UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

Facoltà di Ingegneria

Corso di Dottorato di Ricerca in Ingegneria Informatica ed Automatica
XXII Ciclo
Dipartimento di Informatica e Sistemistica

MULTIPLE CLASSIFIER SYSTEMS IN ADVERSARIAL ENVIRONMENTS:
Challenges and Solutions

ING. FRANCESCO GARGIULO

Ph.D. Thesis

TUTOR
Prof. Carlo Sansone

COORDINATOR
Prof. Francesco Garofalo

November 2009

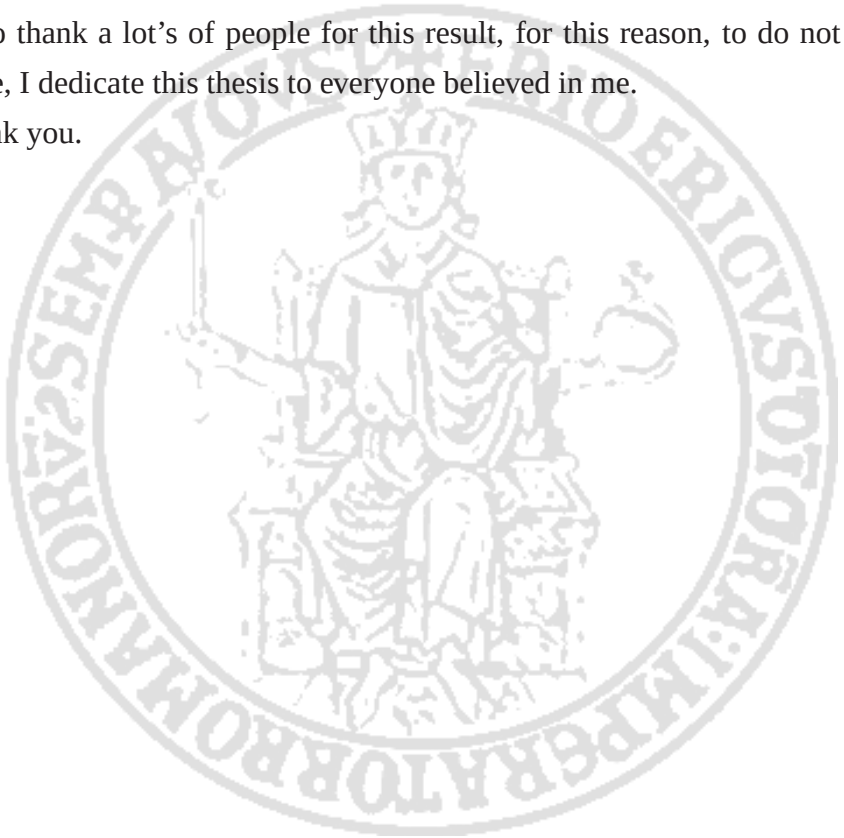
"Nobody is sage enough alone"

Plauto

Acknowledgements

I have to thank a lot’s of people for this result, for this reason, to do not forget someone, I dedicate this thesis to everyone believed in me.

Thank you.



Contents

List of Figures

List of Tables

Chapter 1

Introduction

The automatic recognition of object (*pattern recognition*), and their description, classification and aggregation (*clustering*) are very important fields for a large variety of problems both in the engineering and in the scientific fields.

Watanabe defines a pattern "as opposite of chaos; it is an entity, vaguely defined, that could be given a name." For example a pattern could be a fingerprint image, a handwritten cursive word, a human face, or a speech signal [75][40].

Pattern recognition methods offer technological background for a variety of applications in a modern information society. In some cases, they are however undermined by several kinds of "adversarial" misuses like email and web spam, attacks to computer networks, etc. A classical example of such "adversarial" environments are various evasion techniques used in generation of spam emails. Similar problems arise in web search (web spam) and malware analysis (obfuscation and polymorphism).

The underlying problem is that pattern recognition, as well as data analysis techniques in general, have not been designed to work in adversarial environments.

These considerations give rise with the necessity to define new methodologies to overcome this type of problems, either they are produced during the training phase (*Adversarial Learning*), or they are obtained during the classification phase (*Adversarial Classification*).

To this aim, recently in the area of machine learning the concept of combining

classifiers is proposed as a new direction for the improvement of the performance of individual classifiers. These classifiers could be based on a variety of classification methodologies, and they could achieve different rate of correctly classified samples. The goal of classification result integration algorithms is to generate more accurate system results but a classification more robust to noise.

We found in the Multi Classification System theory also a good support to design and train a classification system in *adversarial environments*. We studied this problem focus on some interesting case studies such as: the cleaning of a noisy/contaminated training set, the spam recognition, the Internet traffic flows detection.

More in general, we tackle the two main challenges directly linked to the general problem of *adversarial environments*, that is: i) **adversarial learning**, in which the labels of training pattern were corrupted; ii) **adversarial classification** in which a malicious user try to camouflage the patterns when the classifier operates on the field.

In particular, in the *adversarial learning* field, we have defined a novel Multiple Classifier System approach designed to clean the noisy/contaminated training set. This system, after an iterative evolution, returns a cleaned training set obtained changing the labels assigned to the samples and considering the training set cleaned when these changes become stable.

The second challenge was the *adversarial classification*. In this case the malicious users try detect the vulnerability of a security system to bypass it. In this context we considered two case studies, in which we proposed some original systems based on a MCS: i) *the spam recognition*, in which the spammer are always looking for some vulnerabilities to brake down the user antispam policies and ii) *the Internet traffic identification*, in which malicious users try to bypass the security policies of an Internet network, using, for example, some allowed protocols to make something different.

1.1 Thesis Outline

After briefly introducing the main context of this thesis, in this section we will give a synthetic outline of the rest of the work. In chapter 2 we will introduce the general problem of the *Adversarial Environment* and we will describe the two challenges directly linked to it, i.e. *Adversarial Learning* and *Adversarial Classification*.

In chapter 3 we will provide some notions about the classification theory, and, after a general introduction of the Multiple Classifiers Systems (MCS), we will give some possible taxonomies.

In chapter 4 we will tackle the problem of the *Adversarial Learning*, in particular for the noisy/contaminated label into the training set. In this chapter we will introduce a methodology to clean a training set, and we will make a comparison between a *simple* classifier trained with the *cleaned* dataset obtained with the proposed approach, and the accuracy obtained with some Multiple Classifier System presented in the literature.

In chapter 5 we will present a typical *Adversarial Classification* problem in the context of the identification of the Internet traffic flows. In this chapter, the traffic problem will be dealt with a statistical approach implemented by a Hierarchical Multiple Classifier System.

In Chapter 6 we will approach another problem of *Adversarial Classification*, i.e. the spam recognition. In this chapter we will describe a modular architecture to adapt the system to new and smarter spammer's attacks.

Finally, in Chapter 7 some conclusions are drawn. Our contribution is pointed out, referring to the previously described work, and some directions and proposals for future works are proposed.

Chapter 2

Adversarial Environments

Machine learning techniques are often used in environments where adversaries can consciously act to limit or prevent accurate performance. A classical example is spam filtering where spammers tailor messages to avoid the most recent spam detection techniques. Further examples of adversarial environments arise in the field of computer security where there is an escalating competition between detection and evasion techniques for various types of malware. In general, one can expect that whenever machine learning is used to provide protection from some illegal activity, adversaries will deliberately attempt to circumvent these approaches.

Pattern recognition systems, and in particular multiple classifier systems, are currently used in several security applications like biometric identity recognition [5][63][55], intrusion detection in computer networks [33][34][49][62] and spam filtering [9][11][22][31][68], in which the task is to discriminate attack samples (e.g., a spam e-mail) from legitimate samples (e.g., legitimate e-mails).

An interesting paper in the context of *machine learning security* is the one by Barreno et al [6] where the authors try to give an answer to the four following questions:

- Can the adversary manipulate a learning system to permit a specific attack? For example, can an attacker leverage knowledge about the machine learning system used by a spam e-mail filtering system to bypass the filtering?
- Can an adversary degrade the performance of a learning system to the extent

		Integrity	*Availability*
Causative	*Targeted*	Permit a specific intrusion	Create a sufficient errors to make system unusable for one person or service
	Indiscriminate	Permit at least one intrusion	Create sufficient errors to make learner unusable
Exploratory	*Targeted*	Find a permitted intrusion from a small set of possibilities	Find a set of points misclassified by the learner
	Indiscriminate	Find a permitted intrusion	

Table 2.1: An Attack Model

that system administrators are forced to disable the IDS? For example, could the attacker confuse the system and cause valid e-mail to be rejected?

- What defences exist against adversaries manipulating (attacking) learning systems?
- More generally, what is the potential impact from a security standpoint of using machine learning on a system? Can an attacker exploit properties of the machine learning technique to disrupt the system?

More in general they made a general taxonomy of the possible attacks to a machine learning system, table 2.1.

For Berreno at al, in the *causative* attacks the adversary has some measure of control over the training of the learner, from the classifier point of view these kinds of attacks are considered as a problem of *Adversarial Learning*.

In the *Exploratory attacks* the adversary do not attempt to influence learning: they instead attempt to discover information about the state of the learner, that is the attackers seek to find intrusions that are not recognized by the learner. From the classifier point of view these attacks can be considered as *Adversarial Classification* problems.

In this thesis we will analyse the two challenges directly linked to the *Adversarial Environment* problem, that is **Adversarial Learning** and **Adversarial Classification** from the Multiple Classifier Systems (MCS) point of view.

It is experimentally demonstrated that the combination of more classifiers can achieve better classification accuracy in respect of a single classifier [15][47].

Only recently, there are few works that are analysing how the MCSs can be robust in an *Adversarial Environment* [10].

2.1 Challenges

The main difference between *Adversarial Learning* and *Adversarial Classification* is in how and where the malicious users try to camouflage the patterns.

While in the first one the attacker contaminates the training pattern to make more difficult the classification problem (from the *learner* point of view), in the second case the attacker changes the patters when the classifier operates on the field to overcome the security system, i.e make more difficult the problem from the *predictor* point of view.

2.1.1 Adversarial Learning

Systems using machine learning have been successfully deployed for fighting spam, fraud, and other malicious activities. These systems typically consist of a classifier that flags certain instances as malicious based on a fixed set of features. For example, spam filters classify each incoming email message as spam or legitimate email by using a set of features such as which words are present. Unfortunately, as classifiers become more widely deployed, the incentive for defeating them increases. In some domains, there is ample evidence that adversaries are actively modifying their behaviour to avoid detection. For instance, senders of junk email often disguise their messages by adding unrelated words, sentences, or even paragraphs more indicative of legitimate email than spam [51].

The earliest theoretical work we know of that approaches learning in the presence of an adversary was done by Kearns and Li [45]. They worked in the context of Valiants Probably Approximately Correct (PAC) learning framework [35, 36], extending it to prove bounds for maliciously chosen errors in the training data. Specifically, they proved that if the learner is to perform correctly, in general the

fraction of training points controlled by the adversary must be less than $\frac{\epsilon}{1+\epsilon}$, where ϵ is the desired bound on classification errors by the learner [4, 6, 30].

Results from game theory may be relevant to adversarial learning systems. In particular, deception games involve players that have partial information and influence the information seen by other players. Some of these games involve continuous variables generated by various probability distributions [7], while others apply to scenarios with discrete states [37]. The game theory and the adversarial learning both ask many of the same questions, and they both address the same underlying issues. Integration of game theoretic concepts could be a promising direction for work in adversarial learning area.

Dalvi et al. examine the learn-adapt-relearn cycle from a game-theoretic point of view [18]. In their model, the learner has a cost for measuring each feature of the data and the adversary has a cost for changing each feature in attack points. If the adversary and learner have complete information about each other and we accept some other assumptions, they find an optimal strategy for the learner to defend against the adversary's adaptations.

Research has also begun to examine the vulnerability of learners to reverse engineering. Lowd and Meek introduce a novel learning problem for adversarial classifier reverse engineering in which an adversary conducts an attack that minimizes a cost function [51]. Under their framework, Lowd and Meek construct algorithms for reverse engineering linear classifiers. Moreover, they build an attack to reverse engineer spam filters [52].

2.1.2 Adversarial Classification

Wittel and Wu [77] discuss the possibility of crafting attacks designed to take advantage of the statistical nature of such spam filters, and they implement a simple attack. John Graham-Cumming [35] describes implementing an attack he calls Bayes vs. Bayes, in which the adversary trains a second statistical spam filter based on feedback from the filter under attack and then uses the second filter to find words that make spam messages undetectable by the original filter.

A recent work of Biggio et al [10] analyses the effectiveness of the Multiple

Classifiers Systems in improving the hardness of evasion. To this aim they develop analytical models of adversarial classification problems and apply them to analyse some strategies currently used to implement MCSs in several applications. They define the hardness of evasion as:

> *For a given feature set, the hardness of evasion is defined as the expected value of the minimum number of features which have to be modified to evade the classifier.*

Very interesting is the figure 2.1 taken from their work and re-proposed here in which is reported an example of the two measures which should be used to evaluate the performance of a classifier in a security system: the classification accuracy against a given strategy used by the adversary (represented by training instances), and the hardness of evasion against a new kinds of attacks.

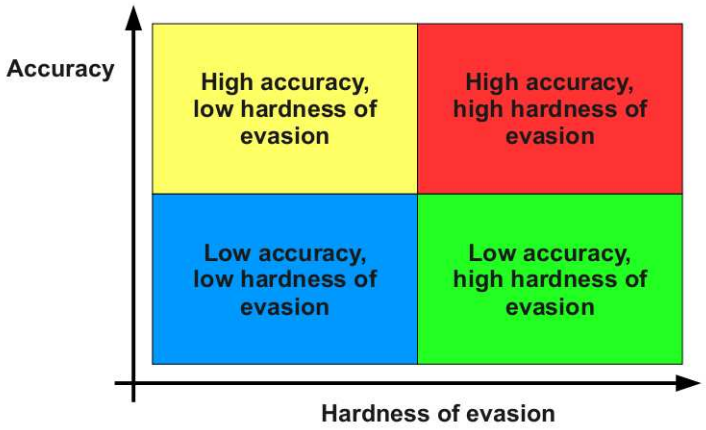


Figure 2.1: Hardness of Evasion vs Accuracy (from [10]

2.2 Applications

We have worked on the two challenges presented before, and in particular we have proposed different methodology to make the system more robust in this kind of environments. In particular we can distinguish three tasks:

- **Adversarial Learning**
  - Noisy/Contaminated Training Set
- **Adversarial Classification**
  - Spam e-mail
  - Internet traffic flows identification

2.2.1 Adversarial Learning: Noisy/Contaminated Training Set

There is not much literature on how noise label should be modelled and dealt with an MCS approach.

AdaBoost [30] has shown to often improve the base learner accuracy. Since its introduction, it has been successfully applied to many problems. Furthermore, the AdaBoost idea has been extended to other sort of problems. Although it has wide-spread success, it is susceptible to the over-fitting problem as pointed out by Dietterich [21]. Oza [61] proposed an approach called AveBoost2 to smooth noise. This approach can be seen as a relaxed version of AdaBoost. When training examples are noisy and therefore difficult to fit, AdaBoost is known to increase the weights of those examples to excess and over-fit them because many consecutive base models may not learn them properly. AveBoost2s averaging does not allow the weights of noisy examples to increase rapidly, thereby mitigating the overfitting problem.

Thiel [73] made a comparison between the single classifier and an ensemble. In his paper the attention is focused on which impact a dataset with *soft* labels has on a noisy training set.

Melville and Mooney [57] introduced a new kind of multiple classifier system to take into account the noise label problem; they called it DECORATE. DECORATE, (Diverse Ensemble Creation by Oppositional Relabelling of Artificial Training Examples) uses an existing ”strong” learner (one that provides high accuracy on the training data) to build an effective diverse committee in a fairly simple, straightforward manner. This is accomplished by adding different

randomly constructed examples to the training set when building new committee members. These artificially constructed examples are given category labels that disagree with the current decision of the committee, thereby easily and directly increasing diversity when a new classifier is trained on the augmented data and added to the committee.

2.2.2 Adversarial Classification: SPAM

It is a well-known story that e-mail has grown from a tool used by few universities and scientists to a ubiquitous communication tool, evolving from simple plain text into a powerful multimedia message. At the same time, following the growth of e-mail production and diffusion, spam has changed from a little and sometimes bothering problem into a multi-billion dollar problem. The presence of spam, in fact, can seriously compromise normal user activities, forcing to navigate through mailboxes to find the - relatively few - interesting e-mails, so wasting time and bandwidth and occupying huge storage space.

The types of those messages vary: some of them contains advertisements, other e-mails provides winning notifications, and sometimes we get messages with executable files, which finally emerge as malicious codes, such as viruses and Trojan horses. In addition, spam e-mails may often have unsuitable content (as a pornographic material advertising) that is illegal and sometimes dangerous for non adult users.

The recognition of spam content is not a trivial problem: there are some factors that are related with human perception, economic behaviour, legal context, that are hardly measurable or summarized in adequate features. The same definition of *spam e-mails* requires a common agreement that is not easy to find.

In our opinion, *all* kind of spam e-mails have several common characteristics, such as: *i*) they are unsolicited, *ii*) they have a commercial content, even though the content itself is continuously evolving, trying to outsmart the classical countermeasures adopted by anti-spam filters.

This kind of task belong to the **adversarial classification** problems, since there is an intelligent, adaptive adversary who tries to camouflage patterns (spam

e-mails) to evade the security system.

Consequently, a great variety of technical methodology have been implemented in current anti-spam systems [11]. The common technical solutions propose filtering strategies based on sender address and/or body content. We focused our attention on that measures related to e-mail contents, in particular both *texts and images*, rather then on networking and identity strategies [68], since our goal is to develop a personal antispam system.

2.2.3 Adversarial Classification: Traffic Identification

In the last years, networking research has started facing a problem not foreseen when the first *Internet* protocols were originally designed: network traffic classification, that is, associating traffic flows to the applications that generated them [56]. Originally each network application used known protocols and transport-level ports that easily allowed their identification. Since a few years back, this is not true any more [42, 59]. The number of network applications using proprietary undisclosed protocols has grown at an incredible rate (Skype, P2P-IPTV); the typical association application/port is often forged; in some cases traffic is encrypted, whereas sometimes it is encapsulated into traditional protocols. Beyond the need to understand which kind of traffic is carried on Internet links, the identification of traffic hidden in flows using well-known ports represents a challenging task. For these reasons, new approaches to traffic identification are needed. By traffic identification here we mean identification of a particular (or a group of) applications of interest.

This is a typical case of study for the *Adversarial Classification* problem. In this case some malicious users try to overcame the classification system in different ways. A possibility is to spoof a protocol into another.

Chapter 3

Classifier Ensembles

Recently in the area of machine learning the concept of combining classifiers has been proposed as a new direction for the improvement of the performance of individual classifiers. These classifiers could be based on a variety of classification methodologies, and they could achieve different rates of correctly classified individuals. The goal of classification result integration algorithms is to generate more certain, precise and accurate system results. Dietterich [21] provides an accessible and informal reasoning, from statistical, computational and representational viewpoints, on why ensembles can improve results.

The combination of multiple classifiers can be considered as a generic pattern recognition problem in which the input consists of the results of the individual classifiers, and the output is the combined decision [72].

Organization of the Chapter

After a general presentation of the pattern recognition problem, we will discuss a general taxonomy of base classifiers; after that we will describe Multiple Classifier Systems, hereinafter *MCS* and we will consider the *pros* and *cons* of some common topologies. We will describe the combination approaches, giving, in the last section, some theoretical details on the Dempster-Shafer combination approach, and on why this approach could be important to overcome some limits of the bayesian one. Finally we will make some practical considerations.

3.1 Pattern Recognition

Watanabe defines a pattern "as opposite of chaos; it is an entity, vaguely defined, that could be given a name." For example a pattern could be a fingerprint image, a handwritten cursive word, a human face, or a speech signal [75][40].

Given a pattern, its recognition/classification may consist of one of the following two tasks:

1. **Supervised Classification**: the input pattern is identified as a member of a predefined class
2. **Unsupervised Classification**: the input pattern is assigned to a hitherto unknown class.

Generally speaking, the design of a pattern recognition system essentially involves the following three aspects:

1. data acquisition and preprocessing
2. data representation
3. decision making

The most popular approaches could be divided into: 1) Template matching, 2) Statistical classification, 3) Syntactic or structural matching, 4) Neural networks.

Template matching is one of the simplest the earliest developed approaches. Matching is a generic operation in pattern recognition which is used to determine the similarity between two entities of the same type. In general, a template or a prototype is always available. Often, the template itself is learned from the training set.

Statistical classification is based on a representation in terms of d features or characteristics. In this case each pattern is seen as a point in a d-dimensional space. Given a set of training patterns from each class, the objective is to establish decision boundaries in the feature space which separate patterns belonging to

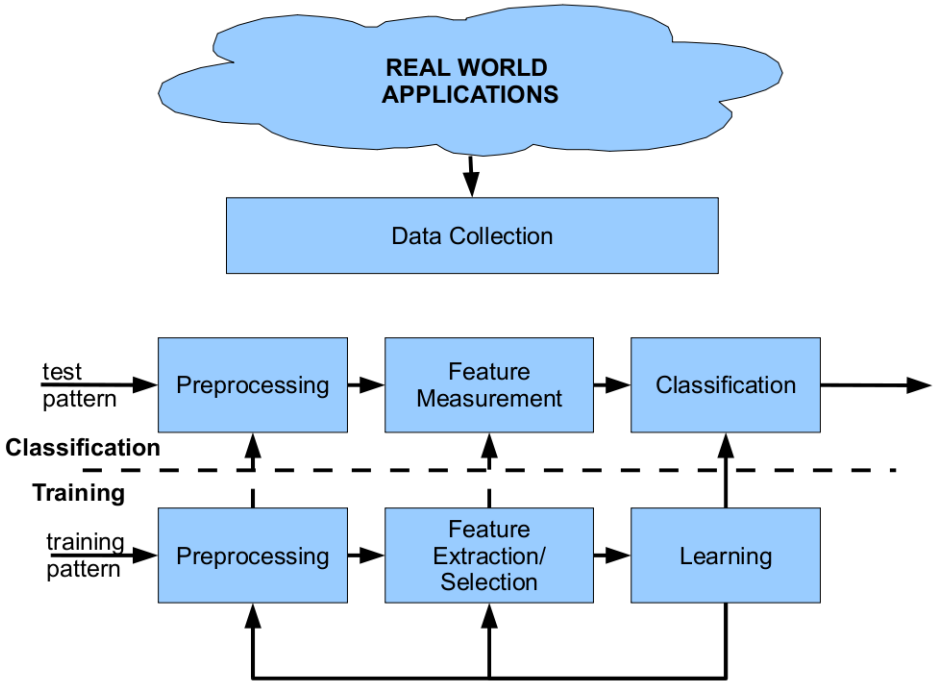


Figure 3.1: General Statistical Pattern Recognition Model

different classes. In this case the recognition system is operated in two modes: training (learning) and classification (testing) as shown in figure 3.1.

In most of the recognition problems involving complex patterns, it is more appropriate to adopt a hierarchical perspective where a pattern is viewed as being composed of simple sub-patterns. In **Syntactical pattern recognition**, a formal analogy is drawn between the structure of patterns and the syntax of the language.

Neural networks can be viewed as massively parallel computing systems consisting of an extremely large number of simple processors with many interconnections. The main characteristics of neural networks are that they have the ability to learn complex non linear input-output relationship, use sequential training procedures, and adapt themselves to the data.

Approach	Representation	Recognition Function	Typical Criterion
Template matching	Samples, pixels, curves	Correlation, distance, measure	Classification error
Statistical	Features	Discriminant function	Classification error
Syntactic or Structural	Primitives	Rules, grammar	Acceptance error
Neural networks	Samples, pixels, features	Network function	Mean square error

Table 3.1: Pattern Recognition Models

3.1.1 Feature selection and extraction

Such a representation requires the definition of the possible categories which have to be recognized, and also the description of the entities to classify in terms of a certain number of parameters. Such parameters are usually referred to as features [51]. Features are usually represented in arrays, and can be distinguished according to the type of value they can assume. They are usually grouped into two sets, as depicted in figure 3.2: quantitative features and qualitative features.

The conceptual boundary between feature extraction and classification is somewhat arbitrary: an ideal feature extractor would yield a representation that makes the job of the classifier trivial; conversely, an omnipotent classifier would not need the help of a sophisticated feature extractor. The distinction is forced for practical, rather than theoretical reasons. Generally speaking, the task of feature extraction is much more problem and domain dependent than classification, and thus requires knowledge of the domain.

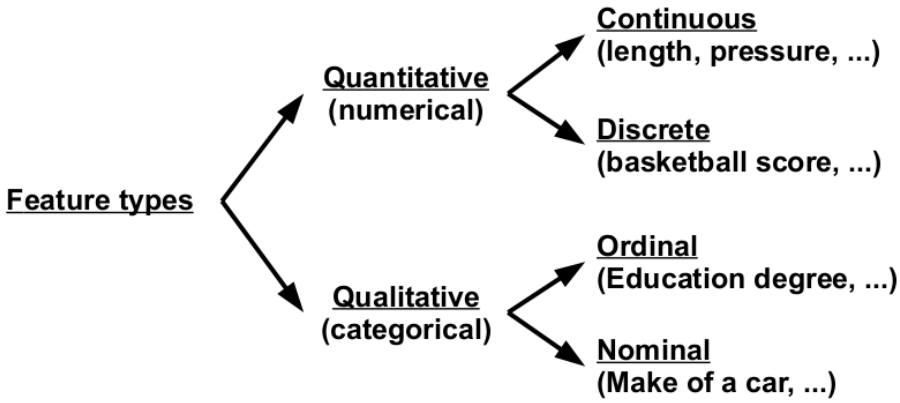


Figure 3.2: Types of Features

3.1.2 Error Evaluation and Classification Accuracy

An important role in classification theory is played by error evaluation. Given a labelled dataset, the most straightforward strategy for evaluating the performance of a classification system is just counting the number of committed errors. Often, the relative amount of errors is given, with respect to the total number of analysed samples. Assume that a labelled data set Z_{ts} of size N_{ts} is available for testing the accuracy of our classifier, D. An estimation of the error is obtained by running D on all the objects in Z_{ts} and find the proportion of misclassified objects

$$error(D) = \frac{N_{error}}{N_{ts}}$$

To find out how the errors are distributed across the classes we construct a confusion matrix using the testing data set, Z_{ts}. The entry e_{ij} of such a matrix denotes the number of elements from Z_{ts} whose true class is C_i, and which are assigned by D to class $\hat{C}_j$. In table 3.2 a general confusion matrix for M classes classification is shown.

In the case of one class classification, the problem of classification is simply reduced to recognize whether a specific sample belongs to the considered class.

True Class	Predicted Class $\hat{C}_1$	$\hat{C}_2$	...	$\hat{C_M}$
C_1	e_{11}	e_{12}	...	e_{1M}
C_2	e_{21}	e_{22}	...	e_{2M}
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
C_n	e_{M1}	e_{n2}	...	e_{MM}

Table 3.2: Confusion matrix for M classes classification

If this is not the case, the sample is simply not assigned to the class of interest. The problem can be formally represented by naming two possible classification outcomes, namely Positive and Negative, representing the only two possible options taken into account. In fact, in such a case, the occurrence of a particular class is searched for. Anything outside such a class is tagged as Negative. The corresponding confusion matrix is represented by table 3.3

True Class	Assigned Class $\hat{P}$	$\hat{N}$
P	TP	FN
N	FP	TN

Table 3.3: Confusion matrix for one class classification

In such a case, the elements of the confusion matrix are named, respectively, True Positives (TP), False Negatives (FN), False Positives (FP) and True Negatives (TN). Such quantities can also be expressed as relative to the total amount of patterns or samples belonging to either the class of interest, or not belonging to it.

3.1.3 Results Evaluation Methodologies

Suppose that we have a data set Z of size N, containing n-dimensional feature vectors describing N objects. We would like to use as much data as possible to build the classifier (training), and also as much unseen data as possible to test its performance more thoroughly (testing). However, if we use all data for training and the same data for testing, we might over-train the classifier so that it perfectly

learns to classify the available data and fails on unseen data. That is why it is important to have a separate data set on which to examine the final product. The main alternatives for making the best use of Z can be summarized as follows:

- *Resubstitution (R-method).* Design classifier D on Z and test it on Z.
- *Hold-out (H-method).* Traditionally, split Z into halves, use one half for training, and the other half for calculating $\hat{P}_D$. $\hat{P}_D$ is pessimistically biased. Splits in other proportions are also used. We can swap the two subsets, get another estimate $\hat{P}_D$ and average the two. A version of this method is the data shuffle where we do L random splits of Z into training and testing parts and average all L estimates of $\hat{P}_D$ calculated on the respective testing parts.
- *Cross-validation (called also the rotation method or p-method).* We choose an integer K (preferably a factor of N) and randomly divide Z into K subsets of size $N = K$. Then we use one subset to test the performance of D trained on the union of the remaining $Kx1$ subsets. This procedure is repeated K times, choosing a different part for testing each time. To get the final value of $\hat{P}_D$ we average the K estimates. When $K = N$, the method is called the *leave-one-out (or U-method).*
- *Bootstrap.* This method is designed to correct the optimistic bias of the R-method. This is done by randomly generating L sets of cardinality N from the original set Z, with replacement. Then we assess and average the error rate of the classifiers built on these sets.

3.1.4 Base Classifiers Taxonomy

Statistical Patter Recognition provides a variety of classifier models. A possible taxonomy is shown in figure 3.3.

One solution is to try to estimate $P(w_i)$ and $p(\mathbf{x}|C_i), i = 1, \ldots, c$, from Z and substitute the estimates $\hat{P}(C_i)$ and $\hat{p}(\mathbf{x}|C_i)$ in the discriminant functions $g_i(\mathbf{x}) = P(C_i)p(\mathbf{x}|C_i), i = 1, \ldots, c$. This is called the *plug-in* approach to classifier design. Approximating $p(\mathbf{x}|C_i)$ as a function of $\mathbf{x}$ divides classifier methods

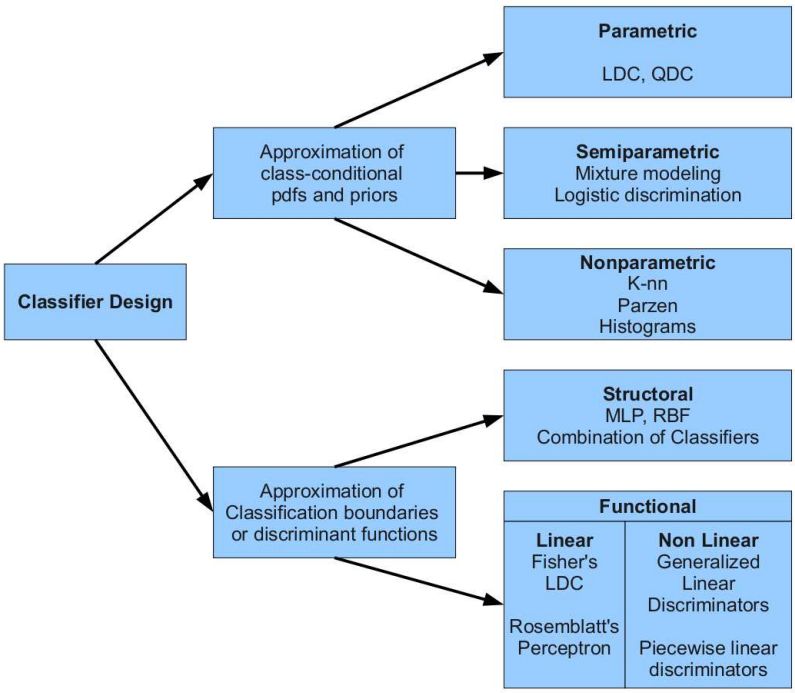


Figure 3.3: A Taxonomy of methods for classifier design

into two big groups: *parametric* and *non parametric*. On the other side of the diagram there are classifier design methods that are not derived by approximating the *pdfs* but rather by devising decision boundaries or discriminant functions empirically. The distinction between the groups is not clear-cut. For example, *radial basis function*(RBF) network from the group of structural approximation of the discriminant functions can be moved to the group of functional approximation, or even to the group of semi-parametric *pdf* modelling. Similarly, the k-nearest neighbour (k-nn) method, although theoretically linked with nonparametric *pdf* estimation, produces a direct estimate of the discriminant functions and can be put under the heading of structural designs for approximating the discriminant functions. There is no consensus on a single taxonomy, or even about the definition of parametric and nonparametric classifiers. Lippmann lists five types of classifiers:

- probabilistic (LDC, QDC, Parzen);
- global (multilayer perceptron (MLP));
- local (radial basis function neural networks (RBF));
- nearest-neighbour type (k-nn, learning vector quantization neural networks (LVQ));
- rule-forming (binary decision trees, rule-based systems).

Holmstrom et al. consider another grouping:

- classifiers based on density estimation:
  - parametric (LDC, QDC);
  - nonparametric (k-nn, kernel methods, finite mixtures, RBF).
- classifiers based on regression:
  - parametric (linear regression, logistic regression, MLP);
  - nonparametric (projection pursuit, additive models).

- other classifiers (e.g., prototype-based: LVQ, k-nn for small k)

Some authors distinguish between neural and nonneural classifiers, local and global classifiers, and so on.

3.2 Categorization of Combination Methods

Combination of multiple classifiers is a fascinating problem that can be considered from many prospectives, and combination techniques can be grouped and analysed in different ways. In terms of implementation, a categorization of combination methods can be made by considering the combination topologies or structures employed, as described in [65]. We could have different MCS depending on [46]:

- Types of classifier outputs
  - *Type 1*: The classifier produces only a label without any information about the classification accuracy.
  - *Type 2*: The classifier gives an ensemble of possible classes ranked in order of importance.
  - *Type 3*: The classifier gives a vector of scores associated to each possible class.
- Trainable or not-Trainable combiners.
- Topology.

Lu [53] categorizes *MCS* topologies into three categories: Cascading , Parallel and Hierarchical.

In a **cascading classifier**, the classification result generated by a classifier is used as an input to the next classifier. The results obtained through each classifier are similarly passed onto the next classifier until a result is obtained through the final classifier in the chain. The main disadvantage of the use of this methodology is the inability of later classifiers to correct mistakes made by earlier classifiers.

Parallel classifiers integrate the results of all classifiers in a singular location. The main design decision that has to be made in the implementation of such a configuration is the selection of a representative combination methodology. If the decision process is well designed, the system can reach peak performance. Some of the more popular and successful combinatorial methods include majority voting, belief integration and the "stacking until convergence" method. However, the improper selection of a combinatorial strategy could accentuate the influence of poorly performing classifiers, which could eventually adversely affect the overall performance.

Hierarchical classifiers combine both parallel and cascading classifier configurations to obtain optimal performance. The use of such a methodology can compensate the disadvantages encountered through the use of a cascading integration. Hierarchical systems could also be used to introduce error checking, which would nullify the influence of poorly performing classifiers.

A more comprehensive and topical categorization of multi-classifier topologies is presented in [48]. This categorization divides topologies into conditional, hierarchical, hybrid and multiple-parallel topologies.

3.2.1 Conditional Topology

This strategy first selects one classifier to perform the task of classification. If this classifier fails to correctly identify the presented data, another classifier is selected, as shown in the figure 3.4. Most implementations include a primary classifier, which is usually selected as the first classifier to be selected. The selection of the next classifier can either be a static decision or maybe based on the values obtained through the use of the primary classifier. Examples methods for dynamic selection include decision trees. This process can continue for as long as there are classifiers available or the pattern is correctly classified. If the primary classifier is an efficient one, the process is computationally efficient. The queue of selected classifiers could be organized in order for the computationally heavy classifiers to be only selected at the end of the classifier queue. One difficult aspect of such an implementation is the selection of a process by which the failures and successes of

a classifier can be evaluated. This method can become overly complicated when the number of available classifiers increases.

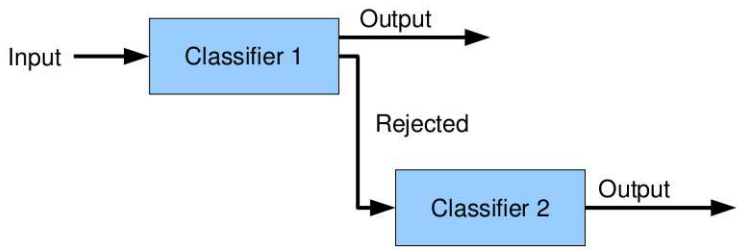


Figure 3.4: Conditional Topology Example

3.2.2 Hierarchical (Serial) Topology

This topology employs a method where classifiers are applied in succession. Each classifier applied to the data is used to reduce the number of possible classes to which such input data belongs to. As the data passes through the classifiers, the decision becomes more and more focused. The common strategy for the design of the classifier queue is to insert classifiers ordered according to decreasing error values. That is to say the classifier with the highest error is used first, whereas the classifier with the lowest error is used last. Of course, there should be safeguards to ensure that the classes selected by each classifier always include the correct class. If not, the next classifier will not have the option of selecting the correct output class.

In the figure 3.5 we show an example of hierarchical topology where each base classifier is a binary one, that can distinguish between the true class and all the rest.

3.2.3 Multiple (Parallel) Topology

This is the most common implementation of a multi-classifier system. All the classifiers first operate in parallel on the input and the results are then pooled to

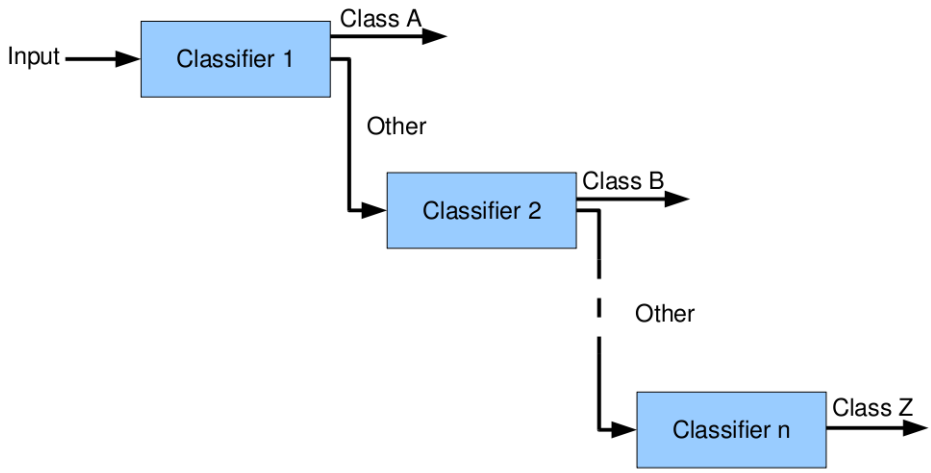


Figure 3.5: Hierarchical Topology Example

obtain a consensus result. This methodology does incur in a cost as it is computationally heavy, with each classifier having to be executed before the final result is obtained.

Parallel combinations can be implemented using different strategies, and the combination method depends on the types of information produced by the base classifiers.

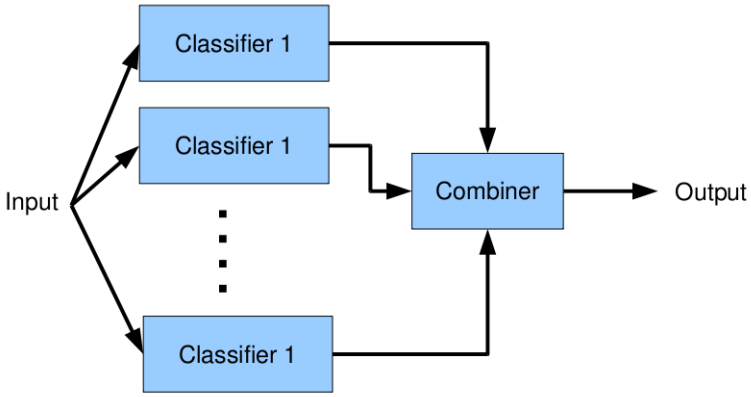


Figure 3.6: Parallel Topology Example

3.2.4 Hybrid Topology

A hybrid topology based system incorporates a mechanism for the selection of the best classifier for a given input. It is obvious that certain classifiers perform better than others on certain data. Thus, the selection of an appropriate classifier would streamline the entire classification process.

This topology could be considered a trade off between parallel and serial topology, a possible example is shown in fig. 3.7. The major disadvantage of this architecture is its complexity, even if we reach better performance with respect to the others topologies.

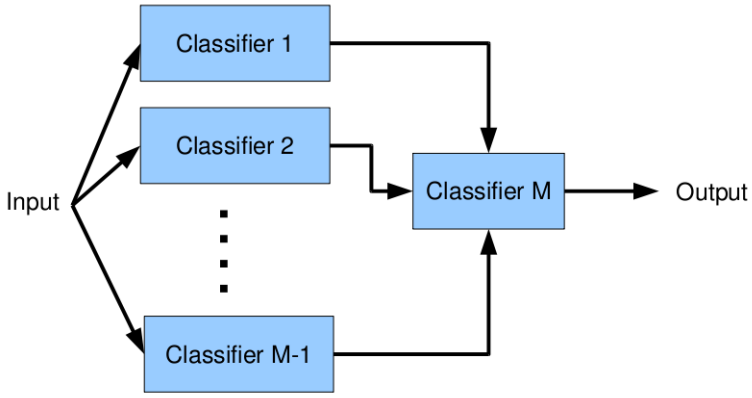


Figure 3.7: Hybrid Topology Example

Hierarchical and multiple topologies are also known as *selection-based* and *fusion-based*, respectively. The classification presented by [48] is more topical and relevant than the one presented by [53] due to the consideration of hybrid systems. Hybrid systems are rapidly gaining popularity among researchers due to the limitations of each system.

3.3 The Combiners

The type of combiners that we can use depends on the base classifiers output. If the base classifiers output is of *Type 1*, we can have different kind of combiners as, for example:

- Majority Voting
- Weighted Majority Voting
- Behaviour Knowledge Space
- Dempster-Shafer

3.3.1 Majority Voting

Dictatorship and majority vote are perhaps the two oldest strategies for decision making. Their roots can be traced back to the era of ancient Greek city states and the Roman Senate. The majority criterion became established in 1356 for the election of German kings, by 1450 was adopted for elections to the British House of Commons, and by 1500 as a rule to be followed in the House itself.

This combiner is based on a *democratic* method, even used in democratic countries: the Vote. Each classifier gives its own evaluation; the final result will be given from the class with more votes. In this case the combiner has to count only the occurrences of each class, and evaluate which class has the greatest number of votes.

If we want to formalize this concept, we can assume that the outputs of the classifier will be denoted with a binary vector of size M, $[d_{i,1}, \ldots, d_{i,c}]^T \in \{0,1\}^M, i = \{1, \ldots B\}$, where B is the number of classifiers involved into the ensemble, M is the number of the possible classes, and where $d_{i,j} = 1$ if the i^{th} classifier votes the class C_j for the actual sample, while $d_{i,j} = 0$ otherwise. So the system will decide for the class C_k if :

$$\sum_{i=1}^{B} d_{i,k} = \max_{j=1}^{M} \sum_{i=1}^{B} d_{i,j} \tag{3.1}$$

That is to say if the number of votes obtained by the C_k class is the maximum of the evaluation obtained from all the possible classes.

3.3.2 Weighted Majority Voting

A variation of the previously described technique is the *weighted majority voting*. In this case, for each classifier, we have also a weight. Obviously this weight will be defined before the classification process. If we want to formalize this method, we can consider the outputs of each classifier as in the previous method. In this case we have to consider an other coefficient vector b_i, that represents the weights associated to the i^{th} classifier. In this case C_k will be given as output class if:

$$\sum_{i=1}^{B} b_i d_{i,k} = \max_{j=1}^{M} \sum_{i=1}^{B} b_i d_{i,j} \tag{3.2}$$

It's worth noting that if the weights b_i are all the same, *weighted majority voting* is exactly the same as majority voting.

A good way to choose the weights could be the following one, as demonstrated in [46].

> *If we consider an ensemble of M independent classifiers, each of them with an his own accuracy, p_i, in which their accuracy will be combined through the weighted majority voting. The accuracy of the combination is maximized put the votes in accord with the following method:*

$$b_i \propto \log \frac{p_i}{1 - p_i} \tag{3.3}$$

3.3.3 Bayesian Combination

The Bayesian Combination rule is based on the *a posteriori* probability. In fact, to an input pattern *x* it will assign the class that maximizes such probability. If we denote it as $s_i(x)$, for the sake of simplicity, hereinafter the x will be ignored, the output of the M classifiers involved into the ensemble, and with w_k the generic class. The combiner has to choose the class that maximize the quantity:

(3.4) $$p(w_k|s_1, s_2, \ldots, s_M)$$

This is the best combination method that we can use to reduce the error probability. The problem regards the knowledge of all the conditional probabilities for the available classes. This information is often unknown. To overcome this problem, it's possible to use some decision rules directly derived from the bayesian formalism, that are an approximation of eq. 3.4

The principal combination rules are:

- Product Rule
- Sum rule
- Max rule
- Min rule
- Median rule

Product Rule

If we use the Bayes Rule it's possible to rewrite eq. 3.4 as:

(3.5) $$p(w_k|s_1, s_2, \ldots, s_M) = \frac{p(w_k)p(s_1, s_2, \ldots, s_M|w_k)}{p(s_1, s_2, \ldots, s_M)}$$

It's possible to rewrite the denominator as:

(3.6) $$p(s_1, s_2, \ldots, s_M) = \sum_{l=1}^{N} p(s_1, s_2, \ldots, s_M|w_l)p(w_k)$$

where N is the number of the possible classes. Now, if we assume that the outputs of all the classifiers are **conditionally independent**, we can rewrite the conditional probability as:

$$p(s_1, s_2, \ldots, s_M|w_k) = \prod_{i=1}^{M} p(s_i|w_k) \tag{3.7}$$

consequently eq. 3.5 becomes:

$$p(w_k|s_1, s_2, \ldots, s_M) = \frac{p(w_k)^{-M+1} \prod_{i=1}^{M} p(w_k|s_i)}{\sum_{l=1}^{N} \prod_{i=1}^{M} p(s_i|w_l)p(w_k)} \tag{3.8}$$

To maximize eq. 3.8, it's necessary to maximize its numerator with respect to k, that is:

$$\max_k \{p(w_k)^{-M+1} \prod_{i=1}^{M} p(w_k|s_i)\} \tag{3.9}$$

Eq. 3.9 represents the product rule. In fact we try to maximize the product of the conditional probability of each classifier, with respect to all the classes. One of the major problems of this technique is that it's linked to the possibility that one or more classifiers give a result close to zero. In this case, the product will give us a value close to zero , and the combiner will fail.

Sum rule

To define the sum rule we have to make the hypothesis that all the *a priori* probabilities and the *a posteriori* probabilities are very close each other:

$$p(w_k|s_i) = p(w_k)(1 + \delta_{i,j}) \qquad with \quad \delta_{i,j} << 1 \tag{3.10}$$

After this we can substitute eq. 3.10 into eq. 3.9, and we can obtain:

$$p(w_k)^{-M+1} \prod_{i=1}^{M} p(w_k|s_i) = p(w_k) = p(w_k) \prod_{i=1}^{M} (1 + \delta_{i,j}) \tag{3.11}$$

After that if we expand the second member product and we don't consider the second order terms, we obtain:

$$\max_k\{(1-M)p(w_k) + \sum_{i=1}^{M} p(w_k|s_i)\} \tag{3.12}$$

Eq.3.12 represents the *sum rule*. The limit are in the initial hypothesis which is very restrictive. That is true only in a very few cases.

Max rule

This rule is obtained directly from the sum rule, in fact it's obtained as an approximation of the sum with the maximum into the eq. 3.12

$$\max_k\{(1-M)p(w_k) + M \max_{i=1}^{M} p(w_k|s_i)\} \tag{3.13}$$

Min rule

This rule is obtained starting from eq. 3.9 with an approximation of the product with the minimum.

$$\max_k\{p(w_k)^{-M+1} \min_{i=1}^{M} p(w_k|s_i)\} \tag{3.14}$$

Median rule

Finally, the median rule is obtained starting from eq. 3.14, using the median instead of the minimum:

$$\max_k\{p(w_k)^{-M+1} med_{i=1}^{M} p(w_k|s_i)\} \tag{3.15}$$

Obviously to use this rule it's necessary that the hypothesis that the *a priori* probabilities are the same is satisfied.

3.3.4 The Dempster-Shafer approach

The theory of Dempster and Shafer (D-S theory) has been frequently applied to deal with uncertainty management and incomplete reasoning.

In many applications, information is collected using several independent sources and it is needed to integrate such pieces of information in order to improve the reliability of the decision making process. The *Dempster-Shafer theory of evidence*, is a framework for such purpose that has found applications in diverse areas such as expert systems, accounting, robotics, medical imaging, documental retrieval, computer vision, pattern matching. and automatic target recognition.

Differently from the classical Bayesian theory, D-S theory can explicitly model the absence of information, while in case of absence of information a Bayesian approach attributes the same probability to all the possible events.
The DempsterShafer theory could narrow down a hypothesis set with the accumulation of evidence and it allows for a representation of the *ignorance* due to the uncertainty in the evidence. When the ignorance reaches the value zero, the DempsterShafer model reduces to the standard Bayesian model. Thus, the DempsterShafer theory could be considered as a generalization of the theory of probability.

Some theoretical issues

Let θ be a finite, non-empty set consisting of all the possible values of a certain attribute. The set θ serves as our universal set, and it is called the *frame of discernment*. A *mass function*, also called *basic probability assignment*, is a mapping m from the set of all subsets of θ into the closed interval $[0, 1]$ such that

$$m(\varnothing) = 0 \qquad \sum_{A \subseteq 2^{\theta}} m(A) = 1 \tag{3.16}$$

The function value $m(A)$ measures the degree of evidence that is assigned to the subset and (1) reflects that the total evidence is one. The simplest mass function corresponds to the case when there is no available evidence at all (i.e.,*total ignorance*), in this case we set $m(\theta) = 1$ and $m(A) = 0$ for all other subsets of θ.

When assigning a *bpa*, there are some requirements which have to be met. They descend from the fact that the *bpa* is still a probability function, hence has to respect the constraints for mass probability functions. Each *bpa* is such that $m : 2^\theta \rightarrow [0, 1]$, where θ indicates the so called *frame of discernment*. Usually, the frame of discernment θ consists of M mutually exclusive and exhaustive hypotheses A_i, $i = 1, \ldots, M$. A subset $\{A_i, \ldots, A_j\} \subseteq \theta$ represents a new hypothesis. As the number of possible subsets of θ is 2^θ, the generic hypothesis is an element of 2^θ.

For example, if we only consider two hypotheses (classes), namely *Positive*(P) and *Negative*(N); hence, the frame of discernment is $\theta = \{\{P\}, \{N\}\}$ and $2^\theta = \{\{P\}, \{N\}, \{P, N\}\}$, whereas in the Bayesian case only the events $\{\{P\}, \{N\}\}$ would be considered.
$\{P\}$ and $\{N\}$ are referred to as *simple events* or *singletons*, while $\{P, N\}$ is referred to as *composite event*. Furthermore, also the following properties have to hold:

$$m(\varnothing) = 0 \qquad \sum_{A \subseteq 2^\theta} m(A) = 1$$

The aim of assigning a *bpa* is to describe the reliability of a particular classifier in reporting a specific event. Such a representation is suitable for combination, but as we want to deal with combined results in the same way, we also impose the constraint that the combination of several *bpa* by means of the D-S rule still has to be a *bpa*. The uncertainty in the final decision will be inversely proportional to the extent to which the base classifiers agree. If we have n base classifiers, the combination rule is such that:

$$m(A) = K \sum_{\bigcap_{i=1}^n A_i = A} \prod_{i=1}^n m_i(A_i)$$

where:

$$\begin{aligned} K^{-1} &= 1 - \sum_{\bigcap_{i=1}^n A_i = \varnothing} \prod_{i=1}^n m_i(A_i) \\ &= \sum_{\bigcap_{i=1}^n A_i \neq \varnothing} \prod_{i=1}^n m_i(A_i) \end{aligned}$$

It is worth observing that the normalizing factor K is independent from any specific value of A. The value K can therefore be considered a constant, once the bpas are fixed.

3.4 Well Known MCS Approaches

There is no definitive taxonomy. Jain, Duin and Mao (2000) list eighteen classifier combination schemes; Witten and Frank (2000) detail four methods of combining multiple models: bagging, boosting, stacking and error-correcting output codes whilst Alpaydin (2004) covers seven methods of combining multiple learners: voting, error-correcting output codes, bagging, boosting, mixtures of experts, stacked generalization and cascading. Here, the literature in general is reviewed, with, where possible, an emphasis on both theoretical and practical advices, then the taxonomy from Jain, Duin and Mao (2000) is provided, and finally four ensemble methods are focussed on: bagging, boosting (including AdaBoost), stacked generalization and the random subspace method.

Table 3.4 provides a taxonomy of ensemble methods which was taken from Jain, Duin and Mao (2000).

3.4.1 Boosting

Boosting was inspired by an on-line learning algorithm called *Hedge(β)*. This algorithm allocates weights to a set of strategies used to predict the outcome of a certain event.The weight of strategy s_i, if properly scaled, can be interpreted as the probability that s_i is the best (most accurate) predicting strategy in the group. The distribution is updated on-line after each outcome. Strategies with the correct prediction receive more weight while the weights of the strategies with incorrect predictions are reduced.

Boosting is related to the general problem of producing a very accurate prediction rule by combining rough and moderately inaccurate rules-of-thumb.The general boosting idea is to develop the classifier team D incrementally, adding one classifier at a time. The classifier that joins the ensemble at step k is trained

Scheme	Architecture	Trainable	Adaptive	Info-level	Comments
Voting	Parallel	No	No	Abstract	Assumes independent classifiers
Sum, mean, median	Parallel	No	No	Confidence	Robust; Assumes independent confidence estimators
Product, min, max	Parallel	No	No	Confidence	Assumes independent features
Generalized ensemble	Parallel	Yes	No	Confidence	Considers error correlation
Adaptive weighting	Parallel	Yes	Yes	Confidence	Explores local expertise
Stacking	Parallel	Yes	No	confidence	Good utilization of training data
Borda count	Parallel	Yes	No	Rank	Converts ranks into confidences
Logistic regression	Parallel	Yes	No	Rank confidence	Converts ranks into confidences
Class set reduction	Parallel cascading	Yes/No	No	Rank confidence	Efficient
Dempster-Shafer	Parallel	Yes	No	Confidence	Fuses non-probabilistic confidences
Fuzzy integrals	Parallel	Yes	No	confidence	Fuses non - probabilistic confidences
Mixture of local experts (MLE)	Gated parallel	Yes	Yes	Confidence	Explores local expertise; joint optimization
Hierarchical MLE	Gated parallel hierarchical	Yes	Yes	Confidence	Same as MLE; hierarchical
Associative switch	Parallel	Yes	Yes	Abstract	Same as MLE, but non joint optimization
Bagging	Parallel	Yes	No	confidence	Needs many comparable classifiers
Boosting	Parallel hierarchical	Yes	No	Abstract	Improves margins; unlikely to overtrain , sensitive to mislabels; needs many comparable classifiers
Random subspace	Parallel	Yes	No	Confidence	Needs many comparable classifiers
Neural trees	Hierarchical	Yes	No	confidence	Handles large numbers of classes

Table 3.4: Ensemble Methods

on a data set selectively sampled from the training data set Z. The sampling distribution starts from uniform, and progresses towards increasing the likelihood of "difficult" data points. Thus the distribution is updated at each step, increasing the likelihood of the objects misclassified at step $k-1$.

The classifiers in D are the trials or events, and the data points in Z are the strategies whose probability distribution we update at each step. The algorithm is called *AdaBoost* which comes from ADAptative BOOSTing. There are two implementation of AdaBoost: with *reweighting* and with *resampling*.

AdaBoost

AdaBoost is one of the best-known and best-performing ensemble classifier learning algorithms. It constructs a sequence of base models, where each model is constructed based on the performance of the previous model on the training set. In particular, AdaBoost calls the base model learning algorithm with a training set weighted by a distribution. After the base model is created, it is tested on the training set to see how well it learned.

The figure 3.4.1 shows AdaBoost's pseudocode. AdaBoost constructs a sequence of base models $h_t \, for \, t \in \{1, 2, \ldots, T\}$, where each model is constructed based on the performance of the previous base model on the training set. In particular, AdaBoost maintains a distribution over the m training examples. The distribution $\mathbf{d}_1$ used in creating the first base model gives equal weight to each example ($d_{1,i} = 1/m \, \forall i \in \{1, 2, \ldots, m\}$). AdaBoost now enters the loop, where the base model learning algorithm L_b is called with the training set and $\mathbf{d_1}$. The returned model h_1 is then tested on the training set to see how well it learned. The total weight of the misclassified examples (ϵ_1) is calculated. The weights of the correctly-classified examples are multiplied by $\epsilon_1/(1\epsilon_1)$ so that they have the same total weight as the misclassified examples. The weights of all the examples are then normalized so that they sum to 1 instead of $2\epsilon_1$. AdaBoost assumes that L_b is a weak learner, i.e., $\epsilon_t < 1/2$ with high probability. Under this assumption, the total weight of the misclassified examples $\epsilon_t < 1/2$ is increased to $1/2$ and the total weight of the correctly classified examples $1\epsilon_t > 1/2$ is decreased to $1/2$. This is

$AdaBoost((x_1, y_1), \ldots, (x_m, y_m), L_b, T)$

Initialize $d_{1,i} = \frac{1}{m} \quad \forall i \in \{1, 2, \ldots m\}$.

for $t = 1, 2, \ldots, T$,

$h_t = L_b(\{(x_1, y_1), \ldots, (x_m, y_m)\}, \mathbf{d_t})$

Calculate the error of $h_t : \epsilon_t = \sum_{i:h_t(x_i) \neq y_i} d_{t,i}$

if ($\epsilon_t \geq 1/2$) then,

set $T = t - 1$ and abort this loop.

$\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$

Calculate distribution d_{t+1}:

$$w_i = d_{t,i} \times \begin{cases} \beta_t, & if h_t(x_i) = y_i \\ 1, & otherwise \end{cases}$$

$$d_{t+1,i} = \frac{w_i}{\sum_{i=1}^{m} w_i}$$

return the final hypothesis:

$h_{fin}(x) = argmax_{y \in Y} \sum_{t:h_t(x)=y} \log \frac{1}{\beta_t}$

Figure 3.8: AdaBoost algorithm

done so that, by the weak learning assumption, the next model h_{t+1} will classify at least some of the previously misclassified examples correctly. Returning to the algorithm, the loop continues, creating the T base models in the ensemble. The final ensemble returns, for a new example, the one class in the set of classes Y that gets the highest weighted vote from the base models. Each base models vote is proportional to its accuracy on the weighted training set used to train it.

3.4.2 Bagging

Bagging is introduced by (Breiman 1996) as an acronym for *Bootstrap AGGregatING*. The idea of bagging is simple and appealing: the ensemble is made of classifiers built on bootstrap replicates of the training set. The classifier outputs are combined by the plurality vote. The meta-algorithm, which is a special case of model averaging, was originally designed for classification and is usually applied

to decision tree models, but it can be used with any type of model for classification or regression. The method uses multiple versions of a training set by using the bootstrap, i.e. sampling with replacement. Each of these data sets is used to train a different model. The outputs of the models are combined by averaging (in the case of regression) or voting (in the case of classification) to create a single output.

Bagging is only effective when using unstable (i.e. a small change in the training set can cause a significant change in the model) non-linear models.

3.4.3 Stacked Generalization

Stacked generalization (or stacking) (Wolpert 1992) is a different way of combining multiple models, that introduces the concept of a meta learner. Although an attractive idea, it is less widely used than bagging and boosting. Unlike bagging and boosting, stacking may be (and normally is) used to combine models of different types. The procedure is as follows:

1. Split the training set into two disjoint sets.
2. Train several base learners on the first part.
3. Test the base learners on the second part.
4. Using the predictions from 3) as the inputs, and the correct responses as the outputs, train a higher level learner.

Note that steps 1) to 3) are the same as cross-validation, but instead of using a winner-takes-all approach, the base learners are combined, possibly non-linearly.

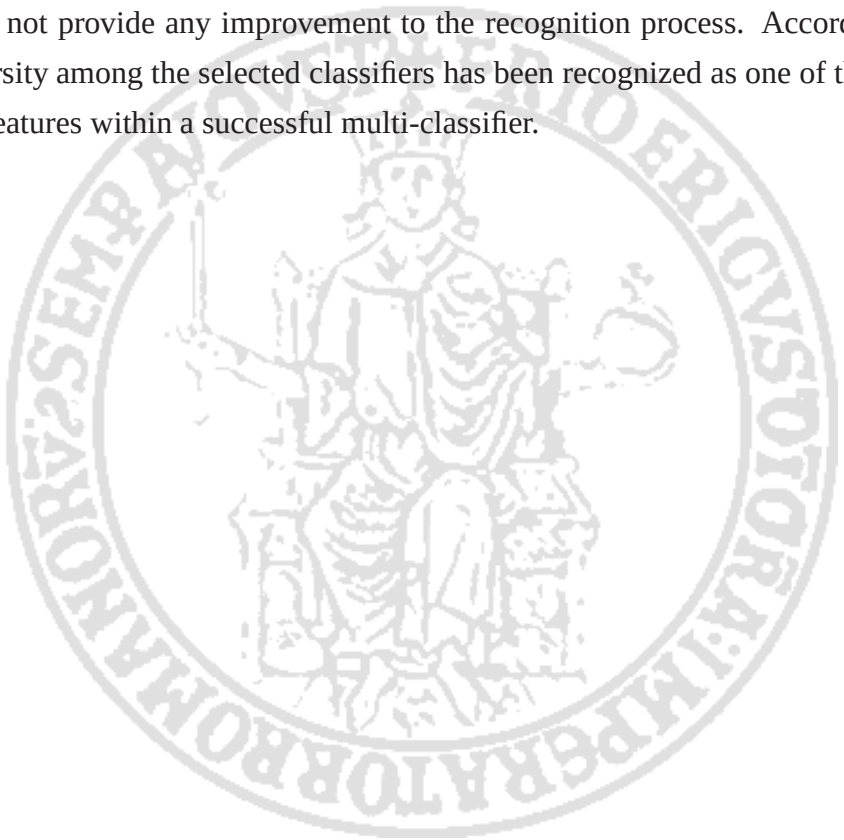
3.4.4 Random Subspace Method

The random subspace method (RSM) (Ho 1998) is a relatively recent method of combining models. Learning machines are trained on randomly chosen subspaces of the original input space (i.e. the training set is sampled in the feature space). The outputs of the models are then combined, usually by a simple majority vote.

3.5 Some Considerations

In conclusion, real-life situations are not as simple and straightforward. Most data sets are not of good quality and contain a substantial quantity of noise. Such erroneous data can mislead the training paradigm which can lead to wrong approximations. Secondly, most training paradigms have very clear-cut limitations on their operation. For example, the rule of thumb for the proper training of a neural network is that the paradigm should be presented with at least 10 times as much data as there are connections within the network. Less data can lead to the neural network reaching global minimums in its training error and consequently, returning bad approximations on the function. A small sized neural network with 2-3 hidden layers and 10 inputs will have at least 50 connections, which in turn leads to a requirement of at least 500 training samples for proper training. Most of the complex data sets currently being used are often of much higher dimensions and consequently require large networks for proper approximations. This in turn leads to the requirement of needing large data sets, which is often left unfulfilled. Due to the limitations mentioned above, it has been experimentally observed that the construction of a perfect classifier for any given task is often impossible. Therefore, the best that system designers have to work with are classifiers and paradigms which provide near approximations of the functions expected. Of course, when different paradigms are used to approximate the same function, the approximations generated can vary due to different interpretations of the data and noise being made. This diversity among different learning paradigms had lead to the development of the Multi-Classifier System (MCS), which attempts to combine the approximations of different training paradigms to obtain better results. Such systems are analogous to a company board of directors, where the board is usually constituted of people who have varying levels of qualifications and expertise. For example, a board is usually constituted of an economist, an accountant, a management consultant and a marketing consultant. It is very rare that a board will have one person who is specialized in all these fields of expertise, and are therefore compelled to make decisions in consensus with all the members of the board. A decision making process of this sort, where the final decision is gen-

erated by combining the opinions of all the members of the board is exactly how a MCS works. Of course, there can be many variations to this theme, where different members of the board could be given extra decision making capabilities based on the type of decision to be made. Intuitively, it makes sense that a combination classifiers or experts provides better results than a singular decision maker. However, this is dependent on how independent and diverse the individual classifiers are. If all the classifiers provide similar and correlated results, the aggregated result will not provide any improvement to the recognition process. Accordingly, the diversity among the selected classifiers has been recognized as one of the key design features within a successful multi-classifier.



Chapter 4

Self-Organizing Classifier ensemble for Adversarial Learning

In supervised classification we cannot work without labels that can be associated with our training data. Obtaining labels, hard or soft, is a process prone to errors. That means that a classification algorithm can have falsely labelled data in its training set, and this, in extreme cases, might render it useless. Sometimes the mislabelling samples could be forced by a training set contamination made by some malicious users (*Adversarial Learning*). This kind of training set contamination is also known as *Poisoning Attack* [1].

In this chapter we deal with to find out what is the impact of noise-contaminations on the labels, and how it is possible to clean a training set with a MCS approach. We will describe this kind of approach, named SOCIAL, and we made several experiments to verify the robustness to the noise and to the contamination (*smart*-noise) of a classifier trained with a *cleaned* training set.

We will show that the performance obtained by a simple classifier trained with the cleaned training set and by some "state-of-the-art" MCS trained on the original dataset, are comparable and sometimes the simple classifier is even better in terms of accuracy.

We will demonstrate that our system can move the computational complexity from the classification system to the training set *cleaning system*, giving advantages in terms of computational complexity, interpretation of the problem (for example through a set of rules) robustness in case of *adversarial learning* prob-

lems.

4.1 Some MCS approach for Label Noise

There is not much literature on how noise label should be modelled and dealt with an MCS approach.

AdaBoost [30] has shown to often improve the base learner accuracy. Since its introduction, it has been successfully applied to many problems. Furthermore, the AdaBoost idea has been extended to other sort of problems. Although it has wide-spread success, it is susceptible to the over-fitting problem as pointed out by Dietterich [21]. Oza [61] proposed an approach called AveBoost2 to smooth noise. This approach can be seen as a relaxed version of AdaBoost. When training examples are noisy and therefore difficult to fit, AdaBoost is known to increase the weights of those examples to excess and over-fit them because many consecutive base models may not learn them properly. AveBoost2s averaging does not allow the weights of noisy examples to increase rapidly, thereby mitigating the overfitting problem.

Thiel [73] made a comparison between the single classifier and an ensemble. In his paper the attention is focused on which impact a dataset with *soft* labels has on a noisy training set.

Melville and Mooney [57] introduced a new kind of multiple classifier system to take into account the noise label problem; they called it DECORATE. DECORATE, (Diverse Ensemble Creation by Oppositional Relabelling of Artificial Training Examples) uses an existing ”strong” learner (one that provides high accuracy on the training data) to build an effective diverse committee in a fairly simple, straightforward manner. This is accomplished by adding different randomly constructed examples to the training set when building new committee members. These artificially constructed examples are given category labels that disagree with the current decision of the committee, thereby easily and directly increasing diversity when a new classifier is trained on the augmented data and added to the committee.

4.2 The SOCIAL Approach

SOCIAL is the acronym of ***S**elf-**O**rganizing **C**lass**I**fier ensemble for **A**dversarial **L**earning* and is a Multiple Classifier Systems with a parallel topology (sec 3.2.3) where a statistical characterization of each base classifiers is dynamically updated by looking at the ensemble of these classifiers.

This system, after an iterative evolution, returns a cleaned training set. This result is obtained changing the labels assigned to the samples and considering the training set cleaned when these changes become stable.

SOCIAL is specifically designed to approach with training sets with noisy labels, i.e. for an *adversarial leaning* problem. The principle behind is that a community through a democratic approach can remove most of its own initial mistakes and so it can improve itself.

4.2.1 System Evolution and Terminal Condition

The main parameter used here is the *Degree of Truth*, hereinafter **DoT**[1]. This value is defined in the range $[0, 1]$ and it represents the probability that the labels assigned to the sample are corrects.

The DoT distribution requires to be initialized. Making the assumption that the noise distribution is unknown, a possible criteria is to assign 1 to the DoT for each sample. This means that we trust the labels assigned to the training set samples.

Another important parameter is the *Classifier Reliability*, hereinafter **R**. This parameter is associated to all the base classifiers and it represents a degree of belief on the correctness of the classifier with respect to the ensemble decisions.

The last important parameter is δ_{dB}, that is the value used for the terminal condition. This value is calculated as the ratio (in decibel) between the number of the samples that change their labels across two consecutive steps.

[1]The concept of *DoT* is often used in the context of *fuzzy theory* [69], in this case, statements are described in terms of membership functions, that are continuous and have a range $[0, 1]$. For example, given the measured value of a parameter, the membership function gives the *degree of truth* that the parameter is "high" or "low".

The system behaviour is characterized by:

1. a bootstrap step, in which SOCIAL put the $DoT = 1$ for each sample and $\delta_{dB} = +\infty$

2. an iterative evolution, in which the **R** associated to each base classifier is upgraded and the base classifiers are combined to redefine the DoT and the $label$ for each sample.

3. a terminal condition, in which there is a comparison between δ and a suitable threshold (τ).

In figure 4.1 the system evolution is represented. After the bootstrap phase, the system, iteratively, makes a base classifier statistical characterization and then combines all the classifiers' outputs weighted by their performance estimation in order to evaluate if the label of samples must change. Finally, the terminal condition is checked and, if it is matched, the system returns a "cleaned" training set as well as the **R** for each classifier.

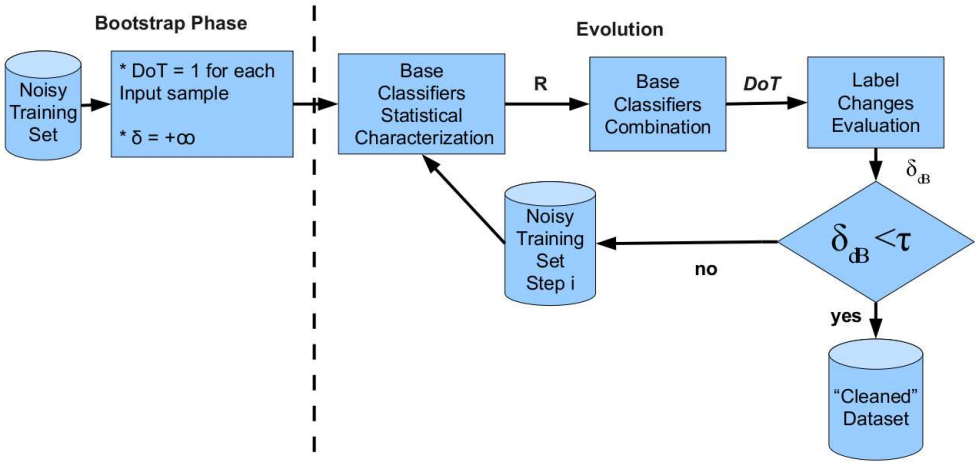


Figure 4.1: SOCIAL: Flow Diagram

4.2.2 Base Classifiers Statistical Characterization

All the information about the performance of a classifier with respect to a specific training set are given by the *Confusion Matrix* (sec 3.1.2). It represents how the errors are distributed across the classes. Starting from this matrix we have introduced a new type of matrix, that takes into account also the probability that a training pattern really belongs to a specific class. We called this matrix *Weighted Confusion Matrix*, hereinafter **WCM**.

	Predicted Class			
True Class	$\hat{C}_1$	$\hat{C}_2$	$\dots$	$\hat{C}_M$
C_1	e_{11}	e_{12}	$\dots$	e_{1M}
C_2	e_{21}	e_{22}	$\dots$	e_{2M}
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
C_M	e_{M1}	e_{M2}	$\dots$	e_{MM}

Table 4.1: Confusion Matrix (CM) for M-classes classification

In table 4.1 a general Confusion Matrix for an M-classes problem is shown.

Let us define the training set as $\{(x_1, y_1), \dots, (x_N, y_N)\}$, where the generic x_k is the k-th sample and y_k is its label. N_i represents the number of samples in which $y = C_i$, and N_{ij} represents the number of samples in which $y = C_i$ and $\hat{y} = C_j$, where $\hat{y}$ is the predicted label.

The difference between the *Confusion Matrix* and the weighted version $\mathbf{WCM}$ lies on how the elements e_{ij} are calculated. While in the *Confusion Matrix* the entry e_{ij} denotes the percentage of training set samples whose true class is C_i, and which are assigned by the classifier to class $\hat{C}_j$, (eq. 4.1), in the Weighted Confusion Matrix the same entry denotes the percentage of the training set whose true class is C_i, and which are assigned by the classifier to class $\hat{C}_j$, weighted by the DoT associated to each sample (eq. 4.2).

$$\mathbf{CM} \qquad e_{ij} = \frac{N_{ij}}{N_i} = \frac{\sum_{k=1:y_k=C_i \, and \, \hat{y}_k=\hat{C}_j}^{N} 1}{N_i}, \tag{4.1}$$

Sample	True Class	DoT	Predicted Class
x_1	P	0.8	N
x_2	P	0.6	P
x_3	P	0.9	P
x_4	N	0.2	N
x_5	N	0.5	P
x_6	P	0.7	P
x_7	N	0.4	N
x_8	N	0.9	N
x_9	P	0.3	P
x_{10}	N	0.2	P

Table 4.2: Training Set Classification Example with DoT value

$$\mathbf{WCM} \qquad e_{ij} = \frac{\sum_{k=1:y_k=C_i \, and \, \hat{y}_k=\hat{C}_j}^{N} DoT(k)}{N_i}, \tag{4.2}$$

For example, we can consider the binary classification problem in the table 4.2.

In this case the samples can belong to the classes *Positive* (P) or *Negative* (N) and the entry e_{ij} are evaluated as shown in the table 4.3. It is worth noting that the sum of the elements of each $\mathbf{WCM}$ row is always less than one, while in the case of CM it is always one. This is due to the DoT associated to each sample.

For the sake of brevity, in the following we explicitly evaluate only the entry e_{00} of the matrix, that is, when the *True Class* is P and the *Assigned Class* is $\hat{P}$.

$$\mathbf{CM} \qquad e_{00} = \frac{4}{5} = 0.8 \tag{4.3}$$

$$\mathbf{WCM} \qquad e_{00} = \frac{0.6 + 0.9 + 0.7 + 0.3}{5} = 0.46 \tag{4.4}$$

Starting from the $\mathbf{WCM}$, SOCIAL evaluates the *Classifier Reliability* $\mathbf{R}$ associated to each base classifier.

$$r : \mathbf{WCM} \longrightarrow \mathbf{R} \tag{4.5}$$

CM	Predicted Class		**WCM**	Predicted Class	
True Class	$\hat{P}$	$\hat{N}$	True Class	$\hat{P}$	$\hat{N}$
P	0.80	0.20	P	0.46	0.16
N	0.40	0.60	N	0.14	0.30

Table 4.3: Comparison between $\mathbf{CM}$ and $\mathbf{WCM}$ on the example in tab 4.2

It is possible to make a comparison between the *Probability Theory* and this problem. The $\mathbf{WCM}$ could be considered as a *probability density function* (pdf) while the $\mathbf{R}$ could be considered as a synthetic information extracted from the pdf, as for examples the mean (μ) or the standard deviation (σ).

The information that SOCIAL has to extract from the $\mathbf{WCM}$ depends on which type of fusion it uses. For example, if the fusion block is a Weighted Majority Voting (sec 3.3.2) then the $\mathbf{R}$ will be a vector of "weights" associated to each class, where the single value $R(C_i)$ is evaluated as shown in eq. 4.6.

$$R(C_i) = e_{ii}, \quad \forall i = 1, 2, \ldots, M \tag{4.6}$$

Another examples of function $r()$ will be described in the appendix A where the Dempster-Shafer (sec 3.3.4) combination rule is used as fusion block, and in the appendix B, where the Bayesian Combing rule is considered.

In figure 4.2 it is shown how the system evaluates the WCM starting from the training set. It is worth nothing that in the bootstrap phase the Weighted Confusion Matrix is the normal Confusion Matrix, because the DoT value are put to 1 for each sample.

Another important consideration is that the name SOCIAL directly derives from the DoT values that are evaluated from the ensemble in the previous step, that's why the classifier characterization is made with respect to the others classifiers.

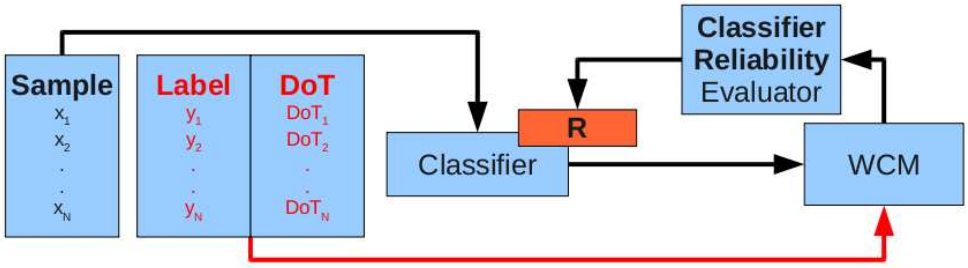


Figure 4.2: Statistical Classifier Characterization Schema

4.2.3 Base Classifiers Combination

After the system has characterized the base classifiers, it has to combine them to obtain the new $label$ and the new DoT for each sample as shown in the figure 4.3.

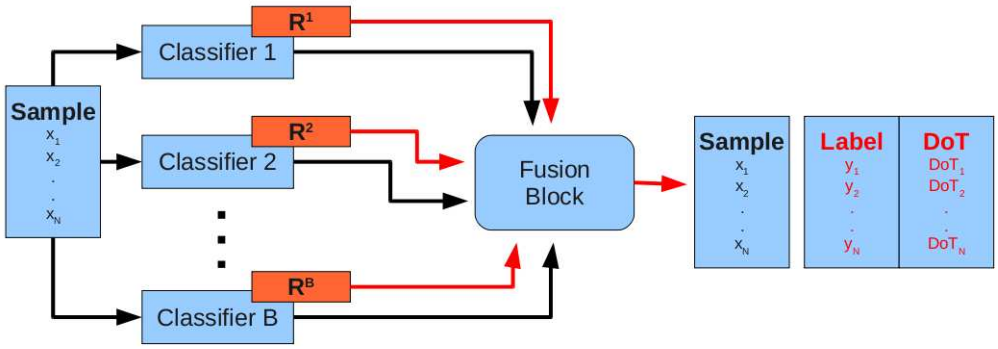


Figure 4.3: Base Classifiers Combination

The Fusion Block implements a function $f()$ defined as:

$$f : ((\hat{y}^1, R^1), \ldots, (\hat{y}^B, R^B)) \longrightarrow (y, DoT) \tag{4.7}$$

Where in eq. 4.7 $\hat{y}^i$ represents the output of the $i-th$ classifier by means of a suitable combining rule.

For example, in the case of using the *Weighted Majority Voting* (sec 3.3.2) as combining rule, $f()$ becomes:

$$f \longrightarrow \begin{cases} y = argmax_j \frac{\sum_{i=1:\hat{y}^i=C_j}^{B} R^i(\hat{y}^i)}{B} \\ \\ DoT = max_j \frac{\sum_{i=1:\hat{y}^i=C_j}^{B} R^i(\hat{y}^i)}{B} \end{cases} \tag{4.8}$$

Other examples of $f()$ will be discussed into the Appendix A and in the Appendix B.

4.2.4 Label Changes Evaluation and Terminal Condition

SOCIAL stops its iterations when the ratio δ_{dB} between the samples that change their labels in the step $t-1$ and that ones that change them in the step t is less than a threshold τ (eq. 4.10) and it will give the *cleaned* dataset.

$$changes(t) = \sum_{i=1}^{N} \Delta_i, \qquad \Delta_i = \begin{cases} 1, & if\ y_i(t-1) \neq y_i(t) \\ 0, & otherwise \end{cases} \tag{4.9}$$

$$\delta_{dB} = \begin{cases} +\infty, & t = 1 \\ 10 * \log_{10} \frac{changes(t-1)}{change(t)}, & Otherwise \end{cases} \tag{4.10}$$

We have experimentally proved that a good value for τ is $1dB$. It is worth noting that during the first step $\delta = +\infty$ and so the system can stop its iterations only starting from the second step.

4.2.5 The SOCIAL Algorithm

In this section we will describe the SOCIAL algorithm using the Weighted Majority Voting as fusion block.

Fig. 4.2.5 shows the pseudocode of the algorithm. SOCIAL has as input the training set $(x_1, y_1(1), \ldots, (x_N, y_N(1))$, the base models learning algorithms $L_1, \ldots, L_B$ and a threshold value τ for the terminal condition.

This algorithm, for each step t, maintains a distribution $DoT(t)$, where each element $DoT_i(t)$ is associated to the sample x_i. This distribution gives the probability that the sample x_i really belongs to the class y_i. During the bootstrap phase, $(DoT_i(1) = 1 \quad \forall i \in 1, 2, \ldots, N)$ for the motivations illustrated previously (sec 4.2.1).

As first operation, SOCIAL evaluates through a K-Fold Cross Validation approach, a function $h^b(t)$ that associates for each sample x_i, for each base classifier b and for each step t, a predicted class $\hat{y}_i^b(t)$. Starting from $\hat{y}_i^b(t)$, SOCIAL evaluates the **WCM** (sec. 4.2.2), where each entry is calculated as:

$$e_{ij}^b(t) = \frac{\sum_{k=1:y_k(t)=C_i \, and \, \hat{y}_k^b(t)=C_j}^{N} DoT_k(t)}{N_i} \tag{4.11}$$

Consequently, it evaluates the Classifier Reliability $\mathbf{R}^b(t)$ for each base classifier b and for each iteration t. The values $R^b(C_i, t)$ are evaluated for each possible class C_i starting from the weighted confusion matrix and calculating the *weighted accuracy* for each class:

$$R^b(C_i, t) = e_{ii}^b(t) \tag{4.12}$$

After the weights evaluation, it applies the **WMV** to each sample and updates the label $y_i(t)$ into the training set and the $DoT(t)$ distribution:

$$y_i(t+1) = argmax_j \frac{\sum_{k=1:\hat{y}_i^k(t)=C_j}^{B} R^k(\hat{y}_i^k(t))}{B}$$

$$DoT_i(t+1) = \max_j \frac{\sum_{k=1:\hat{y}_i^k(t)=C_j}^{B} R^k(\hat{y}_i^k(t))}{B}$$

At this point SOCIAL evaluates the number of samples that change their labels in the step t:

$$changes(t) = \sum_{i=1}^{N} \Delta_i, \qquad \Delta_i = \begin{cases} 1, & if \; y_i(t-1) \neq y_i(t) \\ 0, & otherwise \end{cases} \tag{4.13}$$

At the end, it evaluates the terminal condition, i.e. if the

$$10 * \log_{10} \frac{changes(t-1)}{change(t)} < \tau$$

SOCIAL returns the *cleaned* training set $(x_1, y_1(t)), \ldots, (x_N, y_N(t))$.

4.3 Experimental Results

To figure out how the system perform, we will show the results obtained for two kinds of problems, the first one produced with some synthetic distributions in which noise is added as described in the section 4.3.1, and another with some real scenarios in which the noise is added in a *smart* manner, i.e. imitating a possible malicious user that try to overcame the security system contaminating the training set. In all the tests we will make a comparison among the accuracy obtained through the worst base classifier trained with the training set *cleaned* by SOCIAL and the accuracy obtained by all the base classifiers and by the state of the art Multiple Classifier Systems on the *original* training set.

4.3.1 Noise Model

To experimentally determine the impact of label noise on classification accuracy, we need to artificially add noise according to a certain model. In a two-class case, a given portion of the training data would get randomly selected and the associated label flipped to the opposite class. This method can be extended to the multi-class case, with the label being changed to one of the other classes in a random manner.

4.3.2 Results with Synthetic Data

The first type of experiments are on three synthetic datasets reported in table 4.4.

We have considered three base classifiers:

- **Decision Tree** (DT): The algorithms that are used for constructing decision trees work by choosing a variable at each step that is the next best variable to use in splitting the set of items. *Best* is defined by how well the variable

$SOCIAL((x_1, y_1(1)), \ldots, (x_N, y_N(1)), L_1, \ldots, L_B, \tau)$

$\triangleright$ N is the number of samples, B is the number of base classifiers,
$\triangleright$ τ is terminal condition threshold, M is the number of the classes.

Initialize $DoT_i(1) = 1 \quad \forall i \in 1, 2, \ldots N.$
Initialize $t = 0$
Initialize $\delta = +\infty$
do
$t = t + 1$
for $b = 1, 2, \ldots, B,$
$\triangleright$ Classifier evaluation through a K-fold Cross Validation Approach:
$h^b(t) = L_b((x_1, y_1(t)), \ldots, (x_M, y_M(t)))$
for $i = 1, 2, \ldots, M,$
for $j = 1, 2, \ldots, M,$
$\triangleright$ **WCM** entries evaluation

$$e^b_{ij}(t) = \frac{\sum^N_{k=1:y_k(t)=C_i \; and \; \hat{y}^b_k(t)=C_j} DoT_k(t)}{N_i}$$

$\triangleright$ Classifier Reliability calculation:
$R^b(C_i, t) = e^b_{ii}(t)$
for $i = 1, 2, \ldots, N,$
$\triangleright$ Label y_i updating in the training set:

$$y_i(t+1) = argmax_j \frac{\sum^B_{k=1:\hat{y}^k_i(t)=C_j} R^k(\hat{y}^k_i(t))}{B}$$

$\triangleright$ DoT updating for each sample:

$$DoT_i(t+1) = \max_j \frac{\sum^B_{k=1:\hat{y}^k_i(t)=C_j} R^k(\hat{y}^k_i(t))}{B}$$

$\triangleright$ Label changes evaluation:

$$changes(t) = \sum^N_{i=1} \Delta_i, \qquad \Delta_i = \begin{cases} 1, & if \; y_i(t-1) = y_i(t) \\ 0, & otherwise \end{cases}$$

$\triangleright$ Terminal condition evaluation:
if $t > 1,$
$\delta = 10 * \log_{10} \frac{changes(t-1)}{changes(t)}$
while $\delta > \tau$
return The *cleaned* training set $(x_1, y_1(t)), \ldots, (x_N, y_N(t))$.

Figure 4.4: The SOCIAL Algorithm

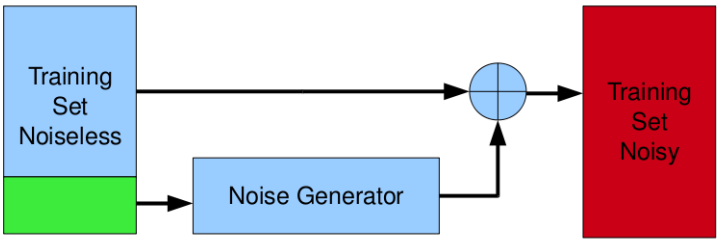


Figure 4.5: The noise generator algorithm.

Distribution	Training Set Samples	Test Set Samples	Classes
Gaussian	4000	1000	2
Mixture of Gaussians	4000	1000	2
Rotated Check Board $(45°)$	4000	1000	2

Table 4.4: Synthetic Datasets Description

splits the set into subsets that have the same value of the target variable. Different algorithms use different formulae for measuring *best*. We used the $C4.5$ Algorithm, in particular the $J48$ implementation of Weka [28]

- **Probabilistic Neural Network** (PNN): The Probabilistic Neural Network was introduced in 1990 by Specht [70] and puts the statistical kernel estimator into the framework of radial basis function networks. PNNs have gained interest because they offer a way to interpret the network structure in the form of a probability density function.

- **K Nearest Neighbourhood** (KNN) with $k = 3$: The k-nearest neighbour algorithm is amongst the simplest of all machine learning algorithms: an object is classified by a majority vote of its neighbours, with the object being assigned to the class most common amongst its k nearest neighbours (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of its nearest neighbour.

We choose these classifiers because they are conceptually different and they can increase the *diversity* among them. It is experimentally demonstrate that for a

Multiple Classifier Systems the diversity of the base classifiers is very important to increase the overall performance [46].

In order to compare, we choose four well known MCS approaches (sec 3.4); in particular:

- DECORATE: **D**iverse **E**nsemble **C**reation by **O**ppositional **R**elabelling of **A**rtificial **T**raining **E**xamples which uses an existing "strong" learner (one that provides high accuracy on the training data) to build an effective diverse committee in a fairly simple, straightforward manner.

- ADABOOST: **ADA**ptive **BOOST**ing, a machine learning algorithm, formulated by Yoav Freund and Robert Schapire. It is a meta-algorithm, and can be used in conjunction with many other learning algorithms to improve their performance. AdaBoost is adaptive in the sense that subsequent classifiers built are tweaked in favour of those instances misclassified by previous classifiers. AdaBoost is sensitive to noisy data and outliers. However it is less susceptible to the over-fitting problem than most learning algorithms.

- MULTIBOOST: **MULTI** class ada**BOOST**, that is an efficient implementation of the ADABOOST, with the possibility to consider multi-class problems.

- BAGGING: **B**ootstrap **AGG**regat**ING**, a machine learning ensemble meta-algorithm to improve classification and regression models in terms of stability and classification accuracy. It also reduces variance and helps to avoid over-fitting. Although it is usually applied to decision tree models, it can be used with any type of model.

General system evaluation with 30% of noise label

The first test is made using the three different synthetic distributions adding a 30% of uniformly distributed noise to them.

In table 4.5, the first four rows represent the accuracy obtained with the Multiple Classifier System trained with the original training set corrupted by a 30%

	Gaussian	Mixture of Gaussians	Rotated Check Board
Multi Classifiers System			
ADABoost	$95,70\%$	$80,60\%$	$74,10\%$
MultiBoost	$95,70\%$	$78,60\%$	$72,30\%$
Decorate	$96,60\%$	$82,00\%$	$84,20\%$
Bagging	$96,40\%$	$79,60\%$	$89,40\%$
Base Classifiers			
Decision Tree	$80,40\%$	$68,40\%$	$70,00\%$
KNN	$86,30\%$	$71,00\%$	$71,00\%$
PNN	$92,60\%$	$75,80\%$	$75,80\%$
Decision Tree trained with the obtained *clean* Dataset			
Decision Tree*	$97,10\%$	$82,40\%$	$91,00\%$

Table 4.5: Synthetic results with 30% of label noise on the training set

of label noise, the successive three rows represent the accuracy obtained with the three base classifiers trained on the original training set. Finally, the last row represents the accuracy obtained with the worst base classifier, in this case the DT, trained with the training set cleaned by SOCIAL.

It is worth noting that SOCIAL makes the classification problem simpler than the original one, and even the worst classifier trained with the *cleaned* training set becomes better, in terms of accuracy, than all the MCSs approaches used and as well as all the base classifiers.

In the figures 4.12, 4.13 and 4.14 is shown how SOCIAL modifies the training set, and how the δ parameter changes, for the three considered datasets.

In particular in each one of them, the first picture represents the accuracy behaviour across the steps. It is worth noting that the behaviour is always the same for each of the three datasets, i.e. there is a first moment in which the accuracy improves, and it corresponds to an *effective* cleaning of the training set, and a second moment, in which the accuracy decreases; it corresponds to a smoothing of the original distribution and a loosing of some information contained in.

It is possible to find the same information in the second picture in which different values of δ_{dB} are represented across the steps. In this case we are monitoring the variation between two consecutive steps. The value is in dB, so that if there

isn't any variations between two consecutive steps, we have a value equal to 0. We experimentally noticed that if we did not want to compromise the initial distribution, and want to preserve most of the information contained in it, a good value for the threshold τ is $1dB$. In this picture the violet dot-line represent an interpolation among three consecutive points of the δ_{dB} line, i.e the blue one, this is due to the fact that sometimes the original line, especially with low level of noise, becomes unstable, and it is difficult to find the correct output step.

The other ten pictures represent a scatterplot of the distribution in each step. The output step is indicated in bold, i.e. the scatterplot of the *cleaned* training set.

In the table 4.5 are shown the results obtained. It's worth nothing that the Decision Tree trained with the *cleaned* training set by SOCIAL, always increases the performance of the base classifiers, and, the *cleaned* training set shows a classification problem simpler than the original one, as demonstrated by the fact that the worst base classifier (DT) obtains an high level of accuracy using the *cleaned* training set.

Some Considerations

For the sake of brevity we presented only some tests on the *rotated check board* dataset to make some considerations that seems to be valid in general.

The first test arise with the evaluation of the SOCIAL robustness to the noise. In this case we added a different percentage of noise to the training set, and we evaluate the accuracy of the DT and the other MCS approaches under test (figure 4.6).

In the figure 4.7 it is shown a comparison among the base classifiers accuracy and the accuracy obtained a Decision Tree trained with the clean training set. Also in this case it's clear that the system is more robust to the noise with respect to the single classifiers.

Another important result is that the system reaches to recognize if the training set is noisy or not. In the figure 4.8 is shown the percentage of the samples that change their labels in the first iteration with respect to the number of training set samples. This percentage is linear dependent to the percentage of added noise.

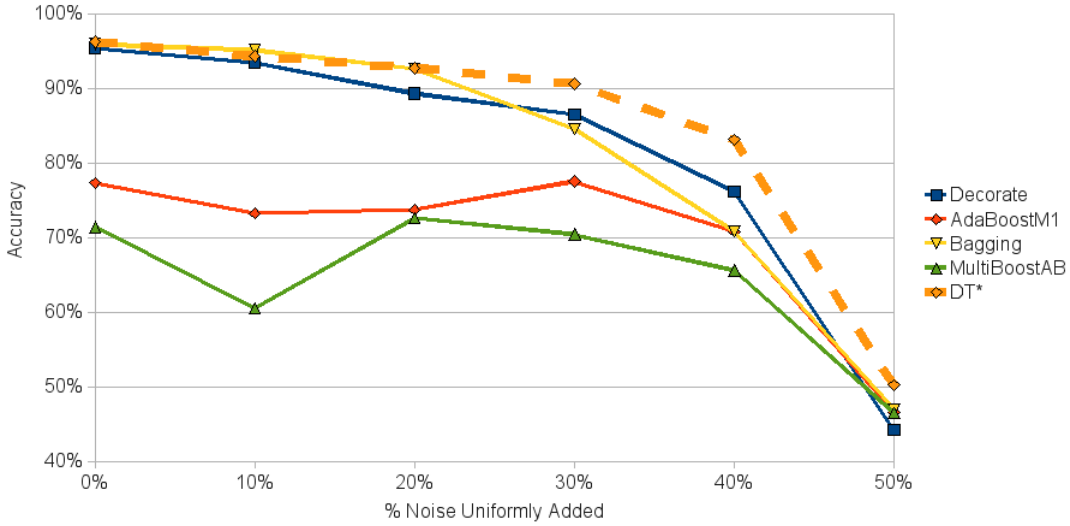


Figure 4.6: Accuracy Comparison with MCS for different % of noise

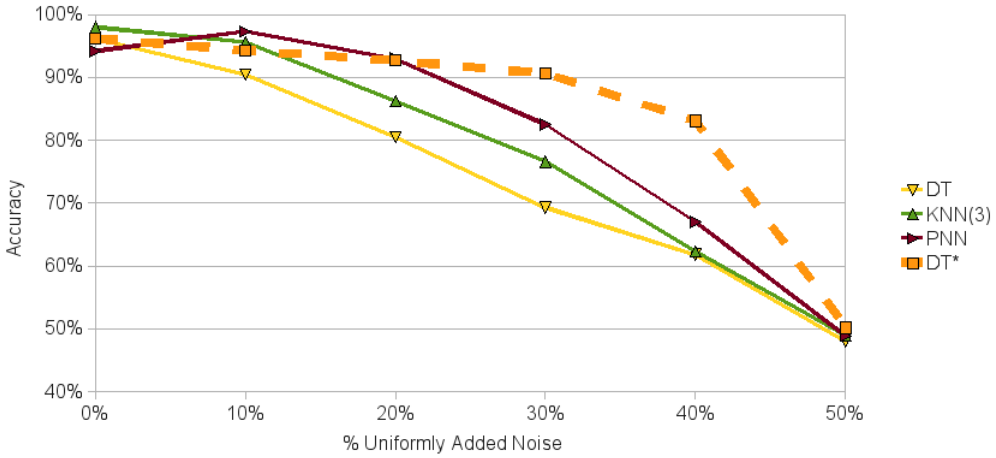


Figure 4.7: Accuracy Comparison with base Classifiers for different % of noise

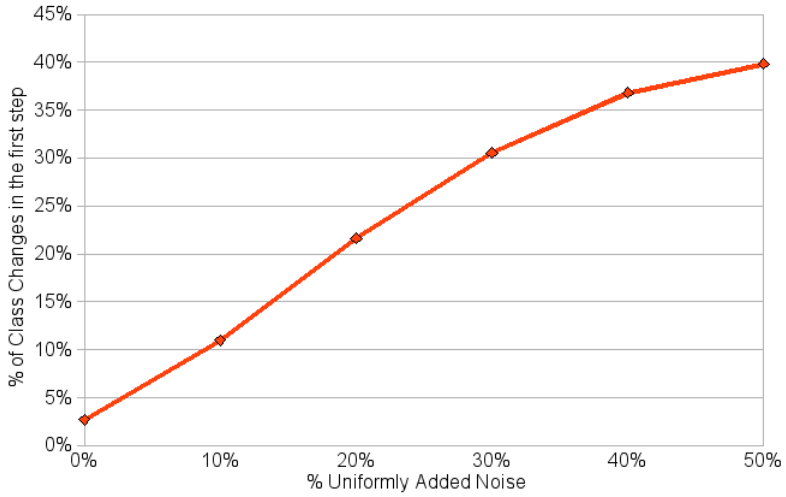


Figure 4.8: % Class changes in the 1^{st} step for different % of noise

4.3.3 Results with real data

We made also two case studies with real data, in particular, in the first one we used some internet packets traces, where the classes were *attack* or *normal*, this dataset was extracted by a larger one presented in the paper [71].

	Attack	**Normal**
Training Set	1540	2400
Test Set	386	600

Table 4.6: Traffic Dataset Description

In this case we added smart noise, simulating that a malicious user put some new attacks in the network, or make a poisoning training set attack, contaminating the training set with some samples that are considered falsely *normal* packets.

A dataset description is made in the table 4.6.

Also in this case the system recognize the presence of noise into the training set. We can monitor this situation giving a look to the percentage of class changes in the first step as it is possible to see in figure 4.9.

We tested SOCIAL with this new data; by giving a look to a comparison with

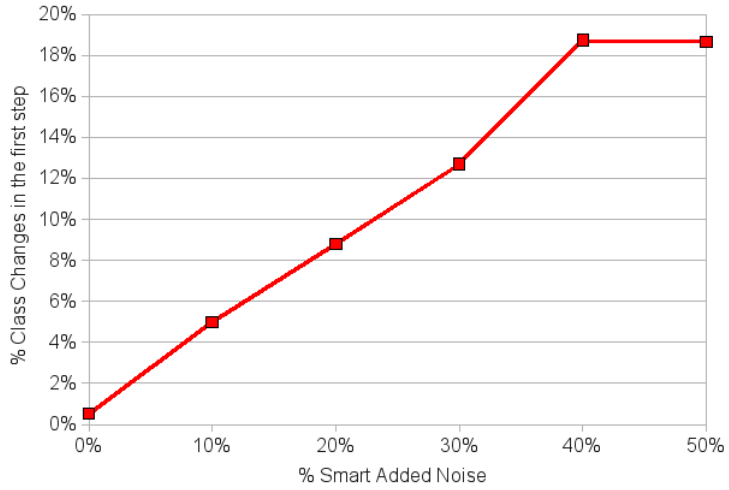


Figure 4.9: % Class changes in the 1^{st} step for different % of noise

the other MCS approaches, in this case SOCIAL's performance are comparable with the other ones (figure 4.10). But it's worth nothing, that also the Decision Tree, trained with the the *clean* training set reaches the same performance of SOCIAL, sometimes it is better than it. The point is that the system can reach the same performance with very simple classifier, and so with a lower computational complexity. An other advantage is the possibility to easily understand the *main* rules behind the classification problem by using a rule generator after the dataset cleaning up.

As an example, we will show how SOCIAL cleans the training set in the case of 30% of *contamination*, figure 4.11

In the chapter 5 will be shown another example for the traffic flow identification.

4.3.4 Key Findings

We find out a methodology that try to *clean* a training set from the noise by using a MCS approach. This system is designed to work in an adversarial learning context, in which a malicious user try to camouflage the training pattern to overcome the classification system. We noticed that SOCIAL reaches a good level of

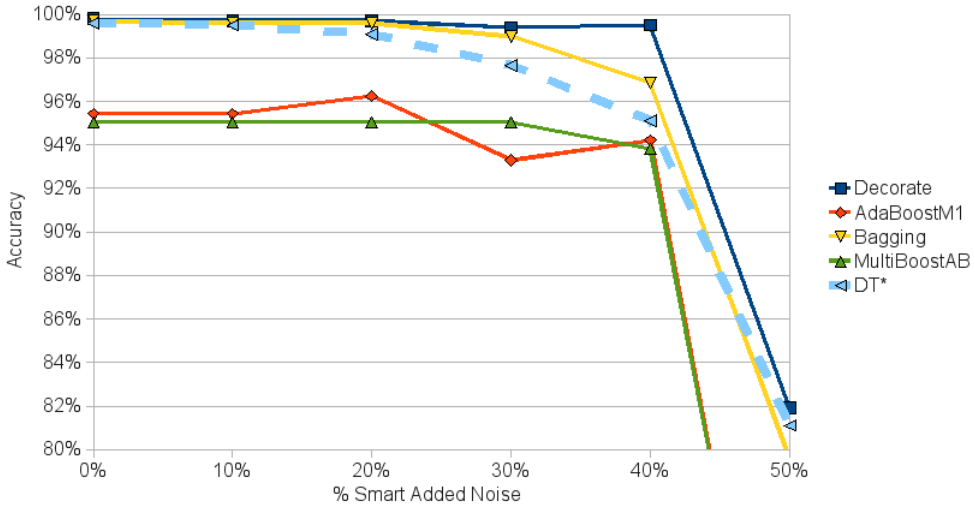


Figure 4.10: Accuracy Comparison for different % of Smart-noise

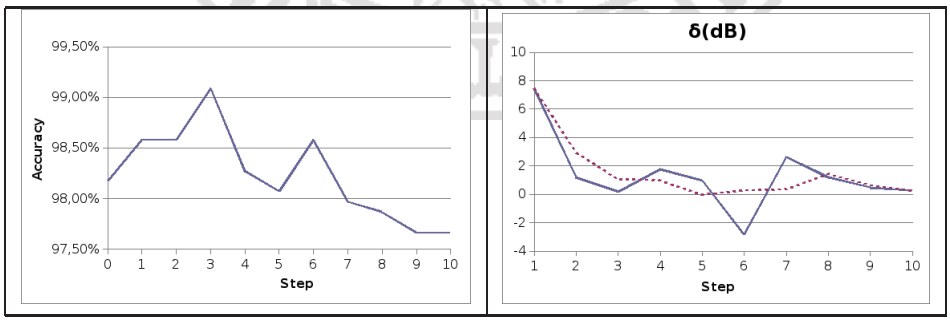
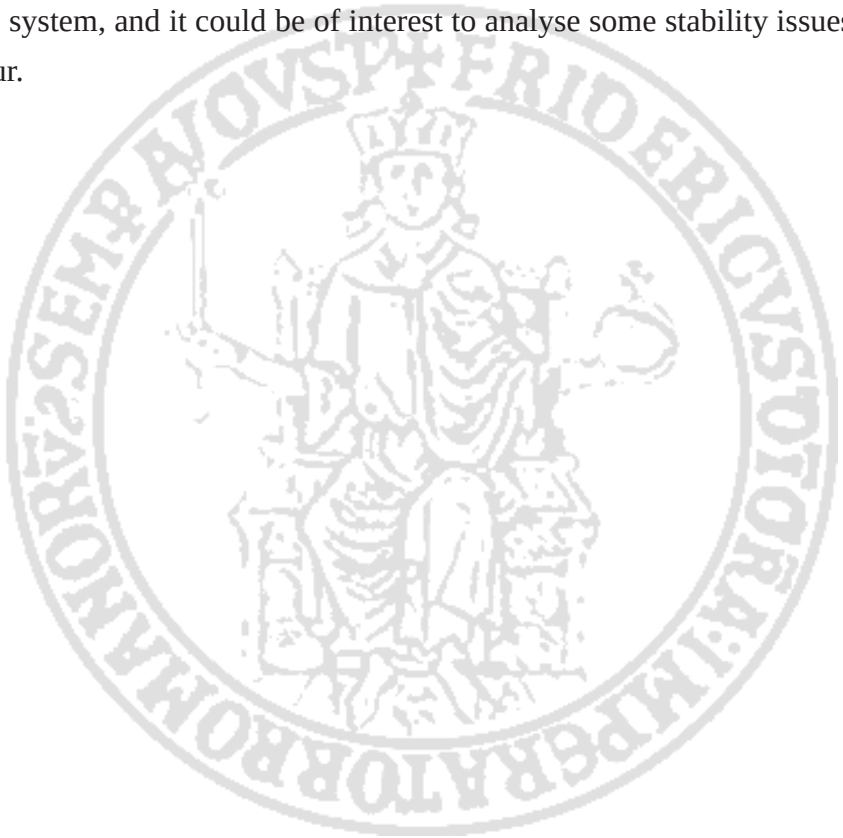


Figure 4.11: Traffic Intrusion Evolution: SOCIAL and DT

robustness to this kind of noise, and it gives a clean dataset that could be also used in a faster and easier classification system. In this way it is possible to overcame the computational complexity linked to the SOCIAL architecture.

SOCIAL gives rise to a simple classification problem. The cons of this is, that if the system is not stopped in time, the sample distribution could be modified, so *damaging* the dataset and the possible performance of the classifier.

As regard to the convergence, the system could also be seen as a non linear dynamic system, and it could be of interest to analyse some stability issues of its behaviour.



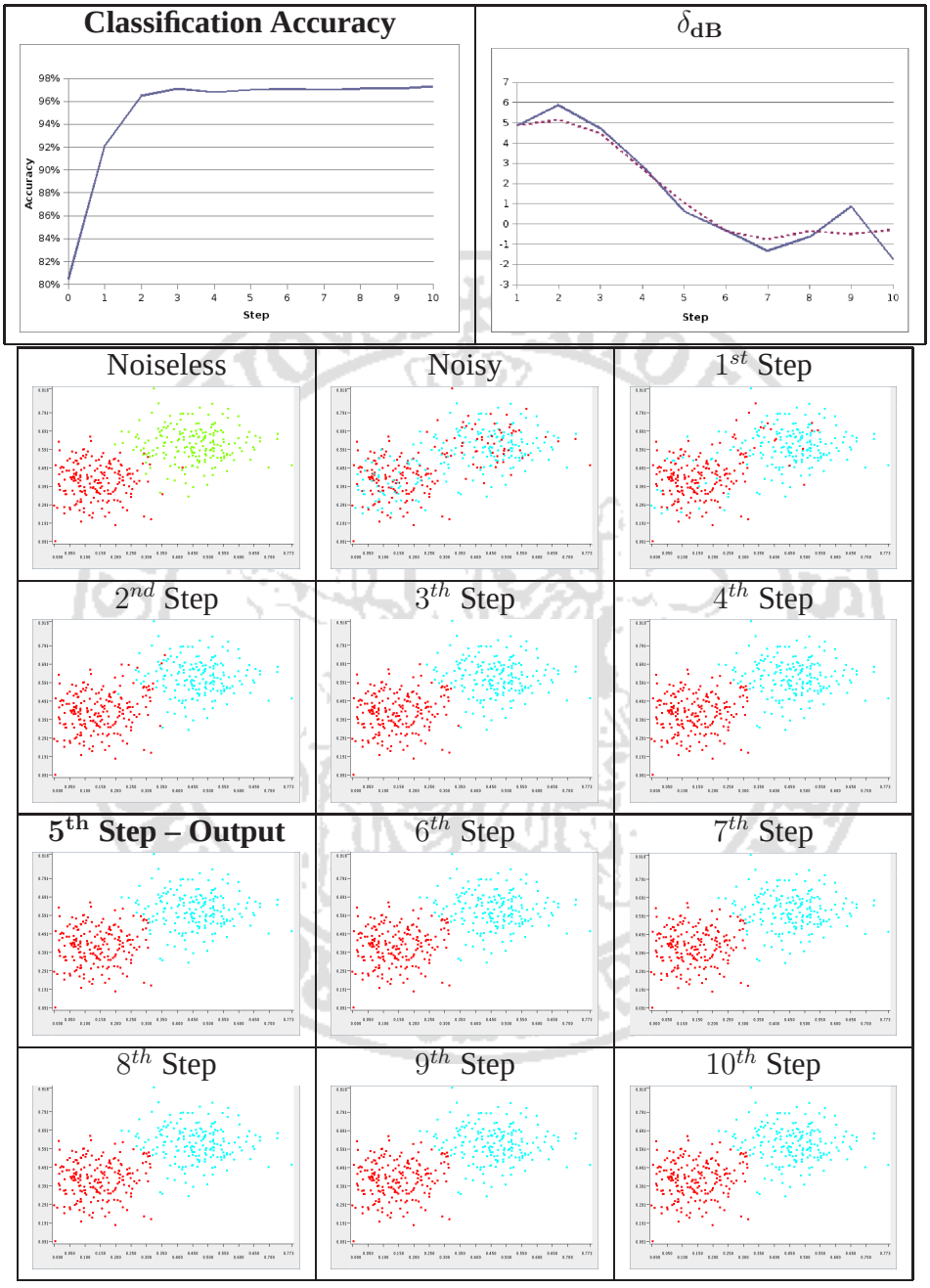


Figure 4.12: Gaussian Distribution starting from 30% of Noise

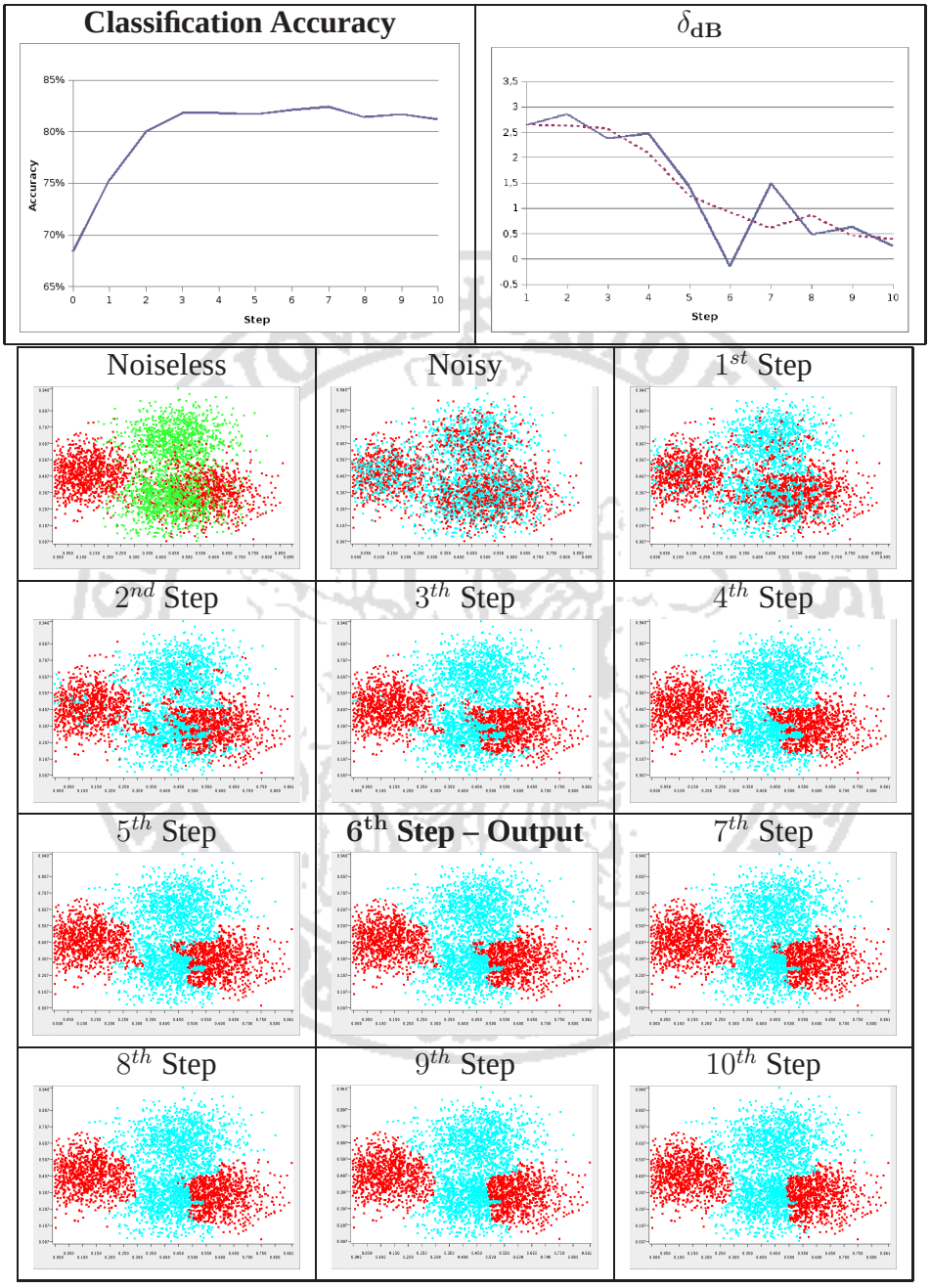


Figure 4.13: Mixture Of Gaussian starting from 30% of Noise

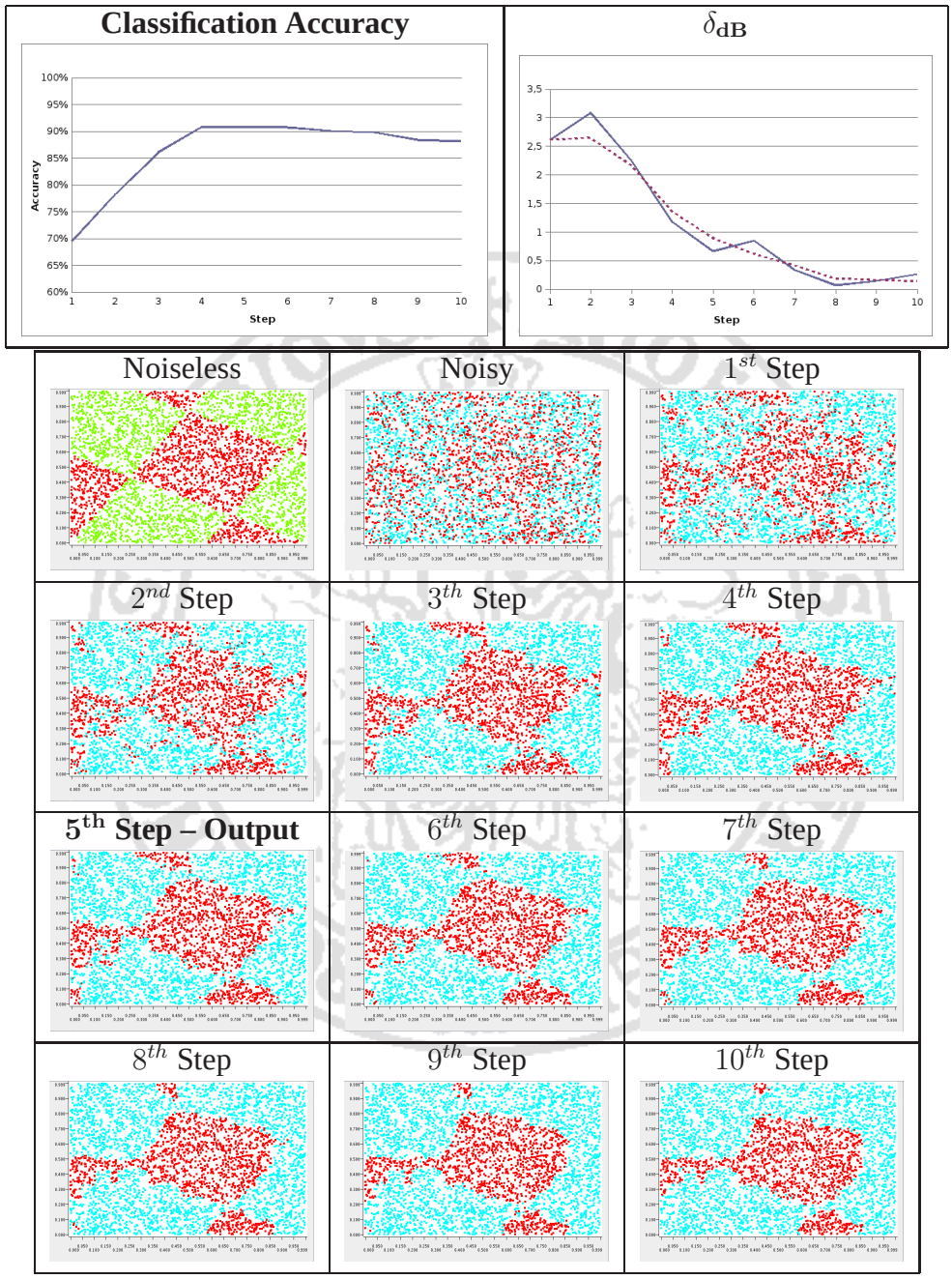


Figure 4.14: Rotated Check Board ($45°$) starting from 30% of Noise

Chapter 5

Network Protocol Verification by a Classifier Selection Ensemble

In the last years, networking research has started facing a problem not foreseen when the first *Internet* protocols were originally designed: network traffic classification, that is, associating traffic flows to the applications that generated them [56]. Originally each network application used known protocols and transport-level ports that easily allowed their identification. Since a few years back, this is not true any more [42, 59]. The number of network applications using proprietary undisclosed protocols has grown at an incredible rate (Skype, P2P-IPTV); the typical association application/port is often forged; in some cases traffic is encrypted, whereas sometimes it is encapsulated into traditional protocols. Beyond the need to understand which kind of traffic is carried on Internet links, the identification of traffic hidden in flows using well-known ports represents a challenging task. For these reasons, new approaches to traffic identification are needed. By traffic identification here we mean identification of a particular (or a group of) applications of interest.

This is a typical case study for the *Adversarial Classification* problem. In this case some malicious users try to overcame the classification system in different ways.

In this chapter we propose a novel identification technique based on packet-level information aiming at exploiting behavioural characteristics of different applications. Specifically, we will describe a method that use of both the *sign pat-*

tern and the *sizes* of the first four packets of each flow to label the flow as accepted (identified) or rejected. The accepted class is the protocol/application conventionally associated with the respective port number. The rejected class is related to applications that try to hide their presence typically with the purpose to circumvent network usage/security policies. The proposed approach is aimed at a high accuracy of identification, being at the same fast and universal. First, it uses the direction signs and the sizes of only the first four packets of each flow (targeted to work online), and second, it does not need to access the payload of the packets (does not affect privacy and works with encrypted packets).

The chapter is organized as follows. Section 5.1 discusses briefly our motivation. Section 5.2 provides details about the techniques at the base of our identification approach. Section 5.3 describes the dataset and the measurement approach used in the experimental validation. We show results of identification of "port 80" traffic in Section 5.4. At the very end, we try to apply SOCIAL, the algorithm presented in Chapter 4, to clean the training sets and we will compare it with the proposed approach.

5.1 Motivation and Related Work

Even if commonly considered unreliable, the classification/identification approach known as *port-based* is still used today for online monitoring. Its advantages are simplicity and speed, as it checks only a single packet-header field. Besides, in some real-world situations there are no effective alternatives. An immediate alternative proposed in the literature (and promptly adopted by the industry) are the *payload-based* approaches based on the inspection of the transport-level packet payload (the data produced by the application). These techniques usually compare packet contents against known signatures of application-level protocols. Such techniques were initially considered very reliable, and were used to build reference data in the evaluation of novel classification approaches [62, 43]. Today, however, their reliability and applicability are undermined by a number of factors. First, there are continuously arising undisclosed proprietary protocols and

techniques of protocol obfuscation (e.g., eMule/eDonkey). Second, several new-generation applications (e.g., instant messaging, file sharing) make use of traditional protocols (e.g. HTTP) to encapsulate their traffic, which deceives *payload-based* classifiers into erroneously associating the traffic to the encapsulating protocol. Third, new-generation applications (e.g., Skype) use packet-payload encryption techniques. In addition, network-level (e.g., IPSEC) and application-level (e.g., ssh) encryption tunnels are being increasingly used in the Internet. Even when they are feasible, *payload-based* approaches face further difficulties: (i) payload inspection requires accessing all user-transmitted data, which may breach privacy laws in some countries; (ii) the computational resources required to inspect the entire content of the packets is usually quite high, making it difficult to deploy such techniques when the traffic volume is large. Because of the growing problems with the *payload-based* approaches, new *statistical-based* classification approaches have been proposed that do not need access to packet content. These approaches use flow characteristics as features to train classifiers from the state-of-the-art machine learning. The explosion of high-quality scientific literature in this field [60, 25, 81, 4, 8, 26, 44, 8, 82, 76, 67, 44, 17] testifies the great interest in researching novel and accurate techniques for traffic classification. It has been demonstrated that the *statistical-based* approaches can achieve high accuracy, and that they appear to be the most promising approaches to face problems like protocol obfuscation, encapsulation, and encryption [79, 8].

In this chapter we propose a technique for the identification of hidden traffic flows using non-intrusive features and based on machine learning drawing upon a recent study by Gargiulo *et al.* [32]. We carry out an extensive experiment with an ensemble of Decision Trees where the input features are the sizes of the first four packets with payload, and the ensemble member that makes the decision is chosen by the combination of signs of these packets.

5.2 The Identification Approach

The requirement for operational speed brings in the idea of classifier selection ensemble where only one of a set of experts has to make a decision [32, 47]. The ensemble consists of member classifiers (*experts*) and an *oracle* that authorises one of the classifiers to pass its decision as the ensemble decision. Generally speaking, the oracle may have pre-defined regions of competence for the classifiers [66] or dynamically allocated regions [78]. Gargiulo *et al.* [32] propose to use the port number as the oracle determining the regions of competence. The directions and sizes of the first four packets of the TCP flow are then used as the features in a further 2-stage classifier (Figure 5.1). The features and the modular architecture were chosen so that the classification is both fast and accurate, and new modules can be trained and added to the system without re-training any already trained part.

5.2.1 The Features

Following [4],[27] and [29], we propose to use only the first four packets and to use the following features (see Section 5.3.2 for details on feature extraction):

• x_0, the port number;

• x_1, x_2, x_3, x_4, the directions of the first four packets, $x_i \in \{0, 1\}$, where 0 means that the packet is transferred from server to client, and 1, from client to server;

• s_1, s_2, s_3, s_4, the payload sizes of the first four packets, where s_i are positive integers. As in [27], we do not consider packets without payload because they are related to connections state information.

5.2.2 Stage 1: Sign Pattern Filter

To illustrate the system we use a data set consisting of network traffic traces at the University of Brescia (Italy) [27]. The known protocols in the training data are: POP3, SMTP, HTTP, msn, FTP and BitTorrent. Table 5.2.2 shows a summary of the training data as distributed across the 16 possible patterns of signs $[x_1, x_2, x_3, x_4]$, from 0000 to 1111.

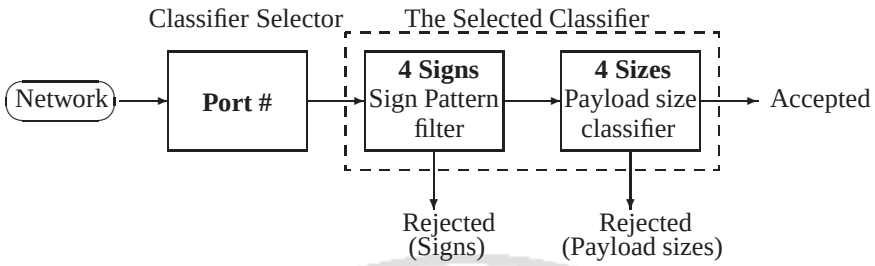


Figure 5.1: The generic classifier ensemble architecture.

Signs	Protocol and port number					
	POP3	FTP	SMTP	msn	BitTorr	HTTP
123 4	110	21	25	1863	6881	80
000 0	0	138	16	0	0	3
000 1	1	75	55	0	0	0
001 0	21	216	543	0	0	0
001 1	0	0	4	0	1	0
010 0	749	21	604	1	0	0
010 1	18823	5845	18186	0	1	0
011 0	17	1	18	0	1	0
011 1	0	0	1	0	0	0
100 0	0	0	0	328	23	5348
100 1	0	0	0	30	520	240
101 0	0	0	0	660	3609	826
101 1	0	0	0	4	753	12
110 0	0	0	0	1	8	427
110 1	0	0	0	0	87	76
111 0	0	0	0	0	9	108
111 1	0	0	0	0	45	23

Table 5.1: Summary of the Brescia network traffic data (training).

The table shows that groups of protocols can be distinguished by the sign patterns. For example, protocols msn (1863), BitTorrent (6881) and HTTP (80) hardly ever begin with a packet from sever to client ($x_1 = 0$). The table suggests that the sign patterns can be used to filter out very quickly flows that do not match the pattern of the class they are supposed to be a part of. In Figure 5.1, this is labelled as the *sign pattern filter*. In this paper we focus on the TCP traffic on port 80, so flows with patterns beginning with $x_1 = 0$ will be rejected by the filter. Next, using the training data, we can choose a rejection threshold, of say, 2%, and filter out all sign patterns where the number of flows is below the threshold. With this filter in place, the "allowed" combinations of signs for the HTTP protocol (80) are 1000, 1001, 1010, and 1100. All other protocols will be rejected by the sign pattern filter.

5.2.3 Stage 2: Decision Tree classifier using payload sizes

A separate classifier is then trained for each sign combination that passes through the sign filter. Here, each classifier has to solve a two-class problem: match versus mismatch of the protocol/application conventionally associated with the respective port number. We chose the *Decision Tree* [24] classifier, since its classification speed makes it very effective for an online implementation [76] and it does not assume any type of probability distribution of the data [32].

The decision process of a Decision Tree classifier is intuitive, since it can be traced as a sequence of simple decisions. The first decision is made at the root; depending on the answer, a branch is selected and the child node is visited. Another decision is made at this node, and so on, until a leaf is reached. The leaf contains a single class label, which is assigned to the object being classified. In our case the C4.5 algorithm was employed for constructing the Decision Tree classifiers. We used the Weka implementation, called J48[1].

The choice of a Decision Tree classifier can be justified by the following example. Figure 5.2 shows the scatter-plot of a dataset of traffic traces taken from the

[1]Weka is an open source collection of data-mining tools and is freely available at the website http://www.cs.waikato.ac.nz/ml/weka.

University of Brescia, Italy (UNIBS, see Section 5.3.1 for the dataset description). The data is filtered so that only flows with sign pattern $[x_1, x_2, x_3, x_4] = 1010$ are displayed. The (x, y) coordinate axes are the first two size features, s_1 and s_2, respectively. The figure shows three protocol classes: BitTorrent (3609 flows), HTTP (826) and msn (660). Two classification regions – HTTP vs the other two classes – can be clearly distinguished. Class HTTP seems the predominant class in Figure 5.2, however, this is not the case. Classes BitTorrent and msn are extremely dense, and are located towards the bottom left corner of the scatter-plot. Figure 5.3 displays an approximation of the 2-D densities of the three classes. The well delineated classification regions of high density suggest that a Decision Tree classifier would be the most suitable choice.

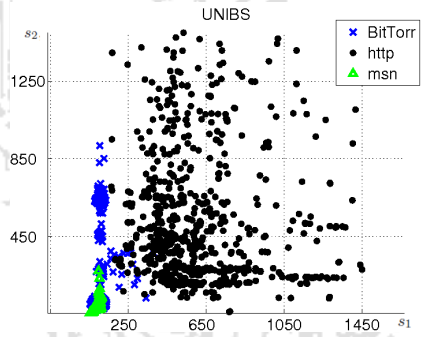
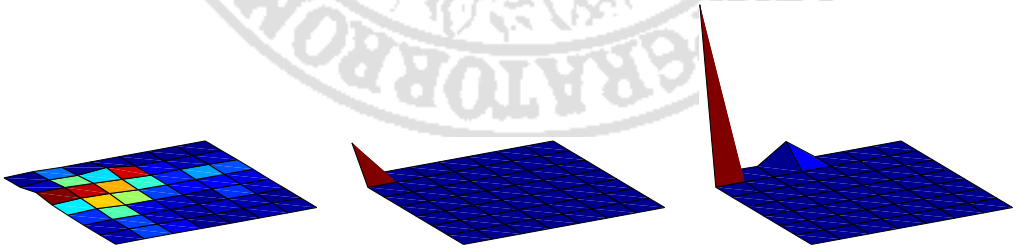


Figure 5.2: Scatter-plot of the UNIBS training data, first two size features.



(a) HTTP (b) msn (c) BitTorrent

Figure 5.3: 2D density of the UNIBS training data with sign pattern 1010.

5.3 Dataset and Feature Extraction

5.3.1 Dataset

To validate the proposed approach we used training datasets from three different institutions: *University of Brescia in Italy (UNIBS)*, *Lawrence Berkeley National Laboratory (LBNL)* and *Cooperative Association for Internet Data Analysis (CAIDA)*. A summary of the content of the three data sets used to train our system is given in Table 5.2. As testing set we used traces from *University of Napoli in Italy (UNINA)*. From this network we collected and used traffic traces related to two different time periods, 2004 (hereinafter denoted as *UNINA2004*) and 2009 (hereinafter denoted as *UNINA2009*). Details about the flows composing these traces are reported in Table 5.3.1.

Table 5.2: Number of flows in the three training data sets.

Protocol	Port	UNIBS	CAIDA	LBNL
POP3	110	19611	9591	1172
SMTP	25	19427	11831	20825
HTTP	80	7063	5930	81984
FTP	21	6296	1652	–
BitTorrent	6881	5057	–	–
msn	1863	1024	–	–
netbios-ssn	139	–	4575	–
HTTPS	443	–	25427	18013
oms	4662	–	–	1716
IMAP4	993	–	–	7677

Protocol	Port	UNINA2004	UNINA2009
HTTP	80	506795	144042
non-HTTP	80	2245	803

Table 5.3: Number of flows in the UNINA data sets used for testing.

Evaluating machine learning algorithms for automated network application identification. As explained in the next Section, in this work we focus our exper-

iments on the identification of HTTP traffic flowing through port TCP 80. Thus, during the training phase, for each sign pattern we train the corresponding payload size classifier (a Decision Tree) assigning all the HTTP flows to one class (the one corresponding to traffic to be accepted), and all the other flows (e.g. from msn, BitTorrent, etc.) that match the considered sign pattern as the other class (traffic to be rejected). For example, in the case of the 1010 combination for the UNIBS dataset we train the classifier with HTTP against msn and BitTorrent (see Table 5.2.2).

5.3.2 Feature Extraction

To extract the nine features we used ***TIE (Traffic Identification Engine)*** [16], an open-source multi-classifier system whose architecture is shown in Figure 5.4.

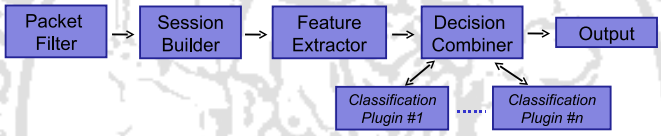


Figure 5.4: Overall Architecture of TIE.

For this work we used TIE for (i) processing and filtering traffic traces, (ii) aggregating packets into sessions, and (iii) extracting features. The features produced by TIE have been fed to a prototype implementation of the identification approach described in Section 5.2. Moreover, we used TIE with a classification plugin based on a payload-inspection technique in order to establish the "ground-truth" of the given traces. This allowed us to label each flow and to evaluate the accuracy of our approach (see Section 5.4). We looked at the payload content using the *TIE-L7* plugin module, which implements the Linux L7-filter classification technique[2].

In the experiments presented we focused on traffic on TCP port 80. The *Packet Capture* TIE module filters out all traffic not pertaining to the port, whereas the TIE module named *Session Builder* is responsible for aggregating the remaining

[2]L7-filter is an application layer packet classifier for linux and is freely available at the website http://l7-filter.sourceforge.net.

packets into bidirectional flows (*biflows*). That is, we consider the common definition of flow tuple while taking into account traffic in both directions: upstream and downstream. The upstream direction is the one of the first packet observed. Moreover, because we are examining TCP traffic, instead of a time-out value we use simple heuristics based on SYN, FIN, RST flags in TCP headers, in order to approximate TCP connections (as described in [16]).

The *Feature Extraction* module is responsible for extracting classification features from each biflow. In order to take into account only properties related to the application, we record the sizes of the transport-level payload, excluding pure TCP packets that do not carry application-level data (e.g., empty ACK packets). The payload sizes are stored in the order they are observed. A sign is added depending on the packet direction, plus for upstream and minus for downstream. Each biflow is assigned a session_id, which can later be used to manually examine the biflow, or to process again the same traffic trace using TIE classifiers and checking the results (as in Section 5.4).

5.4 Experimental Results

In this section we show the results obtained with the proposed identification architecture. In particular, we want to demonstrate that if we train the system with traffic traces taken from different sites (spatial invariance) and in a different time (temporal invariance) we can correctly accept HTTP traffic and reject non-HTTP traffic.

To show the results we choose the following metrics:

- **Overall Accuracy**: The percentage of correctly classified flows.
- **HTTP Accuracy**: The percentage of the correctly classified HTTP flows out of all true HTTP flows (sensitivity).
- **non-HTTP Accuracy**: The percentage of the correctly classified non-HTTP flows out of all non-HTTP flows (specificity).

For assessing the effectiveness of the proposed approach in different time periods (temporal invariance), we carried out cross-testing using UNINA2004 and UNINA2009 traces. In order to verify the spatial invariance of the approach, we train the system with LBNL, CAIDA and UNIBS traces, and then test the system with both UNINA2004 and UNINA2009 traces.

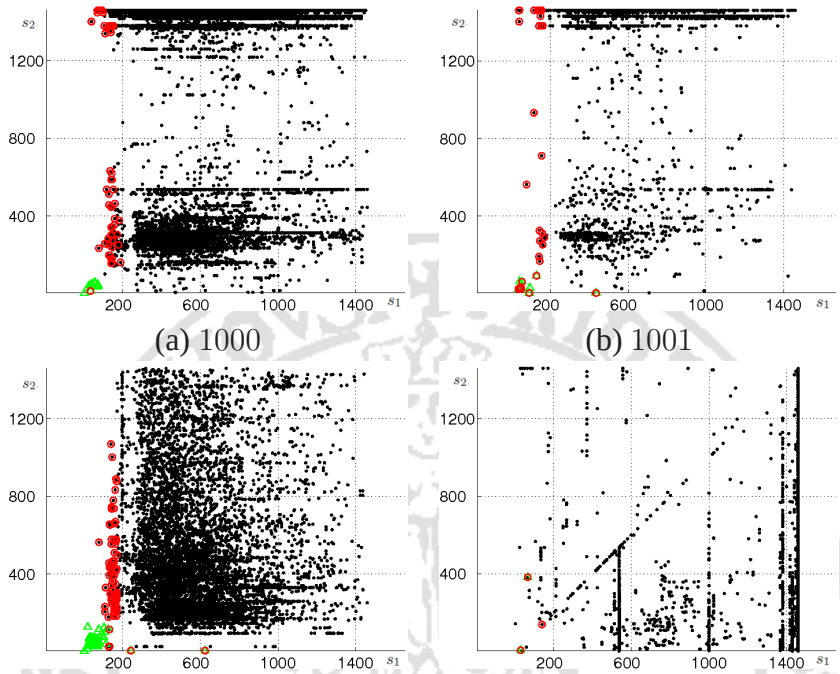
Table 5.4 is obtained by training and testing the system by using a 10-fold cross validation protocol. That is, the whole dataset is divided into 10 folds; 9 of them are used to train the classifiers and the last fold is used for testing. This is carried out for all 10 folds and results are reported as average accuracies. The table shows high accuracy for UNINA2004, and hints about the diversity of non-HTTP traces that might have caused the low specificity for UNINA2009.

Table 5.4: Results obtained using UNINA2004 and UNINA2009 datasets

	Overall Accuracy	HTTP Accuracy	non-HTTP Accuracy
UNINA2004	99.97%	99.99%	96.08%
UNINA2009	99.97%	99.99%	86.23%

Tables 5.5 and 5.6 report the results obtained by training the system with LBNL, CAIDA and UNIBS traces, and testing it with the other two traces. Both tables indicate that the recognition rate of the non-HTTP protocol depends on the training set. The worst results in rejecting non-HTTP flows are obtained when LBNL traces are used for training and the system is tested on UNINA2009 (85.45%). This is due to the fact that class non-HTTP is not very well represented in the LBNL data. Figure 5.5 shows a scatter-plot of the UNINA2009 data for the four "allowed" sign patterns for port 80. The non-HTTP protocols are marked with green triangles. The misclassified protocols are circled. The figure demonstrates a degree of mismatch between the training (LBNL) and the testing (UNINA2009) data. It should be noted however, that the representation in the figure may be misleading because it does not reflect the density of the data. The plot for sign 0100, (a), for example, contain 47616 traces, of which 11 non-HTTP. There is only one mistake in the non-HTTP class (accepting a non-HTTP protocol), which amounts to 91% specificity. The highlighted mistakes are only a

fraction of the true HTTP class that were wrongly rejected (185 biflows in subplot (a), equivalent to 0.39%).



(a) 1000 (b) 1001

(c) 1010 (d) 1100

Figure 5.5: Scatterplot of the UNINA2009 using LBNL training data.

As with Table 5.4, there is a decline in the correct recognition rate of non-HTTP traffic from 2004 to 2009. Again, this may be explained with the hypothesis that some of the new non-HTTP traffic biflows are more similar to normal HTTP compared to the ones in the 2004 data. This notwithstanding, the obtained results remain very good since in the worst case over 85% of non-HTTP flows are rejected. In order to improve the performance the Decision Tree classifiers may be re-trained with new counterexamples. An advantage of the chosen architecture is that it allows us to do that for any of the classifiers without changing the rest of them.

In order to better assess the approach here presented and to further investigate the results obtained, we analysed traffic flowing through port TCP 80 that was labelled by our identification system as ‘rejected’. Firstly, we added a feature in

Table 5.5: Results obtained by testing the system with UNINA2004.

Training Dataset	Overall Accuracy	HTTP Accuracy	non-HTTP Accuracy
LBNL	99.69%	99.73%	88.96%
CAIDA	99.25%	99.25%	97.84%
BRESCIA	96.45%	96.44%	99.64%

Table 5.6: Results obtained by testing the system with UNINA2009.

Training Dataset	Overall Accuracy	HTTP Accuracy	non-HTTP Accuracy
LBNL	99.26%	99.28%	85.45%
CAIDA	98.82%	98.83%	92.73%
BRESCIA	99.21%	99.22%	94.55%

TIE to examine the first few bytes of payload carrying TCP payload exchanged in each biflow. This allowed us to perform a preliminary automated examination of all the biflows and to verify the results of the identification by easily checking application-level packet content. In addition, we manually inspected the biflows, mainly focusing on what was recognized as non-HTTP by the classifier. First of all, such analysis confirmed that all the correctly accepted biflows were actually related to HTTP traffic. For example we observed that almost 94% of the biflows in UNINA2004 started with a standard HTTP GET request, 4% with a POST request, etc. On the other side, we discovered that several 'rejected' biflows we generated by peer-to-peer application-level protocols as eDonkey, Bittorrent, and WinMX. Some of them started with a byte not corresponding to an alphabetic character. Inside this category, most of them started with the byte 0xe3. As reported by Karagiannis *et al.* in [43], this is the first byte exchanged by peers opening a communication session based on the eDonkey2000 protocol (used by the eDonkey and eMule file-sharing applications). Moreover, in both UNINA2004 and UNINA2009 traces, up to 50% of the non-HTTP biflows could not be ascribed to a specific application using either automated or manual payload inspection. However, we manually verified that these biflows did not exchange any HTTP traffic; we therefore conclude that such traffic is generated by applications

using undisclosed proprietary protocols.

The whole analysis described here confirms that the identification approach proposed in this work is very effective in correctly discriminating real HTTP traffic using the well-known port TCP 80. Finally, we observe that in the traces the non-HTTP traffic represents a not negligible portion of the captured traffic. Indeed, after filtering our traces by removing biflows related to non-HTTP traffic, about 5% of the packets were discarded (both in the UNINA2004 and UNINA2009 trace). Moreover, it must be observed that on the UNINA network there were no rules enforced to prevent traffic on non-standard ports. Therefore most of the connections masquerading as HTTP were probably due to the configuration of external peers located in networks where port-based traffic filtering was strictly enforced. It is reasonable to hypothesize that if this was also the case of the UNINA network, then such masquerading traffic would have covered an even higher percentage.

5.4.1 Results obtained with the training set cleaned by SOCIAL

Finally, we tried to apply SOCIAL, the algorithm presented in Chapter 4, to clean the training sets described so far.

Originals				**Cleaned by SOCIAL**			
Training Dataset	Overall Accuracy	HTTP Accuracy	non-HTTP Accuracy	Training Dataset	Overall Accuracy	HTTP Accuracy	non-HTTP Accuracy
LBNL	99.65%	98.32%	93.65%	**LBNL**	99.78%	99.78%	99.34%
CAIDA	99.17%	99.17%	99.56%	**CAIDA**	99.17%	99.17%	99.56%
BRESCIA	98.38%	98.38%	99.56%	**BRESCIA**	98.86%	98.86%	99.67%

Table 5.7: Results obtained by using UNINA2004 “cleaned” by SOCIAL.

To this aim we trained a decision tree classifier with the cleaned datasets and we tested it with UNINA2004 and UNINA2009.

In table 5.7 the results obtained to distinguish between http on not-http on UNINA2004 dataset are shown. The table on the right contains the results obtained on the dataset *cleaned* by SOCIAL, while the table on the left contains the results

obtained with the original dataset. The same thing is proposed in the table 5.8 for the UNINA2009 dataset.

	Originals			Cleaned by SOCIAL			
Training Dataset	Overall Accuracy	HTTP Accuracy	non-HTTP Accuracy	Training Dataset	Overall Accuracy	HTTP Accuracy	non-HTTP Accuracy
LBNL	99.17%	99.26%	51.45%	**LBNL**	99.50%	99.52%	87.68%
CAIDA	98.94% left	98.96%	51.45%	**CAIDA**	98.98%	99.00%	89.13%
BRESCIA	98.55%	98.57%	88.40%	**BRESCIA**	98.40%	98.41%	89.13%

Table 5.8: Results obtained by using UNINA2009 “cleaned” by SOCIAL.

The results shown that the system performs always better than the Decision Tree trained on the original dataset. Another important result is figured out comparing this results with the ones proposed into the table 5.7 and in the table 5.6, where the results obtained with the hierarchical MCS proposed are described, also in this case a simple decision tree is comparable in terms of accuracy with a more sophisticated architecture as the one proposed in this chapter.

It is worth noting that the three training set are quite old, that’s why they reach better values of accuracy on the UNINA2004, while testing them on UNINA2009 the results are not brilliant. Even making this considerations, using SOCIAL to clean the training sets, we reached a good level of accuracy even with the newer training set.

It is possible to see the temporal traffic variation in perfect analogy to the mutation inducted by a malicious user. That’s why the results shown in this tables demonstrate a adversarial classification robustness of the training sets cleaned by SOCIAL.

5.5 Key Findings

We examined the ability of a classifier ensemble system to identify traffic flows that do not belong to their declared class. The system takes the direction signs of the first four packets carrying payload and filters out the most improbable flows. The remaining flows have “acceptable” sign patterns. A decision tree classifier is trained for each sign pattern. Here we focused on TCP on port 80, trying to sepa-

rate true HTTP traffic from non-HTTP traffic flowing through the same port (e.g. in order to circumvent network policies). We imposed four acceptable sign patterns: 1000, 1001, 1010 and 1100. A Decision Tree classifier is designed for each of them. We found that the system is very accurate when trained and tested on data coming from the same distribution (tested through cross-validation on traces from the University of Napoli - UNINA2004 and UNINA2009). Furthermore, the system exhibits very high accuracy in cross-testing, i.e., trained on one network and tested on another. We verified this by training the system on three different data sets (LBNL, CAIDA and BRESCIA) and testing it with UNINA2004 and UNINA2009. We looked in more detail in the worst case (accuracy 85.45%) where the system was trained on LBNL and tested on UNINA2009. It seems that the LBNL data did not have a sufficiently representative non-HTTP class in order to train the system properly. The high overall accuracies in the cross-testing demonstrate what we call the "spatial invariance" of the system. In addition, the system shows "time invariance" in that the accuracy did not drop dramatically from 2004 to 2009 even though some decline was observed. Finally, due to its design and to the use of Decision Trees, the system is very fast, and it can be used online.

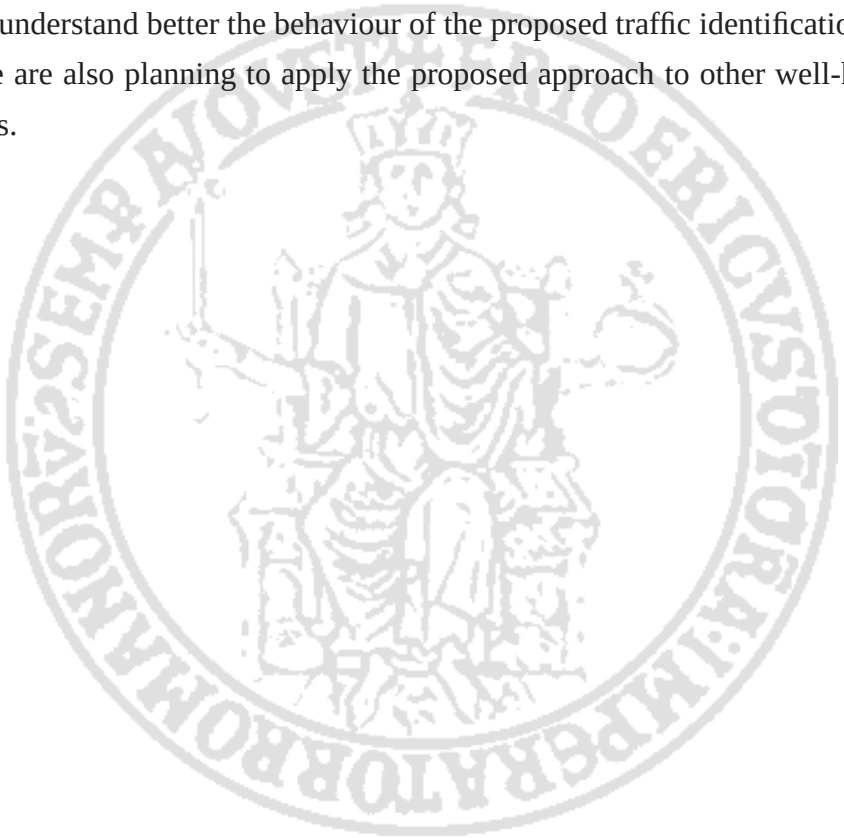
This approach is based on a multi-stage architecture made up of an ensemble of Decision Trees, each one devoted to verify if the flow under test belongs or not to the protocol whose port number refers to. Each Decision Tree is activated by a specific combination of the signs of the first four packets of the flow and performs the verification process by considering the payload sizes of these four packets. We showed results in the case of traffic flows hidden behind "port 80"-based flows. Using real traffic traces from four different networks we showed the high accuracy of the proposed approach, that also demonstrates:

- **spatial invariance**, since it was able to reject non-HTTP traffic captured in a network different from the ones considered during the training phase;
- **temporal invariance**, since it worked well with traffic traces captured in very different temporal periods (over a range of five years);
- **on-line capability**, since only the first four packets are needed for carried

out the verification process;

- **adversarial classification robustness**, since it was robust to different typology of traffic not seen in the training phase.

Another advantage of a classifier based on Decision Trees is that it can be seen as a set of simple decision rules that can be easily interpreted by a domain expert. A further research direction will be a deeper analysis of such decision rules in order to understand better the behaviour of the proposed traffic identification system. We are also planning to apply the proposed approach to other well-known protocols.



Chapter 6

An Anti-Spam System based on a Behaviour-Knowledge Space

It is a well-known story that e-mail has grown from a tool used by few universities and scientists to a ubiquitous communication tool, evolving from simple plain text into a powerful multimedia message. At the same time, following the growth of e-mail production and diffusion, spam has changed from a little and sometimes bothering problem into a multi-billion dollar problem. The presence of spam, in fact, can seriously compromise normal user activities, forcing to navigate through mailboxes to find the - relatively few - interesting e-mails, so wasting time and bandwidth and occupying huge storage space.

The types of those messages vary: some of them contains advertisements, other e-mails provides winning notifications, and sometimes we get messages with executable files, which finally emerge as malicious codes, such as viruses and Trojan horses. In addition, spam e-mails may often have unsuitable content (as a pornographic material advertising) that is illegal and sometimes dangerous for non adult users.

The recognition of spam content is not a trivial problem: there are some factors that are related with human perception, economic behaviour, legal context, that are hardly measurable or summarized in adequate features. The same definition of *spam e-mails* requires a common agreement that is not easy to find.

In our opinion, *all* kind of spam e-mails have several common characteristics, such as: *i*) they are unsolicited, *ii*) they have a commercial content, even though

the content itself is continuously evolving, trying to outsmart the classical countermeasures adopted by anti-spam filters.

This kind of task belong to the **adversarial classification problems**, since there is an intelligent, adaptive adversary who tries to camouflage patterns (spam e-mails) to evade the security system.

Consequently, a great variety of technical methodology have been implemented in current anti-spam systems [11]. The common technical solutions propose filtering strategies based on sender address and/or body content. We focused our attention on that measures related to e-mail contents, in particular both *texts and images*, rather then on networking and identity strategies [68], since our goal is to develop a personal antispam system.

In this chapter we combine the visual clues with the semantic information related to the e-mail body, to determine whether a message is spam. In order to address the problem of combining a non-constant number of modules, since it is not possible to *a priori* known if there is one or more images attached to the e-mail and/or there are textual information to be processed, we propose the use of a *Behaviour Knowledge Space* [39] approach. This also allows us to easily include new modules in our architecture that could be required for addressing new spammers' tricks.

Organization of the Chapter

The chapter is organized as follows: the Section 6.1 describes at a glance the main component of the proposed system; in Sections 6.2 and 6.3 we describe text and image features respectively, while in Section 6.4 we show how to combine them. In Section 6.5 several experiments are discussed, and finally in Section 6.6 we report some considerations.

6.1 System Architecture

As shown in figure 6.1, we design a system that integrates image-based and text-based analysis, the dashed line, in the figure, represents the OCR output that is

filled into the Text Analyser. The mails, initially, are parsed by a Multi-purpose Internet Mail Extensions (MIME) parser, that can retrieve the different parts of the e-mails: the text parts, the attached images or text files, the email subject and the headers. The text is thus processed by a *Text Analyzer* module according to the methodology described in the section 6.2 and its output is a classification result obtained using the feature vector of text part of input email. The images are forwarded to the *Image Analyser* module which gives another classification results for the features vector that is extracted with the techniques described in section 6.3 for the image part of the email. We note that the OCR output of the *Image Analyser* could be used also by *Text Analyser* in order to build its feature vector. The *Fusion block* has the role to combine the previous output furnishing the final classification of each e-mail using the strategy discussed starting from the section 6.4.

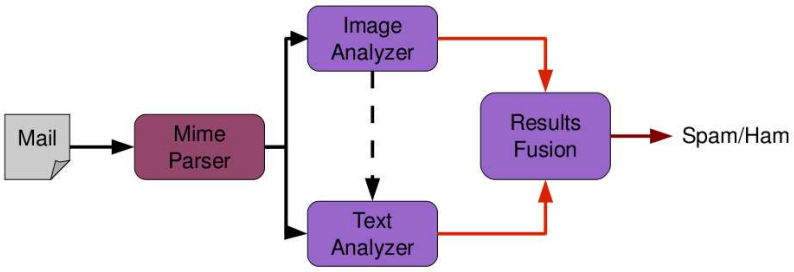


Figure 6.1: The proposed system architecture.

Both the Text and the Image analyser can be implemented by means of different classifiers, each one using different features. In the following, we will describe in details the different feature sets used and the combination process.

6.2 Textual Features

Textual filtering methods are widely deployed; they varies in the inspected content and the proposed methodology. Some filters consider only the header or the body of an e-mail, while other ones take both. These approaches use different mod-

els, considering word-tokens, their frequencies and their combinations. In *rule based-filters* [13] the users define some rules related to the headers or the bodies, considering particular words as *sign* of spam content; anyway, this simple solution is strongly dependent on how the words used by spammers can change.

Differently, *Signature-based* methods do not really deal with whole messages or specific tokens, transforming the message into a *signature*. Clearly, the methods effectiveness is related to the robustness of the signature function. Note that a signature database must be distributed and kept up to date very frequently, due to the rapid variation of spam e-mails. To this regard, some proposals are based on *collaborative solutions*, in particular on Peer-to-Peer (P2P) networks for signature distribution [83, 19]. These approaches are not well suited for developing a personal antispam system.

Other approaches consider spam detection as a *binary classification problem* and several algorithms from the learning theory research field have been used. In these solutions, e-mails are mapped into multidimensional space, each dimension representing the words in the e-mail content; several measures are proposed such as the terms-frequency (tf) or the product between the documents-frequency (df) and terms-frequency, as in [23]. *Statistical filters* based on the the Bayes theory have been also investigated [2, 58].

One of the drawback of these last methods is that they typically do not consider specific countermeasures for taking into account new spammer tricks, so a complete retraining of the system is needed when these attacks arise.

We propose a strategy based on text processing and analysis in order to process both *semantic* and *syntactical* features. Generally speaking, our main idea is to characterize how e-mails belonging to the same class (*ham* or *spam*) do have the same meanings, using a set of semantic features in addition with the detection of special characters (syntactical features) that are typically used into spam context.

In particular, at the semantic level we analyze the whole email content taking into account the word localization in a given context thus measuring the weight of a single word in the document. In this way, we relate the emails content to certain topic by looking at commonly shared words. A topic is described by a region of

Spam Topics
Investment/Business
Health/Medicine
Games,
Software
Leisure/Travel
Adult
Finance
Product/Service.

Table 6.1: The list of contents in spam mails

multidimensional space shared from the vectors of words of different e-mails. In the spam context, example of e-mail topics are reported in table 6.2. In section 6.2.1, we will describe the model used to discover the semantic content of e-mails.

The use of syntactical features is suitable to detect grammar anomalies in the texts. Typically, the ham e-mails do not have particular occurrences of special characters: these one can be thus used as signs of low trustworthiness of the received e-mail;the related developed methodology will be described in section 6.2.2.

6.2.1 Semantic Features

We propose to use a feature set based on a modified version of Vector Space Model (VSM) [54]. This model is based on the representation of documents as vector in multidimensional space. The representation of e-mail textual content in the vector space model has a number of advantages, including the uniform treatment of queries and documents as vectors and the ability to differently weight the different terms; anyway, it suffers from its inability to cope with two classic problems arising in natural languages [41], i.e. synonymy and polysemy. We briefly recall that *synonymy* refers to a case where two different words (say ”pupil“ and ”scholar“) have the same meaning, and *polysemy* refers to the case where a term such as ”play” has multiple meanings according to different contexts. In fact as worst case of the influence of synonymy in similarity measure, we could have two orthogonal vectors with 0 as result of cosine similarity even if there are

two different words that have the same meaning inside those two vectors. The semantic correlation or disambiguation of the these terms can be made looking at the context in which they are placed, for example the terms “scholar” can be correlated to “pupil” if the documents, in which they are, also contains terms like “school”, “book”, “pen” and so on. In that way the shared terms can increase the value of similarity measures. The idea of looking at the whole email document can be seen also as an overcoming of the independence hypothesis used in a bayesian filter technique known as bag-of-words model that is one most used approach for anti-spam filer. In that model the relationships among set of words (joint distribution) are simply factorized. In order to overcome the fault of vector space model to capture the *synonymy* and *polysemy* relationships, we choose a modified version of VSM, the Latent Semantic Analysis or LSA [54]. Despite LSA is a traditional and well accepted technique used to stick out the semantic contents in text-process community, there are few application in the spam framework. LSA is an application of Singular Value Decomposition (SVD) to document-by-term $N \times M$ matrices A. In particular, SVD provides a suitable matrix decomposition as described in the following:

$$A = TSD^T$$

being S=diag$(\sigma_1, \ldots, \sigma_r)$ a $M \times N$ matrix, with $\sigma_i = \sqrt{\lambda_i}$ and $\lambda_i \geq \lambda_{i+1}$ with $1 \leq i \leq r$; the $\lambda_1, ..., \lambda_r$ be the eigenvalues of AA^T, r being the rank of A. Note that A^TA has the same eigenvalues of AA^T.

The values σ_i are also denoted as the *singular values* of A. In the LSA technique, it is used a reduced version of A, $A_k = T_k S_k D_k^T$ that is $M \times N$ matrix and k being a positive integer that is the maximum rank of A_k. After that decomposition we can have a representation of documents and terms in the singular value space, in fact we have a term matrix $L_t = T_k S_k$, called matrix of singular loadings for terms and $L_d = S_k D_k^T$ that is called the matrix of singular loadings for documents. We note that this operation applied on the SVD decomposition has two main properties described as following:

- We have a dimensional reduction of the initial problems, in fact we can represent the documents as features vectors of dimension k using the matrix of singular loadings for documents L_d.
- We obtain a reduction that is representative of the nature of the documents. In fact the S_k matrix have in the diagonal the decreasing order of the singular values, this can use to correlate the document vector that shared common terms using only a subset of their values.

The obtained approximation is computed taking into account the distance between the two matrices X = A-A_k that is minimal according to a Frobenius norm [54]. In other words, we have a reduced space in which the words that have similar co-occurrence patterns are projected (or collapsed) into the same dimension, and in the indexing phase the technique projects the documents into the new generated space with latent semantic dimensions. The choice of k has been derived empirically, with 80 to 100 dimensions being sometimes the optimal choice for collections of about $5,000$ terms by $1,000$ documents [20]. In order to derive the features to learn a classifier during the training phase, we adopt as text features the projection of the document in the space obtained by $L_d = S_k \times D_k^T$, S_k and D_k being the matrices after the SVD reduction. In the testing phase we use also this matrix product $T_k^T Q$ in order to compute the text features thanks to the SVD equation:

$$T_k^T Q = S_k D_k^T \tag{6.1}$$

$(*)_k$ being the matrices obtained after the reduction process and Q being the $N \times 1$ matrix representing the input document.

There are different steps used to process the email text until the generation of text feature both int the training phase and test phase. The different phases are depicted in figure 6.2 and they are described as following:

- The *Preprocessing* module used a set of intelligent filters that we apply to the email documents with or without the OCR recognized set of words. These filters are:

  - The classical stop word list filter, that is used to delete the words that have no particular meaning, although increasing the term vector dimension and thus degrading both performance and results of the system. Typical example of stop word list are adverbs are pronouns.
  - We also propose an intelligent filter that is able to detect and reject the words that are not *human-understandable*, e.g. sequences like "fsdrx", "jkdld". This solution is based on an SVM classifier trained on several features derived from bigrams and trigrams composition of English words. We also build a feature vector containing the ratio between the correct bigrams (trigrams) and all the bigrams (trigrams) for a set of 170000 common English words. Note that the use of this kind of filter has also the aim of enhancing the recognition of the semantic content that can be used in particular spammer attacks, such as the ones which use to put random words into e-mail texts, thus trying to reduce the effectiveness of current antispam algorithms. This filter can also be used to reject the words that are bad recognized by OCR algorithms.
  - A Part of Speech filter (POS) module that is able to detects nouns, verbs and adjective; it is used to reject adjectives that typically do not give further additional information.

- The *Stemming* module implementing the well known Porter Stemmer algorithms [64] that is used to remove the common morphological and inflexional endings from words in English. Here in after, the stemming and preprocessing module will be called Text Processing (TP).

- The *LSI* module that implements the functionality of the model above described; it takes as input the *LSA* model produced during the training phase. We compute the "terms by documents matrix" used in those techniques using the following measures:

  - Term-Frequency (TF)

$$TF_{ij} = \frac{n_{ij}}{\sum_r n_{ij}}$$

n_{ij} being the number of occurrences of the term in document d_j, and the denominator being the sum of number of occurrences of all terms in document d_j.

– Inverse Document Frequency*Term-Frequency (IDF*TF)

$$IDTF_{ij} = TF_{ij} * DF_i = TF_{ij} * log(\frac{N_D}{N_D^{T_i}})$$

N_D being the number of total documents in the corpus and $N_D^{T_i}$ the number of documents in the corpus in which compare the term T_i.

– Entropy Weight (WE)[50]:

$$WE_{ij} = TF_{ij} * (1 + \sum_j \frac{p_{ij} * log_2(p_{ij})}{log_2(N_D)})$$

$p_{ij} = \frac{TF_{ij}}{TF_i^D}$ being the probability to get document j given the term i and TF_i^D being the term frequency of term i on the whole document collection.

6.2.2 Syntactical Features

We propose to use some syntactic features that can be extracted from mail texts, in order to estimate usual and suspected mail formats.

Spammers, in fact, usually try to obfuscate the textual part of an e-mail's body by substituting some characters in order to bypass the effectiveness of antispam filters.

So, we defined another set of features for obtaining a characterization of this kind of obfuscated text. The features we have investigated on are mainly based on the presence of *special* characters, i.e. those characters that should not frequently occur in a legitimate text. The whole set we considered is made up of the following characters: { **!**, **”**, **#**, **$**, **%**, **&**, **’**, **(**, **)**, ***, **+**, **,**, **-**, **.**, **/**, **@**}. Starting from this set we defined six *syntactical features*:

- **text_length:** the number of characters of the whole text

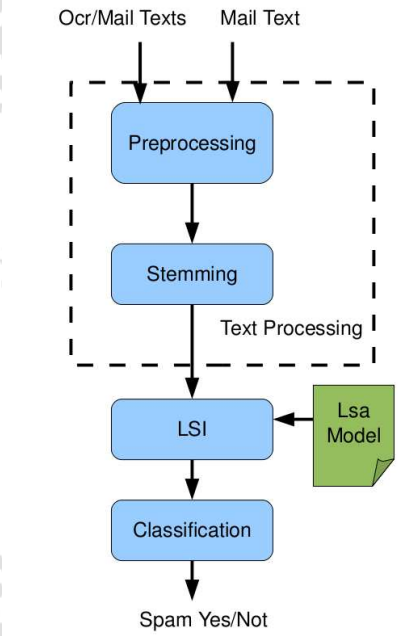


Figure 6.2: The different phases of the Text Analyzer

- **words_number:** the number of words in the text
- **ambiguity:** the ratio between the number of special and normal characters
- **correctness:** the ratio between the number of words that do not contain special characters and the number of words that contain special characters
- **special_length:** the maximum length of a continuous sequence of special characters
- **special_distance:** the maximum distance between two special characters belonging to the above considered set.

6.3 Image Features

Image spam has been extensively studied using several techniques primarily developed from the Image Processing and Computer Vision community, using features related to color distribution [3] or textual areas [3, 80]. A classifier is usually trained on such features, trying to discriminate spam images from legitimate ones. In [22], the authors present features that are focused on simple properties of the image, making classification very fast. In this chapter, however, the authors completely disregard the textual part of the emails.

Other approaches basically try the detect textual areas in images following the idea that images which contain texts are likely to be spam. In [74] the authors propose an algorithm for text localization. They construct a corner detection algorithm based on a circular template to predict the corner points of the text in an image, which is crucial for text localization. The same idea is presented in [12]. The method proposed there extracts edge features of a binarized image by using higher-order local autocorrelation, and then passes these features to a Support Vector Machine (SVM) for classification. In [38] the authors try instead to extract connected components from the image in order to detect the presence of an embedded text.

A quite different approach is followed in Fumera et al. [31], where the authors propose to process each image by using an OCR system for extracting embedded

texts.

All these approaches, however, cannot be effectively used when text within images is voluntarily distorted and/or obfuscated. As it was noted in [9], in fact, now spammers try to make OCR and text detection techniques ineffective without compromising human readability, by placing text on non-uniform background, or by using techniques like the ones exploited in CAPTCHAs[1] (programs that generate and grade tests that humans can pass but current computer programs cannot).

We propose an approach for the detection of the image spam in which two different image processing techniques are used [33]. The first one is devoted to directly extract some global features from each image attached to the e-mails. Such features should also be able to detect if images were adulterated or not, by considering the complexity of the image itself as it is perceived from an human being. The second processing is carried out by means of two steps: first, there is a preprocessing phase with the use of an OCR, then a feature extraction process starting from the OCR output try to characterize it in order to detect if the embedded text has been voluntarily obfuscated and/or distorted.

6.3.1 Visual Features

The first set of features, that we called *visual features*, are directly obtained from the image attached to the mails. In order to give an image characterization that should be able to discriminate between normal and adulterated images, we considered features that describe the image texture from a statistic point of view. As said before, in fact, spammers typically now try to bypass filters that use an OCR for detecting texts within an image by obfuscating such texts with the addition of some noise or by superimposing a texture (see also Figure 6.3 in which it is used *gocr*[2] as Optical Character Recognition tool). So, texture detection can help in in-

[1]The term CAPTCHA (Completely Automated Turing Test To Tell Computers and Humans Apart) was coined in 2000 by Luis von Ahn, Manuel Blum, Nicholas Hopper and John Langford of Carnegie Mellon University. At that time, they developed the first CAPTCHA to be used by Yahoo – http://www.captcha.net/

[2]*gocr* is available at http://jocr.sourceforge.net

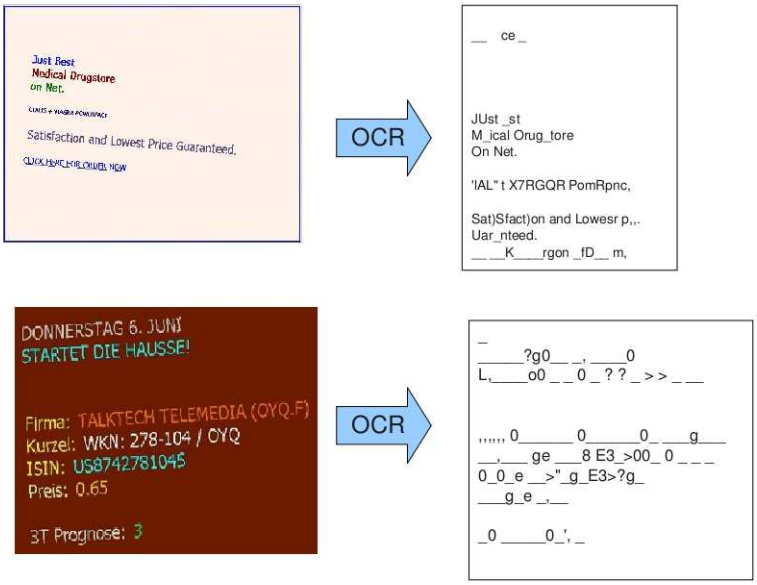


Figure 6.3: Outputs obtained by applying *gocr* to some spam images

dividuating images that contain spam messages. For the sake of simplicity, in the following we will present the considered features in case of gray-level images, but the same operators can be applied to color images too.

We will use $\{I(x,y), 0 \leq x \leq N-1, 0 \leq y \leq M-1\}$ to denote a $N \times M$ image with G gray levels. All the considered statistical texture measures are based on the co-occurrence matrices. Spatial gray level co-occurrence estimates image properties related to second-order statistics. The $G \times G$ gray level co-occurrence matrix $P_{\mathbf{d}}$ for a displacement vector $d = (dx, dy)$ is defined as follows. The entry (i, j) of $P_{\mathbf{d}}$ is the number of occurrences of the pair of gray levels i and j which are a distance **d** apart. Formally, it is given as:

$$P_{\mathbf{d}}(i,j) = |\{((r,s),(t,v)) : I(r,s) = i, I(t,v) = j\}|$$

where $(r,s),(t,v) \in N \times M$, $(t,v) = (r+dx, s+dy)$, and $|.|$ is the cardinality of a set.

As regards the choice of the displacement vector **d**, we considered the four direct neighbors of each pixel, i.e. we used four pairs as values of dx and dy for calculating the number of co-occurrences, namely $(0, 1)$, $(1, 0)$, $(-1, 0)$ and $(0, -1)$. We do not perform a normalization of $P_{\mathbf{d}}$ in order to preserve the dependence of the considered features on the image size.

As suggested in [36], from the co-occurrence matrix it is possible to extract features that can be used for detecting a texture within an image. In particular, we considered the following five features:

- **Contrast**

$$\sum_i \sum_j (i-j)^2 P_{\mathbf{d}}(i,j)$$

  is the difference in terms of visual properties that makes an object (or its representation within an image) distinguishable from other objects and the background. In the visual perception of real world, contrast is determined by the difference in the color and brightness of the object and other objects within the same field of view. In practice, it is the ratio between the brightest and the darkest value of the image. In the case of a B/W image, note that the increase of the contrast is equal to erase gray values.

- **Entropy:**

$$-\sum_i \sum_j P_{\mathbf{d}}(i,j) log P_{\mathbf{d}}(i,j)$$

  is an index of the brightness variation among the pixel in an image. More the values of brightness are different each others, more the entropy will be higher.

- **Energy:**

$$\sum_i \sum_j P_{\mathbf{d}}^2(i,j)$$

  is the spectral content of an image

- **Correlation:**
$$\frac{\sum_i \sum_j (i-\mu_x)(j-\mu_y) P_{\mathbf{d}}(i,j)}{\sigma_x \sigma_y}$$
is an index of the correlation degree among the pixel. Here μ_x and μ_y are the means and σ_x and σ_y are the standard deviations of $P_{\mathbf{d}}(x)$ and $P_{\mathbf{d}}(y)$ respectively, where $P_{\mathbf{d}}(x) = \sum_j P_{\mathbf{d}}(x,j)$ and $P_{\mathbf{d}}(y) = \sum_i P_{\mathbf{d}}(i,y)$

- **Homogeneity:**
$$\sum_i \sum_j \frac{P_{\mathbf{d}}(i,j)}{1+|i-j|}$$
is a measure of the brightness variation within the image. If the image is completely black or white, its homogeneity value will be the maximum. On the contrary, if the image contains several brightness variations, this value will be very low.

Another category of features that can be used for characterizing images from a global point of view is based on the complexity of an image for a human reader. We have chosen to consider a feature also proposed in [9]:

- **Perimetric Complexity:** is defined as the squared length of the boundary between black and white pixels (the perimeter) in the whole image, divided by the black area.

Note that, differently from [9], we evaluate the perimetric complexity on the whole image, after performing a binarization with a fixed threshold.

6.3.2 OCR-based Features

Here we propose to use the same features considered in Section 6.2.2. In this case, however, special characters are extracted from the output of an OCR that has received an attached image as input.

We have noticed, in fact, that characters embedded into an image are opportunely distorted and/or obfuscated in spam e-mails. Thus, most of the words cannot be correctly detected, as we can see in Figure 6.3. Furthermore, several special

characters that typically are not present in commonly used words can appear in the OCR output.

6.4 Combining Text-based and Image-based Classifiers

It has been experimentally shown that the combination of an ensemble of classifiers can be of great benefit in many practical pattern recognition applications. Through the appropriate choice of a combination rule, it is possible to dampen the overall effect of the *independent* errors in each observation domain, thus reaching performance better than those of a single classifier.

The combination of classifiers is then an important part of our architecture. Anyway, there are some problems that must be taken into account in this case:

- It is necessary to define a method for combining a non-constant number of classifiers, since it is not possible to *a priori* known if there is one or more images attached to the e-mail and/or there are textual information to be processed.

- It should be avoided *padding-attacks* from spammers. That is, the possibility that an attacker puts a spam message within a *normal* context, for example by attaching an image containing an embedded spam message to an e-mail that contains *normal* images.

As shown in Figure 6.4 we used a two-stage approach for combining text-based and image-based classifiers. In the figure, "TP" stands for *Text Processing*; it is described in section 6.2.1.

The first stage (denoted as *Classification* in Figure 6.4) consists in a simple 3-state *logical OR*, whose behavior is described in Figure 6.4. In this way we also consider the case in which a classifier cannot be activated. It happens, for example, when there are no images within the e-mail, or when there are no words to be processed by the semantic analysis. In this situations, we assume that the output of the classifier is *undefined*. Note that through this approach we try to

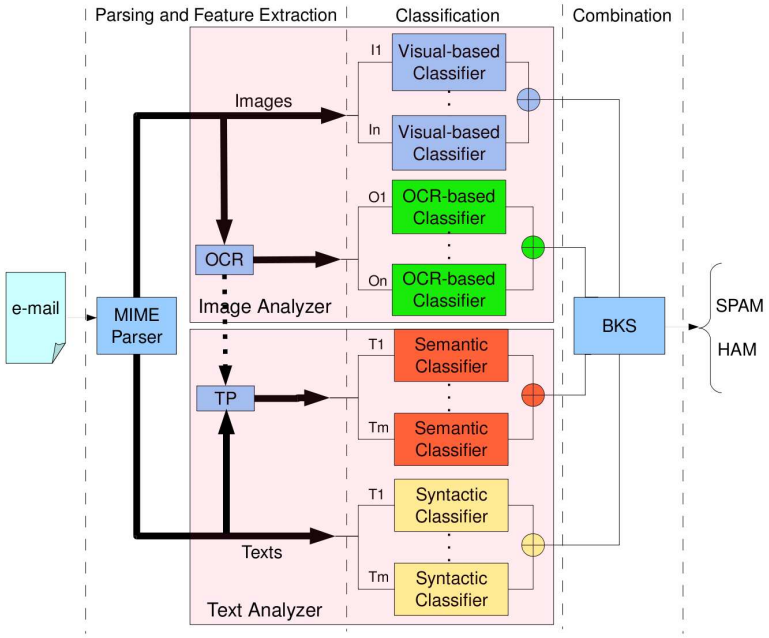


Figure 6.4: The proposed combination approach

address the problem of *padding attacks*, too. Just one correctly classified spam image, in fact, is sufficient so that the block of the visual classifiers declares the email as spam.

Then, at the second stage we adopt a *Behaviour Knowledge Space* (BKS) combining rule [39]. The idea behind this rule is to avoid making unjustified assumption on the classifier ensemble such as classifier independence. In Figure 6.4 an example of how it works is shown.

A BKS is a K-dimensional space where each dimension corresponds to the decision of a classifier. Given an e-mail to be assigned to one of 2 possible classes, the ensemble of K classifiers can in theory provide 2^K different decisions.

We must also consider the case in which the output of the 3-state logical OR is *undefined*. In other words, each set of classifiers can attribute a mail to one out of three possible classes, i.e. $\{Spam, Ham, Undefined\}$ and the number of different decisions becomes 3^K.

Each one of these decisions constitutes one unit of the BKS. In the learning

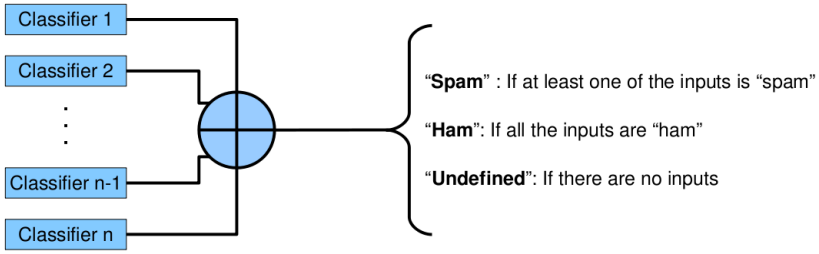


Figure 6.5: The 3-state logical OR

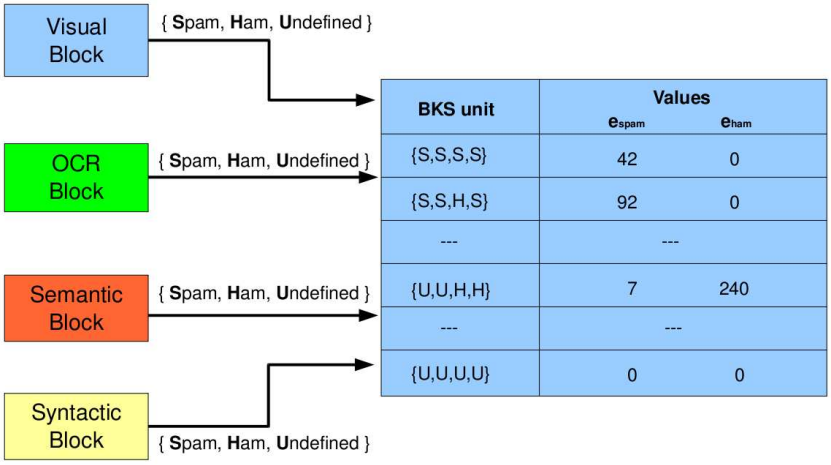


Figure 6.6: The Behaviour Knowledge Space for combining classifiers.

phase each BKS unit can record 2 different values e_i (say, e_{ham} and e_{spam}), by considering that the actual classes are only *ham* and *spam*. Given a suitably chosen training set, each sample x of this set is classified by all the classifiers and the unit that corresponds to the particular classifiers' decision is activated. It records the actual class of x, say C_j, by adding one to the value of e_j. At the end of this phase, each unit can calculate the best representative class associated to it, defined as the class that exhibits the highest value of e_i. This class corresponds to the most likely class, given a classifiers' decision that activates that unit.

In the operating mode, for each e-mail to be classified, the K decisions of the classifiers are collected and the corresponding unit is selected. Then the e-mail is assigned to the best representative class associated to that unit. Since we consider all the possible combinations of classifiers outputs as the number of available classifiers varies, we are implicitly handling the fact that the number of available classifiers can be different for each e-mail.

It is worth noting that the proposed combining scheme could be also easily extended using different feature sets, and then other classifiers. This could be required, for example, for addressing new spammers' tricks. In this case the problem is that the number of BKS unit grows exponentially and so a wider training set is needed in order to achieve good classification results. However, as it will be shown in the following Section, only a subset of all the possible units are typically activated in practice, since some configuration of the classifiers' decisions are not allowed.

6.5 Experimental Results

In the following we will first present the database used for assessing the effectiveness of the proposed approach, then evaluate if the use of both visual and textual features can improve the performance of the system with respect to the use of a single set of features. Finally, we make a comparison of our approach with a state-of-the-art anti-spam filter, i.e. *SpamAssassin* equipped with two different spam image plug-ins.

Total # of e-mails		*e-mails with Images*	
Spam	*Ham*	*Spam*	*Ham*
9173	2479	1802	151

Table 6.2: The dataset used in our tests.

As regards the dataset, whose details are given in Table 6.2, it is composed by 11652 e-mails, 9173 of which contains spam messages. e-mails were collected from the mailboxes of some users of the `studenti.unina.it` mailserver in a period of about three years (2005-2007). This mailserver hosts the mailboxes of all the students of the University of Naples Federico II. Among those e-mails, 151 contain *ham* images and 1802 contain *spam* images.

As regards the first stage of our architecture (the *Classification* one), we chose a *Decision Tree* for implementing each classifier. In particular, a C4.5 (J48) coming from the open source tool *Weka*[3] was selected.

Each single classifier was trained on a set of 1,000 mails (500 for each class) different from those belonging to the dataset reported in table 6.2. In order to train the BKS rule, the dataset was split into two sets. Then, two experiments have been made, by using a set for training and the other one for testing. Results are finally obtained as the average value of the accuracy reported in these two tests.

In Figure 6.5 the performance of the single classifiers and of the proposed systems are reported. Note that the last two single classifiers - third and fourth rows - processed only e-mails with attached images.

It can be noted that the use of the BKS significantly improves the performance of the single classifiers. It must be remarked, in fact, that the visual-based classifier operates on a subset of the whole dataset (only 1953 mails out of 11652). It is also interesting to note that the number of BKS units that are really activated on the whole dataset is only 18, while their total number is 3^4, i.e. 81. This confirms the considerations made in the previous Section.

Finally, in the Figure 6.5 we report a comparison of the results obtained by our system with those obtainable with *SpamAssassin* in its standard configuration and equipped with two plug-ins devised for filtering image spam, namely

[3]http://www.cs.waikato.ac.nz/ml/weka/

Figure 6.7: Some examples of *ham* images

Figure 6.8: Some examples of *spam* images

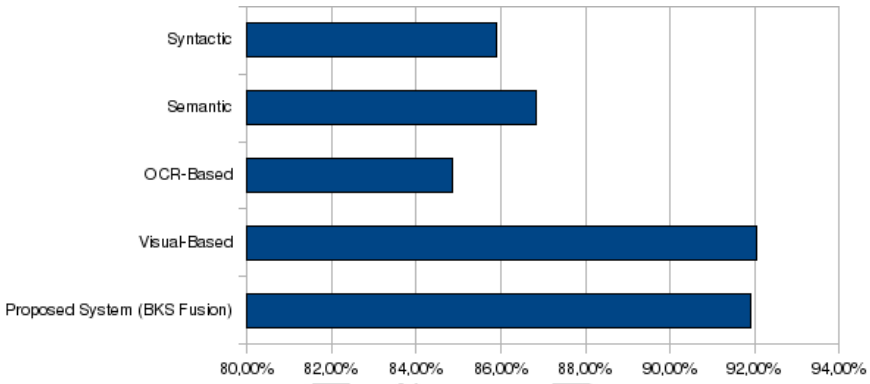


Figure 6.9: The accuracy of the single classifiers and of the proposed system.

Bayes-OCR[4] and *Fuzzy-OCR*. It clearly appears that our approach significantly outperforms both *Bayes-OCR* and *Fuzzy-OCR*, by reaching a significantly higher accuracy. Finally, note the time needed for processing the whole dataset by our system are practically the same needed by SpamAssassin with *Fuzzy-OCR*, while is significantly faster than *SpamAssassin* equipped with *Bayes-OCR*.

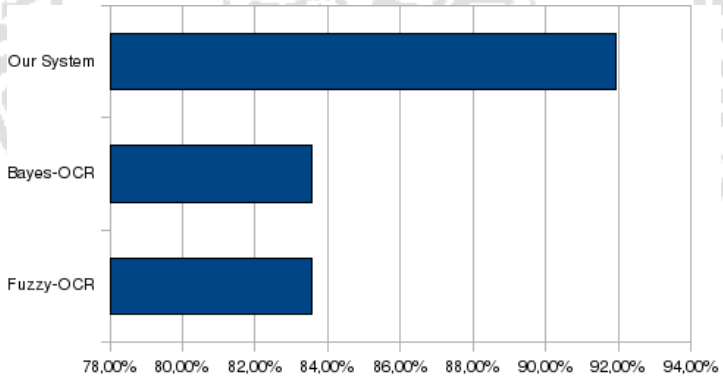


Figure 6.10: Comparison between the proposed system and *SpamAssassin*

[4]This plug-in is available for download at the URL: http://prag.diee.unica.it/n3ws1t0/?q=node/107

6.6 Key findings

In this chapter we presented an approach for addressing the spam e-mail problems, which takes into account some of the recent evolutions of the spammers' tricks as well as the limits of previous methodologies. We proposed to combine visual clues with the semantic information related to the e-mail body by using the Behaviour Knowledge Space rule. This approach allowed us to easily include new modules in our architecture that could be required for addressing new spammers' tricks.

This system can prevent the evasion problems, *adversarial classification*, monitoring the spam and the folders, and adding or updating some modules to the architecture if new kinds of spam are bypassing the antispam security system.

Tests on a dataset of e-mails containing attached images confirmed the effectiveness of the approach and its applicability with respect to other widely used opensource tool such as *SpamAssassin*.

Since the proposed approach has been mainly designed for deploying a personal antispam system, in the future we want to investigate how it is possible to further improve its performance by customizing it with reference to a specific user. This could be done by developing a specific module for taking into account spam images received by the user *A* that are considered as ham by the user *B*, such as, for example, those related to a phishing attack versus the user *A*.

Chapter 7

Conclusions

We demonstrated that Multiple Classifier Systems are a good choice for implementing a pattern recognition system that have to operate in an Adversarial Environment. They provide a good means for tackling the two challenges afforded during this thesis: *Adversarial Learning* and *Adversarial Classification*.

In this chapter we will provide a brief summary of the work behind this thesis, and will draw some general conclusions, by illustrating the key findings.

7.1 Our Contribution

The contribution of this thesis is to find out how an MCS approach could be the better choice in the *Adversarial Environment* problem. We have designed several MCS approaches for different tasks such as the cleaning of a training set (noisy or contaminate), the definition of an antispam system and the identification of Internet traffic flows.

In particular, we have defined a methodology to clean a training set through a MCS approach, named SOCIAL. This system changes the labels associated to the training set samples in accord with a dynamical adaptation of the degree of belief associated to each base classifier. We presented some experimental results in which the goodness of the approach is confirmed. In particular, we made a comparison between a *simple* classifier trained with the *cleaned* dataset obtained with the proposed approach, and the accuracy obtained with some Multiple Clas-

sifier System presented in the literature. The results showed that a simple classifier trained with the training set cleaned by SOCIAL, performs better than some "state-of-the-art" MCS approaches.

Further examples of adversarial learning arise in the field of computer security where there is an escalating competition between detection and evasion techniques for various types of malware. In general, one can expect that whenever machine learning is used to provide protection from some illegal activity, adversaries will deliberately attempt to circumvent these approaches.

Due to the above considerations we have considered some interesting case studies, and we find out what was the impact of an MCS approach for this field. In particular, we have proposed a modular anti-spam approach to deal with the spam transmitted through *text* and *images*. The modular architecture was designed to be updated with new modules in order to efficiently cope with new types of spam.

We proposed also an hierarchical MCS approach for the Internet traffic identification problem, that try to distinguish among the different flows. In this case the real-time feature was really important, and the system was designed to be fast and accurate.

7.2 Key Findings

The core of this thesis consists in the study of the Multiple Classifier Systems to address the problem of *Adversarial Environments*. We demonstrate that this kind of systems are more robust to the noise and/or contamination of the training set label.

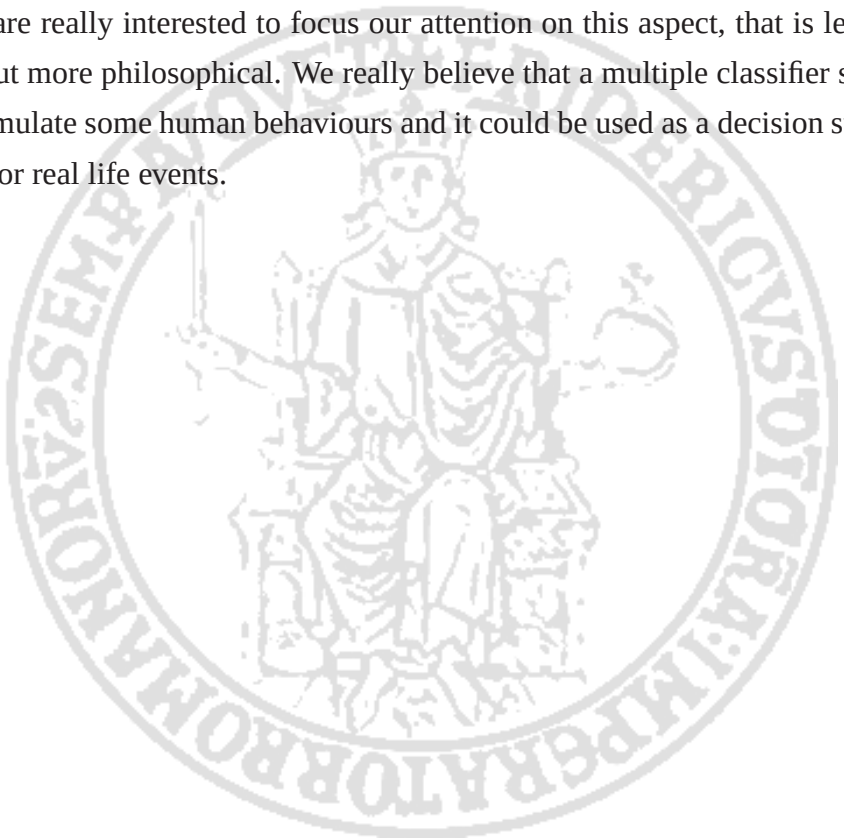
We noticed that there are two possibility for an MCS, to force the diversity among the classifiers in order to improve the accuracy or to force the convergence of the decision. In this last case we are trying to obtain a sub-optimal but more robust classification.

We pointed out that an MCS has a significant similarity with the non-linear dynamic system, and with other physics phenomena.

Moreover, there are important similarities with the human behaviour too. In

fact, if we want a stable society where the optimum is never reached in favour to a stability issue, we are trying to a make a SOCIAL approach. If we want instead to force the finding of an optimum it is more useful the competition instead of the stability. This choice can reach an optimum forcing the diversity of the single component (base-classifier).

Everything the human being has done that works properly, takes inspiration from the nature. Even in this case there is a deep link with the natural evolution, and we are really interested to focus our attention on this aspect, that is less scientific but more philosophical. We really believe that a multiple classifier system could simulate some human behaviours and it could be used as a decision support system for real life events.



Appendix A

Dempster-Shafer Combination Rule

In this section we will apply the general methodology described in the chapter 4 using the Dempster-Shafer theory. This theory will be used to develop the *Fusion Block* and for the base classifiers statistical characterization. That is for the **R** definition we choose to use the *Basic Probability Assignment*, hereinafter bpa.
The theory of Dempster and Shafer (D-S theory) has been frequently applied to deal with uncertainty management and incomplete reasoning. Differently from the classical Bayesian theory, D-S theory can explicitly model the absence of information, while in case of absence of information a Bayesian approach attributes the same probability to all the possible events.
According to the D-S theory, we used as **R** the bpa. It describes the subjective degree of confidence attributed to it. What is modelled, then, is not the analysed phenomenon, but the belief in the base classifiers report about it.
When assigning a bpa, there are some requirements which have to be met. They descend from the fact that the bpa is still a probability function, hence has to respect the constraints for mass probability functions. Each bpa is such that $m : 2^{\theta} \rightarrow [0, 1]$, where θ indicates the so called *frame of discernment*. Usually, the frame of discernment θ consists of N mutually exclusive and exhaustive hypotheses A_i, $i = 1, \ldots, N$. A subset $\{A_i, \ldots, A_j\} \subseteq \theta$ represents a new hypothesis. As the number of possible subsets of θ is 2^{θ}, the generic hypothesis is an element of 2^{θ}.

For example, if we only consider two hypotheses (classes), namely *Positive*(**P**)

and *Negative*(**N**); hence, the frame of discernment is $\theta = \{\{P\}, \{N\}\}$ and $2^\theta = \{\{P\}, \{N\}, \{P, N\}\}$, whereas in the Bayesian case only the events $\{\{P\}, \{N\}\}$ would be considered.
$\{P\}$ and $\{N\}$ are referred to as *simple events* or *singletons*, while $\{P,\ N\}$ is referred to as *composite event*. Furthermore, also the following properties have to hold:

$$m(\varnothing) = 0 \qquad \sum_{A \subseteq 2^\theta} m(A) = 1$$

A.1 Classifier Statistical Characterization

Starting from the same *Weighted Confusion Matrix* (tab. A.1) described in the section 4.2.2, if we use as fusion block the Dempster-Shafer combination rule, than the **R** will be a vector of bpa where each element will be composed by:

(A.1) $$R(C_i) = bpa(C_i)$$

(A.2) $$\begin{aligned} bpa(C_i) = [&m(\{C_1\}) = e_{i1}, \\ &m(\{C_2\}) = e_{i2}, \\ &\vdots \\ &m(\{C_n\}) = e_{iM}, \\ &m(\{C_1, C_2, \ldots, C_M\}) = (1 - \textstyle\sum_{j=1}^{M} e_{ij})] \end{aligned}$$

	Assigned Class			
True Class	$\hat{C_1}$	$\hat{C_2}$	$\ldots$	$\hat{C_M}$
C_1	e_{11}	e_{12}	$\ldots$	e_{1n}
C_2	e_{21}	e_{22}	$\ldots$	e_{2n}
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
C_M	e_{M1}	e_{M2}	$\ldots$	e_{MM}

Table A.1: Weighted Confusion Matrix (WCM) for M classes classification

It is worth noting that in this case we use more information to characterize a single classifier, and so we are making a more accurate base classifiers statistical characterization than using the *Weighted Majority Voting*.

An open issue is to find another function $r()$ that use more information about the **WCM**. In this case we are not still using all the combined hypothesis.

A.2 Class and DoT Estimation

In this section we will describe how the system combine the **R** evaluated with $r()$, and how it's possible to obtain a DoT.

The aim of assigning a bpa is to describe the reliability of a particular classifier in reporting a specific event. Such a representation is suitable for combination, but as we want to deal with combined results in the same way, we also impose the constraint that the combination of several bpa by means of the D-S rule still has to be a bpa. The uncertainty in the final decision will be inversely proportional to the extent to which the base classifiers agree. If we have B base classifiers, the combination rule is such that:

$$m(A) = K \sum_{\bigcap_{i=1}^{B} A_i = A} \prod_{i=1}^{B} m_i(A_i)$$

where:

$$K^{-1} = 1 - \sum_{\bigcap_{i=1}^{B} A_i = \varnothing} \prod_{i=1}^{B} m_i(A_i)$$

$$= \sum_{\bigcap_{i=1}^{B} A_i \neq \varnothing} \prod_{i=1}^{B} m_i(A_i)$$

It is worth observing that the normalizing factor K is independent from any specific value of A. The value K can therefore be considered a constant, once the $bpas$ are fixed.

A.2.1 The two classes case

Now, we want to illustrate how it is possible to evaluate the the $Class$ and DoT described in the section 4.2.3 starting from a bpa values, that in our case represents **R**. We remember that a bpa is a vector of real numbers.

In the simple case of two classes problem, P and N, we have defined this value as:

$$Class = \begin{cases} P, & if \quad \dfrac{m(\{P\}) - m(\{N\})}{1 + m(\{P, N\})} > 0 \\ N, & Otherwise \end{cases} \tag{A.3}$$

$$DoT = abs\left(\frac{m(\{P\}) - m(\{N\})}{1 + m(\{P, N\})}\right) \tag{A.4}$$

The value $\frac{m(\{P\})-m(\{N\})}{1+m(\{P,N\})}$ is defined in such a way that, if its value is $+1$, there is the highest reliability on the hypothesis *P*; if it is -1, it is quite sure that we are observing a N hypothesis; if it is 0, there's the maximum uncertainty, hence the sample should be rejected.
In the first case, in fact, $m(\{P\}) = 0$ while $m(\{N\}) = 1$ and $m(\{N, P\}) = 0$. In the second case, the opposite scenario is verified, as $m(\{P\}) = 0$ while $m(\{N\}) = 1$ and $m(\{N, P\}) = 0$. In the latter case instead, $m(\{N\}) = 0$, $m(\{P\}) = 0$ and $m(\{N, P\}) = 1$.

A.2.2 The M-Classes Case

We have tried to define different strategies to evaluate the DoT defined for a *M-Classes* problem. For the sake of completeness we report both the strategies. The first one is more linked to the Dempster theory, whereas the second one use a transformation in a polar coordination. Both this strategy have their pro and cons, that will be discussed in the respective system sections.

General Case – Theoretical Method

This first formulation is based on the Hypothesis that we want to discriminate between a class P and several of variations of fake Hypothesis. Let us call them for example N_1,N_2, ...,N_n.

The general function which allows us to transform the *bpa* in a detection result descends from observation of the relations between *Belief*, *Plausibility* and *Uncertainty*. Let $A, B \in \theta$; hence:

$$Bel(B) = \sum_{A \subset B} m(A) \;\; ; \;\; Pls(B) = \sum_{A \cap B \neq \emptyset} m(A) \tag{A.5}$$

$$Unc(B) = Pls(B) - Bel(B) \tag{A.6}$$

In the two-event case, we observe that

$$\begin{aligned} Bel(\{N\}) &= m(\{N\}) \\ Pls(\{N\}) &= m(\{N\}) + m(\{N,P\}) \end{aligned}$$

$$\begin{aligned} Bel(\{P\}) &= m(\{P\}) \\ Pls(\{P\}) &= m(\{P\}) + m(\{N,P\}) \end{aligned}$$

Hence,

$$\begin{aligned} Unc(\{N\}) = Unc(\{P\} &= \\ &= m(\{N,P\}) \end{aligned}$$

Then for all the simple and compound hypotheses $H \in 2^\theta$ in which is not present the P Hypothesis we have defined the following parameter:

$$Y_H = \frac{m(\{H\})}{1 + Unc(\{H\})} \tag{A.7}$$

Obviously also for the P class we have the same thing:

$$Y_P = \frac{m(\{P\})}{1 + Unc(\{P\})} \tag{A.8}$$

Important considerations:

1. $0 \leq Y_H \leq 1$
2. - $Y_H = 0 \Longrightarrow m(H) = 0$
   - $Y_H = 1 \Longrightarrow m(H) = 1$ and $Unc(H) = 0$
   - $m(H) = 1 \Longrightarrow Y_H = 1$
3. if $\exists H : Y_H = 1 \Longrightarrow Y_N$ and all the other parameters are zero
4. $0 \leq \sum Y_H \leq 1$
5. from the 3) and 4) it is clear that if a generic $Y_H \rightarrow 1$ then all the other parameters tend to 0.

Starting from the parameters defined in equation A.7 and equation A.8, we can evaluate for each of the N simple Hypothesis F_i the following index:

(A.9) $$I(F_i) = \sum_{H \cap F_i \neq \emptyset} \frac{Y_H}{n_H}$$

Where n_H is the number of simple classes belong to H. The same for the class P.

(A.10) $$I(T) = Y_T$$

With these indexes we have defined new y and the new DoT as:

(A.11)
$$y = \begin{cases} P, & if \quad max_i\{I(F_i)\} - I(T) < 0 \\ \\ N_i, & if \quad max_i\{I(F_i)\} - I(T) < 0 \quad | \quad argmax_i\{I(F_i)\} - I(T) \end{cases}$$

(A.12) $$DoT = max\{I(F_i)\} - I(T)$$

It is important to notice that this final equation in the case of two hypotheses become the equation A.4.

A.2.3 General Case – Polar Coordinate Transformation Method

This method want to overcame the limitation of the previous one. In this case all the classes are equally important. The starting point are the equation A.7 and equation A.8 described in the previous section. With these Y parameters we transform the bpa space in a polar coordinate system. To obtain this, we divide the space into n angular sectors. The amplitude of each sector is:

$$Amplitude = \frac{2\pi}{n} \tag{A.13}$$

In this way we can draw a vector in the middle of each angular sector. The module of these vectors is equal to Y_i values. These vectors represent the belief in each class.

We define a vector that pass exactly in the middle of the angular sector and have a module equal to Y value. In this way for each angular sector we will correspond a class and we can define a function $m()$ that maps each class into an angular sector, that is, taken in input an angle θ, give the the class corresponding to the angular sector in which the angle is in:

$$Class = m(\theta) \tag{A.14}$$

Starting from this considerations, the DoT and the y are calculated starting from the resultant vector $Y_r = |Y_r| * e^{j\alpha}$ as:

$$y = m(\alpha) \tag{A.15}$$

$$DoT = |Y_r| * h(\alpha, Amplitude) \tag{A.16}$$

The h function is needed to give a weight to the position of the vector into the angular sector. If the vector is exactly in the middle, we will be more certain that

the true class was the one defined for that space, instead if it lies on the borderline between two angular sectors, our degree of belief is lower. This function is defined as:

$$h(\alpha, Amplitude) = -\left|\frac{\alpha_r - \frac{Amplitude}{2}}{\frac{Amplitude}{2}}\right| + 1 \tag{A.17}$$

We have chosen the triangular function, but also the Gaussian function could be a good choice.

For example, considering the case in which we have 4-classes C_1, C_2, C_3, C_4, as shown in the figure A.1.

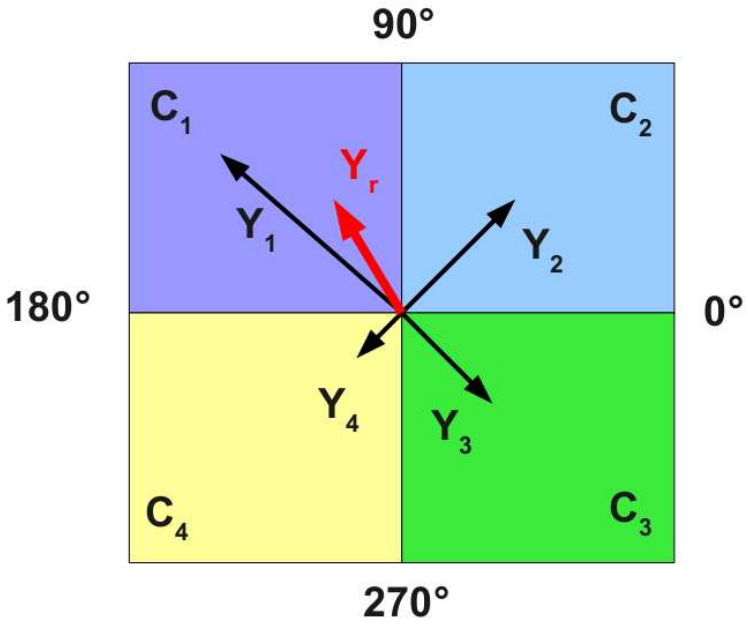


Figure A.1: A graphical example of the Y_r evaluation

In this case, the result y will be C_1 and the DoT will be the module of the red arrow (Y_r) weighted by the $h()$ function.

The cons of this approach is that the DoT depend on the coordinate system, and on the position of the Hypotheses on the axis. If we change the position of an hypothesis the value of the DoT will change.

Also in this case, as the previous one, if we consider only two Hypotheses we return at the equation A.4.

Appendix B

Bayesian Combining Rule

In this section we will apply the general methodology described in the chapter 4 using the Bayesian Combining Rule [14], as also described in section 3.3.3.

This rule will be used to develop the *Fusion Block* and for the classifiers characterization. That is, for defining **R** we choose to use a vector of probability estimation calculated staring from the Weighted Confusion Matrix.

B.1 Classifiers Statistical Characterization

Starting from the same **WCM** (tab. B.1) described in the section 4.2.2, if we use as fusion block the *Bayesian Combining Rule*, than the **R** will be:

	Assigned Class			
True Class	$\hat{C}_1$	$\hat{C}_2$	$\dots$	$\hat{C}_M$
C_1	e_{11}	e_{12}	$\dots$	e_{1n}
C_2	e_{21}	e_{22}	$\dots$	e_{2n}
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
C_M	e_{M1}	e_{M2}	$\dots$	e_{MM}

Table B.1: Weighted Confusion Matrix (WCM) for M classes classification

$$R(C_i) = \frac{e_{ii}}{\sum_{j=1}^{n} e_{ji}}, \quad \forall i = 1, 2, \dots, N \tag{B.1}$$

The equation B.1 expresses the *a posteriori* probability that a classifier gives the correct answer.

As in this case of Dempster-Shafer Combing Rule (Appendix A) we use more information to characterize a single classifier, and so we are making a more accurate base classifiers statistical characterization than using the *Weighted Majority Voting*.

For example, if we consider the WCM in table B.2

	Assigned Class		
True Class	$\hat{A}$	$\hat{B}$	$\hat{C}$
A	0.7	0.1	0.1
B	0.2	0.5	0.1
C	0.3	0	0.6

Table B.2: Possible Weighted Confusion Matrix for a three classes problem

$$R(A) = \frac{0.7}{0.7 + 0.2 + 0.3} = 0.58$$
$$R(B) = \frac{0.5}{0.1 + 0.5 + 0} = 0.83$$
$$R(C) = \frac{0.6}{0.1 + 0.1 + 0.6} = 0.75$$

B.2 Class and DOT Estimation

In this section we will describe how the system combines the **R** evaluated with the $r()$ function, and how it is possible to obtain the DoT.

In particular, if we indicate with $(\hat{y}_i^k = C_j)$ the event that the $k-th$ classifier assigns the input samples x_i to the class C_j, the output class y will be:

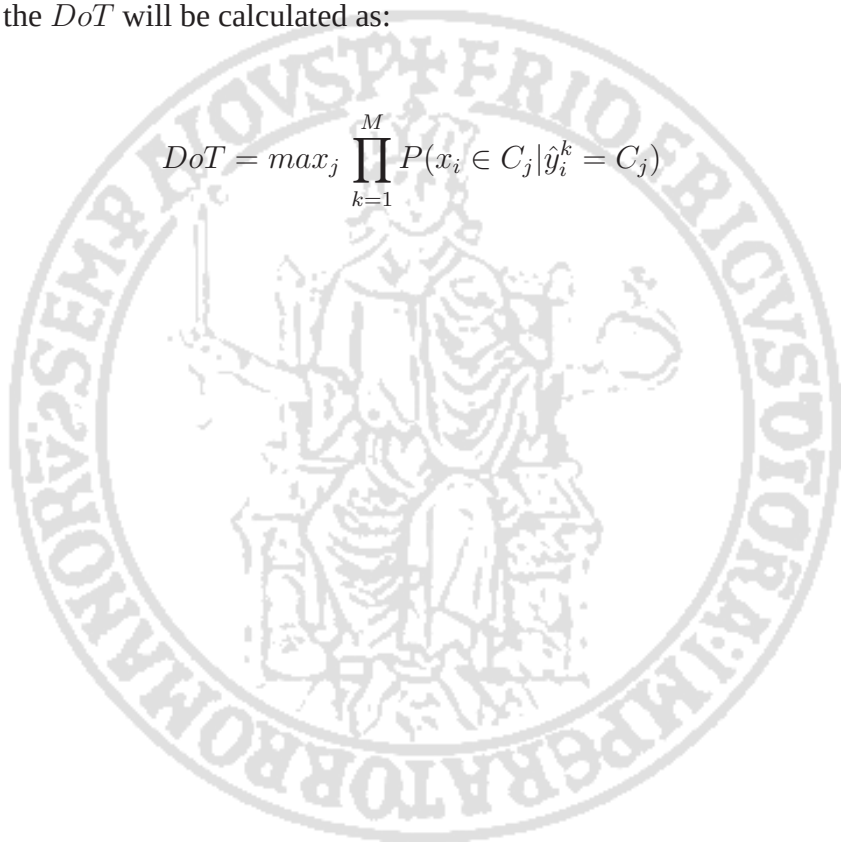
(B.2) $$y = argmax_j \, P(x_i \in C_j | \hat{y}_i^1 = C_j, \hat{y}_i^2 = C_j, \ldots, \hat{y}_i^M = C_j)$$

If the classifiers can be assumed independent among each other and the *a priori* probability is the same for all the classes, it can be shown that the eq. B.2 can be written as:

$$C = argmax_j \prod_{k=1}^{M} P(x_i \in C_j | \hat{y}_i^k = C_j) \tag{B.3}$$

And the DoT will be calculated as:

$$DoT = max_j \prod_{k=1}^{M} P(x_i \in C_j | \hat{y}_i^k = C_j) \tag{B.4}$$



Bibliography

[1] C. Abad and R. Bonilla. An analysis on the schemes for detecting and preventing ARP cache poisoning attacks. In *Proc. of ICDCS Workshops*, page 60. IEEE Computer Society, 2007.

[2] I. Androutsopoulos, J. Koutsias, K.V. Chandrinos, G. Paliouras, and C.D. Spyropoulos. An evaluation of naive bayesian anti-spam filtering. In G. Potamias, V. Moustakis, and M.n van Someren, editors, *Proc. of the 11th European Conf. on Machine Learning (ECML)*, pages 9–17, Barcelona, Spain, 2000.

[3] H.B. Aradhye, G.K. Myers, and J.A. Herson. Image analysis for efficient categorization of image-based spam e-mail. In *Proc. of the Eighth Intern. Conf. on Document Analysis and Recognition (ICDAR)*, pages II: 914–918, 2005.

[4] T. Auld, A.W. Moore, and S.F. Gull. Bayesian neural networks for internet traffic classification. *IEEE Transactions on Neural Networks*, 18(1):223–239, 2007.

[5] C. Barbu, R. Iqbal, and J. Peng. An ensemble approach to robust biometrics fusion. In *Biometrics*, page 56, 2006.

[6] M. Barreno, B. Nelson, R. Sears, A.D. Joseph, and J.D. Tygar. Can machine learning be secure? In Ferng-Ching Lin, Der-Tsai Lee, Bao-Shuh Lin, Shi-uhpyng Shieh, and Sushil Jajodia, editors, *ASIACCS*, pages 16–25. ACM, 2006.

[7] V. J. Baston and F. A. Bostock. An evasion game with barriers. *SIAM Journal on Control and Optimization*, 26(5):1099–1105, September 1988.

[8] L. Bernaille, R. Teixeira, and K. Salamatian. Early application identification. In *CoNEXT '06: Proceedings of the 2006 ACM CoNEXT conference*, pages 1–12, New York, NY, USA, 2006. ACM.

[9] B. Biggio, G. Fumera, I. Pillai, and F. Roli. Image spam filtering using visual information. In *Proc. of the 14th Intern.l Conf. on Image Analysis and Processing (ICIAP)*, pages 105–110, 2007.

[10] B. Biggio, G. Fumera, and F. Roli. Multiple classifier systems for adversarial classification tasks. In Jon Atli Benediktsson, Josef Kittler, and Fabio Roli, editors, *MCS*, volume 5519 of *Lecture Notes in Computer Science*, pages 132–141. Springer, 2009.

[11] E. Blanzieri and A. Bryl. A survey of learning-based techniques of email spam filtering. Technical Report DIT-06-056, Informatica e Telecomunicazioni, University of Trento, 2006.

[12] H. Cheng, Z. Qin, Q. Liu, and M. Wan. Spam image discrimination using support vector machine based on higher-order local autocorrelation feature extraction. In *Proc. IEEE Conference on Cybernetics and Intelligent Systems*, pages 1017–1021, 2008.

[13] W.W. Cohen. Learning rules that classify E-mail. In *Papers from the AAAI Spring Symposium on Machine Learning in Information Access*, pages 18–25, 1996.

[14] L.P. Cordella, P. Foggia, C. Sansone, and M. Vento. Reliability parameters to improve combination strategies in multi-expert systems. *Pattern Anal. Appl*, 2(3):205–214, 1999.

[15] L.P. Cordella and C. Sansone. A multi-stage classification system for detecting intrusions in computer networks. *Pattern Anal. Appl*, 10(2):83–100, 2007.

[16] A. Dainotti, W. de Donato, and A. Pescapè. TIE: A community-oriented traffic classification platform. In Maria Papadopouli, Philippe Owezarski, and Aiko Pras, editors, *TMA*, volume 5537 of *Lecture Notes in Computer Science*, pages 64–74. Springer, 2009.

[17] A. Dainotti, W. de Donato, A. Pescapè, and P.S. Rossi. Classification of network traffic via packet-level hidden markov models. In *GLOBECOM*, pages 2138–2142. IEEE, 2008.

[18] N. Dalvi, P. Domingos, M., S. Sanghai, and D. Verma. Adversarial classification. In *Proceedings of SIGKDD04*, pages 99–108, 2004.

[19] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. P2P-based collaborative spam detection and filtering. In Germano Caronni, Nathalie Weiler, and Nahid Shahmehri, editors, *Proc. of the Fourth Intern. Conf. on Peer-to-Peer Computing*, pages 176–183. IEEE Computer Society, 2004.

[20] S. Deerwester, S.T. Dumais, G.W. Furnas, T.K. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41:391–407, 1990.

[21] T.G. Dietterich. Ensemble methods in machine learning. In *Multiple Classifier Systems*, pages 1–15, 2000.

[22] M. Dredze, R. Gevaryahu, and A. Elias-Bachrach. Learning fast classifiers for image spam. In *Proc. of the second Conference on e-mail and Anti-Spam (CEAS)*, pages 487–493, 2007.

[23] H. Drucker, D. Wu, and V. N. Vapnik. Support vector machines for Spam categorization. *IEEE Transactions on Neural Networks*, 10(5):1048, September 1999.

[24] R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern classification*. Wiley, pub-WILEY:adr, second edition, 2001.

[25] J. Erman, A. Mahanti, and M.F. Arlitt. Internet traffic identification using machine learning. In *GLOBECOM*. IEEE, 2006.

[26] J. Erman, A. Mahanti, M.F. Arlitt, I. Cohen, and C.L. Williamson. Offline/realtime traffic classification using semi-supervised learning. *Perform. Eval*, 64(9-12):1194–1213, 2007.

[27] A. Este, F. Gargiulo, F. Gringoli, L. Salgarelli, and C. Sansone. Pattern recognition approaches for classifying ip flows. In *SSPR/SPR*, pages 885–895, 2008.

[28] E. Frank, M.A. Hall, G. Holmes, R. Kirkby, and B. Pfahringer. WEKA - A machine learning workbench for data mining. In Oded Maimon and Lior Rokach, editors, *The Data Mining and Knowledge Discovery Handbook*, pages 1305–1314. Springer, 2005.

[29] E.P. Freire, A. Ziviani, and R.M. Salles. On metrics to distinguish skype flows from http traffic. In *Proc. Latin American Network Operations and Management Symposium LANOMS 2007*, pages 57–66, 2007.

[30] Y. Freund and R.E. Schapire. Experiments with a new boosting algorithm. In *Proc. 13th International Conference on Machine Learning*, pages 148–146. Morgan Kaufmann, 1996.

[31] G. Fumera, I. Pillai, and F. Roli. Spam filtering based on the analysis of text information embedded into images. *Journal of Machine Learning Research*, 7:2699–2720, 2006.

[32] F. Gargiulo, L.I. Kuncheva, and C. Sansone. Network protocol verification by a classifier selection ensemble. In Jon Atli Benediktsson, Josef Kittler, and Fabio Roli, editors, *MCS*, volume 5519 of *Lecture Notes in Computer Science*, pages 314–323. Springer, 2009.

[33] F. Gargiulo, C. Mazzariello, and C. Sansone. A self-training approach for automatically labeling IP traffic traces. In Marek Kurzynski, Edward

Puchala, Michal Wozniak, and Andrzej Zolnierek, editors, *Computer Recognition Systems 2*, volume 45 of *Advances in Soft Computing*, pages 705–717. Springer, 2008.

[34] G. Giacinto, R. Perdisci, M. Del Rio, and F. Roli. Intrusion detection in computer networks by a modular ensemble of one-class classifiers. *Information Fusion*, 9(1):69–82, 2008.

[35] J. Graham-Cumming. How to beat a bayesian spam filter. In *Proceedings of the Spam Conference*, 2004. Available: http://www.jgc.org/SpamConference011604.pps.

[36] R.M. Haralick. Statistical and structural approaches to texture. *Proceedings of IEEE*, 67(5):786–804, May 1979.

[37] J.P. Hespanha. Deception in non-cooperative games with partial information, September 02 2000.

[38] H. Huang, W. Guo, and Y. Zhang. A novel method for image spam filtering. In *Proc. 9th International Conference for Young Computer Scientists (ICYCS)*, pages 826–830, 2008.

[39] Y.S. Huang and C.Y. Suen. A method of combining multiple experts for the recognition of unconstrained handwritten numerals. *Pattern Analysis and Machine Intelligence*, 17(1):90–94, January 1995.

[40] A.K. Jain, R.P.W. Duin, and J. Mao. Statistical pattern recognition: A review. *IEEE Trans. Pattern Anal. Mach. Intell*, 22(1):4–37, 2000.

[41] D. Jurafsky and J.H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition (Prentice Hall Series in Artificial Intelligence*. Prentice Hall; 2 edition, 2009.

[42] T. Karagiannis, A. Broido, N. Brownlee, k. claffy, and M. Faloutsos. Is P2P dying or just hiding? In *IEEE Globecom*, volume 3, pages 1532–1538, November 2004.

[43] T. Karagiannis, A. Broido, M. Faloutsos, and K. claffy. Transport layer identification of P2P traffic. In *4th Internet Measurement Conf. (IMC)*, pages 121–134, October 2004.

[44] T. Karagiannis, K. Papagiannaki, and M. Faloutsos. BLINC: multilevel traffic classification in the dark. In Roch Guérin, Ramesh Govindan, and Greg Minshall, editors, *SIGCOMM*, pages 229–240. ACM, 2005.

[45] M. Kearns and M. Li. Learning in the presence of malicious errors. *SIAM Journal on Computing*, 22(4):807–837, August 1993.

[46] L.I. Kuncheva. Classifier ensembles for changing environments. In *Multiple Classifier Systems*, pages 1–15, 2004.

[47] L.I. Kuncheva. *Combining Pattern Classifiers: Methods and Algorithms*. Wiley-Interscience, 2004.

[48] L. Lam. Classifier combinations: Implementations and theoretical issues. In *IWMCS: International Workshop on Multiple Classifier Systems*. LNCS, 2000.

[49] D. Lei, Y. Xiaochun, and X. Jun. Optimizing traffic classification using hybrid feature selection. In *Proc. Ninth International Conference on Web-Age Information Management WAIM '08*, pages 520–525, 2008.

[50] K.E. Lochbaum and L.A. Streeter. Comparing and combining the effectiveness of latent semantic indexing and the ordinary vector space model for information retrieval. *Information Processing and Management*, 25(6):665–76, 1989.

[51] D. Lowd and C. Meek. Adversarial learning. In Robert Grossman, Roberto J. Bayardo, and Kristin P. Bennett, editors, *KDD*, pages 641–647. ACM, 2005.

[52] D. Lowd and C. Meek. Good word attacks on statistical spam filters. In *CEAS*, 2005.

[53] Y. Lu. Knowledge integration in a multiple classifier system. *Appl. Intell*, 6(2):75–86, 1996.

[54] C. Manning and H. Schtze. *Foundations of Statistical Natural Language Processing*, chapter 16: Text Categorization, pages 575–608. The MIT Press, Cambridge, US, 1999.

[55] E. Marasco and C. Sansone. Improving the accuracy of a score fusion approach based on likelihood ratio in multimodal biometric systems. In Pasquale Foggia, Carlo Sansone, and Mario Vento, editors, *ICIAP*, volume 5716 of *Lecture Notes in Computer Science*, pages 509–518. Springer, 2009.

[56] M. Mellia, A. Pescapè, and L. Salgarelli. Traffic classification and its applications to modern networks. *Computer Networks*, 53(6):759–760, 2009.

[57] P. Melville and R.J. Mooney. Diverse ensembles for active learning. In Carla E. Brodley, editor, *ICML*, volume 69 of *ACM International Conference Proceeding Series*. ACM, 2004.

[58] V. Metsis, I. Androutsopoulos, and G. Paliouras. Spam filtering with naive bayes - which naive bayes? In *Proc. of the second Conference on e-mail and Anti-Spam (CEAS)*, Mountain View, CA, USA., 2006.

[59] A.W. Moore and K. Papagiannaki. Toward the accurate identification of network applications. In Constantinos Dovrolis, editor, *PAM*, volume 3431 of *Lecture Notes in Computer Science*, pages 41–54. Springer, 2005.

[60] A.W. Moore and D. Zuev. Internet traffic classification using bayesian analysis techniques. In Derek L. Eager, Carey L. Williamson, Sem C. Borst, and John C. S. Lui, editors, *SIGMETRICS*, pages 50–60. ACM, 2005.

[61] N.C. Oza. Aveboost2: Boosting for noisy data. In *Multiple Classifier Systems*, pages 31–40, 2004.

[62] V. Paxson. Bro: A system for detecting network intruders in real time. *Proceedings of the 7th security symposium. (USENIX Association: Berkeley, CA)*, 1998.

[63] V. Piuri and F. Scotti. Fingerprint biometrics via low-cost sensors and webcams. In *Biometrics: Theory, Applications, and Systems*, pages 1–6, 2008.

[64] M.F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[65] R.K. Powalka, N. Sherkat, and R. J. Whitrow. Multiple recognizer combination topologies, April 19 1995.

[66] L.A. Rastrigin and R.H. Erenstein. *Method of Collective Recognition*. Energoizdat, Moscow, 1981. (In Russian).

[67] L. Salgarelli and T. Karagiannis. Comparing traffic classifiers. *Computer Communication Review*, 37(3):65–68, 2007.

[68] G. Schryen. *Anti-Spam Measures: Analysis and Design*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[69] P. Smets and P. Magrez. The measure of the degree of truth and the grade of membership. *Fuzzy Sets Syst.*, 25(1):67–72, 1988.

[70] D.F. Specht. Probabilistic neural networks. *Neural Networks*, 3:109–118, 1990.

[71] J. Stolfo. Cost-based modeling and evaluation for data mining with application to fraud and intrusion detection. results from the jam project by salvatore;, 2000.

[72] C.Y. Suen and L. Lam. Multiple classifier combination methodologies for different output levels. In *Multiple Classifier Systems*, pages 52–66, 2000.

[73] C. Thiel. Classification on soft labels is robust against label noise. In Ignac Lovrek, Robert J. Howlett, and Lakhmi C. Jain, editors, *KES (1)*, volume 5177 of *Lecture Notes in Computer Science*, pages 65–73. Springer, 2008.

[74] M. Wan, F. Zhang, H. Cheng, and Q. Liu. Text localization in spam image using edge features. In *Proc. International Conference on Communications, Circuits and Systems (ICCCAS)*, pages 838–842, 2008.

[75] Watanabe. Pattern recognition as induction. In Watanabe, editor, *Pattern Recognition*, pages 97–183, 1985.

[76] N. Williams, S. Zander, and G. Armitage. A preliminary performance comparison of five machine learning algorithms for practical ip traffic flow classification. *SIGCOMM Comput. Commun. Rev.*, 36(5):5–16, 2006.

[77] G.L. Wittel and S.F. Wu. On attacking statistical spam filters. In *Proceedings of the First Conference on Email and Anti-Spam (CEAS)*, 2004. Available: http://www.ceas.cc/papers-2004/170.pdf.

[78] K. Woods, W. P. Kegelmeyer, and K. Bowyer. Combination of multiple classifiers using local accuracy estimates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19:405–410, 1997.

[79] C.V. Wright, F. Monrose, and G.M. Masson. On inferring application protocol behaviors in encrypted network traffic. *Journal of Machine Learning Research*, 6:2745–2769, 2006.

[80] C.T. Wu, K.T. Cheng, Q.A. Zhu, and Y.L. Wu. Using visual features for anti-spam filtering. In *Proc. IEEE Conference on Image Processing (ICIP)*, pages III: 509–512, 2005.

[81] S. Zander, T.T.T. Nguyen, and G.J. Armitage. Automated traffic classification and application identification using machine learning. In *LCN*, pages 250–257. IEEE Computer Society, 2005.

[82] S. Zander, T.T.T. Nguyen, and G.J. Armitage. Self-learning IP traffic classification based on statistical flow characteristics. In Constantinos Dovrolis, editor, *PAM*, volume 3431 of *Lecture Notes in Computer Science*, pages 325–328. Springer, 2005.

[83] F. Zhou, L. Zhuang, B. Y. Zhao, L. Huang, A.D. Joseph, and J. Kubiatowicz. Approximate object location and spam filtering on peer-to-peer systems. In Markus Endler and Douglas C. Schmidt, editors, *Proc. of ACM/IFIP/USENIX International Middleware Conference*, volume 2672 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2003.