UNIVERSITÀ OF SIENA
Department of Information Engineering (DII)

Dottorato di Ricerca in Ingegneria Informatica - XVIII Ciclo

Time Series Analysis of Textual Data.

Candidate
Marco Turchi

Advisor
Prof. Marco Maggini

A. A. 2004/2005

Acknowledgements

I want to dedicate this work to my family, Ergense, Maria, Stefania, Marco and Giovanna. They helped and supported me in all these years, in particular in the last period. Without their words anything would have been possible. I hope that my niece enjoys her time in studies as I have done in mine, and she will have the courage to choose always the most difficult way.

I would like to thank my advisor Prof. Marco Maggini who supported me in reaching this goal and encouraged my research in this field. I am also grateful to Prof. Marco Gori, whose experience has been very important for my studies. They supported my choices and gave me the opportunity to study abroad. I can not forget the senior people of my group, Monica Bianchini, Franco Scarselli and Edmondo Trentin, they helped me to grow, and gave me helpful suggestions, when the road ahead was foggy.

I thank all my Italian and American colleagues/friends Leonardo Rigutini, Lorenzo Sarti, Michelangelo Diligenti, Irina Branovic, Gabriele Monfardini, Marco Ernandes, Giovanni Angelini, Ernesto Di Iorio, Augusto Pucci, Quan Le, Marco Stefani, Dragomir Yankov, Gunes Erkan, Rafael Diaz and Margherita Bresco. I spent very nice and interesting time with all these guys, talking about serious arguments, like sports, women, and funny topics like research.

I forgot them in my degree thesis acknowledgments, and I do not want to do the same here. How could I forget the four musketeers of my life: Alessandro, Nicola, Francesco e Carmelo. We have different lives, we are far from each other, but our friendship get over these things and our adventures are a part of me.

During my period in Davis, I met a lot of friends, and I would like to thank some of them. Roberto, our unforgettable sushi dinners, dreaming fiorentina

steak and Italian wine, Andrea and Isabel, Giuseppe, Marina, Elisabetta, Marco, and Ebrima. All these guys helped me to feel at home.

Arianna has entered in my life in the last months, but she has been with me during this period, all the nights in front of the computer, supporting me.

At the end, I would say thanks to all the people who have tried to hinder me: your behavior have only strengthened me.

At last a special thank goes to Prof. Nello Cristianini. He hosted me at Davis, helped me to produce this work and improved me as a person. I met him the first time in Siena on July 2004, that meeting has changed my life.

Preface

"If there is too much data to move around, take the analysis to the data" with this sentence Jim Gray, ACM Turing Award 1998, started a talk last year at the Computer Science Department in UC Davis. These words gave the attestation that the direction that we had taken was correct.

This work moved from two thoughts that buzzed in my mind from much time. The main idea is very close to Gray's sentence: data is there, and we have only to start to use it. In the last twenty years, the Web has grown so fast that at the moment the most popular search engines are not able to crawl all the available documents. Till fifteen years ago, libraries were the most important sources of documents, now the users look at the Web as the principal tool to find relevant information. Everyone writes email, uses instant messages, newsgroups or blogs, publishes his/her own web page. Millions of companies publish balance sheets, technical reports or buy free space for their advertisements on the Web. Each government produces official bulletins every day. These data can feature different formats. Usually, we think data as documents or sequences of words, but this is not the only case. They can be everything, a sequence of numbers, letters, words, phrases, paragraphs and so on. They can be long sequences, like a book chapter or a DNA sequence, or short like the temperature on a weather forecasting site. In some contexts, an new alphabet can be defined and new structures can come out. An avalanche of data becomes available every second, and only a little part of it have been analyzed or processed and shown with a new apparel.
What can we do? To be honest, in many cases we do not actually know, but saying "we do not know" is equal to say "we can do everything". We guess

that with a little bit of fantasy a lot of potential useful applications will come out. Sometime novel ideas produce phenomena that change people's habits, like it happened with the growth in popularity of Google or SkyPe.

The second motivation arose from the fact that in my quite short study/work carrier I had dealt with many "classical" documents. Despite the direction taken to face the task of automatic processing of this kind of fata, every time we end up with the problem of selecting the best features to represent the documents. A lot of efforts have been spent to find the best representation cutting, merging, shaking features in all possible ways.
What would happen if someone gave us just the relevant content of each document? It would be a kind of magic!!! In some contexts this magic can actually happen. In news, blogs, newsgroups, each item is short, is dense, and only important words and sentences are there. Journalists, moderators and administrators filter the information published in these environments, select the right words to attract people; in other words they perform feature selection.

This work picks up these two ideas, or, how we prefer to call them, challenges. My thesis tries to provide two "simple" proposal that would like to be two original ways to face text analysis.
The first problem we studies is somehow loosely related to practical application, but represents an interesting aspect in human language theory, especially in the era of the "global" society. Nowadays, just walking down the street, different languages arrive to our ears. In a multiracial society, we meet people which come from different cultures. How different languages develop, differentiate with respect each other, is an interesting field in linguistic. These problems motivated part of our reasearch that aimed at the study of the *Indo-European language evolution*. In chapter 3, we propose some statistical and computer science techniques to find the phylogenetic trees of the Indo-European family. The Web provides us with the dataset: the translation of the Human Right Constitution in 50 different languages. Our experimental results are totally comparable with the trees obtained with classical linguistic approaches. The main difference is the type of data used;

in fact linguists select a sequence of specific words. In particular these terms are related with things that are related to everyday life of the human beings, like sun, earth, water, mother and so on. Otherwise, in our approach we do not use any selected words, but entire documents that have been downloaded from the Web.

The second main proposal in this thesis is more related to practical applications on the Web. Web-mining is becoming a hot topic of research because it can provide tools to analyze the "trends" of the information available on the Web exploiting the huge amount of data that is published everyday. In particular we focused our attention on news. News are an example of short documents which contain the most significative sentences and words. Someone has read a complete newspaper article, a bulletin, a story and he/she has extracted a short summary. From a broad point of view, news show a continuous shift in their characteristics. Every hour, on-line newspapers or press agencies, like UsaToday, Cnn, International Herald Tribune or Fox News, refresh the information creating a "clean" and interesting time series of data. In general, there are some periods when media are attracted by some particular world events. In that period, they create data that deal mainly with which argue about these events. This phenomenon could be detected inside the news, because many terms that are semantically correlated appear. We suppose that all these connected words can be emitted from a common source of terms. When another important event occurs, media change their focus of attention, and they start to produce different terms, and this is the sign that the sources have changed. Our goal is to detect when these changes happen. Besides, inside these sequences of news there are some trends. They are represented by continuous periods of time, where a given word appears, grows and disappears. In chapter 4, we present some approaches that are able to detect change points and most significative events in a flow of news. This work merges statistical and computer science techniques trying to emphasize both the disciplines.

The keyword of both the works is *time*. In fact, it is the hidden variable that guides the evolution, that scans the news and that synchronizes the events. Time is used in different ways in both the approaches. In fact, in

language evolution from documents we reconstruct the development in the past to explain the present, while in news trend analysis time allows us to spot important events and the related characteristics.

In general, this thesis describes my efforts to apply new and old techniques, to improve my skills, and provide a "little" contribution to the state of art of text mining. The hope is that I was actually able to achieve this goal. This work represents the end of my doctorate period of study shared between Italy and America, and it has inside something from both countries and from all people that have supported me.

Finally a personal comment after analyzing 8 months of worlds events it seems there are very few positive events, and too many negative ones. I hope I was just unlucky in choosing my dataset... but I am not so sure.

Contents

Chapter 1

Time Series Analysis

1.1 The Problem

The sentence "Time Series Analysis of Textual Data" merges two different worlds: Statistics and Text Analysis. These fields have worked together in a lot of applications, in particular many statistical methods have been exploited in text analysis during its development in the last years. Our work merges these worlds again, and it was born from the idea of introducing the variable *time* inside textual data representation.

This necessity was due to the fact that textual data evolve, change, disappear, is created everyday, and sometimes also every hour, and all these fluctuations can be exploited to extract more information than in the normal text analysis approaches. News and blogs are only an example of how fast data can evolve.

Another interesting characteristics of *time* in textual data consists on the capacity of documents to contain hidden information that is related to the temporal evolution of languages. It is little bit weird, but there are some tasks, in which it is possible to discover temporal relations between "objects" by analyzing contemporary documents. An example is language evolution, where it is possible to reconstruct the evolution tree of languages using particular words or nowadays documents.

Our work represents a possible approach to face these two aspects of time inside textual data. Time series are the instrument used to analyze

and formalize these problems. The idea is to create time series associating each observation with documents in different instants of time. We will show in the next chapters how this is possible. When the time series have been constructed, different approaches can be applied on it. We have analyzed them with the aim of discovering change points and the optimal segmentation in a flow of news, or for reconstructing the evolution tree of a set of correlated languages.

In the first case, we suppose that the same focused source of words produces terms on a particular topic till the time series does not show an evident drift. When a change point arrives in the time series, it means that new words have been produced and hence the source changed. A segment can be seen as the period during which a source produces its words.

In the second case, the tree is the representation of the time series. In fact, each node has a correlation with a particular instant of time, and the length of each branch identifies the flow of time.

In the next sections, we will introduce the theory at the basis of time series analysis and the well known standard problems of change points and segment detection. We will give a brief introduction on the state of art of these problems.

1.2 Introduction to Time Series Analysis

A time series is a collection of observations made sequentially through time. Many time series are routinely recorde in economics, finance (the price index series), in the physical sciences (rainfall on successive days), marketing (sales figure in successive weeks or months), demographic, process control data, and so on.

A time series is said to be *continuous* when the observations are made continuously through time, while it is *discrete*, when observations are taken only at specific time instants, usually equally spaced. In the following of this thesis, we consider discrete time series. Discrete time series can arise in several ways. Given a continuous time series, we can simple the value

at equal intervals of time to yield a discrete time series, called a sampled series. Obviously, this approach produces a loss of information, that depends on sampling rate. We can have a different type of discrete time series when a variable does not have an instantaneous value, but we can aggregate its values over equal intervals of time.

The special feature of time series analysis is the fact that successive observations are usually not independent on each other and that the analysis must take into account the time order of observations. When successive observations are dependent, future values may be predicted from past observations. If a time series can be predicted exactly, it is said to be *deterministic*. Most time series are *stochastic* in that the future is only partly determined by past values, so that exact predictions are impossible and must be replaced by the idea that the future values have a probability distribution, which is conditioned by the knowledge of the past values.

Time series can be analyzed under different points of view, *description*, *explanation*, *prediction* and *control*. If we are interested to study the *description* of time series, the time plot is a good instrument to do that. We are able to obtain a simple descriptive measure of the main properties of the series. The graph will not only reveal trends and seasonal variations, but will also reveal any outliers or strange observations that do not appear to be consistent with the rest of data. Methods that are designed to be insensitive to outliers are called robust. Other features to look for in a time plot include sudden or gradual changes.

The analyst should also check the possible presence of turning points, where an upward trend suddenly changes to a downward trend. When observations concern two or more variables, it may be possible to use the variation in one time series to *explain* the variation in the other series. This may lead to a deeper understanding of the mechanism that generated a given time series.

Given an observed time series, one may want to *predict* the future values of the series. For instance this is an important task for marketing and financial time series.

Time series are sometimes collected or analyzed so as to improve *control*

over some physical or economic system. Control problems are closely related to prediction in many situations.

1.2.1 Preprocessing of data by descriptive techniques

We focus our attentions on the descriptive techniques, because they can be crucial for leaning the data and then for getting a feel of their features, before trying to devise an hypothesis for a suitable model. These techniques are a kind of preprocessing of data, and can help the application of more complicate approaches. Usually, time series contains trends, seasonality or other systematic components that can seriously compromise the result.

The variations inside a time series can be imputed to different sources:

- *Seasonal variation*: many time series have variation with annual periodicity. It is very clear in temperature reading, where high temperatures are measured during the summer and low temperature during the winter. If the seasonal variations are not of direct interest, they can be removed, deseasonalizing data.

- *Other cyclic variation*: some time series exhibit variations that correspond to a fixed period due to some physical cause. They can exhibit also oscillations, that do not have a fixed period but which are predictable to some extent, i.e. business cycles in the marketing field.

- *Trend*: the definition of a trend is not very easy. Usually, it is defined as a long term change in the mean level, but is not clear what long term means, because it is correlated with the aim of the time series analysis. Sometimes long term is defined in relation with the number of points available.

- *Other irregular fluctuations*: once trends and cycles are removed, we are left with a series of residuals that may or may not be random.

All these phenomenons create some points where the series show a change. These changes can be more or less evident, but in general these points are

called **change points** or cut points. A time series is said *stationary* if there is not systematic change in the mean (no trend); and in the variance, and if the strictly periodic variations have been removed. In general, probability theory is more suited for stationary time series, so often it is required to transform a non-stationary time series in to a stationary one.

Analysis of Trend

Now we describe briefly the techniques for analyzing of series that contain a trend. As we have just explained above, a precise definition of trend is very difficult, and in particular, it is difficult to qualify exactly the value of "long term". The simplest type of trend is the "linear trend + noise", for which the observation at time t is a random variable X_t, given by:

$$X_t = \alpha + \beta t + \varepsilon_t \tag{1.1}$$

where α, β are constants and ε_t denotes a random error process with zero mean. The mean level at time t is given by $m_t = (\alpha + \beta t)$; this is sometimes called "the trend term". Equation 1.1 is a deterministic function of time and is called a *global* linea trend. In practice, this provides an unrealistic model, and the last generation models emphasize the concept of *local* linear trend. One possibility is to fit a piecewise linear model for which the trend line is locally linear but with change points where the slope and intercept of the linear trend change. Anyway, with the piecewise-linear model sudden changes in slope often seem unnatural. Extending the idea, it seems more natural to allow the parameters α and β to evolve in time. It is common to assume that α and β evolve stochastically, producing a *stochastic* trend. There are some other non-linear trend models, i.e. quadratic or exponential or logarithmic growth. In general, it is very difficult to decide what form of trend is appropriate. For time series analysis, the approach to trend modeling depends if the final goal is to measure and/or remove the trend. In our case, the aim is to describe a trend. Now we show some general approaches to estimate trends:

- *Curve fitting*
  The idea consists to fit a simple function of time such as a polynomial curve (linear, quadratic, etc). Some interesting curves are the Gompertz and logistic curve ([1] [2]), which are particular types of non-linear functions.
  The Gompertz curve can be written as:

$$\log x_t = a + br^t \tag{1.2}$$

  where a, b, r are parameters with $0 < r < 1$. It could be written also in exponential form:

$$x_t = \alpha \exp[\beta \exp(-\gamma t)] \tag{1.3}$$

  with $\gamma > 0$. The logistic curve is given by:

$$x_t = a/(1 + be^{-ct}) \tag{1.4}$$

  Both these curves are S-shaped and approach an asymptotic value as $t \to \infty$.
  In general, the curve fitting methods provide a measure of the trend, and the difference between the observations and the corresponding values of the curve provides an estimation of local fluctuations.

- *Filtering*
  The filtering approach converts one time series $\{x_t\}$ into another $\{y_t\}$ by a linear operation

$$y_t = \sum_{r=-q}^{+s} a_r x_{t+r} \tag{1.5}$$

  where $\{a_r\}$ is a set of weights.
  In the moving average method [3] the weights are chosen so that $\sum a_r = 1$. Moving averages are often symmetric with $s = q$ and $a_j = a_{-j}$. Some

examples of commonly used values for the parameters are

1. $\{a_r\} = \frac{1}{2q+1}$ for $r = -q, ..., +q$
2. $\{a_r\} = (\frac{1}{4})^{2q}$ for $r = -q, ..., +q$. If q gets large, the weights approximate to a normal curve
3. Define a set of parameters $\{a_r\}$ such that their value fit a polynomial curve, not to the whole series, but to local set of points.

When a symmetric filter is chosen, then it is possible to have an end effect problem. It means that the smoothing function can be calculated for $t = (q + 1)$ to $t = N - q$, but in some situations it is necessary to get the smoothed values up to $t = N$. The idea is to use an asymmetric filter that only involves the past and present points. A popular technique is the exponential smoothing:

$$y_t = \sum_{j=0}^{\infty} \alpha(1 - \alpha)^j x_{t-j} \tag{1.6}$$

where α is a constant such that $0 < \alpha < 1$. In this case the weights $\{a_j\} = \alpha(1 - \alpha)^j$ decrease geometrically with j.
We can define:

$$Res(x_t) = \{x_t\} - \{y_t\} = \sum_{r=-q}^{+s} b_r x_{t+r} \tag{1.7}$$

that is the residual from the smoothed value, and can be used to analyze the local fluctuations.
The main problem is the choice of the filter. There are not general rules, but usually filters are designed to produce an output with emphasis on variation at particular frequencies.

- *Filters in Series*
  In that case, a sequence of filters is built, where each element compute different features on the time series.

- *Differencing*
  A special type of filtering is simply to difference a given time series until it becomes stationary. A new series $\{y_t\}$ is obtained from the original series as $y_t = x_t - x_{t-1} = \nabla x_t$ for $t = 2, \ldots, N$. Sometimes second-order differencing is required.

Analysis of Seasonal Variation

Three seasonal models that are commonly used are:

- Additive Case: $\{X_t\} = m_t + S_t + \varepsilon_t$
- Multiplicative Case:
  1. $\{X_t\} = m_t S_t + \varepsilon_t$
  2. $\{X_t\} = m_t S_t \varepsilon_t$

where m_t is the deseasonalized mean level at time t, S_t is the seasonal effect at time t and ε_t is the random error. The seasonal indices $\{S_t\}$ are usually assumed to change slowly through time so that $S_t \simeq S_{t-s}$ where s is the number of observations per period. The indices are usually normalized so that they sum to zero in the additive case, or average to one in the multiplicative case. When the seasonal and the error terms are not exactly multiplicative, some difficulties arise.

As for trend, we can be interested to study or to remove seasonality. In the first case, the common approach is to calculate a smoothing function that depends on the period. For a one year period, we can use:

$$Sm(x_t) = \frac{(\frac{1}{2})x_{t-6} + x_{t-5} + \ldots + x_{t+5} + (\frac{1}{2})x_{t+6}}{12}$$

This smoothing function estimates the deseasonalized level of the series; the seasonal effects can be estimated by calculating $x_t - Sm(x_t)$ or $x_t / Sm(x_t)$. If the seasonal effect have to be removed, a simple linear filter can be used.

1.2.2 Stochastic Processes

Time series can be represented using a plenty of different models. We will focus our attention on the *stochastic processes* category, particularly on the *stationary* stochastic processes.

Most of the physical phenomena in the real world depend on random elements. In [4], a *stochastic process* is defined as "a statistical phenomenon that evolves in time according to probabilistic laws". In the mathematics of probability, a stochastic process is a random function. In the most common applications, the domain over which the function is defined is a time interval or a region of space. This random function is a random variable X defined on a probability space (Ω, Pr) with values in a space of functions F, i.e. $F = \Omega$. The space F in turn consists of functions $I \rightarrow D$. Thus a stochastic process can also be viewed as an indexed collection of random variables $\{X_i\}$, where the index i ranges through an index set I, defined on the probability space (Ω, Pr) and taking values on the same codomain D (often the real numbers R). In our case, we consider index i as an instant of time t, $\{X_t\}$. $\{X_t\}$ is continuous if $-\infty < t \in R < \infty$, or discrete if $t = 0, \pm 1, \pm 2, \ldots$. In time series analysis, there are some constraints on this definition, in fact the order of observations is determined by time, and usually at each instants of time corresponds only one observation. We can consider a time series as a possible realization of the stochastic process, and the analysis deals with the evaluation of the properties of the underlying probability model from this observed data.

Stochastic processes can be described in different ways, but the easiest one is to evaluate and study the first and second moments. These are *means* and the *autocovariance function.*

- *means*:

$$\mu(t) = E[X_t] \text{ for each } t \tag{1.8}$$

- *autocovariance function (acv)*: is the covariance of X_{t_1} with X_{t_2}

$$\gamma(t_1, t_2) = E\{[X_{t_1} - \mu(t_1)][X_{t_2} - \mu(t_2)]\} \tag{1.9}$$

The variance is a particular case of *acv* when $t_1 = t_2$, and is defined as:

$$\sigma^2(t) = VAR[X_t] \text{ for each } t.$$

An important class of stochastic processes are the *stationary* ones. We have just introduced this concept, and now we formalize its meaning with respect to stochastic processes.

A time series is *strictly stationary*, if the joint distribution of $X_{t_1}, \ldots X_{t_n}$ is the same as the joint distribution of $X_{t_1+\epsilon}, \ldots X_{t_n+\epsilon}$ for all $t_1, \ldots t_n, \epsilon$. In general, it means that moving the time origin by a value ϵ, it has not effect on the joint distribution, that depends only on the interval of time between two successive instants of time. These definitions have some particular influence in equations 1.8 and 1.9. Mean and variance are completely independent on t: $\mu(t) = \mu$ and $\sigma^2(t) = \sigma^2$. Moreover, the joint distribution of X_{t_1} and X_{t_2} depends only on the difference $t_1 - t_2 = \epsilon$. This value is called lag. So $\gamma(t_1, t_2) = \gamma(\epsilon) = E\{[X_t - \mu][X_{t+\epsilon} - \mu]\ \} = Covariance[X_t, X_{t+\epsilon}]$, where in general $Covariance(X, Y) = E\{[X - \mu_X][Y - \mu_Y]\}$.

In general, *acv* depends just on the instant of time when X_t is measured. We define:

$$\rho(\epsilon) = \frac{\gamma(\epsilon)}{\gamma(0)} \tag{1.10}$$

which is called $autocorrelationfunction$ (*ac*), and it measures the correlation between X_t and $X_{t+\epsilon}$. The idea is to normalize the *acv*, such that $\rho(\epsilon)$ does not depend on the unit in which time series is measured. For stochastic processes, there are three important properties of *ac* [4]:

- it is an even function of lag: $\rho(\epsilon) = \rho(-\epsilon)$
- $|\rho(\epsilon)| \leq 1$
- it does not uniquely identify the underlying model.

If the *strictly stationariety* is relaxed, we have a different class of processes called *second-order stationary*. For these processes we only require to have a constant mean and their *ac* depends only on the lag: $E[X_t] = \mu$ and

$Covariance[X_t, X_{t+\epsilon}] = \gamma(\epsilon)$. An important class of processes that belong to this class are the normal processes. Their joint distribution is a multivariate normal for all instants of time, and a normal multivariate distribution is completely characterized by its first and second moment.

1.2.3 Models of Stochastic Processes

Now we analyze some discrete stochastic processes. These classes are the most used, and some hybrid models are derived by merging them.

Random processes

A process is random if it is a sequence of random variables, $\{Z_t\}$, which are mutually independent and identically distributed. The random variables are assumed to be normally distributed with mean equal zero and variance σ_Z^2. The independence constraint involves that:

$$\gamma(\epsilon) = Covariance[Z_t, Z_{t+n}] = \begin{cases} \sigma_Z^2 & n = 0 \\ 0, & n = \pm 1, \pm 2, \dots \end{cases}$$

and it means that values at distinct time steps are uncorrelated, and the *ac* is given by

$$\rho(n) = \begin{cases} 1 & n = 0 \\ 0, & n = \pm 1, \pm 2, \dots \end{cases}$$

So the mean and the *acv* are constant and do not depend on time. This assumption involves that these processes are second-order stationary, and from the independence hypothesis they are also strictly stationary. An example of a process that belongs to this class is the white noise.

Random walks

A random walk is a process $\{X_t\}$ such that:

$$X_t = X_{t-1} + Z_t$$

where Z_t is a discrete time purely random process with mean μ and variance σ_Z^2. We find that $E[X_t] = t\mu$ and $Var[X_t] = t\sigma_Z^2$, so the process is not stationary, because its mean and variance change with t. It is interesting to note that the first difference of a random walk is stationary, in fact:

$$\nabla X_t = X_t - X_{t-1} = Z_t.$$

Moving average processes (MA)

Suppose that Z_t is a purely random process with mean zero and variance σ_Z^2, $\{X_t\}$ is a moving average processes if

$$X_t = \beta_0 Z_t + \beta_1 Z_{t-1} + \ldots + \beta_q Z_{t-q} \tag{1.11}$$

where $\{\beta_t\}$ are constants. The Zs are scaled so that $\beta_0 = 1$.
We have:

$$E[X_t] = 0$$

$$Var[X_t] = \sigma_Z^2 \sum_{i=0}^{q} \beta_i^2$$

since the Zs are independent. We also know

$$\begin{aligned} \gamma(n) &= Covariance[X_t, X_{t+n}] = \\ &= Covariance[\beta_0 Z_t + \ldots + \beta_q Z_{t-q}, \beta_0 Z_{t+n} + \ldots + \beta_q Z_{t+n-q}] = \\ &= \begin{cases} 0 & n > q \\ \sigma_Z^2 \sum_{i=0}^{q-n} \beta_i \beta_{i+n} & n = 0, 1, \ldots, q \\ \gamma(-n) & n < 0 \end{cases} \end{aligned} \tag{1.12}$$

since

$$Covariance[Z_s, Z_t] = \begin{cases} 0 & s \neq t \\ \sigma_Z^2 & s = t \end{cases}$$

Since $\gamma(n)$ does not depend on t, the mean is constant, andthe MA process is second order stationary for all values of the β_i. If the Zs are normally distributed, then so are the Xs, and we have a strictly normal process.

The *ac* of a MA process is given by

$$\rho(n) = \begin{cases} 1 & n = 0 \\ \frac{\sum_{i=0}^{q-n} \beta_i \beta_{i+n}}{\sum_{i=0}^{q} \beta_i^2}, & n = 1, 2, \ldots, q \\ 0 & n < q \\ \gamma(-n) & n < 0 \end{cases}$$

Definition 1.2.1 *A process* $\{X_t\}$ *is said to be* invertible *if the random disturbance at time t, sometime called* innovation*, can be expressed as a convergent sum of present and past values of* X_t

$$Z_t = \sum_{i=0}^{\infty} \pi_j X_{t-j}$$

being $\sum |\pi_j| < \infty$.

This definition means that the process can be rewritten in the form of an autoregressive process, possibly of infinite order. The imposition of the invertibility condition ensures that there is a unique MA process for a given *ac*.

Using the invertibility definition, the equation 1.11 may be rewritten in the following way:

$$X_t = (\beta_0 + \beta_1 B + \ldots + \beta_q B^q) Z_t = \theta(B) Z_t$$

where B is the backward shift operator, defined by

$$B^j X_t = X_{t-j} \text{ for all } j$$

and $\theta(B)$ is a polynomial of order q in B.
A process $MA(q)$ is invertible if all the roots of the equation $\theta(B) = \beta_0 + \beta_1 B + \ldots + \beta_q B^q = 0$ lie outside the unit circle that means that the roots have their modules greater than unity.

Autoregressive processes (AR)

Suppose that Z_t is a purely random process with mean zero and variance σ_Z^2. A process X_t is said to be *autoregressive* of order p if

$$X_t = \alpha_1 X_{t-1} + \ldots + \alpha_p X_{t-p} + Z_t \tag{1.13}$$

In this kind of processes, X_t is regressed on past values of X_t rather than on separate predictor variables. The AR processes have been applied when it is reasonable to assume that the present value of a time series depends linearly on the immediate past values together with a random error.

A very interesting case of an AR process is the first order process, where $p = 1$.

$$X_t = \alpha X_{t-1} + Z_t.$$

It is well know as Markov process. By a substitution into equation 1.13 we have:

$$X_t = \alpha(\alpha X_{t-2} + Z_{t-1}) + Z_t = \alpha^2(\alpha X_{t-3} + Z_{t-2}) + \alpha Z_{t-1} + Z_t.$$

This equation can be expressed as an MA process in the form:

$$X_t = Z_t + \alpha Z_{t-1} + \alpha^2 Z_{t-2} + \ldots$$

provided $-1 < \alpha < 1$, so that the sum converges. This means that there is a duality between AR and MA. The AR equation, can be written as:

$$(1 - \alpha B)X_t = Z_t$$

using the backward shift operator. In this form is clear that $E[X_t] = 0$ and $Var[X_t] = \sigma_Z^2(1 + \alpha^2 + \alpha^4 + \ldots)$ thus the variance is finite provided that $|\alpha| < 1$, in which case $Var[X_t] = \sigma_X^2 = \frac{\sigma_Z^2}{(1-\alpha^2)}$. The acv is given by $\gamma(n) = \alpha^n \frac{\sigma_Z^2}{(1-\alpha^2)} = \alpha^n \sigma_X^2$. For $n < 0$, $\gamma(n) = \gamma(-n)$. It is clear that $\gamma(n)$ does not depend on t, and the AR process of order one is second order stationary, provided that $|\alpha| < 1$, and the ac is $\rho(n) = \alpha^n$ for $n = 0, 1, 2, 3, \ldots$.

In the same way, a general order AR process can be expressed as an MA process of infinite order and this may be done using the backward shift operator.

$$(1 - \alpha_1 B - \ldots - \alpha_p B^p)X_t = Z_t$$

or equivalently

$$X_t = f(B)Z_t$$

where

$$f(B) = (1 - \alpha_1 B - \ldots - \alpha_p B^p)^{-1} = (1 + \beta_1 B + \beta_2 B^2 + \ldots).$$

So the relation between the αs and βs has been found.

Expressing X_t as an MA process, we have that the mean $E[X_t] = 0$. The variance is finite provided that $\sum \beta_i^2$ converges, and this is a necessary condition for stationarity. The acv is given by

$$\gamma(n) = \sigma_Z^2 \sum_{i=0}^{\infty} \beta_i \beta_{i+n} \text{ where } \beta_0 = 1.$$

The convergence of $\sum |\beta_i|$ is a sufficient condition for stationarity.

The procedure to compute the ac for a general order AR is more difficult

than for the *acv*. In fact, the $\{\beta_i\}$ may be algebraically hard to find. We can use the Yule-Walker equation [5] that is a set of difference equations and has a general solution

$$\rho(n) = A_1\pi_1^{|n|} + \ldots + A_p\pi_p^{|n|}$$

where $\{\pi_i\}$ are the roots of the so-called auxiliary equation

$$y^p - \alpha_i y^{p-1} - \ldots - \alpha_p = 0.$$

The constants $\{A_i\}$ are chosen to satisfy the initial condition depending on $\rho(0) = 1$, which means that $\sum A_i = 1$. The first $(p-1)$ Yule-Walker equations provide $(p-1)$ restrictions on the $\{A_i\}$ using $\rho(0) = 1$ and $\rho(n) = \rho(-n)$.

From the general form of $\rho(n)$, it is clear that $\rho(n)$ tends to zero as n increases provided that $|\pi_i| < 1$ for all i, and this is a necessary and sufficient condition for the process to be stationary.

It can be shown that an equivalent way of expressing the stationary condition is to say that the roots of the equation

$$\phi(B) = 1 - \alpha_1 B - \ldots - \alpha_p B^p = 0$$

must lie outside the unit circle.

ARMA processes

The $ARMA$ model is an hybrid combination of MA and AR processes. It contains p AR terms and q MA terms, and it is said an $ARMA$ process of order (p, q).

$$X_t = \alpha_0 X_{t-1} + \ldots + \alpha_p X_{t-p} + Z_t + \beta_1 Z_{t-1} + \ldots + \beta_q Z_{t-q}. \tag{1.14}$$

Using the backward shift operator B, equation 1.14 may be written in the form

$$\phi(B)X_t = \theta(B)Z_t$$

where $\phi(B)$ and $\theta(B)$ are polynomials of order p, q, such that

$$\phi(B) = 1 - \alpha_1 B - \ldots - \alpha_p B^p$$

and

$$\theta(B) = 1 + \beta_1 B + \ldots + \beta_q B^q.$$

The condition on the model parameters to make the process stationary and invertible are that the values of $\{\alpha_i\}$ are such that the roots of $\phi(B) = 0$ lie inside the unit circle, and the values of $\{\beta_i\}$ are such that the roots of $\theta(B) = 0$ lie outside the unit circle.

The advantage of use an $ARMA$ process lies in a principle called "*Principle of Parsimony*". It says that we want a model with as few parameters as possible, without loss of an adequate representation of data.

It helps to express an $ARMA$ model as a pure MA process in the following form

$$X_t = \psi(B) Z_t \tag{1.15}$$

where $\psi(B) = \sum \psi_i B^i$ is the MA operator. By a comparison of equation 1.15 with equation 1.14, we see that $\psi(B) = \frac{\theta(B)}{\phi(B)}$. In the same way, we can express the $ARMA$ model as a pure AR process

$$\pi(B) X_t = Z_t \tag{1.16}$$

where $\pi(B) = \frac{\phi(B)}{\theta(B)}$ and conventionally we write $\pi(B) = 1 - \sum_{i \geq 1} \pi_i B^i$. By a comparison of equations 1.15 and 1.16, we have

$$\pi(B)\psi(B) = 1.$$

The ψ or π weights may be obtained by division or by equating powers of B in an equation such as

$$\psi(B)\phi(B) = \theta(B).$$

ARIMA processes

Often the time series are not stationary; in particular if a time series is not stationary in mean, the series can be differenced. If X_t is such a series, it can be substituted by $\nabla^d X_t$. This model is called an "integrated" model, because if we want to remove the non stationarity in the original model, it has to be integrated or summed.

$$W_t = \nabla^d X_t = (1 - B)^d X_t$$

and the form of a general autoregressive integrated moving average model is

$$W_t = \alpha_1 W_{t-1} + \ldots + \alpha_p W_{t-p} + Z_t + \ldots + \beta_q Z_{t-q}.$$

This can be written also in a different way

$$\phi(B) W_t = \theta(B) Z_t$$

and this describes the dth difference of X_t, and it is said to be an $ARIMA$ process of order (p, d, q). In practice the first differencing is often found to be adequate to make a series stationary.

General linear processes

A general class of processes may be written as an MA process in the form

$$X_t = \sum_{i=0}^{\infty} \psi(i) Z_{t-i}. \tag{1.17}$$

A sufficient condition for stationarity is that $\sum_{i=0}^{\infty} |\psi(i)| \leq \infty$. The process that can be described by equation 1.17 are called *general linear process*.

1.2.4 Fitting a Model to an Observed Time Series

In the previous section, we introduced some different models, now, we describe some techniques to fit one of these models to an observed time series. We suppose that we are able to estimate the *ac* and the *acv* using equations in section 1.2.2.

Fitting an AR process

If an autoregressive process is thought to be appropriate, there are two problems that have to bo solved:

- What is the order of the process?
- How can we estimate the parameters of the process?

It is convenient to start from the second questions first.

Estimating parameters of an AR process
Suppose we have an AR process of order p, mean μ given by:

$$X_t - \mu = \alpha_1(X_{t-1} - \mu) + \ldots + \alpha_p(X_{t-p} - \mu) + Z_t.$$

Given N observations $x_1 \ldots x_N$, the parameters μ, $\alpha_1 \ldots \alpha_p$ may be estimated by least squares by minimizing

$$S = \sum_{t=p+1}^{N} [X_t - \mu - \alpha_1(X_{t-1} - \mu) - \ldots - \alpha_p(X_{t-p} - \mu)]^2$$

If the Z_t process is normal, then the least square estimates are also maximum likelihood estimates [6], the first p values in the time series being fixed.

Let us analyze a particular case where p is equal to 1. In the first-order case, we have:

$$\hat{\mu} = \frac{\bar{x}_{(2)} - \hat{\alpha_1}\bar{x}_{(1)}}{1 - \hat{\alpha_1}} \tag{1.18}$$

and

$$\hat{\alpha_1} = \frac{\sum_{t=1}^{N-1}(x_t - \hat{\mu})(x_{t+1} - \hat{\mu})}{\sum_{t=1}^{N-1}(x_t - \hat{\mu})^2} \tag{1.19}$$

where $\bar{x}_{(1)}$, $\bar{x}_{(2)}$ are the averages of the first and the last $N-1$ points. We impose that

$$\bar{x}_{(1)} \simeq \bar{x}_{(2)} \simeq \bar{x}$$

that implies that approximately

$$\hat{\mu} = \bar{x}.$$

Substituting these approximation inside equation 1.19, we obtain:

$$\hat{\alpha_1} = \frac{\sum_{t=1}^{N-1}(x_t - \bar{x})(x_{t+1} - \bar{x})}{\sum_{t=1}^{N-1}(x_t - \bar{x})^2} \tag{1.20}$$

Another approximation, which is used, consists to change the denominator of 1.20 such that

$$\hat{\alpha_1} \simeq \frac{c_1}{c_0} = r_1$$

where c_k is the usual estimator for the theoretical autocovariance coefficient $\gamma(k)$ at lag k. It is

$$c_k = \sum_{t=1}^{N-k}(x_t - \bar{x})(x_{t+k} - \bar{x})/N$$

and r_1 is called autocorrelation coefficient. This approximate estimator for $\hat{\alpha_1}$ is also intuitively appealing since r_1 is an estimator for $\rho(1)$ and $\rho(1) = \alpha_1$ for the first order AR process.

Determining the order of an AR process

It is difficult to assess the order of an AR process from the sample *ac* alone. One easy approach consists to fit the data with AR processes having progressively higher order, and to compute the residual sum of square for each

value of p that is then plotted against p. The value of p is determined by the point where the curve flattens out.

Another measure can help in determining the order, and it is the partial autocorrelation function. When fitting an AR(p) model, the last coefficient α_p will be denoted by π_p and measures the excess correlation at lag p which is not accounted for by an AR(p-1) model. It is called the *pth* partial autocorrelation coefficient and when plotted against p gives the partial *ac*. The first partial autocorrelation coefficient is simply equal to $\rho(1)$ and this is equal to α_1 for an AR(1). The sample partial *ac* is usually estimated by fitting AR processes of successively higher order and taking $\hat{\pi}_1 = \hat{\alpha}_1$ when the AR(1) process is fitted, taking $\hat{\pi}_2 = \hat{\alpha}_2$ when the AR(2) process is fitted and so on. It can be shown that the partial *ac* of an AR(p) process cuts off at lag p so that the correct order is assessed as the value of p beyond which the sample value of $\{\pi_j\}$ are not significantly different from zero.

Fitting a MA process

As for the AR process the same two questions can be posed.

Estimating parameters of a MA process

Estimation is more difficult for an MA process than an AR process, because an efficient explicit estimator cannot be found and because the errors in a MA are a non linear function of the parameters. We analyze a first order MA process

$$X_t = \mu + Z_t + \beta_1 Z_{t-1} \tag{1.21}$$

where μ, β_1 are constants and Z_t is a purely random process. The idea is to be able to write the residual sum of squares solely in terms of the data x and the parameters. One possible approach can be summarized in four steps:

- Select suitable starting value for μ and β_1, such as $\mu = \bar{x}$ and the value of β_1 given by the solution of $r_1 = \frac{\hat{\beta}_1}{(1+\hat{\beta}_1^2)}$
- Calculate the corresponding residual sum of squares using equation 1.21 recursively in the form $Z_t = -\mu + X_t - \beta_1 Z_{t-1}$. Taking $z_0 = 0$,

we calculate $z_1 = x_1 - \mu$, and then $z_2 = x_2 - \mu - \beta_1 Z_1$ and so on until $z_N = x_N - \mu - \beta_1 Z_{N-1}$. The residual sum of square $\sum_{t=1}^{N} z_t^2$ is calculated conditional on the given values of the parameters and $z_0 = 0$

- Repeat these two steps for other neighboring values of μ and β_1 so that the residual sum of square $\sum_{t=1}^{N} z_t^2$ is computed on a grid points in the (μ,β_1) plane
- Determine by inspection the value of μ and β_1 that minimize $\sum_{t=1}^{N} z_t^2$.

This procedure gives least squares estimates, which are maximum likelihood estimates with the condition $z_0 = 0$ provided that Z_t is normally distributed. Nowadays the best values of μ and β_1 are found by some iterative optimization procedure.

Determining the order of a MA process

If a MA process is thought to be appropriate for a given set of data, the order of the process is evident from the sample *ac*. The theoretical *ac* on a MA process has a very simple form in that it cuts off the lag q, so we should look for the lag beyond which the value of r_k are close to zero.

Estimating Parameters of an ARMA process

An ARMA process is considered to be appropriate for a given time series. The estimation problems for ARMA model are similar to those for a MA model. The residual sum of squares can be calculated at every point on a suitable grid of the parameter values, that allows use to determine the minimum of the cost function.

Nowadays exact maximum likelihood estimate are preferred, since the extra computation involved is not longer a limiting factor. The least squares estimates can be used as starting value for the exact maximum likelihood estimation.

Estimating Parameters of an ARIMA process

A lot of time series are not stationary so all the models that have been introduced cannot be applied directly. The main idea is to difference an observed time series until it appears to come from a stationary process. Usually, on real data, the first order differencing of noseasonal data is adequate. For seasonal data see [7].

Residual Analysis Test

When a model has been fitted on a time series, a test is necessary to understand if the model fits correctly the data. The most easy way is to compute the residual

$$Residual = \text{ observation - fitted value.}$$

If we have a good model, then we expect the residual to be random and close to zero, and model validation usually consist of plotting residuals in various ways to see whether this is the case. With time series models we have the added feature that the residuals are ordered in time and it is natural to treat them as a time series.

Two useful instruments used to analyze the residuals are the plot of the residuals and the *correlogram* of the residuals, where the correlogram is a graph in which the sample autocorrelation coefficients r_k are plotted against the lag k for $k = 0, 1, 2 \dots M$, where M is usually much less than N. The plot will help us to find outliers and any obvious autocorrelation or cyclic effects. The correlogram of the residuals will allow us to examine the autocorrelation effect more closely. Let $r_{z,k}$ denote the autocorrelation coefficient at lag k of the residual $\{\hat{z}_t\}$. If we could fit the true model with the correct model parameter values, then the true error form a purely random process and their correlogram is such that each autocorrelation coefficient is approximately normally distributed with mean 0 and variance 1/N. In practice, the true model is unknown and the correlogram of the residual from the fitted model has different properties. In general, if we fit the wrong form of model, then the distribution of residual autocorrelation will be quite different, and we

hope to get some significant values so that the wrong model is rejected.

Instead of looking at the residual autocorrelations one by one, it is possible to carry out what is called a *portmanteau lack-of-fit test*. It looks at the first K values of the residual correlogram at once. The test is

$$Q = N \sum_{k=1}^{K} r_{z,k}^2$$

where N is the number of terms in the differenced series and K is typically chosen in the range 15 to 30. If the fitted model is appropriate, then Q should be approximately distributed as a χ^2 with (K-p-q) degree of freedom. Unfortunately, the χ^2 approximation can be rather poor for $N < 100$.

Other different tests have been introduced, for more information see [4].

1.3 Segmentation and Change Points Analysis

At this point we have introduced what time series are, but we did not say anything about what we can do with them. The most important studies about time series concern the forecast of some values of the series or the discover abrupt changes in the distribution, **change points**. In this dissertation, we are interested to analyze the second topic, that can be reformulated also as the definition homogenous groups in data.

The analysis of change points consists of two distinct problems: estimate of the number and location of change points, and determination of their magnitude. Let us consider a sequence of observations where x_1, $x_2 \dots x_t$ come from some process, and x_{t+1}, $x_{t+2} \dots x_N$ come from some other process. Then we say there is a change point between x_t and x_{t+1}. Often in the interest of asymptotic results, the time indices are considered to lie on the unit interval, and some point $t^* \in (0, 1)$ is a change point, where the x_i lying in $(0, t^*)$ come from one distribution, and those in $(t^*, 1)$ come from the other.

Some authors use to apply some typical simplifying assumption. They suppose that the observations are either i.i.d. from some known family of

distributions, or they follow some relatively simple stochastic process. In recent years, different approaches have been introduced, in particular non-parametric approaches, because they can generalize this set-up. An useful approach to detect jumps in the mean of an smooth function is to take an estimate of the function from the left and the right of the definition interval. When there is a large discrepancy, the hypothesis that the function is continuous is rejected, and the presence of a change point is claimed.

An helpful heuristic that we must consider is to identify some parameters for which we want to detect the changes. It is clear that these parameters are constant or continuous far from the change points, but they have a jump or a discontinuity where the change point is located.

The change point analysis problem has been analyzed also under another point of view. In fact, researchers are often interested in defining algorithms to detect homogenous groups in the data. Given a set of K elements, each element having assigned a weight w_i, and a numerical measure, a_i, and given a positive integer G that is less than K, find a systematic and practical procedure for grouping the K elements into G mutually exclusive and exhaustive subsets such that the weights of each group are minimized using a defined measure. At the end, each group is delimited by change points. This kind of approach is called *segmentation*.

The literature about change point detection and segmentation is wide and much of the literature ignores other branch. We will present in the rest of the chapter a brief outline, focusing our attention on the statistical approaches for change point detection and then on the segmentation techniques introduced in the last years.

1.3.1 What is a Change Point?

Identifying change points with finite amounts of data can require slight shift from the asymptotic models. For instance, if a process is assumed to have a continuous mean between change points, there always exists a satisfactory function with no change points given the data. Further refinements are necessary, which usually are implied by whatever cost function is used to

estimate the existence of change point. For instance, if difference between mean in windows to the left and right of a point is used to estimate where there might be a discontinuity in the mean function, then a limitation in the first derivative of this mean function is implied.

Another question is how to deal with change points that are not well localized. For instance, many methods assume that a process will be in one state, and then it will change to another. This is not true in a lot of situations, where there is a slight drift in the state of the process. Where should we place the change point? The change point can be detected by looking at differences between disparate stretches of data, even if no close by stretches register any differences. If this is the case, where should we say the change point is?

For these reasons, we prefer to work with models that allow for slow change in their state without claiming change points. This forces us to use purely local estimates, along with a loss of representational power. However, we feel that the ability to deal consistently with a large class of problems overcomes this assumption.

1.3.2 Statistical Approaches for Change Points Detection

We present a brief overview on some statistical approaches for change point detection. Approaches can be divided by whether they are looking for changes in the mean or changes in the variance, whether they are parametric or non-parametric in their approach, and whether they assume the state of the system will be constant between changes points, or will be allowed to vary between change points. We will analyze three different approaches: detecting changes in i.i.d. Random Variables, in Mean and Variance.

Detecting Changes in i.i.d. Random Variables

A series of papers have been written on detecting change points for the manufacturing process. Pignatello et al. [8] considered a model where observations are i.i.d. $N(\mu, \sigma^2)$ when the process is in control, and σ^2 changes when the process is out of control. Windows of fixed width are defined, and σ^2 is estimated on these intervals. A change point is declared when this estimate for some interval goes outside some given range. The main goal is to determine as precisely as possible where the system went out of control and this is the location of change point.

The same authors [8] proposed a maximum likelihood estimate with a step change. It means that before the change point, observations x_i are distributed $N(\mu, \sigma_1^2)$, and after the change point x_i are distributed $N(\mu, \sigma_2^2)$. σ_1^2, σ_2^2, and the location of the change point are all estimated by maximum likelihood.

These are only some interesting papers, a number of other papers have been written, and different approaches proposed. This family of techniques requires extensive knowledge of the system to allow for proper parametrization. While this may be possible for manufacturing applications, this is not possible when the goal is to gain understanding of some new system.

Detecting Changes in Mean

In [9], the authors assumed a piecewise continuous function with i.i.d. errors. In this work, they determined where change points were located using the difference between the left and right kernel estimates of the mean function. Where this difference is large, there is evidence for a discontinuity in the mean function. When dealing with an unknown number of change points, the authors proposed to check only at a subset of the points which the function is defined on. By letting the number of points to grow slower that the number of observations, the false positive rate could be controlled.

Wavelets techniques have been used to detect change points. Wavelets are sets of filters at different levels of resolution. The lower levels of resolution give a general picture of what the data looks like, while the higher

levels contain the fine details. In [10], the authors assumed a signal with additive noise where following a wiener process. Change points could be either discontinuities or sharp cuspids in the mean function. The authors observed that the signal is described by the lower resolution wavelets coefficients (general pattern), and that large values at the higher resolutions (fine detail) correspond to either jumps or sharp cuspids in the mean function.

Hall et al. [11] face the same problem, but instead of using wavelets, they propose a two stage method where they first look at the difference between left and right kernel estimates of the mean, like in [9]. Where the difference is large, they look for the maximal difference between left and right locally linear estimates in a neighborhood of the estimated change point. This refinement gives a faster rate of convergence for the estimated location of the change point.

Gregoire and Hamrouni [12] consider at the processes of the form

$$Y_i = f(X_i) + \sigma(X_i)\epsilon_i$$

where ϵ_i are i.i.d. random variables with mean equal to zero and variance equal to one. f is smooth between change points. Change points are jumps in f, or one of its derivatives. They propose to look at the difference between the left and right locally linear estimates of f. Where the difference between them is large, a change point can be declared.

Detecting Changes in Variance

In [13], the authors propose to exploit using an iterated applications of cumulative sum of squares (CUSUM) algorithm to detect an unknown number of changes of variances. Their model assumes that observations are i.i.d. $N(\mu, \sigma^2)$ with no change points, where change points denote changes in σ^2.

The CUSUM algorithm considers a statistical test that is the maximum deviation of the cumulative sum of squares from what would be expected if

there were no change points

$$D(t) = (\frac{\sum_{i=1}^{t} x_i^2}{S} - \frac{i}{T})$$

where $S = \sum_{t=0}^{T} x_i^2$ is the sum of squares of all observations, and T is the number of observations. When D is large, there is a evidence for change point. Under the null hypothesis (no change point), D describes a Brownian Bridge.

After the detection of a change point, the series before and after the detected change point is tested by the same method for the presence of additional change points. This tends to find too many change points, so a later step of confirming these detected change points is required.

The main shortcoming of this method is that it assumes the variance to be constant between change points, which is an assumption violated by many locally stationary processes. However, CUSUM makes an excellent baseline to compare new methods to.

Guttorp and al. [14] proposed a CUSUM based approach that could deal with long range dependencies in the data. A discrete wavelet transform (DWT) is performed on the data. This has been found to remove long range dependencies in data. The resulting wavelet coefficients at each level are treated as independent. An iterated CUSUM algorithm, such as that proposed in [13] can be performed at each level of resolution, and the change point can be identified.

To deal with the lack of spatial resolution the DWT suffers from at the coarser scales, Guttorp and al. [14] suggested using the DWT to test for change points, but exploiting the maximum overlap discrete wavelet transform to refine the location. When a change point is detected using CUSUM on the DWT, the change points location can be estimated to be wherever the CUSUM statistics achieves a local maximum (or minimum) on the MODWT at the same resolution.

Another interesting method was introduced in [15]. In this work, the author investigated changes in the time varying power spectrum, which is

equivalent to the local covariance structure. He divided the series into blocks of fixed width, and estimated the power spectrum of each block. He proposes a test to look for differences between any block and all the preceding blocks, back to the previous change point. If any of these hypothesis tests fails, a change point is declared before the start of the current block.

By testing against all the previous blocks, Adak hoped to adapt to gradual changes that may be occurring, but which are not big enough to be detectable between adjacent blocks. In testing whether two power spectrums are different, a variety of Kolmogorov-Smirnov type test is proposed.

1.3.3 Statistical Approaches for Segmentation

The segmentation problem has been studied since the '60 years, and in the last years has had new impulse thanks to the interest in DNA sequence segmentation.

One of the milestones about segmentation of time series is an article by Fisher [16]. In this paper the author presents two different approaches for segmentation. In fact, he distinguishes the unrestricted problem, where no restriction or side conditions are imposed on the partition allowed, from the restricted problem, where such conditions are imposed a priori on the basis of previous knowledge, theory or for convenience. We are interested on the second approach, because a time series is an ordered sequence of data, and the groups have to meet this previous knowledge. The problem is: given a set of K elements, each element having assigned a weight w_i, and a numerical measure, a_i, and given a positive integer G that is less than K, find a systematic and practical procedure for grouping the K elements into G mutually exclusive and exhaustive subsets such that the weighted sum of each squares

$$D = \sum_{i=1}^{K} w_i(a_i - \bar{a}_i)^2$$

is minimized, where $\bar{a}_i$ denotes the weighted mean of those a's that are assigned to the subset to which element i is assigned. Besides, if $P =$

$(P_1, \ldots, P_G)$ is a possible partitioning of data, such that $P_1 < P_2 < \ldots < P_G$, P is an optimal k-segmentation if there is no k-segmentation P' such that $D_{P'} < D_P$. The solution to this problem has been found using dynamic programming. The additive property of the square distance function is used by the dynamic programming to reduce the computational time. It means that iteratively it is possible to compute the partition P moving the value of G. This algorithm has no stopping criterion. Least-squares segmentation does not indicate how to choose an optimal value for G. As G increases towards K, the lowest achievable total D must decrease towards zero because if each segment has just one point hence $D_{i..i} = 0$.

An interesting solution of Fisher's no stopping criteria has based on Minimum Message Length (MML) criterion ([17], [18], [19]). In [20], [21] and [22], the authors have implemented and tested a MML based solution to the segmentation of time series and compared it with some other techniques including Bayes Factors [23], AIC [24] and MDL [25]. In these works, they specify the change point to a precision that the data guarantees. This creates dependencies between adjacent segments and without knowledge of the Fisher's dynamic programming algorithm they have used heuristic search strategies. An evolution of these approaches has been proposed in [26]. In this paper, the authors use a simple Minimum Message Length criterion and Fisher's dynamic programming algorithm to perform numerical Bayesian integration over the change point location parameters. They give an estimate of the number of segments, which they then use to estimate the change point positions and segments by minimizing the MML criterion. This unorthodox coding scheme has the advantage that because they do not define the change point positions, they do not need to worry about the precision to which the change point positions are determined and therefore reduce the number of assumptions and approximations involved.

A generic approach has been developed in [27], where the authors use dynamic programming to search the exponential large space of the partitions of N data points. A partition of an interval I is defined as a set of M blocks, and blocks are sets of data points. This method defines also an additive

fitness function that assigns a value to any partition P in the form

$$V(P) = \sum_{m=1}^{M} g(B_m)$$

where $g(B_m)$ is the fitness of a block B_m. They exhibit an efficient dynamic programming algorithm that finds an optimal partition P^{max}: $V(P^{max}) \geq V(P)$ for all partitions P. This algorithm can be applied whenever any subpartition of an optimal partition is optimal (Principle of Optimality). This deterministic algorithm finds the partition of I that maximizes the (additive) fitness function. The excellent results have been found with the posterior probability of the model for each segment, given the data in that segment.

Since the beginning of the '90ies years, also the bioinformatics world has considered the segmentation problems. Researchers in Bioinformatics are interested in the segmentation of DNA sequence segmentation (see [28]) and some different techniques have been proposed.

In [29], the authors' goal is to find statistical methods of determining secondary protein structure, but the approach is applied to DNA sequence segmentation as well. They present an algorithm for computing the maximum likelihood estimate for the number of changed segments given a restriction on the minimum length of changed segments. To compute the maximum likelihood solution the authors might consider breaking up the sequence into all allowed configurations to find the segmentation which maximizes the log-likelihood ratio. This approach requires huge amounts of computing resources and it becomes impractical for sequences that are long or contain many changed segments. The dynamic programming helps the authors, and allows them to compute the best partitioning of data which features the minimum size of the changed segments. The determination of the l best segments is performed sequentially and it depends from the best $l-1$ segments. This approach can be explained in 3 steps.

1. Find the best segment which does not overlap any two of the best $l-1$ segments;

2. Find the best splitting and expansion for each of the best $l-1$ segments;

3. Choose from step 1 and 2 the segment that provides the greater increase in the log-likelihood.

Given the restriction on the minimum size of the changed segments, the log-likelihood will eventually decrease with additional segments, or it will be impossible to add more segments. The maximum likelihood estimate of the number of changed segments is the value of l which maximizes the log-likelihood.

In [30] and [31], Churchill proposed a Hidden Markov chain to model the segmentation of DNA sequence. The Hidden Markov model assumes that the different segments can be classified into a finite set of states. In each state, the nucleotide data is assumed to follow a probability distribution, for example a zero-order Markov chain. The states are assumed to switch from one to the other at random with low probability, since the states are unobserved and random in occurrence they form a hidden Markov chain. Note that under this model the lengths of the segments follow geometric distributions with parameters given by the transition probabilities of the unobserved chain. The unknown distribution of the state and the distributions on the states can be estimated from the data using the EM algorithm, and the Bayesian information criterion can be used to determine the number of necessary states.

A lot of Bayesian approaches have been developed for the segmentation of sequences of data. A theoretical advantage of the Bayesian approach to this estimation problem is that one can take advantage of the structure of the problem to marginalize over unknown parameters to obtain the global optimum. There are many early references for Bayesian methods for single and multiple change points. In [32], the authors present a unified approach which directly includes segments, allows for the incorporation of multiple sequences and addresses computation issues via the Gibbs sampler. To be more precise, let $S = (S_1, \ldots, S_K)$ be the collection of sequences. Suppose that the number of segments in each sequence is Q, and Θ is the parameter vector for the sequence model. Let v_{kq} indicate the position of the last observation

in the qth segment of the kth sequence, and let $V_k = (v_{k1}, \ldots, v_{kQ})$, where $v_{kQ} = n_k$ is the length of the kth sequence. Now let $M_k = (M_{k1}, \ldots, M_{kQ})$ where m belongs to $\{1, \ldots, L\}$, indicating different types of segments. Let $S_k(i, j)$ denote the ith through the jth observations from the kth sequence. Under the assumption that observations within a segment are independent from observations in other segments (the Markov assumption), given the partition, the probability of observing that $S_k(i, j)$ has $q + 1$ partitions with the last segment of type m is

$$P(S_k(i,j)|\Theta, v_{k,q+1} = j, m_{k,q+1} = m)$$

$$= \sum_{v=1}^{j-1} \sum_{l=1}^{L} P(S_k(1,v)|\Theta, v_{k,q} = v, m_{k,q} = l)$$

$$.P(S_k(v+1,j)|\Theta, m_{k,q+1} = m).P(m_{k,q+1} = m|m_{k,q} = l)$$

where $P(m_{k,q+1} = m|m_{k,q} = l)$ is the probability of observing the segment from i to j of model type m.

Chapter 2

Classical Text Analysis

2.1 Vector Space Representation

One of the most difficult task in text analysis is to find an informative way to represent a document. Same methods have been developed, but the most important is the *Vector Space Model*, introduced by Salton [33]. The main idea is to use the single words to represent the whole document regardless their position in the text. Suppose that the vocabulary is assigned, it defines a |V|-dimensional vectorial space and the documents are represented as vectors in such space. Each entry $a_{i,j}$ in the vector represents the number of times that the word i occurs in the document j. This way to represent text documents is know also as "**Bag of Words**" (**BOW**).
The success or failure of the vector space method is based on the term weighting. There has been much research on term weighting techniques but little consensus on which method is best. There are four different ways to weight each term.

- **Binary**
  The weight is 1 if the word appears in that document, 0 else:

$$\begin{cases} a_{i,j} = 1 & if\ w_j \in d_i \\ a_{i,j} = 0, & otherwise \end{cases} \tag{2.1}$$

  This representation looses completely the position and the number of

occurrences of each word inside the document.

- **Word Count (WC)**
  This value represents the number of occurrences of the term w_i in the document d_j

$$a_{i,j} = \sharp w_i \in d_j \tag{2.2}$$

- **Term Frequency (TF)**
  This quantity is the WC normalized by the total number of terms w inside the document d

$$a_{i,j} = \frac{\sharp w_i \in d_j}{\sharp w \in d_j} \tag{2.3}$$

  The sum of the TFs for each term inside a document is equal to 1. This frequency depends on the length of the document. If we compare different terms in different texts, the TF implicates that a term that appears a lot of times in a small document is most important than a term that appears a lot of times but in a long text.

- **Term Frequency - Inverse Document Frequency (TF-IDF)**
  To assess the global relevance of a term the TF is adjusted by a factor based on the distribution of the documents containing w_i in the entire collection D.

$$tfidf(d_j, w_i) = TF \times \log \frac{\sharp d}{\sharp d_{w_i \in d}} \tag{2.4}$$

  To normalize the tfidf in the [0, 1] interval, the tfidf value is computed as:

$$a_{i,j} = \frac{tfidf(d_j, w_i)}{\sqrt{\sum_{k=1}^{|V|}(tfidf(d_j, w_k))^2}} \tag{2.5}$$

  where V is the vocabulary. The idea beyond this scheme is that the more often a term occurs in a document, the more it is representative of the content inside the document. The more documents the term occurs in, the less discriminating it is. This schema was introduced in

[34], [35] and it is still the most used.

These methods have had a lot of variants, but in the original form they are yet the widely most used, also because the new proposed techniques did not show significative improvements for clustering or classification tasks.

2.1.1 Structural Problems and Solutions

This representation has some structural problems which entails an objective difficulty to work using it. The most important ones are due to the high number of words in the dictionary and the loss of the context of each word. In fact, also for a small set of documents, the vocabulary has to cover all the words inside dataset and this means that the vocabulary's dimension grows very fast, assuming a high value. Besides, each term is extracted from its context and this implies that it is difficult to disambiguate the correct meaning of each word. Particularly, this affects the representation in case of homonymous words (words written in the same way with different meaning, i.e. bank) or synonymous words (different writings but the same meaning, i.e. board, plank). The BOW representation considers independent all the terms inside the document, and it does not help to reconstruct the initial information. This is due to the fact that the structure of the sentences is lost irreparably.

All these structural problems imply that the resulting vector for a document is very sparse (most of the documents contain about 1 - 5% of the total number of terms in the vocabulary) and contains many common and low informative words.

To be able to manage this kind of representation, these problems would require a solution. There are some important techniques which try to face this issue. They can be split in two different approaches. The first one selects a subset of words, trying to discover the most informative one (**Feature Selection**). Otherwise, the second one works on the matrix words-by-documents which represents the whole dataset, and tries to reduce the dimension of the space for document representation (**Vector Space Reduction**).

Feature Selection

Often most of the words are uniformly distributed over all topics and thus their informative contribution is uniformly spread on the classes. Many auxiliary verbs, adverbs, conjunctions, pronouns etc. are absolutely useless since they do not give a description of the topic.

The idea consists in selecting a subset of the features that are really informative. This is a hard task, because the meaning of word "informative" is subjective, and depends on the context.

A shortcut is not to find the "informative" words, but to remove those words that clearly are not. When documents are analyzed, each term is compared with a list of common words, the **stop words list**, as articles, adverbs, conjunctions and so on. These terms are removed reducing considerably the dimension of the vocabulary.
In all languages, suffixes may be added to words to indicate a particular form of that word: in English is common used the suffix "$-ed$" for the pass form, or "$-s$" and "$-es$" for the plural form of name. Usually the linguists call stem or lemma the base form of the word and the process reporting a word w_i to its base form is called stemming or lemming. Removing all suffixes and prefixes can be helpful. It is not easy, and there are stemmers only for the most important languages. The most used is the *Porter stemmer* [36]. It is a rule-based algorithm in which a list of suffixes and the corresponding rules are given: when the suffix is identified it is removed using the corresponding rule to obtain a valid word.

The previous methods can help to solve some problems, but they are not a complete solution, because the actual reduction is minimal. In [37] and [38], the author analyzed the distribution of the words in a given collection and affirmed that the probability that an item occurs n times in the collection is inversely proportional to the number of its occurrences:

$$P_n \propto \frac{1}{n^a}$$

where a is close to 1. Whence:

$$Frequency \times Position \simeq Constant$$

It means that the product between the frequency of each word and its rank can be considered equal to a constant.

Luhn in [39] used this important property to rank the importance of terms. He proposed that the most relevant words belong to an intermediate set of frequencies, and the other are irrelevant and can be removed. The **Luhn reduction** removes words using the general frequency of them in the whole corpus, but it does not consider the distribution of the words in the classes or in the simple documents: a very frequent word could appear many times in a class and almost never in the other classes and it could be a valid discriminative feature for that class; using Luhn reduction it can be removed by the vocabulary loosing an important element for further processing.

A more general idea is based on a *informativeness function* that evaluates the quality all the terms and removes the less informative according to an empirical rules. Many different functions have been proposed in the literature each one based on different assumptions. They can be used separately or in cascade. The most used are: **Information Gain** [40], **Gain Ratio** [40], **Chi-Square** [41] and [42], **Odds Ratio** [43] and [44], **Correlation coefficient**[45] and **Relevancy Score** [46]. All these methods compute the relevancy of each word referring to a specific category. In order to obtain the score of w_i in a global and category-independent sense, different techniques can be used to extract a global weight from the category weight. The common globalization techniques are founded on sum, weighted sum and maximum of the category function. Among all these approaches, Information Gain and Chi-Square provide the best results, allowing the reduction of the dimension of vocabulary to 100-300 words.

Vector Space Reduction

The task of this approach consists in reducing the vector space forms by all documents. It is possible to use different techniques that come from matrix decomposition theory.

Singular Value Decomposition (SVD) decomposes a matrix into the product of three particular matrices. Given $X \in \Re^{w,d}$, given r its rank, X can be written as:

$$X = U\Sigma V \tag{2.6}$$

where $U \in \Re^{w,r}$ is the left singular vector matrix, $\Sigma \in \Re^{r,r}$ is the singular values diagonal matrix, $V \in \Re^{w,d}$ is the right singular vector matrix. **Latent Semantic Indexing (LSI)**, introduced in [47], uses this matrix decomposition approach to find a new space of dimensionality $r \leq w$ by projecting the words-by-documents matrix into the new space, where U is the projection matrix, and $Z = \Sigma V$, with $Z \in \Re^{r,d}$, is the representation of X in such space. The new representation of X is $Z = RX$ in which $R = U^{-1}$ relates the old coordinate system to the new one. Each row of R is a linear combination of the elements of the old space. To reduce a space, the first k eigenvalues of X are selected, with $1 \leq k \leq r$, that are the first k values on the diagonal of Σ. The new matrix is:

$$X = U\Sigma V \Rightarrow \widetilde{X}_k = U_k \Sigma_k V_k \tag{2.7}$$

where $U \in \Re^{w,k}$, $\Sigma \in \Re^{k,k}$ and $V \in \Re^{k,d}$. Note that U_k is the projection matrix and $Z = \Sigma_k V_k$ is the projection of the $\widetilde{X}_k$ into the new space. In [48], the author proof that $\widetilde{X}_k$, called k-truncated SVD, is the best approximation of X with rank k using the Frobenius norm. SVD is able to have a drastic reduction of the space, but it requires high computational resources.

Another interesting method used to reduce the dimensionality is **Random Projection**, where the original space is projected into a new space with low dimension, using a matrix R whose column sum to 1. Let $X \in \Re^{w,d}$

be the input matrix, we use a random projection matrix $R \in \Re^{k,w}$ to map X into a new space spanned by the rows of R. The new matrix in the lower space is:

$$\overline{X}_k = RX. \tag{2.8}$$

This approach is based on the Johnson-Lindenstrauss lemma ([49] and [50]). It claims that any n points in an Euclidean space can be projected in a $O(\frac{\log n}{\varepsilon^2})$ -dimensional space without distorting the distances between any pairs of points by more than a factor of $(1+\varepsilon)$, for any $0 < \varepsilon < 1$. This method works in the correct way till R is an orthogonal matrix, but usually it is not, so an additional orthogonalization step is required. This step is computationally expensive. The results that can be obtained with this method are comparable to the other conventional dimensionality reduction methods.

The **Concept Matrix Decomposition** [51] is based on cluster algorithms and their ability to discovery latent concepts in a set of documents. These concepts are vectors that represent some topics, that are used to map the input matrix X into a new lower dimension space. The cluster algorithm makes a partition of the original documents in k disjoint sets. Inside each set a normalized centroid c_j is defined, called Concept Vector. Let us define a Concept Matrix, whose column j is the Concept Vector of the set j:

$$C_k = [c_1 c_2 \dots c_k] \tag{2.9}$$

where $C_k \in \Re^{w,k}$. For each different partition of the whole set of documents, it is possible to define a C_k. This matrix is used to reduce the original matrix X in the following way:

$$\widetilde{X}_k = C_k Z^* \tag{2.10}$$

where $\widetilde{X}_k$ is the reduced matrix called Concept Decomposition and $Z^* \in \Re^{k,d}$ is obtained solving the minimal square problem:

$$Z^* = \arg\min_Z \|X - C_k Z\|_F^2 \tag{2.11}$$

This problem is solved using the QR factorization of the concept matrix. The results obtained with this technique are a little bit worse than the others approaches.

2.2 Document Clustering

Two of the most important tasks of Data Mining applies to documents are *Classification* and *Clustering*. They are methods used to organize documents using different rules and philosophy. *Document Classification* is supervised task, where the class which the document belongs to is unknown, and class membership is the value that we want to estimate. The system (or classifier) is trained using labeled data, that are previously catalogued by an expert. *Document Clustering* is an unsupervised task, where an unlabeled set of documents has to be split into a numbers of *clusters*. Documents that belong to the same cluster have to be similar according some similarity rule. Clustering does not need the help of an expert, because it extracts the rules from raw data without knowing the label of the class. In some algorithms the number of clusters is known, and it helps to extract the rules. Differently from Classification, Clustering has the problem of result evaluation; with huge datasets a lot of different partitions are possible and all are potentially correct. To evaluate the clustering result some indexes have been developed, trying to analyze different aspects of clustering.

Both tasks are the milestones of text analysis. In this dissertation, we are going to work with clustering, so we focus our attention onto this particular problem.

The Clustering task consists in finding a partitioning of the feature space into regions, where features of the same cluster are "$close$", and features of different clusters are "far". The most important requirement of a clustering algorithm is its scalability, that is the capacity of dealing with large amount of objects. However they also must be rather robust to the presence of noisy objects (outliers) and to the particular form of the distribution of the data in the space (not spherical regions).

The clustering algorithms can be categorized based on the type of clustering procedure, on the type of input data, on the similarity function exploited or on the possibility of an object to be assigned to more than one cluster. An interesting categorization of the clustering algorithms is based on the type of clusters produced:

- **Hierarchical Clustering**:
  it produces a hierarchical partitioning of the space D, i.e. a tree T, called dendrogram, of nodes. Each node represents a subset of the space. The root of T represents the whole space as a unique cluster and the intermediate nodes are the subsequent partitions of it until the leaves that contain the individual elements. There are two types of hierarchical clustering: *agglomerative*, where, starting from the leaves, the closer clusters are merged till only one cluster remains, and *divisive*, where, starting from the root, each cluster is divided into subclusters till each cluster contains a single point. Some algorithms are: *AGNES (AGlomerative NESting)* [52], *Voorhhes algorithm* [53], *Scatter/Gather* [54].

- **Partitioning Clustering**
  it decomposes the whole set of data in a set of clusters. It tries to evaluate a number of partitions which optimize a particular objective function. This function can privilege a global or local structure of data, and it is optimized by an iterative procedure. Some algorithms are: *K-means*[55] [56], *Spherical K-means* [51] *Partitioning Around Medioids* [52].

- **Density-Based Clustering**
  The main idea is to merge close object in the space evaluating their density. Density Based Spatial Clustering of Applications with Noise (DBSCAN) is the most important density-base algorithm [57].

- **Fuzzy Clustering**
  The algorithms that belong in this category are called not-exclusive, because each point in the space can belong to more than one cluster,

and for each cluster the point has a degree of membership. In that way, the idea is to be able to model the uncertainty that is present inside real data. Usually, it is possible to have the soft version of the most important cluster algorithms, *c-means* is an example.

Clustering usually has high computational costs, hence many algorithms find the sub-optimal solution using reasonable approximation.
Now we discuss the most popular cluster algorithm, *k-means*, and the agglomerative hierarchical approach that has been used in this work.

2.2.1 k-means

K-means is one of the most used clustering algorithms, because it is easy and very intuitive. It is an iterative and heuristic algorithm that does not guarantee the convergence of the objective function to a global optimal.

Expectation-Maximization (EM)

K-means is a particular case of a more general algorithm: *Expectation-Maximization* (EM) [58]. This algorithm is developed to solve the unsupervised maximum likelihood estimation problems. Suppose we have a set of data $\chi = \{x_1, x_2, \dots x_N\}$, that were generated from a probability distribution and are vectors i.i.d. (identically and identically distributed). Given a density probability function $p(x|\theta)$, where θ is a vector of parameters that represents a distribution (i.e if p is a gaussian distribution, θ is made of mean and variance), we want to approximate the spatial distributions of χ by k distribution of probability characterized by $\Theta = \{\theta_1, \theta_2 \dots \theta_k\}$. The likelihood of the elements given the model is:

$$p(\chi|\Theta) = \prod_{i=1}^{N} p(x_i|\Theta) = \mathfrak{L}(\Theta|\chi) \tag{2.12}$$

where $\mathfrak{L}(\Theta|\chi)$ represents the probability that data χ has been generated by Θ. It is called *likelihood*. $\mathfrak{L}$ can be seen as a function of Θ, so a vector Θ^*

that maximizes it can be found:

$$\Theta^* = \arg\max_{\Theta} \mathfrak{L}(\Theta|\chi).$$

Usually, the log function of $\mathfrak{L}(\Theta|\chi)$ is maximized, and it is possible without any kind of problems, because the logarithm is a monotonic growing function:

$$\Theta^* = \arg\max_{\Theta} \{\log \mathfrak{L}(\Theta|\chi)\} \tag{2.13}$$

The choice of $p(x|\theta)$ is fundamental for the possibility of solving the problem.

The *EM* algorithm is a technique that is able to simplify the likelihood function assuming the presence of some hidden parameters. It assumes that a set of functions C_j with $j = 1, .., k$ depending on the parameters set Θ exists and it is hidden. We can write the equation 2.12 as:

$$p(x_i|\Theta) = \sum_{j=1}^{k} p(C_j)p(x_i|C_j, \theta). \tag{2.14}$$

Substituting 2.14 in the logarithm version of the likelihood and deriving with respect to a generic Θ_s and equaling to 0, we obtain:

$$\nabla_\Theta \log \mathfrak{L}(\Theta|\chi) = \sum_{i=1}^{n} \frac{1}{p(x_i|\Theta)} \nabla_\Theta \left\{ \sum_{j=1}^{k} p(C_j)p(x_i|C_j, \Theta) \right\} = 0 \tag{2.15}$$

and assuming θ_h and θ_l to be independent:

$$\begin{aligned} \nabla_\Theta \log \mathfrak{L}(\Theta|\chi) &= \sum_{i=1}^{n} \frac{1}{p(x_i|\Theta)} \nabla_\Theta \{p(C_s)p(x_i|C_s, \Theta)\} = \\ &= \sum_{i=1}^{n} \frac{p(C_s)}{p(x_i|\Theta)} \nabla_\Theta p(x_i|C_s, \Theta) = 0 \end{aligned} \tag{2.16}$$

After some passages and using Bayes rule, finally we obtain:

$$\Theta^* : \sum_{i=1}^{n} p(C_s|x_i, \Theta^*) \nabla_{\Theta^*} \log \{p(x_i|C_s, \Theta^*)\} = 0 \tag{2.17}$$

$$p(C_s|x_i, \Theta^*) = \frac{p(x_i|C_s, \Theta^*)p(C_s)}{p(x_i|\Theta)} \tag{2.18}$$

The equations (2.17) and (2.18) suggest a way to estimate the optimal parameter set Θ^*: the maximum can be found by an iterative gradient ascent algorithm.

- **E step**
  the expected value of likelihood $\log \{p(\chi|C_s, \Theta)\}$ is evaluated assuming the current parameters Θ to be optimal:

$$\log p(\chi|C, \Theta^{(t+1)}) = E[\log p(\chi|C, \Theta)|C, \Theta^{(t)}]$$

- **M step**
  the parameters Θ are updated using the $p(\chi|C, \Theta^{(t+1)})$ estimated in the **step E** thus maximizing the likelihood:

$$\Theta^{(t+1)} = \arg\max_{\Theta} \left\{\log p(\chi|C, \Theta^{(t+1)})\right\}$$

If we suppose that the points in the space are displaced similarly to *clouds* condensed around some mean points, we can model each function $p(x_i|C_j, \Theta)$ using *gaussian distributions* where the parameters θ_j are the mean values μ_j (centroids). From the equation (2.17) and (2.18) and using $p(x_i|C_j, \Theta) = N_{\mu_s,\sigma}(d_i)$ we have:

$$\widehat{\mu}_s = \frac{\sum_{i=1}^{n} (p(C_s|d_i, \widehat{\mu})x_i)}{\sum_{i=1}^{n} p(C_s|x_i, \widehat{\mu})} \tag{2.19}$$

$$\tag{2.20}$$

and the iterative procedure becomes:

- **E step**
  the values of $N_{\mu_j,\sigma_j}(x_i)$ are evaluated for each function j and for each element x_i;

- **M step**
  the parameters μ_j are updated using the incremental version of the equation (2.19):

$$\widehat{\mu}_s^{(t+1)} = \frac{\sum_{i=1}^{n} \left(p(C_s|x_i, \widehat{\mu}^{(t)})x_i\right)}{\sum_{i=1}^{n} p(C_s|x_i, \widehat{\mu}^{(t)})}.$$

k-means algorithm

The EM algorithm has been used in clustering. The estimation algorithm places the k probability functions $P(d_i|C_j, \theta)$ for $j = 1, ..., k$ in the space and it attempts to reflect the distribution of the points in the space more accurately as possible. The placement of such functions is done by computing the values of the parameters set Θ that maximizes the likelihood. By representing the clusters with the parameters Θ and the membership function of an element to a cluster with the hidden function C_j, we have translated the EM estimation algorithm into the *k-Means* algorithm.

k-Means [55] sets the $P(C_s|d_i, \mu)$ probability function to a binary assignment function to have an exclusive clustering algorithm:

$$P(C_s|x_i, \widehat{\mu}_s) = \begin{cases} 1 & \text{if } \widehat{\mu}_s : \min_j \|x_i - \widehat{\mu}_j\|. \\ 0 & else \end{cases} \tag{2.21}$$

Each point x_i is assigned exclusively to the cluster C_j having the closest centroids $c_j = \mu_j$. The iterative steps are:

$$\begin{cases} \widehat{\mu}(0) = \mu_0 \\ \widehat{\mu}_s(t+1) = \frac{\sum_{x_i \in C_s(t)} x_i}{|C_s(t)|} \end{cases} \tag{2.22}$$

The k-means algorithm can thus be summarized by the following procedure:

1. Choose of k initial values for the centroids $\mu(0)$;
2. Partition the set χ using the current centroids;
3. Update the centroids using the equation (2.22);
4. Repeat the steps 2-3 until the centroids change more than a given threshold.

This algorithm moves the gaussian functions (moving their means μ) toward the center of the *clouds* of points in the space. The k-means assigns each point to the cluster with the closest centroid, minimizing the distance between each point in the cluster and the centroid:

$$e(C_j) = \sum_{x_i \in C_j} D(c_j - x_i) \tag{2.23}$$

The original version of k-means used the *Euclidean distance* as $D()$ function. A version using the *cosine distance* as function has been used by Steinbach et Al. in [56] and it has been named *BiSec-k-means*

The k-means algorithm is very simple and highly efficient but has some important drawbacks:

- it may find local minima and appropriate techniques to find the global minimum, as *genetic algorithms* or *deterministic annealing*, are necessary;
- the number of clusters k must be defined in advance;
- it builds clusters with regular shapes (ellipsoids or spheroids) and it badly recognizes the data with different distributions shapes. To solve this problem, usually the number of clusters k is increased, but it is not the optimal solution;

- it needs to define a mean of points to compute the centroid and in some case it may not have sense, because centroid is not a real point inside the dataset.

To solve some of these problems, different methods to find the optimal initial centroids, measures of similarity and different strategies to evaluate the centroids have been proposed in the literature.

Outliers

Outliers are isolated points that if they are assigned, they could deform irreparably the partition. Unfortunately, K-means is indifferent to these points, which are managed as normal points. In fact, it assigns them to their closest centroid moving it towards the isolated point: this alteration can be negative since at the following step, many new points will be incorrectly inserted into the cluster. A solution could be to increase the number of clusters, but usually this solution just reduces the effect of outliers and does not solve the problem.

K-means Initialization

K-means is an heuristic algorithm and the solution depends on the initial conditions. For this reason, K-means does not guarantee the global solution, but only a local one. The choice of the initial centroids (seeds) is a well known problem in data mining.
An easy solution is the choice of a random sequence of seeds, k vectors without any kind of rule. It is easy, but it is not the optimal solution, because it is possible that two or more vectors are too close. In this case, K-means is not able to create a good partition of the data. Another simple solution consists to run K-means several times with different seeds, and choose the initial centroids that optimize the objective function.

An interesting method chooses seeds which are more distant on the average. This solution guarantees that the cluster ares enough distant, and so a better partition of data can be found.

In [59], the authors use information inside data to initialize centroids close to the real seeds. The heuristic is based on the sub sampling of data. On each sub-sample is applied K-means, and on the solutions that have been obtained, the refine algorithm selects the best centroids. Suppose that CM_i, with $i = 1, \ldots, J$, where J is the number of sub-samples, is the solution of the clustering algorithm on the sub-sample i, and $CM = \bigcup_i CM_i$. They apply the clustering algorithm on CM using CM_i as seeds. This iterative approach produces at each step FM_i solutions. The initial cetroids on the whole dataset are chosen selecting the FM_i that gives the best value of the objective function. In general, this method gives centroids that reduce the dependence of the clustering algorithm from the seeds.

2.2.2 Hierarchical Clustering: Agglomerative

Hierarchical clustering is a way to investigate grouping in your data, simultaneously over a variety of scales, by creating a cluster tree. The tree is not a single set of clusters, but rather a multi-level hierarchy, where clusters at one level are joined as clusters at the next higher level. This allows to decide what level or scale of clustering is most appropriate in an application.
To perform hierarchical cluster analysis on a data set, we follow this procedure:

1. *Find the similarity or dissimilarity between every pair of objects in the data set.* In this step, the distances between objects are calculated using different measurements.

2. *Group the objects into a binary, hierarchical cluster tree.* In this step, the pairs of objects, that are in close proximity, are linked together. This linkage step uses the distance information generated in step 1 to determine the proximity of objects to each other. As objects are paired into binary clusters, the newly formed clusters are grouped into larger clusters until a hierarchical tree is formed.

3. *Determine where to divide the hierarchical tree into clusters.* In this step, the objects in the hierarchical tree are divided into clusters. The

clusters are created by detecting natural groupings in the hierarchical tree or by cutting off the hierarchical tree at an arbitrary point.

Finding the Similarities Between Objects

There are many ways to calculate the distance information. Given an m-by-n data matrix X, which is treated as m (1-by-n) row vectors $x_1, x_2, ..., x_m$, the various distances between the vectors x_r and x_s are defined as follows:

- Euclidean distance
  $d_{rs}^2 = (x_r - x_s)(x_r - x_s)'$
- Standardized Euclidean distance
  $d_{rs}^2 = (x_r - x_s)D^{-1}(x_r - x_s)'$
  where D is the diagonal matrix with diagonal elements, which denotes the variance of the variable X_j over the m objects.
- Mahalanobis distance
  $d_{rs}^2 = (x_r - x_s)V^{-1}(x_r - x_s)'$
  where V is the sample covariance matrix.
- City Block metric
  $d_{rs} = \sum_{j=1}^{n} |x_{rj} - x_{sj}|$
- Minkowski metric
  $d_{rs} = \{\sum_{j=1}^{n} |x_{rj} - x_{sj}|^p\}^{\frac{1}{p}}$
  For the special case of p = 1, the Minkowski metric gives the City Block metric, and for the special case of p = 2, the Minkowski metric gives the Euclidean distance.
- Cosine distance
  $d_{rs} = (1 - \frac{x_r x_s^{'}}{(x_r' x_r)^{1/2}(x_s' x_s)^{1/2}})$
- Hamming distance
  $d_{rs} = (\sharp(x_{rj} \neq x_{sj})/n)$

The values in the data set can be optionally normalized before calculating the distance information. In a real world data set, variables can be measured against different scales.

Defining the Links Between Objects

Once the proximity between objects in the data set has been computed, it is possible to determine which objects in the data set should be grouped together into clusters. The links between objects are built taking the distance information generated by step 1 and linking pairs of objects that are close together into binary clusters (clusters made up of two objects). It then links these newly formed clusters to other objects to create bigger clusters until all the objects in the original data set are linked together in a hierarchical tree called *dendrogram*.
There are different algorithms used to generate the hierarchical cluster tree information. These linkage algorithms are based on different ways of measuring proximity between two groups of objects. If n_r is the number of objects in cluster r and n_s is the number of objects in cluster s, and x_{ri} is the ith object in cluster r, the definitions of these various measurements are as follows:

1. Single linkage, also called *nearest neighbor*, uses the smallest distance between objects in the two groups.
   $d(r,s) = \min(dist(x_{ri}, x_{sj}))$, $i \in (1 \dots n_r)$, $j \in (1 \dots n_s)$

2. Complete linkage, also called *furthest neighbor*, uses the largest distance between objects in the two groups.
   $d(r,s) = \max(dist(x_{ri}, x_{sj}))$, $i \in (1 \dots n_r)$, $j \in (1 \dots n_s)$

3. *Average linkage* uses the average distance between all pairs of objects in cluster r and cluster s.
   $d(r,s) = \frac{1}{n_r n_s} \sum_{i=1}^{n_r} \sum_{j=1}^{n_s} dist(x_{ri} x_{sj})$

4. *Centroid linkage* uses the distance between the centroids of the two groups.
   $d(r,s) = dist(\bar{x_r}, \bar{x_s})$

where
$\bar{x_r} = \frac{1}{n_r} \sum_{i=1}^{n_r} x_{ri}$
and $\bar{x_s}$ is defined similarly. The centroid method can produce a cluster tree that is not monotonic. This occurs when the distance from the union of two clusters to a third cluster is less than the distance from either r or s to that third cluster. In this case, sections of the dendrogram change direction. This is an indication that we should use another method.

5. *Ward linkage* uses the incremental sum of squares; that is, the increase in the total within-group sum of squares as a result of joining groups r and s. It is given by
$d(r,s) = n_r n_s \frac{d_{rs}^2}{(n_r+n_s)}$
where d_{rs}^2 is the distance between cluster r and cluster s defined in the Centroid linkage. The within-group sum of squares of a cluster is defined as the sum of the squares of the distance between all objects in the cluster and the centroid of the cluster.

Creating Clusters

After you create the hierarchical tree of binary clusters, you can divide the hierarchy into larger clusters. Clusters can be created in two ways.
Finding the Natural Divisions in the Data Set: in the hierarchical cluster tree, the data set may naturally align itself into clusters. This can be particularly evident in a dendrogram diagram where groups of objects are densely packed in certain areas and not in others. The inconsistency coefficient of the links in the cluster tree can identify these points where the similarities between objects change. This value compares the height of a link in a cluster hierarchy with the average height of neighboring links. If the object is consistent with those around it, it will have a low inconsistency coefficient. If the object is inconsistent with those around it, it will have a higher inconsistency coefficient. That function compares each link in the cluster hierarchy with adjacent links two levels below it in the cluster hierarchy. This is called the depth of the comparison. Using the inconsistent function, you can specify

other depths. The objects at the bottom of the cluster tree, called leaf nodes, that have no further objects below them, have an inconsistency coefficient of zero.
Specifying Arbitrary Clusters: instead of letting the cluster function create clusters determined by the natural divisions in the data set, the number of clusters can be specified.

2.2.3 Measures of Clustering Effectiveness

Once the space is clustered, some measures to evaluate the clustering quality are necessary. An important categorization of the evaluation methods of the document clustering quality is based on the knowledge they need:

- *External measures*
  They measure the quality of clustering algorithms comparing the generated clusters with the original classes. These methods presume to know the label of the documents in the test set.

- *Internal measures*
  They compare different sets of clusters without knowing the original label of the examples (no external knowledge).

External measures

External measures suppose that the labels of the examples are available, so we can compute the *confusion matrix* $\mathbf{M}$ (or *contingency table*) of the clustering process. Each element $m_{h,j}$ is the number of elements originally labeled L_h and assigned to cluster C_j by the algorithm. We define n_h to be the number of elements originally in the class L_h and m_j to be the number of elements assigned to the cluster C_j:

$$m_j = \sum_{h=1}^{|L|} m_{h,j} \quad \text{and} \quad n_h = \sum_{j=1}^{|C|} m_{h,j}$$

Finally m is the total number of elements in the collection:

$$m = \sum_{j=1}^{|C|} m_j = \sum_{h=1}^{|L|} n_h$$

Classification Error. When the number of clusters coincides with the number of the original classes, it is possible to associate each cluster to the class with the highest number of elements inside it. By this relation, the elements on the diagonal of the contingency table represent the correctly classified documents and we can compute the accuracy (or the classification error) of the cluster C_j as:

$$a_j = \frac{m_{j,j}}{m_j} \tag{2.24}$$

$$e_j = 1 - a_j = 1 - \frac{m_{j,j}}{m_j} \tag{2.25}$$

The overall accuracy (classification error) is the average value of the accuracies of the clusters: $a = \frac{\sum_j a_j}{|C|}$. The limit of this approach is that it ignores how the incorrectly classified documents are distributed among the other clusters: being uniformly and randomly distributed is worse than all being assigned to a single cluster.

Entropy. We can use the entropy to measure the composition of the generic cluster C_j. Entropy measures how a distribution of probability is uniform: more the value of the entropy is close to 0, less the distribution has an uniform trend. If we compute the conditional probability of the cluster C_j of having an element of the class A_h, that is $p(d_i \in C_j | d_i \in L_h) = p_{h,j}$ and we consider it for $h = 1, ..., |L|$, we have the distribution of the classes L_h in the cluster C_j. We can suppose that a good clustering algorithm produces clusters with a high number of documents coming from a single class L_h and

not many from the others. From this assumption, it follows that the more the distribution is sharp, the more the cluster purity is high. We can use entropy to measure how the curve sharp:

$$E_j = \sum_{h=1}^{|L|} p_{h,j} \log(p_{h,j}). \tag{2.26}$$

We can estimate the value of $p(d_i \in C_j | d_i \in L_h)$ by measuring the number of documents $d_i \in L_h$ that have been assigned to the cluster C_j, that is $m_{h,j}$, with respect to the total number of elements in the cluster C_j, m_j:

$$E_j = \sum_{h=1}^{|L|} \frac{m_{h,j}}{m_j} \log\left(\frac{m_{h,j}}{m_j}\right). \tag{2.27}$$

The values of E_j have not an uniform upper bound and they must be normalized with respect to the worst case that is the uniform distribution of the classes C_h in the cluster C_j:

$$\begin{aligned} E_0 &= \sum_{h=1}^{|L|} \frac{1}{|L|} \log\left(\frac{1}{|L|}\right) \\ &= -\log\left(|L|\right). \end{aligned} \tag{2.28}$$

The relative entropy of each cluster C_j becomes:

$$E_j^* = \frac{\sum_{h=1}^{|L|} \frac{m_{h,j}}{m_j} \log\left(\frac{m_{h,j}}{m_j}\right)}{-\log(|L|)}. \tag{2.29}$$

The total entropy of clustering is evaluated by weighting the relative entropy of each cluster by the its cardinality and normalizing the result with respect to the total number of elements in the set:

$$E^* = \frac{\sum_{j=1}^{|C|} m_j E_j^*}{m} \tag{2.30}$$

Measured based on the matrix $\mathfrak{A}$. A further 2×2 table can be computed $\mathfrak{A} = \{a_{r,c} : r, c \in \{0, 1\}\}$ by considering the pairs of documents: the value 0 stands for the number of pairs of documents occurring in the same group, while the value 1 for the number of pairs occurring in different groups. Moreover the rows are related to the original classes while the columns to the generated clusters obtaining the following possible cases:

- $a_{0,0}$ is the number of pairs of documents belonging to the same class and assigned to the same cluster by the clustering algorithm:

$$\begin{aligned} a_{0,0} &= \sum_j \sum_h \binom{m_{h,j}}{2} = \\ &= \sum_{j,h} \frac{m_{h,j}[m_{h,j} - 1]}{2} \end{aligned} \tag{2.31}$$

- $a_{0,1}$ is the number of pairs belonging to the same class but assigned to different clusters:

$$\begin{aligned} a_{0,1} &= \sum_h \sum_j \sum_{t<j} m_{h,j} m_{h,t} = \\ &= \sum_h \binom{n_h}{2} - a_{0,0} \end{aligned} \tag{2.32}$$

- $a_{1,0}$ is the number of pairs of documents occurring in the same cluster but not in the same class:

$$\begin{aligned} a_{1,0} &= \sum_j \sum_h \sum_{f<h} m_{h,j} m_{f,j} = \\ &= \sum_j \binom{m_j}{2} - a_{0,0} \end{aligned} \tag{2.33}$$

- $a_{1,1}$ is the number of pairs of documents occurring in neither the same class, nor in the same cluster:

$$\begin{aligned} a_{1,0} &= \sum_{j}\sum_{h}\sum_{t<j}\sum_{f<h} m_{h,j}m_{f,t} = \\ &= \binom{m}{2} - (a_{0,0} + a_{0,1} + a_{1,0}) \end{aligned} \tag{2.34}$$

Moreover we can evaluate:

- The column and row sums of A:

$$a_{0,\bullet} = \sum_{c=0}^{1} a_{0,c} = \sum_{h} \binom{n_h}{2} \tag{2.35}$$

$$a_{\bullet,0} = \sum_{r=0}^{1} a_{r,0} = \sum_{j} \binom{m_j}{2} \tag{2.36}$$

- The total number of pairs in the data set:

$$\sum_{r}\sum_{c} a_{r,c}a_{r,c} = M = \binom{m}{2} \tag{2.37}$$

By these values, many measures have been proposed in literature:

1. *Rand*
   In [60], Rand supposes that the clustering algorithm quality is high if it puts in the same cluster the largest number of pairs appearing in the same class and if it assigns to different clusters the larger number of pairs appearing in different classes:

$$\mathcal{R} = \frac{a_{0,0} + a_{1,1}}{M} \tag{2.38}$$

In [61], the author proposed an adjusted form of the Rand measure:

$$\mathcal{K} = n(n-1)[1-\mathcal{R}] = 2(a_{0,10} + a_{1,0}) \tag{2.39}$$

2. *Jaccard*
   Jaccard coefficient was used in [62] and it measures the percentage of pairs of documents correctly assigned to the same cluster with respect to the number of the pairs of documents that appear in the same class and have been assigned to the same cluster ($a_{0,0}$), the number of pairs of documents appearing in the same class but assigned to different clusters ($a_{0,1}$) and the number of pairs of documents assigned to the same cluster but belonging to different classes ($a_{1,0}$):

$$\mathcal{J} = \frac{a_{0,0}}{a_{0,0} + a_{0,1} + a_{1,0}} \tag{2.40}$$

3. *Fowlkes and Mallows*
   In [63], the author introduces two symmetric criteria:

$$\begin{aligned} W' &= \frac{a_{0,0}}{a_{\bullet,0}} \\ W'' &= \frac{a_{0,0}}{a_{0,\bullet}} \end{aligned} \tag{2.41}$$

   In [64], Fowlkes and Mallows use the geometric mean of the two measures:

$$\begin{aligned} \mathcal{F} &= \frac{a_{0,0}}{\sqrt{\widetilde{a}}} \\ \widetilde{a} &= a_{0,\bullet} a_{\bullet,0} \end{aligned} \tag{2.42}$$

4. Γ *statistic*
   In [65] the authors used the Γ statistic function to measure the clus-

tering quality:

$$\begin{aligned} \Gamma &= \frac{Ma_{0,0} - \widetilde{a}}{\sqrt{\widetilde{a}(M - a_{0,\bullet})(M - a_{\bullet,0})}} \\ \widetilde{a} &= a_{0,\bullet}a_{\bullet,0} \end{aligned} \tag{2.43}$$

These functions can be used even when the classes are unknown. They can compare in fact the results of two clustering algorithm by building confusion matrices $\mathbf{M}$ and $\mathfrak{A}$ using the results of the clustering algorithms.

These external criteria are used to verify the null hypothesis H_0 of random distribution of the data: the clustering algorithm is performed r times on r subsets X_i of the entire data set; if the number of times that the value of the external measure passes a threshold, then the H_0 hypothesis is not verified and the data have really a not-random distribution. Given a labeled data set X composed by $P = \{P_1, P_2, ..., P_k\}$, and a partitioning of the data $C = \{C_1, C_2, ..., C_k\}$ produced by a clustering algorithm, the Monte Carlo procedure can be briefly summarized in the following steps:

1. for $i = 1, .., r$
   (a) generate a data-set X_i as subset of X labeled according to P;
   (b) run the clustering algorithm and produce the C_i partitioning;
   (c) compute $F(C_i)$ by using an index defined before
2. plot the $F(C_i)$ values and compare them with $F(C)$
3. accept or reject the H_0 Hypothesis.

Internal measures

When the labels set is not available, the external measures are unusable and the internal ones must be used. As we briefly outlined before, these measures evaluate the clustering quality without knowledge on the data set by evaluating some properties that a good cluster set should have.

Intra-Cluster distance. Ideally a good clustering algorithm should produce clusters containing highly homogeneous documents. The intra-cluster distance measures the similarity between all internal documents. Different functions can be used:

- *Average internal similarity function.* The average similarity between all the documents internal to the cluster:

$$Intra_{AV}(C_j) = \frac{\sum_{d_i \in C_j} \sum_{d_h \in C_j} D(d_i, d_h)}{|C_j|(|C_j| - 1)} \tag{2.44}$$

  It measures the average distance between documents inside the cluster. The more the points in a cluster are similar, the more the cluster is good.

- *Maximum internal distance function.* The maximum value of similarity between two elements in the same cluster:

$$Intra_{MAX}(C_j) = \max_{d_i, d_h \in C_j} D(d_i, d_h) \tag{2.45}$$

  It measures the distance between the two least similar documents in the cluster. The more they are similar, the more the quality of the cluster C_j is high.

Inter-Cluster distance. A good clustering algorithm should produce groups which are highly dissimilar. The Inter-Cluster distance is a group of external functions that evaluate the quality of the clustering by measuring the similarity among the clusters. These functions are used in agglomerative hierarchical clustering algorithms to select the clusters that must be merged.

- *Minimum external distance function (Single-Link).* Single-Link function [66] measures the distance between the two closest points belonging

to the two clusters:

$$Inter_{MIN}(C_i, C_j) = \min_{d_i \in C_i, d_j \in C_j} D(d_i, d_j) \tag{2.46}$$

- *Average external distance function (Average-Link).* Average-Link function [53] evaluates the average distance between points in two clusters:

$$Inter_{AV}(C_i, C_j) = \frac{\sum_{d_i \in C_i} \sum_{d_j \in C_j} D(d_i, d_j)}{|C_i||C_j|} \tag{2.47}$$

- *Maximum external distance function (Complete-Link).* This function uses the distance between the most distant points in two different clusters:

$$Inter_{MAX}(C_i, C_j) = \max_{d_i \in C_i, d_j \in C_j} D(d_i, d_j) \tag{2.48}$$

- *Distance between centroids.* The distance between two clusters can be approximated by evaluating the distance between the corresponding centroids [67].

$$D(C_i, C_j) \simeq D(\widehat{c_i}, \widehat{c_j}) \tag{2.49}$$

where $\widehat{c_j} = \frac{\sum_{d_h \in C_j} d_h}{|C_j|}$ stands for the centroid of the cluster C_j.

Silhouette index. The silhouette value measures how similar each document d_i is to the documents in the same cluster compared to elements in the other clusters. It is defined as:

$$S_i = \frac{(b_i - a_i)}{\max(a_i, b_i)} \tag{2.50}$$

where a_i is the average similarity between d_i and all the other points in the cluster C_i at which d_i belongs:

$$a_i = \frac{\sum_{d_h \in C_i} D(d_i, d_h)}{|C_i|} \tag{2.51}$$

b_i is the minimum average similarity between d_i and all elements in the other clusters $C \neq C_i$:

$$b_i = min_{C \neq C_i} \frac{\sum_{d_h \in C} D(d_i, d_h)}{|C|} \tag{2.52}$$

The silhouette values are evaluated for each element of the cluster and they range in $[-1, +1]$:

- $S_i \simeq 1$ the document d_i has been very well clustered.
- $S_i \simeq 0$ the element d_i can lie between two clusters.
- $S_i < 0$ the document d_i has been wrongly clustered.

This measure is usually used to evaluate the best number of clusters in an algorithm. The total average silhouette value for a clustering partitioning is evaluated by averaging the S_i for each element in the partitioning:

$$S = \frac{\sum_i S_i}{n} \tag{2.53}$$

The optimal number of clusters can be estimated by that which gives the largest values for S [52].

2.2.4 Number of Clusters Detection

Some algorithms require the number of clusters that they have to find inside data as input, k-means is one of these. The quality of the results depends on this value, because a high value can create clusters with few documents

or empty, otherwise a small number entails a bad distribution of data for each cluster, i.e. few big clusters, and some small. The definition of the right number of clusters is a "difficult" problem. If data are bi-dimensional, they can be plotted, and it helps to define the number of clusters. Clearly, it is not a solution, because normally data have thousands of dimensions. In section 2.1.1 we have seen some feature selection or vector space reduction approaches, but they are not able to map a dataset into just two dimensions without to loss of a lot of information. Furthermore clustering is an unsupervised method so it makes no sense to check data using a supervisor.
An empirical approach consists in applying the clustering algorithm several times, each time changing the number of clusters. Results are evaluated using particular indexes, see section 2.2.3. To evaluate the results, the value of the objective function is not used, because it decreases till the number of clusters equals the total number of documents. In fact, increasing the number of clusters, each partition has fewer documents and it is more compact improving the value of the objective function. The number of clusters that has the best index, is chosen. This value depends on the particular heuristic used in the algorithm, and it is not necessarily the optimal value, but it increases the knowledge of the problem and it is good as initial value for the next analysis.

2.3 Pseudo-Supervised Clustering in Textual Domains

Dimensionality reduction techniques applied to the document domain may have interesting properties derived from the statistical distribution of terms in the document corpus. In particular, the SVD decomposition was shown to improve the retrieval performance due to the potential discovery of latent semantic relationships among different words, whilst the concept matrix decomposition exploits the prototypes of certain homogenous sets of documents. Thus, the projection basis can be chosen in order to extract a proper rep-

resentation which should bias a clustering algorithm towards yielding more "meaningful" partitions with respect to human criteria. However, notice that the choice of the clusters is not an obvious task even for a human expert, since many partitions can be feasible for a given document set. Moreover, we can choose the set of documents used to extract the projection basis by the feedback of an expert. This procedure can be embedded in an iterative algorithm in which an expert evaluates the cluster quality and enriches the set used to compute the projection basis by a proper partition of the documents it contains.

Using this approach, we designed two pseudo-supervised clustering algorithms [68], in which the human evaluation is (eventually) used to refine the projection basis for the dimensionality reduction.

2.3.1 Pre-clustering

We defined both a pseudo-SVD and a pseudo-CMD using the proposed framework. The first step of both the algorithms is a pre-clustering of the set used to compute the projection basis. This step requires to choose the documents to be used to compute the basis and may consider an expert's feedback to properly partition this set. Then, the whole document corpus is projected to the reduced space and a unsupervised clustering is performed on the projected vectors.

Given a document collection $\mathcal{D}$, we identify a subset $\mathcal{T}$ of $\mathcal{D}$ and a starting value for k (i.e. the number of clusters). These values can be set using the a-priori knowledge (human pre-clustered set) or performing a clustering algorithm for different values of k and choosing the better one using the measures of quality for the clusters (e.g. Silhouette index). It is important that the $\mathcal{T}$ subset and k^* reflect the statistical properties of the entire dataset. Therefore, the subset must be wide enough to represent the variability in the entire corpus.

2.3.2 Pseudo-SVD and Pseudo-CMD

Given the partitioning π^* of the $\mathcal{T}$ in k^* clusters and the whole document set $\mathcal{D}$, we can approximate its word-by-document matrix X by projecting it on the basis vectors collected as columns of the matrix S as

$$\widetilde{X}_{V(k)} = SZ \ .$$

The matrix Z contains the projected representations of the documents. The matrix S depends on the chosen projection technique (SVD or CMD).

Then the document set $\mathcal{D}$ is partitioned applying the k-means algorithm to the projected representations.

Pseudo SVD

Given the partitions π_j, $j = 1, \ldots, k^\star$ of the set $\mathcal{T}$, we would like to choose a new reference system to represent the documents, which maximizes the dissimilarity between the different clusters and maintains the larger amount of information. Thus, for each cluster π_j we compute the set of principal directions to represent that cluster. If v_j is the number of principal components selected for cluster j, we obtain $V = \sum_{j=1}^{k^*} v_j$ components for the new basis system. The principal directions for each cluster are obtained by a SVD of the corresponding word-by-document matrix. Thus the matrix S is composed by juxtaposing the U_j matrices, $j = 1, ..., k^*$, obtained by performing the SVD to the word-by-document matrix of cluster j,

$$S = \left[\begin{array}{cccc} U_1 & U_2 & ... & U_{k^*} \end{array} \right] \ .$$

Then we define the *Pseudo SVD* of X as the matrix $\widetilde{X}_k = SZ$ with Z^* computed as the solution of the minimum square problem

$$Z^\star = \arg \min_Z \ \|X - C_k Z\|_F^2 \ .$$

Pseudo CMD

This method aims at deriving a basis which describes the peculiar features of each partition π_j by searching the elements common to documents of each cluster. Let $X_j \in \mathbf{R}^{\mathbf{w}\times\mathbf{d_i}}$ be the word-by-document matrix for cluster j. For each cluster we can construct the corresponding concept vector as shown in equation (2.9). then, we can define the set of *word clusters* which will be used to represent the features distinctive of each cluster. A word belongs to the word cluster W_j if the term weight in the concept vector π_j is greater than the weight of the same word in all the other concept vectors. For each partition X_i we keep only the components related to the words in the word cluster W_i, and we set to zero all the others, obtaining a new $w \times d_i$ matrix X'_i.

We can further sub-partition each cluster X'_i, to obtain more than one direction for each original partition. If v_i is the chosen number of directions for the cluster i, then we obtain a set $C_i = \{c_{i1}, \ldots, c_{iv_i}\}$ of concept vectors. Each sub-partition corresponds to a word vector W_{ij} for $j = 1, ..., v_i$ obtained as described previously. Thus, each partition π_i is represented by a set of directions represented by the concept vectors c_{ij}, where only the components corresponding to the not null elements in the associated word vectors are not set to zero. These vectors are collected in a matrix D_i. Finally, these matrices are juxtaposed to obtain the $w \times V$ matrix S,

$$S = \left[\begin{array}{cccc} D_1 & D_2 & ... & D_{k*} \end{array} \right] .$$

Thus, the *Pseudo CMD* can be written as

$$X^* = SZ$$

where Z is a $V \times d$ matrix. To compute Z we can take advantage of a property of S which derives from its construction. Since a word (vector component) is assigned to a unique word cluster, in the matrix S there are several disjoint groups of words. Each word is clearly associated to a single vector and only

in the corresponding column it has a not null value. Since each vector of S has an unitary norm, we have

$$S^T S = SS^T = I \Rightarrow S = ortonormal\ matrix\ ,$$

and, therefore:

$$S^T X^* = S^T SZ \Rightarrow S^T X^* = Z$$

Since each word is exclusively associated to a direction, we obtain a reduced model where the values are all not negative. This method has a very low computational cost compared to the other methods (SVD, CMD, Pseudo-SVD). However, it causes a greater loss of information.

2.3.3 Results

We tested the Pseudo-SVD and Pseudo-CMD algorithms described in section 2.3 on a data-set composed by long documents. Each document in the data-set can cover different topics.

Data Preparation

We have evaluated our algorithms using a data-set composed by papers from conferences on computer science. These documents are usually coded in PDF format and in the first processing step we extracted the text from PDF files. In order to avoid noise due to limitations of the PDF parsers and to errors in the original documents, we filtered the terms extracted from the data-set using the Aspell-0.50.4.1 library [69]. Each string is analyzed by Aspell that returns the most similar word in its dictionary or null. Then, to reduce the dictionary dimension, we defined a stop-word list to remove common words and we applied the Luhn Reduction with lower threshold $ts_{low} = 0.2$, and upper threshold $ts_{up} = 15$. To have a view on the topic distribution in the data set, we manually pre-clustered it into 10 topics, removing unknown or ambiguous documents (see table 2.1).

N.	Name	N. Files
1	Fuzzy Control	112
2	Biological Evolutionary Computation	240
3	Agent Systems	118
4	Global Brain Models	171
5	Wavelets Applications	68
6	Chaotic Systems	70
7	Neural Networks	134
8	Clustering and Classification	86
9	Image Analysis and Vision	114
10	Independent and Principal Component Analysis and SVM	104

Table 2.1: Distribution of the topics in the data-set.

Method	Entropy	Accuracy	Human evaluated
PCMD 4	0.7105	0.3743	0.4199
PCMD 7	0.7084	0.3387	0.4110
PCMD 10	0.7093	0.2892	0.3358
PSVD 4	0.6449	0.4719	0.6609
PSVD 7	0.6229	**0.4838**	**0.7097**
PSVD 10	**0.6115**	0.4691	0.6874
k-means	0.6862	0.3895	0.5731

Table 2.2: Quality of the different clustering methods when partitioning the data-set into 10 clusters.

Experimental Results

We applied to the data-set the following three clustering algorithms: k-means, Pseudo-SVD (PSVD) and Pseudo-CMD (PCMD). Each algorithm was applied setting the number of clusters to 10 (the number of chosen topics). For PSVD and PCMD we varied the number of principal components $v \in \{4, 7, 10\}$. Totally, we obtained seven different partitions of the data set and, for each of them, we estimated the quality of the clustering, evaluating the *Accuracy*, the *Conditional Entropy*, and the accuracy based on a human evaluation. In table 2.2 we can see the average values of the classification

accuracy, the entropy and the human evaluated accuracy for each clustering algorithm.

According to accuracy, the best method is the PSVD with $v = 7$, while according to the entropy, the most performing one is PSVD with $v = 10$. The case PSVD 7 is the best according to accuracy and it is the second one with respect to the entropy value. It could be the candidate as the best method. Even for the case PSVD 10, we can do the same consideration: it is the best case according to entropy and it has good classification accuracy. However we can note that the accuracy/entropy combined results for PSVD are better than the others ones, while the ones for PCMD are the worse. In figure 2.1 we report the composition of the clusters for the case PSVD 7 when considering the ten classes listed in table 2.1. Analyzing the results,

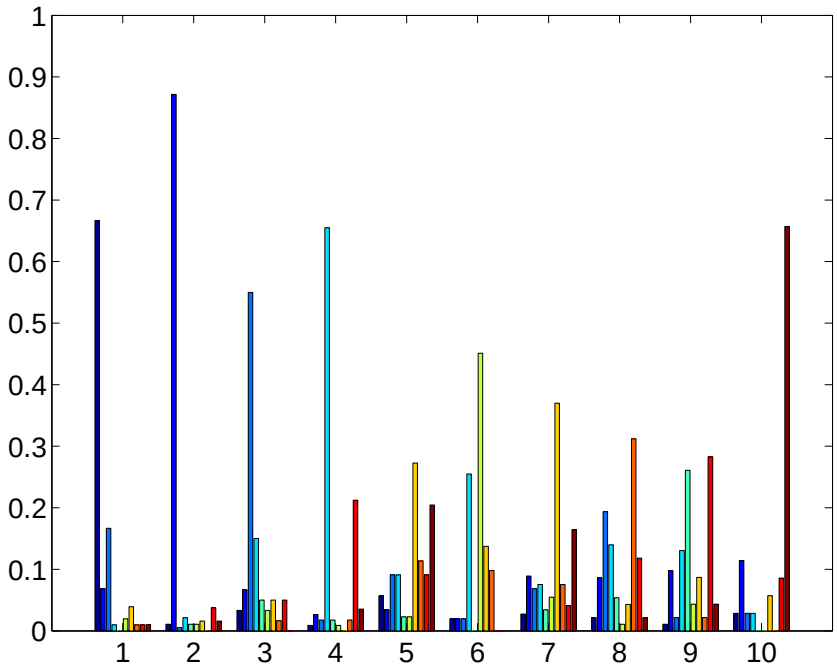

Figure 2.1: Topic distribution in the 10 clusters for the Pseudo-SVD algorithm with v=7.

we note that the values of accuracy and entropy are very low (under the 50% of correct results and over 0.6 of entropy). This happens because the data set has many transversal topics that false the results. To point out this

fact we have performed a manual evaluation of the clusters for each clustering algorithm and we have evaluated the accuracy using the expert's evaluations. The results are much better and show higher accuracy percentages (about 65-70%). An important fact that appears from table 2.2 is that the order of quality of the clustering algorithms is unchanged: the PSVD algorithms seem to be the better than k-means and PCMD and, in particular, the PSVD 7 is still the best one. By analyzing the results of human evaluation (figure

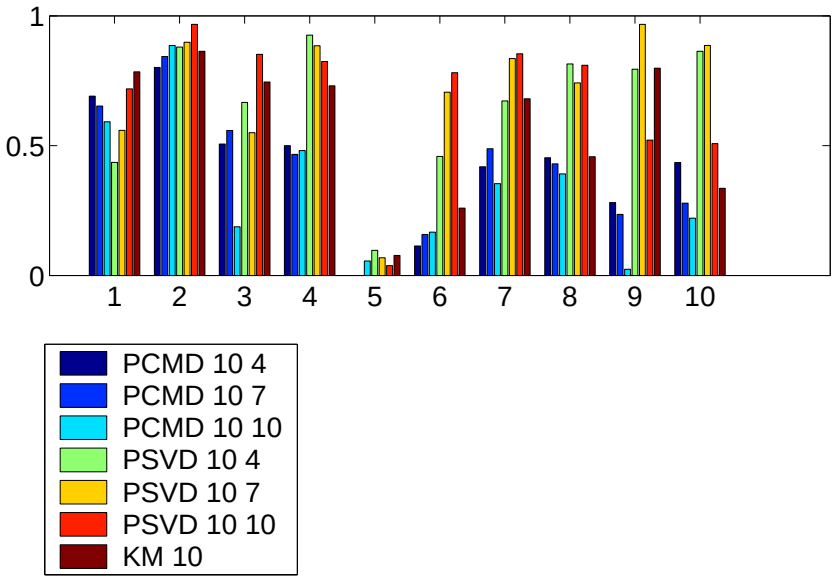

Figure 2.2: Human expert's evaluation of cluster accuracy.

2.3.3), we can see that there are some classes with very low accuracy (for example class 5). This class corresponds to the topic 'Wavelets' and it has such a low accuracy because many documents dealing with wavelets, deal mainly with 'Image Analysis and Vision' and they were not inserted in that cluster. This happens for many other documents in many other clusters and it derives from the choice of a *flat clustering* scheme. Moreover, the manual pre-clustering process created clusters according to one possible scheme, i.e. according to the method used in each paper (ANN, Fuzzy Control, PCA etc...). This partition, however, doesn't take into account the fact that the ANN, Genetic Algorithms or the other ones can be applied to various fields (aerospace, robotic, bio-informatics and others). Thus, different partitions

are consistent with this data-set.

Chapter 3

Language Evolution

3.1 Why does a language evolve?

Biological evolution is the process by which all forms of life change slowly over time because of slight variations in the genetic sequences that one generation passes down to the next. It has been known for some time, now, that the majority of molecular mutations are selectively neutral, that is do not affect the fitness of the phenotype and hence are free to accumulate. The corresponding statistical model of sequence evolution (The Neutral Theory of Evolution, by Motoo Kimura) is a centerpiece of modern genomics. In that model, evolution corresponds to a trajectory in the space of all possible DNA sequences, with most steps being neutral with respect to selection, and mostly equivalent to a random walk. That neutral mutations can reach fixation for purely statistical reasons has been known for a long time.

Similar considerations can be made for the evolution of languages: neutral mutations accumulate, and some can become fixed in the population, over time. This creates a random walk, that can partly be reconstructed by simply keeping track of some statistical markers in the sequence, as done in DNA sequence evolution. The evolution theory considers four evolutive factors: mutation, selection, genetic drift and migration. All these factors can be applied to the languages, but in that case we have to consider also another one: transmission. For genes there is only one type of transmission, but for languages there are a lot of different way to transfer language knowledge

between people. The first has been done between family and children. In fact, everyone learns words, easy grammar rules from their parents, but it is only a small part during life. In fact, when people grow, other social factors enter. For example, best friends, school teachers, radio, television, etc. At the end, the individual language is a mosaic created by a lot of transmissive.

A language can be subjected to three type of evolutions: phonetic, semantic and grammatical. The phonetic changes correspond to the change of sounds of the words, and they are the first easy thing that can be noticed inside a language. The semantic changes involve the change of meaning of some words. An example is the Italian word "donna", which means female, but at the beginning, it means wife, from the latin "mulier". The grammar is a more stable part of a language, but this does not mean that it can not change. Usually, a grammar change requires more time than semantic and phonetic mutations. In the past, each change of a language has had a geographical change or migrations of the populations which use them. The geographical barriers like seas, mountains, rivers, etc, constrain migrations reducing the diffusion of genes and languages and influencing the evolution.

An interesting phenomena inside language evolution is called: lexical diffusion. A novelty can be diffused between persons, but also between words. Although each language keeps a lot of grammar and pronunciation irregularities, there is a trend to homogenize and to extend the rules. For example, the English verbs have became more regular than some centuries ago. In general, some changes in one word bias the closer phonetic words. A classic example are the Grimm's laws. They assert:

- Regular shifting of consonants in groups (discovered in germanic languages).
- The classical voiceless stops (k,t,p) became voiceless aspirates (h,th,f) in English and mediae (h,d,f) in German, e.g., the initial sounds of Latin pater, English father, German Vater, and in the middle of Latin frater, English brother, German Bruder.
- The classical unaspirated voiced stops (g,d,b) became voiceless stops

(k,t,p) in English and voiceless aspirates (kh,ts,f) in German, e.g., the initial sounds of Latin decem, English ten, German zehn.

- The classical aspirated voiced stops (gh,dh,bh) became unaspirated voiced stops (g,d,b) in English and voiceless stops (k,t,p) in German, e.g., the initial sounds of Sanskrit dhar, English draw, German tragen. [latin was traho].

We investigate the use of statistical properties of languages to analyze linguistic evolution. We call them statistical language signatures (SLS) and we investigate how they evolve over time, how well they reflect ancestral relations between languages, and if they can be used to obtain language trees that are independent of any subjective choice. This approach by-passes any semantic assessment of word similarity or any arbitrary choice of words to be compared. It is repeatable automatically and hence objectively by simply performing statistical comparisons between text documents. The approach is entirely data-driven. We make use of 2 datasets to independently validate our choice of features (SLS) and to analyze aspects of language evolution. A first dataset (containing 50 news stories written in 5 languages) is used to test the hypothesis that out representation is sufficiently stable and sensitive to characterize a language, at least within the domain of the indo-european (IE) family. The second corpus contains translations of the same document ("The universal declaration of human rights") in 42 modern languages.

The fundamental observation is that the SLS of a text does not depend on its semantic content, but rather on the language in which it is written. In other words, all documents in a language have similar statistical signatures. Another key observation is that all languages we examine have their characteristic SLS, and that they can be reliably identified by it. We test both these observations on the datasets, with high statistical confidence.

The consequence of these two - apparently conflicting - observations is that the SLS evolves slowly, drifting over time, and diverging as the languages diverge from a common ancestor. In this, it behaves similarly to the genomic signatures introduced by Karlin and on which our analysis is based [70]. To test this hypothesis, we used the second corpus, and standard

phylogenetic reconstruction algorithms, to reconstruct a tree of the IE family. The resulting tree, entirely based on statistical properties, is generally in agreement with the commonly accepted view of the IE family, although some exceptions are discussed in section 3.7.

In our current analysis we are limited by the use of texts available in the latin alphabet, and hence we focus mostly on european languages. However we believe that the methods can be exported to more general situations, perhaps using standard transliteration methods or - later - even phonetic representations.

3.2 Indo-European Languages

The most widely studied language family in the world is the Indo-European. There are a number of reasons for this:

- Many of the most important languages of the world are Indo-European. These languages are official or co-official in many countries and are important in academic, technical and world organizations.
  Examples: English, Spanish, French, German, Russian. Indeed, more than half the world's population speaks one or more of these languages either as a mother tongue or as a business language.

- Languages that are essential in multinational contexts or with large numbers of speakers.
  Examples: Portuguese, Hindi, German, Bengali.

- Some of the great classical languages of religion, culture and philosophy were Indo-European.
  Examples: Latin, Greek, Persian, Sanskrit, Pali.

- Languages that are scattered around the world as their speakers are part of diasporas.
  Examples: Greek, Yiddish, Polish, Armenian, Romany, Kurdish, Italian, Punjabi, Gujerati.

The Indo-European languages tend to be inflected (i.e. verbs and nouns have different endings depending on their part in a sentence). Some languages (e.g. English) have lost many of the inflections during their evolution. The Indo-European languages stretch from the Americas through Europe to North India. The Indo-European Family is thought to have originated in the forests north of the Black Sea (in what is now Ukraine) during the Neoloithic period (about 7000BC). These people began to migrate between 3500BC and 2500BC, spreading west to Europe, south to the Mediterranian, north to Scandinavia, and east to India. The Indo-European Family is divided into twelve branches, ten of which contain existing languages. We will describe each of these branches separately.

The Celtic Branch

This is now the smallest branch. The languages originated in Central Europe and once dominated Western Europe (around 400BC). The people migrated across to the British Isles over 2000 years ago. Later, when the Germanic speaking Anglo Saxons arrived, the Celtic speakers were pushed into Wales (Welsh), Ireland (Irish Gaelic) and Scotland (Scottish Gaelic). One group of Celts moved back to France. Their language became Breton spoken in the Brittany region of France. Breton is closer to Welsh than to French. Other Celtic languages have became extinct. These include Cornish (Cornwall in England), Gaulish (France), Cumbrian (Wales), Manx (Isle of Man), Pictish (Scotland) and Galatian (spoken in Anatolia by the Galatians mentioned in the Christian New Testament). Welsh has the word order Verb-Subject-Object in a sentence. Irish has the third oldest literature in Europe (after Greek and Latin).

The Germanic Branch

These languages originate from Old Norse and Saxon. Due to the influence of early Christian missionaries, the vast majority of the Celtic and Germanic languages use the Latin Alphabet. They include English, the second most

spoken language in the world, the most widespread, the language of technology, and the language with the largest vocabulary. A useful language to have as your mother tongue. Dutch and German are the closest major languages related to English. An even closer relative is Frisian. Flemish and Afrikaans are varieties of Dutch while Yiddish is a variety of German. Yiddish is written using the Hebrew script. Three of the four (mainland) Scandinavian languages belong to this branch: (Danish, Norwegian, and Swedish). Swedish has tones, unusual in European languages. The fourth Scandinavian language, Finnish, belongs to a different family. Icelandic is the least changed of the Germanic Languages - being close to Old Norse. Another old language is Faroese. Gothic (Central Europe), Frankish (France), Lombardo (Danube region), Visigoth (Iberian Peninsula) and Vandal (North Africa) are extinct languages from this branch. German has a system of four cases and three genders for its nouns. Case is the property where a noun takes a different ending depending on its role in a sentence. An example in English would be the forms: lady, lady's, ladies and ladies'. The genders are masculine, feminine and neuter. English has lost gender and case. Only a few words form their plurals like German (ox, oxen and child, children). Most now add an s, having been influenced by Norman French.

The Latin Branch

Also called the Italic or Romance Languages. These languages are all derived from Latin. Latin is one of the most important classical languages. Its alphabet (derived from the Greek alphabet) is used by many languages of the world. Latin was long used by the scientific establishment and the Catholic Church as their means of communication. Italian and Portuguese are the closest modern major languages to Latin. Spanish has been influenced by Arabic and Basque. French has moved farthest from Latin in pronunciation, only its spelling gives a clue to its origins. French has many Germanic and Celtic influences. Romanian has picked up Slavic influences because it is a Latin Language surrounded by a sea of Slavic speakers. Portuguese and Spanish have been separate for over 1000 years. The most widely spoken of

these languages is Spanish. Apart from Spain, it is spoken in most of Latin America (apart from Portuguese speaking Brazil, and a few small countries like Belize and Guyana). Romansh is a minority language in Switzerland. Ladino was the language spoken by Spain's Jewish population when they were expelled in 1492. Most of them now live in Turkey and Israel. Provincial and Catalan are closely related languages spoken in the south of France and the north-east of Spain, respectively. Note that Basque (spoken in parts of Spain and France) is not an Indo-European language - in fact it is totally unrelated to any other language of the world. Galician is a Portuguese dialect with Celtic influences spoken in the north west of Spain. Finally, Moldavian is a dialect of Romanian spoken in the Moldova. Under the Soviets the Moldavians had to use the Cyrillic alphabet. Now they have reverted back to the Latin alphabet. Apart from Latin, other extinct languages include Dalmatian, Oscan, Faliscan, Sabine and Umbrian. Latin had three genders and at least six cases for its nouns and a Subject-Object-Verb sentence structure. Most modern Romance languages have only two genders, no cases and a Subject-Verb-Object structure.

The Slavic Branch

These languages are confined to Eastern Europe. In general, the Catholic peoples use the Latin alphabet while the Orthodox use the Cyrillic alphabet which is derived from the Greek. Indeed some of the languages are very similar differing only in the script used (Croatian and Serbian are virtually the same language). One of the oldest of these languages is Bulgarian. The most important is Russian. Others include Polish, Kashubian (spoken in parts of Poland), Sorbian (spoken in parts of eastern Germany), Czech, Slovak, Slovene, Macedonian, Bosnian, Ukrainian and Byelorussian. The Slavic languages are famed for their consonant clusters and large number of cases for nouns (up to seven). Macedonian has three definite articles indicating distance; all are suffixes: VOL (ox), VOLOT (the ox), VOLOV (the ox here), VOLON (the ox there).

The Hellenic Branch

The only extant language in this branch is Modern Greek. Greek is one of the oldest Indo-European languages. Mycenaean dates from 1300BC. The Ancient Greek of Homer was written from around 700BC. The major forms were Doric (Sparta), Ionic (Cos), Aeolic (Lesbos), and Attic (Athens). The latter is Classical Greek. The New Testament of the Christian Bible was written in a form of 1st Century AD Greek called Koine. This developed into the Greek of the Byzantine Empire. Modern Greek has developed from this. Greek has three genders and four cases for nouns but no form of the verb infinitive. The language has its own script, derived from Phoenician with the addition of symbols for vowels. It is one of the oldest alphabets in the world and has led to the Latin and Cyrillic alphabets. The Greek Alphabet is still used in science and mathematics. Until the 1970s Greek was a Diglossic language. This means that there were two forms: Katharevoussa used in official documents and news broadcasts and Demotic used in common speech.

The Illyric Branch

Another single language branch. Only Albanian (strongly influenced by the Slavic languages) belongs to this branch. It has been written in the Latin script since 1908; this replaced the Arabic script. Albanian has many avoidance words. Instead of saying wolf, the phrase "may God close its mouth" is used. There are two dialects so different that they could be considered separate languages. Geg is spoken in the north of Albania and Kosovo. Tosk is spoken in southern Albania and north west Greece.

The Anatolian Branch

This branch includes the language of the Hittite civilisation which once ruled central Anatolia, fought the Ancient Egyptians and was mentioned in the Christain Bible's Old Testament. Other languages were Lydian (spoken by a people who ruled the south coast of Anatolia), Lycian (spoken by a Hellenic

culture along the western coastal regions), Luwian (spoken in ancient Troy) and Palaic. All languages in this branch are extinct.

The Thracian Branch

This branch is represented by a single modern language, Armenian. It has its own script. Armenian is spoken in Armenia and Nagorno-Karabakh (an enclave in Azerbaijan). The language is rich in consonants and has borrowed much of its vocabulary from Farsi (Iranian). Nouns have 7 cases and the past tense of verbs take an E prefix like Greek. Two extinct languages from this branch are Thracian (spoken by Spartacus) and Phrygian (spoken in ancient Troy).

The Iranian Branch

These languages descended from Ancient Persian, the literary language of the Persian Empire and one of the great classical languages. The main language of this branch is Farsi (also called Iranian and Persian), the main language of Iran and much of Afghanistan. Kurdish is a close relation. Kurdish is spoken in Turkey, Syria, Iran and Iraq by the Kurds. It is the second largest of the Iranian languages after Farsi. In Turkey it was banned until recently. Pashto (or Pushtu) is spoken in Afghanistan and parts of north west Pakistan. Baluchi is spoken in the desert regions between Iran, Afganistan and Pakistan. These languages are written in the Nastaliq script, a derivative of Arabic writing. It is interesting that you cannot tell which family a language belongs to by the way it is written. Ossetian is found in the Caucasus mountains, north of Georgia. Tadzhik is a close relative of Farsi, written in Cyrillic and spoken in Tadzhikistan (of the former USSR) as well as northern Afghanistan. Avestan is the extinct language of the Zoroastrian religion. Scythian is an extinct language of a warrior people who once lived north of the Black Sea.

The Indic Branch

This branch has the most languages. Most are found in North India. They are derived from Sanskrit (the classical language of Hinduism dating from 1000BC). This gave rise to Pali (the language of Buddhism), Ardhamagadhi (the language of Jainism) and the ancestors of the modern North Indian languages. Of the modern North Indian languages, Hindi and Urdu are very similar but differ in the script. The Hindi speakers are Hindus and use the Sanskrit writing system called Devanagari (writing of the Gods). Urdu is spoken by the Muslims so uses the Arabic Nastaliq script. These two languages are found in north and central India and Pakistan. Nepali is closely related to Hindi.

In India most of the states have their own language. These languages either use Devanagari script or a derivation (if the people are Hindus) or the Arabic Nastaliq script (if the people are Muslims). Bengali (West Bengal as well as Bangladesh), Bhili (Central India), Oriya (in Orissa), Marathi (in Maharashtra), Assamese (in Assam), Punjabi and Lahnda (from the Punjab), Maithili and Maghadi (from Bihar), Kashmiri (Kashmir), Sindhi (the Pakistan province of Sindh - written in Nastaliq), Gujerati (Gujerat in western India), Konkani (in Goa, an ex Portuguese colony, uses the Latin script), Sinhalese (Sri Lanka - uses its own script derived from Pali), Maldivian (Maldives - with its own script).

The most surprising language in this branch is Romany, the Gypsy's language. Gypsies migrated to Europe from India. Sanskrit had three genders as has Marathi; most modern Indic languages have two genders; Bengali has none. The fascinating point about India is that the south Indian languages (like Tamil) are not Indo-European. In other words, Hindi is related to English, Greek and French but is totally unrelated to Tamil. North Indians visiting Madras (in the south) are as baffled by Tamil as a foreigner would be.

The Tokharian Branch

Turfanian and Kuchean are recently identified extinct languages once spoken in north west China. Very little is known about this branch as only a few

manuscripts dating from 600 AD are in existence. The closest relatives of these languages are the Celtic, Hittite and Latin branches.

3.2.1 History

In the 2,000 years before the IndoEuropeans, who remained in the homeland, began to write history, the success of the agricultural revolution brought a population explosion to the Indo-European community. The pressure of population, we may surmise, compelled the migration of successive waves of Indo-Europeans to fertile areas that were not yet cultivated. The linguistic translocation of the Indo-European homeland from northern Europe to Asia Minor requires drastic revisions in theories about the migratory paths along which the IndoEuropean languages must have spread across Eurasia. Thus, the hypothetical Aryans who were said to have borne the so-called Aryan, or Indo-lranian, language from Europe to India, and who were conscripted into service as the Nordic supermen of Nazi mythology, turn out to be the real Indoiranians who made the more plausible migration from Asia Minor around the northern slopes of the Himalaya Mountains and down through modern Afghanistan to settle in India.

Europe is seen, therefore, as the destination, rather than the source, of Indo-European migration. Speakers of the Hittite, Luwian and other Anatolian languages made relatively small migrations within the homeland, and their languages died there with them. The more extensive migrations of speakers of the Greek-Armenian-Indo-Iranian dialects began with the breakup of the main Indo-European language community in the third millennium B.C. Two groups of Indo-Iranian speakers made their way East during the second millennium B.C. One of them, speakers of the Kafiri languages, survives to this day in Nuristan, on the southern slopes of the Hindu Kush in northeast Afghanistan. In Five Continents, a posthumous book recounting his many botanical expeditions between 1916 and 1933, Vavilov speculated that the Kafirs might perpetuate some "original relics" of Indo-lranian. The second group of Indo-Iranians, who followed a more southerly path into the Indus Valley, spoke a dialect from which the historical languages of India

Celtic Branch
Welsh : Irish Gaelic : Scottish Gaelic : Breton Cornish Gaulish : Cumbrian : Manx : Galatian
Germanic Branch
English : Dutch : Flemish : Frisian Afrikaans German : Yiddish : Danish : Swedish Norwegian Faroes : Icelandic Anglo Saxon : Old Norse : Frankish Gothic Lombardo : Visigoth : Vandal
Romance (Latin) Branch
Italian : Sardinian : French : Provencal Catalonian Spanish : Ladino : Galician Portuguese : Romansh Romanian : Moldavian Latin : Oscan Umbrian : Faliscan : Sabine : Dalmatian
Slavic Branch
Russian : Belorussian : Ukrainian : Polish Sorbian Czech : Slovak : Slovene Croatian : Serbian Kashubian : Bulgarian : Macedonian Bosnian:Old Church Slavic
Baltic Branch
Lithuanian : Latvian Prussian
Hellenic Branch
Modern Greek Mycenaean : Koine Byzantine Greek Classical Greek (Attic : Doric, Ionic, Aeolic)
Illyric Branch
Albanian
Anatolian Branch
Hittite : Lydian : Lycian: Luwian : Palaic Thracian
Branch Armenian
Thracian : Phrygian Iranian Branch Farsi : Kurdish : Pashto Baluchi : Ossetian : Tadzhik Persian Avestan : Scythian
Indic Branch
Hindi : Urdu : Nepali : Bengali : Assamese : Oriya Kashmiri : Punjabi Sindhi : Marathi : Gujerati Bhili Lahnda : Maithili : Magahi Konkani Sinhalese : Maldivian : Romany Sanskrit : Pali
Tokharian Branch
Turfanian : Kuchean

Table 3.1: IndoEuropean Languages.

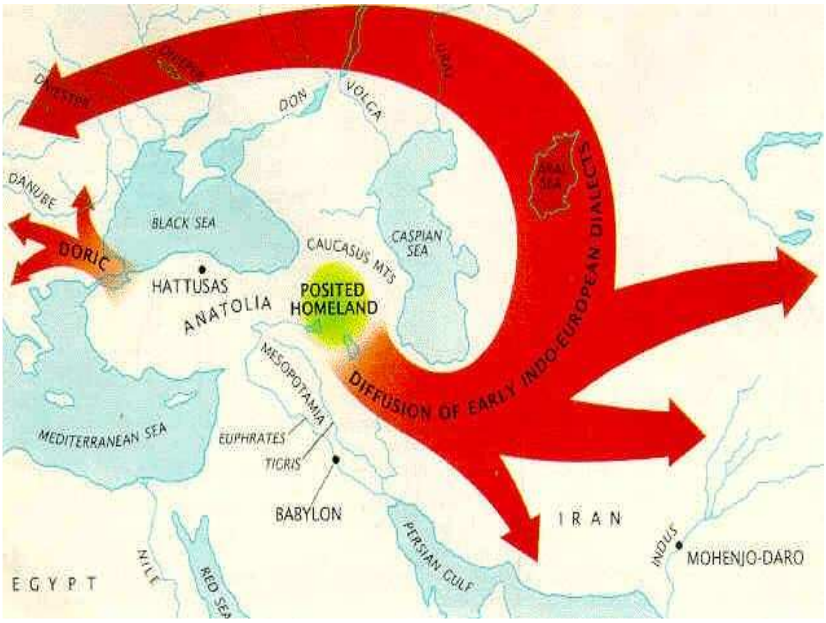

Figure 3.1: Migrations of the Indo-European populations

are descended. Their earliest literary ancestor is embodied in the Rig Veda hymns, written in an ancient variant of Sanskrit. The indigenous peoples of the Indus Valley, known from the archaeological discoveries at their capital Mohenjo-Daro, were apparently displaced by the Indo-Iranians. After the separation of the Indo-Iranians and their departure for the east, the Greek-Armenian community remained for a time in the homeland. There, judging by the numbers of loan words, they had contact with speakers of Kartvelian, Tocharian and the ancient Indo-European languages that later evolved into the historical European languages. One such borrowing from the Kartvelian became the Homeric koas, "fleece." A bilingual cuneiform tablet found in the Hattusas archives records the mythological tale of a hunter in the then already dead Hurrian language along with a translation into Hittite. This remarkable discovery gave us the Hurrian word ashi from which Homer's askos, for "hide" or"fur," apparently stemmed. Before their migration to the Aegean, the Greeks borrowed the Hittite word kursa, which by a familiar phonological shift became bursa, another synonym for "fleece." These words seem to confirm the Greeks' belief that their ancestors had come from western Asia, as recounted in the myth of Jason and the Argonauts, who sought the Golden Fleece in Colchis, on the eastern shore of the Black Sea. The evidence that the Greeks came thence to their historical homeland puts the Greek "colonies" on the northern shore of the Black Sea in a new light. The colonies may now be considered as very early settlements that were established when the Greeks began migrating to their final home in the Aegean. The historical European languages-those that left literary remains- provide evidence that the dialects from which they descended had found their way into central Asia along with the Tocharians. These languages have many words in common. An example is the word for "salmon," once regarded as a weighty argument for a homeland in northern Europe. Salmon abounded in the Baltic rivers of Europe, and the word lax (German Lachs) in the Germanic languages is perhaps echoed by lak, in Hindu, for a lacquer of a pink color that evokes the color of salmon flesh. One species of salmon, Salmo trutta, is found in the streams of the Caucasus, and the lak-s- root denotes "fish" in earlier and later forms of Tocharian as well as in the ancient European

languages. The migration of the speakers of some of the early Indo-European dialects into central Asia is established by loan words from the ric language family, which gave rise to modern Finnish and Hungarian. Under the influence of Finno-Ugric, Tocharian underwent a complete transformation of its system of consonants. Words in the ancient European languages that are clearly borrowed from the Altaic and other languages of central Asia give further testimony to the sojourn of their speakers there. Circling back to the west, the ancient Europeans settled for a time north of the Black Sea in a loosely federated community. Thus, it is not entirely wrong to think of this region as a second homeland for these peoples. From the end of the third through the first millennium B.C., speakers of ancient European languages spread gradually into Europe. Their coming is demonstrated archaeologically by the arrival of the seminomadic "pit grave" culture, which buried its dead in shafts, or barrows.

3.3 Statistical Langauge Features

In general, we model a language as a distribution of probability over the set of all possible words from the (english) alphabet. A document is a sequence from the alphabet, augmented with the symbol 'blk' for the blank-space. We define:

Definition 3.3.1

- *Alphabet:* A=*{a, b, c, d, ... y, z, blk }*
- A_L^k *is a set of words of length k over alphabet A in a particular language* L
- $A_L^* = \bigcup_k A_L^k$

Definition 3.3.2 *Sentence or sequence of words in a particular language L:* $s^L \subset A_L^*$

Definition 3.3.3 *A document in a particular language L:*

$d_i^L = \{s_{i,j}^L | j = 1, ...m_i\}$ *where* m_i *is the number of sentences in the document*

A document is a sequence of sentences, and the length of the document depends from the number of sentences.

It has been known for a long time that the probability of observing a certain character in a linguistic sequence s^L depends strongly on the previous characters, and also is highly dependent on the language in consideration [71]. The frequency with which di-grams (pairs of letters) appear in a language is a very stable property of that language, as is a related quantity known as Karlin's odds ratio in genome analysis. We denote by $C(i, j)$ the number of times that the di-gram (ij) is observed in the document. We can then define a *di-gram frequency* matrix as the matrix whose entry $D(i, j) = \frac{C(i,j)}{(n-1)}$ (where n is the document length). The *odds-ratio* matrix is defined as follows:

$$K(i, j) = \frac{D(i, j)}{P(i)P(j)}$$

where $P(i) = \frac{C(i)}{n}$ is the frequency of a letter i in a document and $C(i) = \sum_j C(i, j)$.

We want to investigate the use of D and K as statistical signatures of a language. We will also use them to assess the proximity between languages, and this means that we need to introduce a concept of distance that is appropriate in the space of matrices $\Re^{27\times27}$. We are in this way defining a metric space where we "embed" a language, and we model language evolution as a trajectory in that space. All this can make sense, however, only if these features are stable: they should be properties of the language, and not of the given document; and they should be able to distinguish between languages. If that can be proven, we can analyze phylogenetic relations between languages in this representation.

An upgrade of this approach involves to the use of something more complex than di-gram. In general, k-gram, with k bigger than 2, can be an interesting choice. A k-gram is a sequence of k letters, and they can be

treated in the same way of di-gram. This representation has some problems, in fact, the space of matrices is not $\Re^{27\times 27}$, but it becomes $\Re^{27^k}$. We pass from a flat space (matrix) to a k dimensional space (cube). That implies the increase of computational time to implement the representation and to compute the distance between languages. A very elegant solution to all these problems is given by kernel methods [72]. This class of algorithms works by efficiently computing the inner products between the images of the sequences in their feature space, by-passing the need to write down their coordinates explicitly. Hence, the embedding of the sequences in the feature space never needs to be carried out explicitly. This fact, together with recent advances in the domain of kernel methods, makes it possible to perform various types of analysis on large sets of sequences. We analyze this approach in section 3.4.3.

3.3.1 Suitability of SLS as Features

Each language has its own statistical signature. In english, digrams such as "th" and "ed" are very frequent, in italian the typical endings in vowels can be seen as high frequencies of digrams "a-", "e-" etc (where we represented the blank symbol by "-"). These differences, that reflect grammatical, phonetic and historical factors, can be readily seen in the feature matrices of the two languages, figure 3.2 and 3.3.

To test the stability of these features within a language, as well as their reliability as discriminators between languages, we have used our first corpus: a set of 50 documents (10 each for English, German, Spanish, Italian and French). We computed the average pairwise distance for documents in the same language and for documents in different languages. We than compared their ratio with the same quantity measured for randomly created sets of 10 documents. We repeated this 10,000 times, and each time the resulting ratio was larger: with p-value < 0.0001 this representation is well correlated to the difference between languages. Indeed, this quantity has been used to implement language classification systems for a long time [73].

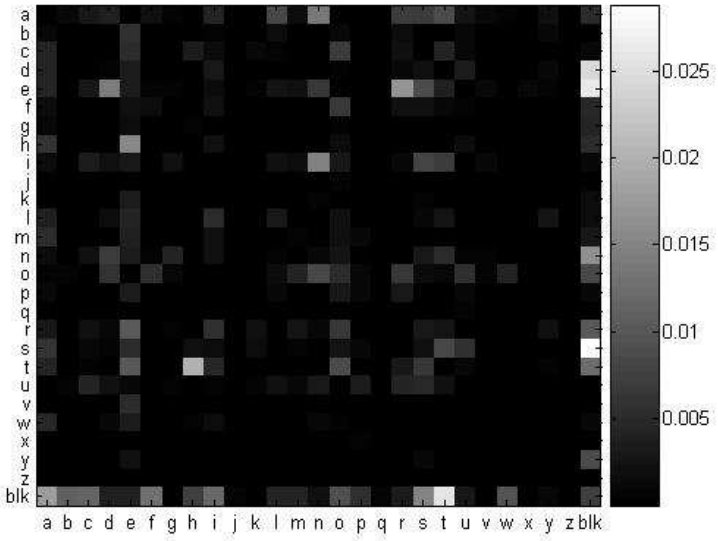

Figure 3.2: English Matrix

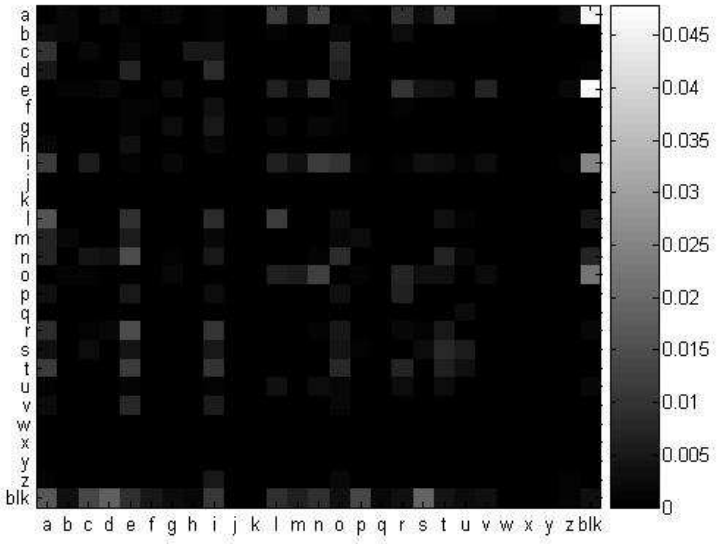

Figure 3.3: Italian Matrix

3.4 Statistical Distance between Languages

After Choosing a signature for each language, we define three different ways of measuring the distance between them. With these definitions, we can model a language as a point in a space, and its evolution as a trajectory in that space. We could even measure its rate of movement, in principle, since we have a notion of distance. Certainly we can define language similarity, and use that as a proxy in phylogenetic reconstruction.

3.4.1 Frobenius Norm and Distance

The Frobenius norm is a matrix norm defined as the square root of the sum of the absolute squares of its elements:

$$\|M^L\|_F = \sqrt{\sum_{i=1}^{\sharp A}\sum_{j=1}^{\sharp A}|m_{ij}^L|^2}$$

We can define the *Frobenius Distance Matrix* as

$$D_F(M^1, M^2) = \|M^1 - M^2\|_F = \langle (m_{i,j}^1 - m_{i,j}^2), (m_{i,j}^1 - m_{i,j}^2) \rangle =$$

$$\sqrt{\sum_{i=1}^{27}\sum_{j=1}^{27}|m_{ij}^1 - m_{ij}^2|^2}$$

The Frobenius distance is equal to the Euclidean distance between matrices. This distance counts only the difference between the occurrences of the same feature in the different matrices. If two matrices have a lot of common features, they are more similar than matrices with few common features.

3.4.2 Karlin or 1-Norm Distance

The 1-norm distance is the Minkowski distance of order 1. In the general way, we can define the Minkowski as:

$$D_{Mnk}(M^1, M^2) = (\sum_{ij} |m^1_{i,j} - m^2_{i,j}|^p)^{\frac{1}{p}}$$

The 1-norm is:

$$D_{Mnk}(M^1, M^2) = \sum_{ij} |m^1_{i,j} - m^2_{i,j}|$$

In bioinformatics, the 1-norm distance is used to compute the distance between two sequences f and g from different organisms or from different regions of a single genome. In [70], the authors use a particular variation of the 1-norm distance, in fact they normalize the 1-norm using the square of the total symbols inside the alphabet A. We adapt this metric to languages where the alphabet has 27 symbols:

$$D_{L1}(M^1, M^2) = \frac{1}{(27)^2} \sum_{ij} |m^1_{i,j} - m^2_{i,j}|$$

3.4.3 Kernels on Strings

We consider the problem of embedding two sequences in a high-dimensional space in such a way that their relative distance in that space reflects their similarity, and that the inner product between their images (and thus also the distance between them) can be computed efficiently. The first decision to be made is what similarity notion should be reflected in the embedding, or in other words, what features of the sequences are revealing for the task at hand. Are we trying to group sequences by length, composition, or some other properties? What type of patterns are we looking for? This section will describe a few possible embeddings that are suitable for the sequences and tasks dealt with in this thesis.

The different approaches used here reduce to various ways of counting substrings or approximate substrings that the two strings have in common.

This is a meaningful similarity notion in biological applications, since evolutionary proximity is thought to result both in functional similarity and in sequence similarity, measured by the number of insertions, deletions and symbol replacements, as well as large rearrangements. Measuring sequence similarity should therefore give a good indication about the functional similarity that bioinformatics researchers would like to capture. Furthermore, the space of substrings in a text of a given language captures certain statistical properties that are known to be highly language-specific (as opposed to topic specific) and that can be expected to evolve slowly. Therefore, measures of similarity based on substring counts can be used to study evolutionary distances between languages as well.

The sequence kernels used in this chapter were introduced in [74] and are reviewed in [72].

Embedding space. All the kernels presented here can be defined by an explicit embedding map from the space of all finite sequences over an alphabet Σ to a vector space F. The coordinates of F are indexed by a subset I of strings over Σ, that is by a subset of the input space. In the following we will study embeddings where I is the set Σ^p of strings of length p, giving a vector space of dimension $|\Sigma|^p$. We use ϕ to denote the feature mapping

$$\phi\colon s \in \Sigma^* \longmapsto (\phi_u(s))_{u \in I} \in F.$$

For this embedding space F, there are several different maps ϕ one can choose from. For example, one possibility is to set the value of the coordinate $\phi_u(s)$ indexed by the string u to be a count of how many times u occurs as a contiguous substring in the input string s. A second possibility is to count the number of approximate matches, allowing a certain maximum number of mismatched characters.

Kernel definitions

The p-spectrum kernel. Perhaps the most natural way to compare two strings in many applications is to count how many (contiguous) substrings of length p they have in common. We define the *spectrum of order p* (or *p-spectrum*) of a sequence s to be the histogram of frequencies of all its (contiguous) substrings of length p. We define the p-spectrum kernel as the inner product of the p-spectra. Formally, the feature space F associated with the p-spectrum kernel is indexed by $I = \Sigma^p$, with the embedding given by

$$\phi_u^p(s) = |\{(v_1, v_2) : s = v_1 u v_2\}|,\ u \in \Sigma^p$$

and the associated p-spectrum kernel between sequences s and t is defined as

$$\kappa_p(s,t) = \langle \phi^p(s), \phi^p(t) \rangle = \sum_{u \in \Sigma^p} \phi_u^p(s)\, \phi_u^p(t).$$

The mismatch kernel. When isolated substitutions are likely to occur in a sequence, the p-spectrum kernel might be too stringent to result in a useful similarity measure. In those cases, it makes sense to use a modification of the p-spectrum, where the feature of a sequence s associated to the substring u is equal to the number of contiguous substrings in s that differ by no more than a maximal number m of characters from u. For two substrings u and v of equal length, we use $d(u,v)$ to denote the number of characters in which u and v differ. The mismatch kernel $\kappa_{p,m}$ is defined by the feature mapping

$$\phi_u^{p,m}(s) = |\{(v_1, v_2) : s = v_1 v v_2 : |u| = |v| = p,\ d(u,v) \leq m\}|.$$

The associated mismatch kernel is defined as

$$\kappa_{p,m}(s,t) = \langle \phi^{p,m}(s), \phi^{p,m}(t) \rangle = \sum_{u \in \Sigma^p} \phi_u^{p,m}(s)\, \phi_u^{p,m}(t).$$

Normalization. The norm of these feature vectors is affected by the length of the sequences. In order to remove this influence we can normalize the feature vectors. This can be done by the following kernel operation

$$\kappa_n(s,t) \leftarrow \frac{\kappa(s,t)}{\sqrt{\kappa(s,s)\kappa(t,t)}}.$$

Distance in feature space. The algorithms we will use are based on distance matrices, while the kernel functions represent inner products in a feature space. However, given the kernel values, the distance between the feature vectors corresponding to two sequences s and t can be computed as

$$d(s,t)^2 = \kappa(s,s) + \kappa(t,t) - 2\kappa(s,t).$$

Kernel matrix and distance matrix. We define the kernel matrix $\mathbf{K}$ and the distance matrix $\mathbf{D}$ as the matrices whose entries represent the similarities and distances (respectively) between all pairs of sequences.

The trie-based implementation

Direct implementations of these kernels would be very slow to evaluate due to the potentially large dimensionality of the feature space, which is exponential in p. Fortunately however, much faster implementations of string kernels can be obtained by exploiting an efficient data structure known as a 'trie'. A *trie* over an alphabet Σ is a tree whose edges are labeled with a symbol from Σ. A *complete trie of depth p* is a trie containing the maximal number of nodes consistent with the depth of the tree being p, each parent node having a downward branch for each alphabet symbol from Σ.

In a complete trie there is a one to one correspondence between the nodes at depth k and the strings of length k, the correspondence being between the node and the string on the path to that node from the root (The string associated with the root node is the empty string ε). Hence, we will refer to the nodes of a trie by their associated string. The key observation behind

the trie-based approach is that one can regard the *leaves* of the complete trie of depth p as the indices of the feature space indexed by the set Σ^p of strings of length p. So the coordinates of the vector $\phi(s)$ (corresponding to the dimensions of the feature space F) are conveniently organized in the trie, which can be used to obtain an efficient search strategy.

The p-spectrum kernel. Let us use this data structure to compute the p-spectrum kernel. The trie-based implementation explores the substring features (corresponding to the trie's leaves) by traversing the trie in a depth-first fashion, each time attaching to the explored node a list of substrings of s that match with the substring corresponding to that node. Initially, a list of all $|s|-p+1$ contiguous length p substrings of s is attached to the root node. Then, recursively, the list of matching substrings of a child node at depth k in the trie can be computed by checking for each of the substrings in its parent's list if the kth character matches the character corresponding to the branch connecting them. Once in a leaf node, the feature value corresponding to it can easily be read of as the number of matching substrings in the list attached to it.

Such lists can be maintained simultaneously for s and t between which we want to compute the kernel. Then, each time a leaf node is reached, the kernel function between them (initially set to zero) can simply be incremented by the product of their feature values for that leaf node.

The depth first strategy to explore the trie has two advantages. First, the potentially long feature vector never has to be stored in memory as a whole. And second and more importantly, for large p, many of the features will be equal to zero and are thus irrelevant in the kernel computation. Very often, this can already be noted early in the tree, when a list of matching substrings at an internal node is empty. Then, the entire space of substrings below this internal node can be ignored. Especially for large p, this allows one to prune the explored space of substrings significantly.

Computational cost. Of course, the lists of matching subsequences attached to the nodes are not constructed explicitly, but instead a vector of

$(|s|-p+1)+(|t|-p+1)$ pointers to their starting positions in the strings s and t are maintained. This makes the complexity of the preprocessing equal to $O(|s|+|t|)$. During the trie traversal, each of the $(|s|-p+1)+(|t|-p+1)$ substrings processed gets passed down at most p times, so the complexity of the main processing is $O(p(|s|-p+1+|t|-p+1))$ giving an overall complexity of

$$O(p(|s|+|t|)).$$

At first sight there appears to be a contradiction between this complexity and the size of the feature space $|\Sigma|^p$, which is exponential in p. The reason for the difference is that even for moderate p there are far fewer substrings present in a sequence s than there are leaves in the trie. In other words, the vector $\phi(s)$ is very sparse. Hence, as explained above, the recursive algorithm will rapidly find subtrees that are not populated, i.e. internal nodes u for which one of the lists is empty. The algorithm effectively prunes the subtree below these nodes since the recursion halts at this stage. Hence, while the complete tree is exponential in size, the algorithm only needs to process $O(p(|s|+|t|))$ of the nodes.

Mismatch kernel. In order to apply the trie-based approach to the mismatch kernel we initialize the lists at the root of the trie in exactly the same way as for the p-spectrum kernel, except that each substring in the list has a number attached indicating the number of mismatches allowed so far (equal to zero at the root node). The key difference in the trie traversal is that when we process a substring it can be added to lists associated with more than one child node, though in all but one case the number of mismatches will be incremented [74].

Computational cost. The complexity of the algorithm has been somewhat compounded by the mismatches. Each substring at a node potentially gives rise to $|\Sigma|$ substrings in the lists associated with its children. If we consider a single substring u at the root node it will reach all the leaves that are at a distance at most m from u, in terms of number of mismatches. There

are

$$\binom{p}{k}|\Sigma|^m = O\left(p^m|\Sigma|^m\right)$$

such strings. So, starting from a single substring at the root node, we can bound the total number of characters to be checked by

$$O\left(p^{m+1}|\Sigma|^m\right).$$

Hence the complexity of the overall computation is bounded by

$$O\left(p^{m+1}|\Sigma|^m\left(|s|+|t|\right)\right)$$

taking into account the number of substrings at the root node. Clearly, we must restrict the number of mismatches if we wish to control the complexity of the algorithm.

Computing an entire kernel matrix. Instead of maintaining a list for two strings s and t at each internal node of the trie, we can maintain a list for a whole set of strings between which we want to compute the kernel functions. Whenever a leaf node is reached, all these kernel functions can be incremented based on the feature values of each of the strings corresponding to that leaf node. This can be carried out efficiently: the traversal of the trie remains linear in the sum of the lengths of all strings, and only the operations at the leaf nodes, where the kernel values are incremented, is inherently quadratic in the number of strings. The result is the full kernel matrix $\mathbf{K}$, containing the kernel function between the ith and jth sequences at position (i, j) and symmetrically at position (j, i). Normalized kernel and distance matrices can then promptly be computed from it.

3.5 Output Algorithms

3.5.1 Neighbor Joining

The neighbor joining algorithm is a standard method in computational biology for reconstructing phylogenetic trees based on pairwise distances between the leaf taxa. It is guaranteed to reconstruct phylogenetic trees exactly when the pairwise distances are treelike. It is due to Saitou and Nei (1987) [75] and has been modified by Studier and Keppler (1988) [76]. The distance matrix between leaf nodes is all that is needed for it to work.

3.5.2 Multidimensional Scaling

Multidimensional scaling (MDS) [77] is a visualization tool for the exploratory analysis of high-dimensional data. It can be used to display the data points in a two-dimensional plane in which distances between the data points are conserved as accurately as possible. The standard approach to MDS makes use of the Euclidian distance, and reduces to the computation of the dominant eigenvectors of the distance matrix between the data points.

We applied the standard MDS approach to the distance matrices as computed from all our metrics. In this way one can get a visual impression about the relative distances between the different sequences, even though the dimensionality of their feature spaces is extremely high-dimensional.

3.6 Dataset

We use two different dataset. The first is made of 50 documents written in 5 languages: Italian, Spanish, French, Germanic and English. Documents have been downloaded from some on-line newspapers. The second language dataset has been introduced by [78]. Their dataset is made of the translation of the “Universal Declaration of Human Rights" [79] in the most important branches, Romance, Celtic, German, Slavic, Ugrofinnic, Altaic, Baltic, and

the Basque language. Our dataset contains 42 languages from this dataset. (See [80, 81, 82] for similar studies with different methods.)

Each document of both datasets has been downloaded in html format. These files have been cleaned from all tags using a parser of our implementation. As a result, we obtained a sequence of words without punctation for each document. We substituted the ASCII characters with only 27 characters: the 26 lower case letters plus the space character. This means that all the upper case letters have been replaced by lower case, and the stressed letters have been substituted by the corresponding letters not stressed (e.g. à, ç, è, became a, c, e), see table 3.2. The other characters that do not belong to our character set have been deleted. The average length of the preprocessed texts is 11.022 characters, and the average number of words is 1768.

Stressed Letters	*Substituted Letter*
ć, č, ç	c
š, ś	s
ž, ż, ź	z
d, ð	d
à, ă, â, å, á, ã, ä, ą	a
ý	y
ù, ú, û, ủ, ű, ü	u
í, î, ì, ï, i	i
è, é, ê, ë, ę, ě	e
ř	r
ô, ó, ò, õ, ő, ö, ø	o
ł	l
ń, ñ, ň	n
æ	ae
θ	t
ġ	g
ß	ss
Þ	th

Table 3.2: Conversion Table.

The fact that each document of the second dataset is a translation of the "Universal Declaration of Human Rights" offers the advantage that they all

have roughly the same length, which facilitates our statistical analysis. The disadvantage however, is that in very close languages, the translation of a word can be the same, or have the same root. This means that our estimated distances for adjacent/far languages may end up closer/farther than other results, where the dataset is made from different documents. Notice that we have included a few non-Indo-European languages, some of the Turkic branch and the complete Finno-Ugric family.

3.7 Results

Using the methods shown above, rooted and unrooted trees and MultiDimensional Scaling are presented for each SLSs. We compare all these plots, showing the strength of our feature and our distance metric. All the rooted trees have been generated using the public software tool NJPlot and all the unrooted trees have been generated using the public software tool Unrooted [83]. These public software implement the Neighbor Joining algorithm.

All the trees that we obtained are mostly compatible with the standard organization of the IE family. That means that not only can our SLS characterize a language, but can act as tags to track its evolution over long periods of time. Clearly this quantity seems to be changing slowly, and we can see from the fine organization of the slavic family or from the organization of languages in the iberian peninsula, it seems to also have a fairly steady drift. It is interesting to note that also the violations of the accepted topology of the tree can give us information about language evolution. For example, languages such as Romanian and English clearly are the result of massive borrowing from nearby languages, and are no longer assigned to their original family (at least not their lexicon, which is what is captured mostly by these representations).

In the *di-grams* representation (figures 3.4, 3.5 and 3.6) there are various problems. Two of these are the incorrect assigning of Icelandic and Albanian. Icelandic belongs to the Scandinavian family, which is a part of the Germanic branch, but it is inserted in to the Turkic family. Albanian constitutes a

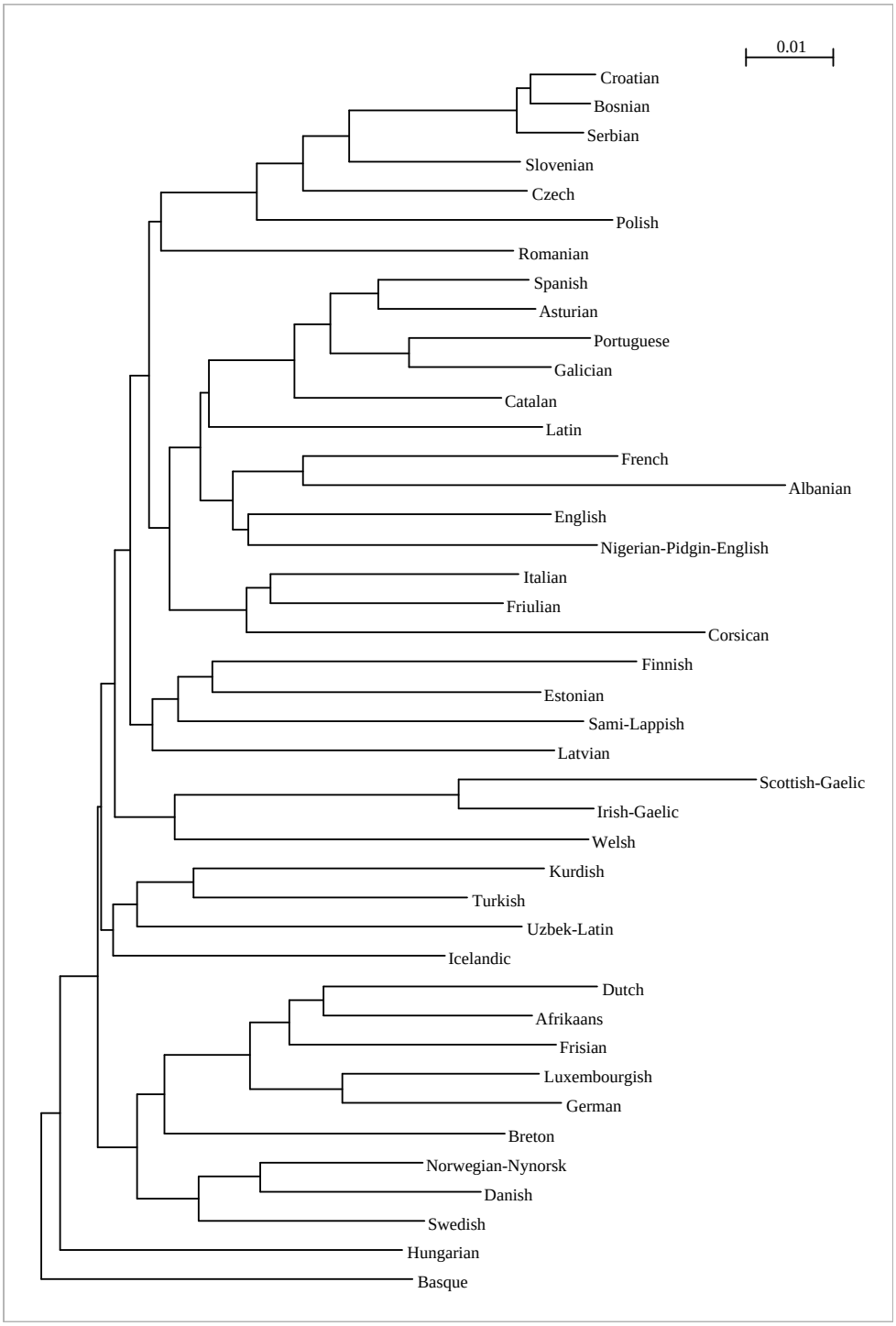

Figure 3.4: The evolutionary rooted tree built using di-gram frequency and Frobenius distance.

3.7 Results

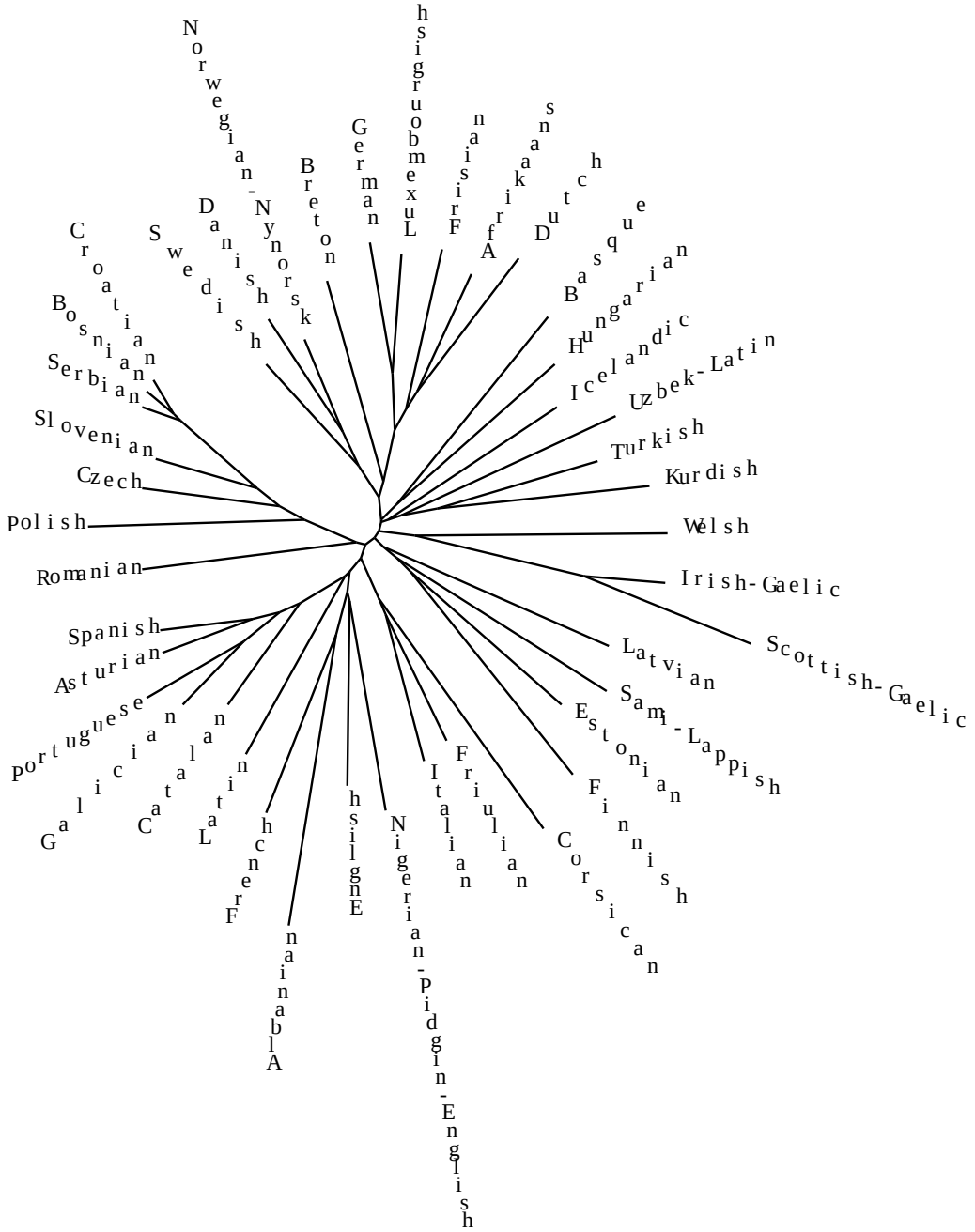

Figure 3.5: The evolutionary unrooted tree built using di-gram frequency and Frobenius distance.

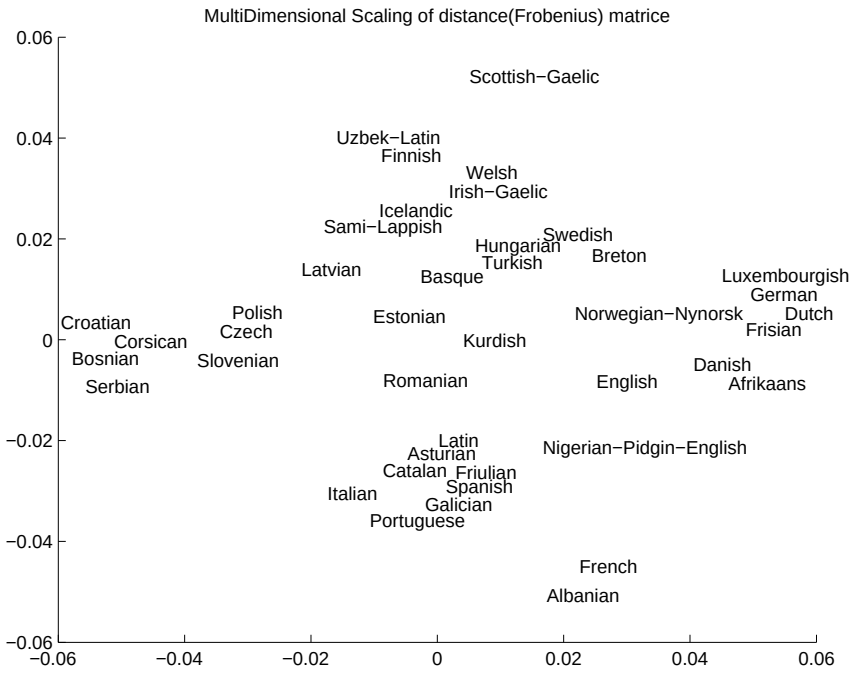

Figure 3.6: MDS of 42 languages using di-gram frequency and Frobenius distance.

separate and independent branch, so there are no reason because it has to be in the core of Romance family. Also the position of the pair English, Nigerian-Pidgin-English is not correct, and also if we know that the words's borrowing can move these languages close to French, they can not be close to French more than the Italian family. Clear, we can see the phenomena of words's borrowing for Romanian. A different argumentation is necessary for Breton, we will analyze it at the end of this section. This representation defines in the correct way the Germanic, Celtic, and Slavic clusters. Finnish, Sami-Lappish, and Estonian belong to the same branch: the Uralic, a subset of the Finno-Ugric languages. In figures 3.4, we keep this relation, but we are not able to join with them the Hungarian.

Using the *odds-ratio* (figures 3.7, 3.8 and 3.9), we have some interesting improvements. This kind of representation seems to be more reactive to fine differences between languages emphasizing the geographic proximities. Romanian is attracted by Albanian, and it is between the Turkic and Slavic family. Latvian needs a quick analysis; it belongs to the Baltic group with Lithuanian, but the absence of Lithuanian in our dataset causes the proximity of Latvian to Czech and Polish in the Slavic family. The Romance family is clear, and English is close to that. The Finno-Ugric languages have been detected, with Hungarian with Estonian and Finnish, but the Sami-Lappish is in the wrong place inside the Germanic branch. Here Icelandic has been correctly gathered from the Scandinavian branch.

The experiments with kernels were performed with p-spectrum and mismatch kernels, with value of $p = 4$, allowing one mismatch. The results obtained with the p-spectrum kernel (figures 3.10, 3.11 and 3.12) show clearly the Germanic, Romance Slavic and Celtic branch and the not Indo-European family Turkic and Finno-Ugric. English continuous to be close to the Romance family, Romanian and Albanian belong to the same branch and Latvian is near to the Slavic. It is interesting to notice that an analysis of the order of branching of various subfamilies, shows that our statistical analysis can capture interesting relations: the split of the Slavic languages, with Slovene diverging much earlier than Serbian, Croatian and Bosnian; the existence of a Scandinavian subfamily within the Germanic family, and its an-

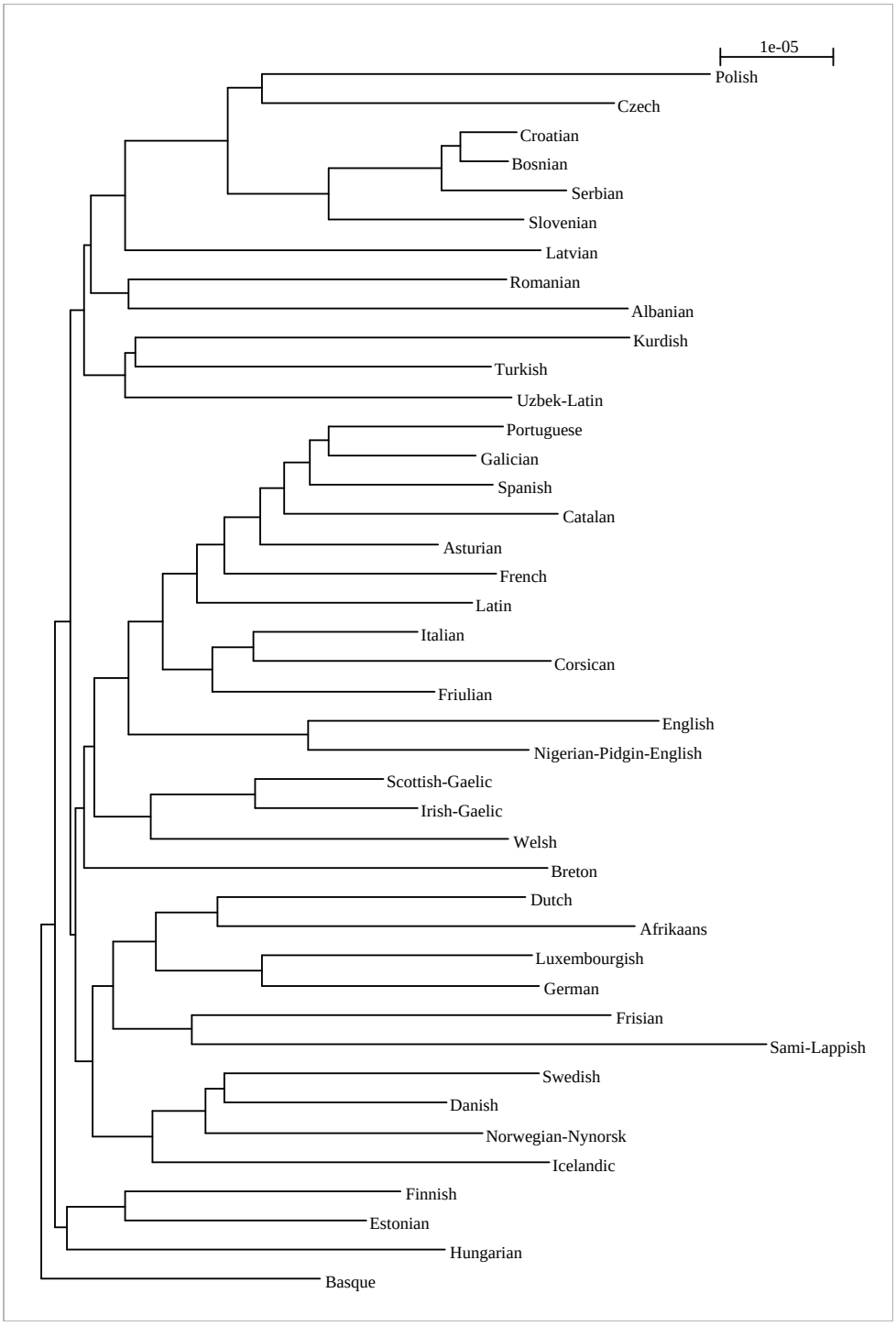

Figure 3.7: The evolutionary rooted tree built using Odds Ratio and Karlin distance.

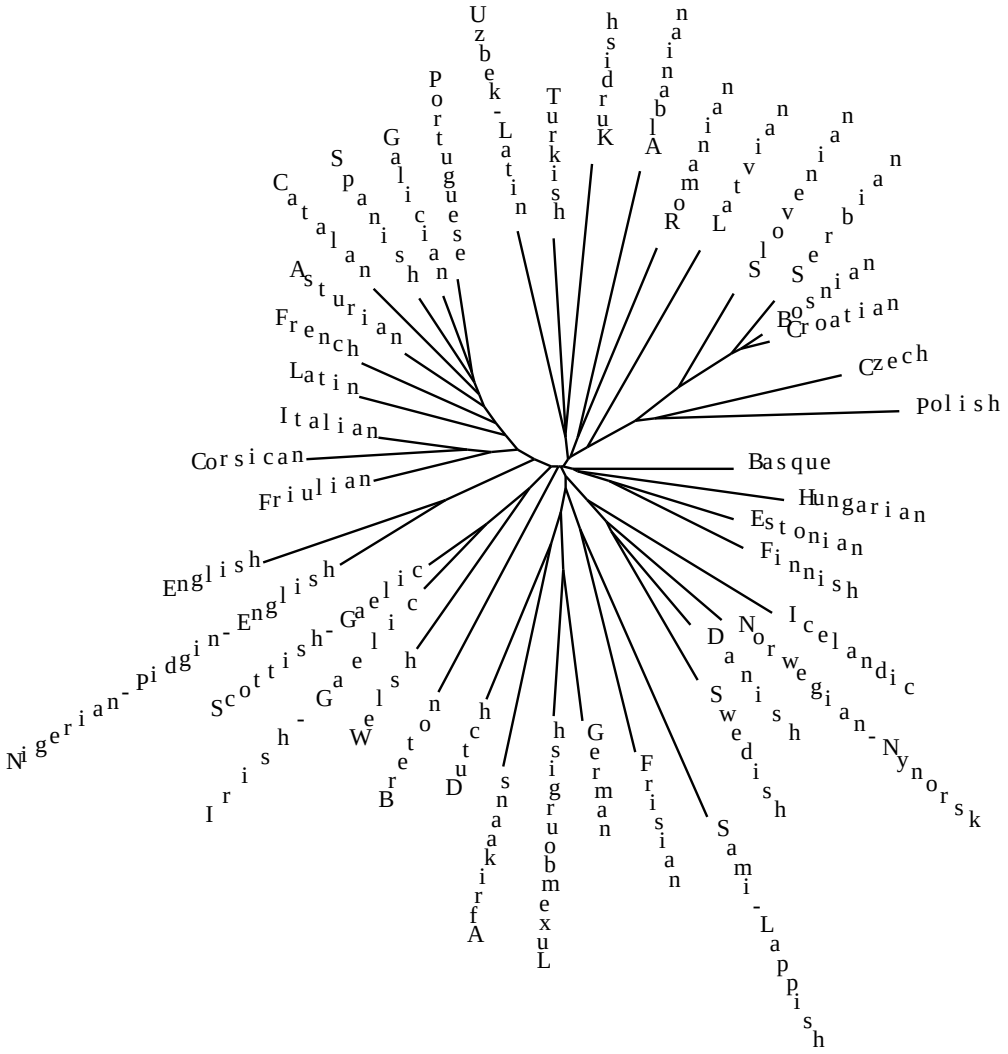

Figure 3.8: The evolutionary unrooted tree built using Odds Ratio and Karlin distance.

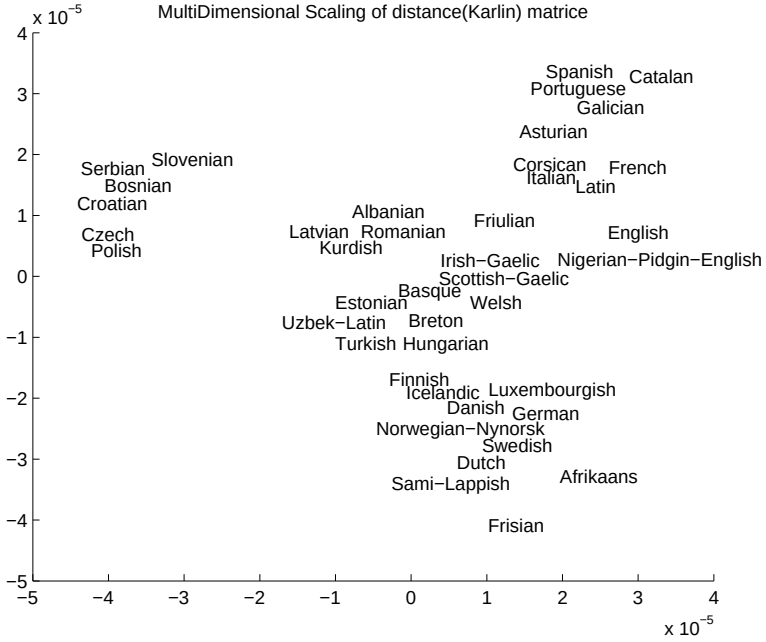

Figure 3.9: MDS of 42 languages using Odds Ratio and Karlin distance.

cient relation with Icelandic; and the very structured Romance family, with Latin equally distant from all of its children, and subfamlies in the Italian and Iberian peninsula organized to match historical developments.

Using the mismatch kernels, with value of $p = 4$, allowing one mismatch we have a light worsening of the plots (figures 3.13, 3.14 and 3.15). It has to the change of position of the Hungarian from the Finno-Ugric family. This shift underlines the recent split of the Slavic languages in the Balkans.

Breton belongs to the Celtic group, Brythonic subgroup and is spoken mainly in Brittany (France), the peninsula of westernmost France lying between the Channel and the Bay of Biscay. In our results, it is positioned close to the Germanic family (figure 3.4 , figure 3.7 and figure 3.13) and only in figure 3.10 it is in the right place close to the Celtic group. In particular we mapped the letters to their nearest english-alphabet counterpart, without using a linguistic criterion. Our assumption was that given the inherently statistical nature of the approach, we could ignore at a first approximation

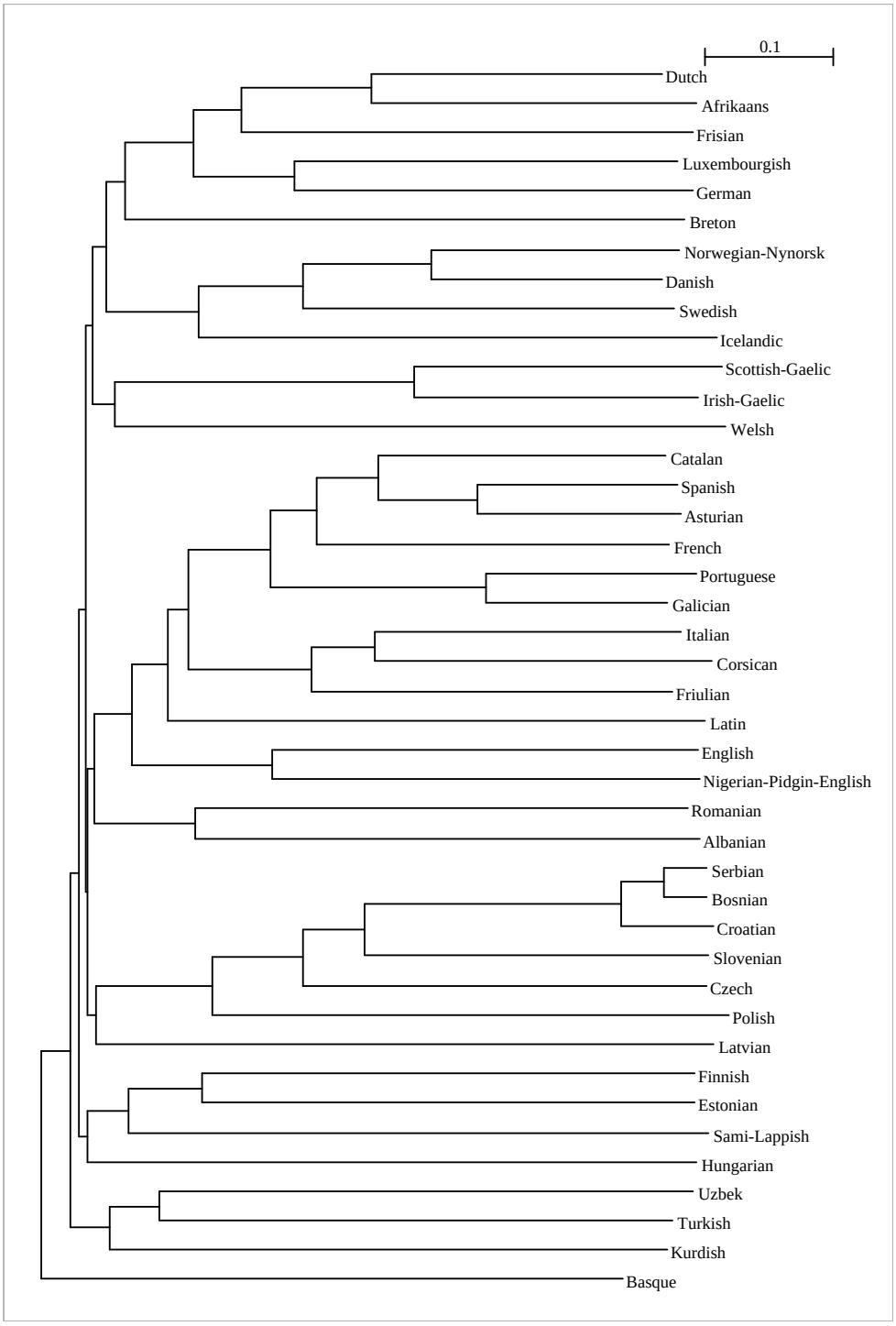

Figure 3.10: The evolutionary rooted tree built using a 4-spectrum kernel.

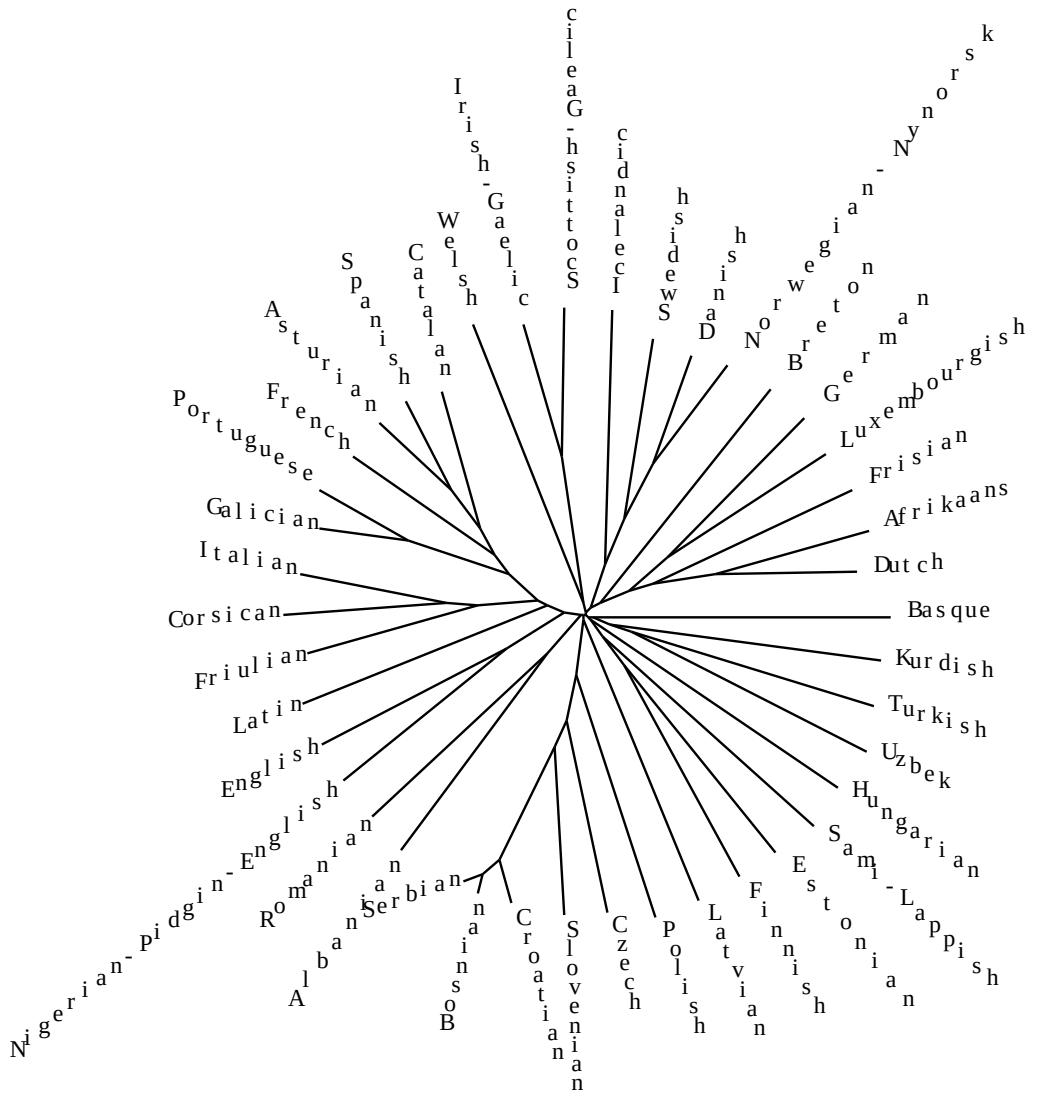

Figure 3.11: The evolutionary unrooted tree built using a 4-spectrum kernel.

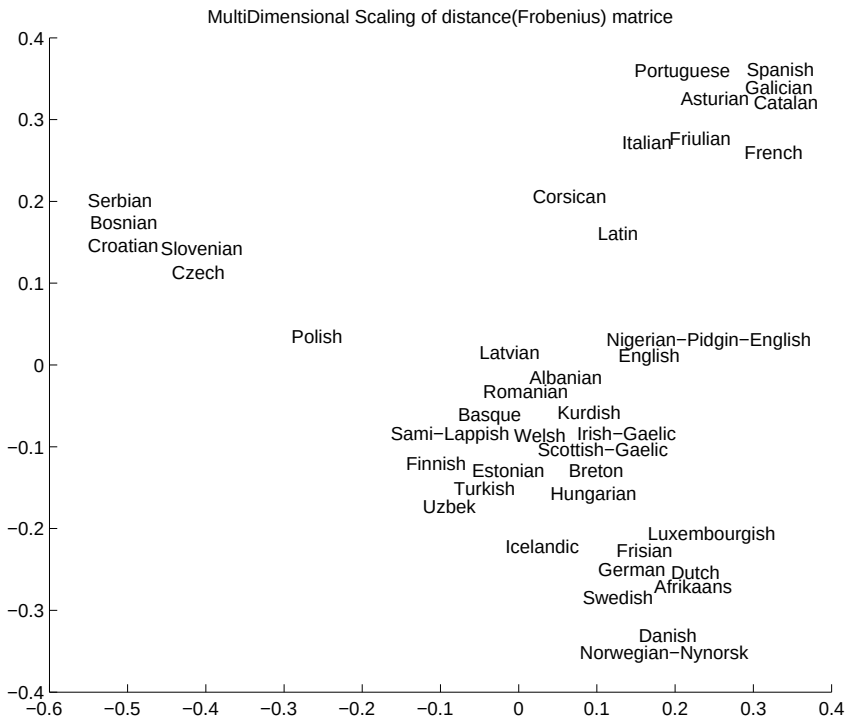

Figure 3.12: MDS of 42 languages using a 4-spectrum kernel distance matrix.

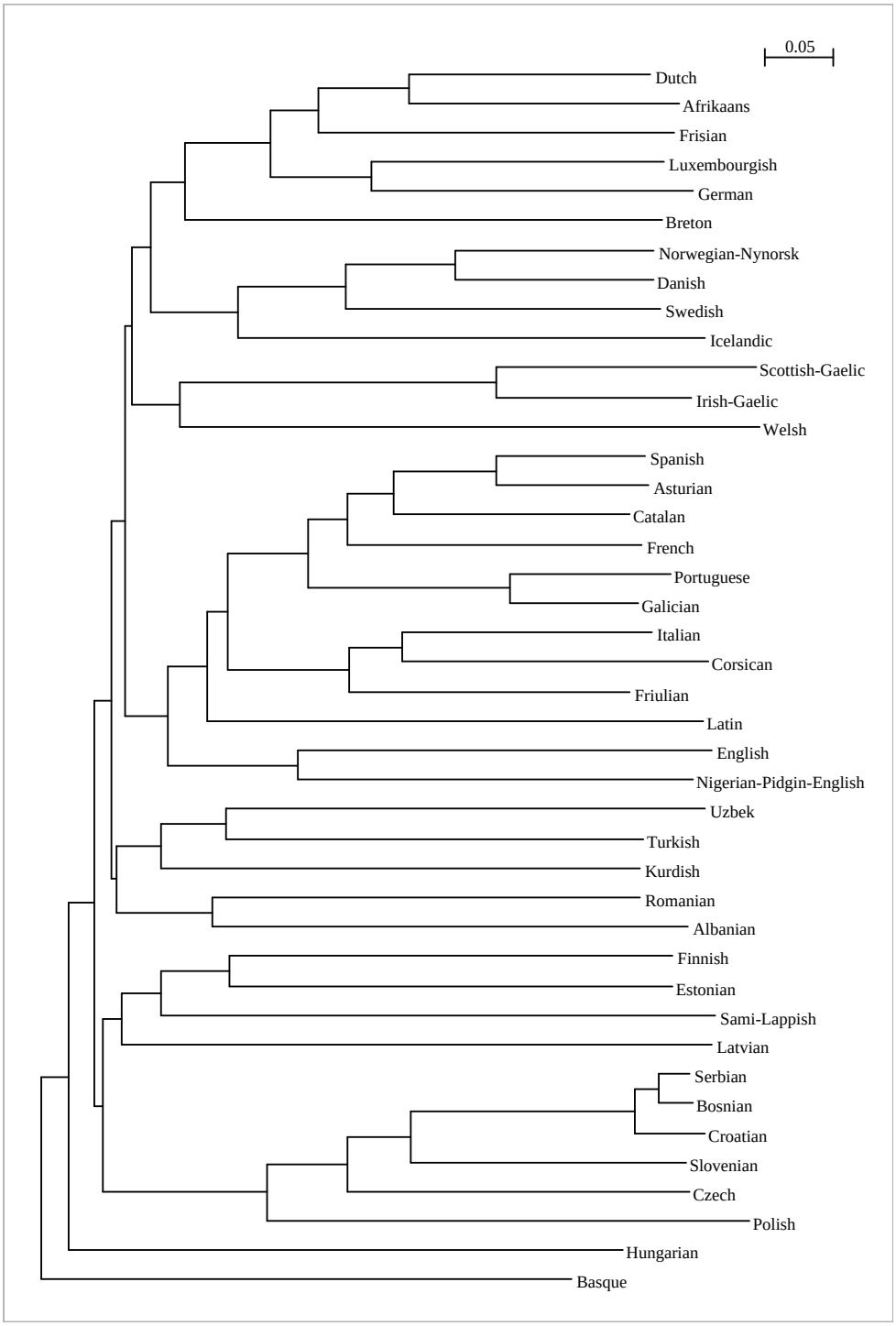

Figure 3.13: The evolutionary rooted tree built using a 4-mismatch kernel with 1 mismatch.

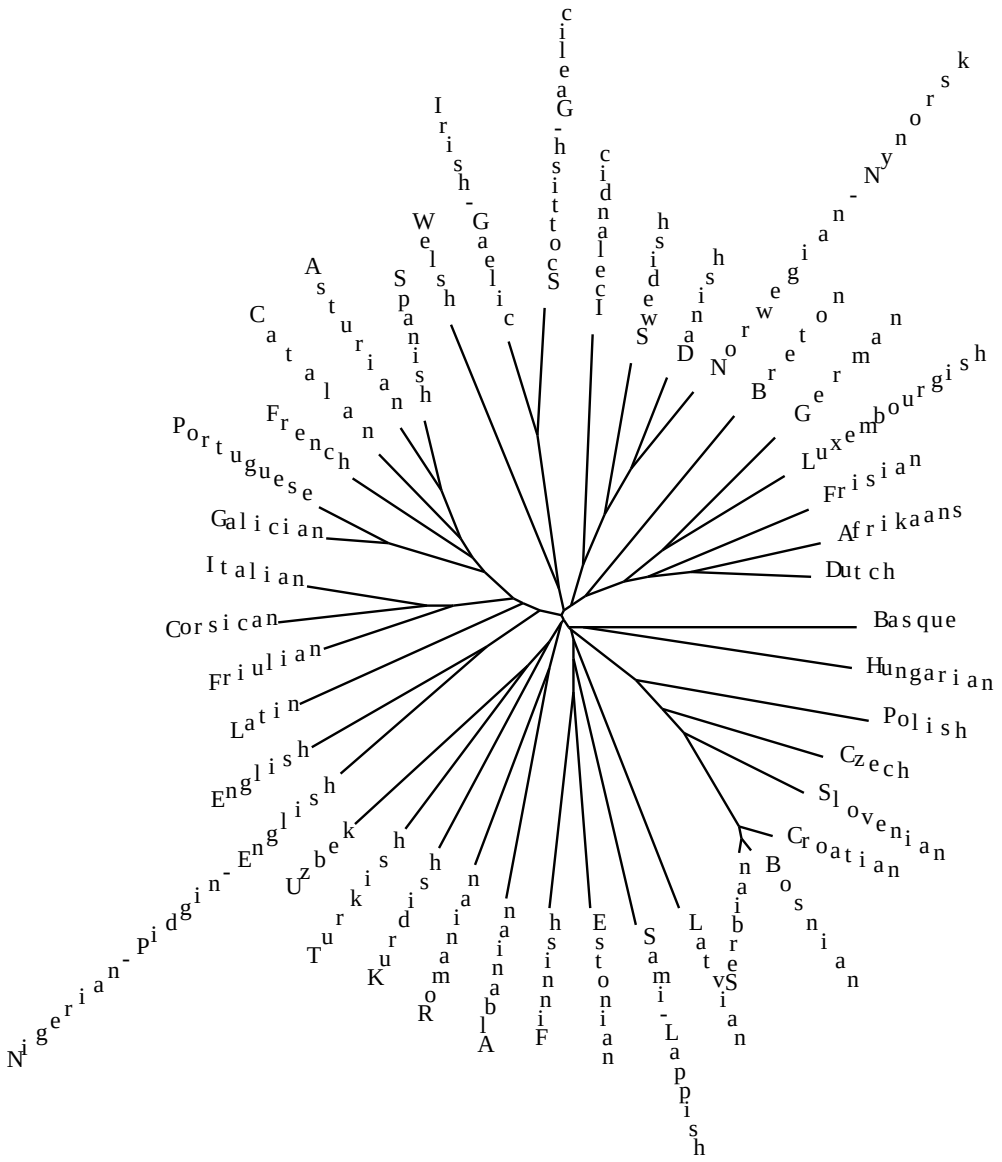

Figure 3.14: The evolutionary unrooted tree built using a 4-mismatch kernel with 1 mismatch.

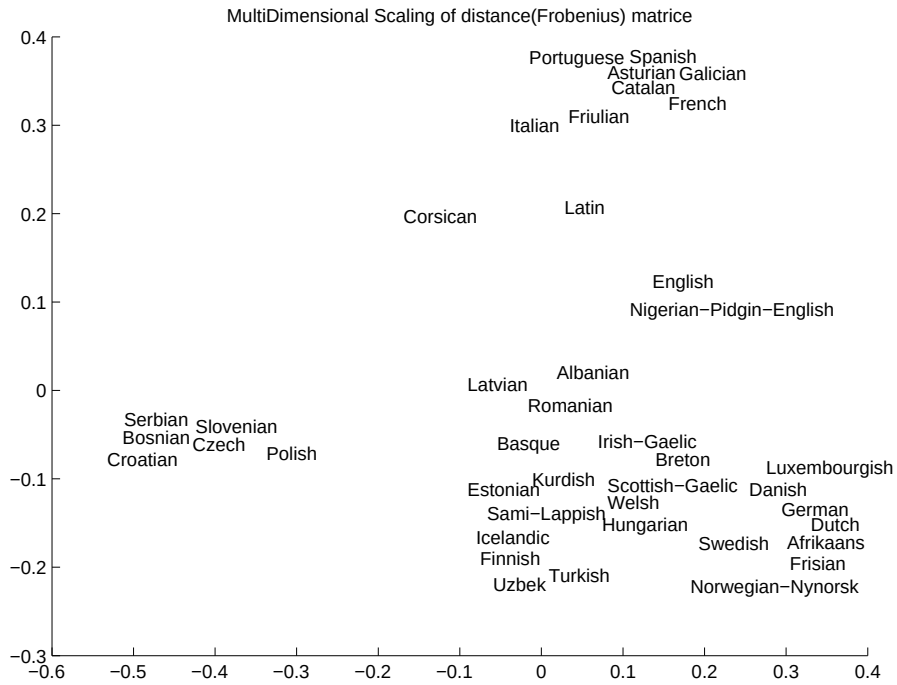

Figure 3.15: MDS of 42 languages using a 4-mismatch kernel with 1 mismatch distance matrix.

the effects of this arbitrary step, modeling them as a small perturbation of the signal. This has been the case for most languages, but Breton is an example of how this rough simplification has proven to be sufficient to mislead the algorithms. In the future, we are planning to make use of the phonetic alphabet, to reduce this effect.

Finally, we have used Basque as an outgroup, and its ancient history is revealed again in our data, where it is seen that it is essentially independent from all other languages in the set.

A look at the MDS plots, figure 3.6 , figure 3.9, 3.12 and figure 3.15, shows that English ends up half way between Romance and Germanic clusters, revealing the large influence of French on its lexicon. In figure 3.6, 3.9, 3.12, Romanian is close to both Slavic and Turkic clusters, again revealing the massive borrowing of words from those neighboring languages, in the years that followed its split from the Romance tree. In fact, languages do not evolve in trees but rather in graphs, and an interesting question is how to reconstruct them from distance matrices.

At the end, various conclusions can be drawn from the experimental results we obtained: the first one is that some aspects of historical linguistics can indeed be investigated by using statistical tools. This rises hopes of applying the same tools to ancient texts, so as to look further back in time. But at the same time, some problems with this approach are visible in the results, directly suggesting various improvements.

First, it is not always the case that this statistical approach is robust enough to ignore the effect of alternative spelling conventions (as seen in the case of Breton). This can be addressed by moving future investigations to documents written using the IPA (international phonetic alphabet). Notice however that it can be argued that even spelling conventions evolve, and are part of the phylogenetic signal we are trying to analyze, as we focus on the evolution of written text. Second, we see the effect of borrowings (as seen in the case of English and Romanian): in many cases the assumption that the evolutionary history of languages can be represented by a tree is not justified, at least with respect to their lexicon. This can be addressed by using tools

from evolutionary biology aimed at reconstructing "phylogenetic networks" rather than trees.

Because of the inherently statistical nature of this approach, however, to a first approximation we believe that all the above effects can be treated as random perturbations, and for most languages they are not sufficient to corrupt the phylogenetic signal. As we refine the method, we expect to find cleaner and more informative patterns in the data.

Chapter 4

Textual Time Series

4.1 Introduction

The second main contribution presented in this dissertation concerns the study of time series obtained from textual documents. The aim is to find particular relations inside data which explain the process related to the production of documents in times.

The main goal of the proposed techniques is to discover trends or change points in sequences of documents. We suppose that there are some topics across evolution of the documents. Each topic is represented by one or more sources that produce a sequence of terms. These sources can produce words or key-phrases for many consecutive days. A classical example is represented by world news which periodically change following the main events that occur every day. In general, when media are talking about one topic, their sources emit terms. When an important event happens, the attention of the media changes, and also the used terms, so new sources substitute totally or partially the old ones.

In table 4.1, we report the most interesting key-phrases discovered on some on-line newspapers, on July 6th 2005 and July 7th 2005 (day of the London bombings attacks). We can easily see how the elements in the second column come out from different sources than the first one. We can associate key-phrases on July 6th to some sources which can be labeled as "2012 Olympic Election" (Key-phrases: New York, Olympic committee, all

July 6th 2005	**July 7th 2006**
the US	the US
New York	Prime Minister Tony Blair
the United States	the United States
chang aa	al Qaida
Prime Minister	the London bombings
US military	Web site
All comments	Thursday July
North Korea	terrorist attacks
Olympic Committee	double decker bus
Al Jazeera	European Union

Table 4.1: The most used key-phrases on July 6th and July 7th.

comments), "Rice tour to North Korea" (North Korea, chang aa), "Iraq war" (US military, Al Jazeera) and "G8 summit" (The US, Prime Minister, the United States). Otherwise, the key-phrases on July 7th belong to "London Attacks" (Prime Minister Tony Blair, al Qaida, the London bombings, Web site, Thursday July, terrorist attacks, double decker bus) and "G8 summit" (the US, the United State, European Union). Even if the topic "G8 summit" is present in both days, for the other sources there are abrupt changes, since three of the old topics have been replaced by "London Attacks".

A new topic can show up in three different ways. The case reported above is frequent, when on a given day a set of terms focussed on the same topic appear. This change of source is abrupt and swift, but in general the event which causes it is short. In other cases, some terms can appear slowly, but with a constant increase, till they arrive at the complete "explosion" of a new topic. These events have long tails before they appear and are relatively long. The last phenomenon could be called hidden topic. In fact, the terms that belong to this topic, appear, when there are not other important events, and disappear when something happens. They are latent and emerge only in particular conditions.

In the next sections of this chapter, we show how it is possible to process the data in order to facilitate the discovery of change points and events. We propose some methods that automatically filter the data and give a time line

with the main events. They suggest easy techniques to detect change on these time lines. The last method allows to plot the trajectory in time of some key-phrases that identify clearly a topic.

4.2 Documents in Time

Each approach requires the right representation of data. This is one of the most tricky tasks in computer science, because the results depend on the capacity of the features to describe the relevant information.

In general, given a document space χ and the feature space $\Im$, we define a function $\phi : \chi \rightarrow \Im$ that maps each document into the feature space:

$$\phi(D) \rightarrow \{f_1, f_2 \dots f_k\} \subseteq \Im$$

where f_j is a feature and its value depends on the particular case where it is used. In documents analysis, each feature usually has a strong relation with a word or a set of them.

This representation of a document in the feature space is independent on the instant of time when you decide to extract $\phi(D)$. It means, see figure 4.1, that $\phi(D)$ depends just on the content of document D.

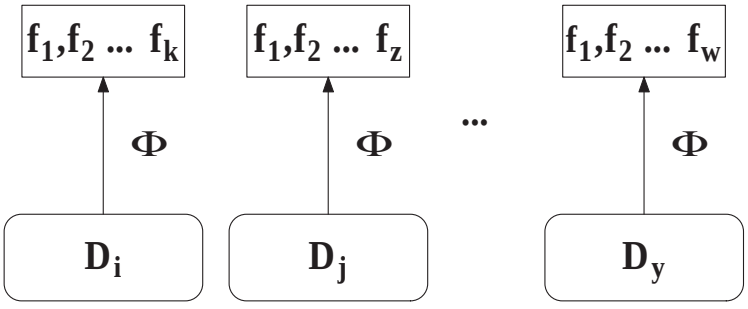

Figure 4.1: Static document feature extraction.

What happens if we introduce a time relation between $\phi(D_i)$? How can we do that?

Before trying to answer these questions, it is interesting to understand when it is necessary. Usually, when we work with document datasets, we have a set of documents (papers, web pages, etc.) that do not grow or, if this happens, there is not a fixed time step. The classic tasks of document classification and clustering, focus on global results studying what happens on the whole dataset. These static problems have been attacked with acceptable results using the vector space representation, as described in chapter 2.1.

On the other hand, there are some cases where the vector space representation is not able to capture the variability of data, because $\phi(D(t_k))$ depends on data at time t_k and t_{k-1}. If a dataset grows regularly, i.e. a news web site is monitored each day, and we are interested to study how it grows, or how the results, obtained with the classic data mining algorithms, change, we have to introduce the variable "*time*" inside our document representation, see figure 4.2.

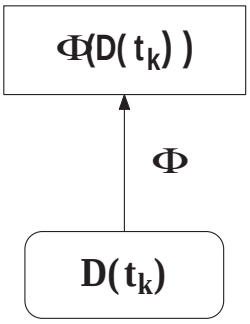

Figure 4.2: Time depend feature representation.

The features at time t_k, $\phi(D_{t_k})$, can be modeled considering at the same time two factors:

1. *Number of past instants of time*

2. *Properties of the past instants of time that can be used*

The *number of past instants of time* defines the temporal relation between the objects (ϑ) at time t and the data at the previous instants. At time t_k the features can depend also on objects at time t_{k-1}, in this case we have

a Markov Model, figure 4.3, or on a set of objects with longer temporally dependence, that we call *background*, figure 4.4.

Formally, we can write $\phi : [\chi, \Omega(T)] \rightarrow [\Im, T]$, where $\Omega(T)$ represents all the possible information that arrives from the previous instants of time and T is the set of the previous instants of time. For short dependence, $T = 1$, we are interested to find the quick fluctuations of features in the short period, and we obtain

$$\phi(D(t_k), \vartheta(t_{k-1})) \in \Im_{t_k}$$

Otherwise for long time dependence, $T > 1$, we compare features with the background, and we want to find only the features that have some relation with the long past period. We have

$$\phi(D(t_k), \vartheta(t_{k-1}, \ldots t_{k-j})) \in \Im_{t_k} \quad \text{with } j = 2 \ldots k \tag{4.1}$$

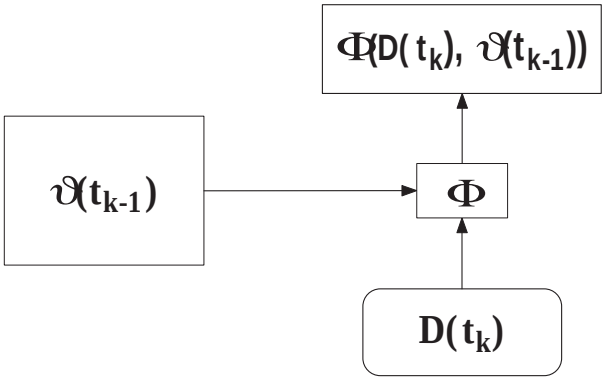

Figure 4.3: Markov Model representation.

If point 1 defines the temporal relation with the past, point 2 defines which are the objects involved in the relation. Normally, features are related with features at the previous times, but they can also be related with entire documents or a sampling of them.

In general, the two factors define four different models:

- *dependence on features at time* t_{k-1}: it is like a classic Markov Model,

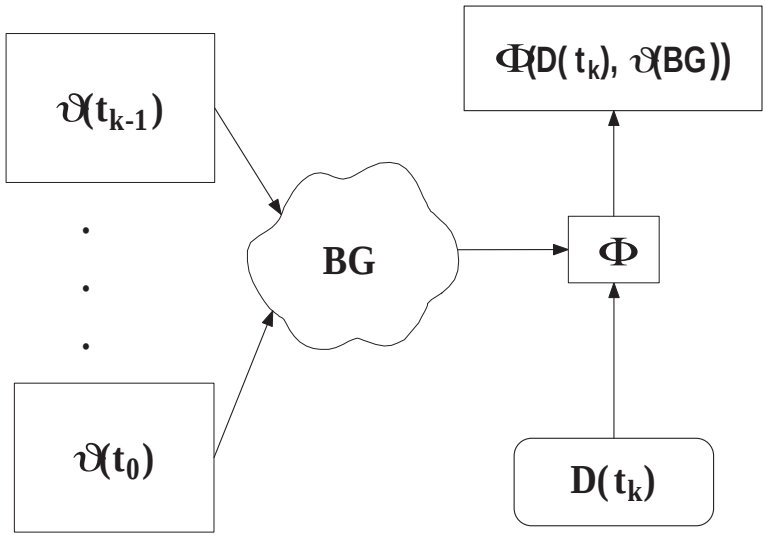

Figure 4.4: Background representation.

where features at time t_k depend only on features at time t_{k-1} and documents at time t_k. $\Im(t_k) \Leftarrow (D(t_k), \phi(t_{k-1}))$. The scheme is shown in figure 4.5

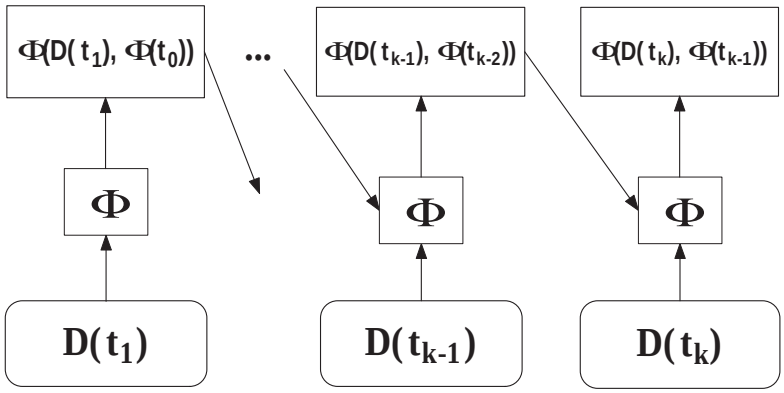

Figure 4.5: Features at time t_k depend on features at time t_{k-1} and documents at time t_k.

- *dependence on features at time* $t_{k-1} \ldots t_{k-j}$ *with* $j = 2, \ldots k$: in this case, all the features at the previous instants of time are grouped inside the *background*. It can be modeled as a set of features, where the

background's dimension depends on the rules that have been used to create it. At time t_k, the features are selected using also the information inside the background: $\phi(t_k) \Leftarrow (D(t_k), \phi(BG))$. See figure 4.6.

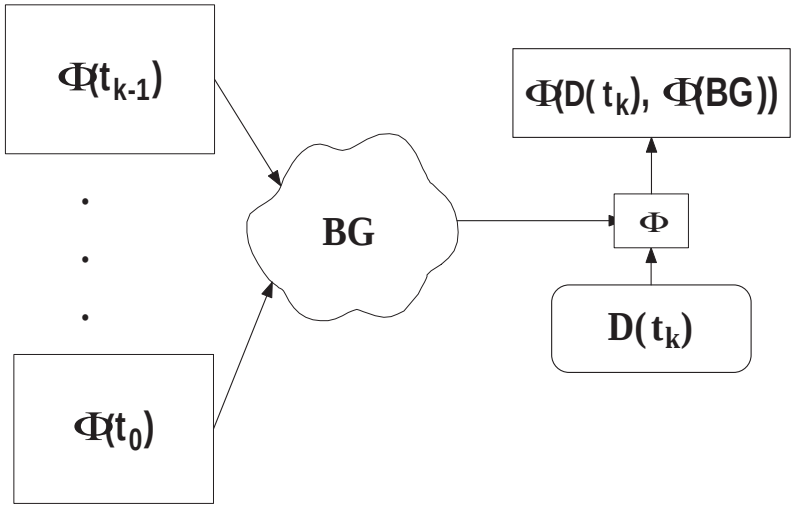

Figure 4.6: Features at time t depend on the features at time $t_{k-1} \dots t_{k-j}$ with $j = 2, \dots k$ and documents at time t_k.

- *dependence on documents at time* t_{k-1}: the dependence is not with the features at previous time, but with the entire documents, figure 4.7. This model could be interesting when we have few documents per instants of time. We have that $\Im(t_k) \Leftarrow (D(t_k), D(t_{k-1}))$
- *dependence on documents at time* $t_{k-1} \dots t_{k-j}$ *with* $j = 2, \dots k$: all the documents that have been seen till time t_{k-1} in the dataset influence the selection of the features at time t_k, see figure 4.8. The background contains indistinctly all the words, including the less informative ones. This is the main difference from point two, where the background is a selection of the most informative features. $\Im(t_k) \Leftarrow (D(t_k), D(BG))$.

These are just some methods that can be used to relate the features at different instants of time. In this section, we referred to a feature without analyzing what the features are. In the next sections, we explain the features

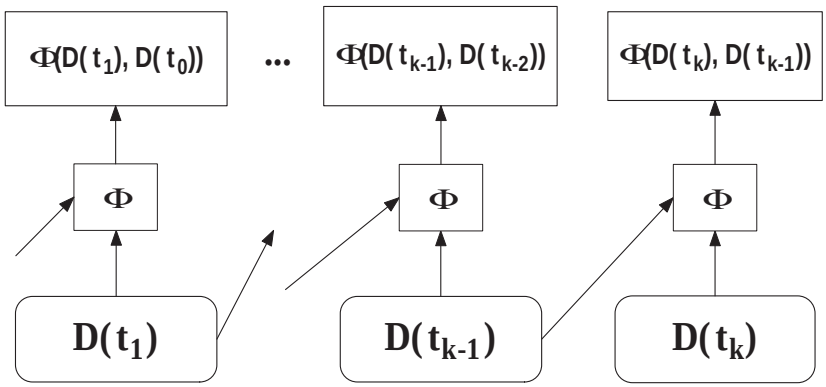

Figure 4.7: Features at time t_k depend on the documents at time t_k and t_{k-1}.

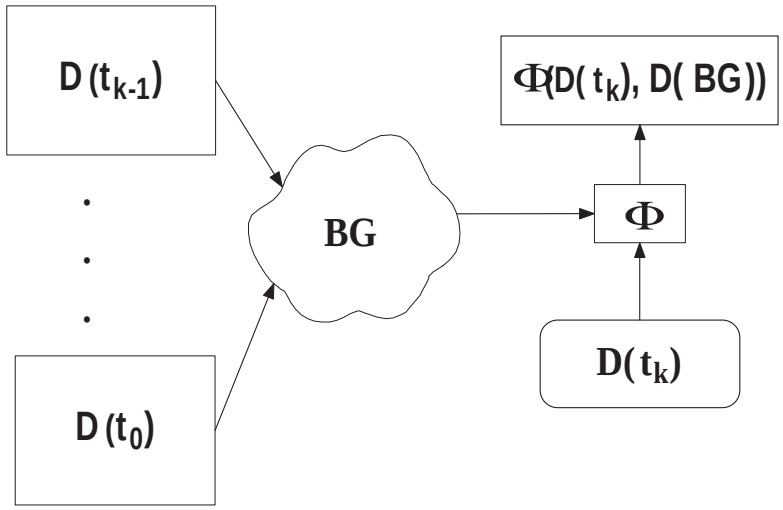

Figure 4.8: Features at time t_k depend on the documents at time t_k and $t_{k-1} \dots t_{k-j}$ with $j = 2, \dots k$.

representations, and we show how these influence the *textual time series* and their studies.

4.3 Key-Phrases Representation

Our approach can be considered a hybrid method. In fact, we use a static technique to extract the single features, and then we aggregate them into a time series creating connections. We suppose that the time series are our general features which represent temporal relations between data. We will see in the next sections, that there are different ways to create a time series which can mapped to the previous typology.

We use a *key-phrase* as an atomic feature. It is a n-gram of words, which are adjacent one to each other, an example is "Pope John Paul II". N-grams are useful when there are long sequences of symbols. In this work, we use it with words, in fact our set of symbols is composed by all the possible words of the English language. Given a document, it is split into a sequence of words. A unique number is associated to each word, such that the document becomes a sequence of numbers.

The algorithm that we use to extract key-phrases is based on a "sliding window". This approach gives us all the possible key-phrases in linear time. A hash table is used to count the occurrences of each key-phrase in the whole dataset every day. At the end, when all documents have been processed, all the key-phrases are sorted by their frequency. The top ten entries are our daily key-phrases.

Some key-phrases have not a particularly meaningful name despite the fact that correlated with a topic, but this property depends on the content of the news.

4.3.1 Key-Phrase Selection

While analyzing the key-phrases every day, we have seen that the Key-Phrases representation causes an imperfection. In fact, it is possible that

in different days the algorithm finds two or more key-phrases which are one subset of the other. A clear example are the key-phrases "Pope John Paul II", "Pope John Paul" and "John Paul II". In this case, we can see how the second and the third are subset of the first, and the third is also subset of the second. The reason are not due to the n-gram finder algorithm seen in the previous section which works correctly, but to the nature of data. Every day data change, and it is possible that one key-phrase is not present in its complete form, or it has a lower frequency value than another which is its subset.

To solve this problem, we developed a Key-Content selection algorithm, see schema "Algorithm 1", which is able to cluster key phrases in a syntactic way using the Bag-of-Words kernel [72].

We present briefly how it works. The Bag-of-Words kernel applied on all the key-phrases creates the kernel matrix K, where each entry contains a rank of similarity between keys-phrases. In this application, the rank of similarity is the number of words that two key-phrases share.

A vector *Lenght* is computed, and it contains the number of words of each key-phrase. For each column i of K, we compute the difference between i and *Length*. If the resultant vector has some entries j equal to 0, it means that the jth key-phrase is a subset of i. At the end of this process, we construct a new matrix S that has a set of key-phrases for each row i which are the subset's key-phrases. In S, we do not have a unique representation of this set, but all the possible combinations. To solve this problem, we apply the second part of the algorithm. In fact, we remove from S all the rows that contain just one element (each key-phrase is subset of itself) then we sort the new matrix moving the set with more elements at the top obtaining a new matrix $S-Reduced-Sort$. For each set t of $S-Reduced-Sort$, we compute the intersection with all the other sets. If for some set q, the intersection is equal to the cardinality of q, it means that the set q is included in t, and it can be removed. At the end we obtain a matrix where each row is a set of key-phrases which are at least one subset of another.

Algorithm 1: SubSet=Key Content Selection

STATE 1. *Compute the Bag of Words kernel on all the key phrases (kernel matrix K)*

STATE 2. *Compute the number of words for each key-phrase (vector Length)*

STATE 3. *For each column i of K:*

- *Compute the difference with the vector Length (vector Vdifference)*
  - *if some entry j of the vector Vdiffernce is equal to 0, key-phrase j is a subset of key-phrase i*
- *A matrix S is created and each row w contains the index of all the key-phrases subset of w*

STATE 4. *Clear the matrix S removing all the rows which contain only one element (matrix S-Reduced)*

- *Remove all the key-phrases which are not subset of other key-phrases*

STATE 5. *Sort the matrix S-Reduced by rows in descending order on the number of key-phrases present for row (matrix S-Reduced-Sort)*

- *Sets with more key-phrases are at the first row positions*

STATE 6. *For each set of S-Reduced-Sort:*

- *Compute the intersection with all the other sets (vector val_ intersection)*
  - *If some entry q of val_ intersection is equal to the cardinality of the qth sets, it is removed (matrix SubSet)*

This algorithm gives good results. In table 4.2 and 4.3, we can see some interesting clusters. Each subset is labeled by the longest string. We suppose that the longest string has more meaning than all the other key-phrases of the same set, and it helps to understand better the topic. Notice, that we apply an algorithm that uses the syntactic proprieties of each key-phrase, but each set gives a kind of semantical cluster. This is evident, because all key-phrases inside a set are contained at least one time in each other.

In the experiments described in section 4.6, this algorithm allowed us to reduce the key-phrase space from 601 to 527 elements, improving the quality of features. When we will use the word "data" in the rest of the chapter we suppose that this algorithm has been applied.

SubSet 1	**SubSet 2**	**SubSet 3**
Pope John Paul II	the U S military	President George W Bush
Pope John Paul	the U S	President George
Pope John	U S military	President Bush
Paul II	U S	George W Bush
John Paul II	S military	George W
John Paul		

Table 4.2: Some interesting results obtained with Key-Content selection algorithm.

SubSet 4	**SubSet 5**	**SubSet 6**
St Peter s Square	Tony Blair	security forces
St Peter s	Prime Minister Tony Blair	Iraqi security forces
St Peter	Minister Tony Blair	Iraqi forces
Peter s Square		

Table 4.3: Some interesting results obtained with Key-Content selection algorithm.

4.4 Algorithm

In this section, we discuss all the algorithms exploited to analyze the time series. We present three approaches base on different techniques, which have as goal to filter data and to create a time line. They suggest some properties of the time line useful to detect change points. These algorithms try to reduce the computational cost using Dynamic Programming [84, 85] in the majority of cases.

All these approaches require a preprocessing of data and different representations of them. We will explain better these steps in section 4.5.

Minimum Change Points Detection

This algorithm is based on the classical approach of the "sliding window". It uses two windows of a fixed length which slide on the data and compute a similarity value at each time step.

Given the dataset, this technique builds a new time series. This method maps the ten key-phrases per day into a new time series which has one observation for time instant. We substitute each string with a unique numerical id such that the database becomes a long sequence of numbers and we define a variable, called *step*, which indicates the length of each window. The observation value is computed by the intersection between all the key-phrase ids that are inside the two windows at each time step. The observation is therefore equal to the number of key-phrases common to the right and left intervals. We move these windows on all the sequences obtained from the whole dataset.

For *step* equals to 10, 30 and 50, we obtain the plots figures 4.9, 4.10 and 4.11. In these plots, we can see how the *step* value affects the granularity of

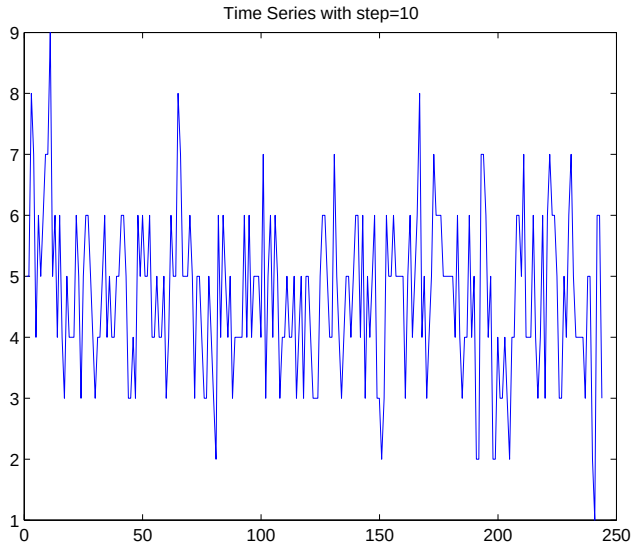

Figure 4.9: Time Series with step =10.

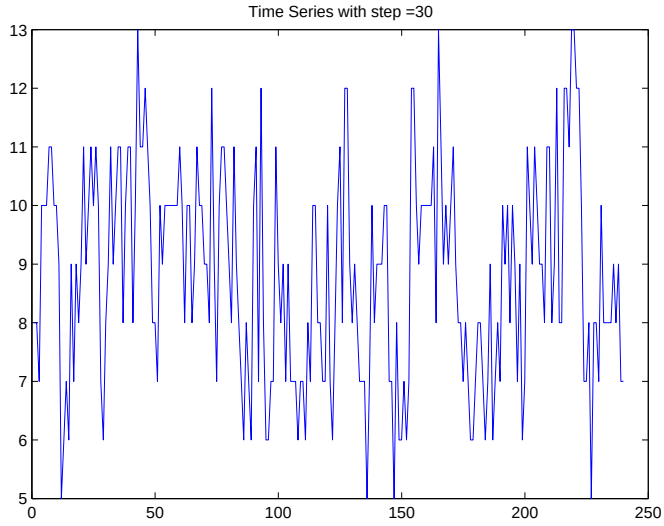

Figure 4.10: Time Series with step =30.

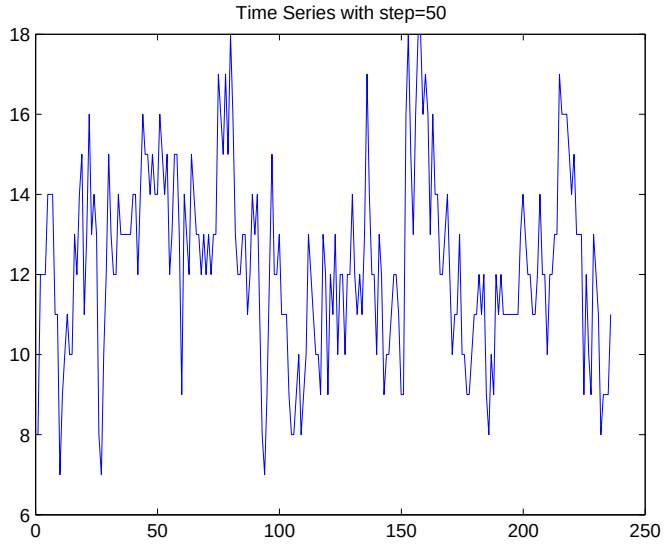

Figure 4.11: Time Series with step =50.

the series. In fact, in figure 4.9, the series has just few spikes that come out from a constant middle part. Increasing the *step* value, the series in figure 4.10 and 4.11 are more variable, losing the middle constant component. This is obvious because enlarging the window, the probability to have key-phrases in common increases.

After the unique time series is defined, we process it to detect the change points. We define a threshold, and we consider change points all the points where the value of the time series is lower than the threshold. The threshold is computed using a classical statistical approach. We define it as n times the standard deviations away from the mean of the time series. The standard deviation of a random variable is the square root of the variance described in the second chapter.

This approach is a particular example where the features of the numerical time series have been mapped into a new domain. A feature in the new domain at time t depends on the old features from $t - step$ to t and from $t + 1$ to $t + step$. Clearly we run this algorithm in batch mode and all the old and future terms have already been extracted.

Hidden Markov Model

Introduction. Hidden Markov Models (HMM) [86] have been the mainstay of the statistical modeling used in modern speech recognition systems. An HMM is more than a probabilistic function of a Markov process. An HMM is specified by five tuple (S, K, Π, A, B), where S and K are the set of states and the output alphabet, Π, A and B are the probabilities for the initial state, state transitions, and symbol emissions, respectively. A random variable X_t maps from the state names to corresponding integers. The symbol emitted at time t depends on both the state at time t and $t + 1$. An alternative formulation is a state emission HMM, where the symbol emitted at time t depend just on the state at time t. The interest in HMM comes from assuming that some set of data was generated by a HMM, and then being able to calculate the probabilities and the most likely underlying state sequence.

There are three important questions that we want to know about an HMM:

1. Given a model $\mu = (A, B, \Pi)$, how do we efficiently compute how likely a certain observation is, that is $P(O|\mu)$, where O is the output sequence?

2. Given the observation sequence O and a model μ, how do we choose a state sequence $(X_1, \ldots, X_{T+1})$ that best explains the observation?

3. Given an observation sequence O, and a space of possible models found by varying the model parameters $\mu = (A, B, \Pi)$, how do we find the model that best explain the observed data?

Normally, we do not know the parameters and we have to estimate them from data (point 3). The first point can be used to decide which is the best model from a set of given HMMs to explain an observation. The second point lets us guess what path was probably followed through the Markov chain, and this hidden path can be used for different applications (e.g. to segment the sequence).

To reach our goal, we have to answer to the second and third questions. In fact, we have some observations obtained from the Event Detection algorithm, and we want to estimate A and B using these observations. When we know A and B, we want to apply this model to data and find the best state sequence.

Let us first consider point 3. Given an observation sequence, we want to find the values of the model parameters $\mu = (A, B, \Pi)$, which best explain what we observed. Using Maximum Likelihood Estimation, we want to find the values that maximize $P(O|\mu)$:

$$\arg\max_{\mu} P(O_{training}|\mu).$$

There is no known analytic method to choose μ to maximize $P(O|\mu)$. But we can locally maximize it by an iterative hill-climbing algorithm, the Baum-Welch algorithm, which is a special case of the EM algorithm, that we have

seen in section 2.2.1. We do not know what the model is, but we can obtain the probability of the observation sequence using some model. We can see which state transitions and symbol emissions were likely used most. By increasing the probability of those, we can choose a revised model which gives a higher probability to the observation sequence. This maximization process is referred to as "training the model" and is performed on training data.

We define $p_t(i,j)$, $1 \leq t \leq T$, $1 \leq i,j \leq N$ as:

$$\begin{aligned} p_t(i,j) &= P(X_t = i, X_{t+1} = j | O, \mu) = \\ &= \frac{P(X_t = i, X_{t+1} = j, O|\mu)}{P(O|\mu)}. \end{aligned}$$

This is the probability of traversing a certain arc at time t given the observation sequence O. We define $\gamma_i(t) = \sum_{j=1}^{N} p_t(i,j)$.

Now if we sum over the time index, this gives us the expectations:
$\sum_{t=1}^{T} \gamma_i(t)$ = expected number of transitions from state i in O
$\sum_{t=1}^{T} p_t(i,j)$ = expected number of transitions from state i to j in O
So we begin with some model μ. We then run O through the current model to estimate the expectations of each model variable. We then update the model parameters to maximize the value of the most likely path. We then repeat this process hoping to converge to optimal values for the model parameters μ.

The reestimation formulas are:

- $\hat{\Pi}_i$= expected frequency in state i at time $t = 1$
- $\hat{a}_{i,j} = \frac{\sum_{t=1}^{T} p_t(i,j)}{\sum_{t=1}^{T} \gamma_i(t)}$
- $\hat{b}_{i,j} = \frac{\sum_{t:O_t=k, 1\leq t\leq T} p_t(i,j)}{\sum_{t=1}^{T} p_t(i,j)}$

Thus, from $\mu = (A, B, \Pi)$, we derive $\hat{\mu} = (\hat{A}, \hat{B}, \hat{\Pi})$. As proved by Baum, we have that:

$$P(O|\hat{\mu}) \geq P(O|\mu).$$

This is a general property of the EM algorithm. Iterating through a number of cycles of parameter reestimation the model will explain better the given observation. The training process is stopped when no significant improvement is obtained in two consecutive cycles. This process of parameter reestimation does not guarantee that we will find the best model, because the reestimation process may get stuck in a local maximum of the likelihood function. In fact, the likelihood function is a complex nonlinear surface and there may be many local maxima. Nevertheless, the Baum-Welch reestimation is usually effective for HHMs.

Now we attack point 2. It was phrased somewhat vaguely as "finding the state sequence that best explains the observations." That is because there is more than one way to think how to do this. We present the Viterbi algorithm. Commonly we want to find the most likely complete path that is:

$$\arg\max_X P(X|O,\mu).$$

To do this, it is sufficient to maximize for a fixed O:

$$\arg\max_X P(X,O|\mu).$$

An efficient trellis algorithm for computing this path is the Viterbi algorithm. Define

$$\delta_j(t) = \max_{X_1,\ldots,X_{T+1}} P(X_1,\ldots,X_{T+1},O_1,\ldots,O_{t-1},X_T=j|\mu)$$

This variable stores the probability of the most probable path that leads to that node for each point in the trellis. The corresponding variable $\Psi_j(t)$ then records the node of the incoming arc that led to this most probable path. Using dynamic programming, we calculate the most probable path through the whole trellis as follows:

- Initialization
  $\delta_j(t) = \pi_j,\ 1 \leq j \leq N$

- Induction
  $\delta_j(t+1) = \max_{1 \le i \le N} \delta_i(t) a_{ij} b_{ijO}$, $1 \le j \le N$
  Store backtrace
  $\Psi_j(t+1) = \arg\max_{1 \le i \le N} \delta_i(t) a_{ij} b_{ijO}$, $1 \le j \le N$

- Termination and path readout. The most likely state sequence is extracted proceding backwards:
  $\hat{X}_{T+1} = \arg\max_{1 \le i \le N} \delta_i(t+1)$
  $\hat{X}_t = \Psi_{\hat{X}_{t+1}}(t+1)$
  $P(\hat{X}) = \max_{1 \le i \le N} \delta_i(t+1)$

Time Series Analysis using HHM. This approach is based on the idea that all sources are activated every day, but only few of them are able to emit a key-phrase. We suppose that each source can emit or not just one key-phrase. A single source is modeled by an HMM with two states, see figure 4.12. This HMM is defined in the following way:

- $S = \{s_0, s_1\}$
- $K = \{0, 1\}$
- $O = \{o_0, o_1\}$

where the state at time t is defines the fact that the emission of that symbol has been performed by the source or not, i.e. the hidden state at time t is the presence or absence of the key-phrases that day. This HMM is defined by E, the (2 by 2) emission matrix, and T, the (2 by 2) transition matrix. Every entry of E is an element of A, and the entries of T belong to B. Each HMM is trained on the same sequence of data, such that the matrix E and T are the same for each key-phrase. The training data are selected from the output of *Event finder* algorithm, see section 4.4.1. In the experiments, we chose the 3 key-phrases that have only one well defined event. These key-phrases are: "Pope John Paul II", "of Hurricane Katrina" and "of New Orleans", see

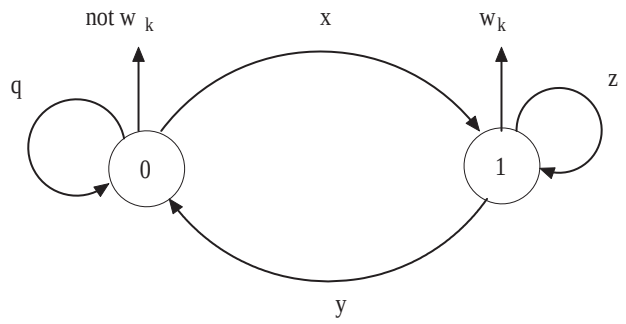

Figure 4.12: The HMM that models a source.

section 4.6. We want to use these key-phrases to train the HMM to recognize uniform and compact sequences. We do that because we assume that change points of this kind of sequences are more meaningful than change points in a sequence with a lot of changes.

Given the dataset, we compute a matrix M $(\#key - phrases \times \#days)$, where each entry is

$$m_{i,j} = \begin{cases} 1 & \text{if a key-phrase } i \text{ is present the day } j \\ 0 & otherwise \end{cases} \tag{4.2}$$

When the HMM has been trained, we run the HMM on the rows of the matrix M using the Viterbi algorithm. This algorithm gives us the state sequence and the value of total probability for each sequence. We define matrix V, where each row is a state sequence. We order V using the total probability obtained from the Viterbi algorithm. The top n rows of this new matrix are scanned and each transition $0 \rightarrow 1$ and $1 \rightarrow 0$ can be considered as change point. In the results section, we will see how this approach can be used as a filter of the key-phrases in order to obtain a good time line over which change points can be detected.

Is it possible to define a global HMM that works as a set of k single HMM? We can propose an Hidden Markov Model approach where all the single HMM work all together. The schema of this global HMM is made by the combination of all the sources. In figure 4.13, we merge only two

sources producing a quite complex model. Unfortunately, this scheme grows

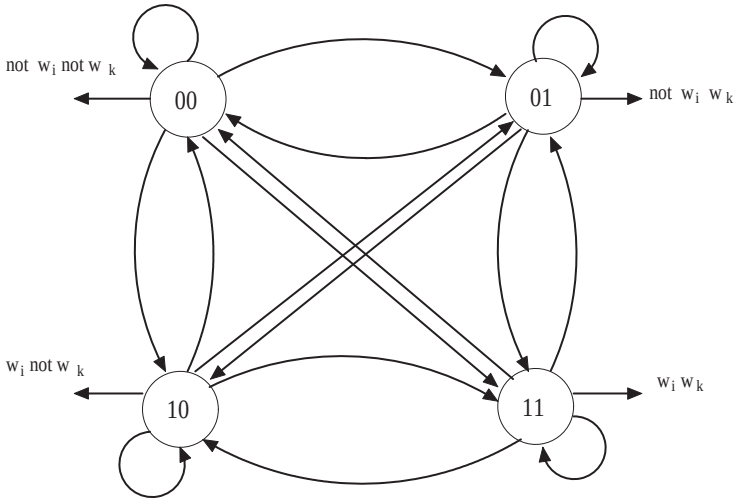

Figure 4.13: Markov chain of two sources.

exponentially in the number of states with respect to the number of sources, and the analysis of the model becomes statistically intractable, because we need too much data to be able to estimate all the entries of the transition and emission matrixes. This problem becomes tractable imposing the independence of the sources. It means that each source is a single HMM that does not communicate with the others. This assumption is strong, but it is exactly what we have done.

Now we reformulate the whole problem using this constrain. We define for each HMM/source:
s_i^t = the source i has emitted at the time t
$\bar{s}_i^t$ = the source i has not emitted at the time t
and
o_i^t = the key-phrases i is present at the time t
$\bar{o}_i^t$ = the key-phrases i is not present at the time t
This HMM has trained on data such that the emission matrix E_i and the transition matrix T_i are strongly diagonal. We define two vectors (1 by 2)

$S_i^k = [s_i^k \bar{s}_i^k]$ and $O_i^k = [o_i^k \bar{o}_i^k]$, and using this formalism, we can write:

$$S_i^{t+1} = S_i^t T_i \tag{4.3}$$

and

$$O_i^{t+1} = S_i^t E_i \tag{4.4}$$

Using these results, we can introduce the general model that describes the whole system with all the sources. When k sources are considered, we have $\mathbf{S}^t = [s_0^t \bar{s}_0^t, s_1^t \bar{s}_1^t, \ldots, s_k^t \bar{s}_k^t]$ and $\mathbf{O}^t = [o_0^t \bar{o}_0^t, o_1^t \bar{o}_1^t, \ldots, o_k^t \bar{o}_k^t]$. The global emission matrix $\mathbf{E}$ is a matrix:

$$\mathbf{E} = \begin{pmatrix} E_0 & 0 & \ldots & 0 \\ 0 & E_1 & \ldots & \vdots \\ \vdots & & \ddots & \vdots \\ 0 & \ldots & \ldots & E_k \end{pmatrix}$$

which has on the diagonal all the singular emission matrices of each key-phrases. In the same way, we construct the global transition matrix $\mathbf{T}$:

$$\mathbf{T} = \begin{pmatrix} T_0 & 0 & \ldots & 0 \\ 0 & T_1 & \ldots & \vdots \\ \vdots & & \ddots & \vdots \\ 0 & \ldots & \ldots & T_k \end{pmatrix}$$

which has on the diagonal all the singular transition matrices of each key-phrases. These matrices have 2^k rows and columns, but only the elements on the diagonal are different from 0. This property is due to the independency of the sources. The equations 4.3 and 4.4 become for the global HMM:

$$\mathbf{S}^{t+1} = \mathbf{S}^t \mathbf{T}$$

and

$$\mathbf{O}^{t+1} = \mathbf{S}^t\mathbf{E}.$$

In that way, we have a general HMM, which is able to manage k independent sources of key-phrases. Ideally, if we use k HMMs separately or this global one, the results are the same.

4.4.1 Event Detection

The goal of this algorithm is to determine particular events in our time series and to build a good time line. A standard assumption is that the raw data collected is somehow processed to generate a sequence of events, which is then mined for interesting episodes. In general, there is not a strict definition of an episode, that intuitively is a pattern of events occurring in some order, and close enough to each other in time. The event detection problem is highly correlated with the change point analysis, in fact it is solved by identify the time point at which the event starts and finishes.

We define an event as a key-phrase or a set of them which are present for a continuous sequences of instants of time (period). Our approach uses a *random walk* to scan each time series obtained for a key-phrase and to discover the trend on each of them. The key-phrases which have a particular shape and persist for q or more instants of time are considered interesting events and their change points are assumed to be possible changes of the source.

A random walk is a formalization of the intuitive idea of taking successive steps, each in a random direction. Random walk can also be looked at as a Markov chain whose state space is given by the integers $i = 0, \pm 1, \pm 2, \ldots$, for some number $0 < p < 1$, $P_{i,i+1} = p = 1 - P_{i,i-1}$. We can call it a random walk because we may think of it as being a model for an individual walking on a straight line that at each point of time either takes one step to the right with probability p or one step to the left with probability $1 - p$. A random walk is a simple stochastic process, see 1.2.3.

The simplest random walk is a path constructed according to the following

rules:

- There is a starting point.
- The distance from one point in the path to the next is a constant.
- The direction from one point in the path to the next is chosen at random, and no direction is more probable than another.

The average straight-line distance between the start and ending points of a random walk of n steps is of the order of $\sqrt{n}$, or more precisely, it asymptotes to $\sqrt{2n/\pi} \approx 0.8\sqrt{n}$. If "average" is understood in the sense of root-mean-square, then the average distance after n steps is higher, but still less than $\sqrt{n}$ times the step length. Suppose we draw a line some distance from the origin of the walk. How many times will the random walk cross the line? The following, perhaps surprising, theorem is the answer: for any random walk, every point in the domain will be crossed an infinite number of times almost surely.

For each row of M a dynamic programming algorithm which implements a random walk is run. We allow the random walk to advance in either direction by a fix step, -1 when $m_{i,j} = 0$, $+1$ when $m_{i,j} = 1$. We impose that when it finds a new 0, and the sum from the beginning to now is equal to 0, the sum can not go below 0. It means that we limit the random walk on the left. This assumption makes the model sensitive to each positive step. In fact, without the left limit the random walk can go deeply into negative values, such that it needs a lot of positive steps to come out, otherwise in our case, each positive step is detected. For each key-phrase, we store the maximum value that the random walk reaches during its walk and the position of this maximum value inside the time series, the initial and the final position of the event where the maximum was placed. The algorithm allows to set some parameters as the length of the event and a confidence interval around the length. Some key-phrases not only correspond to one main event, but they appear more than one time. For each key-phrase we consider only the main event. We define the length of the event as the number of steps from the beginning to the maximum value of the main event.

When this set of key-phrases have been chosen, we plot them in a time line, and for each of them we can consider a change point as each transition $0 \to 1$ and $1 \to 0$.

4.5 Dataset

In this work, the dataset has a crucial role, especially its length; in particular, to obtain a reasonable result we need a long time series. At the moment, we have an ten month long time series, which starts on March 28th, 2005 and ends on January 31th, 2006.

How did we construct it? Every day, we checked some of the most important news agency, see table 4.4. For each of them, we crawled the world news page extracting the title, the content, the source, date and time. All

News Sources
UsaToday
Cnn
Detroit News
International Herald Tribune
Fox News
Al Jazeera

Table 4.4: Sources of World News.

this information was stored in a database. Using the algorithms described in section 4.3 and 4.3.1, we retrieved the key-phrases and performed the feature selection. For each key-phrase, we have also its frequency: the number of times that a key-phrase appears in a given day. Every day we keep the top ten key-phrases and we obtain a time series, where each observation is made of ten strings.

In the previous sections, we have seen the different ways to pass from this time series to a numerical time series, but we prefer to spend some more time to discuss this mapping. The database can be seen in two orthogonal ways. By day, where there is a vector for each day with length equal to the

number of unique key-phrases and only ten entries are different from 0, or by key-phrases, where there is a vector for each key-phrase having length equal to the number of days. In the worst case the key-phrase is present every days. In both cases, the vector can have two different representations. Binary, where each entry indicates if the key-phrase is present (value 1) or not (value 0), or frequency, where each entry is the key-phrase frequency. Thus, the whole dataset is represented by the matrix key-phrases by days. We call M the binary version and KP the frequency version. In figure 4.14, we plot the transpose of the binary matrix M. It is interesting to note that every day new key-phrases appear. At the moment our set of key-phrases is not huge, but this characteristic is in general valid for each vocabulary. This is due to the fact that each language evolves borrowing terms from other languages or modifying the meaning or the typing of some terms. The dates

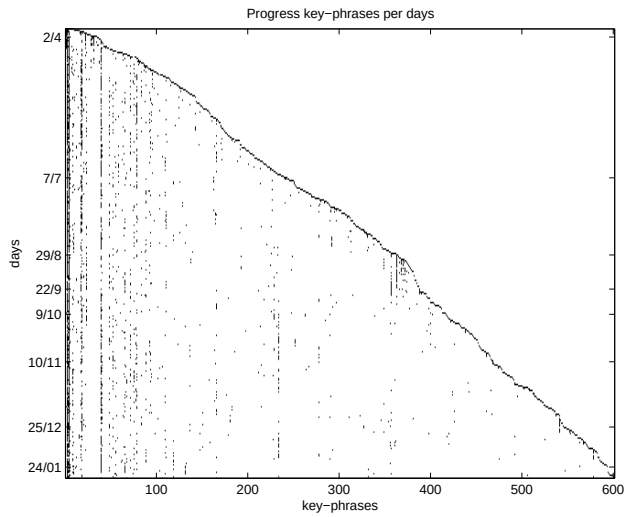

Figure 4.14: Progress of key-phrases per days.

on the vertical axes represent days where some events have occurred. In particular, 2nd April correspond to the death of the Pope John Paul II; 7th July correspond to the London bombing attacks; on August 29th Hurricane Katrina arrived on the Us Gulf coasts, and on September 22nd Hurricane Rita was close to the US coasts; on October 9th the bird flu arrives to Europe,

on November 19th the Jordanian bombing attack happened, on December 25th Christmas celebration and on January 24th Hamas wins the Palestinian legislative election. For more details about the real time line events see the appendix.

The minimum change point detection algorithm uses a different representation of the whole database. It does not use a two dimensional representation (matrix), as we have seen before, but a flat representation. We map each key-phrase to an univocal numerical id. For each day, all the ids are inserted sequentially in a vector $MVec$ transforming the sequence of key-phrases in a sequence of numbers.

In section 4.3.1, we have seen how it is possible to reduce the dimension of the dataset clustering the key-phrases. When the subsets have been defined, there is the problem of transferring all the information from all the key-phrases in the same subset to the corresponding longest string. It depends on the type of representation of the features. If the vectors are in the binary form, the longest string gets the information from the other vectors using the logical operator "or". Otherwise if the vectors use the frequencies, the "sum" operation is exploited. In case of the minimum change point detection algorithm, the id of the key-phrases which belong to the same cluster are substituted by the id of the longest string. This operation does not alter the final results.

4.6 Results

In this section, we present the results of our experiments for the three approaches explained above.

The main goal is to automatically extract the key-phrases which represent the most interesting and consistent events in the time series. We plot these key-phrases to a time line where it is possible to analyze the change points. In figure 4.16, we show the scheme of the processing.

We start our discussion from the matrix M. The algorithm that retrieves the key-phrases has been run, and M has been built. By applying the al-

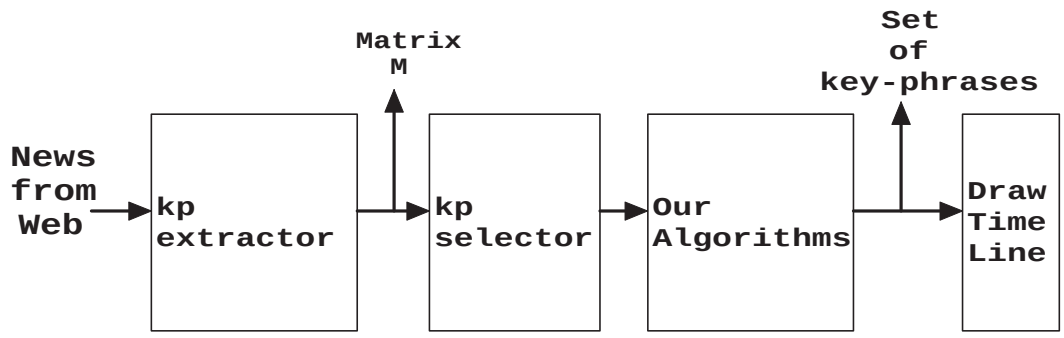

Figure 4.15: Scheme of the processing.

gorithm seen in section 4.3.1, we reduce the key-phrase space obtaining the matrix M'. In figure 4.16, the matrix M' has been plotted. In this figure we can see some interesting relations. In fact, there are some key-phrases that occur a lot of times and are long black lines. Whereas some others are present few times and are represented as black points. We are looking for key-phrases which are associated with small black lines. In fact, this means that they occur for a short continuous period: an event. Now we can start to analyze the results obtain with our methods.

We run the *Minimum Change Points Detection* algorithm with step equal to 30 and threshold, computed using $n = 3/2$, equals to 6.074. The algorithm finds, see figure 4.17, 31 change points. In few cases, this algorithm places the change points close to the real event, for example the first one is near to the "Pope John Paul II death" (April 2nd) , or close to the "Arrive of the Hurricane Katrina to the US coast" 29th August. In the other cases (see Appendix for more detail about the events) it is not able to spot the events. Besides, when it discovers one change point, it finds a sequence of them. This problem is due to the construction of the sliding window. When an important event arrives, the first sliding window starts to find new terms reducing the intersection between the two windows. When all the new terms are deeply inside the first window, the intersection is minimal, and it yields small value that will be discovered as a change point. This situation continues until the event's terms enter into the second sliding windows.

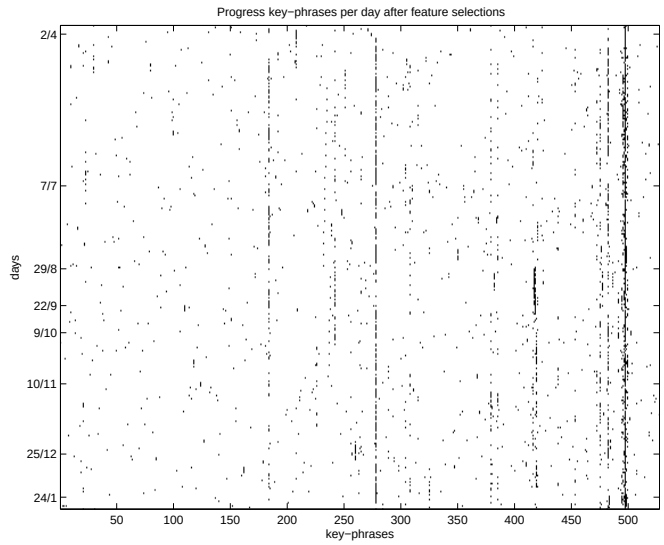

Figure 4.16: Progress of key-phrases per days after key-phrases selection. In this figure we lost the nice curve seen in figure 4.14, because here we order the key-phrases in alphabetical ordered.

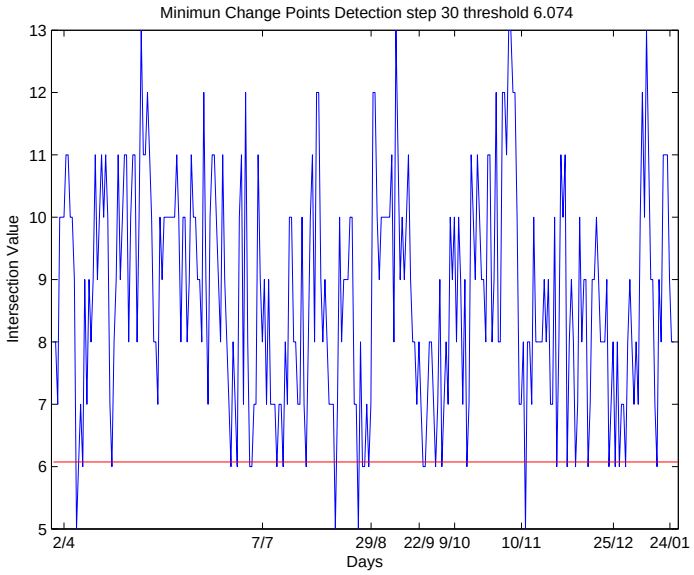

Figure 4.17: Minimum Change Points Detection Algorithm applied using step=30 and threshold with k=3/2.

In general, this algorithm does not give us the key-phrases which produced the event. In fact, assuming that the change points are in the right positions, the key-phrases could be deduced after the change points have been discovered using some measure. In the next chapter we propose a measure to find the key-phrases when the change points are correctly detected.

In the next two algorithms, we change philosophy. In fact, if in the minimum change points detection approach we would like to retrieve the key-phrases when the change points have been found, here we extract the most interesting key-phrases and then we show the change points.

HMM are a good tool to make a selection of the key-phrases. We train one two-state HMM with the same sentences. Then we run it on all the key-phrases. The training sentences have been chosen by the event finder algorithm. In fact, how we will see in the following of this section, the algorithm is able to discover key-phrases which are associated to a well defined continuous period where they are present. These sentences are "Pope John Paul II", "of Hurricane Katrina" and "of New Orleans". How we can see from the plot in figure 4.18, where on the horizontal axis we have time and on the vertical the key-phrases, the sentences have a block where they appear continuatively for a sequence of days. Our idea is to train the HMM to model key-phrases which have state sequences like these. The training sentences define a model given by the following transition and emission matrixes:

$$T = \begin{pmatrix} 0.99613638339342 & 0.00386361660658 \\ 0.04078734322118 & 0.95921265677882 \end{pmatrix}$$

and

$$E = \begin{pmatrix} 0.98326422556156 & 0.01673577443844 \\ 0.15888703553946 & 0.84111296446054 \end{pmatrix}$$

Both these matrices are strongly diagonal, in particular the transition matrix. It means that, for the transition matrix, when the HMM is in a particular state, it continues to remain there. While, for the emission, when a symbol

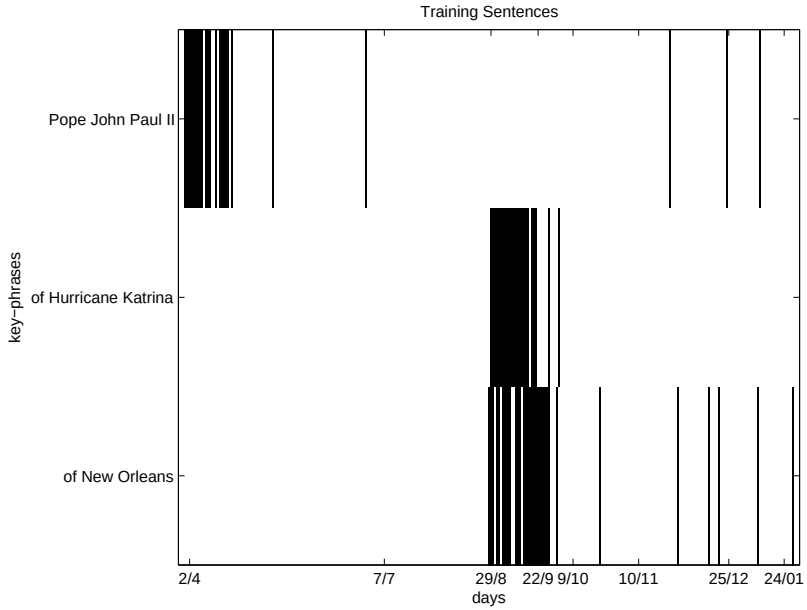

Figure 4.18: The sequences used to train the single two state Hidden Markov Model.

has been observed, it emits quite surely the same symbol. Thinking about the interpretation that we have given to the HMM, it means that when a key-phrase has been observed, the corresponding source has emitted quite surely. In the same way, if the source has emitted, it is likely it will be in the same state the next instant of time.

We ran this HMM on all the key-phrases, and then we ranked the key-phrases using the total probability obtained from the Viterbi algorithm. The top 30 key-phrases are shown in figure 4.19. This set of key-phrases contains some interesting terms, that have the required characteristics; for example "of bird flu", "the West Bank", "bin Laden", "Social Security benefits", "the Social Democrats" which occurs in a very close period of "Schr der"[1] before and after the Germanic federal election, and "human right".

In general, this model has rejected sequences with a lot of spikes or se-

[1]The key-phrase "Schr der" is a clear mistake made by the kp extractor due to the fact that the name Schröder contains the UTF-8 symbol ö and it is not able to read so it substitutes it with space and considers "Schr der" as a key-phrases.

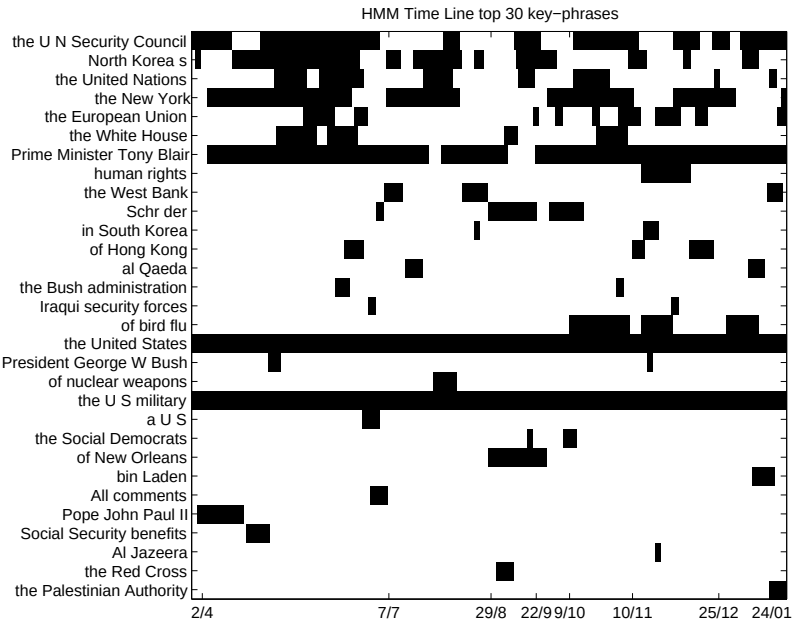

Figure 4.19: The top 30 key-phrases extracted by the two state HMM.

quences with a small period. Otherwise, since it was trained with sequences which emphasize continuous periods, it detects optimal state sequences also for key-phrases like "the United State", "the U S military" or "Prime Minister Tony Blair". These key-phrases have very long periods where the sources have emitted, but the length of the period does not agree with our idea of optimal sequence. Besides, this model is not able to detect sequences with one main block, but it accepts sequences with more than one large period, like "the U N Security Council" or "the United Nations" . These problems will be solved by the following algorithm.

It is clear that this approach is strongly dependent from the training sequences and the problems can be imputed to a not completely correct selection of these examples. To find the right training sequence is not an easy problem, and it requires a lot of skills and time.

Note that in these problems, we analyzed the change points. We chose an interval of time where we know exactly the most important events, and we impose a threshold on the minimum number of change points in that interval. We want that there is at least one change point. In figure 4.20, we have chosen an interval which starts on March 28th and finishes on April 30th. The main events in this period have been the Pope's death, and the election of the new Pope. If we do not consider the key-phrases, like "Prime Minister Tony Blair" and "the United States", that can be considered a mistake of this approach, the other key-phrases are correlated with the Pope events that is a topic that does not arrive abruptly, but it grows slowly. It is interesting to note days like April 1st, or April where some key-phrases appear for the first time and other disappear. An example are the terms "Terri Schiavo", "feeding tube", "The pope" and "St Peter s Square". Where "Terry Schiavo" and "feeding tube" disappear and "The pope" and "St Peter s Square" appear, it can be the right position where to place a change points. These is only a clear example and other can be discovered. When the most interesting key-phrases have been selected for each period, the choice of the day where place the change points is not a hard task and also easy approach can be efficient like in the previous examples. Summarizing, the change points can be placed, when the time line has been cleaned, using the following properties of the continuous

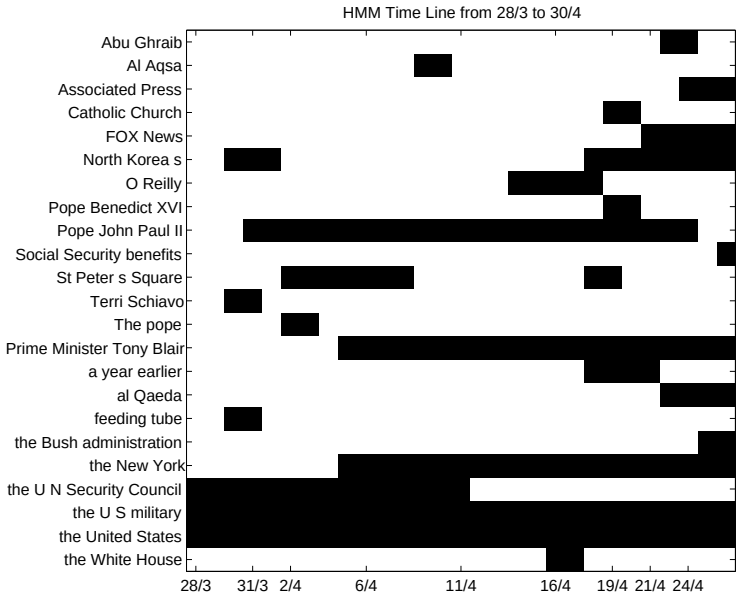

Figure 4.20: The top key-phrases selected from March 28th to April 30th.

blocks:

1. The number of overlaps, assuming that point that high values are possible changes

2. The number of incoming and ingoing blocks for each instant of time, assuming that points with high values are possible changes

3. Select the longest block and use both the properties at the previous points

These ideas will be more understandable with the next algorithm, which will "clean" the time line.

Now we introduce the last algorithm, the *Event Finder*. This method scrolls a random walk on each row of the matrix M, and for each row defines the most compact and continuous segment of days where the key-phrases is present. For each period it stores the maximum value achieved, the day where the maximum is, and the beginning and the ending day. It allows

to define an upper and lower bound on the length of the period to filter only the key-phrases which meet this constrains. We consider the length of each period as the distance between the beginning and the day where the maximum has been placed.

In figures 4.21 and 4.22, we show the output of the algorithm on two particular key-phrases.

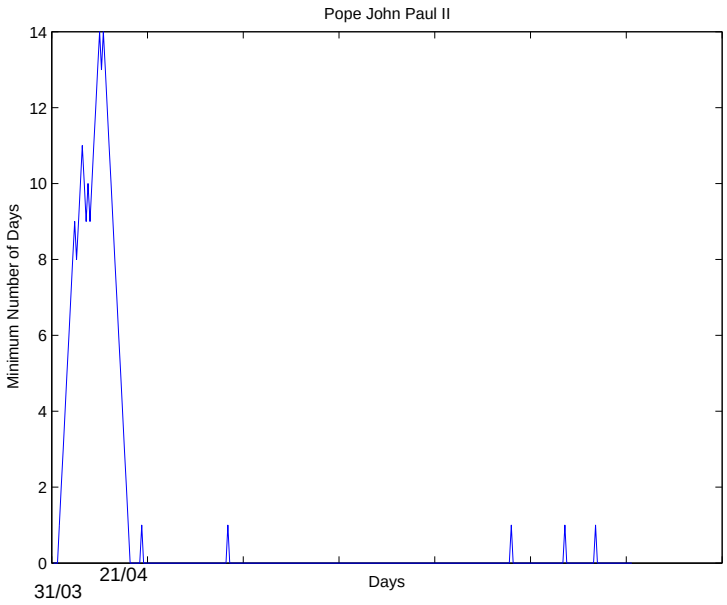

Figure 4.21: Output of the event detector algorithm for the key-phrases "Pope John Paul II".

On the horizontal axis we have the days and on the vertical the minimum number of days needed to achieve that particular point on the graph. These key-phrases have an important main event that has been detected perfectly by the algorithm. It is also possible to show more than one plot of key-phrases which belong to the same topic. In figure 4.23, we plot all the key-phrases that refer to the event Hurricane Katrina. This is very interesting, because the entrance of each key-phrase have a relation with the real event. In this figure, it can be seen clearly. We can notice that the key-phrase "of New Orleans" has come earlier than the entrance of "Hurricane Katrina". This was due

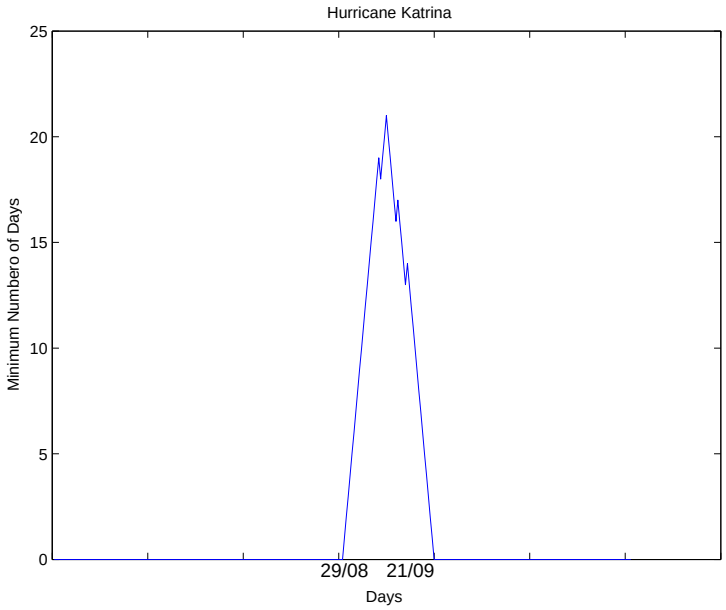

Figure 4.22: Output of the event detector algorithm for the key-phrases "Hurricane Katrina".

to the fact that New Orleans entered into the media before the hurricane became famous and its name was well known because at the beginning a "generic" hurricane could have hit the city. A lot of these patterns can be detected analyzing the overlap of the key-phrases.

The event detector was run on all the matrix searching events with length included between 3 and 23 days. We obtained 34 key-phrases that meet these restrictions and they are shown in figure 4.24. The first important difference from the other approach is that key-phrases like "the United State", "the U S military" or "Prime Minister Tony Blair" have disappeared. In fact those key-phrases have very long continuous periods that do not meet the bound. This algorithm differently from the HMM does not require any kind of training being dependent only from the data and the length. The set of key-phrases contains a lot of new events: the London bombing attack by the key-phrases "de Menezes", the Sharm-el-Sheik bombing attack by "Sharm el Sheik", the Jordanian bombing attack by "Islamic Jihad", Hurricane Rita

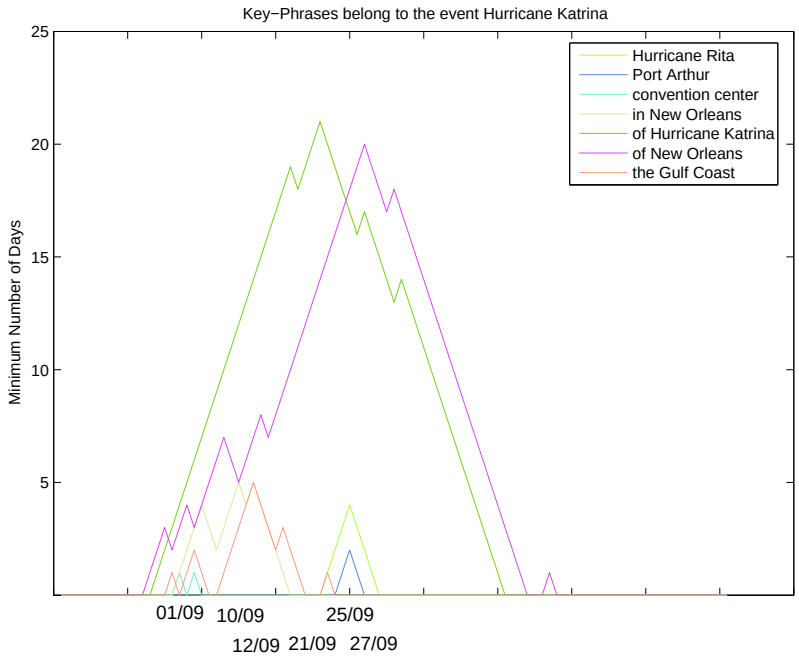

Figure 4.23: All the key-phrases which belong to the event Hurricane Katrina.

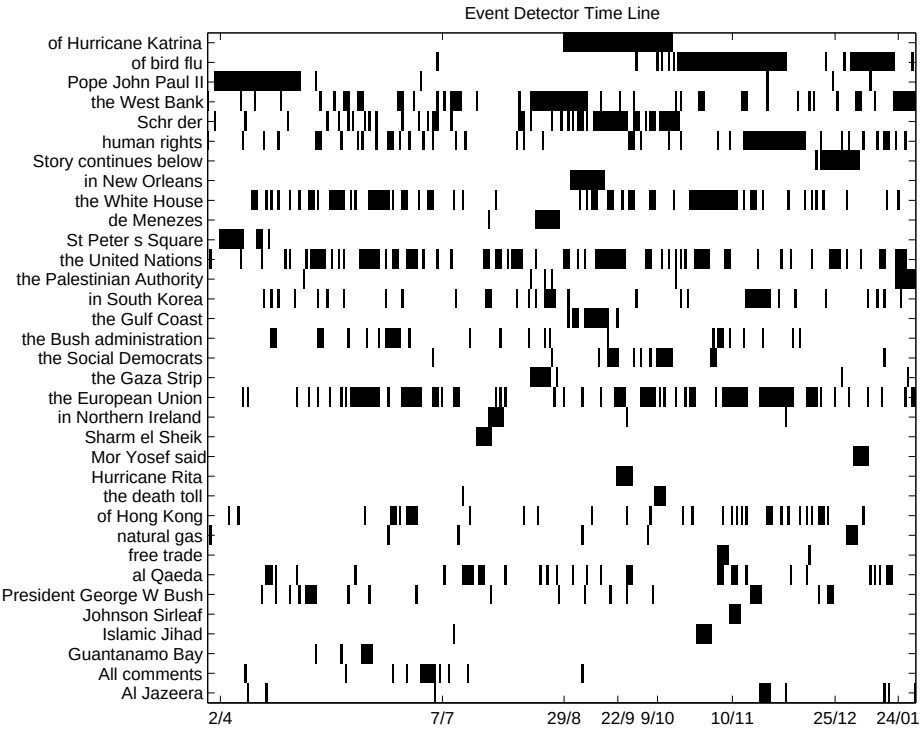

Figure 4.24: All the key-phrases with the length of their main event included between 3 and 23 days.

by "Hurricane Rita", the Pakistan earthquake by "the death toll", Liberian election by "Johnson Sirleaf", barrier seals off road into Gaza Strip by "the Gaza Strip", IRA dumps their army by "in Northern Ireland" and so on. These are the most important key-phrases which identify the most important events. This statement can be proved comparing our result with the real time line in the appendix. This result shows how this algorithm merges temporal and textual information. In fact, it takes sequences of news and gives us the main key-phrases obtained analyzing the time series. This events can be split using the temporal occurrences, focusing our attention only on a small period. An example has been presented in figure 4.27, giving interesting results.

The quality of the sequence for each key-phrases can be improved, because it is not really clear due to many spikes. The algorithm can solve this problem: for each key-phrase, it stores all the information of the main period, so, as we can see in figure 4.25, it is able to clean the sequence giving a more clear representation. Temporal clusters of key-phrases can be highlighted giving further information on the relation between terms. This new visualization is easily analyzable, and the change points can be detected using the simple method seen for the HMM algorithm. Clearly, some information has been lost, but the duality of the algorithm allows to choose the type of representation.

In figure 4.24 and 4.25, the main events have been plotted considering also the descend path of the random walk. This enlarges the real dimension of the event and imposes the presence of a key-phrase also where it is not present. In figure 4.26, we view all the main events showing only the period from the beginning to the maximin value achieved by the random walk. This model can cut key-phrases which have the maximum value at the beginning of the sequence and descend slowly.

Now we analyze the results in a short interval as we have done for the HMM algorithm. We use the same interval that starts on March 28th and finishes on April 30th. Each period has been represented using the real length, and the plot shows the key-phrase sequence in the clean version. Comparing figure 4.27 with figure 4.20, it is evident that the first plot is

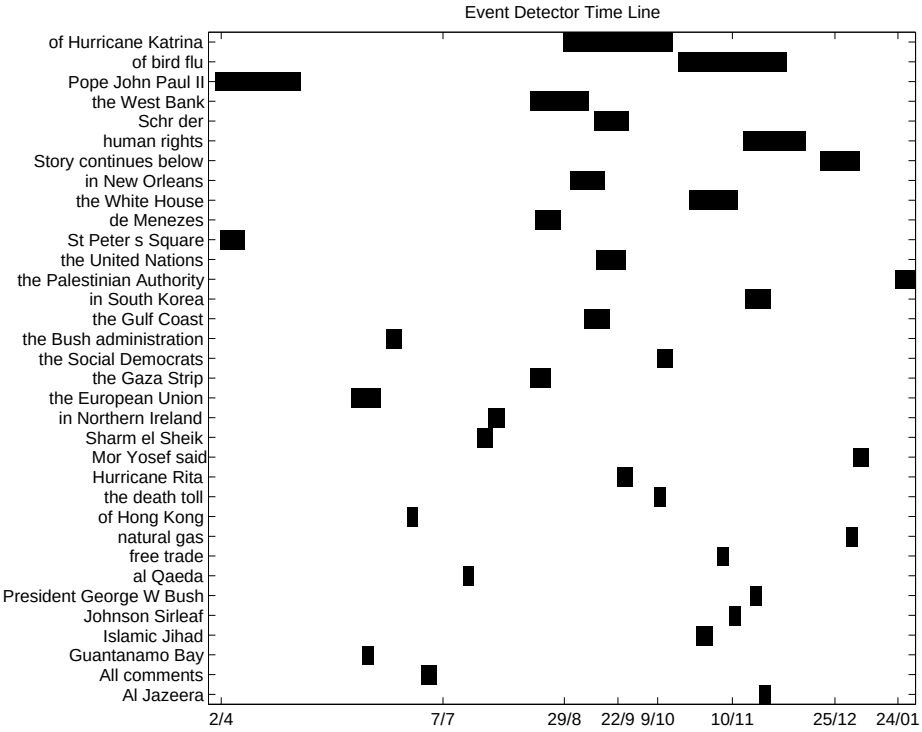

Figure 4.25: Only the main period is shown for the key-phrases. The boundaries have been set equal to 3 and 23 days.

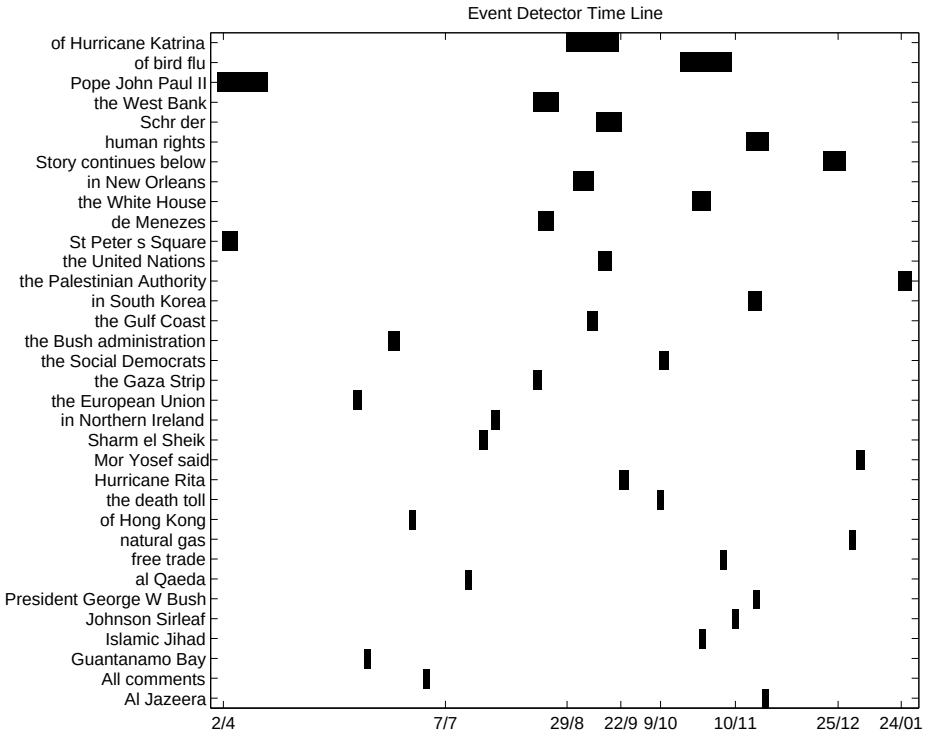

Figure 4.26: Only the main period is shown for the key-phrases. The period has been considered from the beginning to the maximin value achieved by the random walk. The boundaries have been set equal to 3 and 23 days.

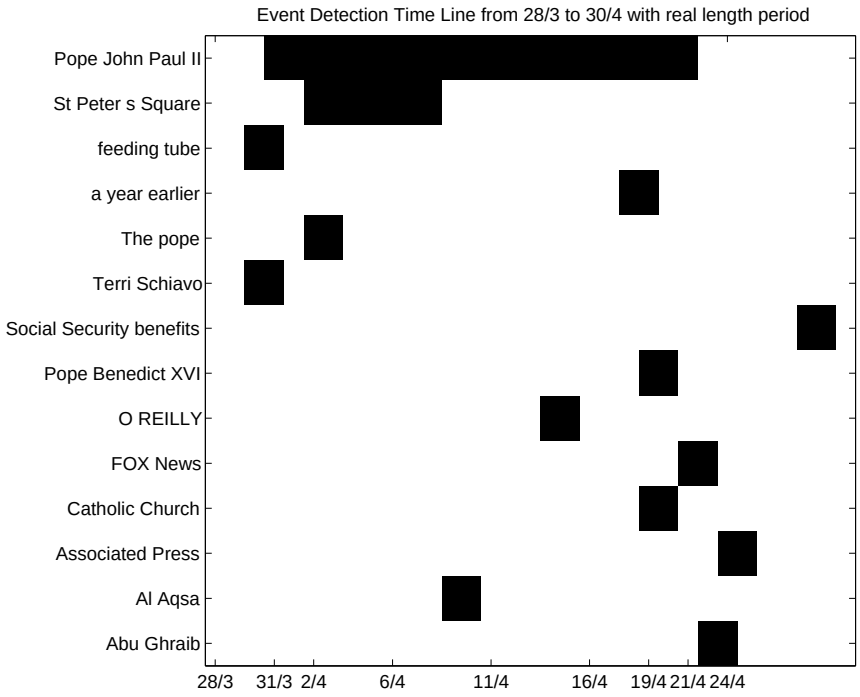

Figure 4.27: Only the main period is shown for the key-phrases that appear in the interval which starts on March 28th and ends on April 30th. The blocks have been considered from the beginning to the maximin value achieved by the random walk. The boundaries have been set equal to 2 and 23 days.

more clear and easy to read than the second. Here many wrong key-phrases are not present, and there are only those key-phrases that have a temporal correlation. Besides, in the first one, we can recognize two clusters of terms, that identify clearly different topics. In fact, the cluster on the top left part of the plot is referred to the Pope's death and the Terry Schiavo's death, while the cluster on the right is shared between the new Pope election and the abuse at the Abu Ghraib prison. The position where to place the change points can be retrieved using the same idea shown in the HMM approach. In this case, the dates March 31st and April 19th are the places where there are many overlaps between key-phrases and a large number of incoming and outgoing key-phrases.

The event detector and the HMM algorithm can be compared also using the cost of computation. The time complexity of the event detection algorithm is $O(m*T)$, where m is the number of key-phrases, and T is the length of each sequence. While the HMM time complexity requires the analysis of the time complexity of the training algorithm, and of the decoding algorithm. The Baum-Welch algorithm has cost $O(\sharp iteration_{EM} * K^2 * T)$, where K is the number of hidden states and $\sharp iteration_{EM}$ is the number of steps computed by the EM algorithm. The computation of the Viterbi algorithm can be performed in $O(m * K^2 * T)$ time. Analyzing these two elements, we see that the time complexity of the HMM approach depends on a unpredictable value like the number of iterations of the EM algorithm. This aspect penalizes the HMM algorithm promoting the event detector.

At the end, we have shown how it is possible to pass from a sequence of news to a time line which contains the most important key-phrases of the most important events. The entire pipeline can be computed automatically and the time line respects the real facts happened during the 8 months analyzed.

The main goal of this work is to bind together classic text analysis features with temporal features. We are able to do that thank to the event detection algorithm which transforms temporal information into constraints for textual features and viceversa. It cleans sequences of key-phrases from

spikes isolating blocks of information and shows the key-phrases in a time line allowing the detection of change points. Some temporal and textual patterns in the time series can be analyzed at the same time. It proposes good solutions to the problems showed by the previous techniques. The use of the variable *time* allows to define clusters of key-phrases using temporal rules. The key-phrases which belong to the same cluster have also semantic correlation. This is obvious, because terms which appear in the same period are emitted with high probability from the same source.

Chapter 5

Works in Progress

5.1 Introduction

In this chapter we propose two new ideas. At the moment, these works are in a preliminary stage and some improvements and tests will be necessary. We however present these methods as a "works in progress" ideas since we think it to be an interesting approaches to verify.

5.2 Language Evolution

We have seen in chapter 3 how "statistical signature" is a good method to address the question of language trajectory during its evolution. We use the same "statistical signature" to analyze a time-series of documents from four romance languages, following their transition from latin. The languages are italian, french, spanish and portuguese, and the time points correspond to all centuries from III bC to XX AD.

We focus on the process of drift of a language in statistical space. We model language evolution as a trajectory in the space of all possible statistical signatures, from an ancestral state to the current one. Modeling this drift is an important long term research goal, and we can only outline our idea. We use a new dataset to measure the distance covered by certain romance languages in the past 22 centuries. We notice some abrupt change points

corresponding to known transitions from latin to national languages.

This preliminary experiment focused on time series analysis of documents spanning 22 centuries within the romance family. We constructed a dataset containing 119 different documents, written in Latin, Italian, Spanish, Portuguese and French, start from 200 BC and including the 20th century. Documents are mostly literary works, chosen to cover uniformly every period and every language. The non latin languages start mostly in the XI century, and have about 12 documents per century.

We measure the distance of each document from the oldest one using the statistical distance between language proposes in chapter 3, and we plot this distance as a function of time. The obvious change point observed in the XI century could be an artefact due to the fact that we could not find earlier documents in the non-latin languages, and is clearly a draw-back of using written language as opposite to spoken one. A more careful choice of the data might help us to reduce the gap in that transition. Also we can find written latin documents throughout the entire period, but we have stopped the latin series more or less where the national languages series started. What is more interesting is that the distance from the origin is in all languages more or less comparable: they all seem to have moved of a comparable amount, in the 22 centuries, although not smoothly (see figure 5.2). We can see the distances between these languages also in figure 5.1, although this multidimensional scaling representation can be misleading (projects into 2 dimensions a 27^2 dimensional dataset).

The idea is in a preliminary stage and more studies are required to can model the drift of languages. This problem opens a new question on the limit of the written languages. At the moment, it is the only source that can give us some information relative to the past times. This is possible because the written language is more stable than the spoken language.

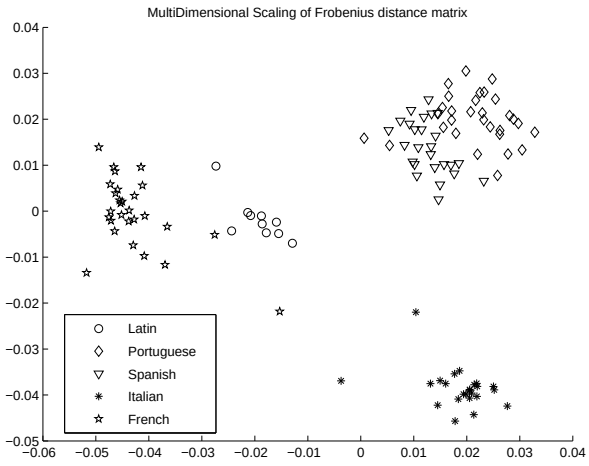

Figure 5.1: Multi Dimensional Scaling of some Romance Languages.

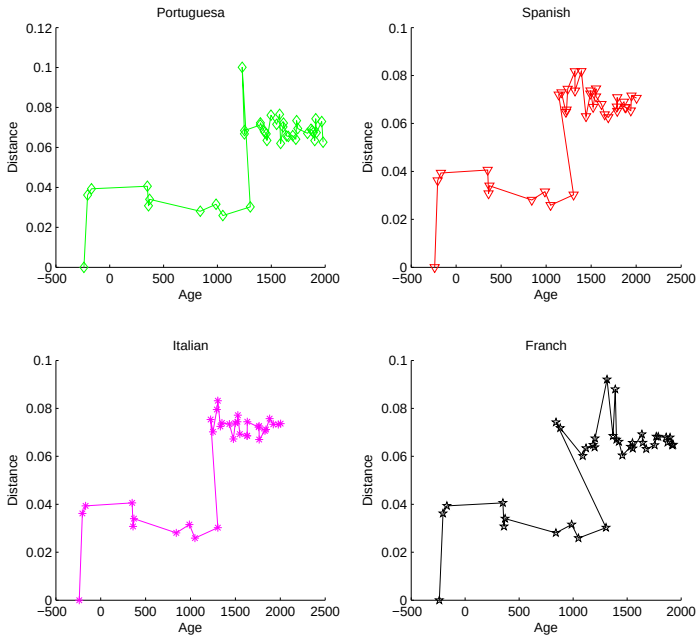

Figure 5.2: Time Series Analysis of some Romance Languages.

5.3 Textual Time Series

In chapter 4 we have proposed three methods to study the time series generated by textual data. There, we use two philosophy to detect time line and change points. In fact, the minimum change points detection approach retrieved the most significative key-phrases when the change points have been detected. While using HMM and the event detection algorithm, we filtered the key-phrases in order to find the optimal time line, then we placed the change points on it.

Here we propose two new approaches to detect change points and then to retrieve the most interesting key-phrases. We present the main ideas that are under investigation and more tests and improvements are necessary.

These methods come from a segmentation algorithm proposed by [27]. Given a sequence of N points, we divide an interval I into subintervals. A partition P of an interval I is a set of M blocks

$$P(I) = \{B_m, m \in \mathfrak{M}\},\ \mathfrak{M} \equiv \{1, 2, \ldots, M\}$$

where the blocks are sets of points defined by index sets $\mathfrak{N}_\mathfrak{m}$

$$B_m = \{X_n, n \in \mathfrak{N}_\mathfrak{m}\}$$

satisfying the usual condition $\bigcup_m B_m = I$ and $B_m \bigcap B_{m'} = \emptyset$ if $m \neq m'$. We define P^* to be the set of all partitions of I into blocks. Take as given an additive *fitness function* that assigns a value to any partition in the form

$$V(P) = \sum_{m=1}^{M} g(B_m)$$

where $g(B_m)$ is the fitness of block B_m. Computationally, the data points must be represented by a data structure that contains sufficient statistics for the model. We use an efficient dynamic programming algorithm (Algorithm 2) that finds an optimal partition $P^{max} \in P^* : V(P^{max}) \geq V(P)$ for all

partitions $P \in P^*$. This algorithm derives the optimal partition of the first $n+1$ data points using previously obtained optimal partitions, i.e., those of the first $1, 2, \ldots, n$ data points. At each iteration we must consider all possible starting locations j, $1 \leq j \leq n$ of the last block of the optimal partition. For each putative j, the fitness function is the fitness of the optimal subpartition prior to j plus the fitness of the last block itself. The former was stored at previous iterations, and the latter is a simple evaluation of V. The desired new optimal partition corresponds to the maximum over all j. Where each optimal partitions start we find a change points.

Algorithm 2: Change Point Detection Using Find Optimal Partition

STATE 1. *Define* $opt(0) = 0$

STATE 2. *Given that* $opt(j)$ *has been determined for* $j = 0, 1, \ldots, n$*:)*

1. *Define* $end(j, n+1) = g(B_{j,n+1})$*;* $B_{j,n+1}$ *is the union of points* $j, j+1, \ldots, n+1$.
2. *Then compute*

$$opt(n+1) = \max_j \{opt(j-1) + end(j, n+1)\}, \text{ for } j = 1, 2, \ldots, n+1$$

3. *The value of* j *where this maximum occurs is stored as* $lastchange(n+1)$

STATE 3. *Repeat 2 until* $n+1 = N$*, when* $opt(N)$*, the optimal partition fitness for all* N *points, has been obtained.*

STATE 4. *Backtrack using the lastchange vector to identify the start points of individual blocks of the optimal partition* P^{max} *in the following way. Let* $n_1 = lastchange(N)$*,* $n_2 = lastchange(n_1 - 1)$*, etc. Then the last block in* P^{max} *contains points* $n_1, n_1 - 1, \ldots, N$*, the next-to-last block in* P^{max} *contains points* $n_2, n_2 + 1, \ldots, n_1 - 1$ *and so on.*

A property of the algorithm is that can be applied whenever any subpartition of an optimal partition is optimal. This implies the Principle of Optimality:

let P^{max} be an optimal partition of I and $P_1 = \{B_m, m \in a\}$ be any subset of the blocks of P^{max}. Then P_1 is an optimal partition of the part of I it covers, namely $I_1 = \bigcup_{m \in a} B_m$. Intuitively, this result follows from the contradiction that a better subpartition of I_1 could be used to construct a partition of I better than P^{max}. The proof relies on the fact that the blockadditivity of the fitness function implies that it is also additive on subpartitions.

Our goal is to find the optimal partition with the lower number of segments. It requires the use of two fitness functions, one for each blocks, and one for the entire partition P. We choose these fitness functions such that they respect the principle of optimality. For each block, we choose:

$$g(B_m) = VAR(B_m) \text{ with } m \in \mathfrak{M} \quad \mathfrak{M} \equiv \{1, 2, \ldots, M\}$$

We can see that the variance is additive on subpartition. In fact, the fitness function of a partition P_i is:

$$V(P_i) = g(B_1) + ...g(B_k) = VAR(B_1) + ... + VAR(B_k) = \sum_{j=1}^{k} VAR(B_j)$$

and it respects the general principle of optimality, because

$$\begin{aligned} V(P) &= \sum_{m=1}^{M} Var(B_m) \\ &= \sum_{m \in a} Var(B_m) + \sum_{m \in b} Var(B_m) \\ &= V(P_1) + V(P_2) \end{aligned}$$

where $P_1 \bigcup P_2 = P$ and $a \bigcup b = \mathfrak{M}$.

Using only this fitness function, the algorithm is not able to minimize the number of partitions, see figure 5.3, where we apply this fitness function on synthetic data.

We introduce an additive function component on V. We subtract the number of blocks for each partition. In that way, we penalize partition with

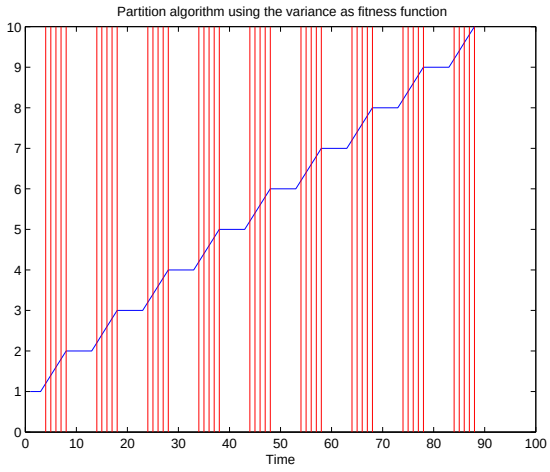

Figure 5.3: We use only the variance as fitness function. The algorithm does not minimize the number of segments.

high number of blocks. The impact of the number of blocks on the fitness function is weighed using a constant factor λ. We obtain:

$$V(P) = (\sum_{m=1}^{M} Var(B_m)) - \lambda M$$

This quantity respects the principle of optimality:

$$\begin{aligned} V(P) &= (\sum_{m=1}^{M} Var(B_m)) - \lambda M \\ &= (\sum_{m \in a} Var(B_m)) - \lambda M_1 + (\sum_{m \in b} Var(B_m)) - \lambda M_2 \\ &= V(P_1) + V(P_2) - \lambda(M_1 + M_2) \end{aligned}$$

and where $P_1 \bigcup P_2 = P$, $a \bigcup b = \mathfrak{M}$ and $M_1 + M_2 = M$.

This change modifies the state 2 of the Algorithm 2 in the follow way:

STATE 2. *Given that opt(j) has been determined for $j = 0, 1, \ldots, n$:)*

1. *For $j, j+1, \ldots, n+1$:*

- *Define* $end(j, n+1) = VAR(B_{j,n+1})$*;* $B_{j,n+1}$ *is the union of points* $j, j+1, \ldots, n+1$*.*
- *The value of* j *is stored as* $lastchange(n+1)$*.*
- *The number of blocks is computed applying the backtrack algorithm on* $lastchange(n+1)$ *and this value is stored in* $NP(j, n+1)$*.*

2. *Then compute*

$$\begin{aligned} opt(n+1) = \quad & \min_j \{opt(j-1) + end(j, n+1) + \\ - \quad & \lambda NumberPartition(n) + \lambda NP(j, n+1)\} \\ & \text{for } j = 1, 2, \ldots, n+1 \end{aligned}$$

3. *The value of* j *where this minimum occurs is stored as* $lastchange(n+1)$
4. *The number of blocks is computed applying the backtrack algorithm on* $lastchange(n+1)$ *and this value is stored in* $NumberPartition(n+1)$

The algorithm works in the same way. For each putative j, we compute the fitness function and the total number of blocks that we obtain adding j. When V is evaluated, we subtract the number of block of the optimal subpartition at the previous step and we add the number of blocks of the new partition. We subtract the number of block of the optimal subpartition at the previous step, because each optimal partition has to depend only its number of blocks. To compute the number of block we use two vectors, NP for number of blocks for each putative j, and $NumberPartition$ for the number of blocks for each optimal subpartition.

This new fitness function allows to reduce the number of segments. We test it on the same synthetic data obtaining encouraging result, see figure 5.4.

We introduce two methods which work using the previous segmentation algorithm with our fitness function. We propose the main ideas of both

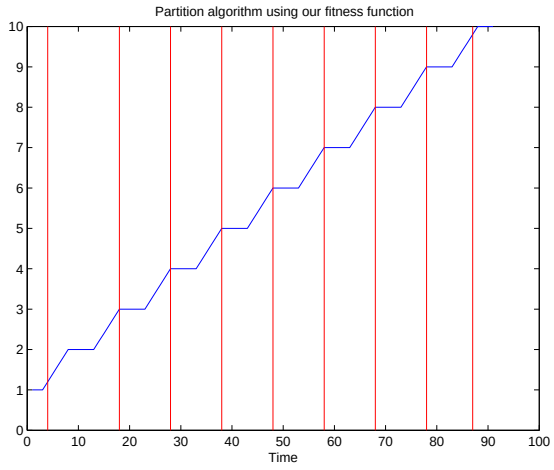

Figure 5.4: We use the variance and the total number of segments as fitness function. The algorithm minimize the number of segments. We set $\lambda = 1.13$

approaches.

The first technique computes the change points for each key-phrase, analyzing each relative time series. We apply the previous algorithm on each row of the matrix KP, see 4.5. Our goal is to find some change points of the whole time series, and not only of each singular key-phrase. For each day this algorithm counts the number of change points present in all the key-phrases and defines as *global* change points the days where there are the highest number of local change points. This approach is very general, because change points come out from the comparisons of all the key-phrases. This method really does not try to respect the initial structure of the data, where for each day we have only ten key-phrases.

The second idea is based on the fact that terms that appear almost in the same days are correlated. This correlation is due to the semantic meaning of the terms. We model this correlation computing the correlation matrix on the rows of the matrix KP, such that each entry gives us a level of correlation between key-phrases. In our case the correlation matrix has the number of rows and columns equal to the total number of key-phrases. Key-phrases that appear almost in the same days, have rows in the KP matrix almost equal, such that the relative value in the correlation matrix is high.

Found the relation between terms, we run a cluster algorithm. We use the hierarchical clustering algorithm describes in 2.2.2. We apply it on the correlation matrix, because we want to split data using semantic relation. This algorithm gives us a set of clusters that contain key-phrases that come from the same sources or adjacent sources.

We compute the change points for each row of the matrix KP as in the first method, but we do not use all the key-phrases to detect the global change points. We define a threshold on the number of elements of the cluster and we select only a set of them. For each day, the algorithm counts the number of change points using only the key-phrases present in each single cluster. According with the idea that each cluster contains terms that come from the same source or adjacent sources, we discover the change points that are strongly correlated with some limited number of sources.

When all the change points have been found, independently by the algorithm used, the last step consists to discover the key-phrases more significative for the change point. The idea is based on the concept that every day some key-phrases appear and other disappear. We define an interval around the change point and we consider relevant all the key-phrases that appear and disappear in this interval. We propose a measure based on the definition of change point in mean proposed in the first chapter.

Given an interval of days I around the change points, it is split in two subintervals I_l and I_r, we define the *Mean Key-Phrases score* for each key-phrases kp_i as:

$$MKPS(kp_i) = \frac{|(Mean(kp_i)_{I_l} - Mean(kp_i)_{I_r})|(\iota(kp_i)_{I_l} + \iota(kp_i)_{I_l})}{Mean(kp_i)} \quad (5.1)$$

where $Mean(kp_i)_{I_l}$ and $Mean(kp_i)_{I_r}$ are the average of the frequency of the key-phrases kp_i computes in I_l and I_r, $Mean(kp_i)$ is the total average of kp_i in the whole series. $\iota(kp_i)$ is a function that counts how many time the kp_i

is present in each interval:

$$\iota(kp_i)_{I_l} = \sum_{d=1}^{length(I_l)} \nu_{i,d}$$

where

$$\nu_{i,d} = \begin{cases} 1 & \text{if the frequency of key-phrase } i \text{ the day } d \text{ is different to } 0 \\ 0 & else \end{cases}$$

This quantity depends on the size of the intervals I_l and I_r.

The meaning of $MKPS$ tries to reflect the typology of key-phrases that we are interested on. The terms that create the change point should have these requirements:

1. have a sensible skip from the average of the left and right interval;
2. be not present in the whole time series;
3. have not only a big spike in one of the two intervals.

These three points are respected by the maximization of $MKPS$ in the follow way. The quantity at the numerator computes the skip between the intervals and penalizes terms that belong few time in the intervals. The total average of kp_i on the whole series at the denominator penalizes terms that are present quite every days. Maximizing $MKPS$, we try to find the optimal compromise between all these factors.

All the methods and measures require further study and tests. In particular, when change points have been detected for each key-phrase, the search of the global change point is not easy. We propose to add all local change points for each day, but it could not be the correct choice. The next steps consist to study this point and to define a different solutions that used mostly the proprieties of the change points.

Appendix

Time Lines

These time lines contain the most important events from April 2005 to December 2005. They have been constructed by hand and have been used to check the results obtained in chapter 4.

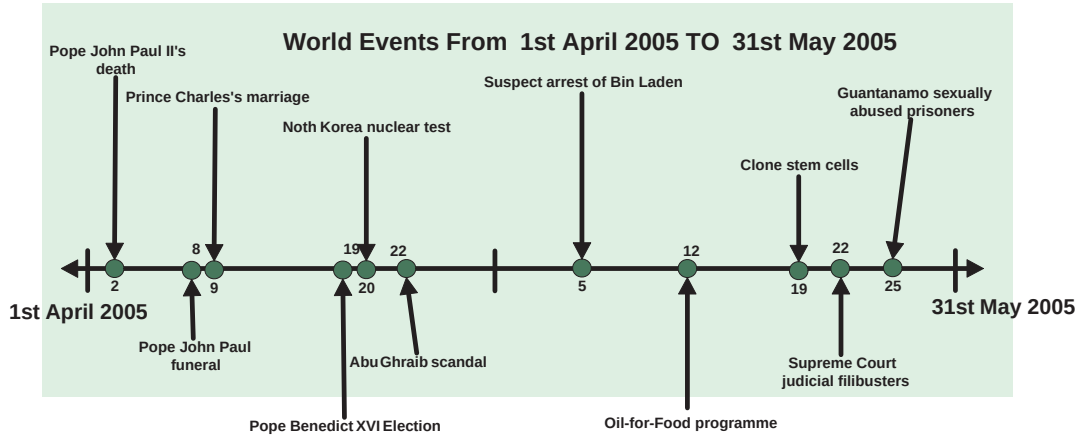

Figure 5: The most important events occurred during the months April and May 2005.

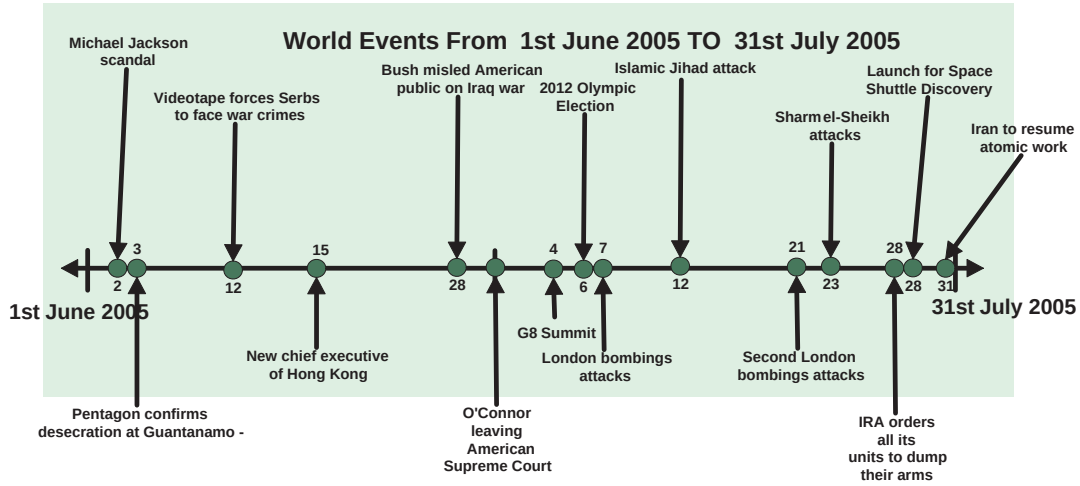

Figure 6: The most important events occurred during the months June and July 2005.

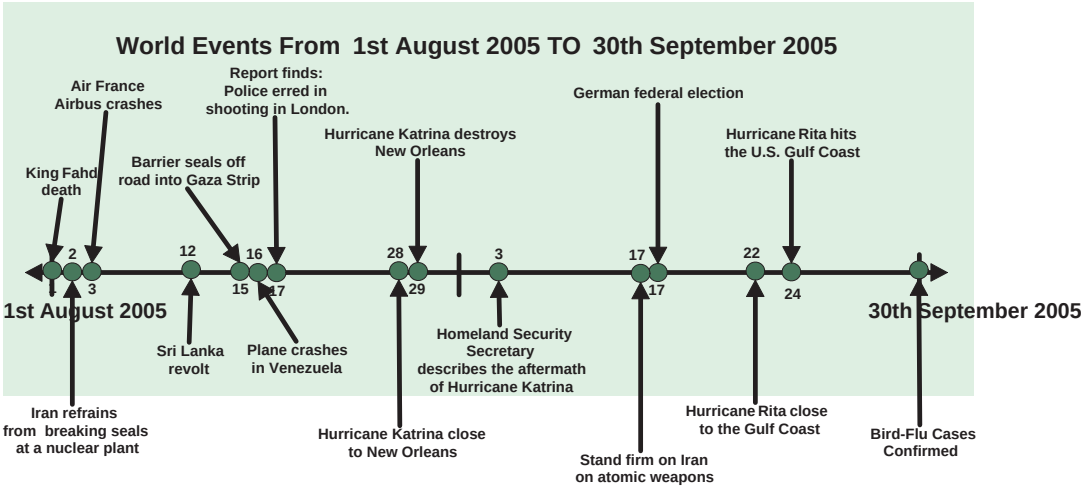

Figure 7: The most important events occurred during the months August and September 2005.

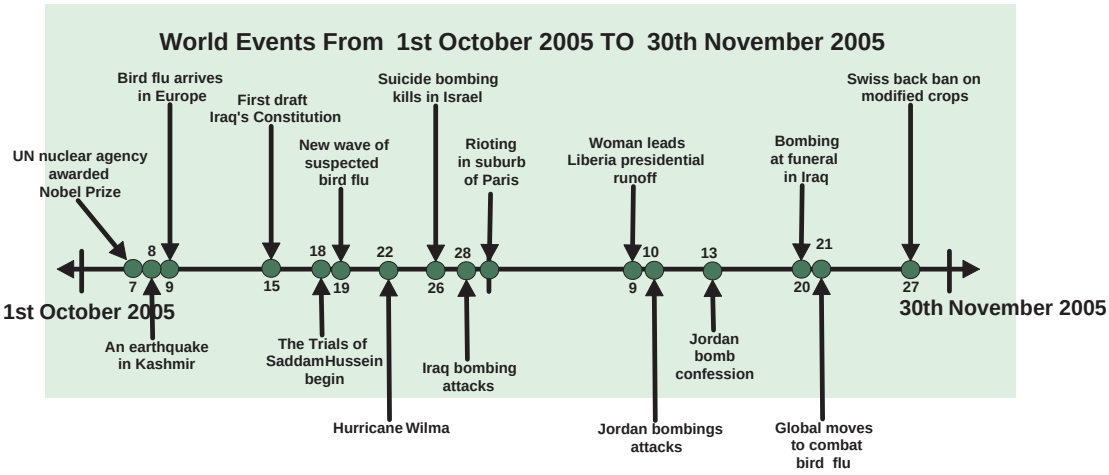

Figure 8: The most important events occurred during the months October and November 2005.

Figure 9: The most important events occurred during the months December 2005 and January 2006.

Bibliography

[1] N. Meade. The use of growth curves in forecasting market development - a review and appraisal. *J. Forecasting*, 3:429–451, 1984.

[2] P.H. Franses. Time series model for business and economic forecasting. *Cambridge Univ. Press*, 1998.

[3] A. Stuart M.G. Kendall and J.K. Ord. *The advance theory of statistics*, volume 3. London: Griffin, 4th edition, 1983.

[4] C. Chatfield. *The analysis of time series. An introduction.* Chapman and Hall.CRC, 6th edition, 2004.

[5] P. Whittle. On the fitting of multivariate autoregressions, and the approximate factorization of a spectral density matrix. *Biometrika*, 50:129–134, 1963.

[6] D.G. Watts G.M. Jenkins. *Spectral Analysis and its Applications.* San Francisco: Holden Day, 1968.

[7] G.M. Jenkins G.E.P. Box. *Time-Series Analysis, Forecasting and Control.* San Francisco: Holden Day, 1970.

[8] J. Pignatiello T. Samuel and J. Calvin. Identifying the time of a step change in a normal process variance. *Quality Engineering*, 10:529–538, 1998.

[9] P. Qiu. Estimation of the number of jumps of the jump regerssion functions. *Comminications in Statistics Theory and Methods*, 23:2141–2155, 1994.

[10] D. Donoho and I. Johnstone. Adapting to unknow smoothness via wavelet shrinkage. *Journal of the American Statistical Association*, 90:1200–1224, 1995.

[11] P. Hall I. Gijbels and A. Kneip. On estimation of jump points in smooth curves. *The Annals of the Institute of Statistical Mathematics*, 51:231–251, 1999.

[12] G. Gregoire and Z. Hamrouni. Change points estimation by local linear smoothing. *Journal Multivariate Analysis*, 83(1):56–83, 2002.

[13] C. Inclan and G.C. Tiao. Use of cumulative sums of squares for retrospective detection of changes of variances. *Journal of the American Statistical Association*, 427:913–923, 1994.

[14] P. Guttorp B. Witcher and D. Percival. Multiscale detection and location of multiple variance changes in the presence of long memory. *Journal of Statistical Computation and Simulation*, 68:65–88, 1999.

[15] S. Adak. Time dependent spectral analisys of nonstationary time series. *Journal of the American Statistical Association*, 93:1488–1501, 1998.

[16] W.D. Fisher. On grouping for maximum homogeneity. *Journal of the American Statistical Association*, 53(284):789–798, December 1958.

[17] C.S. Wallace and D.M. Boulton. An information measure for classification. *Computer Journal*, 11(2):185–194, August 1968.

[18] C.S. Wallace and D.L. Dowe. Minimum message length and kolmogorov complexity. *Computer Journal*, 42(4):270–283, 1999.

[19] C.S. Wallace and P.R. Freeman. Estimation and inference by compact encoding (with discussion). *Journal of Royal Statistical Society series B*, 49:240–265, 1987.

[20] R.A. Baxter and J.J. Oliver. The kindest cut: minimum message length segmentation. In S. Arikawa and A.K. Sharma, editors, *Proc. 7th Int. Workshop on Algorithmic Learning Theory*, volume 1160, pages 83–90, Springer-Verlag Berlin, 1996.

[21] R.A. Baxter J.J. Oliver and C.S. Wallace. Minimum message length segmentation. In R. Kotagiri X. Wu and K. Korb, editors, *Research and Development in Knowledge Discovery and Data Mining (PAKDD-98)*, pages 83–90, Springer, 1998.

[22] J.J. Oliver and C.S. Forbes. Bayesian approaches to segmenting a simple time series. Technical Report 97/336, Dept. Computer Science, Monash University, Australia 3168, December 1997.

[23] R.E. Kass and A.E. Raftery. Bayes factors. *Journal of the American Statistical Association*, 90(430):773–795, 1995.

[24] H. Akaike. Information theory and an extansion of the maximum likelihood principle. In B.N. Petrov and F. Csaki, editors, *Proceeding 2nd International Symposium on Information Theory*, pages 267–281, Akademia Kiado, Budapest, 1973.

[25] J.J. Rissanen. Modeling by shortest data description. *Automatica*, 14:465–471, 1978.

[26] L. Allison L.J. Fitzgibbon and D.L. Dowe. Minimum message length grouping of ordered data. In S. Jain H. Arimura and A. Sharma, editors, *Proc. 11th Int. Workshop on Algorithmic Learning Theory*, pages 56–70, Sydney, 2000.

[27] D. Barnes S. Arabhi A. Alt P. Gioumousis E. Gwin P. Sangtrakulcharoen L. Tan B. Jackson, J.D. Scargle and T.T. Tsai. An algorithm for optimal partition of data on an interval. *IEEE Singal Processing Letter*, 12(2):105–108, 2005.

[28] J.V. Braun and H.G. Muller. Statistical methods for dna sequence segmentation. *Statistical Science*, 13(2):142–162, 1998.

[29] Y.X. Fu and R.N. Curnow. Maximum likelihood estimation of multiple change points. *Biometrika*, 77:563–573, 1990.

[30] A.G. Churchill. Stochastic models for heterogenous dna sequnces. *Bullettin of Mathematical Biology*, 51:79–94, 1989.

[31] A.G. Churchill. Hiddem markov chain and the analysis of genome structure. *Computers in Chemistry*, 16:107–115, 1992.

[32] J.S. Liu and C.E. Lawrence. Unified gibbs method for biological sequence analysis. In *Proceedings of the Biometrics Section*, pages 194–199, Alexandria, VA, 1996. Amer. Statist. Assoc.

[33] A. Wong G. Salton and C.S. Yang. A vector space model for information retrieval. *Journal of the American Society for Information Science*, 18(11):613–620, November 1975.

[34] C.S. Yang G. Salton and C.T. Yu. A theory of term importance in automatic text analysis. *Journal of the American Society for Information Science*, 26(1):33–44, 1975.

[35] G. Salton and M.J. McGill. *An Introduction to Modern Information Retrieval*. 1983.

[36] M.F. Porter. An algorithm for suffix stripping. *Program*, (14):130–137, 1980.

[37] H.P. Zipf. Selective studies and the principle of relative frequency in language. 1932.

[38] H.P. Zipf. Human behaviour and the principle of least effort. 1949. Cambridge,Massachusetts.

[39] H.P. Luhn. The automatic creation of literature abstracts. *IBM Journal of Research and Development*, (2):159–165, 1958.

[40] F. Debole and F. Sebastiani. Supervised term weighting for automated text categorization. 18th ACM Symposium on Applied Computing, 2003.

[41] Y. Yang and J. O. Pedersen. A comparative study on feature selection in text categorization. *In International Conference on Machine Learning*, pages 412–420, 1997.

[42] R. A. Calvo and HA Ceccatto. Intelligent document classification. *Intelligent Data Analysis*, 4(5):411–420, 2000.

[43] D. Mladenic and M. Grobelnik. Feature selection for clasification based on text hierarchy. In *Conference on Automated Learning and Discovery CONALD-98*, 1998.

[44] D. Mladenic. Feature subset selection in text-learning. In *10th European conference on Machine Learning ECML98*, 1998.

[45] W. B. Goh H. T. Ng and K. L. Low. Feature selection, perceptron learning, and a usability case study for text categorization. *20th ACM International Conference on Research and Development in Information Retrieval*, pages 67–73, 1997. Philadelphia Us.

[46] J. O. Pedersen E. Wiener and A. S. Weigend. A neural network approache to topic spotting. *4th Annual Symposium on Document Analysis and Information Retrieval*, pages 317–332, 1995. Las Vegas Us.

[47] S. Deerwester S. Dumais G. Furnas T. Landauer and R. Harshman. Indexing by latent semantic analysis. *Juornal of American Society for Information Science*, 41(6):391–407, 1990.

[48] G. H. Goulub and C. F. Van Loan. *Matrix Computation*. Baltimora, MD, US, 1996.

[49] W.B. Johnson and J. Lindenstrauss. Extensions of lipshitz mapping into hilbert space, in conference in modern analysis and probability. *In Amer. Math. Soc.*, 26:189–206, 1984.

[50] S. Dasgupta and A. Gupta. An elementary proof of the johnson-lindenstraus lemma. Technical report, Technical Report TR-99-006, Berkeley, CA, 1999.

[51] I. S. Dhillon and D. S. Modha. Concept decomposition for large sparse text data using clustering. *Machine Learning*, 42(1-2):143–175, 2001.

[52] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. 1990.

[53] E. M. Voorhees. Implementing agglomerative hierarchic clustering algorithms for use in document retrieval. *Information Processing and Management*, 6(22):465–476, 1986.

[54] J. O. Pedersen D. R. Cutting, D. R. Karger and J. W. Turkey. Scatter/gather: a cluster-based approach to browsing large document collections. In *In 15th International ACM SIGIR Conference on Research and Developement in Information Retrieval*, pages 318–329, 1992.

[55] J. MacQueen. Some methods for classification and analysis of multivariate observations. *In 5th Berkeley Symposium*, 1:281–297, 1967.

[56] M. Steinbach, G. Karypis, and V. Kumar. A comparison of document clustering techniques. *In KDD Workshop on Text Mining*, 2000.

[57] J. Sander M. Ester, H.P. Kriegel and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *In Proceeding of the 2nd International Conference on Knowledge Discovery and Data Mining (KDD96)*, 1996.

[58] J. Bilmes. A gentle tutorial on the em algorithm including gaussian mixture and baum-welch. Technical Report TR-97-021, Internationa Computer Science Institute, Berkeley, CA, May 1997.

[59] P.S. Bradley and U. M. Fayyad. Refining initial points for k-means clustering. In *In Proc. 15th International Conference on Machine Learning*, pages 91–99, Morgan Kaufmann San Francisco CA, 1998.

[60] W. M. Rand. Objective criteria for the evaluation of clustering methods. *Journal of the American Statistical Association*, 66:846–850, 1971.

[61] B. Mirkin. Mathematical classification and clustering. *Kluwer Academic Press*, 1996.

[62] A. Elisseeff A. BenHur and I. Guyon. A stability based method for discovering structure in clustered data. *In Pacific Symposium on Biocomputing*, pages 6–17, 2002.

[63] D. L. Wallace. Comment. *Journal of the American Statistical Association*, 78:569–576, 1983.

[64] E.B. Fowlkes and C.L. Mallow. A methods for comparing two hierarchical clustering. *Journal of the American Statistical Association*, 78:553–569, 1983.

[65] L. Hubert and J. Schultz. Quadratic assignment as a general data-analysis strategy. *British Journal of Mathematical and Statistical Psychology*, 29:190–241, 1976.

[66] R. Sibson. Slink: an optimally efficient algorithm for a complete link method. *The Computer Journal*, 16:30–34, 1973.

[67] J. H. Ward Jr. Hierarchical grouping to optimize an objective function. *Journal of the Americal Statistical Association*, 301(58):235–244, 1963.

[68] Marco Maggini, Leonardo Rigutini, and Marco Turchi. Pseudo-supervised clustering for text documents. In *Web Intelligence*, pages 363–369, 2004.

[69] Kevin Atkinson. Gnu aspell 0.50.5 - manual, 2004.

[70] S. Karlin, J. Mrázek, and A. M. Campbell. Compositional biases of bacterial genomes and evolutionary implications. *Journal of Bacteriology*, 179(12):3899–3913, June 1997. 0021-9193/97/04.0010.

[71] C. E. Shannon. Prediction and entropy of printed english. *Bell Systems Technical Journal*, (30):50–64., 1951.

[72] J. Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004. http://www.kernel-methods.net.

[73] Kenneth R. Beesley. Language identifier: A computer program for automatic natural-language identification of on-line text. *The 29th Annual Conference of the American Translators Association*, pages 47–54, October 1988.

[74] C. Leslie and R. Kuang. *Fast Kernels for Inexact String Matching. In* Conference on Learning Theory. *Columbia University, New York NY 10027, USA, 2003.*

[75] N. Saitou and M. Nei. The neighbour-joining method: a new method for constructing phylogenetic trees. Mol. Biol. Evol*, 1987.*

[76] A.J. Studier and K.J. Keppler. A note on the neighbor joining algorithm of Saitou and Nei. *Molecular Biology and Evolution*, (5):729–731, 1988.

[77] J.B. Kruskal and M. Wish. Multidimensional Scaling. Beverly Hills, CA, 1978.

[78] D. Benedetto, E. Caglioti, and V. Loreto. Language trees and zipping. *Physical Review Letter*, 88(4), January 2002.

[79] Universal declaration of human rights, December 1948. United Nations General Assembly Resolution.

[80] M. Li, X. Li, B. Ma, and P. Vitanyi. Similarity distance and phylogeny. *IEEE Transactions on Information Theory*, 50(12), December 2004.

[81] D. Ringe, T. Warnow, and A. Taylor. Indo-european and computational cladistics. *Transactions of the Philological Society*, 2002. 00 (1):59-129.

[82] M.A. Nowak and D.C. Krakauer. The evolution of language. *Proc. Natl. Acad. Sci. USA*, 1999.

[83] G. Perrière and M. Gouy. *WWW-Query: An on-line retrieval system for biological sequence banks. Biochimie, 1996. http://pbil.univ-lyon1.fr/software/.*

[84] R. Bellman. An introduction to the theory of dynamic programming. The Rand Corporation*, 1953. Santa Monica, Calif.*

[85] R. Bellman. Dynamic programming. Princeton University Press*, 1957. Princeton, N.J.*

[86] Christopher D. Manning and Hinrich Schütze. Foundations of Statistical Natural Language Processing. *MIT Press, 2000. MAN ch2 00:1 1.Ex.*