UNIVERSITÀ DEGLI STUDI DI MILANO
Facoltà di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea Magistrale in Informatica

Multi-view learning for modeling audio-visual patterns

Relatore: Prof. Nicolò Cesa-Bianchi
Correlatore: Dott. Francesco Orabona

Tesi di Laurea di
Marco FORNONI
Matr. N. 701748

Anno Accademico 2008-2009

Acknowledgements

This work has been carried out at Idiap Research Institute of Martigny (Switzerland) and supported by the DIRAC undergraduate internship program. During the months spent on experiments, or on theoretical issues, my personal knowledges and skills have been deeply stressed and very positively affected. I have learned many interesting things about on-line learning algorithms and their application and I had also the opportunity to train my abilities and knowledges in scientific research methods. The final thesis is the result of a remarkable cooperation between many people to whom I am very grateful and I wish to thank.
First of all I am very grateful to Prof. Nicolò Cesa-Bianchi for his support and inspiration and for having given me the chance to make this important research experience. It has been my first abroad research experience and I have to say that when I first asked Prof. Cesa-Bianchi a thesis, I could never imagine that I might have been involved in something like this. I also had the opportunity to collaborate with Francesco Orabona and Barbara Caputo, whom I would like to thank for their support, suggestions and direct contributions to this work.

A special thank finally goes to my parents Bernardo Valente Fornoni and Teresa Mangiatordi, my sister Valeria Fornoni, my girlfriend Raffaella Stoppani, my uncle Germano Fornoni and my aunt Sonia Nissim, my friends Antonio Ricciardi, Edoardo Tosca, Marco Ferramosca, Stefano Mingardi and to all of those who have supported me and have given a special flavor to my life. This work is dedicated to them.

Contents

1 Introduction **4**

1.1 Related work 5

1.2 Contribution of the Thesis 6

1.3 Outline 7

2 Linear-threshold classifiers **8**

2.1 Perceptron and gradient descent classifiers 9

2.1.1 A more general linear-threshold model 11

2.1.2 Convex optimization 14

2.1.3 Bregman divergence optimization 18

2.2 Kernel Methods 22

2.2.1 Dual formulation 22

2.2.2 Reproducing Kernel Hilbert Space theory 26

3 Matrix multi-view Perceptron **31**

3.1 The model 32

3.1.1 Prediction 35

3.1.2 Kernelization 39

3.1.3 Low risk hypothesis extraction 40

3.2 Motivation for further work 44

4 Orthogonal matrix multi-view Perceptron **46**

4.1 The model 46

4.1.1 Prediction 48

4.2 Experiments 51

4.2.1 Adult dataset 51

4.2.2 News20 dataset 53
4.2.3 Eth-80 dataset 54
4.3 Discussion of the results 61

5 Conclusions and future works **62**

A Appendix **65**
A.1 Fourier series in Hilbert spaces 65
A.2 Lemma 66

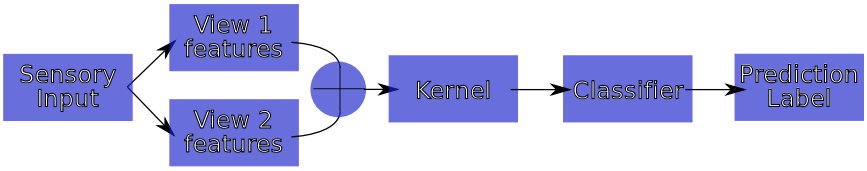
References **72**

Chapter 1

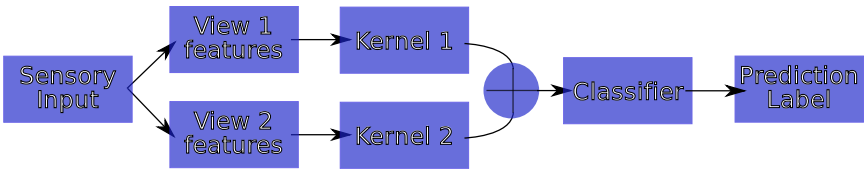
Introduction

Learning from multiple sensory channels is present in almost any biological system and, indeed, multimodality is one of the main reasons why biological systems are able to cope with the complexity of the world. This thesis deals with the problem of learning from multiple sources of information, from an on-line supervised machine learning perspective. From this point of view there are many different ways to approach the problem, for example there are different possible levels of the classification process at which views could be integrated:

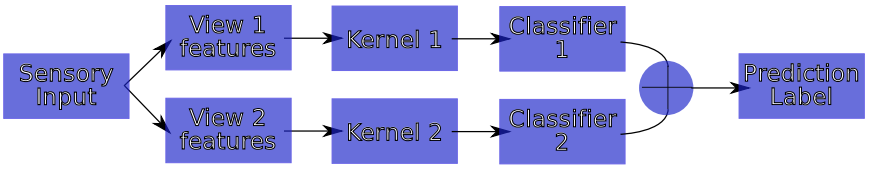
- *low level*, views are combined together at the feature level



- *mid level*, views are combined together while building a classification decision function



- *high level*, views are used separately to produce confidence estimates, which are then combined together



1.1 Related work

Anemuller et al. (2008) have faced this problem from a biologically motivated point of view and have shown that high-level integration produce better results with respect to low-level feature concatenation and a simple mid-level kernel combination strategy. However in (Jie et al., 2008) the same authors show that the low-level feature concatenation is more robust to noise. Moreover, as an interesting example of high-level integration, Luo Jie et. al in (Jie et al., 2009) create a 2 level architecture, where the output of K-classifiers (one for each view) are optimally combined by a top classifier.

There is also a plenty of literature in the field of mid-level integration: Blum and Mitchell (1998) use unlabeled data to boost the performances of a classifier, by co-training different views. Sindhwani and Rosemberg (2008) use a co-regularization technique to construct a Reproducing Kernel Hilbert Space (*RKHS*) and prove bounds for the 2-views problem. Wang and Chen (2009) use a co-regularization technique to create a RKHS for learning M views, from a single-view dataset. Finally, Cavallanti et al. (2008) propose an on-line matrix integration strategy where instances are matrix composed of many view vectors. This algorithm, called *matrix multi-view Perceptron* is a special case of a more general class of potential-based gradient descent learners and its analysis, application and improvement will be the goal of this thesis.

1.2 Contribution of the Thesis

This thesis approaches the problem of multi-view learning from a matrix classification point of view. Its center of mass is matrix multi-view Perceptron algorithm (Cavallanti et al., 2008) and its aim is its theoretical and empirical evaluation, improvement and comparison to other multi-view learning frameworks.

The main contributions of this thesis are:

- *Analysis of matrix Perceptron algorithm.* We study matrix Perceptron algorithm and provide a prediction formula that explicitly highlights how the single-view instance vectors are combined together in the prediction phase. We also provide a technique to apply a popular on-line to batch conversion theorem also to matrix Perceptron in its dual form (suitable for kernel substitution). Moreover we highlight some problems of the matrix approach to multi-view learning, such as: the assumption that all the single-view vectors lie in spaces of the same dimensionality is too restrictive; the inner products between different views raise the problem of how to choose a similarity measure between different views; the computational complexity in time of matrix Perceptron is quadratic with respect to the number of views.

- *Proposal of a modified version of matrix Perceptron algorithm.* We propose a modified version of the original algorithm aimed to solve the above mentioned problems. Specifically we employ a simple orthogonalization technique to turn the original matrix algorithm into a basically vectorial algorithm and we show how its vectorial prediction formula is related to the original matrix prediction formula.

- *Empirical evaluation of different multi-view algorithms.* We show that on two artificial multi-view datasets, our vectorial algorithm performs as well as the original matrix algorithm, while in a true multi-view task where the original algorithm is not applicable, our algorithm competes with another state-of-the-art multi-view algorithm. However, none of the algorithms tested succeeds in outperforming the other algorithms in all the tasks.

1.3 Outline

The report is composed of five Chapters. After this introductory Chapter the reader will learn some basic theory on Potential-Based gradient descent learners, Bregman divergence and some relevant theory of Kernel Methods. In Chapter 3 we present Cavallanti's matrix multi-view Perceptron algorithm with some analysis on its internal behavior, multiple kernel and its points of strength and weakness. In Chapter 4 we propose a different algorithm supposed to address some of the weakness of matrix multi-view Perceptron and we provide some experimental performances evaluations.

Chapter 2

Linear-threshold classifiers

Learning can be seen "as the phenomenon of knowledge acquisition in the absence of explicit programming" (Valiant, 1984). A *learning machine* consists of a *learning protocol* together with a *decision procedure*. *Learning protocol* specifies the way information is obtained from outside, while *decision procedure* specifies how the learning is done, once information has been obtained. Information from outside should be enough for the purpose of learning a solution for the problem, but not too much as the explicit program that solves the problem (if it exists). The learning protocol, together with the decision procedure is also often referred as the *model* of the learning problem.

In on-line approaches, learning is modeled as an iterative process between a student and a teacher: student is provided an instance of a problem and has to try to solve it; teacher then gives the student some feedback that can be used by the student to improve its abilities. The feedback should not convey the explicit solution of the problem, while still making learning possible. In a fully supervised fashion we assume that the feedback given to the student is the correct answer, after the student has given its guess. The student in this way does not receive any explicit programming information, but can capitalize on that information in order to improve its future predictions.

2.1 Perceptron and gradient descent classifiers

In the field of machine learning for pattern classification student's purpose is to learn to correctly classify instances it is faced. Student is often called *learner*, or *forecaster* and student's guess is called *prediction*, while teacher's feedback is called *outcome*.

Learning protocol At each time-step $t = 1, 2, ...$ the forecaster is faced an instance $\mathbf{x}_t$ and has to give its prediction $\hat{y}_t$ about the class of the instance. Once it has given its prediction, the teacher unveils the outcome y_t. The outcome is the information about the correct class the object belonged to. We will concentrate on *binary* classification tasks in which there are only two classes.

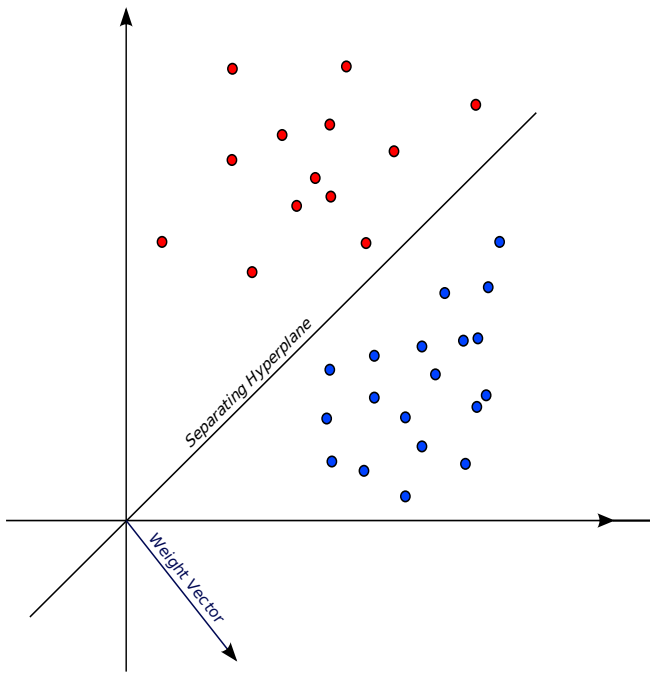


Figure 2.1: An Hyperplane splits the blue class from the red class.

Now that we have defined the learning protocol we can address the problem of how to capitalize on teacher's feedback in order to improve learning machine answer. A lot of work has been done in this field and one of the most famous and long-lived solution is Rosemblat's Perceptron algorithm

Assume $\mathbf{x}_t \in \mathcal{X} \subseteq \mathbb{R}^d$, $\mathbf{w}_t \in \mathbb{R}^d$.
Initialization $\mathbf{w}_0 = \mathbf{0}$.
At each time t=1,2,..... do the following:

```
Observe instance x_t ∈ X;
Predict label y_t ∈ {−1, 1} with ŷ_t = sign(⟨w_{t−1}, x_t⟩);
Observe actual label y_t ∈ {−1, 1};
if ŷ_t ≠ y_t then
    w_t = w_{t−1} + y_t x_t
else
    w_t = w_{t−1}
end if
```

Figure 2.2: Perceptron algorithm

(Rosenblatt, 1958).
Perceptron simple learning procedure can be formalized in this way: instances are vectors $\mathbf{x}_t$ in an *instance space* $\mathcal{X} \subseteq \mathbb{R}^d$, predictions $\hat{y}_t$ and outcomes y_t lie in $\{-1, 1\}$, while a weight vector $\mathbf{w}_t$ is supposed to lie in $\mathbb{R}^d$.

Perceptron Decision procedure At each time-step $t = 1, 2, ...$ Perceptron predicts using the signum of the projection of the instances on the weight vector (multiplied by $\|\mathbf{w}_t\|$), formally: $\hat{y}_t = \text{sign}\left(\langle \mathbf{w}_{t-1}, \mathbf{x}_t \rangle\right)$. This is equivalent to consider the $d-1$ dimensional hyperplane passing through origin and orthogonal to $\mathbf{w}_t$, as a separating hyperplane that splits the positive class from the negative one (see Figure 2.1).

Perceptron algorithm updates its weights *conservatively*, that is it updates if and only if the prediction was wrong: $\hat{y}_t \neq y_t$, or equivalently $\hat{y}_t y_t \leq 0$. Update rule is given by: $\mathbf{w}_t = \mathbf{w}_{t-1} + y_t \mathbf{x}_t$. If we assume $\mathbf{w}_0 = \mathbf{0}$, Perceptron weight vector is a point in the instance space $\mathcal{X}$, since at time T it is a linear combination of the instances: $\mathbf{w}_T = \sum_{t=1}^{T} \mathbb{I}_{\{\hat{y}_t \neq y_t\}} y_t \mathbf{x}_t$, where $\mathbb{I}$ is the indicator function.

It is worth noting that Perceptron algorithm can successfully learn only linearly-separable relations. However, in Section 2.2 we will see how we can overcome this problem by the mean of *kernel functions*.

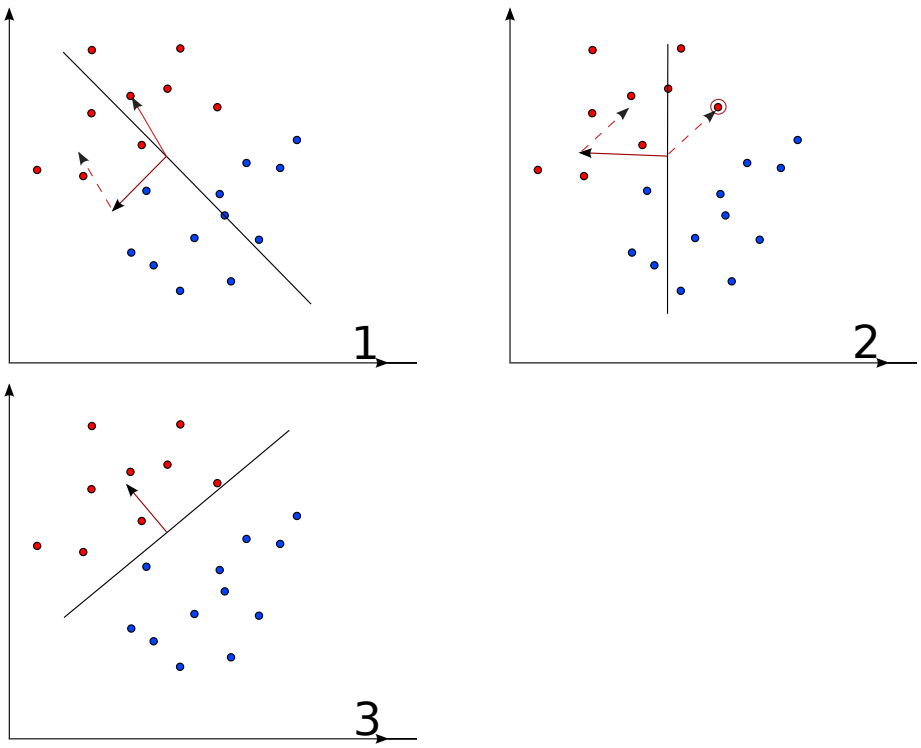


Figure 2.3: Convergence of Perceptron learning updates.

In Figure 2.3 we see an example execution of the Perceptron algorithm. We have two classes: red class ($y_t = 1$) and blue class ($y_t = -1$). Red arrow point towards the Red class, so that points on the arrow side of the hyperplane are classified as red and the remaining are classified as blue. Circled points are missclassified so that at each step, the weight vector is updated as $\mathbf{w}_t = \mathbf{w}_{t-1} + y_t \mathbf{x}_t$.

2.1.1 A more general linear-threshold model

Many on-line supervised classification algorithms can be analyzed as being special cases of a general linear-threshold learning model that makes use of the notions of *distance function* and *loss function*. This model can be formalized as follow.

2.1.1 Model formalization. Every instance $\mathbf{x}_t$ is supposed to belong to an *instance space* $\mathcal{X} \subseteq \mathbb{R}^d$, every outcome y_t to an *outcome space* $\mathcal{Y}$ and every prediction $\hat{y}_t$ is supposed to belong to a *decision space* $\mathcal{D} = \mathcal{Y} = \{-1, 1\}$.

The forecaster produce *hypothesis* functions $f_{\mathbf{w}_t} : \mathbb{R}^d \to \mathcal{D}$ in an *hypothesis space* $\mathcal{H}$. These hypotheses are linear-threshold functions parametric in a *weight vector* $\mathbf{w}_t \in \mathbb{R}^d$ and they have the following form:

$$f_{\mathbf{w}_t}(\cdot) = \text{sign}(\langle \mathbf{w}_t, \cdot \rangle) \tag{2.1.1}$$

The problem is also parametric with respect to a loss function $\ell_t : \mathbb{R}^d \to \mathbb{R}^+$ and a premetric $d : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^+$.

Loss functions were first introduced in *Decision Theory* as a way to formalize the loss suffered by taking a wrong decision and now play a very important rule in machine learning. In our on-line classification task for example, a forecaster will for sure suffer some loss if an instance is missclassified. It is also possible that it will incur some loss even if the instance is correctly classified, however in this case the forecaster will not update its weights. This last policy is known as "*conservative*" and it is not a general and shared approach: there is a version of the Perceptron algorithm, called Ballseptron (Shalev-Shwartz and Singer, 2005) which updates weights more aggressively, when the margin is below a certain threshold, while other aggressive approaches have been explored also in (Crammer et al., 2006).

In any case throughout this work we will consider only conservative algorithms, that update the weight vector only in case of misclassification.

2.1.2 Loss function. A function $\ell_t : \mathbb{R}^d \to \mathbb{R}^+$ whose value $\ell_t(\mathbf{w})$ at time t measure the disagreement between the true outcome y_t and the prediction $\hat{y}_t = f_{\mathbf{w}}(\mathbf{x}_t)$, with respect to weight vector $\mathbf{w} \in \mathbb{R}^d$ is called *loss function.*

We can also define the *cumulative loss* as the overall loss incurred over a finite sequence of decisions.

2.1.3 Cumulative loss function. If an on-line learning algorithm is run with weight a vector $\mathbf{w}$ on a sequence of examples of length T, the cumulative loss function $L_T : \mathbb{R}^d \to \mathbb{R}^+$ suffered by the forecaster is defined as:

$$L_T(\mathbf{w}) \stackrel{def}{=} \sum_{t=1}^{T} \ell_t(\mathbf{w}) \tag{2.1.2}$$

As previously anticipated our general on-line learning model will rely also on the notion on distance function. This function is usually supposed to be a metric (that is to satisfy some axioms from Euclidean geometry), however for our purposes we will consider a more general definition of distance.

2.1.4 Premetric distance function. A function $d : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ such that:

$$d(\mathbf{w}_1, \mathbf{w}_2) \geq 0$$
$$d(\mathbf{w}, \mathbf{w}) = 0$$

where $\mathbf{w}_1, \mathbf{w}_2 \in \mathbb{R}^d$, is called *premetric*.
This is a very relaxed metric definition that does not need to satisfy the triangle inequality, the symmetry axiom and the identity of indiscernibles axiom.

Learning Protocol With the above formalization, we define the learning protocol as follow:

- The forecaster is provided an initial weight vector $\mathbf{w}_0 \in \mathbb{R}^d$
- at each time-step $t = 1, 2, 3, ...$ the forecaster is provided an instance $\mathbf{x}_t \in \mathbb{R}^d$ and is required to give its prediction $\hat{y}_t$
- after having given its prediction $\hat{y}_t$, the forecaster has access to the true outcome y_t
- if $\hat{y}_t \neq y_t$ the forecaster incurs a loss $\ell_t(\mathbf{w}_{t-1})$. The weight vector $\mathbf{w}_{t-1}$ is subsequently updated.

Decision procedure As in Perceptron algorithm prediction is given by $\hat{y}_t = f_{\mathbf{w}_{t-1}}(\mathbf{x}_t) = \text{sign}\left(\langle \mathbf{w}_{t-1}, \mathbf{x}_t \rangle\right)$, but in this model we give a more general strategy for the weight vector update, this strategy can be stated as follow: weight vector has to be updated in order to minimize the loss incurred if $(\mathbf{x}_t, y_t)$ is received again on the next step, while minimizing the distance $d(\mathbf{w}_t, \mathbf{w}_{t-1})$ from the new weight vector $\mathbf{w}_t$ to the old weight vector $\mathbf{w}_{t-1}$. This approach has been adopted for example in (Cesa-Bianchi and Lugosi,

2006, Chapter 11).
Minimizing the loss incurred if $(\mathbf{x}_t, y_t)$ is received again means that the forecaster is supposed to capitalize on the information received, in order to update its parameters vector $\mathbf{w}_{t-1}$, so that if $(\mathbf{x}_t, y_t)$ is presented again on the next step, it will incur a decreased loss. On the other hand, classifier needs also to preserve previous learning, represented by the old $\mathbf{w}_{t-1}$; therefore it has also to minimize the change in the weight vector, as measured by $d(\mathbf{w}_t, \mathbf{w}_{t-1})$. For what we have said we can formalize the above mentioned on-line minimization problem with the following *objective function*:

$$\mathbf{w}_t = \arg\min_{\mathbf{u}\in\mathbb{R}^d} \left(d(\mathbf{u}, \mathbf{w}_{t-1}) + \lambda \ell_t(\mathbf{u})\right) \tag{2.1.3}$$

where λ governs the relative importance of the loss function, compared with the distance function. This model has two main components:

1. the distance function d
2. the loss function ℓ_t

depending on their choice we will obtain different prediction algorithms.

2.1.2 Convex optimization

In this Subsection we will try to analyze some specific instances of the previous general model, for the classification task and we will show also how Perceptron update rule rises up naturally in this framework. We start by stating a basic result from convex analysis which will serve as a pedestal for all the subsequent theory.

2.1.5 Optimality conditions for convex functions. *If a function $f : \mathbb{R}^d \to \mathbb{R}$ is convex and differentiable, a necessary and sufficient condition for a point $\mathbf{w}$ to be a local minimum of f is that $\nabla f(\mathbf{w}) = \mathbf{0}$. Moreover if such a minimum exists, it is a global minimum (or the function is stationary).*

As a consequence, if both the loss function $\ell_t(\mathbf{u}_{t-1})$ and the distance function d are convex and differentiable with respect to the first argument $\mathbf{u} \in \mathbb{R}^d$, we can look for a global minimum of the objective function 2.1.3, by setting:

$$\nabla \left(d(\mathbf{u}, \mathbf{w}_{t-1}) + \lambda \ell_t(\mathbf{u})\right) = \mathbf{0} \tag{2.1.4}$$

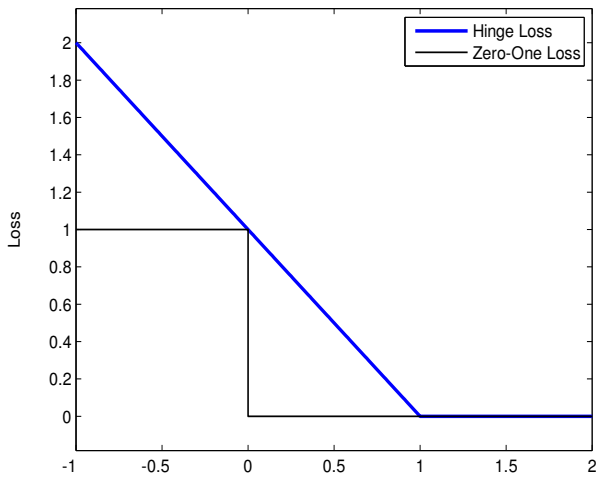


Figure 2.4: The hinge loss (shown in blue) is convex and upperbounds the zero-one loss (in black)

As before we are faced the choice of the premetric and the loss function, but we now restrict our attention to convex and differentiable functions that allow us to use (2.1.4). We start by introducing two loss functions, specific for the binary classification task.

Loss functions for binary classification Many kinds of loss functions are used to address machine learning problems and as a matter of facts the choice of the appropriate loss function for the specific task is one of the most important factors for the success of a learning procedure. In a binary classification task, the forecaster incurs some loss if the signum of its prediction is different from the signum of the outcome. A natural choice for the loss function could then be $\ell_t : \mathbb{R}^d \to \mathbb{R}^+$:

$$\ell_t(\mathbf{u}) \stackrel{def}{=} \begin{cases} 1 & \text{if } y_t\langle \mathbf{u}, \mathbf{x}_t\rangle \leq 0 \\ 0 & \text{otherwise.} \end{cases} \tag{2.1.5}$$

This loss is called *Zero-one Loss*, it is not convex and its derivative is always zero, apart in the point $\langle \mathbf{u}, \mathbf{x}_t\rangle = 0$, where it is not differentiable. Therefore it cannot be minimized by means of Theorem 2.1.5. For this reason, it is usually replaced by another loss function, which is convex and differentiable

and upperbounds the above-mentioned Loss:

$$\ell_t(\mathbf{u}) \stackrel{def}{=} (1 - y_t\langle \mathbf{u}, \mathbf{x}_t\rangle\})_+ = \max\{0, 1 - y_t\langle \mathbf{u}, \mathbf{x}_t\rangle\} \tag{2.1.6}$$

This is the so-called *Hinge Loss*. Technically, this function is not differentiable in the point $\langle \mathbf{u}, \mathbf{x}_t\rangle = 1/y_t = y_t$, however since we are dealing with a conservative update policy we need to update weights in an optimal way only when $\hat{y}_t \neq y_t$. Therefore this loss function has to be minimized only when $\hat{y}_t = \text{sign}\left(\langle \mathbf{u}, \mathbf{x}_t\rangle\right) \neq y_t$ and we may set:

$$\nabla \ell_t(\mathbf{u}) = \nabla \left(1 - y_t\langle \mathbf{u}, \mathbf{x}_t\rangle\right)_+ = -y_t \mathbf{x}_t \, \mathbb{I}_{\{\hat{y}_t \neq y_t\}} \tag{2.1.7}$$

Now that we have introduced a convex and differentiable loss function suitable for our minimization task, we can concentrate our attention to the premetric d, to be minimized. We will of course look for convex and differentiable premetric.

Norm optimization In $\mathbb{R}^d$ a natural choice for d would be $d(\mathbf{u}, \mathbf{x}) = \|\mathbf{u} - \mathbf{x}\| = \sqrt{\langle \mathbf{u} - \mathbf{x}, \mathbf{u} - \mathbf{x}\rangle}$. This function is clearly a premetric and it is convex, because of triangle inequality ($\|\mathbf{u} + \mathbf{x}\| \leq \|\mathbf{u}\| + \|\mathbf{x}\|$). However, instead of minimizing the norm it is often more convenient to minimize $\frac{1}{2}\|\mathbf{u} - \mathbf{x}\|^2$. If we define $f(\mathbf{x}) = \|\mathbf{x}\|$ and $g(x) = \frac{1}{2}x^2$, $g : \mathbb{R}^+ \to \mathbb{R}$ and $\lambda \in (0, 1)$ we see that

$$\begin{aligned}
\frac{1}{2}\|\lambda \mathbf{x}_1 + (1-\lambda)\mathbf{x}_2\|^2 &= g(f(\lambda \mathbf{x}_1 + (1-\lambda)\mathbf{x}_2)) \\
&\leq g(\lambda f(\mathbf{x}_1) + (1-\lambda) f(\mathbf{x}_2)) \\
&\leq \lambda g(f(\mathbf{x}_1)) + (1-\lambda) g(f(\mathbf{x}_2)) \\
&= \lambda \frac{1}{2}\|\mathbf{x}_1\|^2 + (1-\lambda)\frac{1}{2}\|\mathbf{x}_2\|^2
\end{aligned} \tag{2.1.8}$$

where for the first inequality we have exploited the convexity of f and the fact that g is a non-decreasing function, while the second inequality is due to the convexity of g. Therefore also $\frac{1}{2}\|\mathbf{u}\|^2$ is a convex function. Moreover it is also a differentiable function and its gradient is given by:

$$\nabla \frac{1}{2}\|\mathbf{u}\|^2 = \frac{2}{2}\|\mathbf{u}\|\nabla\|\mathbf{u}\| = \|\mathbf{u}\|\frac{\mathbf{u}}{\|\mathbf{u}\|} = \mathbf{u} \tag{2.1.9}$$

Finally it is also easy to verify that $d(\mathbf{u}, \mathbf{x}) = \frac{1}{2}\|\mathbf{u} - \mathbf{x}\|^2$ is a premetric with the same minimum point as $\|\mathbf{u} - \mathbf{x}\|$ (namely the point $\mathbf{u} = \mathbf{x}$). This premetric is therefore suitable for our minimization task.

At this point, we have introduced a convex and differentiable loss function and a convex and differentiable premetric function, suitable for our learning problem. Therefore, by Theorem 2.1.5 we can compute:

$$\begin{aligned}\nabla\left(\frac{1}{2}\|\mathbf{u} - \mathbf{w}_{t-1}\|^2 + \lambda\ell_t(\mathbf{u})\right) &= \nabla\left(\frac{1}{2}\|\mathbf{u} - \mathbf{w}_{t-1}\|^2\right) + \lambda\nabla\ell_t(\mathbf{u}) \\ &= \mathbf{u} - \mathbf{w}_{t-1} + \lambda\nabla\ell_t(\mathbf{u})\end{aligned}$$

and set it equal to zero, to obtain:

$$\mathbf{u} = \mathbf{w}_{t-1} - \lambda\nabla\ell_t(\mathbf{u}) \tag{2.1.10}$$

This is not a closed-form solution, but by substituting $\ell_t(\mathbf{u})$ with its first-order Taylor approximation $\ell_t(\mathbf{w}_{t-1}) + (\mathbf{u} - \mathbf{w}_{t-1})'\nabla\ell_t(\mathbf{w}_{t-1})$, around $\mathbf{w}_{t-1}$ into the original minimization problem we have:

$$\begin{aligned}\nabla\left(\frac{1}{2}\|\mathbf{u} - \mathbf{w}_{t-1}\|^2 + \lambda\left(\ell_t(\mathbf{w}_{t-1}) + (\mathbf{u} - \mathbf{w}_{t-1})'\nabla\ell_t\left(\mathbf{w}_{t-1}\right)\right)\right) = \\ \mathbf{u} - \mathbf{w}_{t-1} + \lambda\nabla\ell_t(\mathbf{w}_{t-1})\end{aligned}$$

which gives the approximated closed solution:

$$\mathbf{u} = \mathbf{w}_{t-1} - \lambda\nabla\ell_t(\mathbf{w}_{t-1}) \tag{2.1.11}$$

This update rule corresponds to the familiar *stochastic gradient descent* update rule in $\mathbb{R}^d$, with respect to the function $\ell_t(\mathbf{w})$. Thus the stochastic gradient descent algorithm arises naturally as an approximate solution for our general decision problem, when the loss function is convex and differentiable in its first argument and the premetric is defined to be half of the squared norm.

If we now further specify this result for the Hinge-Loss, we obtain:

$$\begin{aligned}\mathbf{w}_t \overset{def}{=} \mathbf{u} &= \mathbf{w}_{t-1} - \lambda\nabla\ell_t(\mathbf{w}_{t-1}) \\ &= \mathbf{w}_{t-1} - \lambda\nabla\left(1 - y_t\langle\mathbf{w}_{t-1}, \mathbf{x}_t\rangle\right)_+ \\ &= \mathbf{w}_{t-1} + \lambda y_t\mathbf{x}_t\,\mathbb{I}_{\{\hat{y}_t \neq y_t\}}\end{aligned}$$

which, for $\lambda = 1$ is exactly Perceptron update rule.

2.1.3 Bregman divergence optimization

We now introduce some useful tools (Cesa-Bianchi and Lugosi, 2006, Chapter 11) for the definition and analysis of a larger class of linear classification algorithms, including Perceptron as a special case.

2.1.6 Legendre function. A function $F : \mathcal{A} \to \mathbb{R}$ such that

- $\mathcal{A} \subseteq \mathbb{R}^n$ is nonempty and its interior $\text{int}(\mathcal{A})$ is convex
- F is strictly convex with continuous first partial derivatives throughout $\text{int}(\mathcal{A})$
- if $\mathbf{x}_1, \mathbf{x}_2, ... \in \mathsf{A}$ is a sequence converging to a boundary point of $\mathcal{A}$, then $\|\nabla F(\mathbf{x}_n)\| \xrightarrow[n\to\infty]{} \infty$

is said to be a *Legendre function.*

2.1.7 Bregman divergence. Given a Legendre function $F : \mathcal{A} \to \mathbb{R}$, the non-negative function $D_F : \text{int}(\mathcal{A}) \to \mathbb{R}^+$ defined by:

$$D_F(\mathbf{u}, \mathbf{v}) = F(\mathbf{u}) - F(\mathbf{v}) - \langle \nabla F(\mathbf{v}), (\mathbf{u} - \mathbf{v}) \rangle \tag{2.1.12}$$

is called *Bregman divergence* induced by F.

It is easy to see that Bregman divergence is nothing but the difference between $F(\mathbf{u})$ and its first-order Taylor expansion around $\mathbf{v}$. This divergence is not a metric because it is not symmetric ($D_F(\mathbf{u}, \mathbf{v}) \neq D_F(\mathbf{v}, \mathbf{u})$) and it does not satisfy the triangle inequality. However since F is convex, $D_F(\mathbf{u}, \mathbf{v}) \geq 0$ and it is easy to verify that $D_F(\mathbf{u}, \mathbf{u}) = 0$, it is a premetric.

2.1.8 Theorem. *Bregman divergence is convex and differentiable in its first argument.*

Proof. We now fix $\mathbf{v}$ and consider $D_F(\mathbf{u}, \mathbf{v})$ only as a function of its first argument, so we can write:

$$D_{F_\mathbf{v}}(\mathbf{u}) = F(\mathbf{u}) - \langle \mathbf{u}, \nabla F(\mathbf{v}) \rangle - c$$

where $c = F(\mathbf{v}) - \langle \nabla F(\mathbf{v}), \mathbf{v} \rangle$. Moreover, since F is convex, for every $0 < \lambda < 1$ we have that:

$$\begin{aligned}
D_{F_\mathbf{v}}(\lambda \mathbf{u}_1 + (1-\lambda)\mathbf{u}_2) &= F(\lambda \mathbf{u}_1 + (1-\lambda)\mathbf{u}_2) - \langle \lambda \mathbf{u}_1 + (1-\lambda)\mathbf{u}_2, \nabla F(\mathbf{v}) \rangle - c \\
&\leq \lambda F(\mathbf{u}_1) + (1-\lambda) F(\mathbf{u}_2) - \lambda \langle \mathbf{u}_1, \nabla F(\mathbf{v}) \rangle - \\
&- (1-\lambda) \langle \mathbf{u}_2, \nabla F(\mathbf{v}) \rangle - c \\
&= \lambda \left(F(\mathbf{u}_1) - \langle \mathbf{u}_1, \nabla F(\mathbf{v}) \rangle \right) + \\
&+ (1-\lambda) \left(F(\mathbf{u}_2) - \langle \mathbf{u}_2, \nabla F(\mathbf{v}) \rangle \right) - c \\
&= \lambda \left(D_{F_\mathbf{v}}(\mathbf{u}_1) + c \right) + (1-\lambda) \left(D_{F_\mathbf{v}}(\mathbf{u}_2) + c \right) - c \\
&= \lambda \left(D_{F_\mathbf{v}}(\mathbf{u}_1) \right) + (1-\lambda) \left(D_{F_\mathbf{v}}(\mathbf{u}_2) \right) + \lambda c + (1-\lambda) c - c \\
&= \lambda \left(D_{F_\mathbf{v}}(\mathbf{u}_1) \right) + (1-\lambda) \left(D_{F_\mathbf{v}}(\mathbf{u}_2) \right)
\end{aligned}$$

Finally, since F is differentiable, we have also that:

$$\nabla D_{F_\mathbf{v}}(\mathbf{u}) = \nabla \left(F(\mathbf{u}) - \langle \mathbf{u}, \nabla F(\mathbf{v}) \rangle - c \right) = \nabla F(\mathbf{u}) - \nabla F(\mathbf{v})$$

□

We now state without proving the last important result from convex analysis.

2.1.9 Legendre dual. *Given a Legendre function $F : \mathcal{A} \to \mathbb{R}$, its Legendre dual is the Legendre function F^* defined as:*

$$F^*(\mathbf{u}) = \sup_{\mathbf{v} \in \mathcal{A}} \left(\langle \mathbf{u}, \mathbf{v} \rangle - F(\mathbf{v}) \right)$$

moreover, the Legendre dual F^{**} of F^* equals F and $\nabla F^* = (\nabla F)^{-1}$.

2.1.10 Example: p-norm. As an important example, the Legendre dual of $\Phi(\mathbf{u}) = \frac{1}{2}\|\mathbf{u}\|_p^2$, $p \geq 2$ is $\Phi^*(\mathbf{u}) = \frac{1}{2}\|\mathbf{u}\|_q^2$, where p and q are *conjugate exponents*: $\frac{1}{p} + \frac{1}{q} = 1$. Moreover its gradient is given by:

$$\nabla \Phi(\mathbf{u})_i = \left(\nabla {}^1\!/_2 \|\mathbf{u}\|_p^2 \right)_i = \frac{\operatorname{sign}(u_i)|u_i|^{p-1}}{\|\mathbf{u}\|_p^{p-2}}$$

$$\nabla \Phi^*(\mathbf{v})_i = \left(\nabla {}^1\!/_2 \|\mathbf{v}\|_q^2 \right)_i = \frac{\operatorname{sign}(v_i)|v_i|^{q-1}}{\|\mathbf{v}\|_q^{q-2}}$$

for every $\mathbf{u}, \mathbf{v} \neq \mathbf{0}$. In agreement with Theorem 2.1.9, it is also possible to verify that:

$$\nabla \Phi \left(\nabla \Phi^*(\mathbf{v}) \right) = \mathbf{v}$$

Gradient descent, again By supposing the loss function to be convex and differentiable in $\mathbb{R}^d$ and by choosing $d(\mathbf{u}, \mathbf{w}_{t-1}) = D_\Phi(\mathbf{u}, \mathbf{w}_{t-1})$, where Φ is a Legendre function called *Potential function*, we can restate the minimization problem as follow:

$$\mathbf{w}_t = \arg\min_{\mathbf{u} \in \mathbb{R}^N} \left(D_\Phi(\mathbf{u}, \mathbf{w}_{t-1}) + \lambda \ell_t(\mathbf{u}) \right) \tag{2.1.13}$$

so that by setting its gradient to zero we obtain

$$\begin{aligned}
\nabla \left(\Phi(\mathbf{u}) - \Phi(\mathbf{w}_{t-1}) - (\mathbf{u} - \mathbf{w}_{t-1})' \nabla \Phi(\mathbf{w}_{t-1}) + \lambda \ell_t(\mathbf{u}) \right) &= \mathbf{0} \\
\nabla \Phi(\mathbf{u}) - \nabla \Phi(\mathbf{w}_{t-1}) + \lambda \nabla \ell_t(\mathbf{u}) &= \mathbf{0} \\
\nabla \Phi(\mathbf{u}) &= \nabla \Phi(\mathbf{w}_{t-1}) - \lambda \nabla \ell_t(\mathbf{u}) \\
\mathbf{u} &= \nabla \Phi^* \left(\nabla \Phi(\mathbf{w}_{t-1}) - \lambda \nabla \ell_t(\mathbf{u}) \right)
\end{aligned}$$

where the last equation is due to Theorem 2.1.9.
Again, this is not a closed form solution, but by substituting ℓ_t with its first-order Taylor approximation around $\mathbf{w}_{t-1}$ we obtain the closed form solution:

$$\mathbf{w}_t = \nabla \Phi^* \left(\nabla \Phi(\mathbf{w}_{t-1}) - \lambda \nabla \ell_t(\mathbf{w}_{t-1}) \right) \tag{2.1.14}$$

Moreover, if we specialize this result for the Hinge loss and arbitrarily fix $\nabla \Phi(\mathbf{w}_0) = 0$ we can write a generic sequence of weights as follow:

$$\begin{aligned}
\mathbf{w}_1 &= \nabla \Phi^* \left(\nabla \Phi(\mathbf{w}_0) - \lambda \nabla \ell_1(\mathbf{w}_0) \right) = \nabla \Phi^* \left(\lambda y_1 \mathbf{x}_1 \mathbb{I}_{\{\hat{y}_1 \neq y_1\}} \right) \\
\mathbf{w}_2 &= \nabla \Phi^* \left(\nabla \Phi \left(\nabla \Phi^* \left(\lambda y_1 \mathbf{x}_1 \right) \right) + \lambda y_2 \mathbf{x}_2 \right) = \nabla \Phi^* \left(\lambda y_1 \mathbf{x}_1 \mathbb{I}_{\{\hat{y}_1 \neq y_1} + \lambda y_2 \mathbf{x}_2 \mathbb{I}_{\{\hat{y}_2 \neq y_2\}} \right) \\
&\vdots \\
\mathbf{w}_T &= \nabla \Phi^* \left(\lambda \sum_{t=1}^{T} y_t \mathbf{x}_t \mathbb{I}_{\{\hat{y}_t \neq y_t\}} \right)
\end{aligned}$$

therefore we can split the update rule into two steps:

$$\begin{aligned}
\mathbf{v}_t &= \mathbf{v}_{t-1} + \lambda y_t \mathbf{x}_t \mathbb{I}_{\{\hat{y}_t \neq y_t\}} \\
\mathbf{w}_t &= \nabla \Phi^*(\mathbf{v}_t)
\end{aligned}$$

where we call $\mathbf{v}_t$ the *primal weight*, $\mathbf{w}_t$ the *dual weight* and we have supposed $\mathbf{v}_0 = \nabla \Phi(\mathbf{w}_0) = \mathbf{0}$. The resulting conservative potential-based binary classification algorithm (for hinge loss) is shown in Figure 2.5.

Assume $\mathbf{x}_t \in \mathcal{X} \subseteq \mathbb{R}^d$, $\mathbf{w}_t \in \mathbb{R}^d$ and Φ is a Legendre function.
Initialization $\mathbf{v}_0 = \mathbf{0}$, $\mathbf{w}_0 = \nabla\Phi(\mathbf{v}_0)$.
At each time t=1,2,..... do the following:

1: Observe instance $\mathbf{x}_t \in \mathcal{X}$;
2: Predict label $y_t \in \{-1, 1\}$ with $\hat{y}_t = \text{sign}(\langle \mathbf{w}_{t-1}, \mathbf{x}_t \rangle)$;
3: Observe actual label $y_t \in \{-1, 1\}$;
4: Update the weight vectors with:
$\mathbf{v}_t = \mathbf{v}_{t-1} + \lambda y_t \mathbf{x}_t \mathbb{I}_{\{\hat{y}_t \neq y_t\}} \quad \mathbf{w}_t = \nabla\Phi^*(\mathbf{v}_t)$

Figure 2.5: Conservative potential-based gradient descent algorithm, for the hinge loss

Example: p-Norm algorithm If we set $\lambda = 1$ and $\Phi(\mathbf{u}) = \frac{1}{2}\|\mathbf{u}\|_p^2$ (as in example 2.1.10) the algorithm shown in Figure 2.5 becomes the *p-Norm algorithm* which has Perceptron as a special case, when $p = 2$. The main theoretical result for this class of algorithms is stated below (Gentile, 2003):

2.1.11 p-Norm proof of convergence. *Let* $\mathbf{u} \in \mathbb{R}$ *be an arbitrary comparison vector. Then the number of mistakes* m *made by the p-norm algorithm run on any finite sequence of examples* $(\mathbf{x_1}, y_1), ..(\mathbf{x_n}, y_n) \in \mathbb{R} \times \{-1, +1\}$ *satisfies:*

$$m \leq L_n(\mathbf{u}) + (p-1)\left(X_p\|\mathbf{u}\|_q\right)^2 + X_p\|\mathbf{u}\|_q\sqrt{(p-1)L_n(\mathbf{u})} \tag{2.1.15}$$

where $X_p = max_{t=1,\ldots,n}\|\mathbf{x}_t\|_p$ *and* $L_n(\mathbf{u}) = \sum_{t=1}^{n}\left(1 - y_t\langle \mathbf{u}, \mathbf{x}_t\rangle\right)_+$

Note that this is a relative bound because it employs the notion of "comparison vector", this vector can be any other vector, so it can also be the optimal vector. Note also that if the problem is linearly separable with margin 1, the cumulative loss $L_n(\mathbf{u})$ is zero because there exists a an optimal separating vector $\mathbf{u}$ that always classifies correctly, so that the bound reduces to:

$$m \leq (p-1)\left(X_p\|\mathbf{u}\|_q\right)^2 \tag{2.1.16}$$

2.2 Kernel Methods

As already underlined in the previous section, linear-threshold algorithms can only learn linearly separable relations. However it is often the case that data is not linearly separable by an hyperplane, but a separating hyperplane could still be found by pre-mapping the original vectors $\mathbf{x}_t \in \mathcal{X}$ into a new space $\phi(\mathbf{x}_t) \in F$ called the *Feature Space* (see Figure 2.6).

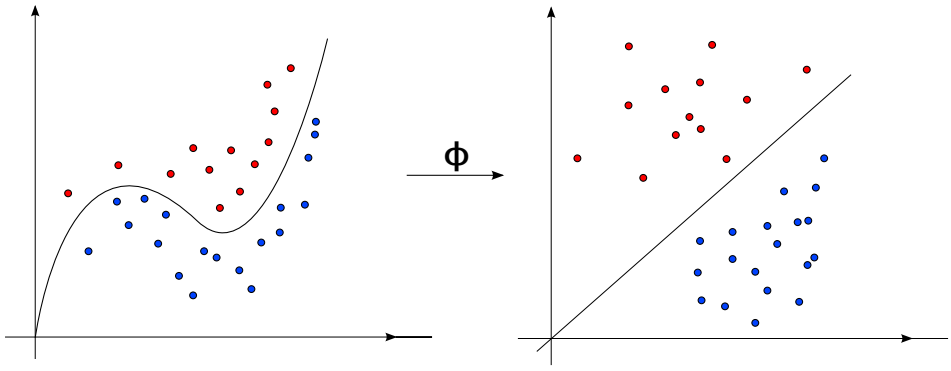


Figure 2.6: ϕ maps the non-linearly separable dataset into a new space where it is linearly separable

In order to achieve linear separability it is often necessary to non-linearly map the original data into a Feature space of higher dimensionality, nevertheless in so doing we can incur the so called *"curse of dimensionality"* problem: algorithms complexity tends to grow exponentially with the growth of dimensionality and if the number of parameters is too large with respect to the number of training samples there could arise also the problem of overfitting.
One way to face these problems comes from kernel theory, that we are going to present in this section[1].

2.2.1 Dual formulation

Many learning algorithms can be re-cast into a *dual formulation* that depends on data only via the inner product. Those algorithms enable us to

[1]For a full treatment of the subject refer to (Scholkopf, 2002) and (Shawe-Taylor and Cristianini, 2004)

substitute the inner product with a kernel function. A kernel function is a function that can be decomposed into a feature map ϕ into an Hilbert space F, applied to both arguments and followed by the evaluation of the inner product in F.

2.2.1 Kernel function. A function $k(\cdot,\cdot)$ that for all $\mathbf{x},\mathbf{z} \in \mathcal{X}$ satisfies:

$$k(\mathbf{x},\mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle \tag{2.2.1}$$

where ϕ is a mapping from $\mathcal{X}$ to an Hilbert space F

$$\phi : \mathbf{x} \ \rightarrow \ \phi(\mathbf{x}) \in F$$

is called *kernel function.*

If $\mathcal{X}$ is an Hilbert space, the simplest example of kernel function is the one obtained considering the identity mapping $\phi(\mathbf{x}) = \mathbf{x}$, in which case $k(\mathbf{x},\mathbf{z}) = \langle \mathbf{x},\mathbf{z} \rangle$. Another simple example is given by the kernel function:

$$\begin{aligned} k(\mathbf{x},\mathbf{z}) &= \langle \mathbf{x},\mathbf{z} \rangle^2 = (x_1 z_1 + x_2 z_2)^2 = x_1^2 z_1^2 + 2 x_1 z_1 x_2 z_2 + x_2^2 z_2^2 \\ &= \left\langle \left(x_1^2, \sqrt{2} x_1 x_2, x_2^2 \right), \left(z_1^2, \sqrt{2} z_1 z_2, z_2^2 \right) \right\rangle = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle \end{aligned}$$

where $\mathbf{x},\mathbf{z} \in \mathbb{R}^2$ and $\phi(\mathbf{x}) = \left(x_1^2, \sqrt{2} x_1 x_2, x_2^2 \right) \in \mathbb{R}^3$.

Even if in the above example we can explicitly deal with the components of the feature vectors, in general this is not possible. This limit is due to the fact that there are many possible feature mappings associated with a single kernel function and, as we will see in the section 2.2.2, kernel functions can also be created without any explicit reference to the feature map. For this reason we usually do not have access to the components of the images $\Phi(\mathbf{x}_i)$ of the instance vectors, but only to the evaluation of inner products $\langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle$ between them. However it is still possible to get many important informations about $\Phi(\mathbf{x})$. As an example we can easily compute the norm of the image of an instance vector.

2.2.2 Norm of feature vectors. If $\mathbf{x} \in \mathcal{X}$ is an instance vector and $k(\mathbf{x}_i,\mathbf{x}_j) = \langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle$, we can compute $\|\Phi(\mathbf{x})\|_2$ in the following way:

$$\|\Phi(\mathbf{x})\|_2 = \sqrt{\|\Phi(\mathbf{x})\|_2^2} = \sqrt{\langle \Phi(\mathbf{x}), \Phi(\mathbf{x}) \rangle} = \sqrt{k(\mathbf{x},\mathbf{x})} \tag{2.2.3}$$

If $\hat{\Phi}(\mathbf{x}) = \frac{\Phi(\mathbf{x})}{\|\Phi(\mathbf{x})\|_2}$ is the normalized image of a vector $\mathbf{x} \in \mathcal{X}$, we can also obtain the normalized kernel $\hat{k}(\mathbf{x}_i, \mathbf{x}_j)$ in the following way

$$\hat{k}(\mathbf{x}_i, \mathbf{x}_j) = \langle \hat{\Phi}(\mathbf{x}_i), \hat{\Phi}(\mathbf{x}_j) \rangle = \left\langle \frac{\Phi(\mathbf{x}_i)}{\|\Phi(\mathbf{x}_i)\|_2}, \frac{\Phi(\mathbf{x}_j)}{\|\Phi(\mathbf{x}_j)\|_2} \right\rangle = \frac{\langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle}{\|\Phi(\mathbf{x}_i)\|_2 \|\Phi(\mathbf{x}_j)\|_2}$$
$$= \frac{k(\mathbf{x}_i, \mathbf{x}_j)}{\sqrt{k(\mathbf{x}_i, \mathbf{x}_i) k(\mathbf{x}_j, \mathbf{x}_j)}} \tag{2.2.4}$$

This function is by construction a kernel function.

2.2.3 Kernel as an Oracle. A kernel function can be thought as a similarity measure between two inputs, as an oracle guessing the relatedness of two data points. The choice of a kernel function may therefore reflect our prior knowledge concerning the instance space $\mathcal{X}$ and should capture our prior belief on the similarity between different examples.

2.2.4 "Kernel Trick". Given an algorithm formulated in such a way that it depends on instances only through their inner product, it is possible to construct an alternative algorithm by replacing the inner products with a kernel function. The algorithm is then said to be *Kernelizable*.

Advantages If an algorithm can be kernelized and if the kernel function is properly chosen, instances $\mathbf{x}_1, \mathbf{x}_2, ...$ are a-priori mapped into a feature space of higher (even infinite) dimensionality where they are possibly linearly separable, or better separable. The advantages of this procedure are that:

- there is no need to compute anything in the higher dimensional space (we do not have access to the coordinates of the higher-dimensional *Feature Space*)
- the number of parameters to be estimated becomes independent of the dimensionality of the Feature Space
- it is possible to rely on well known linear threshold theory and algorithms with convergence proofs

This results in a grown modularity of our previously introduced on-line classification model, that can now be seen as composed of three main elements:

1. the kernel function $k(\mathbf{x}_i, \mathbf{x}_j)$

Assume $\mathbf{x}_t, \mathbf{w_0} \in \mathcal{X} \subseteq \mathbb{R}^d$ and S is an index set.
Initialization $\mathbf{w_0} = \mathbf{0}$, $S = \varnothing$.
At each time t=1,2,..... do the following:

1: Observe instance $\mathbf{x}_t \in \mathcal{X}$;
2: Predict label $y_t \in \{-1, 1\}$ with $\hat{y}_t = \text{sign}\left(\sum_{s \in S} y_s k(\mathbf{x}_s, \mathbf{x}_t)\right)$;
3: Observe actual label $y_t \in \{-1, 1\}$;
4: **if** $\hat{y}_t \neq y_t$ **then**
5: $\quad S = S \bigcup \{t\}$
6: **end if**

Figure 2.7: Perceptron Dual formulation

2. the distance function $d_{\mathbf{w}}$

3. the loss function ℓ_t

A simple example of a kernelizable algorithm is given again by the Perceptron linear-threshold classifier. If we call $S = \{s_1, s_2, ..., s_n\}$ the set of indexes of missclassified training examples, where $n = |S|$, we can re-express the weight vector in the following form:

$$\mathbf{w}_t = \mathbf{0} + y_{s_1}\mathbf{x}_{s_1} + y_{s_2}\mathbf{x}_{s_2} + \cdots + y_{s_n}\mathbf{x}_{s_n} = \sum_{s \in S} y_s \mathbf{x}_s \tag{2.2.5}$$

The weight vector is thus a linear combination of the missclassified instances that, in analogy to Support Vector Machines, are called *support vectors*. If we now take into account the prediction phase we see that:

$$\langle \mathbf{w}_t, \mathbf{x}_t \rangle = \left\langle \sum_{s \in S} y_s \mathbf{x}_s \ , \ \mathbf{x}_t \right\rangle = \sum_{s \in S} y_s \langle \mathbf{x}_s, \mathbf{x}_t \rangle \tag{2.2.6a}$$

by the linearity of the inner product. So that:

$$\hat{y}_t = \text{sign}\left(\sum_{S} y_s \langle \mathbf{x}_s, \mathbf{x}_t \rangle \right) \tag{2.2.6b}$$

The dual-formulated algorithm clearly depends on data only via the inner product $\langle \mathbf{x}, \mathbf{y} \rangle$ and it is therefore possible to replace the inner product with a kernel function, as shown in Figure 2.7.

It is worth noting, however, that not all algorithms can be kernelized, for example, the conservative potential-based algorithm shown in Figure 2.5 cannot be kernelized for all potentials. This is due to the fact that $\mathbf{w}_t = \nabla\Phi\left(\sum_{s\in S} y_s\mathbf{x}_s\right)$ and it is not possible to express the algorithm only in terms of inner product between instances:

$$\langle \mathbf{w}_t, \mathbf{x}_t \rangle = \left\langle \nabla\Phi\left(\sum_{s\in S} y_s\mathbf{x}_s\right) \ , \ \mathbf{x}_t \right\rangle$$

If we would substitute the above inner product with a kernel function we would break the assumption that all instances are a-priori mapped into the same feature space. Of course, if we choose $\Phi(\mathbf{x}) = \frac{1}{2}\|\mathbf{x}\|^2$ we get back the classical Perceptron update rule that, for what we have previously said, can be kernelized.

2.2.2 Reproducing Kernel Hilbert Space theory

Once we have defined what a kernel function is and what algorithms can exploit its definition, the next logical questions are: *what functions are kernel functions? How can we construct them?* In order to answer these questions we have to introduce some more theory.
We start by the the definition of *finitely positive semi-definite* function which, as we will see is the characterizing property of kernels.

2.2.5 Finitely positive semi-definite function. Let $\mathcal{X}$ be a metric space, we say that a function:

$$k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$$

is *finitely positive semi-definite* if it is a symmetric function and for all finite sets $\widetilde{x} \subset \mathcal{X}$, of size n, the $n \times n$ matrix $\mathbf{K}$ whose (i, j) entry is $k(\mathbf{x}_i, \mathbf{x}_j)$ is positive semi-definite, formally:

$$\mathbf{v}'\mathbf{K}\mathbf{v} \geq 0 \qquad \forall\, \mathbf{v} \in \mathbb{R}^n \tag{2.2.7}$$

The above mentioned matrix $\mathbf{K}$ is called Gramian matrix of k at $\widetilde{x}$.

We now state and fully prove the main result of the *Reproducing Kernel Hilbert Space* theory: the characterization Theorem of kernel functions.

2.2.6 Characterization of kernels. *A function*

$$k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$$

is a kernel function $k(\mathbf{x}, \mathbf{z}) = \langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle$ *if and only if it satisfies the finitely positive semi-definite property.*

Proof. Let the $\mathbf{K}$ be the Gramian matrix of a kernel function $k(\mathbf{x}_i, \mathbf{x}_j)$, for $i, j = 1, ..., n$. For any vector $\mathbf{v}$ we have

$$\begin{aligned}\mathbf{v}'\mathbf{K}\mathbf{v} &= \sum_{i,j=1}^{n} v_i v_j k(\mathbf{x}_i, \mathbf{x}_j) = \sum_{i,j=1}^{n} v_i v_j \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle \\ &= \left\langle \sum_{i=1}^{n} v_i \phi(\mathbf{x}_i), \sum_{j=1}^{n} v_j \phi(\mathbf{x}_j) \right\rangle = \left\| \sum_{i=1}^{n} v_i \phi(\mathbf{x}_i) \right\|^2 \geq 0\end{aligned}$$

This proves the 'only if' implication: "*every kernel function satisfies the finite positive semi-definite property*". In order to prove the 'if' part we will assume that k satisfies the finite positive semi-definite property and explicitly construct a feature space associated to it. Let F_0 be the space of functions:

$$F_0 = \left\{ \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_i, \cdot) : \mathbf{x}_i \in \mathcal{X}, \alpha_i \in \mathbb{R} \right\} \tag{2.2.8}$$

This space is closed under multiplication by a scalar and addition of functions. Indeed if $f, g, h \in \mathcal{F}$ are given by:

$$f(\cdot) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_i, \cdot) \qquad g(\cdot) = \sum_{j=1}^{m} \beta_j k(\mathbf{z}_j, \cdot) \qquad \text{and} \qquad h(\cdot) = \sum_{l=1}^{s} \gamma_l k(\mathbf{v}_l, \cdot)$$

we have that:

$$(\lambda f)(\mathbf{x}) \stackrel{def}{=} \lambda f(\mathbf{x}) = \sum_{i=1}^{n} (\lambda \alpha_i) k(\mathbf{x}_i, \cdot)$$

$$(f + g)(\mathbf{x}) \stackrel{def}{=} f(\mathbf{x}) + g(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}_i, \mathbf{x}) + \sum_{j=1}^{m} \beta_j k(\mathbf{z}_j, \mathbf{x}) = \sum_{h=1}^{n+m} \xi_h \, k(\mathbf{y}_h, \mathbf{x})$$

where:

$$\xi_h = \begin{cases} \alpha_h & \text{for } h = 1, ..., n \\ \beta_{h-n} & \text{for } h = n+1, ..., n+m. \end{cases} \qquad \mathbf{y}_h = \begin{cases} \mathbf{x}_h & \text{for } h = 1, ..., n \\ \mathbf{z}_{h-n} & \text{for } h = n+1, ..., n+m. \end{cases}$$

We can also introduce an inner product in it as follows:

$$\langle f, g\rangle = \sum_{i=1}^{n}\sum_{j=1}^{m} \alpha_i \beta_j k(\mathbf{x}_i, \mathbf{z}_j) \tag{2.2.9}$$

$$= \sum_{i=1}^{n} \alpha_i g(\mathbf{x}_i) = \sum_{j=1}^{m} \beta_j f(\mathbf{z}_j) \tag{2.2.10}$$

$\langle f, g\rangle$ is real-valued, symmetric (by the finite positive semi-definite property), bilinear and satisfies:

$$\langle f, f\rangle = \sum_{i,j=1}^{n} \alpha_i \alpha_j k(\mathbf{x}_i, \mathbf{x}_j) = \alpha' \mathbf{K} \alpha \geq 0$$

$$\langle f+g, h\rangle = \sum_{h=1}^{n+m}\sum_{l=1}^{s} \xi_h \gamma_l \, k(\mathbf{y}_h, \mathbf{v}_l)$$

$$= \sum_{i=1}^{n}\sum_{l=1}^{s} \alpha_i \gamma_l \, k(\mathbf{x}_i, \mathbf{v}_l) + \sum_{j=1}^{m}\sum_{l=1}^{s} \beta_j \gamma_l \, k(\mathbf{z}_j, \mathbf{v}_l)$$

$$= \langle f, h\rangle + \langle g, h\rangle$$

$$\langle f, k(\mathbf{x}, \cdot)\rangle = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}, \mathbf{x}_i) = f(\mathbf{x}) \text{ (}\textbf{Reproducing Property}\text{)}$$

$\langle f, f\rangle = 0 \leftrightarrow f = 0$ is easily demonstrated by noting that:

$$f = 0 \rightarrow \langle f, f\rangle = 0$$

$$0 \leq f(\mathbf{x})^2 = \langle f, k(\mathbf{x}, \cdot)\rangle^2 \leq \langle f, f\rangle k(\mathbf{x}, \mathbf{x})$$

so that $\langle f, f\rangle = 0 \rightarrow f = 0$

Hence $\langle f, g\rangle$ is a strict inner product turning F_0 into an inner product space and a metric space, with the metric d given by $d(f, g) = \|f - g\| = \sqrt{\langle f - g, f - g\rangle}$. At this point, in order to transform F_0 into an Hilbert space, we have to prove that every Cauchy sequence in F_0 has a pointwise limit and we have to include all those limit points into F.

In the metric space F, a Cauchy sequence is a sequence of functions $f_1, f_2, ...$ such that:

$$\lim_{n\to\infty} \sup_{m>n} \|f_m - f_n\| = 0 \tag{2.2.11}$$

We now fix $\mathbf{x} \in \mathbb{R}^n$ and consider the sequence of Real numbers $f_1(\mathbf{x}), f_2(\mathbf{x}), ...$ If $f_1, f_2, ...$ is a Cauchy sequence in F_0:

$$(f_m(\mathbf{x}) - f_n(\mathbf{x}))^2 = \langle f_m - f_n, k(\mathbf{x}, \cdot)\rangle^2 \leq \|f_m - f_n\|^2 k(\mathbf{x}, \mathbf{x})$$

and

$$\lim_{n\to\infty} \sup_{m>n} |f_m(\mathbf{x}) - f_n(\mathbf{x})| \leq \lim_{n\to\infty} \sup_{m>n} \|f_m - f_n\| \sqrt{k(\mathbf{x},\mathbf{x})} = 0$$

Where we have used the reproducing property and the Cauchy-Schwartz inequality. Thus for every $\mathbf{x}$, also $f_1(\mathbf{x}), f_2(\mathbf{x}), ...$ is a Cauchy sequence and by the completion of the Real numbers, it has a limit point. If we now define:

$$g(\mathbf{x}) = \lim_{n\to\infty} f_n(\mathbf{x}) \tag{2.2.12}$$

and we take F to be the completion of F_0 with respect to all such limit functions, we obtain an Hilbert space associated with kernel k. We call this Hilbert space the Reproducing Kernel Hilbert Space (*RKHS*).

Finally we choose ϕ to be defined as $\phi : \mathbf{x} \in \mathcal{X} \to \phi(\mathbf{x}) = k(\mathbf{x}, \cdot) \in F$ and we apply the reproducing property to show that

$$\langle \phi(\mathbf{x}), \phi(\mathbf{z}) \rangle = \langle k(\mathbf{x}, \cdot), k(\mathbf{z}, \cdot) \rangle = k(\mathbf{x}, \mathbf{z})$$

This concludes the proof of the "if" implication: "*every function that has the finite positive semi-definite property is a kernel function*". □

Finitely positive semi-definite property is thus the characterizing property of kernel functions and we can exploit this knowledge to prove many useful rules for manipulating kernels.

2.2.7 Closure properties of kernels. *Let k_1, k_2 be kernels over $\mathcal{X} \times \mathcal{X}$, $\mathcal{X} \subseteq \mathbb{R}^n$, $\alpha \in \mathbb{R}^+$, $f(\cdot)$ a real valued function on $\mathcal{X}$, $\phi :\to \mathbb{R}^N$ with k_3 a kernel over $\mathbb{R}^N \times \mathbb{R}^N$, and $\mathbf{B}$ a symmetric positive semi-definite $n \times n$ matrix. Then the following functions are kernels.*

$$k(\mathbf{x}, \mathbf{z}) = k_1(\mathbf{x}, \mathbf{z}) + k_2(\mathbf{x}, \mathbf{z}) \tag{2.2.13}$$

$$k(\mathbf{x}, \mathbf{z}) = \alpha k_1(\mathbf{x}, \mathbf{z}) \tag{2.2.14}$$

$$k(\mathbf{x}, \mathbf{z}) = k_1(\mathbf{x}, \mathbf{z}) k_2(\mathbf{x}, \mathbf{z}) \tag{2.2.15}$$

$$k(\mathbf{x}, \mathbf{z}) = f(\mathbf{x}) f(\mathbf{z}) \tag{2.2.16}$$

$$k(\mathbf{x}, \mathbf{z}) = k_3\left(\phi(\mathbf{x}), \phi(\mathbf{z})\right) \tag{2.2.17}$$

$$k(\mathbf{x}, \mathbf{z}) = \mathbf{x}'\mathbf{B}\mathbf{z} \tag{2.2.18}$$

Proof. Every of these rules is easily proved to produce a finitely positive semi-definite function (Shawe-Taylor and Cristianini, 2004), and thus a kernel function. □

Using the above properties we can now introduce some kernel functions.

2.2.8 Polynomial and exponential kernel. *Let* $\mathbf{x}, \mathbf{z} \in \mathcal{X}$ *be two instance vectors. If* $k1(\mathbf{x}, \mathbf{z})$ *is a kernel function, then also the following functions are kernel functions:*

- $k(\mathbf{x}, \mathbf{z}) = p(k1(\mathbf{x}, \mathbf{z}))$*, where* $p(x)$ *is a polynomial with positive coefficients*
- $k(\mathbf{x}, \mathbf{z}) = \exp(k1(\mathbf{x}, \mathbf{z}))$

Proof. The polynomial kernel results from the application of properties 2.2.13, 2.2.14 and 2.2.15, with (2.2.16) covering the constant term (by taking $f(\cdot) = c$). The exponential kernel results from the fact that the exponential function can be written as $\exp(x) = \lim_{t\to\infty} \sum_{k=0}^{t} \frac{x^k}{k!}$ which is a limit of polynomial functions with positive coefficients. Therefore it is a limit of kernel functions and since the semi-definite property is closed under taking pointwise limits, it is a kernel function. □

2.2.9 Gaussian kernel. *Let* $\mathbf{x}, \mathbf{z} \in \mathcal{X}$ *be two instance vectors and* $\sigma \in \mathbb{R}^+$ *a positive constant. Then, the function:*

$$k(\mathbf{x}, \mathbf{z}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{z}\|^2)}{2\sigma^2}\right) \tag{2.2.19}$$

is a kernel function called Gaussian kernel.

Proof. By Theorem 2.2.8 $\exp\left(\frac{\langle \mathbf{x}, \mathbf{z}\rangle}{\sigma^2}\right)$ is a kernel function. If we normalize this kernel as in (2.2.4) we obtain:

$$\frac{\exp\left(\frac{\langle \mathbf{x}, \mathbf{z}\rangle}{\sigma^2}\right)}{\sqrt{\exp\left(\frac{\|\mathbf{x}\|^2}{\sigma^2}\right)\exp\left(\frac{\|\mathbf{z}\|^2}{\sigma^2}\right)}} = \exp\left(\frac{\langle \mathbf{x}, \mathbf{z}\rangle}{\sigma^2} - \frac{\|\mathbf{x}\|^2}{2\sigma^2} - \frac{\|\mathbf{z}\|^2}{2\sigma^2}\right) = \exp\left(\frac{\|\mathbf{x} - \mathbf{z}\|^2}{2\sigma^2}\right)$$

□

Chapter 3

Matrix multi-view Perceptron

In this section we present the on-line multi-view learning model (Cavallanti et al., 2008) which has been specifically designed to manage the on-line multi-view binary classification task. It is a straightforward matrix extension of the conservative potential-based algorithm (compare in Figure 2.5 and 3.1), provided with a full multi-view analysis and a strict bound on the number of errors.

Assume $\mathbf{X}_t, \mathbf{V}_t \in \mathcal{X} \subseteq \mathbb{R}^{d\times K}$ $\mathbf{W}_t \in \mathbb{R}^{d\times K}$ and $\Phi : \mathbb{R}^{d\times K} \to \mathbb{R}^+$ is convex and differentiable.

Initialization $\mathbf{V}_0 = \mathbf{0}$, $\mathbf{W}_0 = \nabla\Phi(\mathbf{V}_0)$.

At each time t=1,2,..... do the following:

1: Observe instance $\mathbf{X}_t \in \mathcal{X}$;
2: Predict label $y_t \in \{-1, 1\}$ with $\hat{y}_t = \text{sign}(\langle \mathbf{W}_{t-1}, \mathbf{X}_t \rangle)$;
3: Observe actual label $y_t \in \{-1, 1\}$;
4: Update the weight vectors with:
$\mathbf{V}_t = \mathbf{V}_{t-1} + y_t \mathbf{X}_t \mathbb{I}_{\{\hat{y}_t \neq y_t\}} \quad \mathbf{W}_t = \nabla\Phi(\mathbf{V}_t)$

Figure 3.1: Matrix multi-view Perceptron algorithm

3.1 The model

Since we are now dealing with a matrix algorithm, we have to make some modifications to our previous formalization:

- the instance space is now supposed to be $\mathcal{X} \subseteq \mathbb{R}^{d\times K}$, which is itself a vector space and it is also an inner product space, with the Frobenius inner product $\langle \mathbf{A}, \mathbf{B}\rangle = \mathrm{Tr}(\mathbf{A}'\mathbf{B}) = \sum_{i,j} a_{ij}b_{ij}$
- at each time step t the forecaster observes a matrix $\mathbf{X}_t \in \mathcal{X}$, predicts using a matrix $\mathbf{W}_{t-1} \in \mathbb{R}^{d\times K}$ and its performance is evaluated against a competitor matrix $\mathbf{U} \in \mathbb{R}^{d\times K}$.
- we consider convex and differentiable potential functions $\Phi : \mathbb{R}^{d\times K} \to \mathbb{R}^+$ of the form $\Phi = \frac{1}{2}(f \circ \sigma)^2$, where:
  - $r = \min\{d, K\}$
  - $\sigma : \mathbb{R}^{d\times K} \to \mathbb{R}^r$, $\sigma(\mathbf{A}) = [\sigma_1, ..., \sigma_r] = \boldsymbol{\sigma}_A$ and $\sigma_1 \geq \sigma_2 \geq \cdots\cdots \sigma_r \geq 0$ are the Singular Values of A
  - $f : \mathbb{R}^r \to \mathbb{R}^+$ is a vector norm, invariant under permutation of the components of its argument.

Moreover, we specialize the algorithm to the the Schatten $2p$-Norm $f(\sigma(\mathbf{X})) = \|\mathbf{X}\|_{S_{2p}}$. Specifically, as in example 2.1.10, we will consider the Legendre potential function: $\Phi(\mathbf{X}) = \frac{1}{2}\|\mathbf{X}\|^2_{S_{2p}}$ which, for what we have seen in (2.1.8) is still a convex premetric.

3.1.1 Schatten Norm. Given a matrix $\mathbf{X} \in \mathbb{R}^{d\times K}$ the Schatten p-norm of $\mathbf{X}$ is given by:

$$\|\mathbf{X}\|_{S_p} \overset{def}{=} \left(\sum_{\sigma\in\boldsymbol{\sigma}_{\mathbf{X}'\mathbf{X}}} \sigma^{p/2}\right)^{1/p} = \|\boldsymbol{\sigma}_{\mathbf{X}'\mathbf{X}}\|^{1/2}_{p/2} \tag{3.1.1}$$

Note that, by singular value decomposition, $\mathbf{X}'\mathbf{X} = \mathbf{V}\boldsymbol{\Sigma}\mathbf{U}'\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}' = \mathbf{V}\boldsymbol{\Sigma}^2\mathbf{V}'$, so that $\mathrm{Tr}(\mathbf{X}'\mathbf{X}) = \mathrm{Tr}(\mathbf{V}\boldsymbol{\Sigma}^2\mathbf{V}') = \mathrm{Tr}(\mathbf{V}'\mathbf{V}\boldsymbol{\Sigma}^2) = \mathrm{Tr}(\boldsymbol{\Sigma}^2)$ and therefore:

$$\frac{1}{2}\,\mathrm{Tr}\left((\mathbf{X}'\mathbf{X})^p\right)^{\frac{1}{p}} = \frac{1}{2}\,\mathrm{Tr}\left(\left(\boldsymbol{\Sigma}^2\right)^p\right)^{\frac{1}{p}} = \frac{1}{2}\left(\sum_{\sigma\in\boldsymbol{\sigma}_{\mathbf{X}'\mathbf{X}}} \sigma^p\right)^{1/p} = \frac{1}{2}\|\mathbf{X}\|^2_{S_{2p}} = \frac{1}{2}\|\boldsymbol{\sigma}_{\mathbf{X}'\mathbf{X}}\|_p \tag{3.1.2}$$

Moreover, since:

$$\frac{1}{2}\|\mathbf{X}\|_{S_{2p}}^2 = \frac{1}{2}\left(\sum_{\sigma\in\boldsymbol{\sigma}_{\mathbf{X}'\mathbf{X}}} \sigma^p\right)^{1/p} = \frac{1}{2}\left(\sum_{\sigma\in\boldsymbol{\sigma}_{\mathbf{X}}} \sigma^{2p}\right)^{2/2p} = \frac{1}{2}\|\boldsymbol{\sigma}_{\mathbf{X}}\|_{2p}^2$$

we have also that:

$$\|\mathbf{X}\|_{S_{2p}} = \|\boldsymbol{\sigma}_{\mathbf{X}}\|_{2p} \tag{3.1.3}$$

We see that $\Phi(\mathbf{X}) = \frac{1}{2}\|\mathbf{X}\|_{S_{2p}}^2$ is a potential function of the above specified form $\frac{1}{2}(f\circ\sigma)^2$, where $f(\cdot)$ is the vector $2p$-norm $\|\cdot\|_{2p}$ and its dual is easily computed by the following Theorem.

3.1.1 Theorem. *The dual of* $\Phi(\mathbf{X}) = \frac{1}{2}\|\mathbf{X}\|_{S_{2p}}^2$ *is* $\Phi(\mathbf{X})^* = \frac{1}{2}\|\mathbf{X}\|_{S_{2q}}^2$, *where* $\frac{1}{2p} + \frac{1}{2q} = 1$

Proof. For what we have seen in example 2.1.10 if $f(\cdot) = \frac{1}{2}\|\cdot\|_{2p}^2$, then $f^*(\cdot) = \frac{1}{2}\|\cdot\|_{2q}^2$. Moreover by (Lewis, 1995, Theorem 2.4)

$$(f\circ\sigma)^* = f^*\circ\sigma$$

therefore we have:

$$\Phi(\mathbf{X})^* = \left(\frac{1}{2}\|\mathbf{X}\|_{S_{2p}}^2\right)^* = \left(\frac{1}{2}\|\boldsymbol{\sigma}_{\mathbf{X}}\|_{2p}^2\right)^* = \frac{1}{2}\|\boldsymbol{\sigma}_{\mathbf{X}}\|_{2q}^2 = \frac{1}{2}\|\mathbf{X}\|_{S_{2q}}^2$$

□

We still need to prove that $\Phi(\mathbf{X}) = \frac{1}{2}\|\mathbf{X}\|_{S_{2p}}^2$ is a differentiable function and to compute its gradient. These computations are quite long and not very important, so we state them as a Theorem and redirect to (Cavallanti et al., 2008) the interested reader.

3.1.2 Theorem. $\Phi_p(\mathbf{V}) = \frac{1}{2}\|\mathbf{V}\|_{S_{2p}}^2$ *is a differentiable function and its gradient* $\nabla\Phi_p(\mathbf{V})$ *is given by*

$$\begin{aligned}\nabla\Phi_p(\mathbf{V}) = \nabla 1/2\|\mathbf{V}\|_{S_{2p}}^2 &= \mathrm{Tr}\left(\left(\mathbf{V}'\mathbf{V}\right)^p\right)^{\frac{1}{p}(1-p)}\mathbf{V}\left(\mathbf{V}'\mathbf{V}\right)^{p-1} \\ &= \|\mathbf{V}\|_{S_{2p}}^{2(1-p)}\mathbf{V}\left(\mathbf{V}'\mathbf{V}\right)^{p-1}\end{aligned} \tag{3.1.4}$$

At this point we have all the machinery needed to use Schatten p-Norms in Algorithm 3.1, however to better understand how matrix Perceptron can be used in a multi-view context and what it actually computes with respect to the single views, we need to give some more explanations on the multi-view interpretation of this matrix algorithm.

Multi-view interpretation In a multi-view interpretation we see each matrix as being composed of K *single-view* vectors, related to K different views:

- $\mathbf{X}_t = \left[\mathbf{x}_{t,1}, \mathbf{x}_{t,2}, ..., \mathbf{x}_{t,K}\right], \quad \mathbf{x}_{t,k} \in \mathbb{R}^d$
- $\mathbf{V}_t = \left[\mathbf{v}_{t,1}, \mathbf{v}_{t,2}, ..., \mathbf{v}_{t,K}\right], \quad \mathbf{v}_{t,k} \in \mathbb{R}^d$
- $\mathbf{W}_t = \left[\mathbf{w}_{t,1}, \mathbf{w}_{t,2}, ..., \mathbf{w}_{t,K}\right], \quad \mathbf{w}_{t,k} \in \mathbb{R}^d$
- $\mathbf{U} = \left[\mathbf{u}_1, \mathbf{u}_2, ..., \mathbf{u}_K\right], \quad \mathbf{u}_k \in \mathbb{R}^d$

where the index $\{t, k\}$[1] stands for the k view at time t and, as before, we call $\mathbf{V}_t$ and $\mathbf{v}_{t,k}$ *primal weights*, while $\mathbf{W}_t$ and $\mathbf{w}_{t,k}$ are called *dual weights*. Furthermore we set the loss function to be:

$$\ell_t(\mathbf{U}) = \sum_{k=1}^{K} \left[1 - y_t \langle \mathbf{u}_k, \mathbf{x}_{t,k} \rangle\right]_+ = \sum_{k=1}^{K} \ell_t(\mathbf{u}_k) \tag{3.1.5}$$

which amounts to separately penalize the K comparison vectors $\mathbf{u}_k$, each of them evaluated on its own view instance $\mathbf{x}_{t,k}$. In so doing, the performance of our matrix multi-view classifier is evaluated with respect to the sum of the losses of K single-view classifiers.
Matrix Perceptron main theoretical result for this loss function is given below (Cavallanti et al., 2008).

3.1.3 Matrix Perceptron proof of convergence. *Let* $\mathbf{U}$ *be an arbitrary comparison matrix of size* $d \times K$*, then the number of mistakes* m *made by the 2p-norm matrix Perceptron, run on any finite sequence of examples* $(\mathbf{X}_1, y_1), (\mathbf{X}_2, y_2), ..., (\mathbf{X}_n, y_n) \in \mathbb{R}^{d \times K} \times \{-1, +1\}$*, satisfies*

$$m \leq \frac{L_n(\mathbf{U})}{K} + (2p-1)\left(\frac{\mathbf{X}_{S_{2p}} \|\mathbf{U}\|_{S_{2q}}}{K}\right)^2 + \frac{\mathbf{X}_{S_{2p}} \|\mathbf{U}\|_{S_{2q}}}{K} \sqrt{\frac{(2p-1)L_n(\mathbf{U})}{K}} \tag{3.1.6}$$

where $\mathbf{X}_{S_{2p}} = \max_{t=1,...,n} \|\mathbf{X}_t\|_{S_{2p}}$ *and* $\|\mathbf{U}\|_{S_{2q}}$ *is the Schatten 2q-Norm of* $\mathbf{U}$*, with* $2q = \frac{2p}{2p-1}$

[1] in some parts of this section when the time index t is not relevant we omit it, in order to keep the notation uncluttered.

Since $L_n(\mathbf{U})/K = \frac{1}{K}\sum_{k=1}^{K}\ell_t(\mathbf{u}_k)$, a direct comparison to bound 2.1.11 shows that matrix Perceptron performs at least as good as the mean of K Perceptron classifiers.

3.1.1 Prediction

Perceptron prediction is quite simple to understand: it predicts with the signum of the projection of the instances on the weight vector. A similar consideration could be done for matrix Perceptron, in a single view interpretation, where the algorithm works with matrix single-view instances and dual matrix weights. However if we treat it as a multi-view algorithm, where each matrix is made up of K different view vectors, what can we say about it? How are different view instances $\mathbf{x}_k$ and weights $\mathbf{v}_k$ combined together in order to get the prediction?
We start by noting that, in analogy with what we have written in (2.2.5), also for matrix Perceptron we can write:

$$\begin{aligned}\mathbf{V}_t &= \sum_{s\in S_t} y_s\mathbf{X}_s = \sum_{s\in S_t} y_s \begin{bmatrix}\mathbf{x}_{s,1} & \mathbf{x}_{s,2} & \cdots & \mathbf{x}_{s,K}\end{bmatrix} \\ &= \begin{bmatrix}\sum_{s\in S_t} y_s\mathbf{x}_{s,1} & \sum_{s\in S_t} y_s\mathbf{x}_{s,2} & \cdots & \sum_{s\in S_t} y_s\mathbf{x}_{s,K}\end{bmatrix} \\ &= \begin{bmatrix}\mathbf{v}_{t,1} & \mathbf{v}_{t,2} & \cdots & \mathbf{v}_{t,K}\end{bmatrix} \end{aligned} \tag{3.1.7}$$

where, as before, S_t is the index set of the missclassified samples at time t and:

$$\mathbf{v}_{t,k} = \sum_{s\in S_t} y_s\mathbf{x}_{s,k} \tag{3.1.8}$$

Every single $\mathbf{v}_{t,k}$ is computed in a similar way to Perceptron, so we could be tempted to think on matrix Perceptron as K different Perceptrons running in parallel on the K different views. However this interpretation is misleading because:

1. in prediction matrix Perceptron uses the dual weight $\mathbf{W}_{t-1}$, instead of the primal $\mathbf{V}_{t-1}$

2. matrix Perceptron updates the weights on all the single views in a synchronous fashion. Therefore if for example a prediction error occurs (with respect to the matrix Perceptron prediction function $\hat{y}_t =$

sign($\langle \mathbf{W}_{t-1}, \mathbf{X}_t \rangle$)) every single-view weight vector $\mathbf{v}_k$ is updated, even if it would have brought to a correct prediction with respect to the single-view instance vector $\mathbf{x}_k$

Specifically, the way in which different single-view prediction margins are combined together is:

$$\begin{aligned}\langle \mathbf{W}_{t-1}, \mathbf{X}_t \rangle = \operatorname{Tr}\left(\mathbf{W}'_{t-1}\mathbf{X}_t\right) &= \operatorname{Tr}\left(\begin{bmatrix} \mathbf{w}'_{t-1,1} \\ \mathbf{w}'_{t-1,2} \\ \vdots \\ \mathbf{w}'_{t-1,K} \end{bmatrix} \begin{bmatrix} \mathbf{x}_{t,1} & \mathbf{x}_{t,2} & \cdots & \mathbf{x}_{t,K} \end{bmatrix}\right) \\ &= \sum_{k=1}^{K} \langle \mathbf{w}_{t-1,k}\ ,\ \mathbf{x}_{t,k} \rangle \end{aligned} \tag{3.1.9}$$

With respect to the single-view dual weights $\mathbf{w}_{t-1,k}$, matrix Perceptron prediction margin is therefore equal to the sign of the sum of the margins of every single view $\mathbf{x}_k$.

Up to now we have seen how every single-view primal weight vector $\mathbf{v}_k$ is updated and how every single-view dual weight vector $\mathbf{w}_k$ is used in prediction. There is still a point missing: how the single-view primal weight vector $\mathbf{v}_k$ is transformed into the single-view dual weight vector $\mathbf{w}_k$ actually used in prediction. We have an explicit formula for the function that transforms the primal weight matrix $\mathbf{V}$ into the dual weight matrix $\mathbf{W}$, but we do not have an explicit formula for the transforming function applied to every single-view primal weight $\mathbf{v}_k$. In order to find this function we have to express $\mathbf{W}_t$ in terms of $\mathbf{V}_t$ and look for a way to express $\mathbf{w}_k$ in terms of $\mathbf{v}_k$. At first, we note that by Theorem 3.1.2 $\mathbf{W}_t = \nabla\Phi(\mathbf{V}_t) = \|\mathbf{V}_t\|_{S_{2p}}^{2(1-p)}\mathbf{V}_t\left(\mathbf{V}'_{\mathbf{t}}\mathbf{V}_{\mathbf{t}}\right)^{p-1}$ and we can write matrix prediction rule as:

$$\begin{aligned}\hat{y}_t &= \operatorname{sign}(\langle \mathbf{W}_{t-1}, \mathbf{X}_t \rangle) = \operatorname{sign}\left(\langle \nabla\Phi(\mathbf{V}_{t-1}), \mathbf{X}_t \rangle\right) \\ &= \operatorname{sign}\left(\|\mathbf{V}_{t-1}\|_{S_{2p}}^{2(1-p)} \left\langle \mathbf{V}_{t-1}\left(\mathbf{V}'_{t-1}\mathbf{V}_{t-1}\right)^{p-1}, \mathbf{X}_t \right\rangle\right) \\ &= \operatorname{sign}\left(\left\langle \mathbf{V}_{t-1}\left(\mathbf{V}'_{t-1}\mathbf{V}_{t-1}\right)^{p-1}, \mathbf{X}_t \right\rangle\right) \end{aligned} \tag{3.1.10}$$

and, since $\mathbf{V}\left(\mathbf{V}'\mathbf{V}\right)^n = \left(\mathbf{V}\mathbf{V}'\right)^n\mathbf{V}$, we also have that:

$$\hat{y}_t = \operatorname{sign}\left(\left\langle \left(\mathbf{V}\mathbf{V}'\right)^{p-1}\mathbf{V}, \mathbf{X}_t \right\rangle\right). \tag{3.1.11}$$

This last way of writing $\hat{y}_t$ seems not computationally efficient, because $\mathbf{V}\mathbf{V}' \in \mathbb{R}^{d\times d}$, while $\mathbf{V}'\mathbf{V} \in \mathbb{R}^{K\times K}$ and usually $K << d$. However it can be useful in problems where the number of views exceeds the dimensionality of the single-view instance vectors, moreover it will come back in the subsequent analysis.

Prediction with respect to p We are now ready to start the analysis of the behavior of matrix Perceptron algorithm, with respect to the parameter p:

1. when $p = 1$, matrix Perceptron simply computes $\mathrm{Tr}\left(\mathbf{V}'\mathbf{X}\right) = \sum_{k=1}^{K} \langle \mathbf{v}_k \, , \, \mathbf{x}_k \rangle$. Its prediction margin is therefore given simply by the sum of the K single-view margins, given by K Perceptron-like predictions, on the K different single-views instance vectors. In this case every single-view primal weight $\mathbf{v}_k$ is applied only on its own view $\mathbf{x}_k$ and the different views do not interact each-other.

2. if $p = 2$ we have that $\mathbf{W} = \mathbf{V}\left(\mathbf{V}'\mathbf{V}\right)$ and:

$$\mathbf{V}'\mathbf{V} = \begin{bmatrix} \mathbf{v}_1' \\ \mathbf{v}_2' \\ \vdots \\ \mathbf{v}_K' \end{bmatrix} \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_K \end{bmatrix} = \begin{bmatrix} \langle \mathbf{v}_1, \mathbf{v}_1 \rangle & \langle \mathbf{v}_1, \mathbf{v}_2 \rangle & \cdots & \langle \mathbf{v}_1, \mathbf{v}_K \rangle \\ \langle \mathbf{v}_2, \mathbf{v}_1 \rangle & \langle \mathbf{v}_2, \mathbf{v}_2 \rangle & \cdots & \langle \mathbf{v}_2, \mathbf{v}_K \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle \mathbf{v}_K, \mathbf{v}_1 \rangle & \langle \mathbf{v}_K, \mathbf{v}_2 \rangle & \cdots & \langle \mathbf{v}_K, \mathbf{v}_K \rangle \end{bmatrix}$$

$$\begin{aligned} \mathbf{W} &= \mathbf{V}\left(\mathbf{V}'\mathbf{V}\right) \\ &= \begin{bmatrix} \sum_{i=1}^{K} \mathbf{v}_i \langle \mathbf{v}_i, \mathbf{v}_1 \rangle & \sum_{i=1}^{K} \mathbf{v}_i \langle \mathbf{v}_i, \mathbf{v}_2 \rangle & \cdots & \sum_{i=1}^{K} \mathbf{v}_i \langle \mathbf{v}_i, \mathbf{v}_K \rangle \end{bmatrix} \\ &= \begin{bmatrix} \mathfrak{F}(\mathbf{v}_1) & \mathfrak{F}(\mathbf{v}_2) & \cdots & \mathfrak{F}(\mathbf{v}_K) \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{w}_1 & \mathbf{w}_2 & \cdots & \mathbf{w}_K \end{bmatrix} \end{aligned}$$

where $\mathfrak{F}(\cdot)$ is the transformation introduced in appendix A.1.
Exploiting this result, we can write:

$$\hat{y} = \mathrm{sign}\left(\sum_{k=1}^{K} \langle \mathfrak{F}(\mathbf{v}_k) \, , \, \mathbf{x}_k \rangle \right)$$

3. if $p = 3$, it is easy to verify that

$$\mathbf{w}_k = \sum_{j=1}^{K} \left\langle \sum_{i=1}^{K} \langle \mathbf{v}_k, \mathbf{v}_i \rangle \mathbf{v}_i, \mathbf{v}_j \right\rangle \mathbf{v}_j = \mathfrak{F}\left(\mathfrak{F}(\mathbf{v}_k)\right) = \mathfrak{F}^2(\mathbf{v}_k)$$

where the exponent of the $\mathfrak{F}$ does not refer to the power, but to the multiple application of $\mathfrak{F}$ to $\mathbf{v}$.

Therefore, for $p = 3$ we have:

$$\hat{y} = \text{sign}\left(\sum_{k=1}^{K} \left\langle \mathfrak{F}^2(\mathbf{v}_k)\ ,\ \mathbf{x}_k \right\rangle\right)$$

The result for the general case is stated in the following Theorem.

3.1.2 Theorem. *If $\Phi_p(\cdot) = \frac{1}{2}\|\cdot\|^2_{S_{2p}}$, $\mathbf{W}_t = \nabla\Phi(\mathbf{V}_t)$ and $\mathbf{v}_{t,k}$ is a single-view primal weight vector of the multi-view matrix $\mathbf{V}_t$. The corresponding single-view dual vector $\mathbf{w}_{t,k}$ is given by:*

$$\mathbf{w}_{t,k} = \mathfrak{F}^{p-1}(\mathbf{v}_{t,k}) \tag{3.1.12}$$

where $\mathfrak{F}(\cdot)$ is the transformation introduced in appendix A.1. and $\mathfrak{F}^0(\mathbf{v}_k) = \mathbf{v}_k$.

Moreover, by (A.2), in the linear case (when the inner product is not substituted by a kernel function) we also have:

$$\mathbf{w}_k = \mathfrak{F}^{p-1}(\mathbf{v}_k) = (\mathbf{V}\mathbf{V}')^{p-1}\mathbf{v}_k \tag{3.1.13}$$

Matrix Perceptron prediction can then be written as:

$$\hat{y} = \text{sign}\left(\sum_{k=1}^{K} \left\langle \mathfrak{F}^{p-1}(\mathbf{v}_k)\ ,\ \mathbf{x}_k \right\rangle\right) \tag{3.1.14}$$

and, for the linear case, as:

$$\hat{y} = \text{sign}\left(\sum_{k=1}^{K} \left\langle \left(\mathbf{V}\mathbf{V}'\right)^{p-1} \mathbf{v}_k, \mathbf{x}_k \right\rangle\right) = \text{sign}\left(\sum_{k=1}^{K} \mathbf{x}_k'(\mathbf{V}\mathbf{V}')^{p-1}\mathbf{v}_k\right) \tag{3.1.15}$$

This last equation is very similar to what we wrote in (3.1.11), but here we have underlined the role of each single view vector.

In this subsection we have shown that in the general case this algorithm applies $p-1$ times the transformation $\mathfrak{F}(\mathbf{v}_k) = \sum_{i=1}^{K} \mathbf{v}_i \langle \mathbf{v}_k, \mathbf{v}_i \rangle$ to

each single-view primal weight vector $\mathbf{v}_k$ and that in the linear case $\mathfrak{F}(\mathbf{v}_k)$ can be written as $\mathbf{V}\mathbf{V}'\mathbf{v}_k$, so $\mathfrak{F}^{p-1}(\mathbf{v}_k) = (\mathbf{V}\mathbf{V}')^{p-1}\,\mathbf{v}_k$. Moreover, matrix Perceptron prediction margin is given by the sum of the margins obtained by these transformed vectors.

3.1.2 Kernelization

Is matrix Perceptron algorithm a kernelizable algorithm? The answer is yes, as we will see shortly.

We start this analysis by noting that following the above established multi-view interpretation we can write:

$$\mathbf{V}_t = \sum_{s \in S_t} y_s \mathbf{X}_s = \sum_{s \in S_t} y_s \begin{bmatrix} \mathbf{x}_{s,1} & \mathbf{x}_{s,2} & \cdots & \mathbf{x}_{s,K} \end{bmatrix}$$

and

$$\begin{aligned} \mathbf{V}'_{t-1}\mathbf{X}_t &= \sum_{s \in S_{t-1}} y_s \left(\begin{bmatrix} \mathbf{x}'_{s,1} \\ \mathbf{x}'_{s,2} \\ \vdots \\ \mathbf{x}'_{s,K} \end{bmatrix} \begin{bmatrix} \mathbf{x}_{t,1} & \mathbf{x}_{t,2} & \cdots & \mathbf{x}_{t,K} \end{bmatrix} \right) \\ &= \sum_{s \in S_{t-1}} y_s \begin{bmatrix} \langle \mathbf{x}_{s,1}, \mathbf{x}_{t,1} \rangle & \langle \mathbf{x}_{s,1}, \mathbf{x}_{t,2} \rangle & \cdots & \langle \mathbf{x}_{s,1}, \mathbf{x}_{t,K} \rangle \\ \langle \mathbf{x}_{s,2}, \mathbf{x}_{t,1} \rangle & \langle \mathbf{x}_{s,2}, \mathbf{x}_{t,2} \rangle & \cdots & \langle \mathbf{x}_{s,2}, \mathbf{x}_{t,K} \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle \mathbf{x}_{s,K}, \mathbf{x}_{t,1} \rangle & \langle \mathbf{x}_{s,K}, \mathbf{x}_{t,2} \rangle & \cdots & \langle \mathbf{x}_{s,K}, \mathbf{x}_{t,K} \rangle \end{bmatrix} \end{aligned} \tag{3.1.16}$$

so that the computation of $\mathbf{V}'_{t-1}\mathbf{X}_t$ depends on the views $\mathbf{x}_{t,k}$ only through their inner product. Moreover, in case of misclassification we have $\mathbf{V}_t = \mathbf{V}_{t-1} + y_t\mathbf{X}_t$ and $\mathbf{V}'_t\mathbf{V}_t$ can be written as:

$$\begin{aligned} \mathbf{V}'_t\mathbf{V}_t &= (\mathbf{V}_{t-1} + y_t\mathbf{X}_t)'(\mathbf{V}_{t-1} + y_t\mathbf{X}_t) \\ &= \mathbf{V}'_{t-1}\mathbf{V}_{t-1} + y_t\mathbf{V}'_{t-1}\mathbf{X}_t + y_t\mathbf{X}'_t\mathbf{V}_{t-1} + y_t^2\mathbf{X}'_t\mathbf{X}_t \\ &= \mathbf{V}'_{t-1}\mathbf{V}_{t-1} + y_t\left(\mathbf{V}'_{t-1}\mathbf{X}_t + (\mathbf{V}'_{t-1}\mathbf{X}_t)'\right) + \mathbf{X}'_t\mathbf{X}_t \end{aligned} \tag{3.1.17}$$

Therefore $\mathbf{V}'_t\mathbf{V}_t$ can be computed iteratively using the products $\mathbf{V}'_{t-1}\mathbf{X}_t$ and $\mathbf{X}'_t\mathbf{X}_t$, that for what we have seen before it depend on the views $\mathbf{x}_{t,k}$ only through their inner product.

Therefore it is possible to compute $\langle \mathbf{W}_{t-1}, \mathbf{X}_t \rangle = \mathrm{Tr}\left((\mathbf{V}'_{t-1}\mathbf{V}_{t-1})^{p-1}\mathbf{V}'_{t-1}\mathbf{X}_t\right)$ using the view vectors only through their inner product and we can apply the *Kernel Trick* to substitute every inner product: $\langle \mathbf{x}_{t_1,k_1}, \mathbf{x}_{t_2,k_2} \rangle$ with a valid arbitrary kernel $k(\mathbf{x}_{t_1,k_1}, \mathbf{x}_{t_2,k_2})$.

3.1.3 Low risk hypothesis extraction

As already stated in the model formalization 2.1.1, an on-line forecaster produce an hypothesis f_t in a space of hypotheses $\mathcal{H}$ at every time-step t. In principle there is no warranty for the last hypothesis generated to generalize better than the previous ones. How can then we extract a good hypothesis from the ensemble of hypotheses generated by an on-line algorithm? In order to answer this question we have to introduce the definition of *risk*. Risk can be seen as: "the threat or probability that an action or event will adversely or beneficially affect an organization's ability to achieve its objectives".

If we suppose that the learning algorithm is run on an independent and identically distributed ($i.i.d$) sample from an underlying distribution (often unknown), a *risk function* can be defined as the average loss of the labeled instances, with respect to the underlying distribution.

3.1.4 Risk function. If a learning algorithm is run with a weight vector $\mathbf{w}$, on an independent and identically distributed sample drawn from the sample space S of an underlying distribution P. The risk function $R : \mathbb{R}^d \to \mathbb{R}^+$ is defined as:

$$R(\mathbf{w}) \stackrel{def}{=} E\left[\ell_t(\mathbf{w})\right] = \int_S \ell_t(\mathbf{w}) dP \tag{3.1.18}$$

This definition reflects the true expected value of the loss function, with respect to the underlying distribution P. Since most of the time we do not know the underlying distribution, we usually do not have access to the empirical risk, however we can still estimate it with the sample mean.

3.1.5 Empirical risk function. If a learning algorithm is run with weight a vector $\mathbf{w}$, on an independent and identically distributed sequence of examples of length T, the *empirical risk function* $R_{emp} : \mathbb{R}^d \to \mathbb{R}^+$ is defined as:

$$R_{emp}(\mathbf{w}) \stackrel{def}{=} \frac{1}{T} L_T(\mathbf{w}) \tag{3.1.19}$$

where $L_T(\mathbf{w})$ is the cumulative loss of the sequence.

We now introduce other two simple statistics on the ensemble of hypotheses generated by an on-line algorithm and state an important result proved in (Cesa-Bianchi et al., 2004).
Given a sample of length T and an on-line learner A, we call $\mathbf{w}_0, \mathbf{w}_1, ...\mathbf{w}_{T-1}$ the ensemble of hypotheses generated by A and we define the following sample statistic:

$$M_T = \frac{1}{T}\sum_{t=1}^{T} \ell_t(\mathbf{w}_{t-1}) \tag{3.1.20}$$

Moreover if the decision space $\mathcal{D}$ of A is a convex set and the loss function ℓ is convex, the *average hypothesis* $\overline{\mathbf{w}}$ is given by

$$\overline{\mathbf{w}}_T = \frac{1}{T}\sum_{t=1}^{T} \mathbf{w}_{t-1} \tag{3.1.21}$$

3.1.6 Theorem. *Let $\mathbf{w}_0, \mathbf{w}_1, ...\mathbf{w}_{T-1}$ be the ensemble of hypotheses generated by an arbitrary on-line algorithm A working with a convex loss function $0 \leq \ell \leq 1$. Then for any $0 < \delta \leq 1$*

$$P\left(R(\overline{\mathbf{w}}_T) \geq M_T + \sqrt{\frac{2}{T}\ln\frac{1}{\delta}}\right) \leq \delta \tag{3.1.22}$$

This Theorem states that with high probability the expected loss of the average hypothesis is very near to the sample average loss encountered when running the algorithm on a sample of size T, of instance/label pairs. This means that if the samples are i.i.d. and we train an on-line algorithm on a subset of size T_{tr} of a sample space of size T; if T_{tr} is big enough we can expect $\overline{\mathbf{w}}_{T_{tr}}$ to perform as well as $M_{T_{tr}}$ also on the remaining sample of size $T_{te} = T - T_{tr}$.

If we are given a training set of T_{tr} labeled examples, a test set of T_{te} unlabeled instances and we extract the mean hypothesis $\overline{\mathbf{w}}_{T_{tr}}$ from the on-line training, we can thus expect this hypothesis to perform as well as $M_{T_{tr}}$ also on the test set.

Matrix Perceptron mean weight As we have seen, in order to extract a good hypothesis from the generated ensemble, we have to compute the

mean weight vector. For the matrix Perceptron algorithm we can compute it in the following way:

$$\overline{\mathbf{W}}_T = \frac{1}{T}\sum_{t=1}^{T}\mathbf{W}_{t-1} = \frac{1}{T}\sum_{t=1}^{T}\mathbf{V}_{t-1}\mathbf{F}_{t-1} \tag{3.1.23}$$

where $\mathbf{F}_k = (\mathbf{V}_k'\mathbf{V}_k)^{p-1}\|\mathbf{V}_k\|_{S_{2p}}^{2(1-p)}$. This formula is useful when matrix Perceptron algorithm is used in its primal form. However when we are dealing with kernels and matrix Perceptron algorithm is used in its dual form, we have access to $\mathbf{V}_{t-1}$ only through the inner product between single-view instance vectors. We therefore have to find a way to express it with respect to them.

We start by noting that, since $\mathbf{V}_T = \sum_{s\in S_T} y_s\mathbf{X}_s = \sum_{t=1}^{T} y_t\mathbf{X}_t\mathbb{I}_{\{\hat{y}_t\neq y_t\}}$, we can write:

$$\overline{\mathbf{W}}_T = \frac{1}{T}\sum_{t=1}^{T}\left(\sum_{s\in S_{t-1}} y_s\mathbf{X}_s\right)\mathbf{F}_{t-1}$$

This way of writing $\overline{\mathbf{W}}_T$ is very inefficient, because it needs the full sets $S_1, S_2, ...S_{T-1}$ of support matrices, encountered at every time-step t. Moreover it uses all the support matrices in all those sets in order to compute the final average.

We can improve efficiency by noting that the set of support matrices S_t at time-step t is included in the set of support matrices S_{t+1} at time-step $t+1$, so that we have: $S_1 \subseteq S_2 \subseteq ... \subseteq S_T$, as shown in Figure 3.1.3.

Indeed the generic sequence of matrix Perceptron dual weights can be

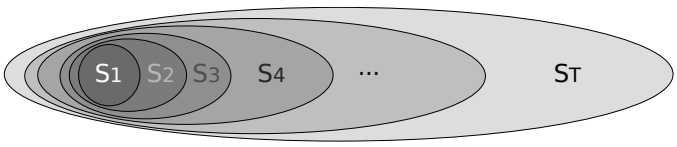


Figure 3.2: Relation between the sets of support vectors S_t, at consecutive time-steps.

written as follow:

$$
\begin{aligned}
\mathbf{W}_0 &= \mathbf{0} \\
\mathbf{W}_1 &= y_1 \mathbf{X}_1 \mathbf{F}_1 \mathbb{I}_{\{\hat{y}_1 \neq y_1\}} \\
\mathbf{W}_2 &= y_1 \mathbf{X}_1 \mathbf{F}_2 \mathbb{I}_{\{\hat{y}_1 \neq y_1\}} \quad + y_2 \mathbf{X}_2 \mathbf{F}_2 \mathbb{I}_{\{\hat{y}_2 \neq y_2\}} \\
&\vdots \\
\mathbf{W}_{T-1} &= y_1 \mathbf{X}_1 \mathbf{F}_{T-1} \mathbb{I}_{\{\hat{y}_1 \neq y_1\}} + y_2 \mathbf{X}_2 \mathbf{F}_{T-1} \mathbb{I}_{\{\hat{y}_2 \neq y_2\}} + ... + y_{T-1} \mathbf{X}_{T-1} \mathbf{F}_{T-1} \mathbb{I}_{\{\hat{y}_{T-1} \neq y_{T-1}\}}
\end{aligned}
$$

and if we sum over columns we obtain:

$$
\begin{aligned}
\sum_{t=1}^{T} \mathbf{W}_{t-1} &= y_1 \mathbf{X}_1 \left(\sum_{t=1}^{T-1} \mathbf{F}_t \right) \mathbb{I}_{\{\hat{y}_1 \neq y_1\}} + y_2 \mathbf{X}_2 \left(\sum_{t=2}^{T-1} \mathbf{F}_t \right) \mathbb{I}_{\{\hat{y}_2 \neq y_2\}} + \\
&+ ... + y_{T-2} \mathbf{X}_{T-2} \left(\sum_{t=T-2}^{T-1} \mathbf{F}_t \right) \mathbb{I}_{\{\hat{y}_{T-2} \neq y_{T-2}\}} + y_{T-1} \mathbf{X}_{T-1} \mathbf{F}_{T-1} \mathbb{I}_{\{\hat{y}_{T-1} \neq y_{T-1}\}} \\
&= \sum_{t=1}^{T-1} y_t \mathbf{X}_t \left(\sum_{j=t}^{T-1} \mathbf{F}_j \right) \mathbb{I}_{\{\hat{y}_t \neq y_t\}}
\end{aligned}
$$

so that:

$$
\begin{aligned}
\overline{\mathbf{W}}_T &= \frac{1}{T} \sum_{t=1}^{T-1} y_t \mathbf{X}_t \left(\sum_{j=t}^{T-1} \left(\mathbf{V}_j' \mathbf{V}_j \right)^{p-1} \|\mathbf{V}_j\|_{S_{2p}}^{2(1-p)} \right) \mathbb{I}_{\{\hat{y}_t \neq y_t\}} \\
&= \frac{1}{T} \sum_{s \in S_T} y_s \mathbf{X}_s \left(\sum_{j=t}^{T-1} \mathbf{F}_j \right) \qquad (3.1.24)
\end{aligned}
$$

As we have seen in (3.1.17), it is possible to compute $\mathbf{V}_t' \mathbf{V}_t$ using the single-view instance vectors only through their inner products and therefore it is also possible to compute $(\mathbf{V}_t' \mathbf{V}_t)^{p-1}$ and $\|\mathbf{V}\|_{S_{2p}}^{2(1-p)} = \mathrm{Tr}\left((\mathbf{V}'\mathbf{V})^p \right)^{\frac{1}{p}(1-p)}$ using the single-view instance vectors only through their inner product.

Concluding, if in the training phase, whenever there is a misclassification error and a support matrix $\mathbf{X}_s$ is stored, the algorithm incrementally computes and stores also $\bar{\mathbf{F}}_s = \sum_{j=t_s}^{T-1} \mathbf{F}_j$ (where t_s is the time-step at which the the misclassification happened), it is then possible to use the mean weight statistic on the test set in the following way:

$$
\langle \overline{\mathbf{W}}_T, \mathbf{X}_{te} \rangle = \frac{1}{T} \sum_{s \in S_T} y_s \, \mathrm{Tr}\left(\bar{\mathbf{F}}_s' (\mathbf{X}_s' \mathbf{X}_{te}) \right) \qquad (3.1.25)
$$

The advantage of this method is that in the testing phase the algorithm has to compute only the $2|S|$ matrix products, while all the operations involved in the computation of $\bar{\mathbf{F}}_s$ have already been completed during the training phase. This way of averaging the weight matrices is thus slow in the training phase, but allows fast predictions after the training has been completed.

3.2 Motivation for further work

Up to now we have presented Matrix Perceptron as a straightforward matrix generalization of the Perceptron algorithm, provided with a full multi-view analysis and a strict convergence bound. We have also shown how to apply kernel trick with respect to the single-view instance vectors and how to extract a low risk hypothesis from its primal and dual formulation. However together with its new ideas and theoretical guarantees matrix multi-view Perceptron algorithm brings also some questions:

- the algorithm computes inner products between different views $k_i, k_j, i \neq j$. This means that all views are truly forced to belong to the same space $\mathbb{R}^d$, as previously stated in our multi-view interpretation. While this fact seems natural in a matrix interpretation, it is quite unnatural in a real multi-view setting: for example audio and visual features are likely not expected to share the same instance space and indeed this limitation is not present in any of the other multi-view approaches cited in the introduction. How can we make this algorithm manage views belonging to different spaces? Or, how can views belonging to different spaces be mapped into a common space, in a significant way?

- we have seen in 2.2.3 that a kernel function can be seen as a similarity measure between two inputs, however, since the algorithm computes inner products between different views, we are faced the problem on how to chose a similarity measure between different views. Indeed in a problem with K views we have to choose $\frac{K(K+1)}{2}$ kernel functions. While there is a lot of knowledge on which is the best kernel for a certain kind of feature vectors, how can a similarity measure be chosen between two different kind of feature vectors (for example an audio

feature vector and a video feature vector)?

- if $\mathbf{X}, \mathbf{Y} \in \mathbb{R}^{d \times K}$, the naive time complexity of matrix multiplication $\mathbf{X}'\mathbf{Y}$ is $O(dK^2)$. Matrix Perceptron algorithm predicts by computing $\mathrm{Tr}\left((\mathbf{V}'\mathbf{V})^{p-1}\,\mathbf{V}'\mathbf{X}_t)\right)$, therefore every prediction step involves p matrix multiplication and its time complexity is $O(pdK^2)$, where p is a parameter of the algorithm, d is the size of each view and K is the number of views. This algorithm is therefore very inefficient for problems with many views. Is there a good computational approximation of this algorithm able to cope with these kind of problems?

In order to turn matrix Perceptron into a more general and efficient multi-view algorithm, applicable to real engineering problems we have to face these problems. With this spirit, in the next Chapter we will propose a modified version of matrix Perceptron, able to cope view vectors of different size and with a lower computational complexity.

Chapter 4

Orthogonal matrix multi-view Perceptron

As we have seen in the previous Chapter, one main problem of matrix Perceptron algorithm is that every view is forced to have the same dimensionality: if for example the instance space is chosen to be $\mathbb{R}^{d\times K}$, every view vector needs to be in $\mathbb{R}^d$, otherwise it is technically not possible to directly apply matrix Perceptron algorithm. However, in many multi-view problems information comes from different sensors, or with a completely different representation and it is unlikely to expect all the view vectors to belong to the same space.

In this Chapter we propose a method to face the above mentioned problem. Specifically we propose to build an orthogonal instance matrix from different view-vectors of different size. This kind of construction will prove to be helpful in facing the above mentioned problems, moreover it will result in a decreased computational time complexity of the original algorithm.

4.1 The model

If we are given K view vectors $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_K$ of size $d_1, d_2, ..., d_K$ we can construct a matrix $\tilde{\mathbf{X}} \in \mathbb{R}^{D\times K}$, where $D = \sum_{i=1}^{K} d_i$, by pre-mapping the view vectors into a new set of vectors $\tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2, \cdots, \tilde{\mathbf{x}}_K \in \mathbb{R}^D$, in the following

way:

$$\widetilde{\mathbf{X}} = \begin{bmatrix} \Phi(\mathbf{x}_1) & \Phi(\mathbf{x}_2) & \cdots & \Phi(\mathbf{x}_K) \end{bmatrix}$$
$$= \begin{bmatrix} \mathbf{x}_1 & \mathbf{0}_1 & \cdots & \mathbf{0}_1 \\ \mathbf{0}_2 & \mathbf{x}_2 & \cdots & \mathbf{0}_2 \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0}_K & \mathbf{0}_K & \cdots & \mathbf{x}_K \end{bmatrix} = \begin{bmatrix} \tilde{\mathbf{x}}_1 & \tilde{\mathbf{x}}_2 & \cdots & \tilde{\mathbf{x}}_K \end{bmatrix} \tag{4.1.1}$$

where $\mathbf{0}_k$ is a zero vector of size d_k. Within this framework we thus have the modified matrices:

- $\widetilde{\mathbf{X}}_t = [\tilde{\mathbf{x}}_{t,1}, \tilde{\mathbf{x}}_{t,2}, ..., \tilde{\mathbf{x}}_{t,K}], \quad \tilde{\mathbf{x}}_{t,k} \in \mathbb{R}^D$
- $\widetilde{\mathbf{V}}_t = [\tilde{\mathbf{v}}_{t,1}, \tilde{\mathbf{v}}_{t,2}, ..., \tilde{\mathbf{v}}_{t,K}], \quad \tilde{\mathbf{v}}_{t,k} \in \mathbb{R}^D$
- $\widetilde{\mathbf{W}}_t = [\tilde{\mathbf{w}}_{t,1}, \tilde{\mathbf{w}}_{t,2}, ..., \tilde{\mathbf{w}}_{t,K}], \quad \tilde{\mathbf{w}}_{t,k} \in \mathbb{R}^D$
- $\widetilde{\mathbf{U}} = [\tilde{\mathbf{u}}_1, \tilde{\mathbf{u}}_2, ..., \tilde{\mathbf{u}}_K], \quad \tilde{\mathbf{u}}_k \in \mathbb{R}^D$

By construction $\tilde{\mathbf{X}}$ is an orthogonal matrix, since we have:

$$\langle \Phi(\mathbf{x}_i), \Phi(\mathbf{x}_j) \rangle = \langle \tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j \rangle = \begin{cases} \|\tilde{\mathbf{x}}_i\|^2 = \|\mathbf{x}_i\|^2 & \text{if } j = i \\ 0 & \text{otherwise.} \end{cases} \tag{4.1.2}$$

and therefore

$$\widetilde{\mathbf{X}}'\widetilde{\mathbf{X}} = \begin{bmatrix} \tilde{\mathbf{x}}_1' \\ \tilde{\mathbf{x}}_2' \\ \vdots \\ \tilde{\mathbf{x}}_K' \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{x}}_1 & \tilde{\mathbf{x}}_2 & \cdots & \tilde{\mathbf{x}}_K \end{bmatrix}$$
$$= \begin{bmatrix} \langle \tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_1 \rangle & \langle \tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_2 \rangle & \cdots & \langle \tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_K \rangle \\ \langle \tilde{\mathbf{x}}_2, \tilde{\mathbf{x}}_1 \rangle & \langle \tilde{\mathbf{x}}_2, \tilde{\mathbf{x}}_2 \rangle & \cdots & \langle \tilde{\mathbf{x}}_2, \tilde{\mathbf{x}}_K \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle \tilde{\mathbf{x}}_K, \tilde{\mathbf{x}}_1 \rangle & \langle \tilde{\mathbf{x}}_K, \tilde{\mathbf{x}}_2 \rangle & \cdots & \langle \tilde{\mathbf{x}}_K, \tilde{\mathbf{x}}_K \rangle \end{bmatrix} = \begin{bmatrix} \|\mathbf{x}_1\|^2 & 0 & \cdots & 0 \\ 0 & \|\mathbf{x}_2\|^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \|\mathbf{x}_K\|^2 \end{bmatrix}$$

By (3.1.8) we see that every $\tilde{\mathbf{v}}_{t,k}$ is a linear combination of the instance vectors: $\tilde{\mathbf{v}}_{t,k} = \sum_{s \in S_t} y_s \tilde{\mathbf{x}}_{s,k}$. Therefore $\widetilde{\mathbf{V}}_t$ is an orthogonal matrix too, and

it can be written as follow:

$$[\tilde{\mathbf{v}}_{t,1}, \tilde{\mathbf{v}}_{t,2}, ..., \tilde{\mathbf{v}}_{t,K}] = \begin{bmatrix} \mathbf{v}_{t,1} & \mathbf{0}_1 & \cdots & \mathbf{0}_1 \\ \mathbf{0}_2 & \mathbf{v}_{t,2} & \cdots & \mathbf{0}_2 \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0}_K & \mathbf{0}_K & \cdots & \mathbf{v}_{t,K} \end{bmatrix} \tag{4.1.3}$$

where $\mathbf{v}_k = \sum_{s \in S_t} y_s \mathbf{x}_{s,k}$.

Moreover, by the expansion of $\mathbf{V}'\mathbf{X}$ in (3.1.16), we see that also $\widetilde{\mathbf{V}}'\widetilde{\mathbf{X}}$ is an orthogonal matrix with elements:

$$\begin{aligned} \widetilde{\mathbf{V}}'_{t-1}\widetilde{\mathbf{X}}_t &= \sum_{s \in S_{t-1}} y_s \begin{bmatrix} \langle \mathbf{x}_{s,1}, \mathbf{x}_{t,1} \rangle & 0 & \cdots & 0 \\ 0 & \langle \mathbf{x}_{s,2}, \mathbf{x}_{t,2} \rangle & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \langle \mathbf{x}_{s,K}, \mathbf{x}_{t,K} \rangle \end{bmatrix} \\ &= \begin{bmatrix} \langle \mathbf{v}_{t-1,1}, \mathbf{x}_{t,1} \rangle & 0 & \cdots & 0 \\ 0 & \langle \mathbf{v}_{t-1,2}, \mathbf{x}_{t,2} \rangle & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \langle \mathbf{v}_{t-1,K}, \mathbf{x}_{t,K} \rangle \end{bmatrix} \end{aligned} \tag{4.1.4}$$

Finally by (3.1.17), $\widetilde{\mathbf{V}}'\widetilde{\mathbf{V}}$ can be expressed as a sum of orthogonal matrices ($\widetilde{\mathbf{V}}'\widetilde{\mathbf{X}}$ and $\widetilde{\mathbf{X}}'\widetilde{\mathbf{X}}$), so that also $\widetilde{\mathbf{V}}'_{t-1}\widetilde{\mathbf{V}}_{t-1}$ is an orthogonal matrix and by definition 4.1 it can be written as:

$$\begin{aligned} \widetilde{\mathbf{V}}'_{t-1}\widetilde{\mathbf{V}}_{t-1} &= \begin{bmatrix} \langle \tilde{\mathbf{v}}_{t-1,1}, \tilde{\mathbf{v}}_{t-1,1} \rangle & \langle \tilde{\mathbf{v}}_{t-1,1}, \tilde{\mathbf{v}}_{t-1,2} \rangle & \cdots & \langle \tilde{\mathbf{v}}_{t-1,1}, \tilde{\mathbf{v}}_{t-1,K} \rangle \\ \langle \tilde{\mathbf{v}}_{t-1,2}, \tilde{\mathbf{v}}_{t-1,1} \rangle & \langle \tilde{\mathbf{v}}_{t-1,2}, \tilde{\mathbf{v}}_{t-1,2} \rangle & \cdots & \langle \tilde{\mathbf{v}}_{t-1,2}, \tilde{\mathbf{v}}_{t-1,K} \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle \tilde{\mathbf{v}}_{t-1,K}, \tilde{\mathbf{v}}_{t-1,1} \rangle & \langle \tilde{\mathbf{v}}_{t-1,K}, \tilde{\mathbf{v}}_{t-1,2} \rangle & \cdots & \langle \tilde{\mathbf{v}}_{t-1,K}, \tilde{\mathbf{v}}_{t-1,K} \rangle \end{bmatrix} \\ &= \begin{bmatrix} \|\mathbf{v}_{t-1,1}\|^2 & 0 & \cdots & 0 \\ 0 & \|\mathbf{v}_{t-1,2}\|^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \|\mathbf{v}_{t-1,K}\|^2 \end{bmatrix} \end{aligned} \tag{4.1.5}$$

4.1.1 Prediction

We are now ready to derive the prediction formula computed by the orthogonal matrix Perceptron. As we have seen in 3.1.10 matrix Perceptron

prediction is computed as:

$$\hat{y}_t = \text{sign}\left(\text{Tr}\left(\left(\mathbf{V}'_{t-1}\mathbf{V}_{t-1}\right)^{p-1}\mathbf{V}'_{t-1}\mathbf{X}_t\right)\right)$$

by equations 4.1.4 and 4.1.5 we have:

$$\begin{aligned}
&\left(\mathbf{V}'_{t-1}\mathbf{V}_{t-1}\right)^{p-1}\mathbf{V}'_{t-1}\mathbf{X}_t \\
&= \begin{bmatrix} \|\mathbf{v}_{t-1,1}\|^2 & 0 & \cdots & 0 \\ 0 & \|\mathbf{v}_{t-1,2}\|^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \|\mathbf{v}_{t-1,K}\|^2 \end{bmatrix}^{p-1} \times \\
&\times \begin{bmatrix} \langle \mathbf{v}_{t-1,1}, \mathbf{x}_{t,1}\rangle & 0 & \cdots & 0 \\ 0 & \langle \mathbf{v}_{t-1,2}, \mathbf{x}_{t,2}\rangle & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \langle \mathbf{v}_{t-1,K}, \mathbf{x}_{t,K}\rangle \end{bmatrix} \\
&= \begin{bmatrix} \|\mathbf{v}_{t-1,1}\|^{2(p-1)}\langle \mathbf{v}_{t-1,1}, \mathbf{x}_{t,1}\rangle & 0 & \cdots & 0 \\ 0 & \|\mathbf{v}_{t-1,2}\|^{2(p-1)}\langle \mathbf{v}_{t-1,2}, \mathbf{x}_{t,2}\rangle & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \|\mathbf{v}_{t-1,K}\|^{2(p-1)}\langle \mathbf{v}_{t-1,K}, \mathbf{x}_{t,K}\rangle \end{bmatrix}
\end{aligned}$$

so that:

$$\text{Tr}\left(\left(\mathbf{V}'_{t-1}\mathbf{V}_{t-1}\right)^{p-1}\mathbf{V}'_{t-1}\mathbf{X}_t\right) = \sum_{k=1}^{K}\|\mathbf{v}_{t-1,k}\|^{2(p-1)}\langle \mathbf{v}_{t-1,k}, \mathbf{x}_{t,k}\rangle$$

and finally:

$$\hat{y}_t = \text{sign}\left(\sum_{k=1}^{K}\|\mathbf{v}_{t-1,k}\|^{2(p-1)}\langle \mathbf{v}_{t-1,k}, \mathbf{x}_{t,k}\rangle\right) \tag{4.1.6}$$

This last equation is exactly what we obtain using (3.1.15). Indeed it is easy to see that whenever $\mathbf{V}$ is an orthogonal matrix:

$$\mathbf{V}\mathbf{V}'\mathbf{v}_j = \sum_{k=1}^{K}\mathbf{v}_k\mathbf{v}'_k\mathbf{v}_j = \sum_{k=1}^{K}\mathbf{v}_k\langle \mathbf{v}_k, \mathbf{v}_j\rangle = \mathbf{v}_j\|\mathbf{v}_j\|^2$$

and

$$\left(\mathbf{V}\mathbf{V}'\right)^{p-1}\mathbf{v}_j = \|\mathbf{v}_j\|^{2(p-1)}\mathbf{v}_j$$

therefore again:

$$\begin{aligned}
\hat{y}_t &= \text{sign}\left(\sum_{k=1}^{K}\left\langle\left(\mathbf{V}_{t-1}\mathbf{V}'_{t-1}\right)^{p-1}\mathbf{v}_{t-1,k}, \mathbf{x}_{t,k}\right\rangle\right) \\
&= \text{sign}\left(\sum_{k=1}^{K}\|\mathbf{v}_{t-1,k}\|^{2(p-1)}\langle \mathbf{v}_{t-1,k}, \mathbf{x}_{t,k}\rangle\right) \qquad (4.1.7)
\end{aligned}$$

We thus see that orthogonal matrix Perceptron prediction is equal to the sum of single-view prediction margins, each one weighted with a power of the norm of the single-view primal weight. The resulting algorithm is shown in Figure 4.1.

Assume $\mathbf{x}_{t,k}, \mathbf{v}_{t,k} \in \mathcal{X}_k \subseteq \mathbb{R}^{d_k}$, for $k = \{1, ..K\}$.
Initialization $\mathbf{v}_{k,0} = \mathbf{0}$, for $k = \{1, ..K\}$.
At each time t=1,2,..... do the following:

1: Observe K instance vectors $\mathbf{x}_{t,k} \in \mathcal{X}_k$;
2: Predict label $y_t \in \{-1, 1\}$ with:
$\hat{y}_t = \text{sign}\left(\sum_{k=1}^{K} \|\mathbf{v}_{t-1,k}\|^{2(p-1)} \langle \mathbf{v}_{t-1,k}, \mathbf{x}_{t,k} \rangle\right)$;
3: Observe actual label $y_t \in \{-1, 1\}$;
4: Update the weight vectors with:
$\mathbf{v}_{k,t} = \mathbf{v}_{k,t-1} + y_t \mathbf{x}_{k,t} \mathbb{I}_{\{\hat{y}_t \neq y_t\}}$

Figure 4.1: Orthogonal matrix multi-view Perceptron algorithm

Problems addressed As we can see in (4.1.7), our new prediction formula involves K inner products between vectors, instead of matrices. Therefore the computational complexity in time of every prediction of our algorithm is decreased to $O(pdK)$ and it is now possible to use it in problems with an high number of views. Moreover the algorithm obtained in this way does not compute inner products between different views. This means that different views are now free to lie in different spaces and if the learning problem has K different views, we have to choose only K kernel functions, one for each view, instead of K^2 kernel functions. In this way we also avoid the problem of how to choose a similarity measure between different views.

Orthogonal matrix Perceptron algorithm thus addresses the problems underlined at the end of previous Chapter, however in order to understand if our solution is really successful we have to measure its classification performances compared to the original algorithm and to other baselines. With this goal in mind in the following part of the thesis we will provide some experimental results in different settings.

4.2 Experiments

In order to assess the performances obtained by the modified algorithm we have tested it on some datasets, comparing it to few significant baselines. All the algorithms have been implemented and tested in MATLAB.

4.2.1 Adult dataset

We run our first experiment on the *a9a Adult* dataset (preprocessed by (Platt, 1998) for binary classification) which consists of 32561 sparse sample vectors in $\mathbb{R}^{123}$. Since this dataset is not multi-view, we processed it again in order to artificially obtain a multi-view dataset in the following way: every sparse and binary feature vector $\mathbf{x}_t \in \mathbb{R}^{123}$ is projected it into $\mathbb{R}^{122}$ and then split into two vectors $\mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^{61}$. From these two vectors we build a matrix $\mathbf{X} = \left[\mathbf{x}_1, \mathbf{x}_2\right] \in \mathbb{R}^{61\times 2}$. In this way we artificially obtain a multi-view dataset where the two views belonged to the same space, so that we could test the original algorithm and compare its performances against our orthogonal version. We run this experiment on 10 different permutations of

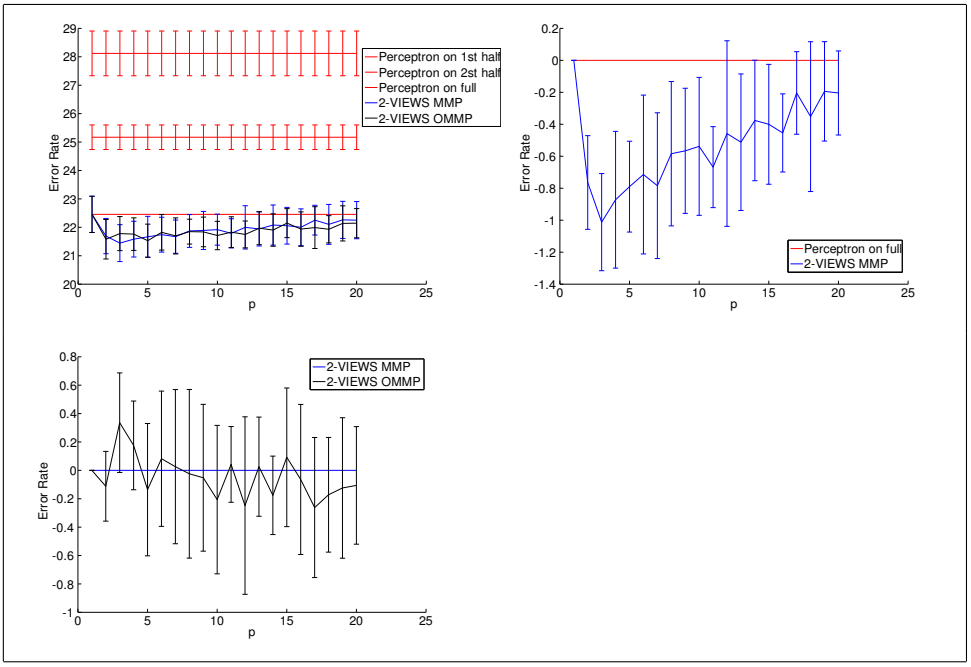


Figure 4.2: Adult a9a. Error Rate with linear kernel.

the order of the instances and for $p = \{1, 2, .., 20\}$. Moreover we used two different kernels:

- linear kernel $\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}'\mathbf{y}$
- Gaussian kernel $\langle \mathbf{x}, \mathbf{y} \rangle = \exp(-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{2\sigma^2})$ (see Theorem 2.2.9)

Results are shown in Figures 4.2 and 4.3, *MMP* stands for "*Matrix Multi-view Perceptron*", while *OMMP* stands for "*Orthogonal Matrix Multi-view Perceptron*". In both the Figures the first plot compares the ER of Perceptron run on the two single-view vectors, Perceptron run on the concatenated single-view vector $\mathbf{x}_c = \left[\mathbf{x}_1', \mathbf{x}_2'\right]'$, MMP and OMMP. In this graph we see that with the proper choice of p both MMP and OMMP can improve the performances of Perceptron. Moreover it is not possible to distinguish MMP performances from OMMP.

In the second and third graphs in order to better discriminate the behavior of the algorithms, variances are computed on the difference between couples of forecasters; this kind of computation diminish the effect of the variance due to the permutations. The second plot compares the ER ob-

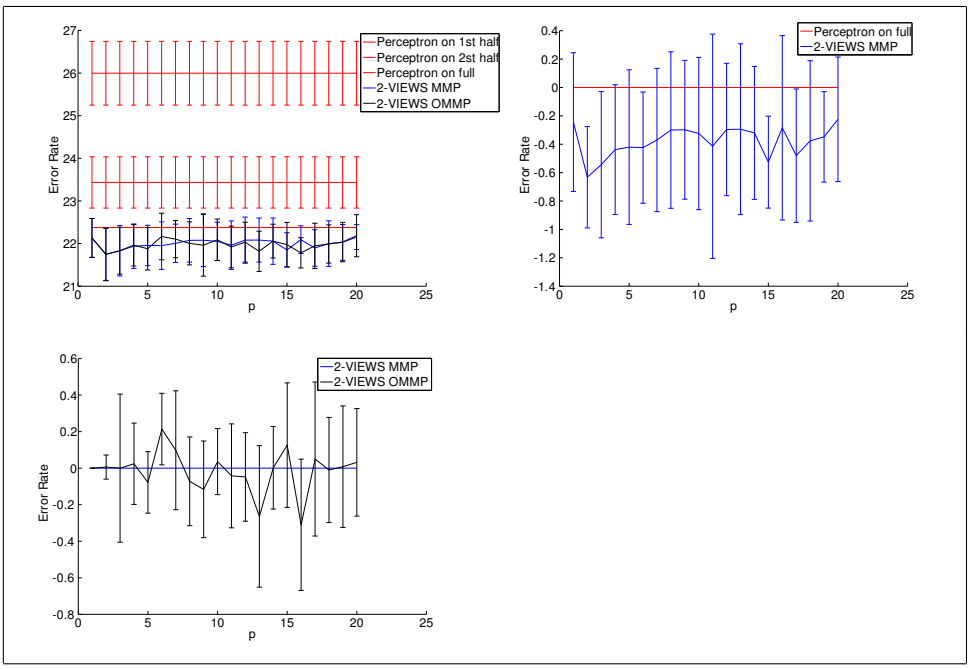


Figure 4.3: Adult a9a. Error Rate with Gaussian kernel.

tained by Perceptron and MMP, showing that with the linear kernel MMP can gain as much as 1%, while with the Gaussian kernel its gain is slightly lower. Finally the third plot compares the performances obtained by MMP and OMMP. Even in this last plot we see that it is not possible to significantly discern the behavior of the two multi-view algorithms. Therefore, with respect to this classification task we can say that our algorithm is a good approximation of the original MMP.

4.2.2 News20 dataset

News20 dataset is a very sparse binary dataset which has been adapted to the binary classification task by (Keerthi and DeCoste, 2005). Being composed of 19996 binary instances in $\mathbb{R}^{1355190}$, the dataset is perfect for linear-threshold algorithms that can take advantage of the high dimensionality of the space. As in case of Adult dataset we used it to artificially create a multi-view dataset and to compare the performances of Perceptron on every single view, Perceptron on the full vector, MMP and OMMP. We run the experiment on 5 different permutations of the order of the instances

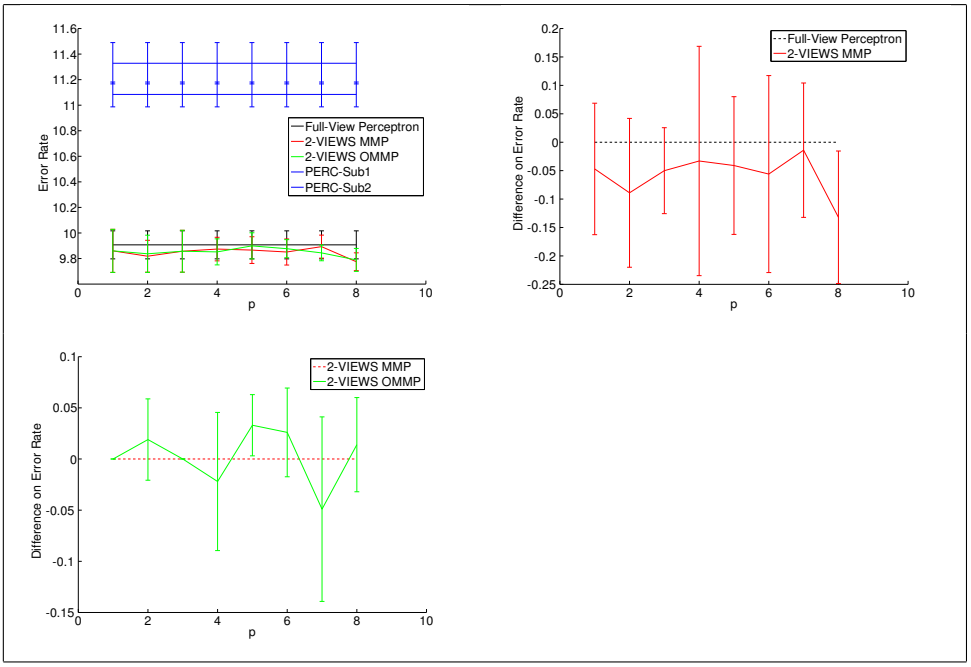


Figure 4.4: News20. 2-views Error Rate with linear kernel

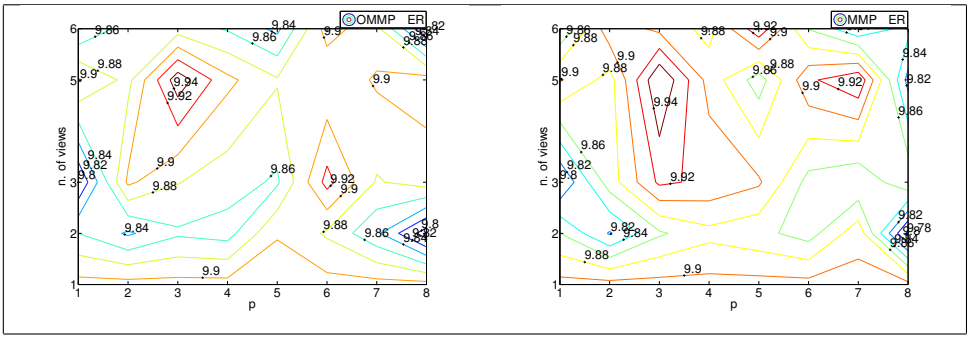


Figure 4.5: News20. Error Rate with linear kernel, with respect to the number of splits (views) and to p.

and for $p = \{1, 2, .., 8\}$, using only the linear kernel. Moreover this time we tested the dataset by splitting every original instance vector into 2, 3, 5 and 6 single-view vectors. Since in this case the original instance vectors were normalized, we also re-normalized the single-view vectors resulting from the splits. This is the reason why in this case the performance obtained by MMP and OMMP with $p = 1$ and linear kernel are not equal to those obtained by Perceptron on the full-length vector.

In Figure 4.4 we plot the results for the 2-views case. As in the case of Adult dataset, there is no appreciable difference in the performances obtained by MMP and OMMP, moreover in this setting there is also no appreciable gain in performances with respect to the Perceptron baseline. This situation does not change by modifying the number of views, as can be seen in Figure 4.5 where we plot the variation in the ER with respect to the number of views and p.

4.2.3 Eth-80 dataset

Eth-80 is a dataset for object categorization introduced in (Leibe and Schiele, 2003). It contains 80 objects from 8 categories: apples, pears, tomatoes, cows, dogs, horses, cups and cars, moreover for each category 10 different objects are provided and each object is represented by 41 images from different viewpoints.

In this experiment we tried to directly compare our multi-view algorithm to the Online Multi-Cue Learning (*OMCL*) algorithm (Jie et al.,

Figure 4.6: Eth-80 dataset of pictures

2009), which was proved to achieve good categorization performances on this dataset. OMCL adopts an high-level integration scheme by combining two levels of classifiers: at the first level a Projectron++ (Orabona et al., 2008) is trained for every view, while on the second level a Passive-Aggressive Online algorithm (Crammer et al., 2006) is used to optimally combine the confidence measures obtained by the classifiers on the first level.

Following the approach introduced in (Jie et al., 2009), every image was represented by four descriptors: one color feature (RGB color histogram), two texture descriptors (Composed Receptive Field Histogram (Linde and Lindeberg, 2004) with two different kinds of filters: Gaussian derivative $L_x L_y$ and gradient direction $DirC$) and a global shape feature (centered masks). The features extracted in this way lie in spaces of different dimensionality, so that for this experiment we could not make use of the original MMP algorithm. Therefore we used our OMMP algorithm with four different kernels applied to the four different views: a Gaussian kernel for the shape feature and three chi^2 kernels (Fowlkes et al., 2004) for the remaining views. The parameters of each kernel function were estimated by cross-

validation [Chapter 1](Bishop, 2006).
Finally, in order to stress the generalization abilities of our algorithm, we applied the technique explained in section 3.1.3 to extract a low risk hypothesis from a training set and assess the performances of this hypothesis on a test set.

Since OMMP is a binary classification algorithm we considered only binary classification problems, specifically we selected the following binary classification problems:

- horses and pears
- cows and dogs

In so doing there is an equal number of samples for each class and the binary learning tasks are balanced. For each problem we had $2 * 10 * 41 = 820$ samples that we divided into a training set containing 70% of the samples and a test set containing the remaining samples, moreover we run every experiment on 10 different random permutations of the samples and for $p = \{1, 2, .., 20\}$.

The first problem was chosen for its simplicity: horse and pears should be easily discriminated. However after having seen the results of this experiment we realized that the problem was too simple, indeed the color and shape views obtained an ER near to zero, while the worst view achieved an ER below 14% in the training phase. Therefore we tried the more difficult task to discriminate cows and dogs. As we can see in Figure 4.9, this task is significantly more difficult than the previous one: the worst view obtains an ER near to 40%, while other two views achieve an ER near to 25%. However the color view still obtains a very low ER also for this problem.

The results of the experiments are summarized in Figures 4.7, 4.8, 4.9 and 4.10. Every Figure is divided into three lines: in the first line we plot the performances obtained by OMMP using two, three and four views, and the performances obtained by the same algorithm using only the single views. In the second line we plot the performances of OMMP compared to OMCL using two and four views, while in the third and last line we compare the performances obtained using two or four views in OMMP, to the performances obtained by the best of the related single-views.

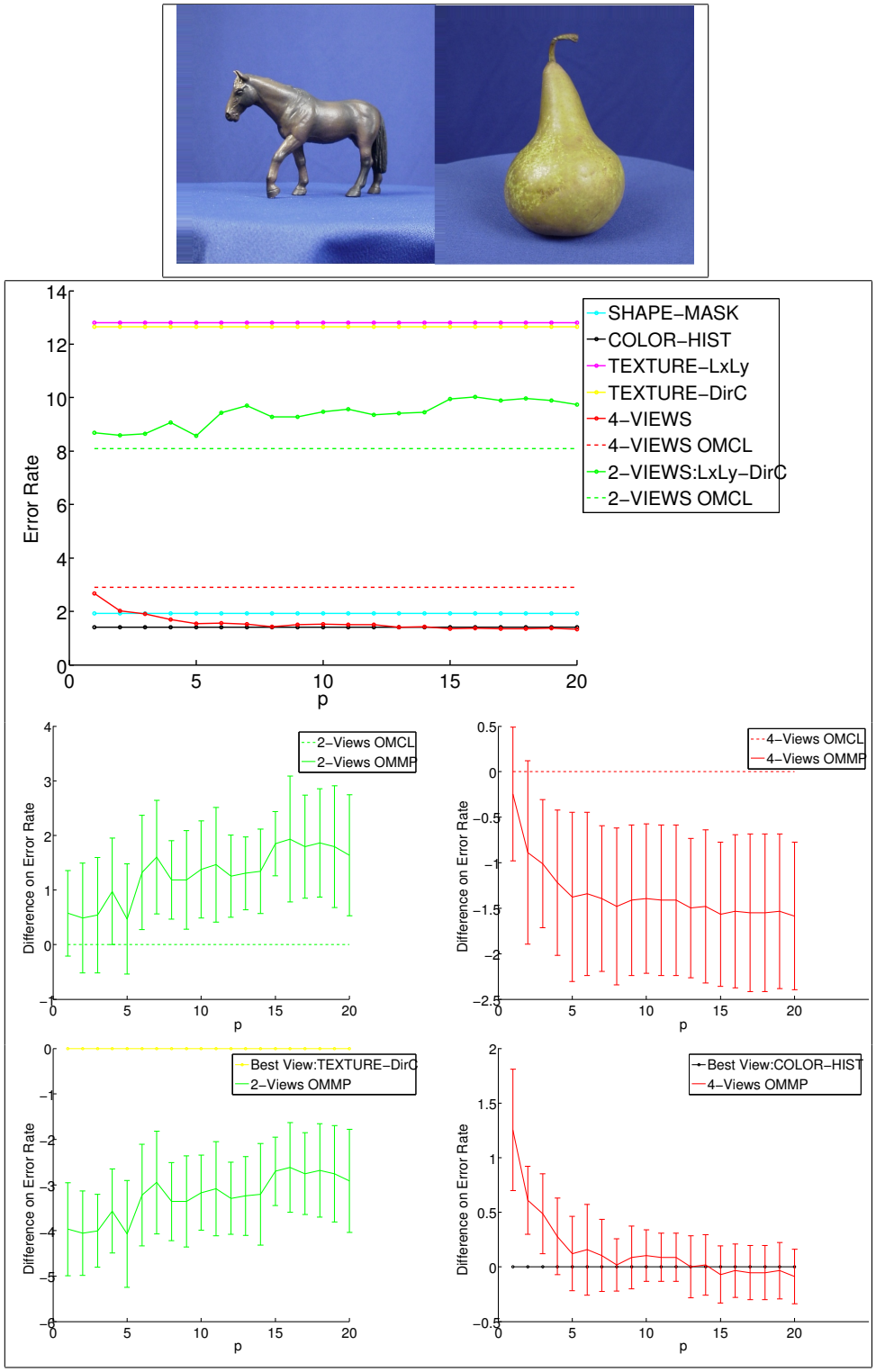


Figure 4.7: Eth-80, horses and pears Training Error Rate

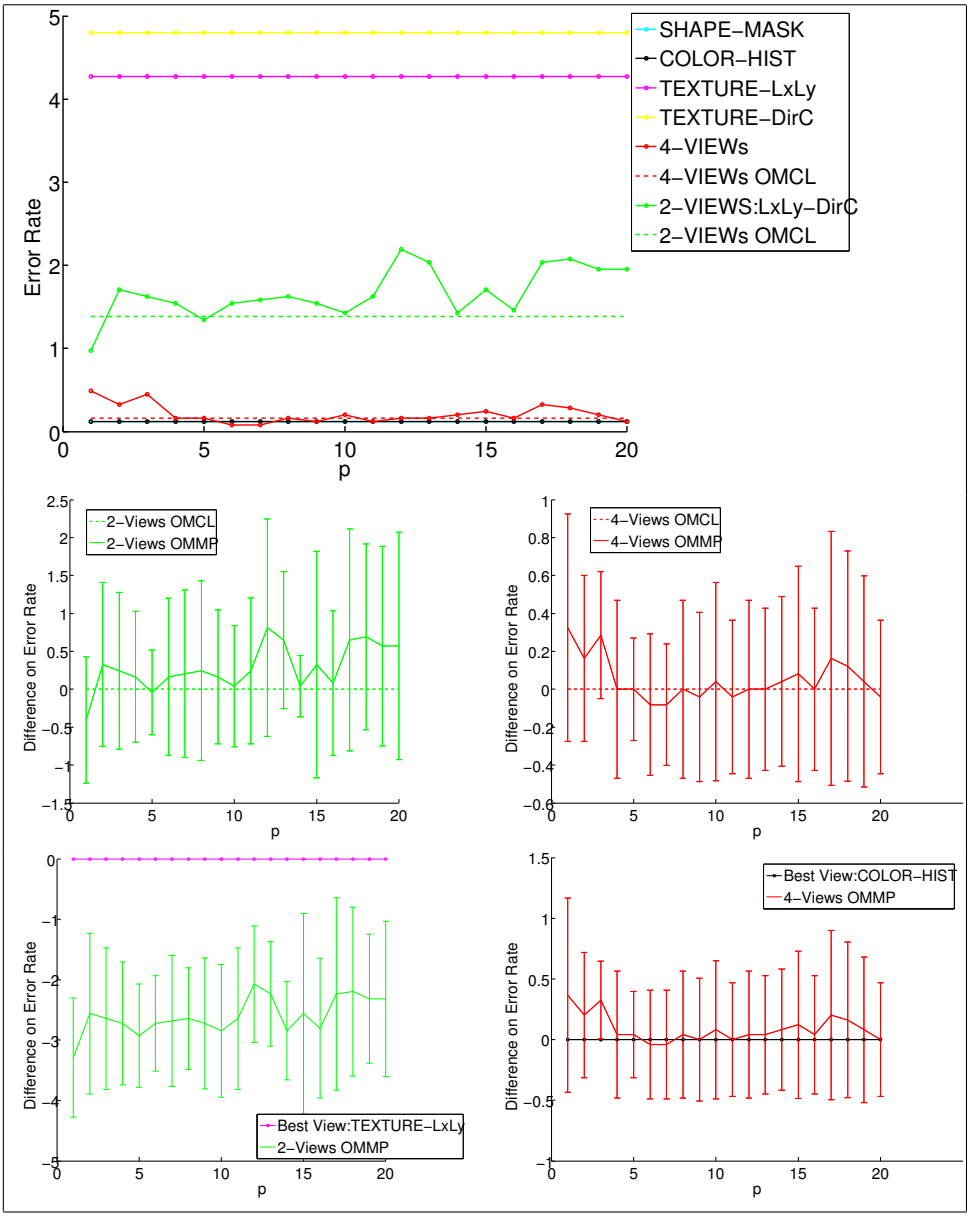


Figure 4.8: Eth-80, horses and pears Testing Error Rate

In the horses and pears experiment the texture views obtain very similar "bad" performances. However by using them together, both OMMP and OMCL algorithms perform much better. With respect to this two views and only in the training phase OMCL algorithm slightly outperforms OMMP. Finally, by using together all the four views, both algorithms do not succeed in improving the almost perfect classification obtained by the color view.

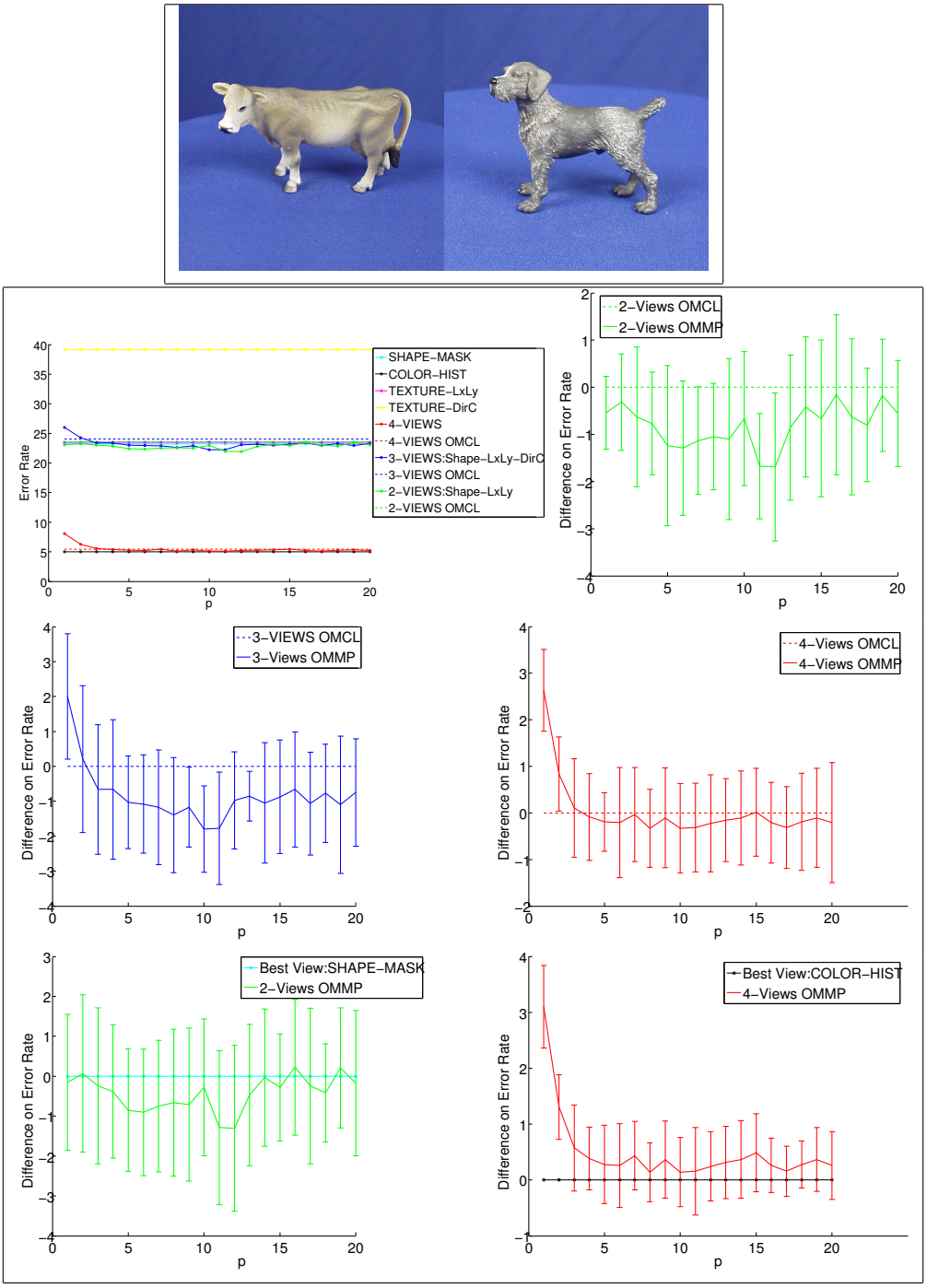


Figure 4.9: Eth-80, cows and dogs Training Error Rate

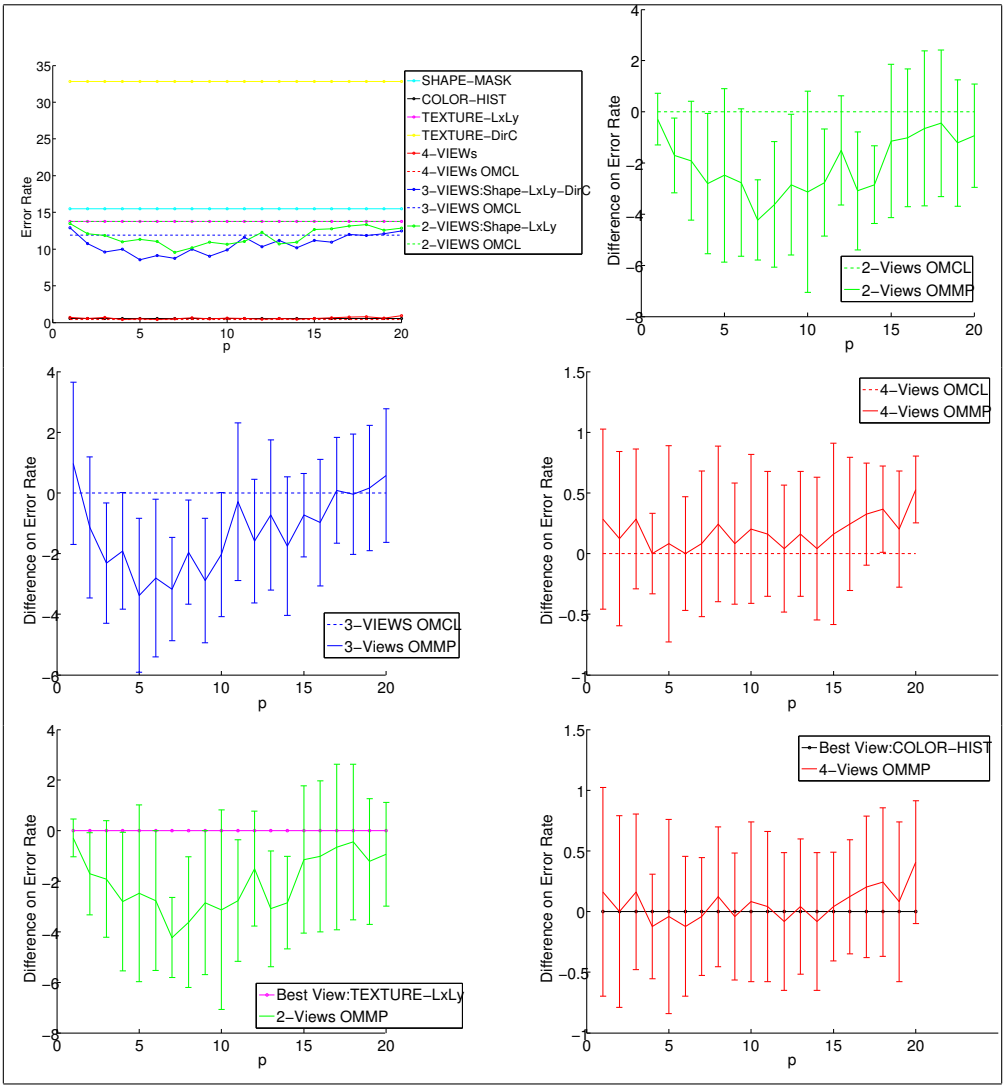


Figure 4.10: Eth-80, cows and dogs Testing Error Rate

In the cows and dogs experiments the texture view with Gaussian derivative $L_x L_y$ and the shape view obtain very similar performances. In the training phase, with respect to these views both OMMP and OMCL fail to significantly improve the performances obtained by the best of the single views. However, in the test phase and for the the correct value of p, OMMP can obtain significantly improved performances with respect to the best single view and to OMCL. Finally, by considering all the four views, both the OMCL and the OMMP algorithms fail to improve the performances ob-

tained by the best single view, which again in the testing phase reaches the almost perfect categorization performance.

Note also that if $\mathbf{X}, \mathbf{V} \in \mathbb{R}^{d\times 1}$ (that is if the multi-view instance matrices are actually single-view vectors) we have:

$$\begin{aligned}\langle \nabla\Phi(\mathbf{V}), \mathbf{X}\rangle &= \|\mathbf{V}\|_{S_{2p}}^{2(1-p)} \operatorname{Tr}\left(\left(\mathbf{V}'\mathbf{V}\right)^{p-1}\mathbf{V}'\mathbf{X}\right) = \|\mathbf{v}\|_2^{2(1-p)} \operatorname{Tr}\left(\|\mathbf{v}\|_2^{2(p-1)}\langle \mathbf{v}, \mathbf{x}\rangle\right) \\ &= \|\mathbf{v}\|_2^{2(1-p)+2(p-1)}\langle \mathbf{v}, \mathbf{x}\rangle = \langle \mathbf{v}, \mathbf{x}\rangle\end{aligned}$$

where $\mathbf{x}, \mathbf{v} \in \mathbb{R}^d$ are the vectors corresponding to $\mathbf{X}, \mathbf{V}$ and $\|\cdot\|_2$ is the usual vector norm. When matrix Perceptron is run with just one view, p becomes thus immaterial and matrix Perceptron algorithm turns out to be exactly Perceptron algorithm. This is the reason why with respect to the single view problems, matrix Perceptron performance is constant with respect to p.

4.3 Discussion of the results

In this Chapter we have presented a variant of the MMP algorithm aimed to solve the problems underlined at the end of the previous Chapter. Moreover we have measured the performances obtained by the resulting OMMP algorithm in several different classification tasks.

In the first two experiments (namely on the Adult and News20 datasets) we have seen that OMMP performances are not discernible from the performances obtained by the original MMP algorithm. This fact could be due to the sparsity of the instance vectors: when two vectors are highly sparse, we could expect their inner product to be near to zero.

On the other hand, in the last experiment we compared the OMMP algorithm to another multi-view algorithm called OMCL. The results of this experiment have highlighted that whenever there is a view that achieves almost perfect classification, both the OMMP and OMCL algorithms using all the four views fail to improve the performances obtained by the best view. However, if the views are more balanced, the two algorithms can actually improve the performances obtained by the best of these views. The comparison between OMMP and OMCL is interesting because the two algorithms stems from two completely different approaches, while still achieving similar classification performances.

Chapter 5

Conclusions and future works

The research work undertaken during these months has helped to shed some more light on the strength and weakness points of the multi-view learning approach adopted by Cavallanti et al. (2008). We have seen that even if MMP algorithm has some bad limitations such the constraints on the dimensionality of the views, it is possible to derive a modified and simplified version of the algorithm that overcomes the limits of the original one. This version of the algorithm, called OMMP is based on a simple orthogonalization technique and can be seen as as specific instance of the original algorithm, when all the instance matrices are supposed to share a common structure (namely the orthogonal form). In this way we obtain an algorithm that can efficiently manage many different views lying in different spaces and that can be formulated in a dual version, allowing the substitution of one kernel function for each view. We have also provided some analytical insights on the behavior of the two algorithms with respect to the single-view vectors and we have successfully applied an on-line to batch conversion technique to the dual formulated multi-view algorithms. Finally we have also compared the performances obtained by the two algorithms and the performances obtained by a third multi-view algorithm Jie et al. (2009). Even if OMMP has obtained good classification performances, compared to both MMP and OMCL, weak experimental results shows that the problem of learning from multiple views is still far from being solved. Hereby we state some of the possible future research directions that we would like to follow:

Bound Even if we have successfully proved some useful theorems related to MMP and OMMP, we still don't have a specific mistake bound for the last algorithm. The only bound applicable to it is the mistake bound of MMP. However we don't have a specific analysis able to describe the behavior of the OMMP algorithm and to bound the number of mistakes. Therefore we cannot theoretically compare the performances obtained by the original MMP algorithm to the performance obtained by the OMMP algorithm. One natural task at this point should thus be to try to make a vectorial analysis of the algorithm in order to obtain a specific mistake bound.

Aggressive updates It is known that it is sometimes possible to improve the classification performances obtained by an on-line classifier, by optimally tuning a learning rate in order to update the weight vector more aggressively, whenever the margin is below a certain threshold (see (Shalev-Shwartz and Singer, 2005) and (Crammer et al., 2006)). While the OMCL algorithm adopts an aggressive strategy for his top level classifier, MMP and OMMP are instead forced to keep a conservative policy. In order to better compete with OMCL and to assess how an aggressive policy could be helpful also in matrix multi-view classification tasks, we have already conducted some researches on how to optimally tune a learning rate for the aggressive policy. However in order to reach the goal there is still some work to do..

Leaving the span Warmuth and Vishwanathan (2006) have shown that some simple sparse linear problem are hard to learn with any algorithm that uses a linear combination of the training instances as its weight vector. However the same problems can be efficiently learned using the Exponentiated Gradient (Kivinen and Warmuth, 1997) algorithm, which basically seeks its solutions in a space larger than the space spanned by the training vectors. In Appendix A.2 we prove that if instances are of a certain form and we artificially add an orthogonal component $\mathbf{V}_\perp$ to the primal weight matrix $\mathbf{V}$, we can affect matrix Perceptron prediction, through the norm of the single-view primal weight vectors $\mathbf{v}_{k\perp}$. Since these vectors are not in the span, they cannot be learned by matrix Perceptron. *Is it possible to improve matrix Perceptron predictions by adding a orthogonal components to*

the weight vectors? Is it possible to learn these orthogonal components, or at least their norm? It would be interesting to try to answer these questions.

Feature vectors combinations Up to now we have explored the possibilities opened by the orthogonalization technique introduced in this Chapter. However, orthogonalization is not the only possible approach to address some of the problems highlighted at the end of Chapter 3. For example if we do not take into account the efficiency problems and the problem of how to chose a similarity measure between different views, we could experiment other approaches such as repetition or zero-padding of the short vectors (or part of te vectors), in order to reach the length of the longest vector.

Group Norms Last but not least we would like to deepen also the relationships between the our algorithm and the framework based on the group norms introduced by Kakade et al. (2009).

Appendix A

A.1 Fourier series in Hilbert spaces

We hereby introduce the definition of *Fourier series* of an element in an Hilbert space.

A.1 Fourier series. If $\mathcal{H}$ is an Hilbert space and $S = \{\mathbf{v}_1, \mathbf{v}_2, ...\mathbf{v}_K : \mathbf{v}_i \in \mathcal{H}\}$ is a set in $\mathcal{H}$, we define the transforming function $\mathfrak{F} : S \to \mathcal{H}$ as

$$\mathfrak{F}(\mathbf{x}_k) = \sum_{i=1}^{K} \langle \mathbf{x}_k, \mathbf{v}_i \rangle \mathbf{v}_i \qquad \mathbf{x}_k \in \text{span}(S) \tag{A.1}$$

Whenever S is an orthonormal set, this transformation is called *Fourier series of* $\mathbf{v}_k$ *by* S.
In the orthonormal case, for every $\mathbf{x}_k \in \text{span}(S)$ we have:

$$\mathfrak{F}(\mathbf{x}_k) = \sum_{i=1}^{K} \langle \mathbf{x}_k, \mathbf{v}_i \rangle \mathbf{v}_i = \mathbf{x}_k$$

as we see in the following examples:

- in general, since $\mathbf{x}_k \in S$, it can be written as $\mathbf{x}_k = \sum_{j=1}^{K} \alpha_j \mathbf{v}_j$, so that:

$$\mathfrak{F}(\mathbf{x}_k) = \sum_{i=1}^{K} \left\langle \sum_{j=1}^{K} \alpha_j \mathbf{v}_j, \mathbf{v}_i \right\rangle \mathbf{v}_i = \sum_{j=1}^{K} \alpha_j \sum_{i=1}^{K} \langle \mathbf{v}_j, \mathbf{v}_i \rangle \mathbf{v}_i = \sum_{j=1}^{K} \alpha_j \mathbf{v}_j = \mathbf{x}_k$$

- if $\mathcal{H}$ is an RKHS, $S = \{k(\mathbf{x}_1, \cdot), k(\mathbf{x}_2, \cdot), ..., k(\mathbf{x}_K, \cdot)\}$ is an orthonormal set of functions and $f \in \text{span}(S)$ (that is $f(\cdot) = \sum_{j=1}^{K} c_j k(\mathbf{x}_j, \cdot)$), we

have:

$$\mathfrak{F}(f) = \sum_{i=1}^{K} \langle f, k(\mathbf{x}_i, \cdot)\rangle k(\mathbf{x}_i, \cdot) = \sum_{i=1}^{K}\sum_{j=1}^{K} c_j \langle k(\mathbf{x}_j, \cdot), k(\mathbf{x}_i, \cdot)\rangle k(\mathbf{x}_i, \cdot)$$
$$= \sum_{i=1}^{K}\sum_{j=1}^{K} c_j k(\mathbf{x}_j, \mathbf{x}_i) k(\mathbf{x}_i, \cdot) = \sum_{i=1}^{K} c_i k(\mathbf{x}_i, \cdot) = f$$

- if $\mathcal{H} = L_2([-\pi, \pi])$ with the usual inner product $\langle f, g\rangle = \int_{-\pi}^{\pi} f\overline{g}\, dx$ and the set of orthonormal functions $e_n(x) = e^{inx}$ we have

$$\mathfrak{F}(f) = \sum_{n=-\infty}^{\infty} \langle f, e_n\rangle\, e_n = \sum_{n=-\infty}^{\infty} \left(\int_{-\pi}^{\pi} f e^{-inx}\, dx\right) e^{inx} = f$$

  the usual Fourier series definition.

- if $\mathcal{H}$ is a subspace of $\mathbb{R}^d$ of size K, with the usual inner product $\langle \mathbf{x}, \mathbf{y}\rangle = \mathbf{x}'\mathbf{y} = \mathbf{y}'\mathbf{x}$, we have:

$$\mathfrak{F}(\mathbf{v}_k) = \sum_{i=1}^{K} \mathbf{v}_i \langle \mathbf{v}_k, \mathbf{v}_i\rangle = \left(\sum_{i=1}^{K} \mathbf{v}_i \mathbf{v}_i'\right) \mathbf{v}_k = (\mathbf{V}\mathbf{V}')\mathbf{v}_k \tag{A.2}$$

  where $\mathbf{V} = \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_K \end{bmatrix}$. If the set $\{\mathbf{v}_1, \mathbf{v}_2, ..., \mathbf{v}_K\}$ is orthonormal, then $\mathbf{V}\mathbf{V}' = \mathbf{I}_K$, so that $\mathfrak{F}(\mathbf{v}_k) = \mathbf{V}\mathbf{V}'\mathbf{v}_k = \mathbf{v}_k$.

A.2 Lemma

In MMP algorithm, as we have previously seen, if $S = \text{span}(\mathbf{X}_1, \mathbf{X}_2, ...)$, the primal weight matrix $\mathbf{V}$ belongs to S by construction. However, by artificially adding an orthogonal component $\mathbf{V}_\perp$ to $\mathbf{V}$ we can affect matrix Perceptron prediction. Specifically, if $p \neq 1$ and the orthogonal part is of a certain type, it affects the prediction only through the norm of its single-view components, as proved in the following lemma.

A.2.1 Lemma. *Let instance matrices $\bar{\mathbf{X}}_t$ be of the form shown in (A.2) and $S = \text{span}(\bar{\mathbf{X}}_1, \bar{\mathbf{X}}_2, ...)$. If we artificially add a matrix $\bar{\mathbf{V}}_\perp \notin S$ of the form shown in (A.4) to the primal weight matrix $\mathbf{V} \in S$, matrix Perceptron prediction is then equal to:*

$$\text{sign}\left(\sum_{k=1}^{K} \langle \mathbf{x}_k, \mathbf{v}_k\rangle \left(\|\mathbf{v}_k\|^2 + \|\bar{\mathbf{v}}_{k\perp}\|^2\right)^{p-1}\right) \tag{A.1}$$

where $\bar{\mathbf{v}}_{k\perp}$ are the single-view components of $\bar{\mathbf{V}}_{\perp}$.

Proof. The proof of the lemma will follow the following path:

1. we will restrict our attention to the case in which instance matrices are of the form shown in (A.2), which is the form shown in (4.1.1) projected into an higher dimensional space;

2. we will restrict the space of solutions to the space of matrices composed by a part that lies in the span of the instance matrices, plus an orthogonal part of a certain specific form

3. we will prove that in this particular case the orthogonal components of the single-view primal weights change matrix Perceptron prediction in the way shown in the statement of the Theorem.

We consider instance matrices $\bar{\mathbf{X}}_t \in \mathbb{R}^{D\times K}$ of the form:

$$\bar{\mathbf{X}}_t = \begin{bmatrix} \tilde{\mathbf{X}}_{D_1} \\ \tilde{\mathbf{0}}_{D_2} \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{x}}_1 & \bar{\mathbf{x}}_2 & \cdots & \bar{\mathbf{x}}_K \end{bmatrix} \tag{A.2}$$

where:

$$\tilde{\mathbf{X}}_{D_1} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{0}_1 & \cdots & \mathbf{0}_1 \\ \mathbf{0}_2 & \mathbf{x}_2 & \cdots & \mathbf{0}_2 \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0}_K & \mathbf{0}_K & \cdots & \mathbf{x}_K \end{bmatrix} = \begin{bmatrix} \tilde{\mathbf{x}}_1 & \tilde{\mathbf{x}}_2 & \cdots & \tilde{\mathbf{x}}_K \end{bmatrix}$$

is the orthogonal matrix $\mathbb{R}^{D_1\times K}$ constructed as in (4.1.1), $\mathbf{x}_k \in \mathbb{R}^{d_k}$, $\mathbf{0}_k$ is a zero vector in $\mathbb{R}^{d_k}$, $\tilde{\mathbf{0}}_{D_2}$ is a zero matrix in $\mathbb{R}^{D_2\times K}$, $D_1 = \sum_{k=1}^{K} d_k$ and $D = D_1 + D_2$, $\tilde{\mathbf{x}}_{\mathbf{k}} \in \mathbb{R}^{D_1}$ and $\bar{\mathbf{x}}_k \in \mathbb{R}^{D}$.

If all the instance matrices share the form of shown in (A.2), every matrix in the span of the instance matrices will have the same form. Therefore if a matrix $\bar{\mathbf{V}}_{\|}$ is in the span, it can be written as:

$$\bar{\mathbf{V}}_{\|} = \begin{bmatrix} \tilde{\mathbf{V}}_{D_1} \\ \tilde{\mathbf{0}}_{D_2} \end{bmatrix} = \begin{bmatrix} \tilde{\mathbf{v}}_1 & \tilde{\mathbf{v}}_2 & \cdots & \tilde{\mathbf{v}}_K \\ \mathbf{0}_{D_2} & \mathbf{0}_{D_2} & \cdots & \mathbf{0}_{D_2} \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{v}}_{1\|} & \bar{\mathbf{v}}_{2\|} & \cdots & \bar{\mathbf{v}}_{K\|} \end{bmatrix} \tag{A.3}$$

where

$$\tilde{\mathbf{V}}_{D_1} = \begin{bmatrix} \mathbf{v}_1 & \mathbf{0}_1 & \cdots & \mathbf{0}_1 \\ \mathbf{0}_2 & \mathbf{v}_2 & \cdots & \mathbf{0}_2 \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0}_K & \mathbf{0}_K & \cdots & \mathbf{v}_K \end{bmatrix}$$

we now define the space of matrices $\bar{\mathbf{V}}_\perp \in \mathbb{R}^{D\times K}$ of the following form:

$$\bar{\mathbf{V}}_\perp = \begin{bmatrix} \tilde{\mathbf{0}}_{D_1} \\ \dot{\tilde{\mathbf{V}}}_{D_2} \end{bmatrix} = \begin{bmatrix} \mathbf{0}_{D_1} & \mathbf{0}_{D_1} & \cdots & \mathbf{0}_{D_1} \\ \dot{\tilde{\mathbf{v}}}_1 & \dot{\tilde{\mathbf{v}}}_2 & \cdots & \dot{\tilde{\mathbf{v}}}_K \end{bmatrix} = \begin{bmatrix} \bar{\mathbf{v}}_{1\perp} & \bar{\mathbf{v}}_{2\perp} & \bar{\mathbf{v}}_{K\perp} \end{bmatrix} \tag{A.4}$$

where:

$$\dot{\tilde{\mathbf{V}}}_{D_2} = \begin{bmatrix} \dot{\mathbf{v}}_1 & \dot{\mathbf{0}}_1 & \cdots & \dot{\mathbf{0}}_1 \\ \dot{\mathbf{0}}_2 & \dot{\mathbf{v}}_2 & \cdots & \dot{\mathbf{0}}_2 \\ \vdots & \vdots & \ddots & \vdots \\ \dot{\mathbf{0}}_K & \dot{\mathbf{0}}_K & \cdots & \dot{\mathbf{v}}_K \end{bmatrix}$$

is the matrix $\mathbb{R}^{D_2\times K}$ constructed as in (4.1.1), $\dot{\mathbf{v}}_k \in \mathbb{R}^{\dot{d}_k}$, $\tilde{\mathbf{0}}_{D_1}$ is a zero matrix in $\mathbb{R}^{D_1\times K}$, $D_2 = \sum_{k=1}^{K} \dot{d}_k$ and $D = D_1 + D_2$.

Finally we define $\bar{\mathbf{v}}_k = \bar{\mathbf{v}}_{k\|} + \bar{\mathbf{v}}_{k\perp}$ and $\bar{\mathbf{V}} = \begin{bmatrix} \bar{\mathbf{v}}_1 & \bar{\mathbf{v}}_2 & \cdots & \bar{\mathbf{v}}_K \end{bmatrix}$, so that:

$$\begin{aligned} \bar{\mathbf{V}} &= \begin{bmatrix} \bar{\mathbf{v}}_{1\|} + \bar{\mathbf{v}}_{1\perp} & \bar{\mathbf{v}}_{2\|} + \bar{\mathbf{v}}_{2\perp} & \cdots & \bar{\mathbf{v}}_{K\|} + \bar{\mathbf{v}}_{K\perp} \end{bmatrix} \\ &= \begin{bmatrix} \bar{\mathbf{v}}_{1\|} & \bar{\mathbf{v}}_{2\|} & \cdots & \bar{\mathbf{v}}_{K\|} \end{bmatrix} + \begin{bmatrix} \bar{\mathbf{v}}_{1\perp} & \bar{\mathbf{v}}_{2\perp} & \bar{\mathbf{v}}_{K\perp} \end{bmatrix} \\ &= \bar{\mathbf{V}}_\| + \bar{\mathbf{V}}_\perp \end{aligned}$$

From the above definitions it follows that:

$$\langle \bar{\mathbf{v}}_{k\|}, \bar{\mathbf{x}}_j \rangle = \begin{cases} \langle \mathbf{v}_k, \mathbf{x}_k \rangle & \text{if } j = k \\ 0 & \text{otherwise.} \end{cases}$$

$$\langle \tilde{\mathbf{v}}_k, \tilde{\mathbf{v}}_j \rangle = \begin{cases} \|\bar{\mathbf{v}}_{k\|}\|^2 = \|\tilde{\mathbf{v}}_k\|^2 = \|\mathbf{v}_k\|^2 & \text{if } j = k \\ 0 & \text{otherwise.} \end{cases}$$

$$\langle \dot{\tilde{\mathbf{v}}}_k, \dot{\tilde{\mathbf{v}}}_j \rangle = \begin{cases} \|\bar{\mathbf{v}}_{k\perp}\|^2 = \|\dot{\tilde{\mathbf{v}}}_k\|^2 & \text{if } j = k \\ 0 & \text{otherwise.} \end{cases}$$

$$\langle \bar{\mathbf{v}}_{k\|}, \bar{\mathbf{v}}_{j\|} \rangle = \begin{bmatrix} \tilde{\mathbf{v}}_k' & \mathbf{0}_{D_2}' \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{v}}_j \\ \mathbf{0}_{D_2} \end{bmatrix} = \langle \tilde{\mathbf{v}}_k, \tilde{\mathbf{v}}_j \rangle$$

$$\langle \bar{\mathbf{v}}_{k\perp}, \bar{\mathbf{v}}_{j\perp} \rangle = \begin{bmatrix} \mathbf{0}'_{D_1} & \dot{\tilde{\mathbf{v}}}'_k \end{bmatrix} \begin{bmatrix} \mathbf{0}_{D_1} \\ \dot{\tilde{\mathbf{v}}}_j \end{bmatrix} = \langle \dot{\tilde{\mathbf{v}}}_k, \dot{\tilde{\mathbf{v}}}_j \rangle$$

$$\left\langle \bar{\mathbf{v}}_{k\|}, \bar{\mathbf{v}}_{j\perp} \right\rangle = \begin{bmatrix} \tilde{\mathbf{v}}'_k & \mathbf{0}'_{D_2} \end{bmatrix} \begin{bmatrix} \mathbf{0}_{D_1} \\ \dot{\tilde{\mathbf{v}}}_j \end{bmatrix} = 0$$

Therefore, if $j \neq k$:

$$\begin{aligned} \langle \bar{\mathbf{v}}_k, \bar{\mathbf{v}}_j \rangle &= \left\langle \bar{\mathbf{v}}_{k\|} + \bar{\mathbf{v}}_{k\perp}, \bar{\mathbf{v}}_{j\|} + \bar{\mathbf{v}}_{j\perp} \right\rangle \\ &= \langle \bar{\mathbf{v}}_{k\|}, \bar{\mathbf{v}}_{j\|} \rangle + \langle \bar{\mathbf{v}}_{k\perp}, \bar{\mathbf{v}}_{j\perp} \rangle + \langle \bar{\mathbf{v}}_{k\perp}, \bar{\mathbf{v}}_{j\|} \rangle + \langle \bar{\mathbf{v}}_{k\|}, \bar{\mathbf{v}}_{j\perp} \rangle = 0 \end{aligned} \tag{A.5}$$

and

$$\left\langle \bar{\mathbf{V}}_{\|}, \bar{\mathbf{V}}_{\perp} \right\rangle = \operatorname{Tr}\left(\bar{\mathbf{V}}'_{\|} \bar{\mathbf{V}}_{\perp} \right) = \sum_{k=1}^{K} \left\langle \bar{\mathbf{v}}_{k\|}, \bar{\mathbf{v}}_{k\perp} \right\rangle = 0 \tag{A.6}$$

where the last equation states that $\bar{\mathbf{V}}_{\|}$ and $\bar{\mathbf{V}}_{\perp}$ are orthogonal, with respect to to the Frobenius inner product.

We can now show how the orthogonal components $\bar{\mathbf{v}}_{k\perp}$ change the predictions, we start by noting that:

$$\begin{aligned} \bar{\mathbf{V}}\bar{\mathbf{V}}' &= \sum_{i=1}^{K} \bar{\mathbf{v}}_i \bar{\mathbf{v}}'_i = \sum_{i=1}^{K} \left(\bar{\mathbf{v}}_{i\perp} + \bar{\mathbf{v}}_{i\|} \right) \left(\bar{\mathbf{v}}_{i\perp} + \bar{\mathbf{v}}_{i\|} \right)' \\ &= \sum_{i=1}^{K} \bar{\mathbf{v}}_{i\perp} \bar{\mathbf{v}}'_{i\perp} + \sum_{i=1}^{K} \bar{\mathbf{v}}_{i\|} \bar{\mathbf{v}}'_{i\|} + \sum_{i=1}^{K} \bar{\mathbf{v}}_{i\perp} \bar{\mathbf{v}}'_{i\|} + \sum_{i=1}^{K} \bar{\mathbf{v}}_{i\|} \bar{\mathbf{v}}'_{i\perp} \end{aligned} \tag{A.7}$$

and we compute $\bar{\mathbf{V}}\bar{\mathbf{V}}'\bar{\mathbf{v}}_k$ by the sum of:

$$\begin{aligned} \sum_{i=1}^{K} \mathbf{v}_{i\perp} \bar{\mathbf{v}}'_{i\perp} \bar{\mathbf{v}}_k &= \sum_{i=1}^{K} \bar{\mathbf{v}}_{i\perp} \bar{\mathbf{v}}'_{i\perp} \left(\bar{\mathbf{v}}_{k\perp} + \bar{\mathbf{v}}_{k\|} \right) \\ &= \sum_{i=1}^{K} \bar{\mathbf{v}}_{i\perp} \langle \bar{\mathbf{v}}_{i\perp}, \bar{\mathbf{v}}_{k\perp} \rangle + \sum_{i=1}^{K} \bar{\mathbf{v}}_{i\perp} \langle \bar{\mathbf{v}}_{i\perp}, \bar{\mathbf{v}}_{k\|} \rangle = \bar{\mathbf{v}}_{k\perp} \|\bar{\mathbf{v}}_{k\perp}\|^2 \\ \sum_{i=1}^{K} \bar{\mathbf{v}}_{i\|} \bar{\mathbf{v}}'_{i\|} \bar{\mathbf{v}}_k &= \sum_{i=1}^{K} \bar{\mathbf{v}}_{i\|} \bar{\mathbf{v}}'_{i\|} \left(\bar{\mathbf{v}}_{k\perp} + \bar{\mathbf{v}}_{k\|} \right) \\ &= \sum_{i=1}^{K} \bar{\mathbf{v}}_{i\|} \langle \bar{\mathbf{v}}_{i\|}, \bar{\mathbf{v}}_{k\perp} \rangle + \sum_{i=1}^{K} \bar{\mathbf{v}}_{i\|} \langle \bar{\mathbf{v}}_{i\|}, \bar{\mathbf{v}}_{k\|} \rangle = \bar{\mathbf{v}}_{k\|} \|\bar{\mathbf{v}}_{k\|}\|^2 \end{aligned}$$

$$
\begin{aligned}
\sum_{i=1}^{K} \bar{\mathbf{v}}_{i\perp}\bar{\mathbf{v}}'_{i\|}\bar{\mathbf{v}}_k &= \sum_{i=1}^{K} \bar{\mathbf{v}}_{i\perp}\bar{\mathbf{v}}'_{i\|}\left(\bar{\mathbf{v}}_{k\perp} + \bar{\mathbf{v}}_{k\|}\right) \\
&= \sum_{i=1}^{K} \bar{\mathbf{v}}_{i\perp}\langle \bar{\mathbf{v}}_{i\|}, \bar{\mathbf{v}}_{k\perp}\rangle + \sum_{i=1}^{K} \bar{\mathbf{v}}_{i\perp}\langle \bar{\mathbf{v}}_{i\|}, \bar{\mathbf{v}}_{k\|}\rangle = \bar{\mathbf{v}}_{k\perp}\|\bar{\mathbf{v}}_{k\|}\|^2 \\
\sum_{i=1}^{K} \bar{\mathbf{v}}_{i\|}\bar{\mathbf{v}}'_{i\perp}\bar{\mathbf{v}}_k &= \sum_{i=1}^{K} \bar{\mathbf{v}}_{i\|}\bar{\mathbf{v}}'_{i\perp}\left(\bar{\mathbf{v}}_{k\perp} + \bar{\mathbf{v}}_{k\|}\right) \\
&= \sum_{i=1}^{K} \bar{\mathbf{v}}_{i\|}\langle \bar{\mathbf{v}}_{i\perp}, \bar{\mathbf{v}}_{k\perp}\rangle + \sum_{i=1}^{K} \bar{\mathbf{v}}_{i\|}\langle \bar{\mathbf{v}}_{i\perp}, \bar{\mathbf{v}}_{k\|}\rangle = \bar{\mathbf{v}}_{k\|}\|\bar{\mathbf{v}}_{k\perp}\|^2
\end{aligned}
$$

Therefore

$$
\begin{aligned}
\bar{\mathbf{V}}\bar{\mathbf{V}}'\bar{\mathbf{v}}_k &= \bar{\mathbf{v}}_{k\perp}\|\bar{\mathbf{v}}_{k\perp}\|^2 + \bar{\mathbf{v}}_{k\|}\|\bar{\mathbf{v}}_{k\|}\|^2 + \bar{\mathbf{v}}_{k\perp}\|\bar{\mathbf{v}}_{k\|}\|^2 + \bar{\mathbf{v}}_{k\|}\|\bar{\mathbf{v}}_{k\perp}\|^2 \\
&= \bar{\mathbf{v}}_{k\|}\left(\|\bar{\mathbf{v}}_{k\|}\|^2 + \|\bar{\mathbf{v}}_{k\perp}\|^2\right) + \bar{\mathbf{v}}_{k\perp}\left(\|\bar{\mathbf{v}}_{k\perp}\|^2 + \|\bar{\mathbf{v}}_{k\|}\|^2\right) \\
&= \left(\bar{\mathbf{v}}_{k\|} + \bar{\mathbf{v}}_{k\perp}\right)\left(\|\bar{\mathbf{v}}_{k\|}\|^2 + \|\bar{\mathbf{v}}_{k\perp}\|^2\right) = \bar{\mathbf{v}}_k\left(\|\bar{\mathbf{v}}_{k\|}\|^2 + \|\bar{\mathbf{v}}_{k\perp}\|^2\right)
\end{aligned}
$$

so that in the general case we have:

$$
\left(\bar{\mathbf{V}}\bar{\mathbf{V}}'\right)^{p-1}\bar{\mathbf{v}}_k = \bar{\mathbf{v}}_k\left(\|\bar{\mathbf{v}}_{k\|}\|^2 + \|\bar{\mathbf{v}}_{k\perp}\|^2\right)^{p-1} \tag{A.8}
$$

For what we have said, if the instances are of the form of (A.2) matrix Perceptron prediction is then equal to:

$$
\begin{aligned}
\hat{y} &= \operatorname{sign}\left(\sum_{k=1}^{K} \bar{\mathbf{x}}'_k(\bar{\mathbf{V}}\bar{\mathbf{V}}')^{p-1}\bar{\mathbf{v}}_k\right) \\
&= \operatorname{sign}\left(\sum_{k=1}^{K} \langle \bar{\mathbf{x}}_k, \bar{\mathbf{v}}_{k\|} + \bar{\mathbf{v}}_{k\perp}\rangle\left(\|\bar{\mathbf{v}}_{k\|}\|^2 + \|\bar{\mathbf{v}}_{k\perp}\|^2\right)^{p-1}\right) \\
&= \operatorname{sign}\left(\sum_{k=1}^{K} \langle \mathbf{x}_k, \mathbf{v}_k\rangle\left(\|\mathbf{v}_k\|^2 + \|\bar{\mathbf{v}}_{k\perp}\|^2\right)^{p-1}\right)
\end{aligned} \tag{A.9}
$$

□

List of Figures

2.1 An Hyperplane splits the blue class from the red class. 9
2.2 Perceptron algorithm 10
2.3 Convergence of Perceptron learning updates. 11
2.4 The hinge loss (shown in blue) is convex and upperbounds the zero-one loss (in black) 15
2.5 Conservative potential-based gradient descent algorithm, for the hinge loss 21
2.6 ϕ maps the non-linearly separable dataset into a new space where it is linearly separable 22
2.7 Perceptron Dual formulation 25

3.1 Matrix multi-view Perceptron algorithm 31
3.2 Relation between the sets of support vectors S_t, at consecutive time-steps. 42

4.1 Orthogonal matrix multi-view Perceptron algorithm 50
4.2 Adult a9a. Error Rate with linear kernel. 51
4.3 Adult a9a. Error Rate with Gaussian kernel. 52
4.4 News20. 2-views Error Rate with linear kernel 53
4.5 News20. Error Rate with linear kernel, with respect to the number of splits (views) and to p. 54
4.6 Eth-80 dataset of pictures 55
4.7 Eth-80, horses and pears Training Error Rate 57
4.8 Eth-80, horses and pears Testing Error Rate 58
4.9 Eth-80, cows and dogs Training Error Rate 59
4.10 Eth-80, cows and dogs Testing Error Rate 60

Bibliography

J. Anemuller, J-H. Bach, B. Caputo, L. Jie, F. Ohl, F. Orabona, R. Vogels, D. Weinshall, and A. Zweig. Biologically motivated audio-visual cue integration for object categorization. *ICCS*, 2008.

C. M. Bishop. *Pattern Recognition and Machine Learning.* Springer, 2006.

A. Blum and T. Mitchell. Combining labeled and unlabeled data with Co-Training. *COLT*, 1998.

G. Cavallanti, N. Cesa-Bianchi, and C. Gentile. Linear algorithms for online multitask classification. *COLT*, 2008.

N. Cesa-Bianchi and G. Lugosi. *Prediction, Learning and Games.* Cambridge University Press, 2006.

N. Cesa-Bianchi, A. Conconi, and C. Gentile. On the generalization ability of on-line learning algorithms. *IEEE Transactions on Information Theory*, 50(9):2050–2057, 2004.

K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer. Online passive-aggressive algorithms. *Journal of Machine Learning Research*, 7: 551–585, 2006.

Charless Fowlkes, Serge Belongie, Fan Chung, and Jitendra Malik. Spectral grouping using the Nyström Method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26:214–225, 2004.

C. Gentile. The robustness of the p-norm algorithms. *Machine Learning*, 53:265–299, 2003.

L. Jie, B. Caputo, A. Zweig, J. Bach, and J. Anemuller. Object category detection using audio-visual cues. *ICVS*, 2008.

L. Jie, F. Orabona, and B. Caputo. An on-line framework for learning novel concepts over multiple cues. *ACCV*, 2009.

Kakade, Shai Shalev-Shwartz, and Ambuj Tewar. On the duality of strong convexity and strong smoothness: Learning applications and matrix regularization. 2009.

S. S. Keerthi and D. DeCoste. A modified finite newton method for fast solution of large scale linear SVMs. *Journal of Machine Learning Research*, 6:341–361, 2005.

J. Kivinen and M. K. Warmuth. Exponentiated gradient versus gradient descent for linear predictors. *Information and Computation*, 132:1–63, 1997.

B. Leibe and B. Schiele. Analyzing appearance and contour based methods for object categorization. *Computer Vision and Pattern Recognition*, 2003.

A. S. Lewis. The convex analysis of unitarily invariant matrix functions. *Journal of Convex Analysis*, 2:173–183, 1995.

O. Linde and T. Lindeberg. Object recognition using composed receptive field histograms of higher dimensionality. *Proc. ICPR*, 2004.

F. Orabona, J. Keshet, and B. Caputo. The Projectron: a bounded kernel-based Perceptron. *ICML*, 2008.

J. C. Platt. *Advances in Kernel Methods - Support Vector Learning.* MIT Press, 1998.

F. Rosenblatt. The Perceptron: A probabilistic model for information storage and organization in the brain. *Cornell Aeronautical Laboratory, Psychological Review*, v65, No. 6:386–408, 1958.

J. Scholkopf, B.and Smola. *Learning with Kernels.* MIT Press, 2002.

S. Shalev-Shwartz and Y. Singer. A new perspective on an old Perceptron algorithm. *COLT*, LNAI 3559:264–278, 2005.

J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

V. Sindhwani and D. S. Rosemberg. An RKHS for multi-view learning and manifold co-regularization. *ICML*, 2008.

L. G. Valiant. A theory of the learnable. *Communications of the ACM*, 27, n. 11, 1984.

Z. Wang and S. Chen. Multi-view kernel machine on single-view data. *Neurocomputing*, 72, 2009.

M. K. Warmuth and S.V.N. Vishwanathan. Leaving the Span. *Journal of the ACM*, 20:1–33, 2006.