Roberto Basili
Alessandro Moschitti

Automatic Text Categorization

From Information Retrieval
to Support Vector Learning

www.aracneeditrice.it
info@aracneeditrice.it

via Raffaele Garofalo, 133 A/B
00173 Roma
(06) 93781065

ISBN 88–548–0292–1

I edizione: novembre 2005

Preface

In the Summer of 2001, at the International Joint Conference on Artificial Intelligence held in Seattle, we presented a text categorization system based on an extension of the empirical approach known as "Rocchio" classifier, fully inspired by the Information Retrieval literature. At that time, Rocchio seemed to represent the best choice for our research projects (categorization of documents from major news agencies, e.g. Reuters and Ansa) given its computational advantages, i.e. very fast learning and classification run time and its reasonable accuracy.

The Rocchio text classifier derives category profiles (e.g. foreign politics or sports) from a representation of the data in a metric space. Such profile building is the actual Rocchio training and relates to the extraction of statistical properties from a vast number of examples of already classified documents. Once profiles are derived, the classification of an incoming news item d is reduced to the process of computing the "similarity" of the vector representing d and the vector expressing the profile of a target class: when these vectors are enough "close" each other the system will accept d as a correct member of the class, otherwise d is judged to not belong to such class. In principle, the larger the set of the "training" examples is, the more accurate the categorization results are.

The major problem of the Rocchio approach is its weakness in producing accurate profiles. The two most important reasons are the following. First, there is no principled way to decide which examples are more important (and how) for the induction of the classification function (i.e. to determine the "closeness" property). Second, no method is available to determine how many examples are needed to reach a given (and possibly required) categorization accuracy. The area in which the two above problems are studied is the Statistical Learning Theory which aims to characterize example-driven learning tasks by determining their feasibility, inherent complexity, convergence criteria and expected error rate. When faced with realistic problems (e.g. automatic categorization systems for news agencies, digital libraries or other Web applications) the above pieces of information are relevant as they critically impact on the engineering choices of the system.

Such theory is also critical for the design of modern Natural Language and Text Processing applications as it provides analytical techniques that are essential to study problems related to coverage, accuracy, robustness and portability of automatic systems. Activities like data collection, manual annotation

and induction of linguistic resources have become essential in the design of any Natural Language application since late '80. Nevertheless, the characterization of such learning tasks is still problematic. Questions like "How many examples do I need to train my system?", "Which features are important for my learning task?", "How do I have to annotate my data?" receive different, competing and often diverging answers in the scientific literature. Moreover, the extremely interdisciplinary nature of the area does not help: linguists are usually not able to figure out how to make use of (or to give computational sense to) their insights on data whereas engineers may collect huge data sets but have not enough insights to make the best use of them.

Any scholar in Artificial Intelligence, Natural Language Processing, Information Retrieval, Computational Linguistics as well as Computer Science, in general, has an inherent interest into the theoretical and methodological results in Statistical Classification. Indeed, such results can be exploited to effectively design a quite large number of different applications ranging from Document Categorization, Spam Filtering to Part-Of-Speech Tagging, Word Sense Disambiguation, Question Answering, Syntactic Parsing and Semantic Role Labeling. This book tries to provide the basic notions of the statistical learning theory and its application to Text Categorization with a specific emphasis on engineering perspectives and principles of best practice valid for real scenario applications.

We think that Automatic Text Categorization is a prototypical problem for several other NLP tasks, thus we hope that the collection of ideas and the empirical evidence discussed in this book (originated by several years of research) will be a useful guide to heterogeneous student communities (e.g. computer science as well as computational linguistics). Although most of the questions on learnability of linguistic phenomena will still remain unanswered after reading this book, some of the results discussed here (i.e. successes and defeats) are strongly connected to many general natural language applications. Hence this book may serve as a guide for the problem solutions of innovative and complex natural language systems.

Roberto Basili and Alessandro Moschitti

Roma. October, 30th 2005

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The success of modern Information Technologies and Web-based services critically relates to the access, selection and managing of large amounts of information usually expressed as textual data. Among the others the Information Retrieval (IR) research has studied and modeled methodology to organize and retrieve targeted data from large collection of unstructured documents.

Mainly, IR provides two facilities for the access to the target information: query based and category browsing. The former has originated search engines like *Google* or *Altavista* whereas an example of the latter, can be observed in the category hierarchy of *Yahoo*, e.g., *Arts*, *Business*, *Computers*, *Culture*, *Education*, *Entertainment*, *Health*, *News*, *Science* and so on. Users can more easily browse the set of documents of their own interests by firstly selecting the category of interest. Additionally, users can specify their information needs by means of a selection of basic categories so that sophisticated IR models can automatically author to them the pertinent documents, i.e. documents which belong to such categories.

The large amount of documents involved in the above processes requires the design of systems that automatically assign category labels to the documents. In the last years, a large variety of models and approaches have been proposed for such task. As a consequence a subarea of IR called automated Text Categorization (TC) has emerged. Usually, TC is based on Machine Learning and statistical techniques inherited from the IR studies. Such methods are applied to a set of documents, manually assigned to the target categories (*training set*), to automatically learn the target classification function.

Learning algorithms attempt to derive statistical properties from the doc-

uments of a target category that are not held by the documents of the other categories. Such properties are then used at classification time to decide the document class. To allow the learning algorithm to successfully carry out such task, we need to provide it with document representations from which interesting properties can be extracted. For example, if we represent documents with the number of vowel types which are present in them, most likely, the algorithm will not extract properties able to distinguish between *Sport* and *Medicine*. On the contrary, the set of document words seems to be sufficient to achieve accurate representations.

Such representation is very common in IR and it is often referred to as *bag-of-words* as documents are encoded as simple multisets of words neglecting important linguistic aspect of natural language like morphological, syntactic and semantic structures. Nevertheless, this approach has shown high accuracy in automated classification, usually provided by the percentage of correct assignments within a set of documents not used for training, i.e. the *test set*.

Correctly measuring the accuracy of a classification system has become a crucial issue as some specific sectors, ranging from changes in management positions to business intelligence or information about terrorist acts, strongly relate on precise selection of the targeted data. Such necessity has produced more and more accurate TC learning models by studying two strategies: (a) improving categorization algorithms by using several theoretical learning models (e.g., [Joachims, 1998; Yang, 1999; Tzeras and Artman, 1993; Cohen and Singer, 1999; Salton and Buckley, 1988; Ng *et al.*, 1997; Moulinier *et al.*, 1996; Apté *et al.*, 1994; Quinlan, 1986; Hull, 1994; Schütze *et al.*, 1995; Wiener *et al.*, 1995; Dagan *et al.*, 1997; Lewis *et al.*, 1996; Ittner *et al.*, 1995]) and (b) designing document representations more sophisticated than *bag-of-words*.

Unfortunately, complex learning algorithms often show a high time complexity for both training and classification. This makes difficult the adoption of such algorithms for operational scenarios, where the number of instances is very large. For instance, web applications require effective data organization and efficient retrieval as for huge and growing amount of documents. To govern such complexity, the current trend is the design of efficient TC approaches [Lewis and Sebastiani, 2001]. A careful analysis of the literature reveals that linear classifiers are the most (computationally) efficient models [Sebastiani, 2002]. These are based on a vector representation of both documents and categories by means of feature weights derived via different approaches [Hull, 1994; Schütze *et al.*, 1995; Wiener *et al.*, 1995; Dagan *et al.*, 1997; Lewis *et al.*, 1996; Cohen and Singer, 1999]. The decision if a document belongs or not to a category is then made measuring the similarity between the target vector pair (i.e., document and category).

Among others there are two linear classifiers which are the most representative of text categorization literature: Rocchio's text classifier [Rocchio, 1971] and Support Vector Machines ($SVMs$) [Joachims, 1999]. The former has origins in the IR literature, is one of the most computationally efficient classifier in both training and classification phases, is based on a heuristic derived from the experience in document retrieval and has been using since the early age of IR to take into account of the user feedback in search engine queries. The latter embodies the latest results in statistical learning theory [Vapnik, 1995], is considered one of the most accurate classifier and shows several important properties such (1) the ability to work in very high dimensional spaces and (2) the possibility via kernel functions to learn non-linear models.

With the same aim of improving accuracy of text classifier several researches on the use of a richer document representation have been carried out. Linguistic structures [Voorhees, 1993; Strzalkowski and Jones, 1996] could embed more information than the simple words which helps TC systems to learn the differences among different categories. Typical structures experimented in IR are complex nominals, *subject-verb-object* relations and word senses. This latter, is particularly useful in representing the document content unambiguously. For example the *slide* as transparency for projectors and the *slide* as sloping chute for children are the same words whereas the meaning is completely different. Such richer representations, are usually extracted by applying some automatic Natural Language Processing (NLP) techniques, but, at the moment they have failed to improve TC.

Although we are not able to design richer linguistic structures to improve TC systems. The current machine learning models based on bag-of-words are enough accurate and efficient to be applied to real scenario applications. Among other, Information Extraction (IE), *Question/Answering* (Q/A) and Text Summarization (TS) are useful applications that can be improved by the use of TC. TC can help in locating specific documents within a huge search space (*localization*) while IE or Q/A support the focusing on specific information within a document (*extraction* or *explanation*). Text classifiers provide for each document a set of categories that indicate what the main subjects of the documents are. For instance, text classifiers can assign categories to small texts also, e.g., paragraphs or passages. This knowledge can be exploited by IE, Q/A

and TS systems to respectively extract the events of a certain type, choose the answers related to the target subject or select the important passages related to a specific domain.

The many potential application fields for which TC may be successfully applied have aroused the interest in defining a methodology for TC system design. Indeed, there are a set of commonly recognized steps that should be applied to obtain an accurate TC system. The next section introduces such techniques.

1.1 Designing a Text Classifier

The design of generic text classifiers includes a set of steps universally recognized by the research community. We will briefly summarize them in the following by postponing to Chapter 3 their in depth discussion:

- *Features design*, where the following pre-processing steps are carried out:
  - *Corpus processing*: filtering and formatting of all the corpus documents.
  - *Extraction of relevant information*: a *stop list* is applied to eliminate function words (that exhibit similar frequencies over all classes) and the interesting linguistic information is extracted. The usual approaches use words as basic units of information but more complex features may be built, e.g. structured patterns like syntagmatic expressions or word senses.
  - *Normalization*: word stemming, carried out by removing common suffixes from words (words after stemming are usually called *stems*.). This is a classical method to approximate a conceptual representation, e.g. the *acquire* concept can also be expressed as *acquisition* and *acquires*. When more complex features are derived via linguistic analysis (i.e. words and/or complex nominals), normalization usually refers to the lemmatization process (i.e. detection of the base form of rich morphological categories such as nouns or verbs[1])

[1]Notice that this is very important for languages with a rich generative morphology where even hundreds of different forms can be derived from the same root.

  - *Feature selection*, which is an attempt to remove non-informative features from documents to improve categorization accuracy and reduce computational complexity. Typical selection criteria are based on statistical quantities like Chi Square, Mutual Information or document frequency.
- *Feature Weighting*: features usually assume different roles in different documents, i.e. they can be more or less representative. Different weights are associated with features via different and possibly diverging models.
- *Similarity estimation* which is modeled via operations in feature spaces. This can be carried out between pairs of documents or between two more complex feature groups (e.g. profiles which are the combination of features coming from different representative documents). Such estimation is typically adopted with quantitative models.
- *Application of a machine learning model*: once the similarity estimation has been defined in most of the cases the target machine learning algorithm can be applied. In this phase, a set of training documents whose correct classifications are known is needed. The output of the learning phase is a model of one or more categories.
- *Inference*: the similarity (or the membership function) between documents and category models is used to make classification decisions. The assignment of an incoming document to a target class is based on a decision function over the similarity scores. Different criteria (i.e. purely heuristics or probability-driven rules) are used in this task.
- *Testing*: the accuracy of the classifier is evaluated by using a set of pre-labeled documents (i.e. *test set*) which are not used in the learning phase (*training set*). The labels produced by the target classifier are compared to those from the gold standard. The result of this phase is usually one or more numerical values which provide a measure of the distance between the human classifications (i.e. the Gold Standard classifications) and the target automatic classifier.

It should be pointed out that there are two types of classifiers: those that can make decisions over a set of categories (multiclassifiers) by outputting a set of category labels and those that can only decide if a document belongs or not

to a target category (binary classifiers). By training a binary classifier for each category, we can design a multiclassifier. A binary classifier can be trained by separating the corpus documents in two different sets: (1) the positive documents that are categorized in the target class and (2) the negative documents that are not categorized in it. To classify a new document, it is enough to apply the pool of binary classifiers and collect their decisions.

Such approach is usually applied to profile-based classifiers since they inherently express binary decisions. They describe each target class (C_i) in terms of a profile, i.e. a vector of weighted features. Such vector is extracted from documents previously categorized under C_i (training set). The classification phase is accomplished by evaluating the similarity between an incoming document d and the different profiles (one for each class).

More in detail, to design *Profile-based* classifiers, we apply phases which are slightly different from the previous design steps:

- *Features Weighting* for documents and profiles:
  - A representation vector $\vec{d}$ of a document d is derived by extracting its features f and assigning the weights $\vec{d}_f$ to them.
  - A representation vector $\vec{C}_i$ of a class C_i is evaluated by considering $\vec{d}$ belonging to C_i (i.e. $d \in C_i$) as positive examples and the vector $\vec{d} \notin C_i$ as negative examples.
- Application of a machine learning model: the output of the learning of profile-based classifiers is the vector $\vec{C}_i$. In this sense the machine learning algorithm can be seen as a weighing model for the category profile vector.
- *Similarity estimation* which is always carried out between unknown (i.e. not classified) documents and the profiles. The similarity is usually derived within the space determined by the features, i.e. between $\vec{d}$ and $\vec{C}_i$.
- *Inference*: A decision function is applied over the similarity scores. The most widely used inference methods are: probability, fixed and proportional thresholds. These are respectively called in [Yang, 1999]: $Scut$ (a threshold for each class is applied and is used to decide whether or not a document belongs to it), $Rcut$ (the best k-ranked classes are assigned to each document) and $Pcut$ (the *test set* documents are assigned to the classes proportionally to their size).

It should be noted that the choices made at each step determine a different classification model which may be more appropriate for an application domain rather than another. For example, it is well known that there is no optimal weighting scheme as its performance strongly depends on the application corpora.

The above rationale does not exactly apply for the choice of the machine learning algorithms as often there are a few models that are more accurate than all the others. In the next section, we briefly report the most famous machine learning approaches used for TC to give a flavor of this prolific research field.

1.2 Machine Learning Approaches to Text Categorization

In the literature several TC models based on different machine learning approaches have been developed. Whatever technology is adopted, the TC system suffers from the trade off between accuracy in retrieval and time complexity of the training/testing phases. Such complexity is critical in operational scenarios since it may prevent the processing of all required data. In the following, we briefly revisit the best known approaches as well as more recent ones. Particular carefulness to operational aspects will be devoted.

Support Vector Machines (SVMs), recently proposed in [Joachims, 1999], are applied to the vector space model described in Chapter 3 to find a category profile which produces the *lowest probability error*[2] on document classification. Such properties allow the SVMs to achieve one of the highest accuracy (about 86%) on a well known TC benchmark called Reuters corpus.

The main SVM problems are their application to operational scenarios where the number of training documents is thousands times higher than the number of documents contained in benchmarks. The disadvantage of SVMs is the training time which is quadratic in the numbers of training examples. The classification phase also can be very slow for nonlinear SVMs [Vapnik, 1995] since it is proportional to the number of support vectors which, in turn, increase proportionally with the number of training documents. This means that, to classify each single document, thousands of support vectors could be involved. As each support vector requires a scalar product with the input documents the time is usually very high.

[2]The exact meaning of *lowest probability error* will be explained in Section 2.

KNN is an example-based classifier [Yang, 1994] which makes use of document to document similarity estimation. It selects a class for a document through a k-Near Neighbor heuristic. For this the algorithm requires the calculation of all the scalar products between an incoming document and those available in the *training set*. The optimization proposed by the EXP-NET algorithm [Yang, 1994] reduces the computational complexity to $O(N \cdot log(N)$ time, where N is the maximum among the number of training documents, the number of categories and the number of features. The KNN time complexity is thus rather high.

Rocchio [Ittner *et al.*, 1995; Cohen and Singer, 1999] often refers to TC systems based on the Rocchio's formula for profile estimation (described in Section 3.2.2). An extension of the algorithm was proposed in [Schapire *et al.*, 1998] and [Lam and Ho, 1998] but both approaches relevantly increase the complexity of the basic model.

PRC [Moschitti, 2003c] is the parameterized version of the Rocchio classifier. It will be presented in Section 3.5 to give an example of the positive impact of a correct parameterization.

RIPPER [Cohen and Singer, 1999] uses an extended profile notion based on co-occurrences and multiwords. A machine learning algorithm allows the contexts (e.g. a windows of n words) of a word w to decide how (or whether) the presence/absence of w contribute actually to the target document classification. As it is based on profiles, it can be very fast in on line classification task, but it has a noticeable learning time. Moreover, given the complexity to derive the suitable multiwords, it is not clear if it can be applied to millions of documents.

CLASSI is a system that uses a neural network-based approach to text categorization [Ng *et al.*, 1997]. The basic units of the network are the perceptrons. Given the amount of data involved in typical operational scenarios the size of the target network makes the training and classification complexity prohibitive.

Dtree [Quinlan, 1986] is a system based on a well-known machine learning method (i.e. decision trees) applied to training data for the automatic derivation of a *classification tree*. The Dtree model selects the relevant words (i.e. features) via an information gain criterion and predicts the target document's categories according to word combinations (see Section 2.1.1 for more details). It efficiently supports on line classification as the category assignment time is proportional to the time required to visit the decision tree.

CHARADE [Moulinier *et al.*, 1996] and SWAP1 [Apté *et al.*, 1994] use machine learning algorithms to inductively extract Disjunctive Normal Form rules from the training documents. Sleeping Experts (EXPERTS) [Cohen and Singer, 1999] are learning algorithms that work on-line. They reduce the computation complexity of the training phase for large applications by updating incrementally the weights of n-gram phrases. The reduced complexity makes them appealing for a real application but their accuracy is far away from the *state-of-the-art*.

Naive Bayes [Tzeras and Artman, 1993] is a probabilistic classifier which uses joint probabilities of words and categories to estimate the conditional probabilities of categories given a document. The naive approach refers to the assumption of word independence. Such assumption makes the computation of the Naive Bayes classifier much more efficient than the exponential complexity of a pure Bayesian approach (i.e. where predictors are made of word combinations). In this case the only problem is its low classification accuracy on every corpus.

In order to establish which classification model is more accurate several referring TC benchmarks have been developed. The most famous one is the Reuters corpus. In particular, previous work has shown (e.g. [Yang, 1999]) that five Reuters versions exist and TC systems perform differently on them. In particular, Table 1.1 reports the performance of the above TC systems on Reuters 22173 or Reuters 21578. Both of these versions provide two splits between training and testing: Apté and Lewis modalities [Sebastiani, 2002]. It is worth noting that the same classifier can achieve different accuracy on different Reuters versions/splits. Thus, Table 1.1 provides only an indicative accuracy[3] comparisons of TC models[4].

Table 1.1 shows that the best figure on the Reuters corpus is obtained by the example-driven KNN classifier (82.3/85%) and by $SVMs$ (86%). Unfortunately, they have a heavier training and classification complexity, which makes their use more difficult within real operational domains thus PRC seems more suitable. Other classifiers having a fast on line classification (e.g. RIPPER, SWAP-1) are based on complex learning whereas the others show lower accuracy.

[3] More precisely, the accuracy measurements used are the microaverage BEP and the microaverage f-measure, which will be defined in Section 3.4.2.

[4] Moreover, the same model is subject to several implementations or enhancements. For example, Yang reports two Naive Bayes $BEPs$: 71% [Yang, 1999] vs. 79.56% [Yang and Liu, 1999].

Table 1.1: Breakeven points of widely known classifiers on Reuters corpus

SVM	*KNN*	**PRC**	RIPPER	CLASSI	Naive Bayes
86%	85/82.3 %	**82.83%**	81/82%	80.2%	71/79.56%
SWAP1	CHARADE	EXPERT	*Rocchio*	Dtree	
79/80.5%	73.8/78.3%	75.2/82.7%	74.8/78.1%	79.4%	

1.3 Book Outline

This book aims to provide the reader with the technology suitable to design and implement moderns automated TC systems. In particular, such technology is based on the statistical learning theory which proposes the automatic design of classification function from examples. Other machine learning models have been developed but the proposed theory represents the current state-of-the-art on TC. Moreover, practical procedures often applied in the TC design are thoroughly described. These include the TC tuning phase and accuracy evaluation. Finally, some advanced topics in TC such as the use of complex document representations and interesting applications are illustrated.

More in detail, the book is organized as follows:

- Chapter 2 provides the basic notions of machine learning along with latest theoretical models developed in recent years. Traditional and simple algorithms based on probability theory such as the Naive Bayes and the decision tree classifiers are described. Then, the PAC learning theory is introduced as a general framework of the modern statistical learning theory. This along with the simple perceptron learning algorithm allows the reader to understand the basic ideas of Support Vector Machines, which, together with kernel methods, are the ultimate contribution of statistical learning theory.

- Chapter 3 describes the typical steps for designing a text classifier. In particular, several weighting schemes and the design of profile-based classifiers are shown in detail. Additionally, the learning and classification algorithms for Rocchio and Support Vector Machines have been comparatively analyzed. The important contributions of this chapter relates to the definition of the Parameterized Rocchio text Classifier (PRC)

and the performance evaluation over different corpora as they show to the reader practical procedures that should be followed in the design of text classifiers.

- Chapter 4 reports advanced TC topics by presenting some studies on the use of Natural Language Processing to extract advanced linguistic features for document representation. These may be divided in two main types: (a) those that use syntactic information, e.g., POS-tags and phrases and (b) those based on semantic information, i.e. word senses. Additionally, proposals on the advanced use of TC for interesting natural language applications, i.e., Information Extraction, Question Answering and Text Summarization are described.

The technical content of the different chapters require a lengthy use of basic notions of linear algebra and geometry with specific formalisms. The final appendixes offer a reference dictionary and a short mathematical compendium to help the reader.

Chapter 2

Statistical Machine Learning

In this chapter, we provide the basic notions of machine learning along with latest theoretical results obtained in recent years [Vapnik, 1995; Cristianini and Shawe-Taylor, 2000].

First, we show traditional and simple algorithms based on probability notions such as the Naive Bayes and the decision tree classifiers. Second, we introduce the PAC learning theory and the Perceptron algorithm to provide the readers with essential concepts of the modern statistical learning theory. Finally, we use the above concepts to illustrate a simplified theory of Support Vector Machines, which, along with the kernel methods are the ultimate product of the statistical learning theory.

2.1 What is Machine Learning?

In high school, during the mathematic or statistic classes, we have encountered techniques that, given a set of points, e.g. $\vec{x}_i$ and the values associated with them, i.e. y_i, attempt to derive the functions that best interpolates the relation $\phi(\vec{x}_i, y)$. For example, linear or polynomial regression as shown in Figure 2.1. These are the first examples of machine learning algorithms. When the output values of the target function are finite and discrete, we consider the regression problem as a classification and this is very interesting for the application on real scenarios like categorization of documents in different subjects.

Before introducing more advanced machine learning techniques, it may be convenient to provide an example which illustrates for what it can be useful (in a computer science domain). Let us suppose that a programmer is commis-

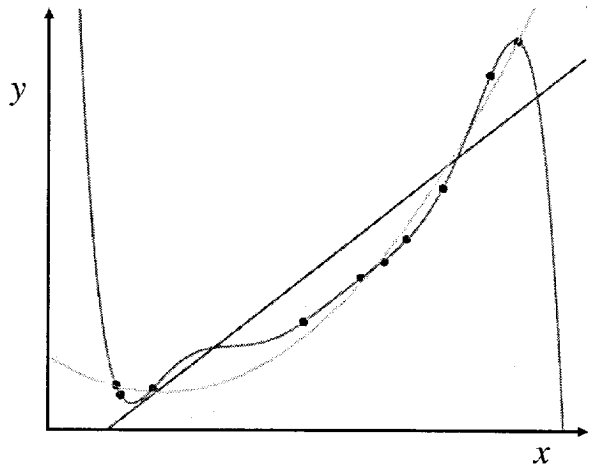

Figure 2.1: Polynomial interpolation of as set of points $<x_i, y_i>$.

sioned to write the following program: given some employee characteristics and a pre-defined employee level hierarchy, assign to each new employee the adequate entry level, automatically. Suppose that the rules to determine such level depends on many variables, e.g. achieved diplomas, previous working experiences, age and so on. Additionally, there is no formal document that explains how to produce such rule set. Let us say that the commissioning company uses such level information only as a way to propose tasks to employees thus the level were heuristically assigned by the human resource director by using an informal algorithm.

The poor programmer would be soon on troubles as to extract algorithmic information from people not used to think in terms of procedures and instructions is in general a challenging task. What could be the solution?

We observe that, there is a lot of data about the link between variables (i.e. the employees) and the output of the target function (i.e. the entry level). The company maintain the data of employees along with their entry levels, thus the programmer should study the data and try to hand-craft the rules from it. However, if the number of employees and the number of their characteristics are large, this may result in a very time consuming and boring task.

Machines have traditionally been built to perform such kind of job, thus,

the solution to the programmer problem should be to write an algorithm which learns from examples the employee classification rules, automatically. This kind of algorithms are a special class of *machine learning methods* called example-driven or inductive learning models. They are standard in the sense that they can be applied to all problems in which there are some data examples and we need a classification function as output.

Given such tools, our not so unlucky programmer should only re-write the examples from the employee database in an input format suitable for the machine learning algorithm and run it to derive the classification function. Such function unlikely will provide a correct entry level in all cases but if the commissioning company (as in this case) accepts an error rate in this procedure, the application of a machine learning approach will be a feasible alternative to the hand-coding. Indeed, another output of the learning process is usually the expected error rate. This value can be estimated by measuring the number of classification mistakes that the classification function commits on a set of employee data (test set) not used for training.

We have introduced what learning models may offer to the solution of real problems. In the next section, we show two simple ML approaches based on Decision Trees and naive probabilistic models.

2.1.1 Decision Trees

From the introduction, we have understood that ML models derive a classification function from a set of training examples (e.g. the employee data) already categorized in the target class (e.g. the entry level). The input for the ML program is the set of examples encoded in a meaningful way with respect to the classification task, i.e. the level assignment. The items describing the individual examples are usually called features and they describe important aspects about the classification objects, e.g. the employees. For instance, the study title is a relevant feature for the entry level whereas the preferred employee food is not relevant thus it should not be included in the example description.

The idea of decision tree classifier (DT) is inspired by a simple principle: the feature that separates the highest number of training examples should be used before the others. To simplify such idea, suppose that we have only two levels (0 and 1) and also the features are binary (e.g. the employee has or not a master degrees). Intuitively, the decision tree algorithm finds the feature which splits the training set S in two subsets S_0 and S_1 such that the proportion of employees of level 0 is *higher* in S_0 than in S whereas the proportion of

employees of level 1 is higher in S_1 than in S. This means that guessing about the employee level in the two new sets is *easier* than in S. As we cannot hope to correctly separate all data with only one feature the algorithm will iteratively find other features that *best* separates S_0 and S_1.

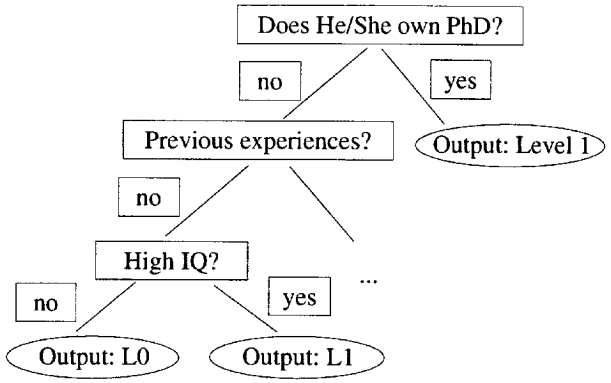

Figure 2.2: Decision tree generated for the classification task of two employee levels.

Figure 2.2 illustrates the decision tree which a DT algorithm may generate. First, the *PhD* attribute is tested. In case the employee owns it the level is surely 1. Second, features such as *Previous Experiences* and *Intelligent Quotient* are tested. Finally, the tests on the leaves should output the final classification in case it had not been output on the internal nodes.

In order to find the most discriminative feature, DTs use the *information gain*[1] quantity. In the general case, we have a set of classes $\{C_1, .., C_m\}$ distributed in the training set S with the probabilities $P(C_i)$, then the entropy H of P is the following:

$$H(P) = \sum_{i=1}^{m} -P(C_i)log_2(P(C_i)) \tag{2.1}$$

Suppose to select a feature f which assumes $\{a_1, .., a_n\}$ values in S. If we split the training examples according to f, we obtain n different subsets, i.e.

[1]Also known as Mutual Information.

$\{S_1, .., S_n\}$. The average entropy is thus:

$$\bar{H}(P^{S_1}, .., P^{S_n}) = \sum_{i=1}^{m} \frac{H(P^{S_i})}{|S_i|} \tag{2.2}$$

where, P^{S_i} is the probability distribution of C_i on the S_i set and $H(P^{S_i})$ is the related entropy.

The DT algorithm evaluates the Eq. 2.2 for each feature and selects the one which has associated the highest value. Such approach uses the probability theory to select the most informative features and generate the tree of decisions. In the next section, we show another machine learning approach in which the probability theory is more explicitly applied to design the decision function.

2.1.2 Naive Bayes

We have seen that machine learning approaches are useful when the information about the target classification function (e.g. the commissioned program) is complex, not explicitly available and not completely accurate. Such aspects determine a degree of uncertainty on the final results which is manifested as the error rate.

Given the random nature of the expected results, the probability theory seems to be well suited for design of a classification function which aims to achieve the highest probability of correct results. Indeed, we can model the output of our target function as the probability to categorize an instance in a target class, given the training data.

More formally, let us indicate with E the classification example and let $\{C_1, .., C_m\}$ be the set of categories in which we want to classify such example. We are interested to evaluate the probability that E belongs to C_i, i.e. $P(C_i|E)$. In other words, we know the classifying example and we need to know its category. Our example E can be represented as a set of features $\{f_1, .., f_n\}$ but we do not know how to relate $P(C_i|f_1, .., f_n)$ to the training examples. Thus, the idea is to use the Bayes' rule to derive a more useful probability form:

$$P(C_i|f_1, .., f_n) = \frac{P(f_1, .., f_n|C_i) \times P(C_i)}{P(f_1, .., f_n)}, \tag{2.3}$$

where

$$\sum_{i=1}^{m} P(C_i|f_1,..,f_n) = \sum_{i=1}^{m} \frac{P(f_1,..,f_n|C_i) \times P(C_i)}{P(f_1,..,f_n)} = 1$$

for definition of probability.

We will choose for the example E the category C_i associated with the maximum $P(C_i|E)$. To evaluate such probabilities, we need to select a category i and count the number of examples that contain the $f_1,..,f_n$ features. Considering that a real scenario training set may contain no more than 10,000 examples, we will unlikely be able to derive reliable statistics as n binary features determine 2^n different examples[2]. Thus, to make practical the Bayesian approach we naively assume that features are independent. Given such assumption, Eq. 2.3 can be rewritten as:

$$P(C_i|f_1,..,f_n) = \prod_{k=1}^{n} \frac{P(f_k|C_i) \times P(C_i)}{P(f_1,..,f_n)} \tag{2.4}$$

As $P(f_1,..,f_n)$ is the same for each i, we do not need it to evaluate the maximal probability. The $P(C_i)$ can be computed by simply counting the number of training examples labeled as C_i, i.e. $|C_i|$ and divide it by the total number of examples in all categories:

$$P(C_i) = \frac{|C_i|}{\sum_{j=1}^{m} |C_j|}$$

To estimates $P(f_k|C_i)$, we derive n_{ik}, i.e. the number of examples categorized as C_i that contain the feature f_k and we divide it by the C_i cardinality, i.e.

$$P(f_k|C_i) = \frac{n_{ik}}{|C_i|}$$

As an example of naive Bayesian classification suppose that we divide human beings in thee categories Allergy, Cold and Well, i.e. according to their health status. The features that we use to categorize human status are $f_1 = sneeze$, $f_2 = cough$ and $f_3 = fever$. Suppose also, that by looking into a medical database, where f_1, f_2 and f_3 were annotated for each patient, we derived the probability distribution of Table 2.1.

[2]If we assume uniform distribution with only 20 features we need more than 1 billion of examples to have a chance to see the test example in the training set.

Prob.	Allergy	Cold	Well
$P(C_i)$	0.05	0.05	0.9
$P(sneeze\|C_i)$	0.9	0.9	0.1
$P(cough\|C_i)$	0.7	0.8	0.1
$P(fever\|C_i)$	0.4	0.7	0.01

Table 2.1: Probability distribution of $sneeze$, $cough$ and $fever$ features inside the Allergy, Cold and Well categories.

If we extract from our target patient the following feature representation $E = \{sneeze, cough, \sim fever\}$, where $\sim$ stands for not $fever$, we can evaluate the probabilities to belong to each category i:

- $P(\text{Allergy}|E) = (0.05)(0.9)(0.7)(0.6)/\text{P(E)}=0.019/P(E)$
- $P(\text{Cold}|E) = (0.05)(0.9)(0.8)(0.3)/P(E) = 0.01/P(E)$
- $P(\text{Well}|E) = (0.9)(0.1)(0.1)(0.99)/P(E) = 0.0089/P(E)$

Thus, the patient should be affected by allergy.

It is worth to note that such probabilities depend on the product of the probabilities of each feature. It may occur, especially when the training corpus is too small, that some of them never appear in some categories; as a consequence, the estimation of the probability of a feature f in the category C_i, $P(f|C_i)$, will be 0. This causes the product of Eq. 2.4 of a category i to be 0, although the contributions of the other features in the product may be high. Assigning a probability equal 0 to a feature is, in general, a rough approximation as the real probability is just too small to be observed in the training data, i.e. we do not have enough data to find an occurrence of f.

To solve the above problem, smoothing techniques are applied. The idea is to give to the features which do not appear in the training data a small probability α. To maintain the overall feature distribution consistent the other features will release a small portion β of their probability such that the overall summation is still 1.

The simplest technique is referred to as the Laplace smoothing. The new feature probability is the following:

$$P(f_k|C_i) = \frac{n_{ik} + a \times p_k}{|C_i| + a}$$

where p_k is a probability distribution and a is the size of a hypothetical set of examples where we imagine to have observed p_k. When we do not know any information about the not observed features, it is logical to assume a uniform distribution, i.e. $p_k = 1/a$ therefore $a = n$ and

$$P(f_k|C_i) = \frac{n_{ik} + 1}{|C_i| + n}.$$

The smoothing techniques improve the Naive Bayes model by providing a better estimation of the probability of the features not observed in the data. However, the independence assumption seems a serious limitation to the accuracy reachable by such approach. Moreover, we would like to have a clear correlation between the number of training instances and the final classifier accuracy. At this purpose, the next section illustrates some results of the statistical learning theory that allows us to link the error probability of a class of learning functions to the number of training examples. The members of such class are called *probability approximately correct* (PAC) functions.

Exercize 2.1 *Classify using a Naive Bayes learning algorithm and the probabilities in Table 2.1 all 8 possible examples, e.g.* $\{sneeze, cough, \sim fever\}$, $\{sneeze, \sim cough, fever\}$,...

Exercize 2.2 *Modify the probabilities in Table 2.1 to classify e.g.* $\{sneeze, cough, \sim fever\}$ *in class Cold with a Naive Bayes classifier.*

Exercize 2.3 *Define a new learning application and apply the Naive Bayes algorithm to it.*

2.2 PAC Learning

We have seen two different ML approaches, i.e. DT and Naive Bayes, with both, we can use some available training examples to learn the classification functions and estimate their accuracy on a test set of new examples. Intuitively, we may think that as the number of training examples increases the accuracy increases as well. Unfortunately, this is not generally true. For example, if we want to learn the difference between Allergy and Cold categories using only the $sneeze$ and $cough$ features, we will never reach high accuracy, no matter how many training examples we have available. This happens because such features do not clearly separate the two classes.

Given such problems, we need some analytical results that helps us to determine (1) if our learning function is *adequate* for the target learning problem and (2) the probability of errors according to the number of available training examples. The class of functions for which we have such analytical data is called the probability approximately correct class.

The statistical learning theory provides mathematical tool to determine if a class of functions is PAC learnable. At the base of such result there is a new statistical quantity designed by two scientists, Vapnik and Chervonenkis, called VC-dimension. This gives a measure of the learning complexity and can be used to estimate the classification error.

In the next sections, we formally define the PAC function class, provide a practical example to derive the error probability of PAC functions and introduce the VC-dimension which automatizes the estimation of such error.

2.2.1 Formal PAC definition

The aim of ML is to learn some functions from a set (Tr) of training examples. These latter can be seen as data points that are associated with some discrete values $C = \{C_1, .., C_n\}$ in case of classification problems or real number $\mathbb{R}$ in case of regression problem. We focus only in the classification problem, i.e. to find a function $f : X \rightarrow C$ by using $Tr \in X$. In general, the training examples are extracted randomly thus we need to deal with a probability distribution D on X.

The function f can be learned by using an algorithm which can generate only a small subset of all possible functions. The algorithm will derive from the examples a function $h \in H$, where H is the class of all possible hypothesis (functions) derivable by it. This suggests that h will hardly be equal to f. Indeed, it is very useful to define a measure of the error of h.

A reasonable measure is the percentage of points for which f and h differ, i.e. the probability that given an example x, $P[f(x) \neq h(x)]$. Note that D is particularly important. As a trivial example, if the probability of an element x_0, $D(x_0)$, is 1 and $f(x_0) = h(x_0)$, the error rate will be 0, independent of the number of $x \in Tr$ such that $f(x) \neq h(x)$.

The above case is very rare and does not occur in practical situations. On the contrary, there is a large class of functions whose error decreases as the number of training examples increases. These constitute the PAC learnable functions. Their formal definition is the following:

- Let the function $f : X \rightarrow C$ belongs to the class F, i.e. $f \in F$, where X is the domain and C is the codomain of f.
- Suppose that the training and the test documents $x \in X$ are generated with a probability D.
- Let $h \in H$ be the function that we learned from the examples provided that we can learn only functions in H, i.e. the hypothesis space.
- The error of h, $error(h)$, is defined as $P[f(x) \neq h(x)]$, i.e. the percentage of miss-classified examples.
- Let m be the size of the training set, then F is a class of PAC learnable functions if there is a learning algorithm such that:
  - $\forall f \in F, \forall D \in X$ and $\forall \epsilon > 0, \delta < 1$
  - $\exists m$ such that $P[error(h) > \epsilon] < \delta$, i.e. the probability that the h's error is greater than ϵ is lower than δ.

In other words, a class of functions F is PAC learnable if we can find a learning algorithm which, given an enough number of training examples, produces a function h such that its error is greater than ϵ with a probability less than δ. Thus by choosing low values for ϵ and δ, we can have a low error (i.e. $< \epsilon$) with high probability (i.e. $1 - \delta$).

Next section clarifies the above idea with a practical example.

2.2.2 An Example of PAC Learnable Functions

Suppose that we need to learn the concept of medium-built people. Given such problem two very important features are the height and the weight of a person. Only one of these features would not be able to characterize the concept of medium-built body . For example, a person which has a height of 1,75 meters may be seen as medium person but if its weight is 130 Kg we would immediately change our idea.

As the above two features assume real number values, we can represent people on a Cartesian chart, where the X-axis and Y-axis correspond to height and the weight, respectively. Figure 2.3 illustrates such idea.

Given such representation, we may assume that the medium-built person concept c is represented by a rectangle which defines the maximum and minimum weight and height. Suppose that we have available some training examples, i.e. the measures of a set of people which may or may not have a

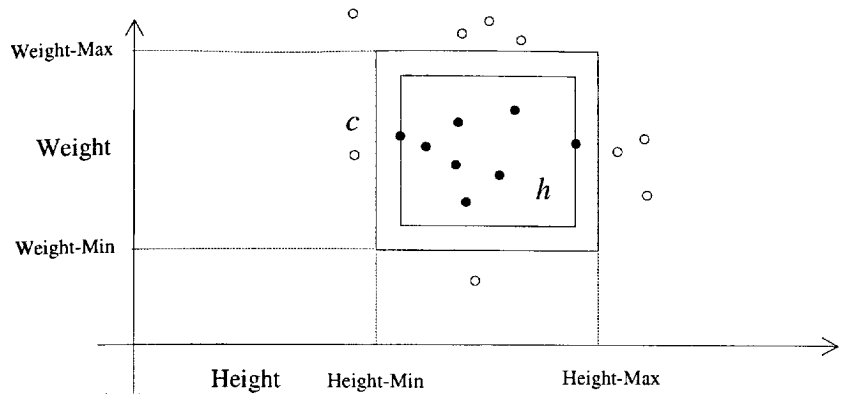

Figure 2.3: The medium-built person concept on a Cartesian chart.

medium-build body, we can represent them in the chart. The white points, those that are outside the rectangle c are not medium-built people all the others (black points) are instead in the class of medium people.

As we assumed that our hypothesis c has a rectangular shape whose edges are parallel to the axises, the ML algorithm should learn h from the rectangle set H, i.e. the set of hypothesis. Suppose that the algorithm outputs the smaller rectangle of Figure 2.3 as the learned hypothesis h. Since the error is defined as $P[f(x) \neq h(x)]$, we can evaluate it by dividing the area between the rectangles c and h by the area of c[3].

Let our learning algorithm be the following:

Select the smallest (axis aligned) rectangle that includes all training examples corresponding to medium-built people.

We would like to verify that this is a PAC algorithm. To do this, we fix an error ϵ, a target probability δ and evaluate the $P[error(h) > \epsilon] < \delta$, i.e. the probability of generating a *bad* h hypothesis. This is equal to the probability that m training examples are classified correctly by a hypothesis h (this happens as our learning algorithm selects only this kind of hypothesis) which has an error higher than ϵ.

Since h is *bad*, it has an error higher than ϵ thus the probability that it

[3]It can be proven that this is true for any distribution D.

classifies correctly one training example is $< 1 - \epsilon$. Given the Cartesian representation of Figure 2.4.A, this means that the rectangle of a *bad* h is included in the smallest rectangle of *good* hypotheses (i.e. the hypothesis of area equal to $1 - \epsilon$).

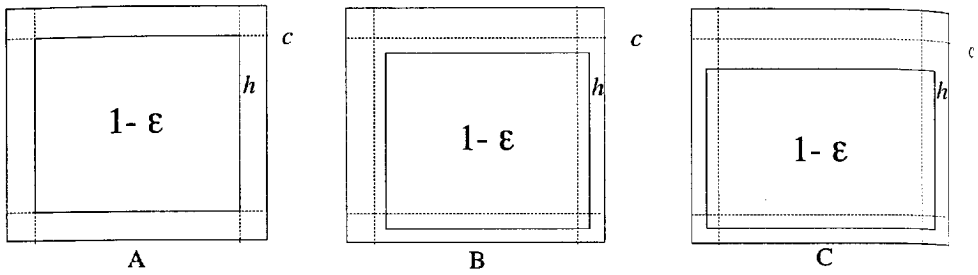

Figure 2.4: Probabilities of *bad* and *good* hypotheses.

Let us consider the 4 strips between c and h. We observe that a *bad* hypothesis cannot contemporary touch all four strips. This is illustrated by the frames B and C. It follows that, a necessary condition to generate a *bad* hypothesis is to keep all the m points at least outside of one of the 4 strips. *Necessary* means that it must happen each time we generate a bad hypothesis, consequently, the probability of m *points out of at least one strip* is higher than a hypothesis to be *bad*. In other words, it is an upperbound of the probability to generate a *bad* hypothesis. To simplify its evaluation, we divide it in the following steps:

1. the probability that a point x is out of one strip, $P(x \text{ out of 1 strip}) = (1 - \epsilon/4)$;
2. the probability that m points are out of one strip, $P(x \text{ out of 1 strip})^m = (1 - \epsilon/4)^m$;
3. the probability that m points are out of 4 strips $< 4P(x \text{ out of 1 strip})^m = 4(1 - \epsilon/4)^m$;

Therefore, $P[error(h) > \epsilon] < 4(1 - \epsilon/4)^m$ and we can find the m corresponding to a fixed δ by imposing $\delta \leq 4(1 - \epsilon/4)^m \Rightarrow$

$$m > \frac{ln(\delta/4)}{ln(1 - \epsilon/4)}$$

From Taylor's succession we know that

$$-ln(1 - y) = y + y^2/2 + y^3/3 + .. \Rightarrow (1 - y) < e^{(-y)}$$

We can apply the above inequality to $ln(1 - \epsilon/4)$ to obtain

$$m > \frac{ln(\delta/4)}{ln(1 - \epsilon/4)} \Rightarrow m > \frac{4ln(4/\delta)}{\epsilon}.$$

This result proves that the medium-built people concept is PAC learnable as we can reduce the error probability as much as we want, provided that we have an enough number of training examples.

It is interesting to note that a general upperbound for PAC functions can be evaluated by considering the following points:

1. the probability that a *bad* hypothesis is consistent with m training examples (i.e. classifies them correctly) is $(1 - \epsilon)^m$;
2. the number of *bad* hypothesis is less than the total number of hypothesis $N \Rightarrow$
3. $P(h \text{ bad and consistent with } m \text{ examples}) = N(1 - \epsilon)^m < Ne^{-\epsilon^m} = Ne^{-m\epsilon} < \delta \Rightarrow$

$$m > \frac{1}{\epsilon}(ln\frac{1}{\delta} + lnN). \tag{2.5}$$

We can use the Eq. 2.5 when N is finite. For example, if we want to learn a Boolean function of n variable, the number N of such functions is $2^{2^n} \Rightarrow m > \frac{1}{\epsilon}(ln\frac{1}{\delta} + 2^n ln2)$

In most of the cases the above bound is not useful and we need to derive the one specific to our target problem as we have done for the medium-built concept. However, when the feature space is larger than 2 the manual procedure may become much more complex. In the next section we will see a characterization of PAC functions via VC dimension which makes more systematic the detection of PAC property.

2.2.3 The VC-dimension

The previous section has shown that some function classes can be learned with any accuracy and we have noted that such property depends on the properties of the adopted learning algorithm. For example, the fact that we use rectangles

as our hypothesis space (the one from which our algorithm selects h) instead of circles or lines impacts on the *learning capacity* of our algorithm.

Indeed, it is easy to show that using lines, we would never have been able to separate medium-built people from the other people whereas the rectangle class is quite effective to do this. Thus, we need a property that allows us to determine which hypothesis class is more appropriate to learn a target function $f \in F$. Moreover, we note that, in most of the cases, we do not know the nature of the target f. We know only the training examples, consequently, our property should be derived only by them and by the function class H that we have available.

The Vapnik and Chervonenkis (VC) dimension aims to characterize functions from a learning point of view. The intuitive idea is that different function classes have different data point separation capacity: some of them can just separate some configurations of points whereas others can separate a much larger number of configurations, i.e. they are in some sense more general purpose. The VC dimension captures this kind of property.

Intuitively, VC dimension, i.e. the learning capacity, impacts on the generalization ability during the learning as suggested by the following statements:

- A function selected from a high class capacity is expected to *easily* separate the training points since it has the capacity to adapt to any training set. This means that the learned function is too specific to such training data (it may overfit data), hence, the probability to separate correctly the test set is lower.

- On the contrary, a function that belongs to a low capacity class can separate a lower number of data configurations thus if it is successful for the current training points, the probability to separate well the test data is higher.

The definition of VC dimension depends on the concept of shattering a set of points.

Def. 2.4 ***Shattered Sets***
Let us consider binary classification functions $f \in F$, $f : X \rightarrow \{0,1\}$. We say that $S \subseteq X$ is shattered by a function class F if $\forall S' \subseteq S$, $\exists f \in F$:

$$f(x) = \begin{cases} 0 \text{ iff } x \in S' \\ 1 \text{ iff } x \in S - S' \end{cases} \tag{2.6}$$

The definition says that a set of points S is shattered by a function class F if for any assignment of the points in S into $\{0,1\}$, we can find $f \in F$ that reproduces such assignments.

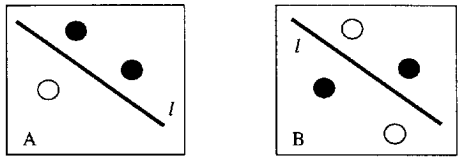

Figure 2.5: VC dimension of lines in a bidimensionale space.

A graphical interpretation is given in Figure 2.5. In the frame A, we have 3 points represented in a two-dimensional space. The class L that we chose is the one of linear functions. For any assignment of points (white is 0 and black is 1), we can find a line $l \in L$ that separates them. From l we can derive the shattering function f by assigning $f(x_1, x_2)$=0 *iff* $x_2 < l(x_1)$ and 1 otherwise, i.e. if the point is under the line, we assign 0 to it and 1 otherwise. Consequently, a set of tree points can be shattered by linear functions.

On the contrary, in the frame B, we can see 4 points which cannot be shattered. More precisely, there are not 4 points that can be shattered by linear functions since we can always draw a tetragon by using 4 points and assign the same color to the opposite vertices. If the line assigned the same color to the opposite vertices there will always be a vertex on the same side of such two points which has a different color.

Def. 2.5 ***VC dimension***
The VC dimension of a function class F is the maximum number of points that can be shattered by F.

Since Figure 2.5.A shows a set of tree points shattered by a linear function, such class has at least a VC dimension of 3 in the bidimensional space. We have also proved that 4 points cannot be shattered, consequently, the VC dimension of linear functions on the plane is exactly 3. It is important to note that, selecting linear dependent points, i.e. they lie on the same lines, is not a very good idea as we cannot hope to shatter three points in which those external have the same color and the internal ones has a different color.

In particular it can be proven (see [Burges, 1998]) the following:

Theorem 2.6 *Consider some set of m points in $\mathbb{R}^n$. Choose any one of the points as origin. Then the m points can be shattered by oriented hyperplanes if and only if the position vectors of the remaining points are linearly independent.*

As a consequence we have the

Corollary 2.7 *The VC dimension of the set of oriented hyperplanes in $\mathbb{R}^n$ is n+1, since we can always choose $n+1$ points, and then choose one of the points as origin, such that the position vectors of the remaining n points are linearly independent, but can never choose $n+2$ such points (since no set of $n+1$ vectors in $\mathbb{R}^n$ can be linearly independent).*

This Corollary is useful to determine the VC dimension of linear functions in an n dimensional space. Linear functions are the building block of Support Vector Machines, nevertheless, there are other examples of classifiers which have different VC dimension such as the rectangle class. The following example is useful to understand how to evaluate the VC dimension of geometric classifiers.

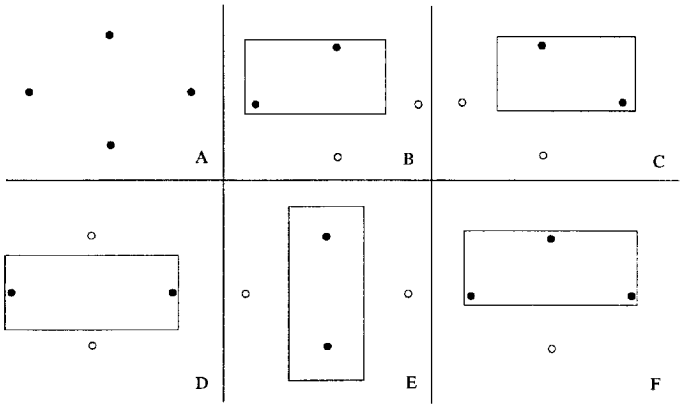

Figure 2.6: VC dimension of (axis aligned) rectangles.

Example 2.8 ***The VC dimension of (axis aligned) rectangles***
To evaluate the VC dimension of rectangles, we (1) make a guess about its value, for instance 4, (2) show that 4 points can be shattered by rectangles and (3) prove that no set of 5 points can be shattered.

Let us choose 4 points that are not aligned like in Figure 2.6.A. Then, we give all possible assignments to the 4 points. For example, Figure 2.6.B shows two pairs of adjacent points which have the same color. In Section 2.2.2, we established that points inside the rectangle belongs to medium-built people, i.e. they are positive examples of such class. Without loss of generality we can maintain such choice and use the black color to indicate that the examples are positive (or that they are assigned to 1). The only relevant aspect is that we need to be consistent with such choice for all assignments, i.e. we cannot change our classification algorithm while we are testing it on the point configurations.

From the above convention, it follows that given the assignments B, C, D, E and F Figure 2.6, we need to find the rectangles that contain only black points and leave the white points outside. In the Figure 2.6, the rectangles C, D, E, F that separate half positive and half negative examples are shown.

It is worth to note that if we have 3 positive (or 3 negative) examples, finding the shattering rectangles is straight forward (see Frame E), consequently, we have proven that the VC dimension is at least 4.

To prove that is not greater than 4, let us consider a general 5 point set. We can create 4 different rankings of the points by sorting in ascending and descending order by their x-cordinate and by their y-cordinate. Then, we color the top point of each of the 4 lists in black and the 5th point in white. This latter will be included (by construction) in the rectangle of the selected 4 vertices. Since any rectangular hypothesis h that contains the four points must contain the previous rectangle, we cannot hope to not include the 5th point in h. Consequently, no set of 5 points can be shattered by the rectagle class.

Finally, we report two theorems on the sample complexity which derive an upper and lower bounds of the number of training examples and one theorem on the probability error of a hypothesis given the VC dimension of its class. These theorems make clear the link between VC dimension and PAC learning.

Theorem 2.9 *(upper bound on sample complexity, [Blumer* et al., *1989])*
Let H and F be two function classes such that $F \subseteq H$ and let A an algorithm that derives a function $h \in H$ consistent with m training examples. Then, $\exists c_0$ such that $\forall f \in F$, $\forall D$ distribution, $\forall \epsilon > 0$ and $\delta < 1$ if

$$m > \frac{c_0}{\epsilon}\left(VC(H) \times ln\frac{1}{\epsilon} + \frac{1}{\delta}\right)$$

then with a probability $1-\delta$,

$$error_D(h) \leq \epsilon,$$

where VC(H) is the VC dimension of H *and* $error_D(h)$ *is the error of* h *according to the data distribution* D.

Theorem 2.10 *(lower bound on sample complexity, [Blumer* et al., *1989])*
To learn a concept class F *whose VC-dimension is* d, *any PAC algorithm requires* $m = O(\frac{1}{\epsilon}(\frac{1}{\delta} + d))$ *examples.*

Theorem 2.11 *(Vapnik and Chervonenkis, [Vapnik, 1995])*
Let H *be a hypothesis space having VC dimension* d. *For any probability distribution* D *on* $X \times \{-1, 1\}$, *with probability* $1-\delta$ *over* m *random examples* S, *any hypothesis* $h \in H$ *that is consistent with* S *has error no more than*

$$error(h) \leq \epsilon(m, H, \delta) = \frac{2}{m}(d \times ln\frac{2e \times m}{d} + ln\frac{2}{\delta}),$$

provided that $d \leq m$ *and* $m \geq 2/\epsilon$.

Exercize 2.12 *Compare the upper bounds on sample complexity of rectangles derived in Section 2.2.2 with the one derivable from theorem 2.9.*

Exercize 2.13 *Evaluate the VC dimensions of triangles aligned and not aligned to the axis.*

Exercize 2.14 *Evaluate the VC dimension of circles.*

2.3 The Support Vector Machines

In the previous sections, we have seen that classification instances can be represented with numerical features and these in turn can be associated with points of an n-dimensional space. If our goal is to learn binary classification functions, lines or hyperplanes are interesting classifiers as it is easy to treat them analytically. Indeed, they constitute the basic building block of the theory of advanced learning methods called Support Vector Machines ($SVMs$).

Given the complexity of these learning models, we first introduce the Perceptron algorithm which can be considered the simplest $SVMs$ and then we define the theory and algorithm of advanced $SVMs$. One of their important

properties is the possibility to use kernel functions to deal with non linear classification problems. Thus, a conclusive section will introduce the kernel theory and its application to advanced learning tasks, e.g. the classification of syntactic-parse trees.

2.3.1 Perceptrons

In Section 2.2.3, we have seen that linear functions can implement binary classifiers where the learning algorithm simply finds a line that separates the two target classes. One advantage of such mathematical objects is their simplicity that allows us to design efficient learning algorithms, i.e. efficient approaches to find the separating line or hyperplane in high dimensional spaces.

The reader may wonder if such simplicity limits the ability of the learning algorithms that we can design. In other words, if we used only linear functions could we learn any possible *learnable function*? It is clear that with only one hyperplane, we cannot learn any function. For example, Figure 2.5 shows in the Frame B four points that cannot be separated. However, this is not a definitive limitation of linear functions as:

1. By modeling our learning problem more effectively, i.e. by choosing more appropriate features, the target problem could become linearly separable. For example, a four point configuration can be shattered in a three-dimensional space, thus we need just to add a significant feature to solve the problem of Figure 2.5.B.

2. We can use linear functions in cascade. The resulting function is more expressive and, depending on the number of levels in such cascade, we can design any function.

The thesis that linear function combinations are sufficient to derive any learnable relation from examples is supported by the observation that human beings' brain is structured with such sort of devices.

Figure 2.7 shows one animal neuron. It is constituted by one set of inputs, i.e. the dendrites, which are connected to a cellular body, i.e. soma, via synapses. Synapses are able to amplify or attenuate the signal. The neuron output is the axon whose filaments are connected to the dendrites of other neurons. When a chemical signal is transmitted on the dendrites, it is amplified by the synapses before entering into the soma. If the overall signal, coming

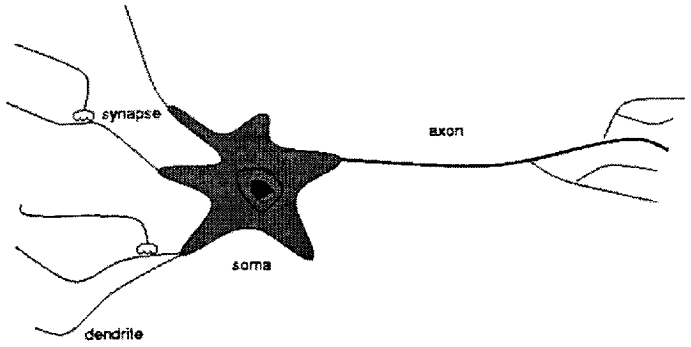

Figure 2.7: An animal neuron.

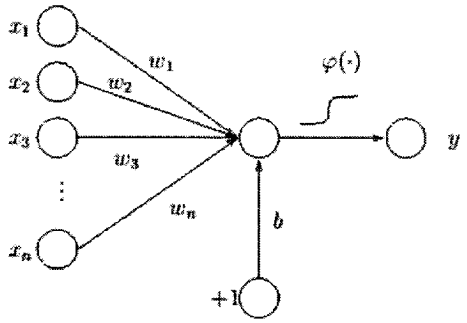

Figure 2.8: An artificial neuron.

from different synapses, overcomes a certain threshold, the soma will transmit a signal to the other neurons by means of the axon.

The artificial version of the neuron is often referred to as *Perceptron* and can be sketched as in Figure 2.8. Each dendrite is an input x_i associated with a weight w_i. The product between the weights and the input signals are summed together and if such summation overcomes the threshold b the output y will be 1, otherwise it will be 0. The interesting aspect is that the output of such neuron can be modeled with a simple hyperplane whose equation is:

$$y = w_1 x_1 + .. + w_n x_n + b = \vec{w} \cdot \vec{x} + b = 0 \tag{2.7}$$

where the final perceptron classification function output is obtained by applying the signum function to y, i.e.

$$f(\vec{x}) = sgn(\vec{w} \cdot \vec{x} + b) \tag{2.8}$$

Eq. 2.8 shows that linear functions are equivalent to neurons which, combined together, constitute the most complex learning device that we know, i.e. the human brain. The signum function simply divides the data points in two sets: those that are over and those that are below the hyperplane. The major advantage of using linear functions is that given a set of training points, $\{\vec{x}_1, .., \vec{x}_m\}$, each one associated with a classification label y_i (i.e. $+1$ or -1), we can apply a learning algorithm that derives the vector $\vec{w}$ and the scalar b of a separating hyperplane, provided that at least one exists.

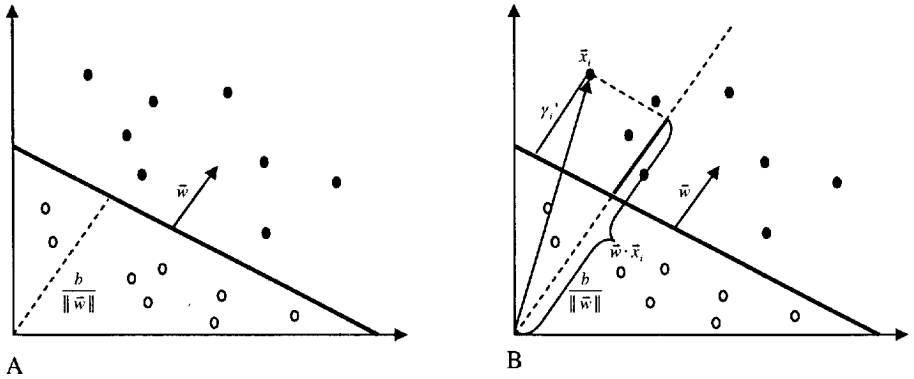

Figure 2.9: Separating hyperplane and geometric margin.

For example, Figure 2.9.A shows a set of training points (black positives and white negatives) along with a separating hyperplane in a 2-dimensional space. The vector $\vec{w}$ and the scalar $-b/||\vec{w}||$ are the gradient vector and the distance from the origin to such hyperplane, respectively. In fact, from Eq. 2.7, $-b = \vec{w} \cdot \vec{x}$ thus $-b/||\vec{w}|| = \vec{w}/||\vec{w}|| \cdot \vec{x}$, where $\vec{x}$ is any point lying on the hyperplane and $\vec{w}/||\vec{w}|| \cdot \vec{x}$ is the projection of $\vec{x}$ on the gradient (i.e. the normal to the hyperplane).

The perceptron learning algorithm exploits the above properties along with the concept of margin and geometric margin.

Def. 2.15 *The **functional margin** γ_i of an example $\vec{x}_i$ with respect to a hyperplane $\vec{w} \cdot \vec{x} + b = 0$ is the product $y_i(\vec{w} \cdot \vec{x}_i + b)$.*

Def. 2.16 *The **geometric margin** γ_i' of an example $\vec{x}_i$ with respect to a hyperplane $\vec{w} \cdot \vec{x} + b = 0$ is $y_i(\frac{\vec{w}}{||\vec{w}||} \cdot \vec{x}_i + \frac{b}{||\vec{w}||})$.*

In Figure 2.9.B, it is immediate to see that the geometric margin γ_i' is the distance of the point $\vec{x}_i$ from the hyperplane as:

- $\frac{\vec{w}}{||\vec{w}||} \cdot \vec{x}_i$ is the projection of $\vec{x}_i$ on the line passing for the origin and parallel to $\vec{w}$;
- to the above quantity is subtracted the distance of the hyperplane from the origin, i.e. $\frac{b}{||\vec{w}||}$. It follows that we obtain the distance of $\vec{x}$ from the hyperplane.
- When the example $\vec{x}$ is negative, it is located under the hyperplane thus the product $\vec{w} \cdot \vec{x}_i$ is negative. If we multiply such quantity by the label y_i (i.e. -1), we make it positive, i.e. we obtain a distance.

function Perceptron(training-point set: $\{\vec{x}_1, .., \vec{x}_m\}$)
begin
  $\vec{w}_0 = \vec{0}$; $b_0 = 0$; $k = 0$;
  $R = \max_{1 \leq i \leq m} ||\vec{x}_i||$
  repeat
    n_errors = 1;
    for ($i = 1$ **to** m)
      if $y_i(\vec{w}_k \cdot \vec{x}_i + b_k) \leq 0$ **then**
        $\vec{w}_{k+1} = \vec{w}_k + \eta y_i \vec{x}_i$;
        $b_{k+1} = b_k + \eta y_i R^2$;
        $k = k + 1$;n_errors = 0;
      end(if)
  until n_errors;
  return k, $\vec{w}_k$ and b_k ;
end

Table 2.2: Rosenblatts perceptron algorithm.

Given the above geometric concepts, the algorithm for perceptron learning in Table 2.2, results very clear. At step k=0, $\vec{w}$ and b are set to 0, i.e. $\vec{w}_0 = \vec{0}$ and $b_0 = 0$, whereas R is set to the maximum length of the training set vectors, i.e. the maximum among the distances of the training points from the origin. Then, for each $\vec{x}_i$ the functional margin $y_i(\vec{w}_k \cdot \vec{x}_i + b_k)$ is evaluated. If it is negative it means that $\vec{x}_i$ is not correctly classified by the hyperplane, i.e. y_i disagrees with the point position with respect to the hyperplane. In this case, we need to correct the hyperplane to classify the example correctly. This can be done by rotating the current hyperplane (i.e. by summing $\eta y_i \vec{x}_i$ to $\vec{w}_k$) as shown in the charts A and B of Figure 2.10 and by translating the hyperplane of a quantity $\eta y_i R^2$ as shown in the chart C.

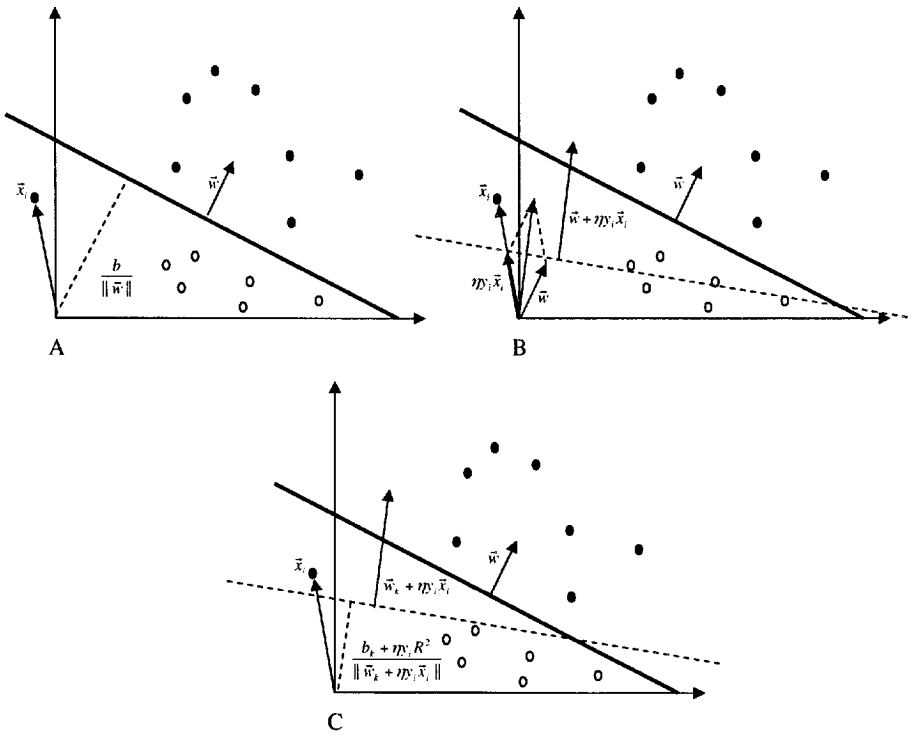

Figure 2.10: Perceptron algorithm process.

The perceptron algorithm always converges when the data points are lin-

early separable as stated by the following

Theorem 2.17 *(Novikoff) Let S be a non-trivial training and let $\gamma > 0$ and $R = max_{1 \leq i \leq m} ||\vec{x}_i||$. Suppose that there exists a vector $\vec{w}_{opt}$ such that $||\vec{w}_{opt}|| = 1$ and $y_i(\vec{w}_{opt} \cdot \vec{x}_i + b_{opt}) \geq \gamma \; \forall i = 1, .., m$. Then the number of mistakes made by the perceptron algorithm on S is at most $(\frac{2R}{\gamma})$.*

This theorem proves that the algorithm converges in a finite number of iterations bounded by $(\frac{2R}{\gamma})$ provided that a separating hyperplane exists. In particular:

- the condition $\vec{w}_{opt}$ such that $||\vec{w}_{opt}|| = 1$ tells that we are considering normalized vectors, i.e. $\vec{w}_{opt} = \frac{\vec{w}_{opt}}{||\vec{w}_{opt}||}$, thus the functional margin is equal to the geometric margin.
- $y_i(\vec{w}_{opt} \cdot \vec{x}_i + b_{opt}) \geq \gamma$ is equivalent to state that for such hyperplane the geometric margin of the data points are $\geq \gamma > 0$, i.e. any point is correctly classified by the $\vec{w}_{opt} \cdot \vec{x} + b_{opt} = 0$ hyperplane.

If the training data is not separable then the algorithm will oscillate indefinitely correcting at each step some misclassified example.

An interesting property showed by the Novikoff theorem is that the gradient $\vec{w}$ is obtained by adding vectors proportional to the examples $\vec{x}_i$ to $\vec{0}$. This means that $\vec{w}$ can be written as a linear combination of training points, i.e.

$$\vec{w} = \sum_{i=1}^{m} \alpha_i y_i \vec{x}_i \tag{2.9}$$

Since the sign of the contribution $\vec{x}_i$ is given by y_i, α_i is positive and is proportional (through the η factor) to the number of times that $\vec{x}_i$ is incorrectly classified. *Difficult points* that causes many mistakes will be associated with large α_i.

It is interesting to note that, if we fix the training set S, we can use the α_i as alternative coordinates of a dual space to represent the target hypothesis associated with $\vec{w}$. The resulting decision function is the following:

$$h(x) = sgn(\vec{w} \cdot \vec{x} + b) = sgn\left(\left(\sum_{i=1}^{m} \alpha_i y_i \vec{x}_i\right) \cdot \vec{x} + b\right) =$$

$$= sgn\left(\sum_{i=1}^{m} \alpha_i y_i (\vec{x}_i \cdot \vec{x}) + b\right) \tag{2.10}$$


```
function Perceptron(training-point set: {x_1, .., x_m})
begin
  α = 0; b_0 = 0;
  R = max_{1≤i≤m} ||x_i||
  repeat
    n_errors = 1;
    for (i = 1 to m)
      if y_i( Σ_{j=1}^{m} α_j y_j (x_j · x) + b) ≤ 0 then
        α_i = α_i + 1;
        b = b + y_i R^2;
        n_errors = 0;
      end(if)
  until n_errors;
  return α and b ;
end
```

Table 2.3: Dual perceptron algorithm.

Given the dual representation, we can adopt a learning algorithm that works in the dual space described in Table 2.3.

Note that as the Novikoff's theorem states that the learning rate η only changes the scaling of the hyperplanes, it does not affect the algorithm thus we can set $\eta = 1$. On the contrary, if the perceptron algorithm starts with a different initialization, it will find a different separating hyperplane. The reader may wonder if such hyperplanes are all equivalent in terms of the classification accuracy of the test set; the answer is no: different hyperplanes may lead to different probability errors. In particular, the next section shows that the maximal margin hyperplane minimizes a probability error upperbound on the space of all possible hyperplanes.

2.3.2 Maximal Margin Classifier

The PAC theory suggested us that for a class of target functions, a hypothesis h that is learned consistently with the training set provides low probability error and we can show an analytical bound for such error. This idea can be applied to hyperplanes to estimate the final probability error but also to suggest a refinement of the learning algorithm. Indeed, one of the interesting results

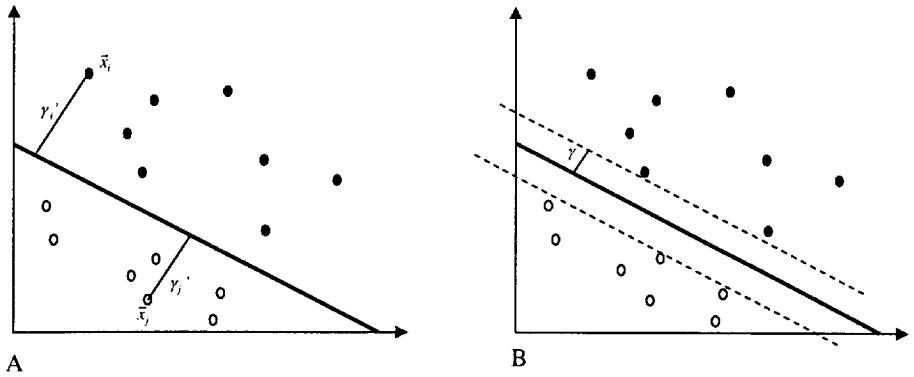

Figure 2.11: Geometric margins of two points (part A) and margin of the hyperplane (part B).

of the statistical learning theory is that to reduce such probability, we need to select the hyperplane (from the set of separating hyperplanes) that shows the maximum distance between positive and negative examples. To understand better this idea let us introduce some definitions:

Def. 2.18 *The **functional (geometric) margin distribution** of a hyperplane $(\vec{w},b)$ with respect to a training set S is the distribution of the functional (geometric) margins of the examples, i.e. $y_i(\vec{w} \cdot \vec{x}_i + b) \forall \vec{x}_i \in S$.*

Def. 2.19 *The **functional (geometric) margin of a hyperplane** is the minimum functional (geometric) margin of the distribution.*

Def. 2.20 *The **functional (geometric) margin of a training set** S is the maximum functional (geometric) margin of a hyperplane over all possible hyperplanes. The hyperplane that realizes such maximum is called the **maximal margin hyperplane**.*

Figure 2.11 shows the geometric margins of the points $\vec{x}_i$ and $\vec{x}_j$ (part A) and the geometric margin of the hyperplane (part B) whereas Figure 2.12 shows two separating hyperplanes that realizes two different margins.

Intuitively, the larger the margin of a hyperplane is, the lower the probability of error is. The important result of the statistical learning theory is that

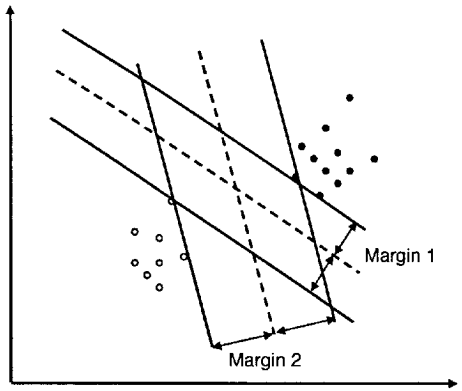

Figure 2.12: Margins of two hyperplanes.

an analytical upperbound to such error is correlated to the hyperplanes and the maximal margin hyperplane is associated with the lowest bound.

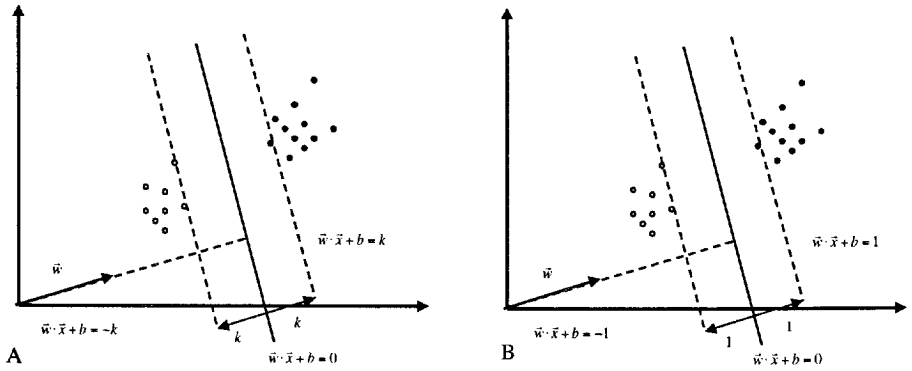

Figure 2.13: Margins of two hyperplanes.

In order to show such analytical result let us focus on finding the maximal margin hyperplane. Figure 2.13 shows that given a hyperplane $\vec{w} \cdot \vec{x} + b = 0$, a necessary condition to be a maximal margin hyperplane is that (a) two *frontier* hyperplanes (negative and positive frontiers) exist and (b) they hold

the following properties:

1. their equations are $\vec{w} \cdot \vec{x} + b = k$ and $\vec{w} \cdot \vec{x} + b = -k$, i.e. they are parallel to the target hyperplane and are both located at a distance of k ($\frac{k}{||\vec{w}||}$ if $\vec{w}$ is not a normalized vector;

2. such equations satisfy the constraints $y_i(\vec{w} \cdot \vec{x}_i + b) \geq k \; \forall x_i \in S$, i.e. they both separate the data points S.

3. A hyperplane whose distance from the frontier hyperplanes is maximal is the maximum margin hyperplane.

First, property 1 follows from a simple consideration: suppose that the nearest positive example $\vec{x}^+$ is located at a distance of γ_i from a hyperplane h_1 and the nearest negative example $\vec{x}^-$ is located at a distance of γ_j ($\neq \gamma_i$), and the h_1 margin is the minimum between γ_i and γ_j. If we select a hyperplane h_2 parallel to h_1 and equidistant from $\vec{x}^+$ and $\vec{x}^-$, it will be at a distance of $k = \frac{\gamma_i + \gamma_j}{2}$ from both $\vec{x}^+$ and $\vec{x}^-$. Since $k \geq min\{\gamma_i, \gamma_j\}$, the margin of h_2 equidistant from the frontier points is always greater or equal than other hyperplanes.

Second, previous property has shown that the nearest positive examples is located on the frontier $\vec{w} \cdot \vec{x} + b = k$ thus all the other positive examples $\vec{x}^+$ have a functional margin $\vec{w} \cdot \vec{x}^+ + b$ larger than k. The same rational applies to the negative examples but, to deal with positive quantities, we multiply $(\vec{w} \cdot \vec{x}_i + b)$ by the label y_i, thus, we obtain the constrain $y_i(\vec{w} \cdot \vec{x}_i + b_k) \geq k$.

Finally, the third property holds since $\frac{k}{||\vec{w}||}$ is the distance from one of the two frontier hyperplanes which, in turn, is the distance from the nearest points, i.e. the margin.

From these properties, it follows that the maximal margin hyperplane is derived by solving the optimization (maximization) problem below:

$$\begin{cases} max \quad \frac{k}{||\vec{w}||} \\ y_i(\vec{w} \cdot \vec{x}_i + b) \geq 1 \quad \forall \vec{x}_i \in S \end{cases} \tag{2.11}$$

where $\frac{k}{||\vec{w}||}$ is the objective function, $y_i(\vec{w} \cdot \vec{x}_i + b) = 1 \quad \forall \vec{x}_i \in S$ are the set of linear equality constraints $h_i(\vec{w})$ and $y_i(\vec{w} \cdot \vec{x}_i + b) > 1 \quad \forall \vec{x}_i \in S$ are the set of linear inequality constraints, $g_i(\vec{w})$. Note that (1) the objective function is quadratic since $||\vec{w}|| = \vec{w} \cdot \vec{w}$ and (2) we can rescale the distance among the

data points such that the maximal margin hyperplane has a margin of exactly 1. Thus, we can rewrite Eq. 2.11 as follows:

$$\begin{cases} max \quad \frac{1}{||\vec{w}||} \\ y_i(\vec{w} \cdot \vec{x}_i + b) \geq 1 \quad \forall \vec{x}_i \in S \end{cases} \tag{2.12}$$

Moreover, we can transform the above maximization problem in the following minimization problem:

$$\begin{cases} min \quad ||\vec{w}|| \\ y_i(\vec{w} \cdot \vec{x}_i + b) \geq 1 \quad \forall \vec{x}_i \in S \end{cases} \tag{2.13}$$

Eq. 2.13 states that to obtain a maximal margin hyperplane, we have to minimize the norm of the gradient $\vec{w}$ but it does not provide any analytical evidence on the benefit of choosing such hyperplane. On the contrary, the PAC theory provides the link with the error probability with the following theorem:

Theorem 2.21 *(Vapnik, 1982) Consider hyperplanes $\vec{w} \cdot \vec{x} + b = 0$ in a $\mathbb{R}^n$ vector space as hypotheses. If all examples $\vec{x}_i$ are contained in a ball of radius R and*

$$\forall \vec{x}_i \in S, \quad y_i(\vec{w} \cdot \vec{x}_i + b) \geq 1, \quad with \quad ||\vec{w}|| \leq A$$

then this set of hyperplanes has a VC-dimension d bounded by

$$d \leq min(R^2 \times A^2, n) + 1$$

The theorem states that if we set our hypothesis class H_A to be the set of hyperplanes whose $\vec{w}$ has a norm $\leq A$ then the VC dimension is less or equal than $R^2 \times A^2$. This means that if we reduce $||\vec{w}||$, we obtain a lower A and consequently a lower VC dimension, which in turn is connected to the error probability by the Theorem 2.11. This proves that, when the number of training examples is fixed, a lower VC-dimension will produce a lower error probability. In other words, as the maximum margin hyperplane minimizes the bound on the error probability, it constitutes a promising hypothesis for our learning problem.

Other interesting properties of the maximum margin hyperplane are derived from the optimization theory of convex functions over linear constraints. The main concepts of such theory relate on the following definition and theorem:

Def. 2.22 *Given an optimization problem with objective function $f(\vec{w})$, and equality constraints $h_i(\vec{w}) = 0$, $i = 1, .., l$, we define the Lagrangian function as*

$$L(\vec{w}, \vec{\beta}) = f(\vec{w}) + \sum_{i=1}^{l} \beta_i h_i(\vec{w}),$$

where the coefficient β_i are called Lagrange multipliers.

Theorem 2.23 *(Lagrange) A necessary condition for a normal point $\vec{w}^*$ to be a minimum of $f(w)$ subject to $h_i(\vec{w}) = 0$, $i = 1, .., l$, with f, $h_i \in C$ is*

$$\frac{\partial L(\vec{w}^*, \vec{\beta}^*)}{\partial \vec{w}} = \vec{0} \tag{2.14}$$

$$\frac{\partial L(\vec{w}^*, \vec{\beta}^*)}{\partial \vec{\beta}} = \vec{0} \tag{2.15}$$

for some values of $\vec{\beta}^*$. The above conditions are also sufficient provided that $\partial L(\vec{\beta}^*)$ is a convex function of $\vec{w}$.

Proof (necessity)
A continue function has a local maximum (minimum) when the partial derivatives are equal 0, i.e. $\frac{\partial f(\vec{w})}{\partial \vec{w}} = \vec{0}$. Since, we are in presence of constraints, it is possible that $\frac{\partial f(\vec{w}^*)}{\partial \vec{w}} \neq \vec{0}$. To respect such equality constraints, given the starting point $\vec{w}^*$, we can move only perpendicularly to $\frac{\partial h_i(\vec{w}^*)}{\partial \vec{w}}$. In other words, we can only move perpendicularly to the subspace V spanned by the vectors $\frac{\partial h_i(\vec{w}^*)}{\partial \vec{w}}, i = 1, .., l$. Thus, if a point $\frac{\partial f(\vec{w}^*)}{\partial \vec{w}}$ lies on V, any direction we move causes to violate the constraints. In other words, if we start from such point, we cannot increase the objective function, i.e. it can be a minimum or maximum point. The V memberships can be stated as the linear dependence between $\frac{\partial f(\vec{w}^*)}{\partial \vec{w}}$ and $\frac{\partial h_i(\vec{w}^*)}{\partial \vec{w}}$, formalized by the following equation:

$$\frac{\partial f(\vec{w}^*)}{\partial \vec{w}} + \sum_{i=1}^{l} \beta_i \frac{\partial h_i(\vec{w}^*)}{\partial \vec{w}} = \vec{0} \tag{2.16}$$

where $\exists i : \beta_i \neq 0$. This is exactly the condition 2.14. Moreover, Condition 2.15 holds since $\frac{\partial L(\vec{w}^*, \vec{\beta}^*)}{\partial \vec{\beta}} = (h_1(\vec{w}^*), h_2(\vec{w}^*), ..., h_l(\vec{w}^*))$ and all the constraints $h_i(\vec{w}^*) = 0$ are satisfied for the feasible solution $\vec{w}^*$. □

The above conditions can be applied to evaluate the maximal margin classifier, i.e. the Problem 2.13, but the general approach is to transform Problem 2.13 in an equivalent problem, simpler to solve. The output of such transformation is called dual problem and it is described by the following definition.

Def. 2.24 *Let $f(\vec{w})$, $h_i(\vec{w})$ and $g_i(\vec{w})$ be the objective function, the equality constraints and the inequality constraints (i.e. $\geq$) of an optimization problem, and let $L(\vec{w}, \vec{\alpha}, \vec{\beta})$ be its Lagrangian, defined as follows:*

$$L(\vec{w}, \vec{\alpha}, \vec{\beta}) = f(\vec{w}) + \sum_{i=1}^{m} \alpha_i g_i(\vec{w}) + \sum_{i=1}^{l} \beta_i h_i(\vec{w})$$

The Lagrangian dual problem *of the above primal problem is*

$$\begin{aligned} &maximize \quad \theta(\vec{\alpha}, \vec{\beta}) \\ &subject\ to \quad \vec{\alpha} \geq \vec{0} \end{aligned}$$

where $\theta(\vec{\alpha}, \vec{\beta}) = inf_{w \in W} \ L(\vec{w}, \vec{\alpha}, \vec{\beta})$

The *strong duality theorem* assures that an optimal solution of the dual is also the optimal solution for the primal problem and vice versa, thus, we can focus on the transformation of Problem 2.13 according to the Definition 2.24.

First, we observe that the only constraints in Problem 2.13 are the inequalities $g_i(\vec{w}) = [y_i(\vec{w} \cdot \vec{x_i} + b) \geq 1 \quad \forall \vec{x_i} \in S]$.

Second, the objective function is $\vec{w} \cdot \vec{w}$. Consequently, the primal Lagrangian[4] is

$$L(\vec{w}, b, \vec{\alpha}) = \frac{1}{2}\vec{w} \cdot \vec{w} - \sum_{i=1}^{m} \alpha_i [y_i(\vec{w} \cdot \vec{x_i} + b) - 1], \tag{2.17}$$

where α_i are the Lagrange multipliers and b is the extra variable associated with the threshold.

Third, to evaluate $\theta(\vec{\alpha}, \vec{\beta}) = inf_{w \in W} \ L(\vec{w}, \vec{\alpha}, \vec{\beta})$, we can find the minimum of the Lagrangian by setting the partial derivatives to 0.

$$\frac{\partial L(\vec{w}, b, \vec{\alpha})}{\partial \vec{w}} = \vec{w} - \sum_{i=1}^{m} y_i \alpha_i \vec{x_i} = \vec{0} \quad \Rightarrow \quad \vec{w} = \sum_{i=1}^{m} y_i \alpha_i \vec{x_i} \tag{2.18}$$

[4] As $\vec{w} \cdot \vec{w}$ or $\frac{1}{2}\vec{w} \cdot \vec{w}$ is the same optimization function from a solution point of view, we use the $\frac{1}{2}$ factor to simplify the next computations.

$$\frac{\partial L(\vec{w}, b, \vec{\alpha})}{\partial b} = \sum_{i=1}^{m} y_i \alpha_i = 0 \tag{2.19}$$

Finally, by substituting Eq. 2.18 and 2.19 into the primal Lagrangian we obtain

$$\begin{aligned} L(\vec{w}, b, \vec{\alpha}) &= \frac{1}{2}\vec{w}\cdot\vec{w} - \sum_{i=1}^{m}\alpha_i[y_i(\vec{w}\cdot\vec{x_i}+b)-1] = \\ &= \frac{1}{2}\sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j \vec{x_i}\cdot\vec{x_j} - \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j \vec{x_i}\cdot\vec{x_j} + \sum_{i=1}^{m}\alpha_i \\ &= \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j \vec{x_i}\cdot\vec{x_j} \end{aligned} \tag{2.20}$$

which according to the Definition 2.24 is the optimization function of the dual problem subject to $\alpha_i \geq 0$. In summary, the final dual optimization problem is the following:

$$\begin{aligned} maximize \quad & \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j \vec{x_i}\cdot\vec{x_j} \\ subject\ to \quad & \alpha_i \geq 0, \quad i = 1,..,m \\ & \sum_{i=1}^{m} y_i \alpha_i = 0 \end{aligned}$$

where $\vec{w} = \sum_{i=1}^{m} y_i \alpha_i \vec{x_i}$ and the $\sum_{i=1}^{m} y_i \alpha_i = 0$ are the relation derived from Eq. 2.18 and 2.19. Other conditions establishing interesting properties can be derived by the Khun-Tucker theorem. This provides the following relations for an optimal solution:

$$\begin{aligned} \frac{\partial L(\vec{w}^*, \vec{\alpha}^*, \vec{\beta}^*)}{\partial \vec{w}} &= \vec{0} \\ \frac{\partial L(\vec{w}^*, \vec{\alpha}^*, \vec{\beta}^*)}{\partial \vec{\beta}} &= \vec{0} \\ \alpha_i^* g_i(\vec{w}^*) &= 0, \quad i = 1,..,m \\ g_i(\vec{w}^*) &\leq 0, \quad i = 1,..,m \\ \alpha_i^* &\geq 0, \quad i = 1,..,m \end{aligned}$$

The third equation is usually called Karush-Khun-Tucker condition and it is very interesting for Support Vector Machines as it states that $\alpha_i^* \times [y_i(\vec{w}\cdot\vec{x_i}+b)-1] = 0$. On the one hand, if $\alpha_i^* = 0$ the training point $\vec{x_i}$ does not affect $\vec{w}$ as stated by Eq. 2.18. This means that the separating hyperplane and the associated classification function do not depend on such vectors. On the other hand, if $\alpha_i^* \neq 0 \Rightarrow [y_i(\vec{w}\cdot\vec{x_i}+b)-1] = 0 \Rightarrow y_i(\vec{w}\cdot\vec{x_i}+b) = -1$, i.e. $\vec{x_i}$ is located on the frontier. Such data points are called support vectors (SV) as they support the classification function. Moreover, they can be used to derive the threshold b by evaluating the average between the projection of a positive and a negative SV on the gradient $\vec{w}^*$, i.e.:

$$b^* = -\frac{\vec{w}^*\cdot\vec{x}^+ + \vec{w}^*\cdot\vec{x}^-}{2}$$

The error probability upperbound of support vector machines provides only an evidence of the maximal margine effectiveness. Unfortunately, there is no analytical proof that such approach produces the *best* linear classifier. Indeed, it may exist other bounds lower than the one derived with the VC dimension and the related theory. Another drawback of the maximal margin approach is that it can only be applied when the training data is linearly separable, i.e. the constraints over the negative and positive examples must be satisfied. Such *hard* conditions provided the idea of the alternative approach name, i.e. Hard Margin Support Vector Machines. As opposite, the next section introduces the Soft Margin Support Vector Machines, whose optimization problem allows the linear function to commit a *certain number of error.*

2.3.3 Soft Margin Support Vector Machines

In real scenario applications, training data is often affected by noise due to several reasons, e.g. classification mistakes of annotators. These may cause the data not to be separable by any linear function. Additionally, the target problem itself may be not separable in the designed feature space. As result, the Hard Margin SVMs will fail to converge.

In order to solve such critical aspect, the Soft Margin Support Vector Machines have been designed. Their main idea is to allow the optimization problem to provide solutions that can violate a certain number of constraints. Intuitively, to be as much as possible consistent with the training data, such number of errors should be the lowest possible. This trade-off between the separability

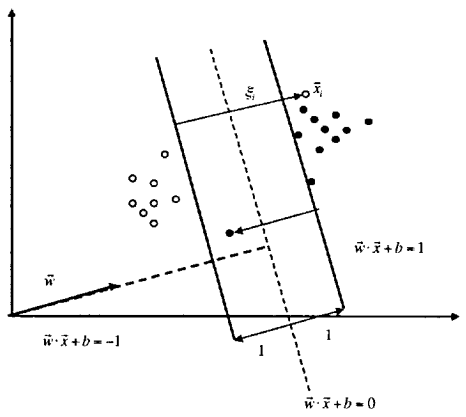

Figure 2.14: Soft Margin Hyperplane.

with highest margin and the number of errors can be specified by (a) introducing *slack variables* ξ_i in the inequality constraints of Problem 2.13 and (b) adding the minimization the number of errors in the objective function. The resulting optimization problem is

$$\begin{cases} min \quad ||\vec{w}|| + C\sum_{i=1}^{m}\xi_i^2 \\ y_i(\vec{w}\cdot\vec{x_i}+b) \geq 1-\xi_i, \quad \forall i = 1,..,m \\ \xi_i \geq 0, \quad i = 1,..,m \end{cases} \tag{2.21}$$

whose the main characteristics are:

- The constraint $y_i(\vec{w}\cdot\vec{x_i}+b) \geq 1-\xi_i$ allows the point $\vec{x}_i$ to violate the hard constraint of Problem 2.13 of a quantity equal to ξ_i. This is clearly shown by the outliers in Figure 2.14, e.g. $\vec{x}_i$.

- If a point is misclassified by the hyperplane then the slack variable assumes a value larger than 1. For example, Figure 2.14 shows the misclassified point x_i and its associated slack variable ξ_i which is necessarily > 1. Thus, $\sum_{i=1}^{m}\xi_i$ is an upperbound to the number of errors. The same property is held by the quantity, $\sum_{i=1}^{m}\xi_i^2$, which can be used as an alternative bound.

- The constant C tunes the trade-off between the classification errors and the margin. The higher C is, the lower number of errors the optimal solution commits. For $C \to \infty$, Problem 2.21 approximates Problem 2.13.

- Similarly to the hard margin error probability upperbound, it can be proven that minimizing $||\vec{w}|| + C\sum_{i=1}^{m}\xi_i^2$ minimizes the error probability of classifiers which are not perfectly consistent with the training data, e.g. they do not necessarily classify correctly all the training data.

- Figure 2.15 shows that by accepting some errors, it is possible to find better hypothesis. In the part A, the point $\vec{x}_i$ prevents to derive a good margin. As we accept to mistake $\vec{x}_i$, the learning algorithm can find a more suitable margin (part B).

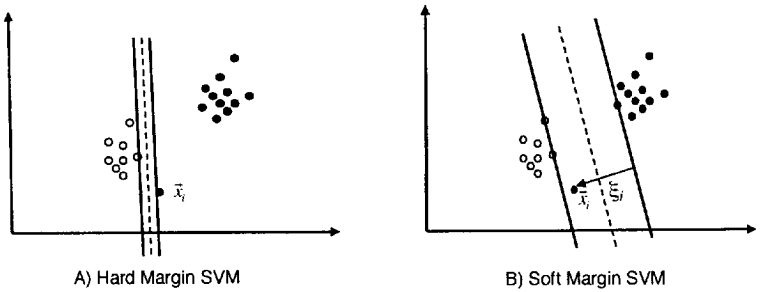

Figure 2.15: Soft Margin vs. Hard Margin hyperplanes.

As it has been done for the hard optimization problem, we can evaluate the primal Lagrangian:

$$L(\vec{w},b,\vec{\xi},\vec{\alpha}) = \frac{1}{2}\vec{w}\cdot\vec{w} + \frac{C}{2}\sum_{i=1}^{m}\xi_i^2 - \sum_{i=1}^{m}\alpha_i[y_i(\vec{w}\cdot\vec{x_i}+b)-1], \tag{2.22}$$

where α_i are Lagrangian multipliers.

The dual problem is obtained by imposing stationarity on the derivatives

respect to $\vec{w}$, $\vec{\xi}$ and b:

$$\begin{aligned}
\frac{\partial L(\vec{w},b,\vec{\xi},\vec{\alpha})}{\partial \vec{w}} &= \vec{w} - \sum_{i=1}^{m} y_i\alpha_i\vec{x}_i = \vec{0} \quad \Rightarrow \quad \vec{w} = \sum_{i=1}^{m} y_i\alpha_i\vec{x}_i \\
\frac{\partial L(\vec{w},b,\vec{\xi},\vec{\alpha})}{\partial \vec{\xi}} &= C\vec{\xi} - \vec{\alpha} = \vec{0} \\
\frac{\partial L(\vec{w},b,\vec{\xi},\vec{\alpha})}{\partial b} &= \sum_{i=1}^{m} y_i\alpha_i = 0
\end{aligned} \tag{2.23}$$

By substituting the above relations into the primal, we obtain the following dual objective function:

$$\begin{aligned}
w(\vec{\alpha}) &= \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{m} y_iy_j\alpha_i\alpha_j\vec{x_i}\cdot\vec{x_j} + \frac{1}{2C}\vec{\alpha}\cdot\vec{\alpha} - \frac{1}{C}\vec{\alpha}\cdot\vec{\alpha} = \\
&= \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{m} y_iy_j\alpha_i\alpha_j\vec{x_i}\cdot\vec{x_j} - \frac{1}{2C}\vec{\alpha}\cdot\vec{\alpha} = \\
&= \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{m} y_iy_j\alpha_i\alpha_j\big(\vec{x_i}\cdot\vec{x_j} + \frac{1}{C}\delta_{ij}\big),
\end{aligned} \tag{2.24}$$

where $\delta_{ij} = 1$ if $i = j$ and 0 otherwise (Kronecker's delta). The objective function is subject to the usual constraints:

$$\begin{cases} \alpha_i \geq 0, \quad \forall i = 1,..,m \\ \sum_{i=1}^{m} y_i\alpha_i = 0 \end{cases}$$

The above dual can be used to find a solution of Problem 2.21, which extends the applicability of linear functions to classification problems not completely linearly separable. The separability property relates not only on the available class of hypotheses, e.g. linear vs. polynomial functions, but it strictly depends on the adopted features. Their roles is to provide a map between the example data and vectors in $\mathbb{R}^n$. Given such mapping, the scalar product provides a measure of the *similarity* between pairs of examples or, according to a colder interpretation, it provides a partitioning function based on such features.

The next Section shows that, it is possible to substitute the scalar product of two feature vectors with a function between the data examples directly. This allows us to avoid the explicit feature design and consequently enables us to

define similarly measures, called kernel functions, in a more flexible way. Such kernel functions, in turn, define implicit feature spaces.

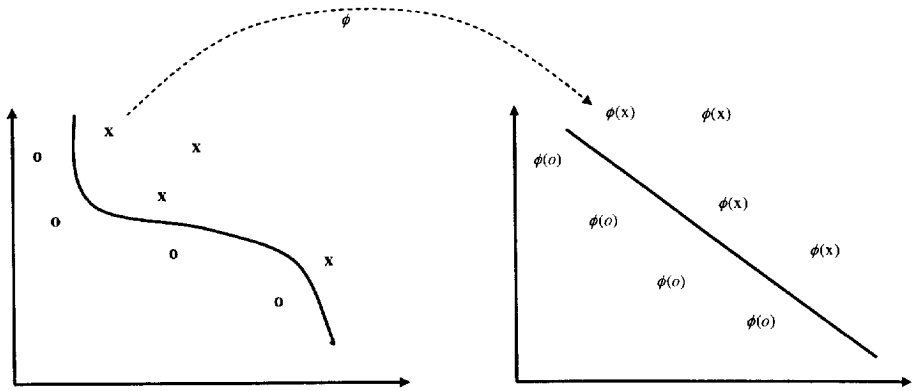

Figure 2.16: A mapping ϕ which makes separable the initial data points.

2.4 Kernel Methods

One of the most difficult tasks on applying machine learning is the feature design. Features should represent data in a way that allows learning algorithms to separate positive from negative examples. In SVMs, features are used to build the vector representation of data examples and the scalar product between example pairs quantifies (sometimes, it simply counts the number of common features) how much they are similar. Instead of encoding data in feature vectors, we may design kernel functions that provide such similarity between example pairs without using an explicit feature representation. The reader may object that the learning algorithm still requires the supporting feature space to model the hyperplane and the data points, but this is not the case, if we work in the dual space.

The real limit of the kernel functions is that they must generate a well defined inner product vector space. Such property will hold if the Mercer's conditions are satisfied. Fortunately, there are many kernels, e.g. polynomial, string, lexical and tree kernels that satisfy such conditions and give us the possibility to use them in SVMs.

The above kernels allow us to define in a more abstract way our leaning problem and in many cases allow us to solve non linear problems by remapping the initial data points in a separable space as shown by Figure 2.16. The following example illustrates one of the case in which a non-linear function can be expressed in a linear formulation in a different space.

Example 2.25 *Overcoming linear inseparability*
Suppose that we want to study the force interactions between two masses m_1 and m_2. m_1 is free to move whereas m_2 is located at a distance r and blocked at its position. The two masses are subject to the Newtown's gravity law:

$$f(m_1, m_2, r) = C\frac{m_1 m_2}{r^2},$$

thus mass m_1 naturally moves towards m_2.

We apply a force f_a opposite to f and we note that only sometimes we are able to invert the motion of m_1. To study such phenomenon, we carry out a set of experiments with different experimental parameters, i.e. m_1, m_2, r and f_a and we observe the result of our action: success if we get m_1 further from m_2 and failure, if m_1 approaches m_2.

Each successful experiment can be considered a positive example whereas unsuccessful experiments are considered as negative examples. We choose the experiment parameters as the target feature set and we apply $SVMs$ to learn the classification of new experiments (f_a, m_1, m_2, r) in successful or unsuccessful, i.e. if $f_a - f(m_1, m_2, r) > 0$ or otherwise, respectively. To do this, $SVMs$ should learn $f(m_1, m_2, r)$ but, since the gravity law is clearly non-linear, the Hard Margin $SVMs$ will, in general, not converge and (2) the soft margine $SVMs$ will provide inaccurate results.

The solution is to map our initial feature space in another feature space, i.e. $(f_a, m_1, m_2, r) \rightarrow (lnf_a, ln(m_1), ln(m_2), ln(r)) = (k, x, y, z)$. Since $ln\big(f(m_1, m_2, r)\big) = ln(C) + ln(m_1) + ln(m_2) - 2ln(r) = c + x + y - 2z$, we can express the logarithm of gravity law with a linear combination of the transformed features. More in detail, points above the ideal hyperplane $k - (c + x + y - 2z) = 0$, i.e. points that satisfy $k - (c + x + y - 2z) > 0$ (or equivalently that satisfy $f_a - f(m_1, m_2, r) > 0$), are successful experiments whereas points below such hyperplane refer to unsuccessful experiments. Since this proves that a separating hyperplane of the training set exists, $SVMs$ will always converge (with an error dependent on the number of training data).

2.4.1 The Kernel Trick

Section 2.3.1 has shown that the Perceptron algorithm, used to learn linear classifiers, can be adapted to work in the dual space. In particular, such algorithm (see Table 2.3) clearly shows that feature vectors always appear in the scalar product, consequently, we can replace the feature vectors $\vec{x}_i$ with the data objects o_i by substituting the scalar product $\vec{x}_i \cdot \vec{x}_j$ with the kernel function $k(o_i, o_j)$, where the o_i are the initial objects mapped into $\vec{x}_i$ by using a feature representation, $\phi(.)$. Thus $\vec{x}_i \cdot \vec{x}_j = \phi(o_i) \cdot \phi(o_j) = k(o_i, o_j)$.

Similarly to the Perceptron algorithm, the dual optimization problem of Soft Margin $SVMs$ (Eq. 2.24) contain feature vectors only inside a scalar product, which can be substituted with $k(o_i, o_j)$. Therefore, the kernelized version of the soft margin $SVMs$ is

$$\begin{cases} maximize \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j \big(k(o_i, o_j) + \frac{1}{C}\delta_{ij}\big) \\ \alpha_i \geq 0, \quad \forall i = 1, .., m \\ \sum_{i=1}^{m} y_i \alpha_i = 0 \end{cases}$$

Moreover, Eq. 2.9 for the Perceptron appears also in the Soft Margin $SVMs$ (see conditions 2.23), hence we can rewrite the SVM classification function as in Eq. 2.10 and use kernel inside it, i.e.:

$$h(x) = sgn\left(\sum_{i=1}^{m} \alpha_i y_i k(o_i, o_j) + b\right)$$

The data object o is mapped in the vector space trough a feature extraction procedure $\phi : o \rightarrow (x_1, ..., x_n) = \vec{x}$, more in general, we can map a vector $\vec{x}$ of one feature space into another feature space:

$$\vec{x} = (x_1, ..., x_n) \rightarrow \phi(\vec{x}) = (\phi_1(\vec{x}), ..., \phi_n(\vec{x}))$$

This supports the general definition of kernels:

Def. 2.26 *A kernel is a function k, such that $\forall\ \vec{x},\vec{z} \in X$*

$$k(\vec{x},\vec{z}) = \phi(\vec{x}) \cdot \phi(\vec{z})$$

where ϕ is a mapping from X to an (inner product) feature space.

Note that, once we have defined a kernel function that is effective for a given learning problem, we do not need to find which mapping ϕ it corresponds to. It is enough to know that such mapping exists. The following proposition states the conditions that guaranteed such existence.

Proposition 2.27 *(Mercer's conditions)*
Let X be a finite input space with $K(\vec{x},\vec{z})$ a symmetric function on X. Then $K(\vec{x},\vec{z})$ is a kernel function if and only if the matrix

$$k(\vec{x},\vec{z}) = \phi(\vec{x}) \cdot \phi(\vec{z})$$

is positive semi-definite (has non-negative eigenvalues).

The proof of such proposition is the following (from [Cristianini and Shawe-Taylor, 2000]). Let us consider a symmetric function on a finite space $X = \{x_1, x_2, ..., x_n\}$

$$\boldsymbol{K} = \left(K(x_i, x_j)\right)_{i,j=1}^{n}$$

Since $\boldsymbol{K}$ is symmetric there is an orthogonal matrix $\boldsymbol{V}$ such that $\boldsymbol{K} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}'$ where $\boldsymbol{\Lambda}$ is a diagonal matrix containing the eigenvalues λ_t of $\boldsymbol{K}$, with corresponding eigenvectors $\vec{v}_t = (v_{ti})_{i=1}^{n}$, i.e. the columns of $\boldsymbol{V}$. Now assume that all the eigenvalues are non-negatives and consider the feature mapping:

$$\phi : \vec{x}_i \rightarrow \left(\sqrt{\lambda_t} v_{ti}\right)_{t=1}^{n} \in \mathbb{R}^n, i = 1, .., n.$$

We now have that,

$$\phi(\vec{x}_i) \cdot \phi(\vec{x}_j) = \sum_{t=1}^{n} \lambda_t v_{ti} v_{tj} = (\boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}')_{ij} = \boldsymbol{K}_{ij} = K(x_i, x_j).$$

This proves that $K(\vec{x},\vec{z})$ is a *valid* kernel function that corresponds to the mapping ϕ. Therefore, the only requirement to derive the mapping ϕ is that the eigenvalues of $\boldsymbol{K}$ are non-negatives since if we had a negative eigenvalue λ_s associated with the eigenvector $\vec{v}_s$, the point

$$\vec{z} = \sum_{i=1}^{n} \boldsymbol{v}_{si} \phi(\vec{x}_i) = \sqrt{\boldsymbol{\Lambda}}\boldsymbol{V}'\vec{v}_s.$$

in the feature space would have norm squared

$$||\vec{z}||^2 = \vec{z} \cdot \vec{z} = \vec{v}_s' \boldsymbol{V} \sqrt{\boldsymbol{\Lambda}} \sqrt{\boldsymbol{\Lambda}} \boldsymbol{V}' \vec{v}_s = \vec{v}_s' \boldsymbol{V} \boldsymbol{\Lambda} \boldsymbol{V}' \vec{v}_s = \vec{v}_s' \boldsymbol{K} \vec{v}_s = \lambda_s < 0,$$

which contradicts the geometry of that space.

2.4.2 Polynomial Kernel

The above section has shown that kernel functions can be used to map a vector space in another space in which the target classification problem becomes linearly separable. Another advantage is the possibility to map the initial feature space in a richer feature space which includes a high number of dimensions (possibly infinite). For example, the polynomial kernel maps the initial features in a space which contains both the original features and all the possible feature conjunctions, e.g. given the components x_1 and x_2 the new space will contain $x_1 x_2$. This is interesting for text categorization applications as from the features like `hard`, `rock` and `disk`, the polynomial kernel automatically derives the feature `hard rock` or `hard disk` which may help to disambiguate between *Music Store* from *Computer Store* categories.

The great advantage of using kernel functions is that we do not need to store, e.g. in the computer memory, the vectors of the new space to evaluate the inner product. For example, suppose that the initial feature space has a cardinality of 100,000 features, i.e. the size of the vocabulary in a text categorization problem, only the word pairs would be 10^{10}, which cannot be managed by many learning algorithms. The polynomial kernel can be used to evaluate the scalar product between pairs of vectors of such huge space by using only the initial space and vocabulary, as it is shown by the following passages:

$$\begin{aligned}
(\vec{x} \cdot \vec{z})^2 &= \left(\sum_{i=1}^{n} x_i z_i\right)^2 &&= \left(\sum_{i=1}^{n} x_i z_i\right)\left(\sum_{j=1}^{n} x_i z_i\right) \\
&= \sum_{i=1}^{n}\sum_{j=1}^{n} x_i x_j z_i z_j &&= \sum_{i,j \in \{1,..,n\}} (x_i x_j)(z_i z_j) \\
&= \sum_{k=1}^{m} X_k Z_k &&= \vec{X} \cdot \vec{Z}
\end{aligned}$$

where

- $\vec{x}$ and $\vec{z}$ are two vectors of the initial space,

- $\vec{X}$ and $\vec{Z}$ are the vectors of the final space and

- $X_k = x_i x_j$, $Z_k = z_i z_j$ with $k = (i-1) \times n + j$ and $m = n^2$.

Therefore, (a) the mapping between the two space is $\phi(\vec{x}) = (x_i x_j)$ for $j = 1, .., n$ and for $i = 1, .., n$, (b) to evaluate $\vec{X} \cdot \vec{Z}$, we just compute the square of the scalar product in the initial space, i.e. $(\vec{x} \cdot \vec{z})^2$ and (c) the final space contains conjunctions and also the features of the initial space ($x_i x_i$ is equivalent to x_i).

Additionally, since $x_i x_j = x_j x_i$, the conjunctions receive the double of the weight of single features. The number of distinct features are n for $i = 1$ and $j = 1, .., n$; $(n-1)$ for $i = 2$ and $j = 2, .., n$; ... and 1 for $i = n$ and $j = n$. It follows that the overall number is

$$n + (n-1) + (n-2) + .. + 1 = \sum_{k=1}^{n} k = \frac{n(n+1)}{2}$$

Another way to compute such number it to consider that, to build all the monomials, the first variable can be chosen out of $n+1$ possibilities (n symbols to form conjunctions and the empty symbol for the single feature) whereas for the second variable only n chances are available (no empty symbol at this time). We will obtain all permutations of each monomial of two variables. Therefore, to obtain distinct features, we need to divide the number of monomials, i.e. $(n+1)n$, by the number of permutations of two variables, i.e. $2! = 2$. The final quantity can be expressed with the binomial coefficient $\binom{n+1}{2}$.

Given the above observation, we can generalize the kernel from degree 2 to a degree d by computing $(\vec{x} \cdot \vec{z})^d$. The results are all monomials of degree d or equivalently all the conjunctions constituted up to d features. The distinct features will be $\binom{n+d-1}{d}$ since we can choose either the empty symbol up to $d-1$ times or n variables.

A still more general kernel can be derived by introducing a constant in the scalar product computation. Hereafter, we show the case for a degree equal to

two:

$$(\vec{x} \cdot \vec{z} + c)^2 = \Big(\sum_{i=1}^{n} x_i z_i + c\Big)^2 = \Big(\sum_{i=1}^{n} x_i z_i + c\Big)\Big(\sum_{j=1}^{n} x_i z_i + c\Big) =$$
$$= \sum_{i=1}^{n}\sum_{j=1}^{n} x_i x_j z_i z_j + 2c\sum_{i=1}^{n} x_i z_i + c^2 =$$
$$= \sum_{i,j \in \{1,..,n\}} (x_i x_j)(z_i z_j) + 2c\sum_{i=1}^{n} \big(\sqrt{2c}x_i\big)\big(\sqrt{2c}z_i\big) + c^2$$

Note that the second summation introduces n individual features (i.e. x_i) whose weights are controlled by the parameter c which also determines the strength of the degree 0. Thus, we add (n+1) new features to the $\binom{n+1}{2}$ features of the previous kernel of degree 2. If we consider a generic degree d, i.e. the kernel $(\vec{x} \cdot \vec{z} + c)^d$, we will obtain $\binom{n+d-1}{d} + n + d - 1 = \binom{n+d}{d}$ distinct features (which have at least distinct weights). These are all monomials up to and including the degree d.

2.4.3 String Kernel

Kernel functions can be applied also to discrete space. As a first example, we show their potentiality on the space of finite strings.

Let Σ be a finite alphabet. A string is a finite sequence of characters from Σ, including the empty sequence. For string s and t we denote by $|s|$ the length of the string $s = s_1, .., s_{|s|}$, and by st the string obtained by concatenating the string s and t. The string $s[i:j]$ is the substring $s_i, .., s_j$ of s. We say that u is a subsequence of s, if there exist indices $\vec{I} = (i_1, ..., i_{|u|})$, with $1 \leq i_1 < ... < i_{|u|} \leq |s|$, such that $u_j = s_{i_j}$, for $j = 1, ..., |u|$, or $u = s[\vec{I}]$ for short. The length $l(\vec{I})$ of the subsequence in s is $i_{|u|} - i_i + 1$. We denote by Σ^* the set of all string

$$\Sigma^* = \bigcup_{n=0}^{\infty} \Sigma^n$$

We now define the feature space, $F = \{u_1, u_2..\} = \Sigma^*$, i.e. the space of all possible substrings. We map a string s in $\mathbb{R}^\infty$ space as follows:

$$\phi_u(s) = \sum_{\vec{I}: u = s[\vec{I}]} \lambda^{l(\vec{I})} \tag{2.25}$$

for some $\lambda \leq 1$. These features measure the number of occurrences of subsequences in the string s weighting them according to their lengths. Hence, the inner product of the feature vectors for two strings s and t give a sum over all common subsequences weighted according to their frequency of occurrences and lengths, i.e.

$$K(s,t) = \sum_{u \in \Sigma^*} \phi_u(s) \cdot \phi_u(t) = \sum_{u \in \Sigma^*} \sum_{\vec{I}: u = s[\vec{I}]} \lambda^{l(\vec{I})} \sum_{\vec{J}: u = t[\vec{J}]} \lambda^{l(\vec{J})} =$$

$$= \sum_{u \in \Sigma^*} \sum_{\vec{I}: u = s[\vec{I}]} \sum_{\vec{J}: u = t[\vec{J}]} \lambda^{l(\vec{I}) + l(\vec{J})} \tag{2.26}$$

The above equation defines a class of similarity functions known as string kernels or sequence kernels. These functions are interesting for text categorization as it allows the learning algorithm to quantify the matching between two different words, phrases, sentences or whole documents. For example, given two strings, *Bank* and *Rank*:

- B, a, n, k, Ba, Ban, Bank, an, ank, nk, Bn, Bnk, Bk and ak are the substrings of $Bank$.
- R, a, n, k, Ra, Ran, Rank, an, ank, nk, Rn, Rnk, Rk and ak are the substrings of $Rank$.

Such substrings are the features in the Σ^* that have non-null weights. These are evaluated by means of Eq. 2.25, e.g. $\phi_{\mathtt{B}}(\mathtt{Bank}) = \lambda^{(i_1 - i_1 + 1)} = \lambda^{(1-1+1)} = \lambda$, $\phi_{\mathtt{k}}(\mathtt{Bank}) = \lambda^{(i_1 - i_1 + 1)} = \lambda^{(4-4+1)} = \lambda$, $\phi_{\mathtt{an}}(\mathtt{Bank}) = \lambda^{(i_2 - i_1 + 1)} = \lambda^{(3-2+1)} = \lambda^2$ and $\phi_{\mathtt{Bk}}(\mathtt{Bank}) = \lambda^{(i_2 - i_1 + 1)} = \lambda^{(4-1+1)} = \lambda^4$.

Since Eq. 2.26 requires that the substrings in *Bank* and *Rank* match, we need to evaluate Eq. 2.25 only for the common substrings, i.e.:

- $\phi_{\mathtt{a}}(\mathtt{Bank}) = \phi_{\mathtt{a}}(\mathtt{Rank}) = \lambda^{(i_1 - i_1 + 1)} = \lambda^{(2-2+1)} = \lambda$,
- $\phi_{\mathtt{n}}(\mathtt{Bank}) = \phi_{\mathtt{n}}(\mathtt{Rank}) = \lambda^{(i_1 - i_1 + 1)} = \lambda^{(3-3+1)} = \lambda$,
- $\phi_{\mathtt{k}}(\mathtt{Bank}) = \phi_{\mathtt{k}}(\mathtt{Rank}) = \lambda^{(i_1 - i_1 + 1)} = \lambda^{(4-4+1)} = \lambda$,
- $\phi_{\mathtt{an}}(\mathtt{Bank}) = \phi_{\mathtt{an}}(\mathtt{Rank}) = \lambda^{(i_1 - i_2 + 1)} = \lambda^{(3-2+1)} = \lambda^2$,
- $\phi_{\mathtt{ank}}(\mathtt{Bank}) = \phi_{\mathtt{ank}}(\mathtt{Rank}) = \lambda^{(i_1 - i_3 + 1)} = \lambda^{(4-2+1)} = \lambda^3$,

- $\phi_{\mathtt{nk}}(\mathtt{Bank}) = \phi_{\mathtt{nk}}(\mathtt{Rank}) = \lambda^{(i_1 - i_2 + 1)} = \lambda^{(4-3+1)} = \lambda^2$,
- $\phi_{\mathtt{ak}}(\mathtt{Bank}) = \phi_{\mathtt{ak}}(\mathtt{Rank}) = \lambda^{(i_1 - i_2 + 1)} = \lambda^{(4-2+1)} = \lambda^3$.

It follows that $K(\mathtt{Bank}, \mathtt{Rank}) = (\lambda, \lambda, \lambda, \lambda^2, \lambda^3, \lambda^2, \lambda^3) \cdot (\lambda, \lambda, \lambda, \lambda^2, \lambda^3, \lambda^2, \lambda^3)$ $= 3\lambda^2 + 2\lambda^4 + 2\lambda^6$.

From this example, we note that short non-discontinuous strings receive the highest contribution, e.g. $\phi_{\mathtt{B}}(\mathtt{Bank}) = \lambda > \phi_{\mathtt{an}}(\mathtt{Bank}) = \lambda^2$. This may seem counterintuitive as longer string should be more important to characterize two textual snippets. Such inconsistency disappears if we consider that when a large string is matched, the same will happen for all its substrings. For example, the contribution coming from $Bank$, in the matching between the "*Bank of America*" and "*Bank of Italy*" strings, includes the match of B, a, n, k, Ba, Ban,...

Moreover, it should be noted that Eq. 2.26 is rather expensive from a computational point of view. To solve such problem, in [Lodhi *et al.*, 2000] was designed a method for its fast computation trough a recursive function.

First, a kernel over the space of strings of length n, Σ^n is computed, i.e.

$$K_n(s,t) = \sum_{u \in \Sigma^n} \phi_u(s) \cdot \phi_u(t) = \sum_{u \in \Sigma^n} \sum_{\vec{I}: u = s[\vec{I}]} \sum_{\vec{J}: u = t[\vec{J}]} \lambda^{l(\vec{I}) + l(\vec{J})}.$$

Second, a slightly different version of the above function is considered, i.e.

$$K'_i(s,t) = \sum_{u \in \Sigma^n} \phi_u(s) \cdot \phi_u(t) = \sum_{u \in \Sigma^i} \sum_{\vec{I}: u = s[\vec{I}]} \sum_{\vec{J}: u = t[\vec{J}]} \lambda^{|s| + |t| - i_1 - j_1 + 2},$$

for $i = 1, .., n-1$. $K'_i(s,t)$ is different than $K_n(s,t)$ since, to assign weights, the distances from the initial character of the substrings to the end of the string, i.e. $|s| - i_1 + 1$ and $|t| - j_1 + 1$, are used in place of the distances between the first and last characters of the substrings, i.e. $l(\vec{I})$ and $l(\vec{J})$.

It can be proved that $K_n(s,t)$ is evaluated by the following recursive relations:

- $K'_0(s,t) = 1$, for all s,t,
- $K'_i(s,t) = 0$, if min $(|s|,|t|) < i$,
- $K_i(s,t) = 0$, if min $(|s|,|t|) < i$,

- $K'_i(sx,t) = \lambda K'_i(s,t) + \sum_{j:t_j=x} K'_{i-1}(s,t[1:j-1])\lambda^{|t|-j+2}, i = 1,..,n-1,$

- $K_n(sx,t) = K_n(s,t) + \sum_{j:t_j=x} K'_{n-1}(s,t[1:j-1])\lambda^2.$

The general idea is that $K'_{i-1}(s,t)$ can be used to compute $K_n(s,t)$ when we increase the size of the input strings of one character, e.g. $K_n(sx,t)$. Indeed, K'_i and K_i compute the same quantity when the last character of the substring $u \in \Sigma^i$, i.e. x, coincides with the last character of the string, i.e. the string can be written as sx. Since $K'_i(sx,t)$ can be reduced to $K'_i(s,t)$, the recursion relation converges. The computation time of such process is proportional to $n \times |s| \times |t|$, i.e. an efficient evaluation.

2.4.4 Lexical Kernel

The most used Information Retrieval paradigm is based on the assumptions that (a) the semantic of a document can be represented by the semantic of its words and (b) to express the similarity between document pairs, it is enough to consider only the contribution from matching terms. In this view, two words that are strongly related, e.g. synonyms, do not contribute with their *relatedness* to the document similarity.

More advanced IR models attempt to take the above problem into account by introducing term similarities. Complex and interesting term similarities can be implemented using external (to the target corpus) thesaurus, like for example the Wordnet hierarchy [Fellbaum, 1998]. This intuitively provides a sort of similarity metric based on the length of the path that connects two terms in the hierarchy. Once a term similarity is designed, document similarities, which are the core functions of most TC algorithms, can be designed as well.

Given a term similarity function σ and two documents d_1 and $d_2 \in D$ (in the document set), we define their similarity as:

$$K(d_1,d_2) = \sum_{f_1 \in d_1, f_2 \in d_2} (w_1 w_2) \times \sigma(f_1,f_2) \qquad (2.27)$$

where w_1 and w_2 are the weights of the words (features) f_1 and f_2 in the documents d_1 and d_2, respectively. The interesting property is that such similarity is a valid kernel function and can, therefore, be used in SVMs. To prove this, it is enough to show that Eq. 2.27 is a specialization of the general definition of convolution kernels formalized in [Haussler, 1999] and reported hereafter.

Def. 2.28 *General Convolution Kernels*
Let $X, X_1, .., X_m$ be separable metric spaces, $x \in X$ a structure and $\vec{x} = x_1, ..., x_m$ its parts, where $x_i \in X_i \quad \forall i = 1, .., m$. Let R be a relation on the set $X \times X_1 \times .. \times X_m$ such that $R(\vec{x}, x)$ holds if $\vec{x}$ are the parts of x. We indicate with $R^{-1}(x)$ the set $\{\vec{x} : R(\vec{x}, x)\}$. Given two objects x and $y \in X$ their similarity $K(x,y)$ is defined as:

$$K(x,y) = \sum_{\vec{x} \in R^{-1}(x)} \sum_{\vec{y} \in R^{-1}(y)} \prod_{i=1}^{m} K_i(x_i, y_i) \qquad (2.28)$$

In the case of lexical kernel, X defines the document set (i.e. $D = X$), and X_1 is the vocabulary of the target document corpus ($X_1 = V$). It follows that $x = d$ (a document), $\vec{x} = x_1 = f \in V$ (a word which is a part of the document d) and $R^{-1}(d)$ defines the set of words in the document d.

Since $\prod_{i=1}^{m} K_i(x_i, y_i) = K_1(x_1, y_1)$, then $K_1(x_1, y_1) = K(f_1, f_2) = (w_1 w_2) \times \sigma(f_1, f_2)$, i.e. Eq. 2.27. Consequently, the lexical kernel is a valid convolution kernel.

An alternative way to prove that Eq. 2.27 is a valid kernel is the simple consideration that the kernel $K(d_1, d_2)$ can be written as $\vec{w}'_1 \boldsymbol{M}' \cdot \boldsymbol{M} \vec{w}_2$, where $\vec{w}_1$ and $\vec{w}_2$ are the vectors of weights associated with d_1 and d_2, and $\boldsymbol{M}$ and $\boldsymbol{M}'$ are the matrix and its transposed of the term similarities, i.e. the matrix defined by $\sigma(f_1, f_2)$. As shown in appendix, $\boldsymbol{P} = \boldsymbol{M}' \cdot \boldsymbol{M}$ is positive semi-definite, thus $K(d_1, d_2) = \vec{w}'_1 \boldsymbol{P} \vec{w}_2$ satisfies the Mercer's conditions.

The lexical kernel has successfully been applied to improve document categorization [Basili *et al.*, 2005] when few documents are available for training. Indeed, the possibility to match different words using a σ similarity allows SVMs to recover important semantic information.

2.5 Tree Kernel Spaces

The polynomial and the string kernels have shown that, starting from an initial feature set, they can automatically provide a very high number of interesting features. These are a first example of the usefulness of kernel methods. Other interesting kernel approaches aim to automatically generate large number of features from structures. For example, tree kernels are able to extract many types of tree fragments from a target tree. Their purpose is to model syntactic information in a target learning problem. In particular, tree kernels seem

well suited to model syntax in natural language applications, e.g. for the extraction of semantic predicative structures like *bought(Mary, a cat, in Rome)* [Moschitti, 2004].

Indeed, literature work shows that defining linguistic theories for the modeling of natural languages (e.g. [Jackendoff, 1990]) is a complex problem, far away from a sound and complete solution. Consequently, the links between syntax and semantic are not completely understood yet, thus the design of syntactic features for the automatic learning of semantic structures requires a remarkable research effort and deep knowledge about the target linguistic phenomena.

The kernel approach, which does not require any noticeable feature design effort, can provide the same accuracy of manually designed features and sometime it can suggest to the designer new solutions to improve the model of the target linguistic phenomenon.

The kernels that we consider in next sections represent trees in terms of their substructures (fragments). Such fragments define the feature space which, in turn, is mapped into a vector space, e.g. $\mathbb{R}^n$. The kernel function measures the similarity between trees by counting the number of common fragments. These functions have to recognize if a common tree subpart belongs to the feature space that we intend to generate. For such purpose, the fragment type needs to be described. We consider three important characterizations: the SubTrees (STs), the SubSet Trees (SSTs) and a new tree class, i.e. the Partial Trees (PTs).

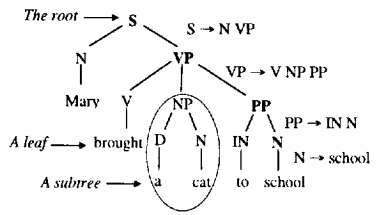

Figure 2.17: A syntactic parse tree.

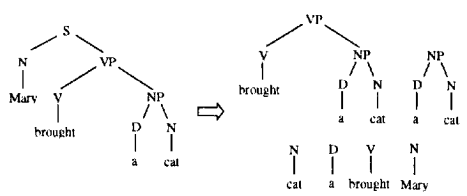

Figure 2.18: A syntactic parse tree with its SubTrees (STs).

2.5.1 SubTree, SubSet Tree and Partial Tree Kernels

Trees are directed, connected acyclic graphs with a special node called root. Their recursive definition is the following: (1) the root node, connected with one or more nodes (called children), is a tree and (2) a child can be a tree, i.e. a SubTree, or a node without children, i.e. a leaf.

In case of syntactic parse trees each node with its children is associated with a grammar production rule, where the symbol at left-hand side corresponds to the parent and the symbols at right-hand side are associated with the children. The terminal symbols of the grammar are always associated with the leaves of the tree. For example, Figure 2.17 illustrates the syntactic parse of the sentence `"Mary brought a cat to school"`.

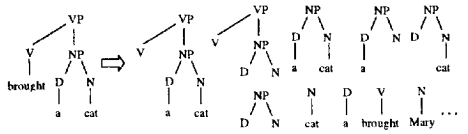

Figure 2.19: A tree with some of its SubSet Trees (SSTs).

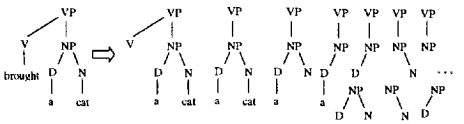

Figure 2.20: A tree with some of its Partial Trees (PTs).

We define as a **SubTree** (ST) any node of a tree along with all its descendants. For example, the line in Figure 2.17 circles the SubTree rooted in the `NP` node. A **SubSet Tree** (SST) is a more general structure which not necessarily includes all the descendants. The only restriction is that an SST must be generated by applying the same grammatical rule set which generated the original tree, as pointed out in [Collins and Duffy, 2002]. Thus, the difference with the SubTrees is that the SST's leaves can be associated with non-terminal symbols. For example, `[S [N VP]]` is an SST of the tree in Figure 2.17 and it has the two non-terminal symbols `N` and `VP` as leaves.

If we relax the constraint over the SSTs, we obtain a more general form of substructures that we defined as **Partial Trees** (PTs). These can be generated by the application of partial production rules of the original grammar. For example, `[S [N VP]]`, `[S [N]]` and `[S [VP]]` are valid PTs of the tree in Figure 2.17.

Given a syntactic tree, we may represent it by means of the set of all its STs, SSTs or PTs. For example, Figure 2.18 shows the parse tree of the sentence `"Mary brought a cat"` together with its 6 STs. The number of SSTs is always higher. For example, Figure 2.19 shows 10 SSTs (out of all 17) of

the SubTree of Figure 2.18 rooted in VP. Figure 2.20 shows that the number of PTs derived from the same tree is still higher (i.e. 30 PTs). These different substructure numbers provide an intuitive quantification of the different information level among the tree-based representations.

2.5.2 The Kernel Functions

The main idea of the tree kernels is to compute the number of the common substructures between two trees T_1 and T_2 without explicitly considering the whole fragment space. For this purpose, we slightly modified the kernel function proposed in [Collins and Duffy, 2002] by introducing a parameters σ which enables the ST or the SST evaluation. For the PT kernel function we designed a new algorithm.

The ST and SST Computation

Given a tree fragment space $\{f_1, f_2, ..\} = \mathcal{F}$, we defined the indicator function $I_i(n)$ which is equal to 1 if the target f_i is rooted at node n and 0 otherwise. It follows that:

$$K(T_1, T_2) = \sum_{n_1 \in N_{T_1}} \sum_{n_2 \in N_{T_2}} \Delta(n_1, n_2) \tag{2.29}$$

where N_{T_1} and N_{T_2} are the sets of the T_1's and T_2's nodes, respectively and $\Delta(n_1, n_2) = \sum_{i=1}^{|\mathcal{F}|} I_i(n_1) I_i(n_2)$. This latter is equal to the number of common fragments rooted at the n_1 and n_2 nodes. We can compute Δ as follows:

1. if the productions at n_1 and n_2 are different then $\Delta(n_1, n_2) = 0$;
2. if the productions at n_1 and n_2 are the same, and n_1 and n_2 have only leaf children (i.e. they are pre-terminals symbols) then $\Delta(n_1, n_2) = 1$;
3. if the productions at n_1 and n_2 are the same, and n_1 and n_2 are not pre-terminals then

$$\Delta(n_1, n_2) = \prod_{j=1}^{nc(n_1)} (\sigma + \Delta(c_{n_1}^j, c_{n_2}^j)) \tag{2.30}$$

where $\sigma \in \{0, 1\}$, $nc(n_1)$ is the number of the children of n_1 and c_n^j is the j-th child of the node n. Note that, as the productions are the same $nc(n_1) = nc(n_2)$.

When $\sigma = 0$, $\Delta(n_1, n_2)$ is equal 1 only if $\forall j \quad \Delta(c_{n_1}^j, c_{n_2}^j) = 1$, i.e. all the productions associated with the children are identical. By recursively applying this property, it follows that the SubTrees in n_1 and n_2 are identical. Thus, Eq. 2.29 evaluates the SubTree (ST) kernel. When $\sigma = 1$, $\Delta(n_1, n_2)$ evaluates the number of SSTs common to n_1 and n_2 as proved in [Collins and Duffy, 2002].

To include the leaves as fragments it is enough to add, to the recursive rule set for the Δ evaluation, the condition:

0. if n_1 and n_2 are leaves and their associated symbols are equal then $\Delta(n_1, n_2) = 1$

We will refer to such extended kernels as ST+bow (*bag-of-words*) and SST+bow. Moreover, we use the decay factor λ as follows[5]:$\Delta(n_x, n_z) = \lambda$ and $\Delta(n_x, n_z) = \lambda \prod_{j=1}^{nc(n_x)} (\sigma + \Delta(c_{n_1}^j, c_{n_2}^j))$.

The Δ computation complexity is $O(|N_{T_1}| \times |N_{T_2}|)$ time as proved in [Collins and Duffy, 2002]. We will refer to this basic implementation as the Quadratic Tree Kernel (QTK).

The PT Kernel Function

The evaluation of the Partial Trees is more complex since two nodes n_1 and n_2 with different child sets (i.e. associated to different productions) can share one or more children, consequently they have one or more common substructures, e.g. `[S [DT JJ N]]` and `[S [DT N N]]` have the `[S [N]]` (2 times) and the `[S [DT N]]` in common.

To evaluate all possible substructures common to two trees, we can (1) select a child subset from both trees, (2) extract the portion of the syntactic rule that contains such subset, (3) apply Eq. 2.30 to the extracted partial productions and (4) sum the contributions of all children subsets.

Such subsets correspond to all possible common (non-continuous) node subsequences and can be computed efficiently by means of sequence kernels [Lodhi *et al.*, 2000]. Let $\vec{J_1} = (J_{11}, .., J_{1r})$ and $\vec{J_2} = (J_{21}, .., J_{2r})$ be the

[5] To have a similarity score between 0 and 1, we also apply the normalization in the kernel space, i.e. $K_{normed}(T_1, T_2) = \frac{K(T_1, T_2)}{\sqrt{K(T_1, T_1) \times K(T_2, T_2)}}$.

index sequences associate with the ordered child sequences of n_1 and n_2, respectively, then the number of PTs is evaluated by the following Δ function:

$$\Delta(n_1, n_2) = 1 + \sum_{\vec{J}_1, \vec{J}_2, l(\vec{J}_1)=l(\vec{J}_2)} \prod_{i=1}^{l(\vec{J}_1)} \Delta(c_{n_1}^{\vec{J}_{1i}}, c_{n_2}^{\vec{J}_{2i}}) \quad (2.31)$$

where $l(\vec{J}_1)$ indicates the length of the target child sequence whereas $\vec{J}_{1i}$ and $\vec{J}_{2i}$ are the i^{th} children in the two sequences. We note that:

1. Eq. 2.31 is a convolution kernel [Haussler, 1999] (see the definition in Section 2.4.4).

2. Given a sequence of common children, $\vec{J}$, the product in the Eq. 2.31 evaluates the number of common PTs rooted in n_1 and n_2. In these PTs, the children of n_1 and n_2 are all and only those in $\vec{J}$.

3. By summing the products associated with each sequence we evaluate all possible PTs (the root is included).

4. Tree kernels based on sequences were proposed in [Zelenko *et al.*, 2003; Culotta and Sorensen, 2004] but they do not evaluate all tree substructures, i.e. they are not convolution kernels.

5. We can scale down the contribution from the longer sequences by adding a decay factor λ:

$$\Delta(n_1, n_2) = \lambda + \sum_{\vec{J}_1, \vec{J}_2, l(\vec{J}_1)=l(\vec{J}_2)} \lambda^{d(\vec{J}_1)+d(\vec{J}_2)} \prod_{i=1}^{l(\vec{J}_1)} \Delta(c_{n_1}^{\vec{J}_{1i}}, c_{n_2}^{\vec{J}_{2i}})$$

where $d(\vec{J}_1) = \vec{J}_{1l(\vec{J}_1)} - \vec{J}_{11}$ and $d(\vec{J}_2) = \vec{J}_{2l(\vec{J}_2)} - \vec{J}_{21}$.

Finally, as the sequence kernels and the Eq. 2.30 can be evaluated efficiently, we can carry out also Eq. 2.31, efficiently. Let N and M be the maximum number of children in the nodes of T_1 and T_2, respectively. The worst case complexity is the product between the complexities of the non-continuous sequence kernel, i.e. $O(N \times M^2)$ and the SST kernel, i.e. $O(|N_{T_1}| \times |N_{T_2}|)$.


```
function Evaluate_Pair_Set(Tree T1, T2) returns NODE_PAIR;
LIST L1,L2;
NODE_PAIR Np;
begin
  L1 = T1.ordered_list;
  L2 = T2.ordered_list; /*the lists were sorted at loading time*/
  n1 = extract(L1); /*get the head element and*/
  n2 = extract(L2); /*remove it from the list*/
  while (n1 and n2 are not NULL)
    if (production_of(n1) > production_of(n2))
      then n2 = extract(L2);
      else if (production_of(n1) < production_of(n2))
        then n1 = extract(L1);
        else
          while (production_of(n1) == production_of(n2))
            while (production_of(n1) == production_of(n2))
              add(<n1, n2>, Np); n2=get_next_elem(L2);
                    /*get the head element and move the pointer
                    to the next element*/
            end
            n1 = extract(L1); reset(L2); /*set the pointer at the first element*/
          end
  end
  return Np ;
end
```

Table 2.4: Pseudo-code for Fast Tree Kernel (FTK) evaluation.

2.5.3 A Fast Tree Kernel Computation

The worst case computation time of the ST and SST tree kernels is quadratic in the number of nodes of the parse trees, i.e. the number of pairs of the two evaluating trees. Nevertheless, as it was observed in [Collins and Duffy, 2002], many pairs contain two nodes associated with two different production rules; for them we have a null Δ function value. Thus, to reduce the computation time of two trees T_1 and T_2, we can consider only the following pair set: N_p $=\{<n_1, n_2> \in N_{T_1} \times N_{T_2} : p(n_1) = p(n_2)\}$, where $p(n)$ is the production rule at node n.

We can efficiently build N_p by: (1) extracting the lists, L_1 and L_2, of the production rules from T_1 and T_2, (2) sort them in the alphanumeric order and (3) scan them to find the node pairs $<n_1, n_2>$ such that $p(n_1) = p(n_2) \in L_1 \cap L_2$. Step 3 may require only $O(|N_{T_1}| + |N_{T_2}|)$ time, but, if $p(n_1)$ appears r_1 times in T_1 and $p(n_2)$ is repeated r_2 times in T_2, we need to consider $r_1 \times r_2$ pairs. The formal algorithm is given in Table 2.4.

Note that: (1) the list sorting can be carried out in $O(|N_{T_1}| \times log(|N_{T_1}|))$ during the data preparation steps and (2) the algorithm's worst case occurs when the parse trees are both generated using a single production rule, i.e. the two internal **while** cycles carry out $|N_{T_1}| \times |N_{T_2}|$ iterations. On the contrary, the kernel time complexity for two identical parse trees is still linear provided that in their generation few productions are used many times.

A similar algorithm can be used also to improve the speed of the PT evaluation. As the PT kernel operates on nodes, instead of on productions, we substituted the list of productions with the lists of nodes. The number of pairs with identical nodes is higher than the number of pairs with identical productions, thus, the PT computation receives a lower benefit from our Fast Tree Kernel (FTK) algorithm.

2.6 Conclusions

In this chapter we have shown the basic approaches of traditional machine learning such as *Decision Trees* and *Naive Bayes* and we have introduced the basic concepts of the statistical learning theory such as the characterization of learning via the PAC theory and the Perceptron algorithm. In particular, a simplified theory of Support Vector Machines ($SVMs$) along with kernel methods have been presented.

These latter let the reader foreseen all the potential applications of $SVMs$: Polynomial, String, Lexical and Tree kernels appears formidable tools to deal with natural language applications.

Chapter 3

Automated Text Categorization

This chapter accurately describes the phases concerning the design and implementation of the TC models introduced in Section 1. As the analysis of all methods developed in IR is beyond the purpose of this book, we only provide some of the most representative examples of corpora, weighting schemes, profile-based classification models, score adjustment techniques, inference policies and performance measurements (sections 3.1, 3.2, 3.3 and 3.4, respectively).

One of the interesting contributions of this chapter is the study on the Rocchio classifier parameterization as it shows how TC models should be tuned to improve their accuracy. In particular, a model for the automatic selection of Rocchio parameters is presented in Section 3.5. The important aspect of this approach is that *parameter tuning* is carried out in analogy with traditional feature selection techniques. This allows the reader to acquire a unified view of the relationship between weighting schemes and feature selection.

Additionally, by adopting the above model as a test case, the chapter describes how new TC approaches should be experimented to assess their properties and relevance in the current literature. An estensive evaluation is discussed in a didactic perspective: it involves the comparison (Section 3.6) among the basic Rocchio classifier, the Parameterized Rocchio Classifier (PRC), which is the proposed model and Support Vector Machines, which represent the best figure classifier on several corpora.

Finally, Section 3.7 summarizes the main concepts behind the design of TC systems.

3.1 Document Preprocessing

Section 2.2 has introduced that, to design automatic text classifiers, we need a sufficient number of documents labeled in the target category collection. Fortunately, as many press Companies (e.g. News Agencies) and scientific fields (e.g. Medical area) widely apply categorization schemes to organize the textual information related to their activities, many labeled corpora are available for research purposes.

Next sections show some of the most famous TC benchmarks, the format in which they are usually made available to the researchers and the pre-processing steps needed before application of statistical TC models.

3.1.1 Corpora

Experiments in Text Categorization aim to evaluate the system accuracy and to compare it against previous work. For this reason, several TC benchmarks have been built. Hereafter, we describe the most famous TC corpora, highlighting their main characteristics.

- The *Reuters-21578*[1] collection Apté split. It includes 12,902 documents for 90 classes, with a fixed split between *test set* (here after RTS) and the learning data LS (3,299 vs. 9,603). As stated in Section 1.2 different Reuters versions [Yang, 1999; Sebastiani, 2002] have been used for testing TC algorithms. *Reuters-21578* is the most used version in the TC literature, thus, it can be considered the main reference collection. A description of some categories of this corpus is given in Table 3.1.

- The Ohsumed collection[2], including 50,216 medical abstracts. Given the large number of documents and categories, many researchers have used subsets of documents and categories for their experiments. For example, in [Joachims, 1998] only the first 20,000 documents categorized under the 23 *MeSH diseases*[3] categories have been used. Although, literature results are not comparable as they refer to different document

[1] Once available at `http://www.research.att.com/~lewis` and now available at `http://kdd.ics.uci.edu/databases/reuters21578/reuters21578.html`.

[2] It has been compiled by William Hersh and it is currently available at `ftp://medir.ohsu.edu/pub/ohsumed`.

[3] Table 3.2 gives a description of some the disease categories.

subsets, they can give an indication of Ohsumed classification complexity. For instance, no TC research on Ohsumed reports an accuracy over 70% thus this corpus is more *difficult* than Reuters, for which classifiers reach 86% of accuracy[4].

- The *20NewsGroups*[5] (20NG) corpus contains 19,997 texts from the Usenet newsgroup collection, divided in 20 categories. Only the subject and the body of each message should be considered as being part of the experimenting documents. It is worth to note that some of the newsgroups are very closely related to each other (e.g., *IBM computer system hardware/Macintosh computer system hardware*), while others are highly unrelated (e.g., *misc forsale/social religion and christian*). This corpus is different from Reuters and Ohsumed because it includes a larger vocabulary and words typically assume more meanings in different contexts. Additionally, the stylistic writing is very different as it relates to *e-mail* dialog rather than technical summaries in Ohsumed or event reports of the Reuters News.

- The ANSA collection, which includes 16,000 news items in Italian from the ANSA news agency, classified in 8 categories (2,000 documents each). ANSA categories relate to typical newspaper contents (e.g., Politics, Sport and Economics). It is worth noting that this collection is closer to operational scenarios: some documents are not correctly assigned to the categories and many others are repeated more than once. Moreover, it is possible to find some English and German texts mixed with those in Italian.

The documents of the above corpora are made available in various formats. We have notice four main data structures:

- *The SMART format*, in which the documents for all categories are given in a unique file. Each document has a header. This allows the designer to extract its title, its *id* and its set of categories [6].

- *The SGML format*, in which tags, e.g., `<title>` and `<\title>`, allow

[4] More precisely, they achieve 86% of microaverage f-measure, which will be defined in Section 3.4.2.

[5] Available at *http://www.ai.mit.edu/people/jrennie/20Newsgroups/*.

[6] In this format Yang made available the SMART, the Reuters and Ohsumed corpora.

Table 3.1: Description of some Reuters categories

Category	Description
Acq	Acquisition of shares and companies
Earn	Earns derived by acquisitions or sells
Crude	Crude oil events: market, Opec decisions,..
Grain	News about grain production
Trade	Trade between companies
Ship	Economic events that involve ships
Cocoa	Market and events related to Cocoa plants
Nat-gas	Natural Gas market
Veg-oil	Vegetal oil market

Table 3.2: Description of some Ohsumed categories

Category	Description
Pathology	Pathological Conditions
Cardiovascular	Cardiovascular Diseases
Immunologic	Immunologic Diseases
Neoplasms	Neoplasms
Digestive Sys.	Digestive System Diseases
Hemic & Lymph.	Hemic & Lymphatic Diseases
Neonatal	Neonatal Disorders & Abnormalities
Skin	Skin & Connective Tissue Diseases
Nutritional	Nutritional & Metabolic Diseases
Endocrine	Endocrine Diseases
Disorders	Disorders of Environmental Origin
Animal	Animal Diseases

the user to extract the needed information. The Lewis' Reuters version that includes also the Apté split is provided in this format.

- *The raw format*: this is the most usual format in real scenario applications. Users, e.g. newspaper agencies, provide database dumps without specific data structures and no markup. The results are collections of documents, one file or directory for each category. If files are used, documents are simply separated by one or more empty lines. Additional information that indicates the multi-labeled documents, is needed.

Whatever the format of the documents is, a pre-processing step is needed to convert them in a format suitable for the application of statistical modules.

3.1.2 Tokenization, Stoplist and Stemming

In this phase, relevant features are extracted from documents. Usually, all words and other alphanumeric strings (i.e. tokens) are considered in such extraction. There are many ways to design tokens. For example:

(a) by selecting all character sequences separated by space. This means that alphanumeric strings like *Alex69* as well as more generic strings like *tokenization_dir* are included in the resulting feature set.

(b) by considering alphabetic or numeric character sequences separated by all other characters. In this case, the feature set contains only the usual words and numbers. The size of this feature set is smaller than the set of the point (a).

Afterwards, the set of tokens is extracted, it can be improved by removing features that occur uniformly across all categories. For this purpose a word list, called *stoplist*, is prepared and used in the preprocessing phase. It contains irrelevant features like the *function words*, e.g. *What*, *Who*, *at*, *he* and *be*. Their removal improves at least the efficiency of the target TC models.

Additionally, as some concepts may be expressed by different words (see Section 4), the Recall of TC systems can be enhanced by normalizing the surface word forms of the same concept. For this purpose, there are language dependent methods like word stemming. Word stemming is based on two stages: suffix stripping and conflation. Suffix stripping can be achieved by applying a set of rules. For example *biology*, *biologist*, *biologists* can be reduced to *biology*. In this phase, some errors may occur as there are always some exceptions to the rules. The error rate of word stemming has been measured around 5% [Van Rijsbergen, 1979]. Stemming, is usually applied to the design of text classifiers nevertheless, there is no study that proves the superiority of the stemmed words over the simple words.

Another important phase of TC pre-processing is the feature selection. As it is done for the stoplist, a set of non-informative words are detected and

removed from the feature set. The main difference with the stoplist technique is that the selection of the word list is fully automatic.

3.1.3 Feature Selection

The previous section has shown that in TC, normally, the cardinality of the feature set is equal to the number of corpus tokens, i.e. hundreds of thousands. This size prevents the applicability of many learning algorithms. For example, the neural networks designed for TC use a number of nodes that is quadratic in the number of features. Consequently, they cannot be applied to large corpora. Feature Selection techniques have been introduced to reduce the dimensionality of feature spaces.

Automated feature selection methods remove non-informative terms according to corpus statistics and rebuild new (i.e. reduced or re-mapped) feature spaces. Such statistics are based on the distribution of features f and categories c in the corpus documents. More in detail, they are based on four different counts:

- A is the number of documents in which both f and c occur, i.e. (f, c);
- B is the number of documents in which only f occurs, i.e. $(f, \bar{c})$;
- C is the number of documents in which only c occurs, i.e. $(\bar{f}, c)$;
- D is the number of documents in which neither f nor c occur, i.e. $(\bar{f}, \bar{c})$;
- N is the total number of documents, i.e. $A + B + C + D$.

The above quantities can be used to estimate the Chi-square statistics (χ^2), the Pointwise Mutual Information (PMI), or the Mutual Information (MI) [Yang and Pedersen, 1997] as follows:

$$\chi^2(f,c) = \frac{N \times (AD - CB)^2}{(A+C)(B+D)(A+B)(C+D)}$$

$$PMI(f,c) = log\frac{P(f,c)}{P(f) \times P(c)}$$

$$MI(f) = -\sum_{c \in \mathcal{C}} P(c)log(P(c)) + P(f)\sum_{c \in \mathcal{C}} P(c|f)log(P(c|f)) + P(\bar{f})\sum_{c \in \mathcal{C}} P(c|\bar{f})log(P(c|\bar{f}))$$

where

- $P(f, c)$ is the probability that f and c co-occurs and can be estimated by A/N;
- $P(f)$ is the probability of f, estimated by $(A + B)/N$;
- $P(c)$ is the probability of f, estimated by $(A + C)/N$;
- $P(c|f)$ is the probability of c by considering only the documents that contain f. It can be estimated by $\frac{P(f,c)}{P(f)}$.
- $P(\bar{f})$ is the probability that f does not occur, estimated by $(C + D)/N$;
- $P(c|\bar{f})$ is the probability of c by considering only the documents that do not contain f. It can be estimated by $\frac{P(\bar{f},c)}{P(\bar{f})}$. In turn, $P(\bar{f}, c)$ is estimated by C/N.
- $\mathcal{C}$ is the collection of categories, i.e. $\{c_1, c_2, ..., c_n\}$. Note that PMI and χ^2 are defined on only two categories, i.e. c and *not* c whereas MI can be evaluated on $n > 2$ categories[7].

For example, we can apply the above formulas to evaluate the PMI as follows:

$$PMI(f,c) = \frac{A \times N}{(A+C)(A+B)}$$

Once the PMI is computed for all features, we can rank the features according to PMI values and remove those that have a lower position. Thus, when we design a binary classifier for the category c, we will include only the feature selected by PMI in the document representation.

Unfortunately, if we change the target category c, the value of PMI or χ^2 for a given feature f will change as well. Consequently, features will be ranked differently in different categories. This prevents us to use a uniform feature set for all the categories and it will not be possible to train a multiclassifier in the reduced space. Hence, we need to combine the different selector values in a unique score. The following equations describe two alternatives (for PMI and

[7]Note that the MI measures the difference in entropy of the class distribution with and without the knowledge about the feature f. This was discussed in Section 2.1.1 about the training algorithm of Decision Trees.

χ^2) to design *global* selectors for a category collection C [Yang and Pedersen, 1997]:

$$PMI_{max}(f) = \max_{c \in C} \quad MI(f,c)$$

$$PMI_{avg}(f) = \sum_{c \in C} P(c) \times MI(f,c)$$

$$\chi^2_{max}(f) = \max_{c \in C} \quad \chi^2(f,c)$$

$$\chi^2_{avg}(f) = \sum_{c \in C} P(c) \times \chi^2(f,c)$$

Each of the above selectors produces a ranking of features f that depends on all the classes. For example, the $PMI_{avg}(f)$ combines different $PMI(f,c)$ by means of their average. As pointed out in [Yang and Pedersen, 1997] χ^2 and MI provide the *best* selectors for the feature set reduction and the increase of text classifier accuracy. Also in [Dumais *et al.*, 1998], MI was found to be the best selector for SVMs.

Moreover, it should be noted that the ranking is uniform across all the categories. Hence, It may filter out features which are not globally informative but are very relevant for few classes. The global selection cannot optimize the classification of a specific category. Consequently, binary classification along with a local feature selection seems to be more promising.

The scores provided by feature selectors can be used as weights of features within the vector representation of documents. Traditionally in IR, when the purpose is to assign a weight to a feature rather than remove it, the selectors are called weighting schemes. Several weighting schemes based on specific heuristics have been designed in IR. They are used to improve the learning algorithm generalization ability.

3.2 Weighting Schemes

Weighting schemes are used in IR to assign a higher importance to terms that have a higher *indexing ability* with respect to a target document collection. They help IR systems to rank the retrieved documents according to the expected user relevance. Traditionally, weights are heuristic combinations of different corpus statistics, e.g. *Term Frequency* (TF) and *Inverse Document Frequency* (IDF). The former quantity attempt to model the importance of a feature inside the document: if a word is repeated many times in a document,

it should be important for such document. The latter quantity is used to assign a global importance: the more a term is frequent, the less its capacity of selecting topic information is.

Many weighting approaches have been studied in [Salton, 1989] for document retrieval. The results show that retrieval systems applied to different domains benefit from the use of specific weighting schemes. Similar weighting strategies and conclusions have been obtained in TC since similar justifications for the design of weighting schemes can be derived.

First, features (e.g. words) extracted from document content occur with highly variable frequency in texts of different categories. Second, the more a feature is important for a class, the higher its frequency (i.e. TF) is inside the documents of such class. Therefore, the occurrence counts can be used to indicate relevance in content representation. Third, as highly frequent features can be evenly distributed throughout the different categories, frequency alone is not a good model for feature weighting. The inverse documents frequency[8] (IDF) can be used to overcome such a problem since it gives higher weights to features occurring in few documents. Finally, optimal weights can be obtained with the $TF \cdot IDF$ product, and, to allow the system to carry out a direct comparison between two documents, weights are usually normalized with respect to the document size [Salton and Buckley, 1988].

Once document weights are available, several techniques can be applied to extract the category profile weights. For example, to assign to a feature f a category weight, we can sum the weights that f assumes in all the documents of such category.

3.2.1 Document Weighting

To model document weighting schemes, we need to define a few specific parameters. Given a set of training documents, T (i.e. the *training set*), a feature $f \in \{f_1, ..., f_n\}$, a generic document $d \in T$ and the target set of classes C_1, C_2, ..., let the following notations express:

- M, the number of documents in the *training set*,
- n_f, the number of documents in which the feature f appears and

[8] The logarithm of the fraction between the total number of documents and the number of documents in which the feature occurs.

- o_f^d, the occurrences of the feature f in the document d (i.e. the term frequency).

Let us consider the $IDF \cdot TF$ as the first weighting scheme. This is the traditional weighting strategy used in $SMART$ [Salton, 1991]. Given the $IDF(f) = log(\frac{M}{n_f})$, the weight of the feature f in the document d is:

$$w_f^d = \frac{o_f^d \cdot IDF(f)}{\sqrt{\sum_{r=1}^{n}(o_r^d \cdot IDF(r))^2}} \tag{3.1}$$

A second weighting scheme (used in [Ittner *et al.*, 1995]) is the $log(TF) \times IDF$. It uses the logarithm of o_f^d as follows:

$$\grave{o}_f^d = \begin{cases} 0 & \text{if } o_f^d = 0 \\ log(o_f^d) + 1 & \text{otherwise} \end{cases} \tag{3.2}$$

Accordingly, the document weights will be:

$$w_f^d = \frac{\grave{o}_f^d \cdot IDF(f)}{\sqrt{\sum_{r=1}^{n}(\grave{o}_r^d \cdot IDF(r))^2}} \tag{3.3}$$

The third weighting scheme refers to $TF \cdot IWF$. It introduces new corpus-derived parameters:

- N, the number of the total feature occurrences,
- N_f, the total occurrences of the target feature f.

By using the above quantities, the IWF (Inverse Word Frequency) can be defined as:

$$log(\frac{N}{N_f})$$

IWF is used similarly to IDF in the following weighting formula:

$$w_f^d = \frac{o_f^d \cdot (IWF(f))^2}{\sqrt{\sum_{r=1}^{n}(o_f^d \cdot (IWF(r))^2)^2}} \tag{3.4}$$

The above scheme has two major differences with respect to the traditional $TF \cdot IDF$ weighting strategy (i.e. Eq. 3.1).

(1) The IWF [Basili *et al.*, 1999] is used in place of the IDF. Its role is similar to IDF, as it penalizes very highly frequent (and less meaningful) terms (e.g. *say*, *be* and *have*).

(2) The other aspect is the application of the square function to the IWF. As the product $IWF \cdot o_f^d$ is too biased by the feature frequency o_f^d, IWF squared is preferred. A similar adjustment technique was proposed in [Hull, 1994].

3.2.2 Profile Weighting

Once, the appropriate document weighting scheme has been chosen, we can apply several methods to obtain the feature weights in the class profile. The simplest one was designed for the $SMART$ system [Salton and Buckley, 1988]. It is just the summation of the weights that each feature f assumes in different C_i's documents:

$$W_f^i = \sum_{d \in C_i} w_f^d. \tag{3.5}$$

In this representation a profile is considered as a *macro document* made by all the features contained in the documents of the target class. Note that: (a) the above model generates three different profile weighting models when the equations 3.1, 3.3 and 3.4 are used as document weighting schemes, respectively; (b) Eq. 3.5 does not depend on negative examples, i.e. it does not consider the weights that a feature assumes in the other classes.

On the contrary, the classifier based on the *Rocchio's formula* [Rocchio, 1971] is another common weighting scheme which computes profile weights by also taking into account negative information as shown by the next equation:

$$W_f^i = \max\left\{0, \frac{\beta}{|C_i|}\sum_{d \in C_i} w_f^d - \frac{\gamma}{|\bar{C}_i|}\sum_{d \in \bar{C}_i} w_f^d\right\} \tag{3.6}$$

where $\bar{C}_i$ is the set of documents not belonging to C_i. Eq. 3.6 assigns the weight W_f^i to the feature f in the profile i by (a) summing the weights that f assumes in the documents of the class i and (b) subtracting the weights that f assumes in all the other documents. The parameters β and γ control the relative impact of positive and negative examples on the profile weighting.

It is worth to note that the $SMART$ weighting scheme is a special case of the Rocchio's formula in which γ is set to 0 and no normalization is applied. However, the profiles created via $SMART$ procedure (i.e. macro documents) are conceptually different from those designed by the Rocchio's formula (i.e. centroids among positive and negative documents).

After the weights are assigned to the features in both documents and profiles, we can use such numerical quantities to estimate the similarity between documents and categories. The next section introduces the basic approaches used for such purpose.

3.3 Modeling Similarity in Profile-based Text Classification

Given a feature set and the weighting schemes, the corpus documents can be represented by vectors in $\mathbb{R}^n$ as follows:

$$\vec{d} =< w^d_{f_1}, ..., w^d_{f_n} >$$

The category profiles also have a vector representation:

$$\vec{C}_i =< W^i_{f_1}, ..., W^i_{f_n} >$$

To complete such vector space, we need to add a metric function which, by measuring the distance between two vectors, computes the similarity between the target document and category, i.e. between $\vec{d}$ and $\vec{C}_i$. For such purpose, we can adopt the usual dot product:

$$s_{id} = \vec{C}_i \cdot \vec{d} = \sum_f W^i_f w^d_f \tag{3.7}$$

If we assume that the dot product can effectively measure the similarity between document and category vectors, a classification system should rely on this similarity scores to accept or not the document in the category. By applying a simple threshold over the scores, we obtain the traditional linear classifier, i.e. the equation

$$\phi(\vec{d}) = \begin{cases} 1 & \vec{C}^i \cdot \vec{d} > th \\ -1 & otherwise \end{cases} \tag{3.8}$$

which is equivalent to the common linear decision function

$$h(\vec{d}) = sgn(\vec{C}^i \cdot \vec{d} + b)$$

when the threshold th is equal to the constant $-b$. We note that:

First, $\vec{C}^i$ (e.g. produced by the Rocchio's formula) has the same role of the gradient $\vec{w}$ of the $SVMs$ in the classification function, e.g. Eq. 2.8 in Section 2.3.1. Consequently, $\vec{C}^i$ can be seen as product of learning procedures, e.g. based on the Rocchio heuristic. Conversely, the Support Vector learning algorithms can be seen as weighting schemes for the category profile.

Second, the Rocchio heuristic does not assure that the training set will be separated by the hyperplane associated with $\vec{C}^i$. Hence, the Rocchio classifier will in general produce errors on the classification of the training set.

Finally, the application of a threshold on the similarity scores is not the only possible approach. For example, some methods select for a document d the first k categories ranked by the scores that they have with d. The problem of such approach relates to the observation that scores between different documents and profiles are in general not comparable. Feature and weight distributions extracted from different categories produce large weight variance across different documents and profiles. Unfortunately, there is no analytical result that links together the dot products (scores) produced by different categories with the document sets (Eq. 3.7).

To tackle this problem several techniques have been proposed. They change the vector space by re-scaling the scores and projecting them in other spaces where they are comparable. This phase has no definitive name associated with it. Hereafter, we will refer to it as *score adjustment*. In the next sections, we introduce and discuss two effective adjustment methods.

3.3.1 Similarity based on Logistic Regression

A technique to re-map (adjust) scores is based on the Logistic Regression (LR) function. When LR is applied to scores s_{di} an actual estimate of $P(C_i|d)$, i.e. the probability that a document d belong to the class C_i, is obtained [Ittner *et al.*, 1995]. In brief, the LR score adjustment algorithm is the following:

1. All pairs $<s_{di}, belong\_flag>$ for each training document d and for each target class i are evaluated: $belong\_flag$ is set to 1 *iff* $d \in C_i$, and to 0 otherwise.

2. The derived pairs are given as input to the Logistic Regression algorithm, which produces two values, α_i and β_i. These are the parameters of the function:

$$F(\alpha_i, \beta_i, s_{di}) = \frac{e^{\alpha_i + \beta_i \cdot s_{di}}}{1 + e^{\alpha_i + \beta_i \cdot s_{di}}},$$

which is a good approximation of $P(C_i|s_{di})$. The LR function produces the conditional probability $P(d \in C_i|s_{di}) \simeq P(d \in C_i|d, C_i)$.

Inference strategies can be more effectively applied on the $P(C_i|s_{di})$ values as they are distributed throughout all the classes more homogeneously than the source values s_{di}. Moreover, it is worth noting that the logistic function is monotonic ascending. This implies that when we focus on a class C_i the rankings of documents according to $P(C_i|d)$ and to s_{di} are the same.

3.3.2 Similarity over differences: Relative Difference Scores

The LR score adjustment method allows us to *better* rank the scores originated from different classes and this may greatly improve the overall system accuracy. This is especially useful for text classifiers based on the *SMART* weighting scheme (Eq. 3.5), which do not use negative examples. In fact, a way to consider the negative information is to use the scores obtained from the other categories.

Let us suppose that an "odd" test document has low scores with all classes C_i (i.e. $d : s_{di} << 1$ for each category i). By considering only the absolute score value, such document will be rejected by all the classifiers. On the contrary, if the classifiers make their decisions by considering all the scores, e.g. by selecting the categories associated with the k highest scores, the "odd" documents will be assigned to some categories.

However, LR shows an inherent weakness since it does not help to re-rank documents within a single target class. For example, let us examine the situation described in Table 3.3 in which we have two classes, three documents and their scores. Suppose also that the classifier makes its decisions by means of two thresholds (one for each category).

Note that, (a) the document d_2 is "odd" as it shows a low similarity with both the two classes and (b) the other two documents, d_1 and d_3, should be accepted as members of C_1 and C_2, respectively. Thus, whatever threshold is selected for the categories, LR cannot re-rank the documents to make accept

Table 3.3: Scores for a simple Classification Inference case

Document Index	C_1 (score $s_{d,1}$)	C_2 (score $s_{d,2}$)	*Gold Standard*
d_1	7	1	C_1
d_2	0.01	0.8	C_2
d_3	2	5	C_2

d_2 and discard d_1 since LR is a monotonic function. This would prevent to accept d_2 in class C_2, although s_{22} is about eighty times higher than s_{21} !

What we need is a technique able to produce a ranking among documents influenced by their general behavior, i.e. according to their similarity with respect to all classes. If we could re-rank documents according to this cross-categorical information, we would have a ranking for the class C_2 like, $d_3 \succeq d_2 \succeq d_1$. This has to violate the monotonicity of the LR function (as $s_{12} > s_{22}$).

To overcame this problem, a score adjustment technique based on the *differences among similarity scores* has been defined in [Basili *et al.*, 2000] (hereafter called RDS). RDS increases or decreases the similarity scores of a category according to the scores provided by the other categories. Instead of the s_{di}, we derive the m_{di} values which express the average difference between the score of the correct (e.g. i-th) class and the remaining classes. Formally, given a training document d and a class C_i, m_{di} is computed by:

$$m_{di} = \frac{\sum_{j=1}^{n} s_{di} - s_{dj}}{n - 1} \tag{3.9}$$

Eq. 3.9 is the score adjustment approach that we call RDS. Notice that in the simple case defined in Table 3.3, the following values are obtained: $m_{12} = -6$, $m_{22} = 0.79$ and $m_{32} = 3$ correctly suggesting the expected ranking $d_3 \succeq d_2 \succeq d_1$ for class 2. RDS produces scores that explicitly depend on the negative information expressed by documents not belonging to the target class. This can result in an improvement of the TC accuracy.

3.4 Inference Policy and Accuracy Measures

When the similarity scores between a newly incoming document d and the different profiles are available, the assignment of the category labels C_i can

be carried out. The decision function $h : D \rightarrow 2^{\{C_1,...,C_n\}}$ can be defined in terms of the similarity scores s_{di}, i.e. as a r-ary real-valued function $h' : \mathbb{R}^r \rightarrow 2^{\{C_1,...,C_n\}}$. As h' can be applied to a whole set of documents (TS), two different groupings of scores s_{di} are possible: by classes (index $i = 1, ..., n$) or by documents (d such that $d \in TS$). These are defined in [Sebastiani, 2002] as:

- $r = n$: the set of scores that one target document d assumes in all the n categories, i.e. $\{s_{d1}, ..., s_{dn}\}$. This is referred to the *document pivoted classification scheme.*

- $r = |TS|$: the set of scores that *all documents* in TS have in the target category C_i, i.e. $\{s_{1i}, s_{2i}, ..., s_{|TS|i}\}$. This is referred to the *category pivoted classification scheme.*

The accuracy of h can be measured by using the correct category assignments available in the manually labeled TS. Let us refer to such correct labels as the gold standard $GS(TS)$. The differences between the outcome of $h(d)$ and the categories in $GS(d)$ is usually measured by one or more numeric values. These are computed by counting the number of *correct*, *wrong*[9] categories $h(d)$ according to $GS(d)$.

The next sections give details on the possible inference policies embodied by h as well as on the definition of accuracy measurements.

3.4.1 Inference Policy

A decision function h should select the categories which have the highest values among the score groups (e.g. $\{s_{d1}, ..., s_{dn}\}$). This is usually carried out in one of the following strategies [Yang, 1999]:

- probability threshold ($Scut$): for each class C_i a threshold σ_i is adopted such that $C_i \in f(d)$ only if its membership score s_{di} is over σ_i (*category pivoted classification scheme*, i.e. $d \in TS$). The threshold σ_i is an upper limit to the risk of misclassification and has a probabilistic nature.

- fixed threshold ($Rcut$) is based on the assumption that k is the average number of classes assigned to a generic document d. This can be observed usually over the *training set*. Accordingly, $C_i \in h(d)$ only if C_i is one of the first k classes in the ranking evaluated with the s_{di} positive scores (*document pivoted classification scheme*, i.e. $i = 1, .., n$).

- proportional threshold ($Pcut$): a percentage p of the TS documents which have the highest scores with C_i are categorized in C_i (*category pivoted classification scheme*). p is usually estimated from the *training set*, i.e. $p = \frac{C_i}{|C_i \cup \bar{C}_i|}$, where C_i are the documents in the class i and $\bar{C}_i$ are all the other documents.

[9] Notice that an empty set of values output by $h(d)$ corresponds to "*don't know*" choices, i.e. no category is provided for d.

3.4.2 Accuracy Measurements

In TC, several accuracy measures have been proposed, each one with inherent advantages and disadvantages as well. The *error rate* is the ratio between the number of documents not correctly categorized and the total number of documents. According to the above definition, if the *test set* includes a small percentage of documents labeled under a given category, a trivial classifier which rejects all documents of that category will obtain a very low error rate (i.e. a good performance), at least with respect to that category.

Two other measures, i.e. *Precision* and *Recall*, are not affected by such limitation. Given a specific category C_i, their technical definition can be stated in terms of three quantities:

- (*True Positives*) the correct documents found by the decision function, a_i, i.e. the number of documents $d \in TS$ such that $C_i \in h(d)$ **and** $C_i \in GS(d)$.

- (*False Positives*) the number of incorrect documents, b_i, i.e. the number of documents $d \in TS$ such that $C_i \in h(d)$ **and** $C_i \notin GS(d)$.

- (*False Negatives*) the number of not retrieved documents, c_i, i.e. the number of documents $d \in TS$ such that $C_i \notin h(d)$ **and** $C_i \in GS(d)$.

The *Precision* and *Recall* are defined by the above counts:

$$Precision_i = \frac{a_i}{a_i + b_i} \tag{3.10}$$

$$Recall_i = \frac{a_i}{a_i + c_i} \tag{3.11}$$

Both scores depend on inference policies, e.g. they depend on the threshold discussed in the previous section, and are in general inversely proportional.

When the threshold σ_i increases, $Precision$ also increases while $Recall$ tends to decrease and vice versa. This variability between $Recall$ and $Precision$ makes difficult to compare two different classifiers: one could reach the highest *Recall* while the other could achieve the highest *Precision*. Thus, to get a single performance measure, the *Breakeven point* (BEP) is widely adopted.

BEP is the point in which $Recall$ and $Precision$ are equal. It is estimated iteratively by increasing the threshold from the value of Recall = 1 (e.g. $\sigma_i = 0$) until $Precision \leq Recall$. The major problem is that the correct BEP score could not exist (i.e. for no value of the threshold $Recall = Precision$). In this case, a conclusive estimation is the mean between the $Recall$ and $Precision$ (interpolated BEP) such that $|Precision - Recall|$ is the minimum with respect to all the threshold values. However, even this may result artificial [Sebastiani, 2002] when *Precision* is not enough *close* to $Recall$.

The F_1-measure improves BEP definition by imposing the harmonic mean between $Precision$ and $Recall$ as follows:

$$F_1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{3.12}$$

F_1 outputs a more reliable value especially when $Recall$ is highly "*different*" from $Precision$. For example, with a *Precision* of .9 and a *Recall* of .001 the interpolated BEP is the average, i.e. 0.45 while the F_1 is 0.002. This latter corresponds to a more realistic performance indication. However, when comparable $Precision$ and $Recall$ values are obtained, the BEP drawback is not critical, thus it was used in [Yang, 1999; Joachims, 1998; Lewis and Gale, 1994; Apté *et al.*, 1994; Lam and Ho, 1998].

Finally, classification problems may usually involve more than two categories, thus we need a global measure to evaluate the performance of a category pool. In TC, the *microaverage* over all categories is traditionally adopted. According to the definitions 3.10 and 3.11, the Eq. 3.14 and 3.13 define the *microaverage* of $Recall$ and the *microaverage* of $Precision$ for a pool of n binary classifiers.

$$\mu Precision = \frac{\sum_{i=1}^{n} a_i}{\sum_{i=1}^{n} a_i + b_i} \tag{3.13}$$

$$\mu Recall = \frac{\sum_{i=1}^{n} a_i}{\sum_{i=1}^{n} a_i + c_i} \tag{3.14}$$

The above measures are then used to evaluate the *microaverage* of both BEP and f_1 , i.e.:

$$\mu BEP = \frac{\mu Precision + \mu Recall}{2} \tag{3.15}$$

$$\mu f_1 = \frac{2 \times \mu Precision \times \mu Recall}{\mu Precision + \mu Recall} \tag{3.16}$$

3.5 The Parameterized Rocchio Classifier

Machine learning techniques applied to text categorization problems have produced very accurate but, at the same time, computationally complex models. In contrast, systems in real scenario such as Web applications and large-scale information management necessitate fast classification tools. Accordingly, several studies (e.g., [Chuang *et al.*, 2000; Drucker *et al.*, 1999; Gövert *et al.*, 1999]) on improving accuracy of low complexity classifiers have been carried out. They are related to the design of efficient TC models in Web scenarios: feature space reduction, probabilistic interpretation of k-Nearest Neighbor and hierarchical classifiers are different approaches for optimizing speed and accuracy.

In this perspective, there is a renewed interest in the Rocchio formula. Models based on it are characterized by a low time complexity for both training and classification phases. The Rocchio weakness is its accuracy, often much lower than other more computationally complex text classifiers [Yang, 1999; Joachims, 1998].

However, very often, the low performance of a text classifier depends on the non-optimal use of it. Indeed, many classifiers require the tuning of their parameters to achieve high accuracy. Rocchio classifier is one of such model (i.e. it is parametric approach).

Parameterization is an important and critical phase of the application of a learning model since for their nature parameters cannot be derived based on general analytical methods. The usual approach [Kohavi and John, 1997] is, thus, the selection of a set of *interesting* parameter values and the measurement of their impact (in terms of accuracy) on a validation set (wrapper approach). When the number of parameters is large, the exhaustive search becomes prohibitive since we should measure the classifier accuracy for all parameter value combinations which are an exponential number. Thus, a search heuristic is needed.

In case of the Rocchio classifier, defined by Eq. 3.6

$$W_f^i = \max\left\{0, \frac{\beta}{|C_i|}\sum_{d\in C_i} w_f^d - \frac{\gamma}{|\bar{C}_i|}\sum_{d\in \bar{C}_i} w_f^d\right\},$$

the parameters are β and γ. They control the relative impact of positive and negative examples and determine the weights of the features f in the target profile. The setting used for many IR applications was $\beta = 16$ and $\gamma = 4$ (also used for the categorization task of low quality images [Ittner *et al.*, 1995]). Other researchers [Singhal *et al.*, 1997] have found that $\gamma = \beta$ is a good parameter choice but no systematic methodology for parameter setting was proposed.

The major problem of the lack of a systematic parameterization procedure is that γ, β and the threshold constitute a too much large space to be naively explored. To efficiently derive an optimal parameterization a method has been proposed in [Moschitti, 2003c]. The idea is that Rocchio parameter tuning has a feature selection side effect, consequently, to find optimal parameters, we may search for optimal feature subsets. More in detail:

1. The setting of Rocchio parameters can be reduced to the setting of the rate between its two parameters.
2. Different values for the rate imply the removal of some features, i.e. the selection of feature subsets.
3. Only the features in the induced subset affect the accuracy of Rocchio classifier. In particular, the subset can be ranked based on the Rocchio formula values.
4. The parameter rate is inversely-proportional to the cardinality of the feature subset. Thus increasing the rate is equivalent to select the top ranked features.

From the above perspective, the problem of finding optimal parameter ratio can be reduced to the feature selection problem for TC and solved as proposed in [Yang and Pedersen, 1997]. Next section describes in detail such method.

3.5.1 Search Space of Rocchio Parameters

As claimed in the previous section, to improve the accuracy of the Rocchio text classifier, parameter tuning is needed. The exhaustive search of optimal

values for β and γ is not a feasible approach as it requires the evaluation of Rocchio accuracy for all the pairs in the $\mathbb{R}^2$ space.

To reduce the search space, we notice that both γ and β parameters are not needed as β can be bound by the threshold parameter. To show this, it is enough to note that if a classifier accepts a document d in a category C, the scalar product $\vec{C}\cdot\vec{d}$ is greater than a threshold σ, i.e. $\vec{C}\cdot\vec{d} \geq \sigma$. By substituting $\vec{C}$ with the original Rocchio's formula we obtain

$$\left(\frac{\beta}{|C|}\sum_{\vec{d}\in C}\vec{d} - \frac{\gamma}{|\bar{C}|}\sum_{\vec{d}\in \bar{C}}\vec{d}\right)\cdot\vec{d} \geq \sigma$$

and dividing by β,

$$\left(\frac{1}{|C|}\sum_{\vec{d}\in C}\vec{d} - \frac{\gamma}{\beta|\bar{C}|}\sum_{\vec{d}\in \bar{C}}\vec{d}\right)\cdot\vec{d} \geq \frac{\sigma}{\beta} \Rightarrow \left(\frac{1}{|C|}\sum_{\vec{d}\in C}\vec{d} - \frac{\rho}{|\bar{C}|}\sum_{\vec{d}\in \bar{C}}\vec{d}\right)\cdot\vec{d} \geq \tilde{\sigma}.$$

Once ρ has been set, the threshold $\tilde{\sigma}$ which optimizes the accuracy can be derived on the validation set.

The resulting new Rocchio formula is:

$$W_f = \max\left\{0, \frac{1}{|C|}\sum_{d\in C} w_f^d - \frac{\rho}{|\bar{C}|}\sum_{d\in \bar{C}} w_f^d\right\} \tag{3.17}$$

where w_f^d is the component f of $\vec{d}$ and ρ represents the *ratio* between the original Rocchio parameters, i.e. $\frac{\gamma}{\beta}$.

We observe that in Equation 3.17, features with a negative difference between positive and negative weights are set to 0. This aspect is crucial since the 0-valued features do not contribute in the similarity estimation (i.e. they give a null contribution to the scalar product). Thus, the Rocchio model does not use them in the classification phase.

Moreover, if ρ is increased *smoothly*, only the features having a *high* weight in the negative documents will be eliminated (they will be set to 0 value). These features are natural candidates to be irrelevant features to characterize the target category. On one hand, in [Kohavi and John, 1997; Yang and Pedersen, 1997], it has been pointed out that classifier accuracy can improve if irrelevant features are removed from the feature set. On the other hand, the accuracy naturally decreases if relevant and some weakly relevant features are

excluded from the learning [Kohavi and John, 1997]. Thus, by increasing ρ, irrelevant features are removed until the accuracy improves to a maximal point, then weak relevant and relevant features start to be eliminated, causing Rocchio accuracy to decrease.

From the above hypothesis, we argue that:

The best setting for ρ can be derived by increasing it until the Rocchio accuracy reaches a maximum point.

In Section 3.6, experiments will show that the Rocchio accuracy has indeed such behavior. In particular, the ρ/accuracy relationship approximates a convex curve with a unique maximum point.

A linguistic explanation of such phenomenon is that a target class C has its own set of specific terms (i.e. features). We define *specific terms* as the set of words that are very frequent in the target category and occur infrequently in the other categories. For example, *byte* occurs more frequently in a *Computer Science* than a political category, consequently, it is a *specific term* for *Computer Science* (with respect to the *Politic* domain).

The Rocchio formula selects *specific terms* in C also by *looking* at their weights in the other categories C_x. If the negative information is emphasized enough, the *non-specific terms* in C (e.g., terms that occur frequently even in C_x) are removed.

Note that the *non-specific terms* are misleading for a correct categorization. The term *byte* in political documents is not useful for characterizing the political domain. Thus, until the *non-specific terms* are removed, the accuracy increases since the noise is greatly reduced. On the other hand, if negative information is too much emphasized, some *specific terms* tend to be eliminated, causing a decrease of accuracy. For example, *memory* can be considered a *specific term* in *Computer Science*, nevertheless, it can appears in political documents, (e.g. historical memory). By emphasizing the negative weights provided by the political documents, the *memory* feature will eventually be removed from the *Computer Science* profile. This suggests that the specificity of terms in C depends on C_x and can be captured by the ρ parameter.

By exploiting the above findings, a procedure for parameter estimation of ρ over the $training\ set$ has been designed and will be presented in the next section.

3.5.2 Procedure for Parameter Estimation

We propose an approach that uses a set of training documents to learn a profile and a second subset, the *validation set*, to find the ρ value that optimizes the Breakeven Point (i.e. an accuracy measure). This technique allows parameter estimation on data which is independent of both the *test set* (TS) and the training data (as discussed in [Kohavi and John, 1997]).

The initial corpus is divided into a first subset of training documents, called *learning set* LS, and a second subset of documents used to evaluate the performance, i.e. TS. Given a target category, the estimation of its optimal ρ parameter can be carried out according to the following *held-out* procedure:

function FIND-BEST-ρ (LS, Init_Value, Max_Limit);
begin
  Select the estimation set ES from LS;
  $LS_0 = LS - ES$; /*learning set*/
  Let $j = 0$ and ρ_j= Init_Value;
  Let $BEP_0 = 0$;
  do
    $j = j + 1$;
    Train Rocchio on LS_0 with ρ_j in Eq. 3.17;
    Evaluate BEP_j on ES;
    Set $\rho_j = \rho_{j-1} + \Delta\rho$;
  while ($BEP_{j-1} \leq BEP_j$ **and** $\rho_j <$ Max_Limit);
  return ρ_k where $k = argmax_j(BEP_j)$;
end

Table 3.4: Rocchio parameter estimation.

We note that the above algorithm has the following properties.

First, the minimal value for ρ (i.e. the Init_Value) is 0, as a negative rate makes no sense in the feature selection interpretation. The maximal ρ value corresponds to the values that causes all the features to be removed. However, it is not necessary and not recommended to have a very small feature space; one hundred features is a reasonable limit to the smallest size of the space.

Second, the experiments in [Moschitti, 2003c] show that $\rho = 30$ corresponds to a subset of 100 features out of 33,791 initial ones for the *Acquisition* category of the Reuters Corpus. The above feature reduction is rather aggres-

sive as pointed out in [Yang and Pedersen, 1997], thus 30 can be adopted as the maximal limit for ρ. In case we need to carry out an exhaustive search of ρ, its effective maximum limit (corresponding to 0 dimensions) can be evaluated by selecting the maximum ratio between the negative and the positive contributions in Eq. 3.17 for each feature f. For example, in [Moschitti, 2003c], a value of 184.90 was derived from the *Acquisition* category.

Third, the values for $\Delta\rho$ (i.e. the increment for ρ) can be chosen referring to the feature selection paradigm. The subsets adopted in the literature feature selection experiments have a decreasing cardinality [Yang and Pedersen, 1997; Yang, 1999; Joachims, 1998]. They start from the total number of unique features n and then select $n - i \times k$ features in the i-th subset; k varies between 500 and 5,000. When $\Delta\rho = 1$ is used in the estimation algorithm, subsets of similar sizes are generated. Moreover, some preliminary experiments may suggest if smaller values for $\Delta\rho$ produce higher Rocchio accuracy.

Next, the exit **while** condition ($BEP_{j-1} \leq BEP_j$) can be omitted. This increases the parameter search time, but, as we will see in Section 3.6.1, when the number of training examples is low, the ρ/BEP curve is not convex. This means that the algorithm may exit the loop on some local minimum: consequently, the exhaustive search may even be more reliable.

Finally, a more reliable estimation of ρ can be applied if the BEP is evaluated on several estimation sets, i.e. several random splits ES_k and $LS - ES_k$ are generated. Several values $\rho(ES_k)$ can thus be derived at step k. A resulting $\bar{\rho}$ can be obtained by averaging the $\rho(ES_k)$. The Eq. 3.17 parameterized with estimated ρ values will be called the *Parameterized Rocchio Classifier* (PRC).

3.6 Performance Evaluations: PRC, Rocchio and SVMs

This section shows how the experiments in TC should be carried out. As a test case, we verify the hypothesis and claims related to the PRC model. In particular, we would like to:

1. study the relationship between the ρ setting and the accuracy of Rocchio classifier (Section 3.6.1);

2. extract the statistical distribution of ρ parameter from the sample data to observe its variability across categories (Section 3.6.2); and

3. compare PRC with Rocchio accuracy on a well known TC corpus, i.e. *Reuters-21578*, (Section 3.6.2). As PRC is supposed to improve Rocchio, we need to verify and quantify such improvement. A well known corpus is needed since we can compare our results with those from the literature, e.g. [Joachims, 1998; Yang, 1999; Tzeras and Artman, 1993; Cohen and Singer, 1999]. Such results can give us precious indications at design and analysis time. For example, by comparing the accuracy of our system with the one of literature systems (based on the same model), we can discover problems in our algorithm, e.g. program errors.

Additionally, as the accuracy measure of a new model is interesting for many fields, ranging from research centers to industrial sectors, to have a high reliability of the results, the experiments should be carried out over different samples as well as different corpora in different languages. On this line, Section 3.6.3 reports the evaluation of Rocchio, PRC and SVMs on Ohsumed and Reuters corpora. The experiments with SVMs are important since they enable a direct comparison between PRC and one *state-of-the-art* TC model.

Finally, to give to other researchers the possibility to replicate our own results, the experimental setting should be completely described. Such description usually includes:

- The mathematical model description, e.g. the PRC algorithm and its equations. This should include a description of choices adopted in the classification design phases. For example, in our experiments, we will not apply feature selection, the feature weight in a document is evaluated with Eq. 3.3 (i.e. the SMART *ltc* weighting scheme [Salton and Buckley, 1988]) and the Scut policy is adopted.

- The target corpora: in our case we use Reuters-21578, Ohsumed and the ANSA collections; the number of documents in the training, validation and test set should also be reported.

- The adopted performance measures, e.g. the interpolated BEP breakeven point and f_1 (see Section 3.4.2) for the individual category and *microaverage* for global performance, i.e. μBEP and μf_1 (equations 3.15 and 3.16).

- The sets of features used in the experiments, e.g. $Tokens$ or multiwords and their number, e.g. 33,791 for Reuters, 42,234 for Ohsumed

and 55,123 for ANSA. It is often very interesting to describe separated numbers for training, validation and testing.

3.6.1 Relationship between Accuracy and ρ Values

In these experiments, we adopted the fixed split of the Reuters corpus as our *test set* (RTS). The aim is to study as ρ influences the Rocchio accuracy. This latter has been measured by systematically setting different values of $\rho \in \{0, 1, 2, ..., 15\}$ in Eq. 3.17 and evaluating the corresponding BEP.

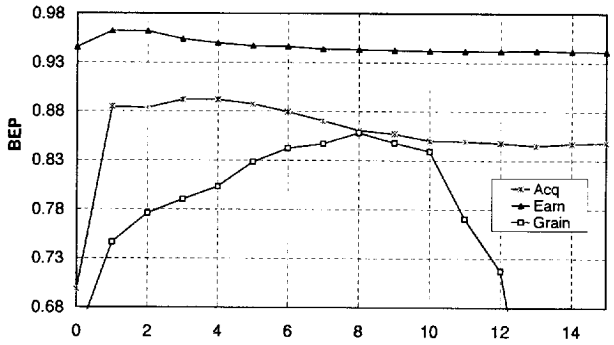

Figure 3.1: BEP of the Rocchio classifier according to different ρ values for *Acq*, *Earn* and *Grain* classes of the Reuters Corpus.

Figures 3.1, 3.2 and 3.3 show the BEP curve of some classes of the Reuters Corpus with respect to ρ value. For *Earn*, *Acq* and *Grain* a large number of training documents is available (i.e. from 2,200 to 500). We observe that, first, the BEP increases according to ρ until a maximum point is reached, then it begins to decrease for higher values of the parameter. Our hypothesis is that after BEP reaches the max point, a further increase of ρ causes the removal of relevant or weakly relevant features to be removed. In this perspective, the optimal ρ setting would correspond to a quasi-optimal feature selection. Second, the different plots suggest that the max BEP is achieved according to different values of ρ. Hence, each category requires a specific independent estimation of the parameter.

The *Trade*, *Interest* and *Money Supply* categories have a smaller number of documents available for training and testing (i.e. from 500 to 100). This

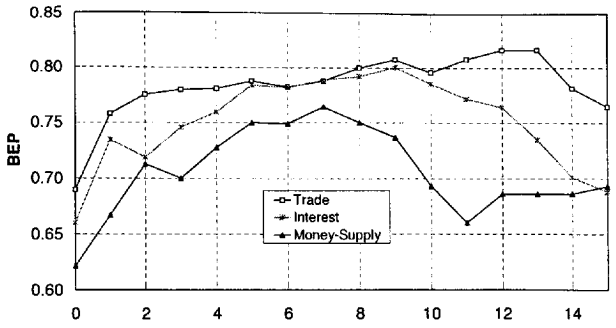

Figure 3.2: BEP of the Rocchio classifier according to different ρ values for *Trade*, *Interest*, and *Money Supply* classes of the Reuters Corpus.

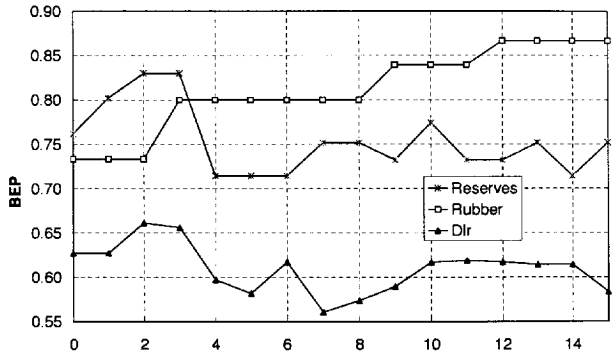

Figure 3.3: BEP of the Rocchio classifier according to different ρ values for *Reserves*, *Rubber* and *Dlr* classes of the Reuters Corpus.

reflects on a less regularity in ρ/BEP relationship. Nevertheless, it is still possible to identify convex curves in their plots. This is important as it allows us to infer that the absolute maximum is into the interval $[0, 15]$. The very small categories (i.e. less than 50 training documents) *Reserves*, *Rubber* and *Dlr* show a more chaotic relationship, and it is difficult to establish an interval that contains the maximum.

It is worth noting that the optimal accuracy is reached for $\rho > 1$, i.e. $\gamma > \beta$. In contrast, it is a common belief that for a classifier the positive information

(embodied by β) should be more relevant than negative information (embodied by γ). This suggests that in Rocchio classifier, the contribute of the feature weights in negative examples should be emphasized

3.6.2 Performance Evaluation on the Reuters fixed *Test Set*

In this experiment the performance of PRC model over the fixed Reuters *test set* (RTS) has been measured. The aim is to provide a direct comparison with other literature results (e.g., [Yang, 1999; Joachims, 1998; Cohen and Singer, 1999; Lam and Ho, 1998]).

Twenty estimation sets $ES_1, ..., ES_{20}$ have been used to estimate the optimal ρ as described in Section 3.5.2. Once the average $\bar{\rho}$ is available for the target category, its profile can be built and its performance can be measured. The μf_1 of PRC on RTS is 82.83%. This outperforms all literature evaluations of the original Rocchio classifier, e.g. 78% obtained in [Cohen and Singer, 1999; Lam and Ho, 1998], 75% in [Yang, 1999] and 79.9% in [Joachims, 1998]. The latter result has been obtained optimizing the parameters on RTS. This means that it represents a biased evaluation. Normally, this is an incorrect procedure but it can be adopted to prove that the proposed model (e.g. SVMs) is superior to the comparing model (e.g. Rocchio) independently of the adopted parameterization procedure, i.e. it is assumed to know the optimal parameters (e.g., γ, β and thresholds) of the comparing model.

The comparison with the literature is interesting but to confirm the improvement of PRC, we should provide our own evaluation of the original Rocchio, i.e. we should implement and parameterized it as proposed in the literature, e.g. $\gamma = 4$ and $\beta = 16$ ($\rho = .25$) and with $\gamma = \beta$ ($\rho = 1$). This is recommended as it is very difficult to exactly replicate the experiments of other researchers, e.g. our tokenizer may be slightly different from the one adopted in literature work, consequently the feature space may be different and produce different accuracy.

The Rocchio results are shown in columns 2 and 3 of Table 3.6. When $\rho = 1$ is used, the Rocchio μf_1 is 78.79% and replicates the results in [Cohen and Singer, 1999; Lam and Ho, 1998] while for $\rho = .25$, the μf_1 is lower, i.e. 72.61%. This is due to the high number of features used in our experiments without applying any feature selection algorithm, i.e. we have more noise data. A low ratio ρ cannot filter an adequate number of irrelevant features and, consequently, the performance is low. As ρ increases, a high number of noisy features is removed and the performance improves. PRC, by determining the

best parameter ρ for each category, improves the Rocchio performance at least by 5 percent points.

Variability of ρ Values across Aamples.

In this section, we study the variability of ρ which supports the explanation for the improved PRC performance. The analysis of the distribution of the ρ values requires the used of *estimation sets*.

ρ values were estimated over 20 samples $ES_1, ..., ES_{20}$. For each category i and for each sample k the best $\rho_i(ES_k)$ value is selected. The results are shown in Table 3.5. The values are reported for 14 categories of the Reuters Corpus that include more than 100 example documents. The name of the categories is shown in Column 1, while their sizes (expressed in number of documents) appear in Column 2. The median, the average and standard deviation of $\gamma_i(ES_k)$ over the 20 samples are reported in columns 3,4 and 5.

When many documents are available for training (i.e. in case of *large categories*), the pointwise estimators (median and mean) seem to represent the optimal ρ values well. They are *near* the last column that represents the optimal ρ evaluated on RTS, e.g. the *Earn* category. In other words, the estimates from the validation set approximate a *general* setting that seems to reflect universal properties of the categories of a given collection.

3.6.3 Cross Evaluation and the n-fold Approach

In order to assess the general performance of both PRC and the original Rocchio classifier, a large empirical evidence is needed on different collections and languages. Also, we evaluated the Support Vector Machine classifier [Joachims, 1998] to estimate the highest TC accuracy achievable on our target corpora.

From the theory of statistical sampling we know that we should operate a sampling of the model performance to achieve reliable estimation. In machine learning such techniques is known as a cross validation. Hereafter, we show the sampling technique that we apply.

1. Generate n random splits of the corpus. For each split j, 70% of data can be used for training (LS^j) and 30% for testing (TS^j).

2. For each split j

Table 3.5: Mean, Standard Deviation and Median of ρ values estimated from samples.

Categories	Size	Me	μ	Std.Dev.	RTS
earn	2544	1	0.8	0.8	1
acq	1520	3	3.8	2.4	3
money-fx	456	10	6.0	5.1	10
grain	374	7	6.9	2.0	8
crude	366	10	7.3	4.7	12
interest	312	9	8.0	2.6	9
trade	312	9	6.0	4.8	12
ship	181	1	3.0	4.5	7
wheat	181	10	7.8	5.1	15
corn	151	10	10.0	1.7	15
dlr	111	0	0.0	0.0	0
Money-supply	110	4	4.3	4.0	7
oilseed	110	10	7.9	4.1	11
sugar	108	10	6.7	4.9	11

(a) Generate m validation sets, ES^j_k of about 10/30% of LS^j.

(b) Learn the classifiers on $LS^j - ES^j{}_k$ and for each $ES^j{}_k$ evaluate: (i) the threshold associated to the BEP and (ii) the optimal parameter ρ.

(c) Learn the classifiers Rocchio, SVMs and PRC on LS^j: in case of PRC use the estimated $\bar{\rho}$.

(d) Evaluate f_1 on TS_j (use the estimated thresholds for Rocchio and PRC) for each category and account data for the final processing of the global μf_1.

3. For each classifier evaluate the mean and the Standard Deviation for f_1 and μf_1 over the TS_j sets.

It is worth noting that:

- In these experiments, the fixed *test set* (RTS) and the *learning set* of the Reuters Corpus are merged to build the new random splits;

- Again, the original Rocchio classifier is evaluated on two different parameter settings selected from the literature (i.e. $\gamma = \beta$ and $\gamma = 4$ and $\beta = 16$); and

- In literature, there is another similar technique used to estimate the TC performance, namely the n-fold cross validation. It consists of (1) dividing the corpus in n non-overlapping samples and (2) taking one sample for testing and the other n for training. Step (2) is repeated n times such that n distinct <test, training set> pairs are obtained. This allows us to carry out n evaluations of the target classification model. The difference with the previous sampling approach is that n-fold uses non-overlapping random splits. In general, this produces a standard deviation greater (i.e. a pessimistic estimation) than the one evaluated on overlapping samples, thus, the resulting confidence limits are more reliable. Moreover, the non-overlapping sampling corresponds to the more realistic view of classifying always new test documents.

Evaluations

Tables 3.6 and 3.7 reports the μf_1 over 90 categories and the f_1 (see Section 3.4.2) for the top 10 mostly populated categories. The original Rocchio accuracy is shown in columns 2, 3, 4 and 5 of the first table. In the second table, columns 2 and 3 refer to PRC while columns 4 and 5 report the SVM accuracy. The RTS label indicates that only the Reuters fixed *test set* has been used to evaluate the results (e.g. only one sample is used). In contrast, the TS^j label means that the measurements are derived averaging the results on 20 (random) splits.

The symbol $\pm$ precedes the Std. Dev. associated with the mean. It indicates the variability of data and can be used to build the confidence limits. We observe that our SVM evaluation on Reuters RTS (85.42%) is in line with the literature (84.2 %) [Joachims, 1998]. The small difference in [Joachims, 1998] is due to the application of a stemming algorithm, a different weighting scheme and a feature selection (only 10,000 features were used there). It is worth noting that the global PRC and SVM outcomes obtained via cross validation are higher than those evaluated on the RTS (83.51% vs. 82.83% for PRC and 87.64% vs. 85.42% for SVMs). This is due to the non-perfectly random nature of the fixed split that prevents a good generalization for both learning algorithms.

Table 3.6: The Rocchio f_1 and μf_1 performance on the Reuters corpus. RTS is the Reuters fixed *test set* while TS^j indicates the evaluation over 20 random samples.

	Rocchio			
Category	RTS		TS^j	
	$\rho = .25$	$\rho = 1$	$\rho = .25$	$\rho = 1$
earn	95.69	95.61	92.57±0.51	93.71±0.42
acq	59.85	82.71	60.02±1.22	77.69±1.15
money-fx	53.74	57.76	67.38±2.84	71.60±2.78
grain	73.64	80.69	70.76±2.05	77.54±1.61
crude	73.58	80.45	75.91±2.54	81.56±1.97
trade	53.00	69.26	61.41±3.21	71.76±2.73
interest	51.02	58.25	59.12±3.44	64.05±3.81
ship	69.86	84.04	65.93±4.69	75.33±4.41
wheat	70.23	74.48	76.13±3.53	78.93±3.00
corn	64.81	66.12	66.04±4.80	68.21±4.82
μf_1 (90 cat.)	72.61	78.79	73.87±0.51	78.92±0.47

We observe that:

First, the cross validation experiments confirm the results obtained for the fixed Reuters split. PRC improves about 5 points (i.e. 83.51% vs. 78.92%) over Rocchio parameterized with $\rho = 1$ with respect to all the 90 categories (μf_1). Note that $\rho = 1$ (i.e. $\gamma = \beta$) is the best literature parameterization. When the standard parameter setting [Cohen and Singer, 1999] is used, i.e. $\rho = .25$, PRC outperforms Rocchio by ~ 10 percent points.

Second, Tables 3.6 and 3.7 show a high improvement also for individual categories, e.g. 91.46% vs. 77.54% for the *grain* category. The last two columns in Table 3.7 report the results for the linear version of $SVMs$[10].

Third, tables 3.8 and 3.9 report the results on the other two corpora, respectively Ohsumed and ANSA. The new data on these tables is the BEP evaluated directly on the TS^j. This means that the estimation of thresholds is not carried out and the resulting outcomes are upperbounds of the real BEP. We have

[10]We have tried to set different polynomial degrees (1,2,3,4 and 5). As the linear version has shown the best performance, we have adopted it for the cross validation experiments.

Table 3.7: f_1 and μf_1 of PRC and $SVMs$ on the Reuters corpus. RTS is the Reuters fixed *test set* while TS^j indicates the evaluation over 20 random samples.

	PRC		$SVMs$	
Category	RTS	TS^j	RTS	TS^j
earn	95.31	94.01±0.33	98.29	97.70±0.31
acq	85.95	83.92±1.01	95.10	94.14±0.57
money-fx	62.31	77.65±2.72	75.96	84.68±2.42
grain	89.12	91.46±1.26	92.47	93.43±1.38
crude	81.54	81.18±2.20	87.09	86.77±1.65
trade	80.33	79.61±2.28	80.18	80.57±1.90
interest	70.22	69.02±3.40	71.82	75.74±2.27
ship	86.77	81.86±2.95	84.15	85.97±2.83
wheat	84.29	89.19±1.98	84.44	87.61±2.39
corn	89.91	88.32±2.39	89.53	85.73±3.79
μf_1 (90 cat.)	82.83	83.51±0.44	85.42	87.64±0.55

Table 3.8: Performance Comparisons among Rocchio, $SVMs$ and PRC on Ohsumed corpus.

	Rocchio (BEP)		PRC		$SVMs$
Category	$\rho = .25$	$\rho = 1$	BEP	f_1	f_1
Pathology	37.57	47.06	48.78	50.58	48.5
Cardiovascular	71.71	75.92	77.61	77.82	80.7
Immunologic	60.38	63.10	73.57	73.92	72.8
Neoplasms	71.34	76.85	79.48	79.71	80.1
Digestive Syst.	59.24	70.23	71.50	71.49	71.1
MicroAv. (23 cat.)	54.4±.5	61.8±.5	66.1±.4	65.8±.4	68.4±.5

used these measurements to compare the f_1 values scored by PRC against the Rocchio upperbounds. This provides a strong indication of the superiority of PRC as both tables show that Rocchio BEP is always 4 to 5 percent points under the f_1 of PRC.

Table 3.9: Performance comparisons between Rocchio and PRC on ANSA corpus

	Rocchio (BEP)		PRC	
Category	$\rho = 0.25$	$\rho = 1$	BEP	f_1
News	50.35	61.06	69.80	68.99
Economics	53.22	61.33	75.95	76.03
Foreign Economics	67.01	65.09	67.08	66.72
Foreign Politics	61.00	67.23	75.80	75.59
Economic Politics	72.54	78.66	80.52	78.95
Politics	60.19	60.07	67.49	66.58
Entertainment	75.91	77.64	78.14	77.63
Sport	67.80	78.98	80.00	80.14
MicroAverage	61.76±.5	67.23±.5	72.36±.4	71.00±.4

Finally, PRC outcome is close to $SVMs$ especially for the Ohsumed corpus (65.8% vs. 68.4%).

As we have seen, the accuracy measure is an important aspect of TC system evaluation. The other critical information is the computational complexity of the learning and classification phases. Indeed, a very accurate classifier that has a huge training and classification time would not be useful in practice. The next section shows the complexity analysis of the PRC model.

3.6.4 PRC Complexity

The evaluation of Rocchio classifier time complexity can be divided into three steps: *pre-processing*, *learning* and *classification*. The *pre-processing* includes the document formatting and the extraction of features. We will neglect this extra time as it is common in almost all text classifiers.

The learning complexity for original Rocchio relates to the evaluation of weights in all documents and profiles. Their evaluation is carried out in three important steps:

1. The IDF is evaluated by counting for each feature the number of documents in which it appears. This requires the sorting of the pair set <*document, feature*> by feature. The number of pairs is bounded by mM, where m is the maximum number of features in a document and M is the number of training documents. Thus, the processing time is $O(m \times M \times log(mM))$.

2. The weight for each feature in each document is evaluated in $O(mM) \times log(n)$ time, where n is the number of unique features.

3. The profile building technique, i.e. the Rocchio formula, is applied. Again, the tuple set <*document, feature, weight*> is sorted by feature in $O(m \times M \times log(m \times M))$ time.

4. All weights that a feature f assumes in positive (negative) examples are summed. This is done by scanning sequentially the <*document, feature, weight*> tuples in $O(mM)$ time.

As result, the overall learning complexity is $O(m \times M \times log(mM)) + O(mM) \times log(n) + O(mM) = O(m \times M \times log(m \times M))$.

The classification complexity of a document d depends on the retrieval of the weights of each feature in d. The total number of unique features, n, is an upperbound of the number of features in a profile, consequently, the classification step takes $O(m \times log(n))$.

In the PRC algorithm, an additional phase is carried out, i.e. the accuracy produced by ρ setting is evaluated on the *validation set* V. This requires (1) the re-evaluation of the profile weights, (2) the classification of V for a target ρ value and (3) repeat steps (1) and (2) for all ρ values.

(1) The re-evaluation of profile weights is carried out by scanning all <*document, feature, weight*> tuples, i.e. $O(mM)$. Note that the tuples need to be sorted only once.

(2) As the classification of one document requires $O(m \times log(n))$, the evaluation of one value for ρ on V takes $O(mM) + O(|V|m \times log(n))$.

(3) The number of ρ values, as described in the previous section, is $k = Max\_Limit/\Delta\rho$. The complexity to measure k values is $k(O(mM) + |V| \times O(m \times log(n)))$. To this quantity, we should add the initial sorting time of the <*document, feature, weight*> tuples, i.e. $O(mM \times log(mM))$. Thus the final complexity is:

$$O(k(mM+|V|\times m\times log(n))\times O(m\times log(mM))+O(mM\times log(mM))$$

The cardinality of the *validation set* $|V|$ as well as k can be considered constants. Indeed, in the PRC interpretation, k is an intrinsic property of the target categories. It depends on feature distribution and is independent of the number of documents or features. Moreover, n is never greater than the product mM. Therefore, the final PRC learning complexity is

$$k(O(mM)+|V|\times O(m\times log(mM)))+O(mM\times log(mM)) = O(mM\times log(mM)),$$

i.e. the complexity of the original Rocchio learning.

The document classification phase of PRC does not introduce additional steps to the original Rocchio algorithm, thus it is characterized by a very efficient time complexity, i.e. $O(m \times log(n))$.

3.7 Conclusions

In this Chapter the basic steps for the design of a general classifier have been described. In particular, the weighting schemes and the similarity modeling sections have shown that a linear learning algorithm corresponds to the application of weighting schemes to the category features.

One of the interesting contributions of this chapter is the feature selection interpretation of the Rocchio classifier since it gives a unified view of weighting schemes and feature selection strategies. The important aspect is that, in such view, the Rocchio's formula can be seen as a traditional feature selector. Moreover, correct procedures that should be followed to validate properties of classification models, e.g. PRC, have been shown. This includes important concepts such as the correct definition of the models, the experimental set-up and the statistical sampling of the performance.

Finally, along with the correct testing of the accuracy of a TC system, it is also important to show its computational costs. The last section has shown how the computational complexity should be evaluated for a new proposed model.

Chapter 4

Advanced Topics in Text Categorization

The previous chapters have motivated the importance of an accurate Text Categorization and have shown that several machine learning approaches have been developed to improve it. With the same aim, other studies attempt to design more effective document representations to allow machine learning models to further increase their accuracy.

Documents are usually described as pairs <*feature, weight*>, consequently, more suitable representations for the learning algorithm can be modeled using either more effective weighting schemes [Singhal *et al.*, 1995; Robertson and Walker, 1994; Buckley and Salton, 1995; Sable and Church, 2001], or by adopting alternative features instead of the simple words. In IR several attempts to design complex and effective features for document retrieval and filtering have been carried out and most of them will be illustrated in the next section.

Additionally, in this chapter, some examples of advanced uses of text categorization systems are shown. Usually, TC systems are employed to organize textual documentation or to design recommending systems. These provide the users with documents relevant to their interests (specified by means of user's profiles). Recommending or management systems are quite intuitive examples of the possible uses of TC. In contrast, Section 4.2 will show some advanced applications of TC that can help the design of other interesting Natural Language processing systems such as Information Extraction, Question/Answering and Text Summarization.

4.1 Advanced Document Representations

Documents embody the full expressions of human linguistic skills. This intuitive observation has led researchers in document retrieval to consider linguistic aspects in the modeling of more complex document representations. Some of the well-known feature models experimented in the last decades are:

- *Lemmas*, i.e. the canonical form of morphologically rich syntactic categories, like nouns or verbs. In this representation, lemmas replace the words in the target texts, e.g., *acquisition* and *acquired* both transform in *acquire*. This should increase the probability to match the target concept, e.g., the act of acquiring against texts that express it in different forms. Lemmatization improves the traditional stemming techniques used in IR. In fact, stems are computed as an approximation of the real root of a word. As a consequence, many words with different meanings have common stems, e.g., *fabricate* and *fabric*, and many stems are not words, e.g., *harness* becomes *har*.

- *Phrases* relate to the sentence subsets in term of subsequences of words. Several phrase types have been defined:

  - *Simple n-grams*, i.e., sequences of words selected by applying statistical techniques. Given a document corpus, all consecutive n-sequences of (non-function) words are generated, i.e. the n-grams. Note that in IR studies, the term n-grams also refers to sequences of n characters (not only words). Statistical selectors based on occurrences and/or document frequencies of word n-grams are applied to select those most representative for the target domain. Typical used selectors are *Mutual Information* or χ^2, described in Chapter 3.
  - *Nouns Phrase*, e.g., Proper Nouns and Complex Nominals. A simple regular expression like N^+ (i.e., every sequence of one or more nouns) based on word categories (e.g., nouns, verbs and adjectives) can be used to select the complex term *Minister of Finance* and discard the illegal term *Minister formally*. The words *Minister* and *Finance*, in the first phrase, are often referred to as *head* and *modifier*, respectively. More modifiers can appear in a complex nominal, e.g., the phrase *Satellite Television System* is composed of the two nouns *Satellite* and *Television* that modify the head *System*.

  - *Linguistic relational structures*, i.e. $<head, modifier_1, .., modifier_n>$ *tuples*. Parsers, e.g., [Charniak, 2000; Collins, 1997; Basili *et al.*, 1998b] are used to detect complex syntactic relations like *subject-verb-object* to select more complex phrases from texts, e.g., $<announces, Minister_{Subj}, plans_{Obj}>$ from *Minister announces plans*. An interesting property is that these tuples can contain non adjacent words, i.e. tuple components can be words that are subject to long distance dependencies. Such tuples can hardly be detected via pure statistical models. In [Strzalkowski and Jones, 1996] only the *subject-verb* and *verb-object* pairs named the *<head, modifier> pairs* have been used.

  The aim of phrases is to improve the Precision on concept matching. For example documents in an *Economic* category could contain the phrase *company acquisition* whereas an *Education* category could include terms like *language acquisition*. If the word *acquisition* is considered alone as feature, it will not be useful to distinguish between the two categories. The whole phrases, instead, give a precise indication of which is the content of the documents.

- *Semantic concepts*, each word is substituted with a representation of its meaning. Assigning the meaning of a content word depends on the definition of word senses in semantic dictionaries. There are two ways of defining the meaning of a word. First, the meaning may be explained by textual definitions like in a dictionary entry or thesaurus. Second, the meaning may be given through other words that share the same sense. WordNet encodes both forms of meaning definitions. Words that share the same sense are said to be *synonyms*. In WordNet, a set of synonym words is called a *synset*. The advantage of using word senses rather than words is a more precise concept matching. For example, the verb *to raise* could refer to: (a) *agricultural texts*, when the sense is *to cultivate by growing* or (b) *economic activities* when the sense is *to raise costs*.

4.1.1 Results of Advanced Representations for Document Retrieval

The above techniques appear feasible for improving IR systems, nevertheless, the use of NLP in IR has produced controversial results and debates. In TREC-5 and TREC-6 [Strzalkowski and Jones, 1996; Strzalkowski and Carballo, 1997], document retrieval based on stems has been slightly improved using

phrases, noun phrases, *head-modifier* pairs and proper names. However, their evaluation was done on *ad-hoc* retrieval mode only, as the less efficient NLP techniques could not be applied to the same *testing-set* of the pure statistical models. This prevented the comparison with the *state-of-the-art* retrieval systems.

In [Strzalkowski *et al.*, 1998; Strzalkowski and Carballo, 1997] a high improvement of retrieval systems was obtained using topic expansion techniques. The initial query was expanded with some related passages not necessarily contained inside the relevant documents. Such NLP techniques were used in TREC-6 to further increase the retrieval accuracy. The success of the above preliminary experiments was not repeated in TREC-8 [Strzalkowski *et al.*, 1999] as the huge amount of data made impossible the correct application of all required steps. The conclusion was that the higher computational cost of NLP prevents its application in operative IR scenario. Another important conclusion was:

NLP representations can increase basic retrieval models (e.g., SMART) that adopt simple stems for their indexing but if advanced statistical retrieval models are used NLP does not produce any improvement. [Strzalkowski et al., *1998].*

In [Smeaton, 1999] a more critical analysis is made. In the past, the relation between NLP and Machine Translation (MT) has always been close. Thus, much of NLP research has been tailored to the MT applications. This may have prevented the NLP techniques to be compatible with task such as retrieval, categorization or filtering. [Smeaton, 1999] claimed that when pure retrieval aspects of IR are considered, such as the statistical measures of word overlapping between queries and documents, recent NLP techniques have little influence on IR. Moreover, NLP is not useful to retrieve documents when they do not contain many, or, any of the query terms.

Current IR is not able to handle cases of different words used to represent the same meaning or concepts within documents or within queries. Polysemous words, which can have more than one meaning, are treated as any other word. Thus, [Smeaton, 1999] suggests to drop the idea of using NLP techniques for IR, instead he suggested to exploit the NLP resources like WordNet.

In this perspective Smeaton used WordNet to define a semantic similarity function between noun pairs. The purpose was to retrieve documents that contain terms similar to those included in the query. Since many words are

polysemous, a Word Sense Disambiguation algorithm was applied. Unfortunately, the accuracy of such algorithm (i.e. between 60/70%) was not sufficient to obtain an improvement of the document retrieval. The semantic similarity led to positive results only after the senses were manually validated.

Other studies using semantic information for improving IR have been carried out in [Sussna, 1993] and [Voorhees, 1993; 1994]. They report the use of word semantic information for text indexing and query expansion respectively. The poor results obtained in [Voorhees, 1994] show that semantic information taken directly from WordNet without performing any kind of WSD is not helping IR at all. In contrast, in [Voorhees, 1998] promising results on the same task were obtained after that the senses of select words were manually disambiguated.

In summary the analysis of the literature reveals that the more likely reasons for the failure of NLP for IR are the following:

- High computational cost of NLP due prevalently to the use of the parser in detecting syntactic relations, e.g., the *<head, modifier>* pairs. This prevented a systematic comparison with *the-state-of-the-art* statistical models on real dataset.
- Small improvements when complex linguistic representation is used. This may be caused either by the NLP errors in detecting such linguistic structures or by the use of NLP derived features as informative as the *bag-of-words*.
- The lack of an accurate WSD tool, in case of semantic representation: (a) The word ambiguity causes the retrieval of a huge number of irrelevant documents if all senses for each query word are introduced, or (b) if a WSD is employed to disambiguate document and query word senses, the retrieval accuracy decrease proportionally to the increase of the WSD error.

4.1.2 Natural Language Processing for Text Categorization

Literature work has shown the failure of NLP for IR. As TC is a subtask of IR, NLP should produce the same results for it. However, there are different aspects of TC that require a separated study as:

- In TC positive and negative documents describing categories are available. This enables the application of theoretical motivated machine

learning techniques that can automatically select relevant features from complex document representations.

- Categories differ from queries as they are fixed, i.e., a predefined set of training documents completely define the target category. This enables the use of feature selection techniques to select relevant features and filtering out those irrelevant also derived from NLP errors.

- There is no query involved in the TC task: (a) documents can be retrieved based on the training documents, which provide a stable routing profile, and (b) the smallest data unit is the document for which it is available a more reliable statistical word distribution for the queries.

- Effective WSD algorithms can be applied to documents whereas this is not the case for queries (especially for short queries). Moreover, an evaluation of WSD tools has been recently carried out in SENSEVAL [Kilgarriff and Rosenzweig, 2000]. The results are an accuracy of 70% for verbs, 75 % for adjectives and 80% for nouns. This last result makes viable the adoption of semantic representation at least for the nouns.

- TC literature studies report contrasting results on the use of NLP for TC. Although, in [Lewis, 1992] is shown that the use of phrases and phrase clusters generates a decrease of classification accuracy on Reuters documents, more recent results, e.g. [Basili *et al.*, 2000; 2001] show that including some syntactic information, such as recognition of proper nouns and other complex nominals in the document representation can slightly improve the accuracy of some weak TC models such as the Rocchio classifier. Other work using phrases [Furnkranz *et al.*, 1998; Mladenić and Grobelnik, 1998; Raskutti *et al.*, 2001; Bekkerman *et al.*, 2001; Tan *et al.*, 2002] report noticeable improvement over the *bag-of-words*. These results require a careful analysis that will be carried out in the next section.

Semantic information for TC was experimented in [Scott and Matwin, 1999]. WordNet senses were used to replace simple words without any word sense disambiguation. The results were mixed as improvements were derived only for small corpus. When a more statistical reliable set of documents was used the adopted representation resulted in performance decrease.

4.1.3 Results in Text Categorization

Literature work has shown the failure of complex representations on ad-hoc document retrieval tasks. However, there are other document retrieval problems such as document filtering and text categorization for which traditional NLP techniques may improve the bag-of-words models. Unfortunately, even for these retrieval tasks, such techniques seem not to be successful. In the following, we report the conclusive discussion of the large experimentation carried out in [Moschitti, 2003b; Moschitti and Basili, 2004].

Syntactic Information in TC

NLP derived phrases seem intuitively superior to the bag-of-words, nevertheless, literature results, e.g. [Moschitti, 2003b; Moschitti and Basili, 2004] have shown that phrases produce small improvement for some Text Categorization algorithms, e.g., Rocchio [Rocchio, 1971], and no improvement for theoretically motivated machine learning algorithms, e.g., Support Vector Machines [Joachims, 1997]. Possible explanations are:

- Word information cannot be easily subsumed by phrase information. As an example, suppose that in the target document representation proper nouns are used in place of their compounding words. Then, suppose that, our task is to design a classifier which assigns documents to a Politic category, i.e. describing political events. The training documents could contain the feature `George Bush` derived by the proper noun *George Bush*. If a political test document contains the `George Bush` feature, it will have chances to be classified in the correct category. On the contrary, if the document contains only the last name of the president, i.e., Bush, the match of the feature `Bush` against the category feature `George Bush` will not be triggered. This is confirmed by the findings in [Caropreso *et al.*, 2001], which show that replacing n-grams with individual tokens produces a decrease of the Rocchio classifier accuracy.

- The information added by the sequence of words is very poor. Note that, a sequence of words may provide a more accurate classification than its compounding words only if two conditions occur:

  (a) The individual words of the sequence appear in the wrong documents. For example the words *George* and *Bush* are included in a

document not related to the political category.

(b) Some documents that contain the whole sequence *George Bush* are correctly categorized in the political category.

Therefore, on the one hand, the sequence *George Bush* is a strong indication of political category; on the other hand, the individual words, *Bush* and *George*, are not related to the political category. This is very unlikely in natural language documents since many co-references between two referentials in which one of them is a word sequence are triggered by a common subsequence (e.g. *Bush* co-refers with *George Bush*). The same situation frequently occurs for the complex nominals, in which the head is used as a shorter coreference. This suggests that rarely terms are not related to their compounding words.

Although, the previous considerations leave little room for an effective use of traditional natural language processing representations, several researches have claimed to have applied them successfully:

- In [Furnkranz, 1998] advanced NLP has been applied to categorize the HTML documents. The main purpose was to recognize student home pages. For this task, the simple word *student* cannot be sufficient to obtain a high accuracy since the same word can appear, frequently, in other University pages. To overcome this problem, the AutoSlog-TS, Information Extraction system [Riloff, 1996] was applied to automatically extract syntactic patterns. For example, from the sentence *I am a student of computer science at Carnegie Mellon University*, the patterns: *I am* <->, <-> *is student*, *student of* <->, and *student at* <-> are generated. AutoSlog-TS was applied to documents collected from various computer science departments and the resulting patterns were used in combination with the simple words. Two different TC models were trained with the above set of features: Rainbow, i.e. a Bayesian classifier [Nigam *et al.*, 2000] and RIPPER. The positive result reported by the authors is a higher Precision when the NLP-representation is used in place of the bag-of-words. This improvement was obtained only for Recall lower than 20%. The explanation was that the above NLP-patterns have lower coverage, thus they can compete with the simple words only in low Recall zone. This result, even if important, cannot be accounted as an evidence of the superiority of the NLP-based TC.

- [Mladenić and Grobelnik, 1998] report the experiments using n-grams. These have been selected by using an incremental algorithm. The web pages in the *Yahoo* categories, *Education* and *References* were used as reference corpus. Both categories contain a sub-hierarchy of many other classes. An individual classifier was designed for each sub-category. The classifiers were trained with the n-grams contained in the few available training documents. The results showed that n-grams produce an improvement about 1 percent point (in terms of Precision and Recall) for *Reference* category and about 4% on the *Educational* category. This latter outcome may represent a good improvement over the bag-of-words, but we have to consider that:

  (a) Although a cross validation was carried out, the experiments were done on 300 documents only.

  (b) The adopted classifier is weak, i.e. a Bayesian model which is not very accurate. Its improvement using n-grams does not prove that the best figure classifier improves too.

  (c) The task is not standard: many sub-categories (e.g., 349 for *Educational*) and few features for each classifier. There are no other researches that have measured the performance on this specific task, thus, it is not possible to compare the results. As the best hypothesis, we can claim that an efficient classifier (averagely accurate) has improved its accuracy, using n-grams. The task involved few data and many categories.

- In [Furnkranz, 1998] is reported the experimentation of n-grams for *Reuters-21578* and *20NewsGroups* corpora. n-grams were, as usual, merged with the words to improve the bag-of-words representation. The selection of features was done using the simple document frequency [Yang and Pedersen, 1997]. RIPPER was trained with both n-grams and simple words. The improvement over the bag-of-words representation, for the Reuters corpus was less than 1%. For *20NewsGroups* no enhancement is reported.

- Other experiments of n-grams using Reuters corpus are reported in [Tan *et al.*, 2002]. Only bigrams were considered. Their selection is slightly different from the previous work, as Information Gain (i.e. Mutual Information) was used in combination with document frequency. The ex-

perimented TC models were Naive Bayes and Maximum Entropy classifiers [Nigam *et al.*, 1999] both trained with bigrams and words. On *Reuters-21578* the authors present an improvement of 2% for both classifiers. The achieved accuracies were 67.07% and 68.90% for Naive Bayes and Maximum Entropy, respectively. Thus, using phrases, we are able to slightly improve TC models which perform about 20% less than the best figure model (SVMs achieve almost 88%). The consequence is that even [Nigam *et al.*, 1999] do not provide clear evidence that simple NLP-derived features as the n-grams, are useful for TC. A higher improvement was reported for the other experimented corpus, i.e. some *Yahoo* sub-categories. Again, we cannot validate these finding as such corpus is not standard. A standard corpus allows researchers to replicate the results. Note also that, it is not possible to compare the performance with [Mladenić and Grobelnik, 1998] as the set of documents and *Yahoo* categories are quite different.

- On the contrary, in [Raskutti *et al.*, 2001], bigrams using SVMs were experimented on the *Reuters-21578*. This enables the comparison with (a) the literature results and (b) the best figure TC model. The feature selection algorithm that was adopted is interesting. They used the n-grams over characters to weight the words and word bigrams inside categories. For example, the sequence of characters "to build" produces the following 5-grams: "to bu", "o bui", "buil" and "build". The occurrences of the n-grams inside and outside categories were used to evaluate the n-gram scores in the target category. In turn n-gram scores are used to weight the characters of a target word. For instance, the character "o" of the word "score" in the "to score b..." receives a contribution from the 5-grams, "o scor", " score", "score", "core", and "ore b". The 5-gram contributions are apportioned giving more weight to the most centered n-gram, i.e. the contributions are multiplied respectively by 0.05, 0.15, 0.60, 0.15 and 0.5. These weights are used to select the most relevant words and bigrams. The above set as well as the whole set of words and bigrams were compared on *Reuters-21578* fixed test set. According to the experiments, SVMs improved about 0.6% when bigrams were added either to all words or to the selected words.

- This may be important because to our knowledge is the first improvement on SVMs using phrases. However, we have to consider that:

  (a) No cross validation was applied. The fact that bigrams improve SVMs on the Reuters fixed test set does not prove that they improve the general SVM accuracy. Indeed, in [Dumais *et al.*, 1998; Moschitti and Basili, 2004], SVMs reach an f-measure over 87%, that is higher than 86.2% obtained by Raskutty et al. with bigrams (even if these latter were experimented with a larger number of categories which determines a more difficult task).

  (b) The improvement on simple words reported in [Raskutti *et al.*, 2001] is 0.6% = 86.2% - 85.6%. If we consider that the Std. Dev. in the experiments in [Moschitti, 2003b; Bekkerman *et al.*, 2001] is 0.4/0.6%, the improvement is not statistically sufficient to assess the superiority of bigrams.

  (c) Only words were used, special character strings and numbers were removed. These strongly affect the results as suggested in [Moschitti and Basili, 2004]. The "only words" model may be improved by bigrams but it provides a baseline lower than the bag-of-words on general strings. Consequently, we cannot state that phrases increase the best figure classification approach. On the contrary, another corpus experimented in [Raskutti *et al.*, 2001], i.e., *ComputerSelect* shows higher accuracy when bigrams are used, i.e. 6 percent points. But again the *ComputerSelect* collection is not standard. This makes difficult to replicate and assess the results.

- The above literature, favorable to the use of phrases in TC, shows that these latter do not affect the accuracy (or at least the best classifier accuracy) on the Reuters corpus. This could be related to the structure and content of its documents, as it has been pointed out in [Raskutti *et al.*, 2001]: Reuters news are written by journalists to disseminate information and hence contain precise words that are useful for classification, e.g., *grain* and *acquisition*, whereas other corpora such as *Yahoo* or *ComputerSelect* categories contain words like software and system, which are useful only in context, e.g., network software and array system.

Semantic Information in TC

Estensive experiments on word senses used as features for TC are described in [Moschitti, 2003b; Moschitti and Basili, 2004]. They show that there is not

much difference between senses and words. This because word senses in category documents tend to be always the same. Moreover, different categories are characterized by different words rather than different senses. The consequence is that words are sufficient surrogates of exact senses.

Another study on the clustering of words which encode similar conceptual information was carried out in [Bekkerman *et al.*, 2001]. They applied the *Information Bottleneck* (IB) feature selection technique to cluster similar features. The important idea was that a classical feature-filtering model cannot achieve good performance in TC as it is usually not related to the adopted machine learning algorithm. The IB method clusters words according to their relationship with categories. More precisely, it attempts to derive a good trade-off between the minimal number of word clusters and the maximum mutual information between the clusters and document categories.

The IB method relates to the distributional clustering approach that has been shown not particularly useful to improve "weak" TC model performance (e.g., Naive Bayes TC). However, a more powerful TC model like SVMs was shown to take advantage of word clustering techniques. SVMs trained with IB derived clusters was experimented on three different corpora: Reuters, *WebKB* and *20NewsGroups*. Only the *20NewsGroups* corpus showed an improvement over the bag-of-words. This was explained by studying the "complexity" of the involved corpora. The above analysis revealed that Reuters and *WebKB* corpora require a small number of features to obtain optimal performance. The conclusion is that if the target corpus is enough complex, the IB can be applied to reduce its complexity (i.e. to reduce the number of relevant features by clustering together those that are similar) and consequently to increase the SVM accuracy. The improvement on *20NewsGroups*, using the cluster representation, was 3 percent points only.

4.2 Some Advanced Applications of Text Categorization

Current Natural Language Processing does not seem appealing to improve the accuracy of TC models, on the contrary TC can be successfully used for NLP applications. The simplest use of TC for Natural Language systems is the enrichment of documents with their category labels. The TREVI[1] system is

[1]TREVI [Basili *et al.*, 1998a] is a distributed object-oriented system, designed and developed by an European consortium under the TREVI ESPRIT project EP23311, for news agencies

an example as its purpose was to provide as much information as possible for the document required by users, e.g., news source, issues date and general categories. Other NLP systems exploit categorization schemes as a navigation method to locate the user needed data. A more complex use of TC relates to the enhancement of IE, Q/A and Summarization systems.

4.2.1 Information Extraction

IE is an emerging NLP technology, whose purpose is to locate specific pieces of information called *facts* (e.g., events or finer grained data), from unstructured natural language texts. This information is used to fill some predefined database table, i.e. *the templates*. Current methods extract such information by using linguistically motivated patterns. Each pattern is a regular expression for which a mapping to a logical form is provided. For example given the following fragment of the Reuters news:

```
WASHINGTON, June 2 - Two affiliated investment firms
told the Securities and Exchange Commission they have
acquired 593,000 shares of Midway Airlines Inc, or 7.7
pct of the total outstanding common stock.  The firms,
Boston-based FMR Corp and Fidelity International Ltd, a
Bermuda-based investment advisory firm, said they bought
the stake "to acquire an equity interest in the company
in pursuit of specified investment objectives...."
```

A typical template that aims to represent information relative to the acquisition of companies may be described by the Table 4.1. Note that to correctly fill the template, a coreference between *Two affiliated investment firms* and *The firms, Boston-based FMR Corp and Fidelity International Ltd* should be detected.

Each different topic, e.g., *bombing events* or *terrorist acts*, requires different customized pattern sets to extract the related *facts*. The construction of pattern base for new topics is a time-consuming and expensive task, thus methods to automatically generate the extraction patterns have been designed.

Categorized documents have been used to enable the unsupervised patterns extraction in AutoSlog-TS [Riloff, 1996]. First, all possible patterns that extract noun phrases are generated from documents, using 15 different heuristics. Second, the documents are processed again to extract all the instances

in two EU languages, English and Spanish.

Buyer	Company	Date	Reported-by	# Shares	Pct
Boston-based FMR Corp and Fidelity International Ltd	Midway Airlines Inc	June 2	Reuters	593,000	7.7

Table 4.1: Example of an Information Extraction template applied to Reuters news from the *Acquisition* category.

that match the patterns, derived during the first step. Finally, the set of patterns are ranked according to the probability that relevant texts contain the target pattern. The relevant texts for a pattern are assumed to be the documents that belong to the target category. This allows the estimation of the relevance probability for a pattern p as the fraction between the number of instances of p in relevant documents and the total number of instances activated by p.

The above method allows the IE designers to save time as the ranked list of patterns can be validated quicker than the manual annotation of the extraction rule from texts. However, the resulting Information Extraction system is clearly domain based and required the manual categorization of the learning documents. An alternative to the manual production of learning data for each application is to use general knowledge valid for any domain. Currently there are two mains linguistic resource based on different knowledge representations: WordNet and FrameNet.

FrameNet is a lexico-semantic database, made recently available[2]. The aim of the FrameNet project is to produce descriptions of words based on semantic frames. Semantic frames, as they have been introduced by [Fillmore, 1982], are schematic representations of situations involving various participants, properties and roles, in which a word may be typically used. The Semantic Frames available from FrameNet are in some way similar to the efforts made to describe the argument structures of lexical items in terms of case-roles or thematic-roles. However, in FrameNet, the role names, which are called Frame Elements (FEs) are local to particular frame structures. For example, the FEs of the ARRIVING frame are *THEME*, *SOURCE*, *GOAL* and *DATE*. They are defined in the following way: the *THEME* represents the object which moves; the *SOURCE* is the starting point of the motion; the *PATH* is a description of the motion trajectory which is neither a *SOURCE* nor a *GOAL*;

[2]FrameNet is available at the Web site: `www.icsi.berkeley.edu/~framenet`.

the *GOAL* is the expression which tells where the theme ends up. A frame has also a description that defines the relations holding between its FEs, which is called the *scene* of the frame. For example, the scene of ARRIVING is: the *THEME* moves in the direction of the *GOAL*, starting at the *SOURCE* along a *PATH*. Additionally, FrameNet contains annotations in the British National Corpus (BNC) of examples of words that evoke each of the frames. Such words are called *target words*, and they may be nouns, verbs or adjectives.

This kind of knowledge can be successfully used for generating domain knowledge required for any new domain. The corpus annotation available from FrameNet enable us to design learning algorithm that (a) categorize sentences in FrameNet frames and (b) allow, once available the target frame, the recognition of extraction rules for any domain [Moschitti *et al.*, 2003].

4.2.2 Question/Answering

IR techniques have been proved to be quite successful at locating, within large collections of documents, those relevant to a user's query. Often, however, the user does not want a whole document but brief answers to specific questions like `How old is the President?` or `Who was the second person on the moon?`. For this new information needs the sole statistical approach of IR is not sufficient. The result is that a new research area that includes IR and NLP techniques has been consolidating, i.e., Question Answering.

Question Answering (Q/A) is a fast growing area of research and commercial interest: from the one hand, it is the only IR subtask that has been proved to be enhanced by NLP; on the other hand, the high capacity of retrieving specific information makes it appealing for business activities, e.g., information management.

The problem of Q/A is to find answers to open-domain questions by searching a large collection of documents. Unlike Internet search engines, Q/A systems provide short, relevant answers to questions. This property is quite appealing as the recent explosion of information available on the World Wide Web has made complex finding information that closely match a specific user need. One of the important feature of Q/A is the fact that both questions and answers are expressed in natural language. In contrast to the IR methods, Q/A approaches deal with language ambiguities and incorporate NLP techniques. All the systems being built in these years exhibit a fairly standard structure: create a query from the user's question, perform IR with the query to locate (segments of) documents likely to contain an answer, and then pinpoint the

most likely answer passage within the candidate documents. Answering questions is thus the problem of finding the best combination of word-level (IR) and syntactic/semantic-level (NLP) techniques. The former produces a set of likely candidate segments and the latter pinpoints the answer(s) as accurately as possible.

An idea to improve Q/A systems is to introduce an additional step that uses TC for filtering incorrect questions and improving the answer ranking. There are two ways to use categorized data in Q/A: (a) to filter paragraphs retrieved by the IR engine and (b) to filter the final answers provided by both IR and NLP processes.

Q/A systems incorporate a paragraph retrieval engine, to find paragraphs that contain candidate answers, e.g. [Clark *et al.*, 1999]. Then, semantic information, e.g. the class of the expected answers, derived from the question processing, is used to retrieve paragraphs and later to extract answers. Typically, the semantic classes of answers are organized in (hierarchical) ontologies and do not relate in any way to topics typically associated with documents, e.g. *Sport*, *Politics* and so on. The ontology of answer classes contains concepts like PERSON, LOCATION or PRODUCT, whereas categories associated with documents are more similar to topics than concepts, e.g., acquisitions, trading or earnings. Given that document categories indicate different semantic information than the class of the expected answer, we may use them to improve the quality of Q/A systems.

An idea to exploit the document category is to implement filtering/re-ranking methods that automatically assigning categories to both questions and texts. The filtering approach allows Q/A systems to eliminate many incorrect answers which do not match the categories of the questions. The ranking approach may use category information to re-rank the answer produced by the basic Q/A systems [Moschitti, 2003a].

4.2.3 Text Summarization

Text Summarization is the process of distilling the most important information from a source to produce an abridged version for a particular user and task [Chinchor *et al.*, 1998; Kan *et al.*, 2001; Hardy *et al.*, 2001]. It is a hard problem of Natural Language Processing as it implies the understanding of the text content. This latter requires semantic analysis, discourse processing, and inferential interpretation (grouping of the content using world knowledge). As current NLP techniques are not enough accurate to accomplish the above

tasks, rather than carrying out true abstraction, approximation are obtained by identifying the most important and central topic(s) of the text, and return them to the reader. Although the summary is not necessarily coherent, the reader can form an opinion of the content of the original text. Indeed, most automated summarization systems today produce extracts only.

Following this last approach, there are two main ways to produce a summary:

- *Information Retrieval-based summaries.* Statistical methods are used to find sentences, which are probably the most representative. Then, the sentences are merged to form an extract (rather than an abstract). The idea is that, in this way, the essence of all text information is retrieved. The meaning of the words or text is not being considered. This has two advantages: (a) the system needs no "world knowledge" and (b) by learning the target domain statistics, e.g., words frequencies, the method can be applied on any text domain or even language. This is a "bottom up" approach: the output is being generated by what is in the text, not by what the user wants to know from it.

- *Information Extraction-based summaries.* In this case, templates that contain the most relevant information, and the patterns for the extraction of the template information are designed for the needed summary type. The system knows what type of words to look for in what context and it extracts that information to fill in the templates. This method is "top down": it finds all and only the information that was asked for. Without a predefined slot the target information is not retrieved. The output text is coherent and balanced unlike the extract generated by IR methods, which may be lacking in balance and cohesion as the sentences are quoted verbatim.

Both techniques can be applied to generate two different types of summaries:

- *Indicative Summaries* that suggest the contents of the document without providing specific detail. They can serve to entice the user into retrieving the full form. *Book jackets*, *card catalog entries* and *movie trailers* are examples of indicative summaries.

- *Informative Summaries* that represent (and often replace) the original document. Therefore, they must contain all the pertinent data necessary to convey the core information and omit ancillary one.

Summaries based on IR models, usually, extract relevant passages for the target queries. An idea to use TC for summarization systems is to introduce the concept of relevance with respect to a category. The indicative and informative summaries can be extracted based on weighting schemes derived from the training data of the target category [Moschitti and Zanzotto, 2002], i.e. we can use profile weights rather than document weights. As a consequence, the summary will be focused on the major subjects of the target category. If we consider that the category profile can be associated with a user profile, this latter can receive summary particularly suited for his/her information needs.

4.3 Conclusions

This chapter has shown that, in the past, many linguistically motivated representations have been modeled but none of them has improved optimal pure statistical approaches based on the simple bag-of-words. The major reasons for such failure are the following: (a) the complex representations provided by the adopted NLP capture just a small piece of information more than the bag-of-words and (b) the errors that occur in their automatic extraction introduce noise in the retrieval process; this decreases the overall performance.

On the contrary the advanced use of TC for natural language applications seems an appealing research area.

Appendix A

Notation

$\boldsymbol{A}$	Matrix A
$\boldsymbol{A}'$	The Transposed Matrix of $\boldsymbol{A}$
b	threshold over s_{di} (in the hyperplane equation)
BEP	Breakeven point
C	a category
C_i	the category i
$\mathcal{C}$	collection of categories
$\|\mathcal{C}\|$	number of categories
$\vec{C}$	vector representation of C, $\vec{C} =< W_{f_1}, .., W_{f_N} >$
$\vec{d}$	vector representation of d, $\vec{d} =< w^d_{f_1}, w^d_{f_2}, ... >$
d	a document
e	Euler's number
f	a feature
f_1	f_1 measure
f_i	the i_th feature of the *corpus*
ϕ	the classification binary function $\phi : D \rightarrow 2^C$
IDF	Inverse Document Frequency
IWF	Inverse Word Frequency
ln	Natural Logarithm
log	Base two Logarithm

μBEP	Microaverage Breakeven point
$\mu Precision$	Microaverage Precision
$\mu Recall$	Microaverage Recall
μf_1	Microaverage f_1 measure
P_i	set of positive documents for C_i
$Precision$	Precision
$\bar{P}_i$	set of negative documents for C_i
$Recall$	Recall
σ	threshold over s_{di} (similarity)
s_{di}	scalar product between document d and category i
$\vec{u} \cdot \vec{v}$	scalar (dot or inner) product between vectors
$\vec{x}$	feature vector
$\vec{x}'$	feature vector transposed
W_f^i	the weight of f in C_i
w_f^d	the weight of f in d

Appendix B

Basic Geometry and Algebraic Concepts

The statistical learning theory is mainly based on geometric and algebraic concepts thus, to improve the understandability of such theory, we recall some important definitions and theorems.

B.1 Vector Spaces

Def. B.1 *Vector Space*
A set V is a vector space over a field F (for example, the field of real or of complex numbers) iff given:

- *a binary operation over vectors of V such as the addition, denoted $\vec{v}+\vec{w}$ where $\vec{v}$ and $\vec{w} \in V$; and*
- *a scalar operation such as the multiplication $a \times \vec{v}$, where $\vec{v} \in V$ and $a \in F$,*

the following properties hold for all $a, b \in F$ and $\vec{u}, \vec{v}$ and $\vec{w} \in V$:

- $\vec{v} + \vec{w} \in V$ *(closure under vector addition).*
- $\vec{u} + (\vec{v} + \vec{w}) = (\vec{u} + \vec{v}) + \vec{w}$ *(associativity of vector addition in V).*
- $\exists \vec{0} \in V : \forall \vec{v} \in V, \vec{v} + \vec{0} = \vec{v}$ *(existence of a neutral element with respect to the addition in V).*

- *For all v in V, there exists an element w in V, such that v + w = 0 (existence of additive inverses in V.)*
- $\vec{v} + \vec{w} = \vec{w} + \vec{v}$ *(commutativity of vector addition in V).*
- $a \times \vec{v} \in V$ *(closure of V under scalar multiplication.)*
- $a \times (b \times \vec{v}) = (ab) \times \vec{v}$ *(associativity of scalar multiplication in V.)*
- *If 1 denotes the multiplicative identity of the field F, then* $1 \times \vec{v} = \vec{v}$ *(neutrality of one).*
- $a \times (\vec{v} + \vec{w}) = a \times \vec{v} + a \times \vec{w}$ *(distributivity with respect to vector addition).*
- $(a+b) \times \vec{v} = a \times \vec{v} + b \times \vec{v}$ *(distributivity with respect to field addition).*

Example B.2 *Vector Space over* $\mathbb{R}$
For all n $\mathbb{R}^n$ *forms a vector space over* $\mathbb{R}$*, with component-wise operations. Let V be the set of all n-tuples,* $(v_1, v_2, ..., v_n)$ *where* $v_i \in \mathbb{R}$*, for* $i = \{1, 2, 3, ...n\}$ *and let the field be* $\mathbb{R}$*, as well. We can define the following two vector space operations:*

- *Vector Addition:*
  For all $\vec{v}, \vec{w}, \in V \Rightarrow \vec{v} + \vec{w} = (v_1 + w_1, v_2 + w_2, ..., v_n + w_n)$.
- *Scalar Multiplication:*
  For all $a \in F$ *and* $\vec{v} \in V \Rightarrow a \times \vec{v} = (av_1, av_2, ..., av_n)$.

Exercize B.3 *Prove that the two operations of Example B.2 define a vector space.*

Def. B.4 *Linear Independence:*
Given a set of linear independent vectors $\vec{v}_1, .., \vec{v}_n \in V$ *and a set of scalar values* $\alpha_1, .., \alpha_n \in \mathbb{R}$*, the linear combination*

$$\alpha_1 \vec{v}_1 + .. + \alpha_n \vec{v}_n,$$

is equal $\vec{0}$ iff $\alpha_1 = .. = \alpha_n = 0$.
In case $\alpha_i \geq 0$ *and* $\sum_{i=1}^{n} \alpha_i = 1$*, the linear combination is called convex.*

Def. B.5 *Normed Vector Space*
If V is a vector space over a field F, a norm on V is a function from V to R. It associates to each vector $\vec{v} \in V$ *a real number,* $||\vec{v}||$*. The norm must satisfy the following conditions:*
For all $a \in F$ *and all* $\vec{u}$ *and* $\vec{v} \in V$*,*

1. $||\vec{v}|| \geq \vec{0}$ *with equality if and only if* $\vec{v} = \vec{0}$.
2. $||a\vec{v}|| = |a| \times ||\vec{v}||$.
3. $||\vec{u} + \vec{v}|| = ||\vec{u}|| + ||\vec{v}||$.

Def. B.6 *Inner Product*
Let V be a vector space and $\vec{u}, \vec{v}$*, and* $\vec{w}$ *be vectors in V and c be a constant. Then an inner product* $(\ ,\ )$[1] *on V is a function with domain consisting of pairs of vectors and range real numbers satisfying the following properties:*

- $(\vec{u}, \vec{u}) > \vec{0}$ *with equality if and only if* $\vec{u} = \vec{0}$.
- $(\vec{u}, \vec{v}) = (\vec{v}, \vec{u})$
- $(\vec{u} + \vec{v}, \vec{w}) = (\vec{u}, \vec{w}) + (\vec{v}, \vec{w})$
- $(c\vec{u}, \vec{v}) = (\vec{u}, c\vec{v}) = c(\vec{u}, \vec{v})$

From the above axioms we derive the following properties:

- $(\vec{v}, \vec{0}) = \vec{0}$
- *If* $(\vec{v}, \vec{u}) = \vec{0}$*,* $\vec{v}$ *and* $\vec{u}$ *are called orthogonal.*
- *Schawrz Inequality:* $[(\vec{u}, \vec{v})]^2 \leq (\vec{u}, \vec{u})(\vec{v}, \vec{v})$*. The equality holds* iff $\vec{u}$ *and* $\vec{v}$ *are linearly dependent.*

Example B.7 *Let V be the vector space consisting of all continuous functions with the standard + and* $\times$ *operators. We define the inner product as follows:*

$$(f, g) = \int_0^1 f(t)g(t)dt$$

[1] Thought this book we will use the classical notation $\vec{u} \cdot \vec{v} = (\vec{u}, \vec{v})$.

For example:

$$(x, x^2) = \int_0^1 x x^2 dx = \left[{}_0^1 \frac{x^4}{4} = \frac{1}{4}\right.$$

The four properties required in Def. B.6 follow immediately from the analogous property of the definite integral:

$$(f + h, g) = \int_0^1 (f(t) + h(t))g(t)dt = \int_0^1 f(t)g(t) + h(t)g(t)dt =$$

$$= \int_0^1 f(t)g(t)dt + \int_0^1 h(t)g(t)dt = (f, g) + (h, g).$$

Example B.8 *The classical scalar product in $\mathbb{R}^n$ is the component-wise product*

$$(u_1, u_2, .., u_n)(v_1, v_2, .., v_n) = (u_1 v_1, u_2 v_2, .., u_n v_n)$$

We recall that

$$cos(\vec{u}, \vec{v}) = \frac{(\vec{u}, \vec{v})}{||\vec{u}|| \times ||\vec{v}||}$$

B.2 Matrixes

Def. B.9 *Transposed Matrix*
Given a matrix $\boldsymbol{A} \in \mathbb{R}^m \times \mathbb{R}^n$ of m rows and n columns, we indicate with $\boldsymbol{A}' \in \mathbb{R}^n \times \mathbb{R}^m$ its transposed, i.e. $\boldsymbol{A}_{ij} = \boldsymbol{A}'_{ji}$ for $i = 1, .., m$ and $j = 1, .., n$.

Def. B.10 *Diagonal Matrix*
Given a matrix $\boldsymbol{A} \in \mathbb{R}^m \times \mathbb{R}^n$, $\boldsymbol{A}$ is a diagonal matrix iff $\boldsymbol{A}_{ij} = 0$ for $i \neq j$ $i = 1, .., m$ and $j = 1, .., n$.

Def. B.11 *Eigen Values*
Given a matrix $\boldsymbol{A} \in \mathbb{R}^m \times \mathbb{R}^n$, an egeinvalue λ and an egeinvector $\vec{x} \in \mathbb{R}^n - \{\vec{0}\}$ are such that

$$\boldsymbol{A}\vec{x} = \lambda\vec{x}$$

Def. B.12 *Symmetric Matrix*
A square matrix $\boldsymbol{A} \in \mathbb{R}^n \times \mathbb{R}^n$ is symmetric iff $\boldsymbol{A}_{ij} = \boldsymbol{A}_{ji}$ for $i \neq j$ $i = 1, .., m$ and $j = 1, .., n$, i.e. iff $\boldsymbol{A} = \boldsymbol{A}'$.

Def. B.13 *Positive (Semi-) definite Matrix*
A square matrix $\boldsymbol{A} \in \mathbb{R}^n \times \mathbb{R}^n$ is said to be positive (semi-) definite if its eigenvalues are all positive (non-negative).

Proposition B.14 *Let $\boldsymbol{A}$ be a symmetric matrix. Then A is positive (semi-) definite iff for any vector $\vec{x} \neq 0$*

$$\vec{x}'\boldsymbol{A}\vec{x} > \lambda\vec{x} \quad (\geq 0).$$

From the previous proposition it follows that: If we find a decomposition $\boldsymbol{A}$ in $\boldsymbol{M}'\boldsymbol{M}$, then $\boldsymbol{A}$ is semi-definite positive matrix as

$$\vec{x}'\boldsymbol{A}\vec{x} = \vec{x}'\boldsymbol{M}'\boldsymbol{M}\vec{x} = (\boldsymbol{M}\vec{x})'(\boldsymbol{M}\vec{x}) = \boldsymbol{M}\vec{x} \cdot \boldsymbol{M}\vec{x} = ||\boldsymbol{M}\vec{x}||^2 \geq 0.$$

Theorem B.15 *Schur Decomposition, (Real Values)*
Every square real matrix $\boldsymbol{A}$ is orthogonally similar to an upper block triangular matrix $\boldsymbol{D}$: $\boldsymbol{A} = \boldsymbol{Q}'\boldsymbol{D}\boldsymbol{Q}$ where each block of $\boldsymbol{D}$ is either a 1#1 matrix or a 2#2 matrix having complex conjugate eigenvalues. $\boldsymbol{D}$ is diagonal iff $\boldsymbol{A}$ is symmetric.