equal (Movies) than the classifier trained on multiple domain. It seems that the most similar domain is the one which drives performances.
In Glorot et al., 2011 they work on a 22 domains, binary-label dataset of 340.000 Amazon reviews. First they extract features in an unsupervised manner, using Stacked Denoising Autoencoders, which is a DL architecture. Then they train an SVM on a domain and evaluate both on the training domain itself and on all the others. They present results both on a reduced, 4 domains and balanced (# positives = # negatives) version of the dataset and on the full dataset. For the full datasets results for each model are given as the average of the Transfer Loss considering all possible couples of Source and Target domains. Transfer Loss is defined as:

$$t(S,T) = e(S,T) - e_b(S,T)$$

Where e is the error of the considered model and e_b is the error of the baseline.

Chapter 4

Data structure

In this section we describe the structure of Amazon's reviews and the corresponding structure of the chosen dataset. Furthermore, we give dataset statistics which we computed for the period of interest.
The dataset He and McAuley, 2016 has been collected by Julian McAuley, currently Assistant Professor at UCSD.

4.1 Amazon.com

Amazon.com, inc. was founded on July 5, 1994. It is the largest Internet retailer in the world (both for revenue and market capitalization) and second largest after Alibaba Group in terms of total sales. According to eMarketer.com its US market share of e-commerce is 49.1% followed by eBay with 6.6%. This makes it by far the most important e-commerce website for the US market and, on top of that, Amazon.com shopping makes up 5% of total US retail.

4.2 Amazon Review Structure

As we can see from 4.1, Amazon review has different components:

1. review title
2. review text
3. star rating, which is an integer from 1 to 5
4. reviewer's username
5. number of people who found the review helpful

Figure 4.1 Sample Amazon.com review

6. review time

Amazon reviews previously featured a count of how many people found the review unhelpful, but it has been recently removed.

4.3 Dataset structure

Here we give a look at the main features of the dataset:

1. 82.83 million reviews
2. spanning May 1996 - July 2014
3. 24 product categories

We will focus our attention mainly on "Electronics" and "Cell Phones and Accessories", furthermore we will take into consideration only the years from 2012 to 2014.

4.3.1 Some descriptive statistics

4.3 plots reviews against dates and we can see that there has been an upward trend in the two years under consideration. In 4.5 we can see the plot of reviews

```
['case', 221836],
['cover', 134032],
['black', 118281],
['samsung', 90951],
['hard', 77407],
['iphone', 74979],
['g', 73711],
['protector', 67790],
['galaxy', 64506],
['screen', 63284],
['amp', 59128],
['battery', 48959],
['skin', 47437],
['x', 46957],
['charger', 46226],
['usb', 45900],
['apple', 45355],
['phone', 43094],
['cable', 41078],
['inch', 38183]
```

```
['great', 228176],
['one', 197438],
['use', 192429],
['works', 191386],
['good', 179933],
['would', 174754],
['product', 163912],
['well', 148498],
['price', 141582],
['like', 140358],
['work', 129791],
['get', 123671],
['bought', 122939],
['time', 109157],
['easy', 103446],
['buy', 98069],
['nook', 97484],
['used', 97384],
['cable', 96088],
['quality', 87455]
```

Figure 4.2 Common words and their number of appearence, on the left we see common words in product names, on the right common words in products' reviews

against rating and we notice that the distribution is skewed to higher ratings.

4.4 Summary

In this section we gave main characteristics and statistics about the dataset we will be working on and we specify the structure both of Amazon reviews and of how they are represented in our data.
In particular, we gave descriptive statistics on the following: number of reviews by date, number of reviews by rating and how it evolved, lengths of reviews.

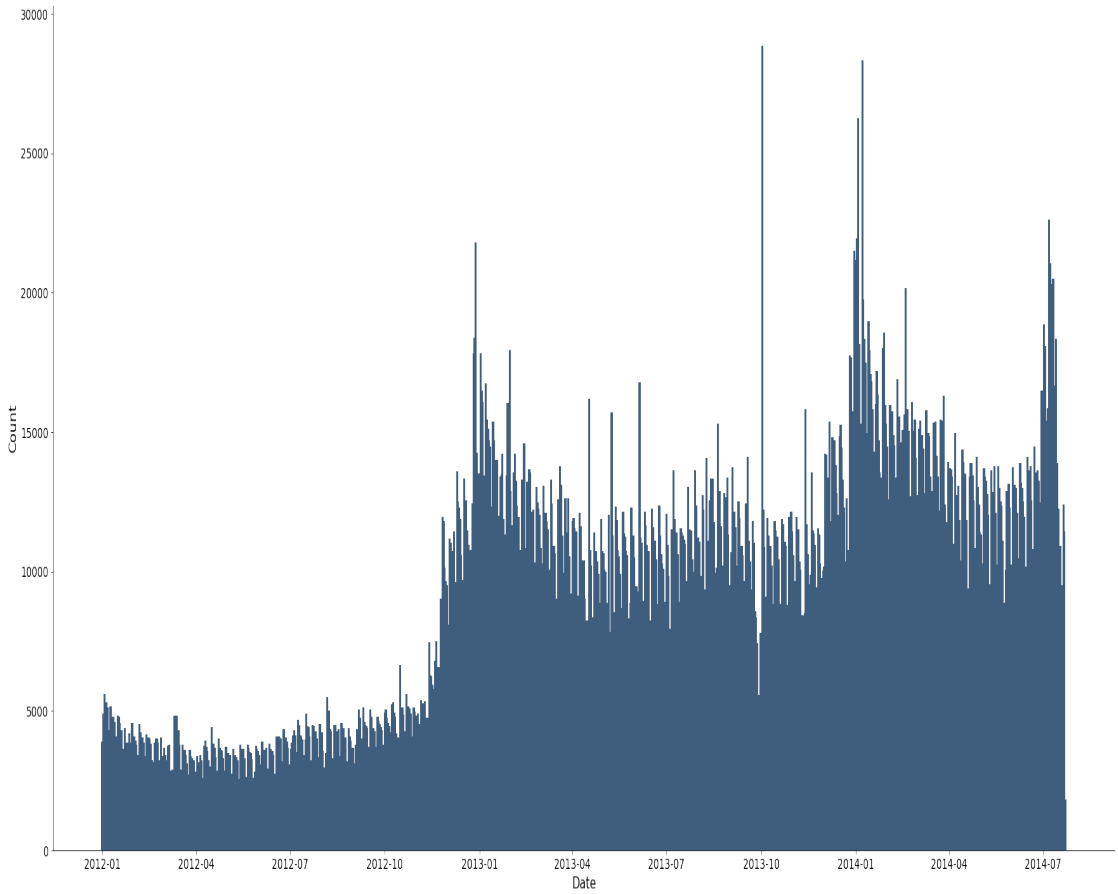


Figure 4.3 "Electronics"+"Cell Phones" reviews by date, 2012/01 - 2014/07

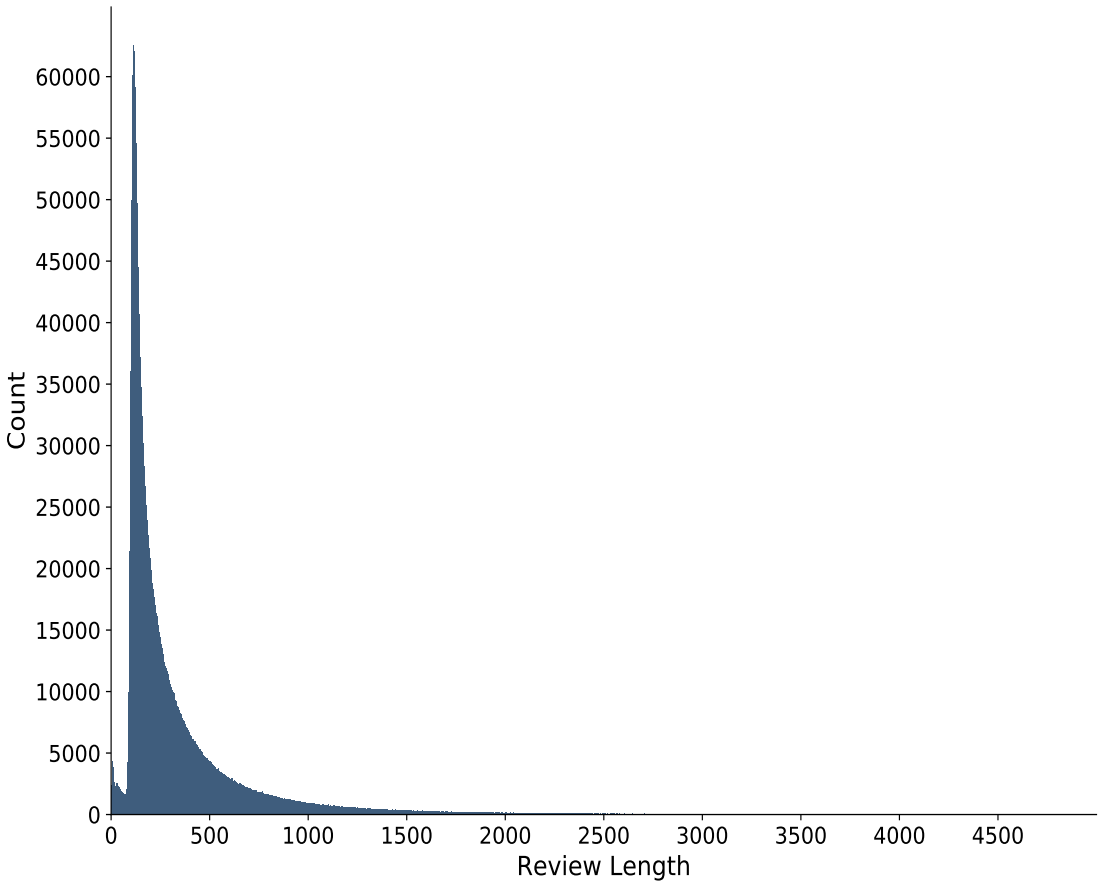
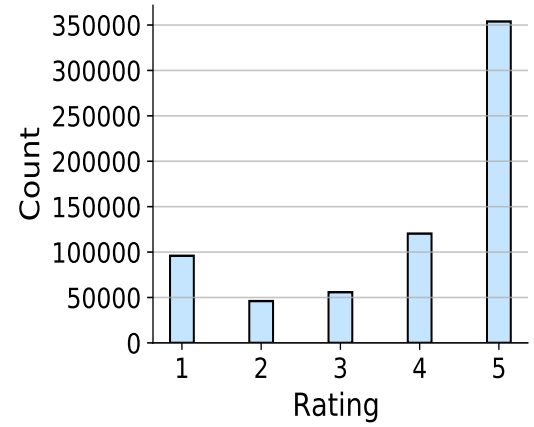
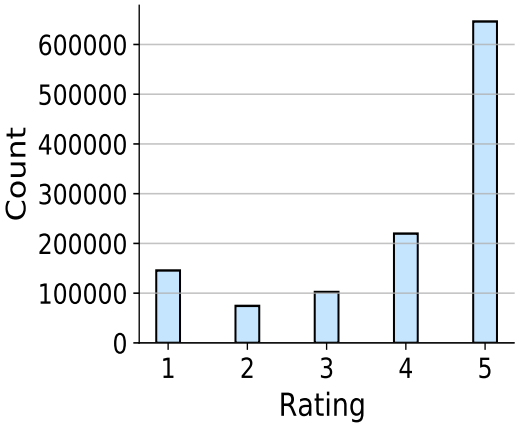


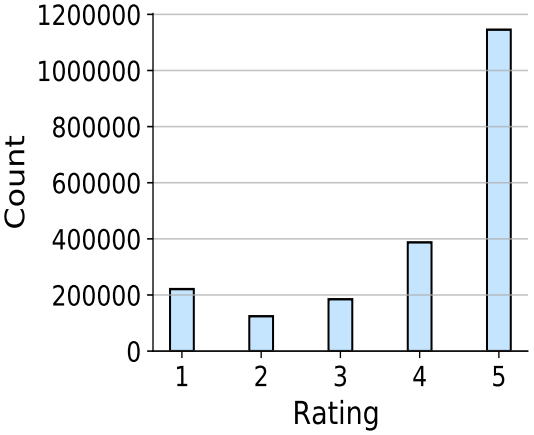
Figure 4.4 Distribution of reviews lengths



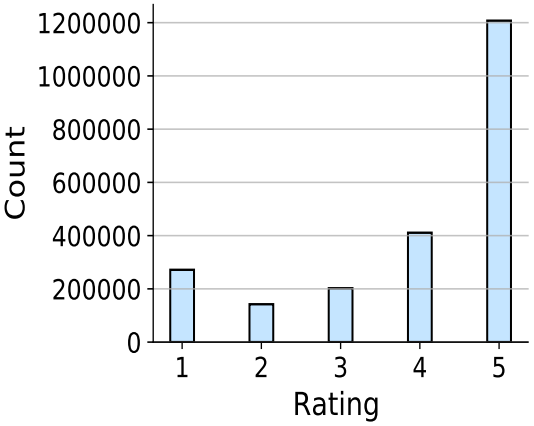
(2012/01/01-2012/07/06)



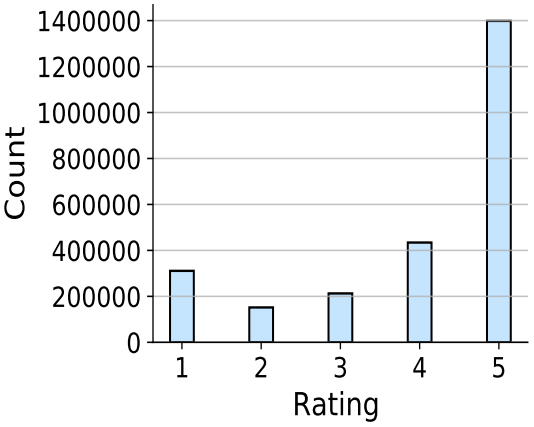
(2012/07/06 - 2013/01/09)



(2013/01/09 - 2013/07/15)



(2013/07/15 - 2014/01/18)



(2014/01/18 - 2014/07/23)

Figure 4.5 Evolution of ratings distribution from 2012/01/01 to 2014/07/23

Chapter 5

Text Clustering

5.1 Proposed approach

This section of our work is split into two separate parts: in the first part we work on product titles, while in the second part we work on product reviews. In the first part we focus more on obtaining the best number of clusters, while in the second part we make a more extensive evaluation of similarities between clusters obtained starting from various text representations, which is something we touch on only briefly in the first part.
In the first part we take the following steps:

- we preprocess and tokenize product titles.
- we vectorize product names according to these three vectorization flows:
  - tf-idf+ SVD
  - Count-vectorization + LDA
  - word2vec + averaging
- for each vector representation we perform K-means clustering and we select best number of clusters with elbow-knee method.
- we compare clusterizations obtained with different vectorization flows using Adjusted Mutual Information score (AMI).

In the second part we take the following steps:

- we preprocess and tokenize product reviews.

- we vectorize product reviews according to the same vectorization techniques used for product titles.
- we perform hierarchical clustering at different granularities and, again, we use AMI score to compare clusterizations.

5.2 Product names clustering

We work on a subset of the dataset, composed of 50,000 product names.

5.2.1 Product Names Preprocessing

Figure 5.1: Example of Product Name

As we can imagine and can see in 5.1, product names are usually quite short strings and do not need much preprocessing.
We will perform only the following simple steps:

- tokenization, splitting at every punctuation mark
- lowercasing
- stopwords removal
- delete "quot" and "amp" tokens, since they are obtained from splitting the encoded version of ampersand and quotes: "&amp", "&quot"

5.2.2 Vectorization of Product Titles

As we can see in 5.2 we follow three different dimensionality reduction workflows and we are going to compare the clusterings obtained with them.

Tf-idf + SVD

In the first dimensionality reduction flow, we start by transforming each document in their tf-idf counterparts. At this point we are left with 58581-dimensional vectors. In order to make clustering feasible we need to reduce the dimension of these vectors, therefore we apply SVD. In particular we set k to 300.

Count-vectorization + LDA

In the second dimensionality reduction flow we transform documents in their bag of words counterparts and then we apply LDA to reduce dimensionality. Given the application, we are not interested in what the topics extracted by LDA are, in fact LDA should enable us to capture various aspect of products, which ideally should be 'product type', 'product colour', 'brand' etc. on the basis of which we will perform clustering
As for SVD we would like to obtain 300-dimensional vectors, therefore we select a number of topics equal to 300 and obtain the representation of documents in terms of the 300 topics extracted by LDA.

Word2vec training + averaging

We are confronted with a vocabulary which is full of uncommon occurences, given that we are dealing with product names. For this reason we prefer not to use pretrained word embeddings and we choose to train new word2vec vectors directly on our corpus. Again, the desired dimension is 300 and this will be the size of the embeddings.
Once we have obtained embeddings, we need a representation for each document. We choose to represent a document as the average of the word vectors corresponding to the words it contains.

5.2.3 K-means

K-means is a clustering technique which usually considers Euclidean distance as similarity metric. Starting from data points and a fixed number k of clusters, its objective is to minimize the Within Cluster Sum of Squares (WSS):

$$_S \sum_{i=1}^{k} \sum_{x \in S_i} (||\mathbf{x} - \mu_i||)^2$$

Since this is a NP-hard problem and is therefore very expensive computationally, some heuristics have been developed, which converge to a local optimum.

Cluster Metrics tf-idf									
Metric	2	100	500	1000	1500	2000	2500	3000	3500
WSS	1.	0.716	0.554	0.495	0.461	0.436	0.415	0.397	0.381
WSSdec	NA	0.283	0.161	0.059	0.033	0.025	0.020	0.018	0.016
BSS	0.044	0.477	0.726	0.818	0.871	0.912	0.944	0.974	1.
BSSinc	NA	0.432	0.249	0.091	0.053	0.040	0.032	0.030	0.025

Table 5.1: WSS Metrics for SVD+tf-idf

Cluster Metrics LDA									
Metric	2	100	500	1000	1500	2000	2500	3000	3500
WSS	1	0.321	0.219	0.186	0.168	0.156	0.147	0.139	0.132
WSSdec	NA	0.678	0.102	0.032	0.017	0.012	0.009	0.007	0.006
BSS	0.249	0.819	0.914	0.944	0.961	0.974	0.984	0.992	1
BSSinc	NA	0.570	0.095	0.029	0.016	0.013	0.009	0.007	0.007

Table 5.2: WSS and BSS metrics for LDA

5.2.4 Cluster number selection

As we said, K means is given in input a number k of clusters. We use the elbow knee method in order to select the optimal number of clusters. The elbow knee method selects the number of clusters after which the WSS decreases very slowly and the Between Clusters Sum of Squares (BSS) increases very slowly: our aim is to have a low WSS, so that points in a cluster are very near each other and a high BSS so that clusters are very well separated. By doing so we minimize inter-cluster variance and maximize intra-cluster variance. After the majority of variance has been explained, the rest is assumed to be noise. We ran the procedures ten times for each vectorizations to ensure consistency of clusterings, given that K-means makes use of random initialization.
In the case of SVD (5.1) we can see there is a strong discontinuity in the rate of WSS decrease and of BSS increase, therefore we can select 500 as best number of clusters.

Even in the LDA case (5.2) and the word2vec case (5.3), k=500 presents a discontinuity both in the rate of WSS decrease and in the rate of BSS increase, therefore it is a good number of clusters.

Cluster Metrics w2v									
Metric	2	100	500	1000	1500	2000	2500	3000	3500
WSS	1	0.539	0.407	0.359	0.331	0.311	0.295	0.281	0.270
WSSdec	NA	0.460	0.131	0.048	0.028	0.019	0.015	0.0134	0.011
BSS	0.236	0.725	0.857	0.906	0.936	0.955	0.973	0.988	1
BSSinc	NA	0.489	0.131	0.049	0.029	0.019	0.017	0.015	0.011

Table 5.3: WSS and BSS metrics for word2vec

Now that we have selected the optimal number of clusters for all vectorizations, we can calculate adjusted mutual information between the partitions obtained by the clusterizations. Our idea is to see whether clusterizations obtained by means of SVD, LDA and word2vec have a certain degree of similarity and, for this reason, can be considered a reliable basis to split products in groups and perform the identification of spam reviews.

5.2.5 Adjusted mutual information between hard clustering

AMI is a variation of mutual information which takes into account the increase of baseline mutual information, that is to say mutual information of two random clusters, when the dimensions of the clusters increase.
Given a set $S = s1, s2, ..., sn$ of n elements and two pairwise-disjoint partitions (hard clusterings) : $U = U1, U2, .., UM$, $V = V1, V2, .., VK$, we are interested to know how much the partitions agree between each other. AMI measures exactly that and it is perfect for our problem, because we are dealing with hard clusterings.

$$AMI = \frac{MI(U,V) - E(MI(U,V))}{max\{H(U), H(V)\} - E(MI(U,V))}$$

where MI(x,y) is the Mutual Information and H(x) is the Entropy.

5.2.6 Clusterings AMI agreements results

Now we can presents the AMI agreement between all the couples of clusterings we took into consideration. In 5.4 we see the results. As we can see in 5.4 there is very good agreement between word2vec and tf-idf and way above chance agreement in the other two cases.

Agreement between clusterings of product titles			
Metrics	tf-idf w2v	tf-idf LDA	w2v LDA
AMI	0.46	0.26	0.26

Table 5.4: AMI between couples of clusterings

Agreement between clusterings of product reviews			
Metrics	tf-idf w2v	tf-idf LDA	w2v LDA
AMI	0.18	0.054	0.53

Table 5.5: AMI scores for product reviews

5.3 Product reviews clusterings

For product reviews we chose to evaluate the evolution of AMI between different clusterizations when varying the number of clusters.

5.3.1 Data

In the previous section we obtained 3 clusterizations of product titles. In this section we consider the clusterization obtained starting from product titles vectorized with tf-idf+SVD. For each cluster we consider the reviews of all its product, provided that the number of reviews is greater than 1000. From the above filtering, we are left with 208 sets of reviews.

5.3.2 Clustering

For each one of the 208 sets of reviews we apply hierarchical clustering. We chose hierarchical clustering, given that it is deterministic, except for tie-breaking, so that a single run is needed in order to ensure consistency. Given a set of reviews and a vectorization we cluster reviews with 10 different granularities: we go from 2 clusters, to $|set\_of\_review|/10$ clusters.

5.3.3 Agreement between clusterings of product reviews

Each score in 5.5 is averaged over all sets of reviews. Furthermore, we averaged over all the granularities, since their scores were very similar.
Results seem to confirm that tf-idf+SVD and w2v+averaging vectorizations give rise to text clusterizations which have substantial agreement in terms of AMI.

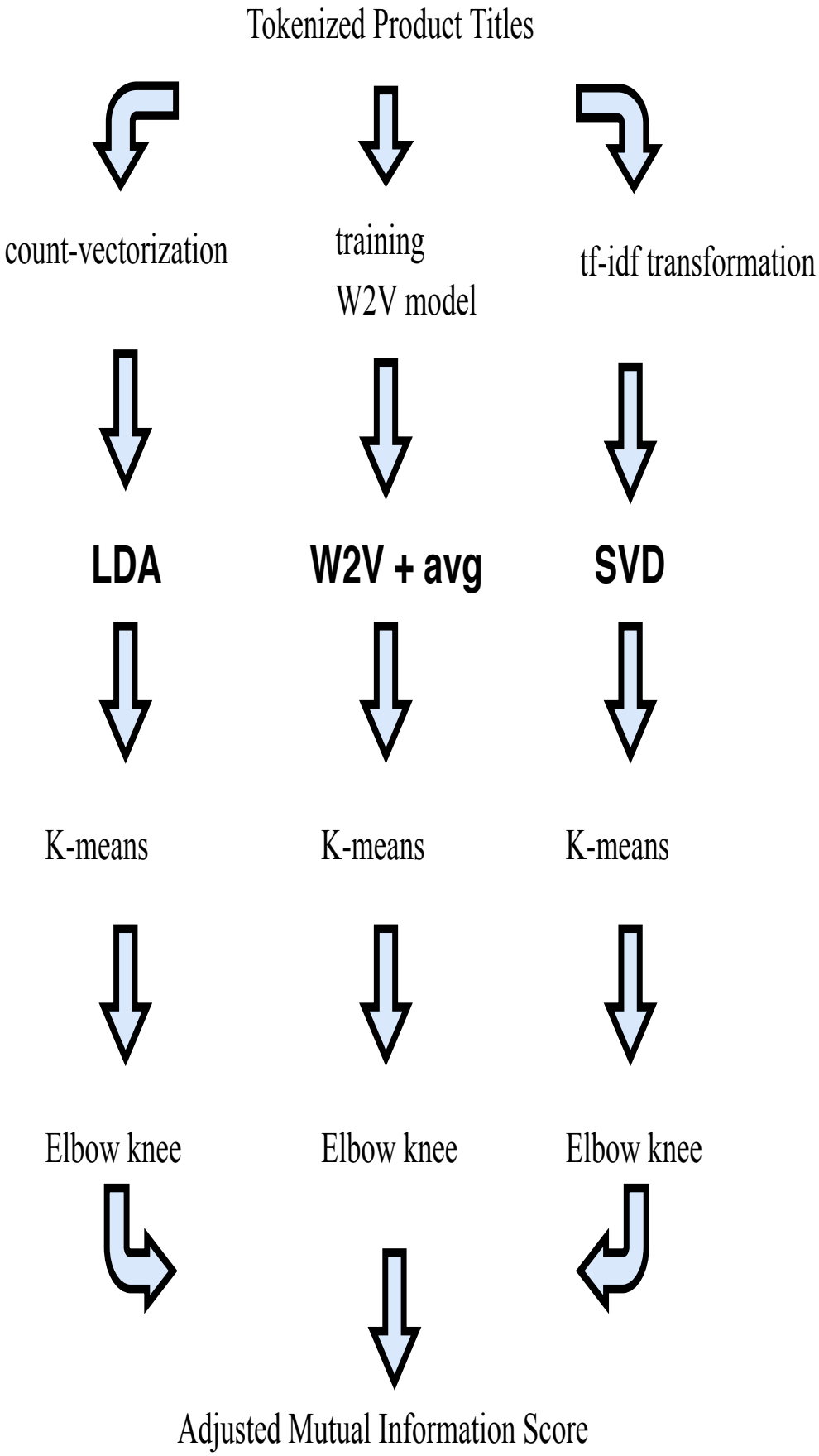


Figure 5.2: The three different Vectorization Processing Flows

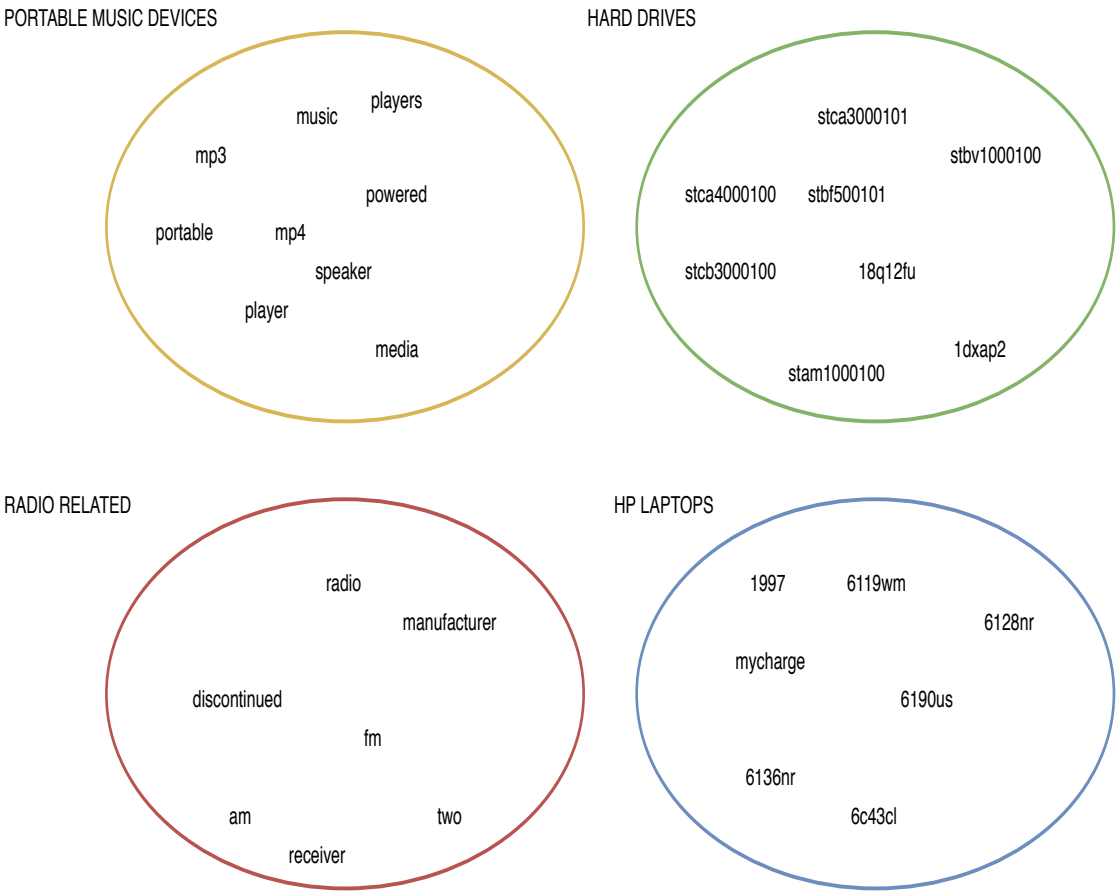


Figure 5.3: Some topics extracted with LDA and most important words

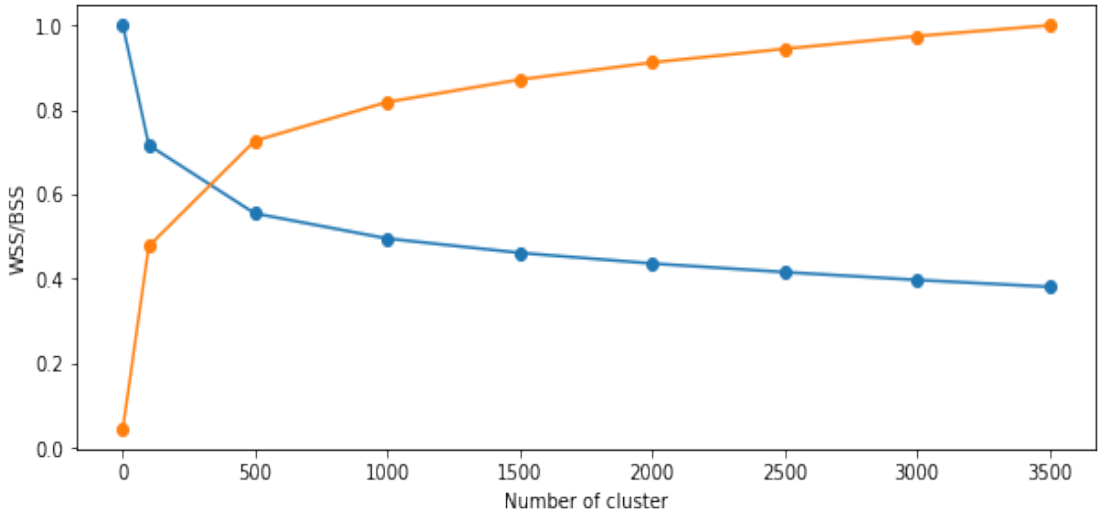


Figure 5.4: Knee elbow graph for SVD+tf-idf

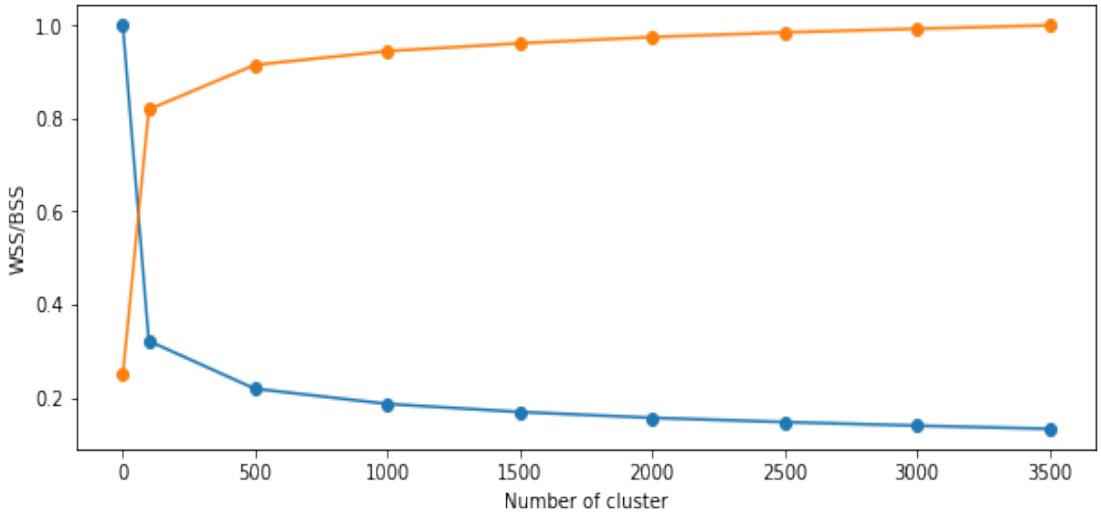


Figure 5.5: Knee elbow graph for LDA

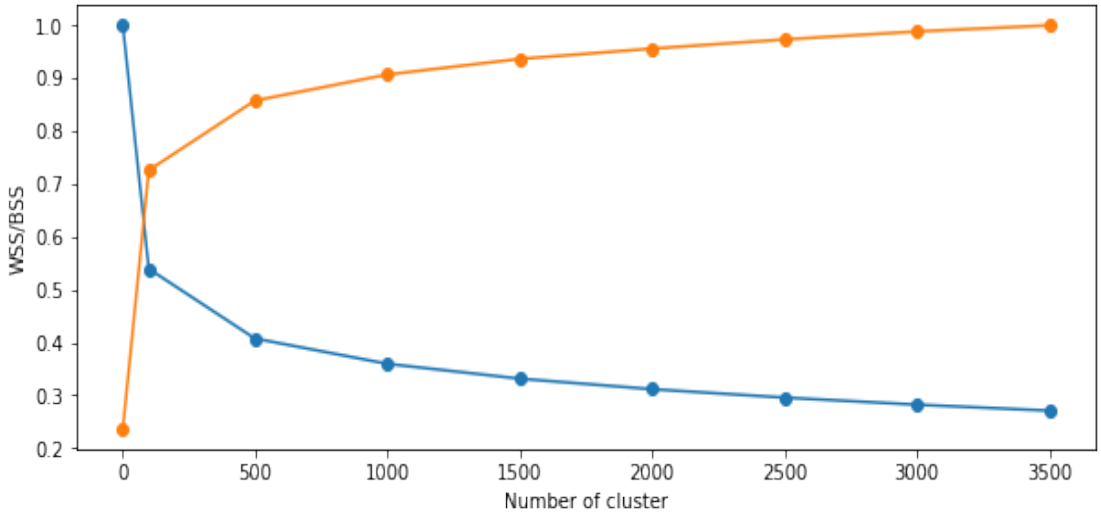


Figure 5.6: Knee elbow plot for word2Vec

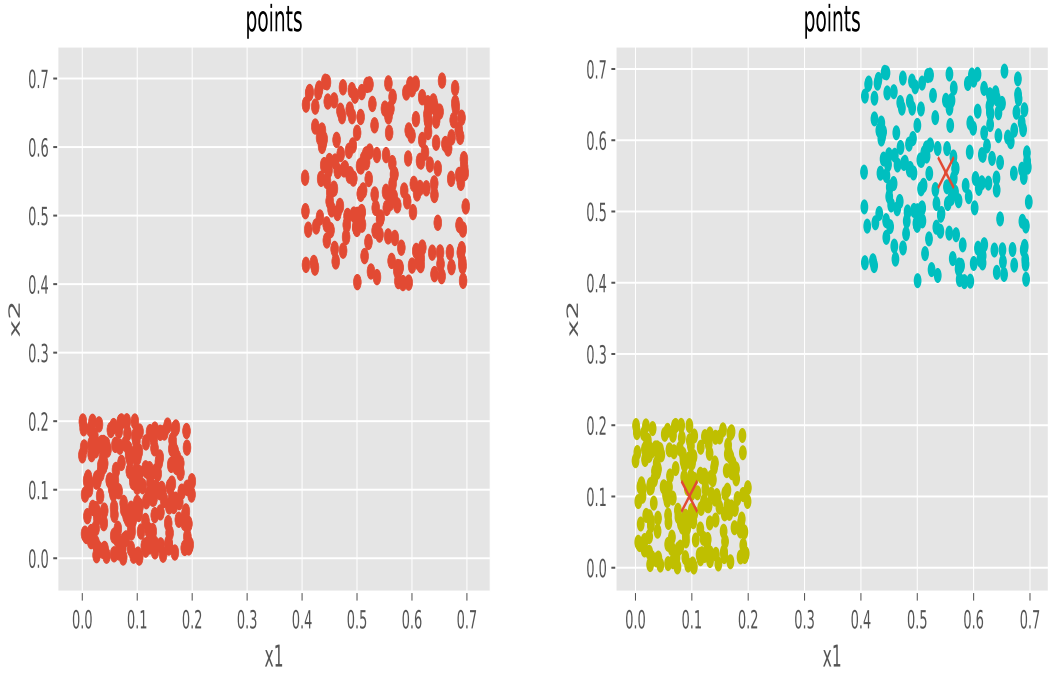


Figure 5.7: Points and K-means clustering

Chapter 6

Opinion Polarity Analysis

For the opinion polarity analysis of our work, we focused on predicting the star rating of reviews, which is interpreted as the sentiment of the review's text and ranges from 1 to 5, where 1 is very negative sentiment and 5 is very positive sentiment. We will compare different approaches: we will start with baselines such as Naive Bayes and SVMs and we will get to Deep Learning approaches. We will see he strengths of both worlds and compare how different Deep Learning architectures perform.

6.1 Workflow description

For this part of our work we will proceed according to standard supervised learning procedure and we will build a train set and a test set. We will use only a part of the Electronics and Mobile Accessories data we have available, in order to speed up training. Our training set consists in 100,000 reviews, while our test set is of 20,000 reviews.
As we said in the introduction, our final aim is to deploy our sentiment analysis engine on a variety of media such as Twitter, online blogs, Facebook pages, online news, Instagram etc., therefore we are interested in building models which are able to generalize to new domains and be robust to noise which is particularly present in Social Networks domain. Given our intentions, we will simulate a setting which is different from training domain. We will proceed according to the following pipeline:

- Standard Training and Test on Electronics and Mobile Phone Accessories.
- Build Train and Test set for this category: Clothing.

- Compare training on a domain + Transfer Learning versus training directly in the domain of interest.

One aspect of our research is the comparison between classical ML algorithms and DL models. In the first part of this chapter we will train and evaluate a wide variety of models, both classic and DL.
Subsequently, we will select a subset of these models to be evaluated in the second (Transfer Learning) step. In the end we will train all of the models for the third step.

6.2 Classic ML algorithms

In the Literature Review section we described two algorithms which are very commonly used in NLP classification problems, that is to say Naive Bayes and SVM.
However, we will not limit ourselves to those. Instead we will consider a variety of ML algorithms such as logistic regression, XGboost, random forests etc.

6.3 Deep Learning

6.3.1 Architectures

We will focus on RNN and CNN. For RNN we will employ the LSTM unit while for CNN we will consider the architecture proposed in Kim, 2014.

6.3.2 Word Embeddings

Our experiments will not make use of pre-trained vectors, but rather both DL architectures will have an Embedding Layer which will be trained together with the rest of the Network.

6.4 Classification Techniques Parameters

In this section we give a detailed description of parameters which are present in some of the applied models.

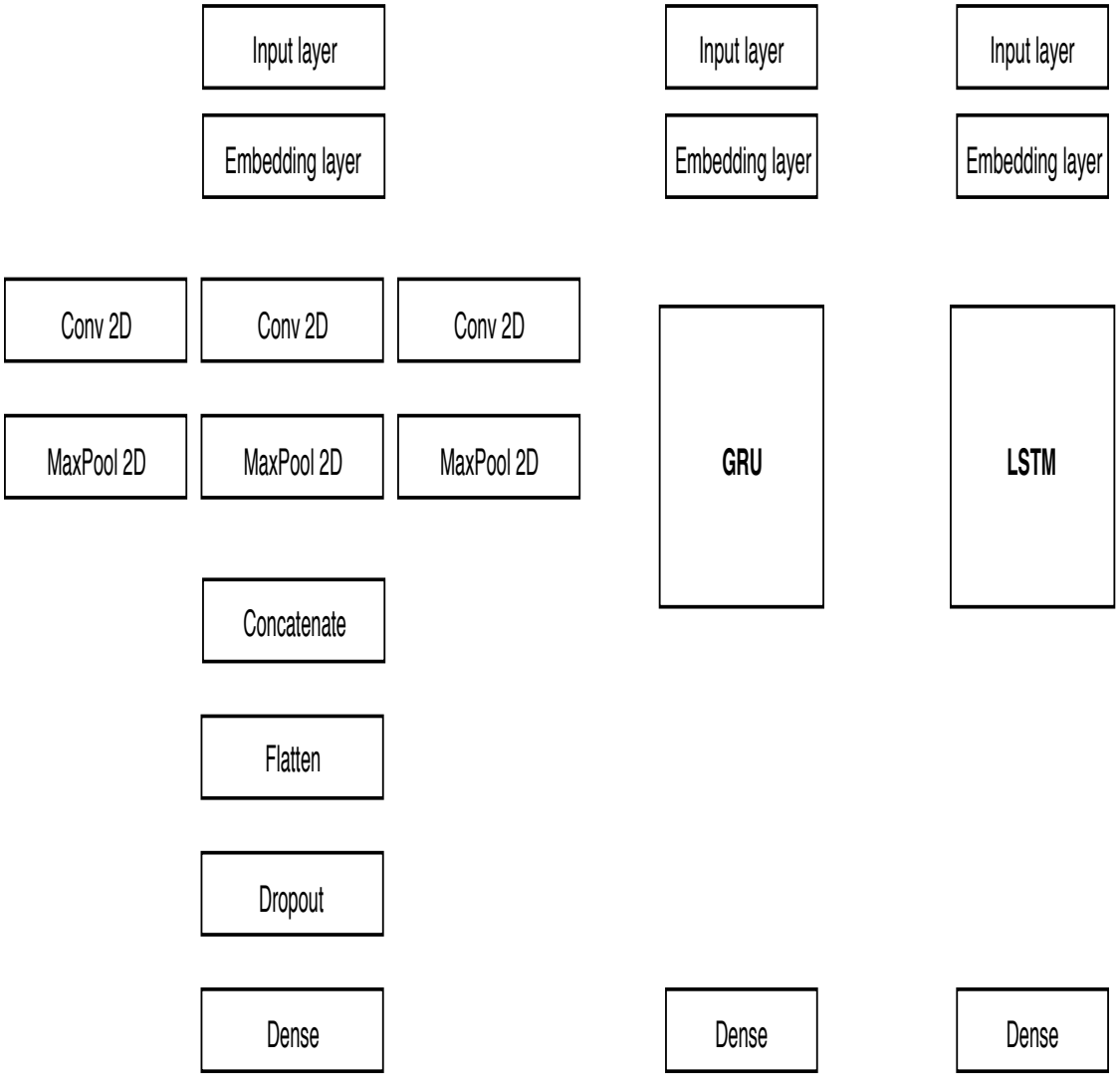


Figure 6.1: Architecture of implemented models

Random Forest

Random Forest has a good of parameters, part of which can be tuned according to how the algorithm works and how the dataset is structured.

- **Number of estimators:**We have seen that Random Forest is an ensamble method. This parameter defines the num- ber of trees to train. Usually, more trees are better but with the increase of the number of learners improvement decreases, increasing trees also impact on the training time, for this reason, we choose 1000 trees.

- **Max depth:**the maximum depth of the tree, this parameter is optional and prevent to trees to grow too much.We decided not to give a max depth.

- **Min sample leaf:** this parameter represents the minimum number of samples required to be at a leaf node. We decide to set 50 as min number of leaf given that a smaller leaf makes the model more prone to capturing noise in train data.

- **Max Features:** this parameter represents the number of features the models look at when searching for the best split. An increase of this parameter usually means an increase in performances, since at each node the number of possible options becomes greater. However the diversity of the trees decreases, they become more correlated and so we may not obtain an increasing this parameter. We decide to set it to $\sqrt{numberoffeatures}$ given that the number of features is quite high.

- **Split Criterion** this function is a measure of the quality. We chose Gini impurity which is a measure of how often we would incorrectly label a random chosen element of the set if we were to label it randomly following the distribution of labels in the subset.

XGboost

Also XGboost has a good number of parameters. Let's see what they are:

- **Learning rate:** this parameters make the contribution of each new base model decrease. We selected XX after cross validation.

- **Max Depth:** It represents the max Depth of a tree. We selected XX in order to avoid overfitting

- **Min sample leaf:** it is the same as in Random Forest

- **Gamma:** represents the minimum loss reduction required to make a split.

- **Objective:** It is the loss function.

6.5 Evaluation

In this chapter we discuss details of evaluation process. We present both algorithms and metrics needed for evaluation.

6.5.1 Cross Validation

The goal of prediction is to be able to generalize results obtained on the the set of data which the model has seen and has been trained on, to unseen data. If we tested the model on the same data we used for training we would obtain a very good score, but this would not be representative of performances on unseen data, therefore we need a different way to assess performances.
Cross-validation works by splitting the dataset n times and evaluate performances on each of these splits.

6.5.2 K-fold

K-fold Crossvalidation is on of the most common kind of Crossvalidation. In this technique the dataset is split in K chunks. Afterwards, each of the K chunks will be used once as a test set. When we select a chunk as a test set, all the other $K-1$ chunks are used as training set.
A good estimate of performances on test set is obtained if we average performances on the K splits.

6.5.3 Evaluation Metrics

Here we give a look at the most significant metrics which are used in Classification problems and we explain our choice.

Binary Classification Confusion Matrix

When dealing with binary classification, given a prediction an the true class label we fall in one of the four cases described by the Confusion Matrix.

F1 for each class		
	Predicted Positive	Predicted Negative
Real Positive	True Positive	False Negative
Real Negative	False Positive	True Negative

Accuracy

Accuracy is the most immediate measure for binary classification, since it gives the ratio of correctly classified samples over all the population:

$$Accuracy = \frac{\sum TruePositive + \sum TrueNegative}{TotalPopulation}$$

Since we deal with a multi-label classification, we are going to redefine accuray as

$$Accuracy = \frac{|C|}{|N|}$$

$$C = i \in N | y_i = \hat{y}_i$$

given that N are the observations, C are the observation which have been correctly predicted, y_i is the real label and $\hat{y}_i$ is the predicted value.

Precision, Recall and F1 measures in Binary Classification

$$Precision = \frac{|TP|}{|TP| + |FP|}$$

As the name says, Precision gives us a measure of how precise classifier prediction arem that is to say how many classified-positive instances are actually positive.

$$Recall = \frac{|TP|}{|TP| + |FN|}$$

Recall on the other hand measures how many of the positive instances are recognized by the classifier.

$$F1measure = 2 * \frac{Precision * Recall}{Precision + Recall}$$

F1 measureis the harmonic mean of Precision and Recall and captures both of them.

Our choice

The task we are dealing with is a multiclass classification. Therefore, we could consider both Accuracy measure and F1 measures. However, our dataset is imbalanced as we can see in figure, therefore we prefer F1 as accuracy would not be really informative.
In particular, for multiclass problem, it is not sufficient to speak of F1 measure. We

have to specify how we deal with the presence of more than 2 classes in the dataset.

6.5.4 F1 for multiclass

F1 in multiclass problem is calculated in one-versus all manner, that is to say TP, FP, FN are calculated as if we were in a binary problem with the considered class versus all the others grouped together.
Main F1 measures for multiclass are:

- micro-F1: which consider consider TP,FP,FN are calculated globally
- macro-F1: which consider each classes separately and then average the F1 scores

However, we would like to obtain a good F1 for each of the classes, since each of the classes is equally important for us. Therefore we consider the F1 for each class separetely.

6.6 Dataset imbalance, undersampling+crossvalidation

By performing crossvalidation on the training set with a variety of models, we see that some classes are really overlooked. This is due to the fact that they are underrepresented. In order to overcome this fact, we implement a crossvalidation with undersampling:

- create all train test splits
- undersample only the train sets, with n samples for each class, where n = number of samples of the minority class in specific train set

We don't want to undersample the test sets, otherwise we would have a completely different distribution from the real test set and this would compromise our results.

As we can see , undersampling, although reducing F1 for Class1 and Class 5, considerably increases F1 for Class 2 and Class 3, so that performances are better overall.

F1 for each class					
Model	Class1	Class 2	Class 3	Class 4	Class 5
MNB	0.626	0.088	0.180	0.332	0.793
BNB	0.570	0.116	0.121	0.270	0.737
RF	0.	0.	0.	0.	0.711
SVM	0.641	0.076	0.175	0.173	0.794

Table 6.1: Performances in normal crossvalidation

F1 for each class					
Model	Class1	Class 2	Class 3	Class 4	Class 5
MNB	0.557	0.256	0.285	0.361	0.693
BNB	0.582	0.211	0.252	0.285	0.747
RF	0.535	0.173	0.242	0.324	0.727
SVM	0.612	0.200	0.282	0.343	0.775

Table 6.2: Performances in undersampling+crossvalidation

6.7 Transfer Learning

First we obtain a small training and test set for the class that we are interested in Clothes. Training and test set will be small, in particular 8000 reviews for training set and 2000 reviews for test set, in order to simulate hand-labelling set. In Twitter, Instagram and other social-network based studies, in fact, we don't have star-rating and the only way to obtain reliable datasets for sentiment analysis is to use hand-labeling, unless some posts presents emoticons, which could be used for automatic sentiment extraction.
In order to evaluate the effect of Transfer Learning we will proceed in two different ways depending on whether the model is or not a DL model. For standard ML models we will proceed in the following way:

- We use partial-fit to train the model on the Electronics dataset
- We use partial-fit to train the model on the small training dataset of Clothes.

For DL Models we do:

- We train models on electronics
- We save weights and reuse them to train model on the small training dataset of Clothes.

After this procedure we will confront the results and we will see how much Tranfer Learning can improve performance on small training datasets and also to what

F1 for each class					
Model	Class1	Class 2	Class 3	Class 4	Class 5
MNB	0.507	0.194	0.298	0.360	0.754
BNB	0.432	0.112	0.187	0.241	0.729
SVM	0.514	0.187	0.281	0.356	0.773

Table 6.3: Undersampling+crossvalidation, NO Transfer Learning

F1 for each class					
Model	Class1	Class 2	Class 3	Class 4	Class 5
MNB	0.412	0.236	0.258	0.324	0.414
BNB	0.478	0.201	0.339	0.283	0.754
SVM	0.331	0.151	0.234	0.292	0.581

Table 6.4: Undersampling+crossvalidation, Transfer Learning

extent the change of domain degrades performances.
In the tables we can see the results of undersampling+crossvalidation both in the case where we train only on the new domain dataset or where we use transfer learning. As we can see, in all crossvalidation experiments by doing Transfer Learning we have an improvement in performance.

6.8 Summary

In this Chapter we focused on Supervised Learning, in particular Sentiment Classification. We started by preprocessing reviews of the Electronics domain, then we went on to try a variety of classic ML models and subsequently we implemented some Deep Learning architectures. We evaluated performances both with standard cross-validation and with undersampled crossvalidation, that is to say undersampling the $K-1$ sets used for training. As we expected, undersampled crossvalidation yielded better performances than normal crossvalidation.
In the second step of this part we experimented with Transfer Learning. We exploited knowledge acquired on Electronics dataset to classify samples from Clothing domain. Pretrained models performed better than models trained only on training set, thus confirming our hypothesis.

Chapter 7

Time Series Prediction: Economic Impact of Reviews

We are interested in time series prediction for the ”economic value” part of our Thesis, so below we will give a brief overview of the method we are going to employ in the process.

7.0.1 Time Series

With the term Time Series we refer to a series of data points indexed in time order. In the most common case it is sequence of points equally spaced in time, a series of discrete data.
Univariate: A single variable varying over time Multivariate: Multiple variable varying over time

7.0.2 Stochastic Process

A stochastic process is a indexed collection of random variables. Stochastic process are used to model time-series data. Since we will be dealing with discrete time-series, we will consider stochastic process models which are indexed by discrete sets.

Weak Stationarity

Weak stationarity is a property of some stochastic processes. It is useful to have it, since almost all weakly stationary process can be modeled by ARMA. The

process X t ;t ∈ Z is weakly- stationary or covariance-stationary if:

$$E[X_t = m_x(t) = m_x(t+\tau)]$$

$$E[(X(t_1) - m_x(t_1))(X(t_2) - m_x(t_2))] = C_x(t_1, t_2) = C_x((t_1 - t_2), 0)$$

7.0.3 White Noise Process

A white noise process is a weakly stationary stochastic process where: $C_x(t_1, t_1 + \tau) = 0 \forall \tau ! = 0$

7.0.4 Moving Average Process

The Moving Average Process is a Stochastic Process which is used to model univariate time series. The value of the output variable depends only on the so called error terms. Error terms are extracted from a Stochastic Process themselves, in particular a White Noise Process.
Depending on the order q

$$X_t = \mu + \varepsilon_t + \vartheta_1 \varepsilon_{t-1} + ... + \vartheta_q \varepsilon_{t-q}$$

7.0.5 Autoregressive Process

The Autoregressive Process is a Stochastic Process which is used to model univariate time series. The value of the output variable depends on a single error terms and on a number of previous values of the output variable itself.

$$X_t = \mu + \varepsilon_t + \vartheta_1 X_{t-1} + ... + \vartheta_q X_{t-p}$$

7.0.6 ARMA Process

ARMA (Autoregressive Moving Average Process) is a model which combines both the AR component and the MA component. For a ARMA(p,q):

$$X_t = \mu + \varepsilon_t + \vartheta_1 X_{t-1} + ... + \vartheta_q X_{t-p} + \varepsilon_t + \gamma_1 \varepsilon_{t-1} + ... + \gamma_q \varepsilon_{t-q}$$

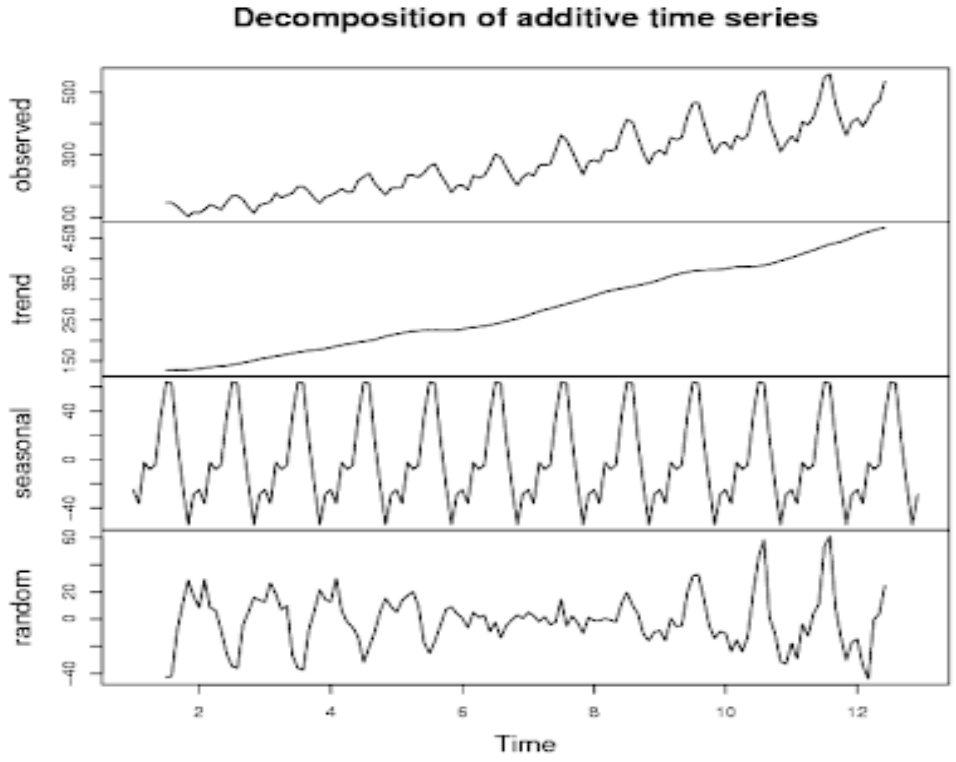


Figure 7.1: TS components

7.0.7 Time Series Components

Time Series found in the real world are usually not stationary. However, if we can find the following components, we can remove them and obtain a stationary one:

- T_t, it is the trend component, a function which can be either linear or nonlinear.
- S_t, it is the seasonal-component, which is a periodic function.
- C_t it is the cyclical component, which represents repeated but non-periodic fluctuations.
- I_t: it is the random component of the TS

Possible ways of combining this component are the additive or multiplicative model. The additive model can be written as:

$$X_t = T_t + C_t + S_t + I_t$$

while the multiplicative model is:

$$X_t = T_t * C_t * S_t * I_t$$

7.1 Evaluation Metrics for Time Series

As in the case of Sentiment Classification we need some quantitative performances indicators. The following are the most used in Time Series Forecasting.

- **MAE:**
$$\frac{\sum_{i=1}^{n} |y_i - \hat{y}_i|}{n}$$
MAE is the average of absolute errors between predicted and true value.

- **MSE**:
$$\frac{1}{n}\sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$
MSE is the average of squared errors beteween predicted and true value.

- **MAPE**:
$$\frac{100}{n}\sum_{i=1}^{n} \frac{|y_i - \hat{y}_i|}{|y_i|}$$
$MAPE$ is the average of normalized absolute errors between predicted and true value.

MAE and MSE are two absolute measures and, therefore, they cannot be used to compare results on different TS, while $MAPE$ consideres the magnitude of the target and is suitable for comparisons between different TS. However, $MAPE$ cannot be calculated if TS has a value equal to 0. In our case MAE and MSE will be used to compare models on the same TS, while $MAPE$, if possible, will be used to compare performances on different TS.

7.2 Our approach

Our objective is to model Google Trends index for a number of queries, in particular we will focus on some of the products which are more presents in our dataset.
Rather than a single model variant we will take into consideration a specific product category and we will aggregate all the products found in the dataset which belong

to the considered category. We focus on two of the top mobile phones brands, which we prefer to address as $brand1$ and $brand2$. For each brand we will consider three categories of products which are designed for phones of those brands, which makes a total of 6 target variables. These are: $(brand1|brand2)cover, charger, battery$. The TS we will consider have daily value for the period comprised 01-01-2012 and 09-27-2012. We chose the following interval, rather than a whole year, because of Google Trends limits. We will compare various models, in particular:

- Naive approach
- ARIMA
- LSTM with various input features

7.2.1 Naive/Persistence Forecast

Naive (or Persistence) Approach is the simplest way to make forecasts. The value of the target variable at time $t + 1$ is assumed unchanged with respect to the value at time t. Therefore, when using this approach, we output the following forecast:

$$\hat{y_{t+1}} = y_t$$

Naive forecast is used as the baseline, in order to evaluate the potential of ARIMA, ARIMAX and LSTM models.

7.2.2 Product extraction

In order to generate the Amazon's review Time Series which we will use as input features, we need to select only the product of interest. We start from the list of products' names and we use regex in order to extract the ones which contain the brand's name and the desired product type. Since a product may be thought for more than one brand, we exclude products name which are thought for both brands.

7.2.3 Product Features extraction

For each product we will extract the following time series:

- Rating 1: Number of reviews with 1-star rating
- Rating 2: Number of reviews with 2-star rating

Number of Products		
Products	Number of Products	Number of reviews 2
brand1 battery	1729	125894
brand1 charger	3846	192337
brand1 cover	24058	210487
brand2 battery	5062	81920
brand2 charger	7545	81571
brand2 cover	37863	281556

Table 7.1: Number of considered products and number of considered reviews

Average values								
Products	Rating 1	Rating 2	Rating 3	Rating 4	Rating 5	Rev Count	Num of Products	Num of Users
brand1 battery	4.92	1.778	1.75	4.64	14.43	27.53	14.05	19.70
brand1 charger	10.28	3.22	3.33	6.91	21.36	45.12	26.51	33.24
brand1 cover	9.041	5.08	6.40	9.96	22.61	53.10	46.50	50.02
brand2 battery	3.144	1.59	1.62	3.34	11.18	20.90	18.69	19.51
brand2 charger	3.62	1.58	1.92	3.71	12.03	22.89	20.50	21.47
brand2 cover	8.57	5.65	7.04	12.22	32.20	65.69	59.98	61.45

Table 7.2: Average of features for each product

- Rating 3: Number of reviews with 3-star rating
- Rating 4: Number of reviews with 4-star rating
- Rating 5: Number of reviews with 5-star rating
- Review Count: Number of reviews written
- Number of Products: Number of different products reviewed
- Number of Users: Number of different Users who wrote a review

Each feature has daily frequency.

Performances: MAE and MSE						
Products	PM MAE	ARIMA MAE	LSTM MAE	PM MSE	ARIMA MSE	LSTM MSE
brand1 battery	4.4875	**4.255**	6.99	**41.5375**	57.157	50.12
brand1 charger	6.617	**6.044**	7	**83.283**	97.35	133.3
brand1 cover	8.11	**7.216**	11.7	100.03	**99.22**	200.4
brand2 battery	10.1625	7.406	**7.31**	153.1875	91.898	**78.993**
brand2 charger	15.96	12.38	**11.64**	376.76	**197.79**	227.78
brand2 cover	15.925	**11.23**	12.29	365.8	205.6	**205.39**

Table 7.3: Performances of different models on considered products

7.2.4 Target Feature extraction

For the LSTM model, together with the above mentioned features, which we extract from Amazon's reviews dataset, we also add two TS features which are derived from lagged target variable, specifically:

- Target Variable Peaks
- ARIMA Forecast

Target Variable Peaks (TVP) feature is a way of detecting whether there has been a sharp increase in Target Variable at time t with respect to its average value in the previous time steps, considering a fixed size time window.
ARIMA Forecast is fed into the LSTM, so that we should be able to harness the already good predictive power of that model and combine it with the Amazon's reviews feature to witness, hopefully, an improve in forecasting accuracy.

7.3 Train and Test set

After having collected all the needed features, we split each Time Series into Train and Test set. Train/Test split is done in a 70/30 fashion: Train data goes from 1/1/2012 to 07/06/2012 while Test data goes from 07/07/2012 to 09-27-2012.

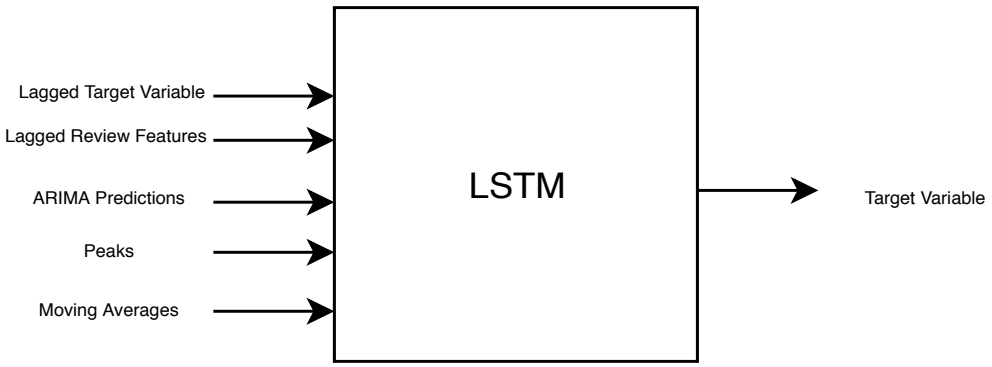


Figure 7.2: LSTM model for economic value prediction

7.4 Evaluation

In 7.3 we are can see the results of the three models we considered. ARIMA usually delivers better performances than the Persistence model baseline and is the best model overall. Our model, despite performing better on some data (brand2 battery MAE and MSE and brand2 charger MAE), does not seem to give us a substantial edge at predicting Google Trends volume index.
Although it may be that Amazon reviews have no predictive power with respect to Google Trends volume index, it may also be the case that what has prevented us from extracting the most value out of them is to be found one of the following factor:

- We considered groups of products rather than single products and the groups may contain too many and diverse products
- We did not choose the right granularity (performances on weekly, monthly must still be evaluated)
- We did n
- We focused on a single domain and we did not explore different kind of products
- Google Trends do not reflect product sales for the products we considered

For the above reason, it still may be worth to inquire further into the predictive power of Amazon reviews on sales data.

Chapter 8

Conclusions and Future Works

In this work we have dealt with online reviews and we have tried to extract economic values from them, given that we recognize the importance they have assumed, both for companies and for customers. Our initial idea was to predict sales data of specific products and assess the prediction power of Amazon.com reviews in this setting. We recognized the difficulty of accessing sales data, given their often proprietary nature, and we chose to use Google Trends query volumes as a proxy for them.

While working on TS data we identified two other tasks which we considered worth of attention. These were: Text Clustering and Transfer Learning for Sentiment Classification. The first step in our work was to review the Literature of the fields we worked in: Text Clustering, Opinion Mining and Time Series Forecasting. After studying main approaches in the Literature, we were ready to tackle the different tasks.

As for Text Clustering we divided the process into two steps: first we focused on clusterization of product titles, subsequently we considered product reviews. For the product titles step, we intended to find the best number of clusters, in fact this step was meant as a way to limit the application of spam reviews detection procedures only to limited groups of products. With the help of 3 methods, we were able to identify a good number of clusters, which, for all 3 of them, was 500, Furthermore, we noticed there was a good level of agreement between the tf-idf+SVD and the word2vec+Average clusterization (0.46 AMI).

This prompted us to inquire further whether it is common for these two vectorizations to yield similar clusterizations. In order to do so we considered each set of reviews obtained after the product clustering phase and we applied Hierarchical clustering to obtain clusterings at vairous granularities. Again we witnessed a good average

agreement between the considered vectorizations: 0.18 AMI.
For the Opinion Mining part, we started off by working on the single domain dataset. We worked in the Supervised Learning Framework, specifically Sentiment Multiclass Classification. The dataset was unbalanced and, consequently, we decided to apply undersampling to the training set in order to increase performances. As we expected, undersampling helped us increase performances, which were evaluated on a single-class basis: F1 for each class was evaluated, since aggregated measures would have hidden low scores for under-represented classes. Deep Learning models, LSTM and CNN, outperformed more traditional Machine Learning approaches, but traditional ML models are to be preferred if we want a more explainable behaviour.
From here we move on to the Transfer Learning part, where we evaluated how the different models are able to exploit knowledge acquired in a pre-training phase so as to classify samples extracted from a domain which has a very limited number of training samples. The pre-training domain we chose was reviews of Electronics, while the Target domain was Clothing. DL outperformed standard ML even in this setting and, more importantly, we witnessed a substantial increase in performances when pre-training models on the Electronic dataset. Given the difference of domains, we are led to believe that pre-training models on multidomain dataset should boost performances on datasets with usually small training datasets such as online news or blogs.
As for the Time Series part, we tried to assess the predictive power of Amazon.com product reviews with respect to the Google Trend search volume index. We started by evaluating the Persistence model baseline in terms of MAE and MSE, then we proceeded with evaluation of ARIMA model. Last, we defined a DL model, which combined Amazon.com reviews features, peaks detection and ARIMA forecasts. Our model did not yield better results than ARIMA but, for a series of reason which we exposed, further inquiry into the predictive power of Amazon.com reviews may still be worth.
The evaluation of Amazon.com predictive power on real sales data is one possible extension of our work, together with: the addition of different DL architectures(e.g. Generative Adversarial Networks) for the Sentiment Classification task, evaluation of the different vectorization flows on text clustering of different datasets, Transfer Learning from Amazon.com reviews to online blogs or Social Networks.
The work done so far has given us the possibility of understanding how much value can be extracted from online reviews and how versatile NLP tools and especially ML ones are when tackling the most varied of challenges.

Acronimi

NLP	Natural Language Processing
BSS	Between Clusters Sum of Squares
WSS	Within Clusters Sum of Squares
OM	Opinion Mining
ML	Machine Learning
WE	Word Embeddings
SVD	SVDSingular Value Decomposition
OM	Opinion Mining
SA	Sentiment Analysis
NN	Neural Network
RNN	Recurrent Neural Network
CNN	Convolutional Neural Network
BOW	Bag of Words
W2V	Word2Vec
LDA	Latent Dirichlet Allocation
SVD	Singular Value Decomposition
AMI	Adjusted Mutual Information

Bibliography

Aue, Anthony and Michael Gamon

2005 "Customizing sentiment classifiers to new domains: A case study", in *Proceedings of Recent Advances in Natural Language Processing (RANLP)*, 3.1, Citeseer, vol. 1, pp. 2-1. (Cit. on p. 29.)

Blei, David M, Andrew Y Ng, and Michael I Jordan

2003 "Latent dirichlet allocation", *Journal of machine Learning research*, 3, Jan, pp. 993-1022. (Cit. on p. 25.)

Brightlocal

2018 *Local Consumer Review Survey, Online Reviews Statistics Trends*, https://www.brightlocal.com/learn/local-consumer-review-survey/#Q1,2018., [Online; accessed 09/01/2018]. (Cit. on p. 1.)

Choi, Hyunyoung and Hal Varian

2012 "Predicting the present with Google Trends", *Economic Record*, 88, pp. 2-9.

Daume III, Hal and Daniel Marcu

2006 "Domain adaptation for statistical classifiers", *Journal of Artificial Intelligence Research*, 26, pp. 101-126. (Cit. on p. 29.)

Esuli, Andrea and Fabrizio Sebastiani

2005 "Determining the semantic orientation of terms through gloss classification", in *Proceedings of the 14th ACM International Conference on Information and Knowledge Management*, ACM, pp. 617-624. (Cit. on pp. 26, 27.)

Facebook users worldwide 2018

2018 https://www.statista.com/statistics/264810/number-of-monthly-active-facebook-users-worldwide/.

Glorot, Xavier, Antoine Bordes, and Yoshua Bengio

2011 "Domain adaptation for large-scale sentiment classification: A deep learning approach", in *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 513-520. (Cit. on p. 30.)

Google's search knows about over 130 trillion pages

2016 https://searchengineland.com/googles-search-indexes-hits-130-trillion-pages-documents-263378.

Harris, Zellig S

1954 "Distributional structure", *Word*, 10, 2-3, pp. 146-162. (Cit. on p. 8.)

He, Ruining and Julian McAuley

2016 "Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering", in *Proceedings of the 25th International Conference on World Wide Web*, International World Wide Web Conferences Steering Committee, pp. 507-517. (Cit. on p. 32.)

Hochreiter, Sepp and Jürgen Schmidhuber

1997 "Long short-term memory", *Neural Computation*, 9, 8, pp. 1735-1780. (Cit. on p. 20.)

Hu, Minqing and Bing Liu

2004 "Mining and summarizing customer reviews", in *Proceedings of the tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, pp. 168-177. (Cit. on pp. 27, 28.)

Jindal, Nitin and Bing Liu

2008 "Opinion spam and analysis", in *Proceedings of the 2008 International Conference on Web Search and Data Mining*, ACM, pp. 219-230.

Joulin, Armand, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov

2016 "Bag of Tricks for Efficient Text Classification", *arXiv preprint arXiv:1607.01759*. (Cit. on p. 10.)

Khandelwal, Urvashi, He He, Peng Qi, and Dan Jurafsky

2018 "Sharp nearby, fuzzy far away: How neural language models use context", in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 284-294. (Cit. on p. 20.)

Kim, Yoon

2014 "Convolutional neural networks for sentence classification", in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1746-1751. (Cit. on p. 50.)

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton

2012 "Imagenet classification with deep convolutional neural networks", in *Advances in Neural Information Processing Systems*, pp. 1097-1105. (Cit. on p. 20.)

Kuang, Da, Jaegul Choo, and Haesun Park

2015 "Nonnegative matrix factorization for interactive topic modeling and document clustering", in *Partitional Clustering Algorithms*, Springer, pp. 215-243. (Cit. on p. 26.)

Liu, Bing

2012 "Sentiment analysis and opinion mining", *Synthesis Lectures on Human Language Technologies*, 5, 1, pp. 1-167. (Cit. on p. 29.)

Liu, Tao, Shengping Liu, Zheng Chen, and Wei-Ying Ma

2003 "An evaluation on feature selection for text clustering", in *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, pp. 488-495. (Cit. on p. 26.)

Mikolov, Tomas, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean

2013 "Distributed representations of words and phrases and their compositionality", in *Advances in Neural Information Processing Systems*, pp. 3111-3119. (Cit. on p. 10.)

Pang, Bo, Lillian Lee, and Shivakumar Vaithyanathan

2002 "Thumbs up?: sentiment classification using machine learning techniques", in *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing-Volume 10*, Association for Computational Linguistics, pp. 79-86. (Cit. on p. 28.)

Pennington, Jeffrey, Richard Socher, and Christopher Manning

2014 "Glove: Global vectors for word representation", in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1532-1543. (Cit. on p. 10.)

Socher, Richard, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts

2013 "Recursive deep models for semantic compositionality over a sentiment treebank", in *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pp. 1631-1642.

Sparck Jones, Karen

1972 "A statistical interpretation of term specificity and its application in retrieval", *Journal of Documentation*, 28, 1, pp. 11-21. (Cit. on p. 9.)

Tai, Kai Sheng, Richard Socher, and Christopher D Manning

2015 "Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks", in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*, pp. 1556-1566.

Turney, Peter D

2002 "Thumbs up or thumbs down?: semantic orientation applied to unsupervised classification of reviews", in *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, Association for Computational Linguistics, pp. 417-424. (Cit. on p. 27.)

Xu, Jiaming, Peng Wang, Guanhua Tian, Bo Xu, Jun Zhao, Fangyuan Wang, and Hongwei Hao

2015 "Short Text Clustering via Convolutional Neural Networks.", in *Proceedings of the 1st Workshop on Vector Space Modeling for Natural Language Processing (VS@HLT-NAACL)*, pp. 62-69. (Cit. on p. 26.)