Machine Learning: Tools, Techniques, Applications (2013-14-I) # 1

Aug. 2013

Chapter 4

Linear discriminant functions

In this chapter we look at the simplest possible boundary that can separate two classes - the straight line or equivalently the hyper-plane in higher dimensional space. The equation of any such hyper-plane will be: $g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + w_0 = 0$ where $\mathbf{w}$ is the slope or weight vector and w_0 is the intercept or threshold. g is called a *discrimination function*. Consider two points $\mathbf{x}_1$ and $\mathbf{x}_2$ on the hyper-plane. Both points satisfy $g(\mathbf{x}_1) = 0 = g(\mathbf{x}_2)$ giving:

$$\begin{aligned}
&\mathbf{w}^T\mathbf{x}_1 + w_0 = \mathbf{w}^T\mathbf{x}_2 + w_0 \\
\Rightarrow &\mathbf{w}^T(\mathbf{x}_1 - \mathbf{x}_2) = 0 \\
\Rightarrow &\mathbf{w} \text{ is } \perp \text{ to hyper-plane } (\mathbf{x}_1 - \mathbf{x}_2) \\
\Rightarrow &\mathbf{w} \text{ is } \perp \text{ to the hyper-plane since } (\mathbf{x}_1 - \mathbf{x}_2) \text{ is a vector on the hyper-plane.}
\end{aligned}$$

Consider figure 4.1 where the hyper-plane is shown in red. The blue weight vector $\mathbf{w}$ is perpendicular to

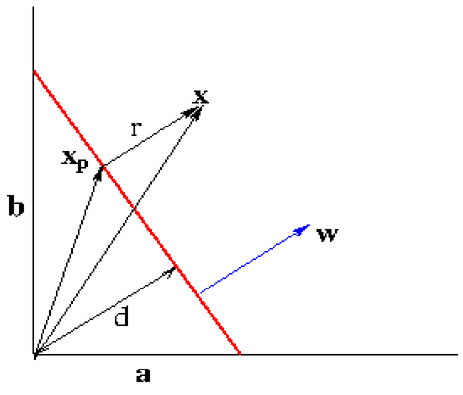

Figure 4.1: Relation between hyper-plane, $\mathbf{w}$, w_0 and an arbitrary vector $\mathbf{x}$.

the hyper-plane. Consider vector $\mathbf{x}$ where $\mathbf{x}_p$ is the point where the perpendicular from $\mathbf{x}$ intersects the hyper-plane. Vectorially, we can write $\mathbf{x} = \mathbf{x}_p + r\frac{\mathbf{w}}{\|\mathbf{w}\|}$. Note that $\frac{\mathbf{w}}{\|\mathbf{w}\|}$ is a unit vector $\perp$ to the hyper-plane

— that is in the direction of $\mathbf{w}$. Now,

$$\begin{aligned}
g(\mathbf{x}) &= g(\mathbf{x}_p + r\frac{\mathbf{w}}{\|\mathbf{w}\|}) \\
&= \mathbf{w}^T(\mathbf{x}_p + r\frac{\mathbf{w}}{\|\mathbf{w}\|}) + w_0 \\
&= \mathbf{w}^T\mathbf{x}_p + w_0 + r\frac{\mathbf{w}^T\mathbf{w}}{\|\mathbf{w}\|} \\
&= r\frac{\mathbf{w}^T\mathbf{w}}{\|\mathbf{w}\|} \text{ since } \mathbf{w}^T\mathbf{x}_p + w_0 = 0 \text{ because } \mathbf{x}_p \text{ is on the hyper-plane.} \\
r &= \frac{|g(\mathbf{x})|}{\|\mathbf{w}\|} \text{ Note that } \mathbf{w}^T\mathbf{w} = \|\mathbf{w}\|^2 \qquad (4.1)
\end{aligned}$$

The $\perp$ distance of any point $\mathbf{x}$ from the hyper-plane is given by equation 4.1. The intercepts a, b (see figure 4.1) on the X and Y axes are given by (assuming $\mathbf{w}^T = [w_1, w_2]$):

$$\begin{gathered}
\mathbf{w}^T[a, 0]^T + w_0 = 0 \\
a = -w_0/w_1 \\
\text{Similarly, } b = -w_0/w_2
\end{gathered}$$

The distance of the hyper-plane from the origin d is: $|g([0,0]^T)|/\|\mathbf{w}\|$ or $w_0/\|\mathbf{w}\|$.

Once we have the separating hyper-plane, assuming binary classification, the classification rule classifies all vectors on one side of the plane as ω_1 and the other side as ω_2. Since points on the hyper-plane satisfy $g(\mathbf{x}) = 0$ this can be written as:

$$\begin{cases} g(\mathbf{x}) > 0 \Rightarrow \omega_1 \\ g(\mathbf{x}) < 0 \Rightarrow \omega_2 \end{cases}$$

If $g(\mathbf{x}) = 0$ then either class label can be assigned.

4.1 Binary classification

We start by looking at classification tasks that have only two classes — binary classification. We start with the separable case — that is we assume that the learning set $\mathbf{L}$ can be classified without error by some hyper-plane. The classification task will be solved by converting it into an optimization problem. Different objective functions will give us different algorithms. We can simplify notation by defining extended $\mathbf{w}$ and $\mathbf{x}$ vectors as $\mathbf{w}' = [w_1, \ldots, w_d, w_0]$ and $\mathbf{x}' = [x_1, \ldots, x_d, 1]$ respectively then the hyper-plane is defined by $\mathbf{w}'^T\mathbf{x}' = 0$.

4.1.1 Perceptron

One simple cost function is the number of misclassifications. Let $\mathbf{Y}$ be the set of vectors from $\mathbf{L}$ that are misclassified by the current hyper-plane. Then we can move the hyper-plane such that such that the size of $\mathbf{Y}$ decreases. We realize this by defining the perceptron cost function:

$$J(\mathbf{w}') = \sum_{\mathbf{x}' \in \mathbf{Y}} (\delta_x \mathbf{w}'^T \mathbf{x}') \qquad (4.2)$$

where δ_x is defined by:

$$\delta_x = \begin{cases} -1 & \text{if } \mathbf{x} \in \omega_1 \\ 1 & \text{if } \mathbf{x} \in \omega_2 \end{cases}$$

Note that $J(\mathbf{x}')$ is always positive since the product of δ_x and $\mathbf{w}'^T\mathbf{x}'$ for misclassified vectors is always positive. So, for a separable learning set $\mathbf{L}$ the minimum value of $J(\mathbf{x}')$ is 0. Notice that $J(\mathbf{x}')$ is a piece wise continuous function so we can use gradient descent to find the minimum value of $J(\mathbf{x}')$. This gives the iterator:

$$\begin{aligned} \mathbf{w}'(t+1) &= \mathbf{w}'(t) - \rho(t)\frac{\partial J(\mathbf{w}')}{\partial \mathbf{w}'}\bigg|_{\mathbf{w}'=\mathbf{w}'(t)} \\ &= \mathbf{w}'(t) - \rho(t)\sum_{\mathbf{x}\in\mathbf{Y}} \delta_x \mathbf{x}' \qquad (4.3) \end{aligned}$$

where $\rho(t)$ is a constant that depends on the iteration number. We can write the following simple algorithm to find a separating hyper-plane:

Algorithm 4.1.1: PERCEPTRON(**L**)

```
$\mathbf{w}' \leftarrow$ randomly generated vector
$\mathbf{Y} \leftarrow \{\mathbf{x} \in \mathbf{L} |\ \mathbf{x}'$misclassified by the hyper-plane$\}$
$\rho \leftarrow$ initial value usually between 0 and 1
while $(\mathbf{Y} \neq \Phi)$
  do
    $\mathbf{w}' \leftarrow \mathbf{w}' - \rho \sum_{\mathbf{x}\in\mathbf{Y}} \delta_x \mathbf{x}'$   comment: Update $\mathbf{w}'$
    $\rho \leftarrow$ update $\rho$ if necessary
    $\mathbf{Y} \leftarrow \{\mathbf{x} \in \mathbf{L} |\ \mathbf{x}'$misclassified by the hyper-plane$\}$
```

The major question regarding algorithm 4.1.1 is whether or not it will converge. We argue below that choosing ρ suitably will always lead to convergence.

A small detour on convergence

Let $e(\mathbf{w})$ be the cumulative error for weight vector $\mathbf{w}$ and learning set $\mathbf{L}$. Let $e_{min}(\mathbf{w}) = \inf_{\mathbf{w}} e(\mathbf{w})$ — it is the minimum possible error on $\mathbf{L}$ (could be non-zero if $\mathbf{L}$ is not separable).

Definition 1. *an algorithm is* asymptotically optimal *if* $\lim_{t\to\infty} P(e(\mathbf{w}(t)) - e_{min} < \eta) = 1$ *for every* $\eta > 0$.

Definition 2. *an algorithm is* finitely optimal *if* $\exists \hat{t} \ni P(e(\mathbf{w}(t)) = e_{min}, \forall t > \hat{t}) = 1$ *independent of* $\mathbf{L}$.

Proof of convergence for perceptron algorithm

Theorem 1. *The perceptron algorithm is finitely optimal.*

Proof. Let $\mathbf{w}'^*$ be a solution (i.e. separating hyper-plane) and $\alpha \in \mathcal{R}\ \alpha > 0$. Subtract $\alpha\mathbf{w}'^*$ from both sides of equation 4.3

$$\mathbf{w}'(t+1) - \alpha\mathbf{w}'^* = \mathbf{w}'(t) - \alpha\mathbf{w}'^* - \rho(t)\sum_{\mathbf{x}\in\mathbf{Y}} \delta_x \mathbf{x}'$$

Take norm square on both sides:

$$\|\mathbf{w}'(t+1) - \alpha \mathbf{w}'^*\|^2 = \|\mathbf{w}'(t) - \alpha \mathbf{w}'^*\|^2 + \rho(t)^2 \|\sum_{\mathbf{x}\in\mathbf{Y}} \delta_x \mathbf{x}'\|^2 - 2\rho(t) \sum_{\mathbf{x}\in\mathbf{Y}} \delta_x (\mathbf{w}'(t) - \alpha \mathbf{w}'^*)^T \mathbf{x}'$$

Note that $\delta_x \mathbf{w}'^T \mathbf{x}' > 0$ for all $\mathbf{x} \in \mathbf{Y}$ since $\mathbf{x}$ is misclassified so $-2\rho(t)\sum_{\mathbf{x}\in\mathbf{Y}} \delta_x \mathbf{w}'^T \mathbf{x}' > 0$ and we can write the above equation as the following inequality after dropping the always negative term:

$$\|\mathbf{w}'(t+1) - \alpha \mathbf{w}'^*\|^2 \leq \|\mathbf{w}'(t) - \alpha \mathbf{w}'^*\|^2 + \rho(t)^2 \|\sum_{\mathbf{x}\in\mathbf{Y}} \delta_x \mathbf{x}'\|^2 + 2\rho(t)\alpha \sum_{\mathbf{x}\in\mathbf{Y}} \delta_x \mathbf{w}'^{*T} \mathbf{x}' \tag{4.4}$$

Let

$$\beta = \max_{\mathbf{Y}\in\mathcal{P}(\mathbf{L}), \mathbf{Y}\neq\Phi} \|\sum_{\mathbf{x}\in\mathbf{Y}} \delta_x \mathbf{x}'\|^2$$

and

$$\gamma = \max_{\mathbf{Y}\in\mathcal{P}(\mathbf{L}), \mathbf{Y}\neq\Phi} \sum_{\mathbf{x}\in\mathbf{Y}} \delta_x \mathbf{w}'^{*T} \mathbf{x}'$$

where $\mathcal{P}(\mathbf{L})$ is the power set of $\mathbf{L}$.

Since $\mathbf{w}'^*$ is the separating hyper-plane every $\mathbf{x}$ is classified correctly and $\delta_x \mathbf{w}'^{*T} \mathbf{x}$ is always negative, consequently γ is always negative. So, equation 4.4 can be written as:

$$\|\mathbf{w}'(t+1) - \alpha \mathbf{w}'^*\|^2 \leq \|\mathbf{w}'(t) - \alpha \mathbf{w}'^*\|^2 + \rho(t)^2\beta^2 - 2\rho(t)\alpha|\gamma| \tag{4.5}$$

Choose $\alpha = \frac{\beta^2}{2|\gamma|}$, substiute for α on the right hand side of equation 4.5 (to get equation 4.6) and unfold from t to 0.

$$\begin{aligned}
\|\mathbf{w}'(t+1) - \alpha \mathbf{w}'^*\|^2 &\leq \|\mathbf{w}'(t) - \alpha \mathbf{w}'^*\|^2 + \beta^2[\rho(t)^2 - \rho(t)] && (4.6)\\
&\leq \|\mathbf{w}'(t-1) - \alpha \mathbf{w}'^*\|^2 + \beta^2[\rho(t)^2 + \rho(t-1)^2 - (\rho(t) + \rho(t-1))]\\
&\leq \dots\\
&\leq \dots\\
&\leq \|\mathbf{w}'(0) - \alpha \mathbf{w}'^*\|^2 + \beta^2[\sum_{k=0}^{t} \rho(k)^2 - \sum_{k=0}^{t} \rho(k)] && (4.7)
\end{aligned}$$

Now choose $\rho(t)$ such that $\lim_{t\to\infty} \sum_{k=0}^{t} \rho(k) = \infty$ (that is diverges) and $\lim_{t\to\infty} \sum_{k=0}^{t} \rho(k)^2 < \infty$ (that is converges). Then $\exists \hat{t}$ such that the rhs in equation 4.7 becomes ≤ 0 at $\hat{t}$. That is: $0 \leq \|\mathbf{w}'(\hat{t}+1) - \alpha \mathbf{w}'^*\|^2 \leq 0$ (note that a norm by definition must be ≥ 0). This implies that $\mathbf{w}'(\hat{t}) = \alpha \mathbf{w}'^*$ that means the algorithm converges. A possible choice for $\rho(t)$ is $\rho(t) = c/(t+\epsilon)$ where c, ϵ are constants. □

Choosing a proper $\rho(t)$ is important for fast convergence. One can show that even if $\rho(t) = \rho$, ρ a suitably bounded constant the algorithm converges.

The algorithm above is called the batch perceptron algorithm. One can instead make an update to $\mathbf{w}'$ after every $\mathbf{x} \in \mathbf{L}$ is classified using the following update rule:

$$\mathbf{w}'(t+1) = \begin{cases} \mathbf{w}'(t) + \rho(t)\mathbf{x}'(t) & \mathbf{x}'(t) \in \omega_1 \wedge \mathbf{w}'(t)^T \mathbf{x}'(t) \leq 0 \\ \mathbf{w}'(t) - \rho(t)\mathbf{x}'(t) & \mathbf{x}'(t) \in \omega_2 \wedge \mathbf{w}'(t)^T \mathbf{x}'(t) > 0 \\ \text{otherwise} \end{cases}$$

The perceptron algorithm with the above update rule also converges.

Perceptron algorithm for non-separable L

A variant of the perceptron algorithm (pocket algorithm) converges, but only asymptotically, to an optimal solution (that is minimum error) even when **L** is not separable.

Algorithm 4.1.2: POCKET(**L**)

```
$\mathbf{w}'(0) \leftarrow$ random initialization
$\mathbf{w}'_s \leftarrow \mathbf{w}'(0)$   comment: stored or pocket weight vector
$h_s \leftarrow 0$   comment: history counter
$t \leftarrow 0$
while (not converged)
    do
        $\mathbf{w}'(t+1) \leftarrow$ update with perceptron rule
        $h \leftarrow$ number of vectors correctly classified by $\mathbf{w}'$(t+1)
        if ($h > h_s*$)
            then
                $h_s \leftarrow h$
                $\mathbf{w}'_s \leftarrow \mathbf{w}'(t+1)$
        $t \leftarrow t+1$
return ($\mathbf{w}'_s$)
```

The convergence condition has to be based on the number of iterations for which $\mathbf{w}'_s$ has not changed. One can prove[1,2] that the pocket algorithm is asymptotically optimal — so it is not possible to put an apriori upper bound on the number of iterations of the while loop.

4.1.2 Least mean square algorithm

Here, we try to minimize the expected square of the error. That is:

$$J(\mathbf{w}') = E[|y - \mathbf{w}'^T\mathbf{x}'|^2]$$
$$\hat{\mathbf{w}'} = \underset{\mathbf{x}'}{\operatorname{argmin}}\ J(\mathbf{w}') \tag{4.8}$$

Minimizing $J(\mathbf{w}')$ gives

$$\frac{\partial J(\mathbf{w}')}{\partial \mathbf{w}'} = 2E[\mathbf{x}'(y - \mathbf{w}'^T\mathbf{x})] = 0 \tag{4.9}$$

This immediately gives $\hat{\mathbf{w}'} = R_{\mathbf{x}'}^{-1}E[\mathbf{x}'y]$, where $R_{\mathbf{x}'} \equiv E[\mathbf{x}'\mathbf{x}'^T]$. R_x is called the correlation or auto-correlation matrix:

$$\begin{bmatrix} E[\mathbf{x}'_1\mathbf{x}'_1] & \dots & E[\mathbf{x}'_1\mathbf{x}'_{d+1}] \\ \dots & \dots & \dots \\ \dots & \dots & \dots \\ E[\mathbf{x}'_{d+1}\mathbf{x}'_1] & \dots & E[\mathbf{x}'d+1\mathbf{x}'d+1] \end{bmatrix}$$

[1]SI Gallant, Perceptron based learning algorithm, IEEE Tran. Neural networks, 1(2), 179-191 (1990)

[2]M Muselli, On convergence properties of pocket algorithm, IEEE Tran. Neural networks, 8(3), 623-629, (1997)

$E[\mathbf{x}'y]$ is the cross-correlation matrix:

$$E[\mathbf{x}'y] = E\left[\begin{bmatrix} \mathbf{x}'_1 y \\ \cdot \\ \cdot \\ \mathbf{x}'_{d+1} y \end{bmatrix}\right]$$

$\hat{\mathbf{w}'}$ can be obtained by solving the above set of linear equations provided $R_{\mathbf{x}'}^{-1}$ exists. The real problem with the above method is that it requires the calculation of the expectation which requires knowledge of the class conditional pdfs and these are not usually known.

Instead this can be calculated by a stochastic approximation scheme.

Stochastic approximation

Given $E[f(\mathbf{x}_k, \mathbf{w})] = 0,\ k = 1, 2, \ldots$ where the $\mathbf{x}_k$s are a sequence of random vectors drawn iid from the distribution then one can approximate the unknown $\hat{\mathbf{w}}$ using the following iterative scheme[3]:

$$\hat{\mathbf{w}}(k+1) = \hat{\mathbf{w}}(k) + \rho(k) f(\mathbf{x}_k, \hat{\mathbf{w}}(k)) \tag{4.10}$$

One can prove that the iteration converges in probability ($\lim\limits_{k\to\infty} P[\hat{\mathbf{w}}(k) - \mathbf{w}] = 1$) to the original equation provided: $\sum_{k=1}^{\infty} \rho(k) \to \infty$ and $\sum_{k=1}^{\infty} \rho(k)^2 < \infty$. This implies $\rho(k) \to 0$. It is also true that: $\lim\limits_{k\to\infty} E[\|\hat{\mathbf{w}}(k) - \mathbf{w}\|^2] = 0$. We can use equation 4.10 to solve: $\frac{\partial J(\mathbf{w}')}{\partial \mathbf{w}'} = 2E[\mathbf{x}'(y - \mathbf{w}'^T \mathbf{x})] = 0$ giving $\hat{\mathbf{w}}(k+1) = \hat{\mathbf{w}}(k) + \rho(k)\mathbf{x}(y_k - \mathbf{x}^T \hat{\mathbf{w}}(k))$ where $(\mathbf{x}_k, y_k)$ are successive elements of $\mathbf{L}$. This if often called the Widrow-Hoff algorithm[4]

There are variants of LMS like using a fixed ρ instead of $\rho(k)$. However, this does not converge to the MSE solution. If $0 < \rho < 2/trace(R_{\mathbf{x}})$ then $E[\hat{\mathbf{w}}(k)] \to \mathbf{w}_{MSE}$ and $E[\|\hat{\mathbf{w}}(k) - \mathbf{w}_{MSE}\|^2] \to$ const, $\mathbf{w}_{MSE}$ is the MSE optimal estimate. The smaller ρ the smaller the variation about $\mathbf{w}_{MSE}$ but also slower the convergence.

4.1.3 Sum of squares algorithm

A closely related algorithm is the sum of error squared on $\mathbf{L}$.

$$J(\mathbf{w}') = \sum_{i=1}^{n} (y_i - \mathbf{w}'^T \mathbf{x}')^2 = \sum_{i=1}^{n} e_i^2 \tag{4.11}$$

In this case we do not need any knowledge of the pdfs. Minimizing with respect to $\mathbf{w}'$ gives:

$$\sum_{i=1}^{n} \mathbf{x}'_i (y_i - \mathbf{x}'^T \hat{\mathbf{w}'}) = 0 \text{ (Note: } \mathbf{w}'^T \mathbf{x}' = \mathbf{x}'^T \mathbf{w}') \tag{4.12}$$

$$(\sum_{i=1}^{n} \mathbf{x}'_i \mathbf{x}'^T_i) \hat{\mathbf{w}'} = \sum_{i=1}^{n} (\mathbf{x}'_i y_i)$$

[3]H Robbins, S Monroe, A stochastic approximation method, Ann. of Math. Stat. 22, 400-407, 1951.

[4]See - B Widrow, MA Lehr, 30 years of adaptive neural networks: Perceptron, madaline, and backpropagation, Proc. of the IEEE,78(9), 1415-1442, 1990.

Let $X = [\mathbf{x'}_1, \ldots, \mathbf{x'}_n]$ and $\mathbf{y} = [y_1, \ldots, y_n]^T$ where X is a $n \times (d+1)$ matrix. Then we have $\sum_{i=1}^{n} \mathbf{x'}_i \mathbf{x'}_i^T = X^T X$, $\sum_{i=1}^{n} \mathbf{x'}_i y_i = X^T \mathbf{y}$. So, equation 4.12 can be written as:

$$(X^T X)\hat{\mathbf{w}'} = X^T \mathbf{y}$$
$$\hat{\mathbf{w}'} = (X^T X)^{-1} X^T \mathbf{y}$$

where $\tilde{X} = (X^T X)^{-1} X^T$ is called the *pseudo inverse*. If $n = d+1$ then the pseudo inverse is a square matrix and $\hat{\mathbf{w}'} = \tilde{X}\mathbf{y}$ is a set of linear equations that can be solved to obtain $\hat{\mathbf{w}'}$. More generally, $\tilde{X}$ is not square as $n > d+1$ and there is no single solution. Then one calculates the limit: $\tilde{X} = \lim_{\epsilon \to 0}(X^T X + \epsilon I)^{-1} X^T$. One can prove that the limit always exists and $\hat{\mathbf{w}'}^T = \tilde{X}\mathbf{y}$ is an MSE solution to $\mathbf{y} = X\hat{\mathbf{w}'}$.

4.2 Multi-class classification - to be completed

4.3 Support vector machines - to be completed

Again we assume we have a binary classification problem. Consider the separating hyper-planes in figure 4.2 below. In figure 4.2 two sets of hyper-planes — $A1$ to $A3$ and $B1$ to $B3$ separate the two classes (shown

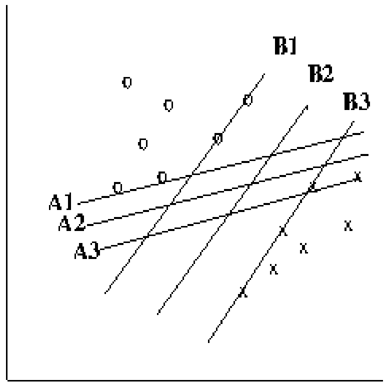

Figure 4.2: Two sets of separating hyper-planes—A1 to A3, B1 to B3 - fig needs to be improved

as o and x). Our intuition says that $A2$ is better than $A1$ or $A3$ and similarly $B2$ is better than $B1$ or $B3$ since $A2$ and $B2$ are at a greater distance from the boundary points and provide more leeway to points in their vicinity to be correctly classified. But which of $A2$ or $B2$ is better? This is harder to say but using the same intuition we may conclude that $B2$ is even further than $A2$ from the boundary points of the two classes and therefore is likely to be better. This basic observation is at the heart of what are called *large margin classifiers* which give rise to a set of classifiers called support vector machines (SVMs). For the purpose of SVMs the *margin* can be defined as the sum of the nearest distances of the convex hull of both classes from the separating hyper-plane, where the hyper-plane is chosen so that it bisects the margin distance. So, the nearest point of the convex hull of each class is equi-distant from the hyper-plane.

Given a separating hyper-plane the distance of the boundary point of class ω_1 say $\mathbf{x}_1$, from the plane is given by $d_1 = |g(\mathbf{x}_1)|/\|\mathbf{w}\|$. Similarly, the distance of a boundary point, $\mathbf{x}_2$ of class ω_2 is given by $d_2 = |g(\mathbf{x}_2)|/\|\mathbf{w}\|$. Since for the desired hyper-plane $d_1 = d_2$ we can write the margin as $2|g(\mathbf{x}_1)|/\|\mathbf{w}\|^2$. Further, if we scale $g(\mathbf{x})$

such that the distance $g(\mathbf{x}_1) = g(\mathbf{x}_2) = 1$ then our margin becomes $2/\|\mathbf{w}\|^2$. Our optimization problem now is to maximize this margin. Unlike earlier objective functions which directly tried to reduce the error and thereby improve classification the margin does not directly say anything about the error. It does not even make explcit that the plane is a separating hyper-plane. So we have to impose constraints to ensure that the hyper-plane is indeed a separating one. We do this as follows: observe that after scaling our classification rule becomes:

$$\begin{cases} g(\mathbf{x}) \geq 1 \Rightarrow \mathbf{x} \in \omega_1 \\ g(\mathbf{x}) \leq 1 \Rightarrow \mathbf{x} \in \omega_2 \end{cases}$$

Since our desired output is $y = \pm 1$ for ω_1 and ω_2 respectively the separation constraint becomes $yg(\mathbf{x}) > 1,\ \forall (x, y) \in \mathbf{L}$ (note we assume that all ω_1 have been replaced by 1 and ω_2 by -1 in $\mathbf{L}$). We, therefore, get the constrained optimization problem:

$$\max_{\mathbf{w}, w_0} \frac{2}{\|\mathbf{w}\|^2}$$

subject to:

$$yg(\mathbf{x}) \geq 1\ \forall (\mathbf{x}, y) \in \mathbf{L}$$

Instead of trying to solve the above problem we convert it to the following equivalent problem that is more tractable:

$$\min_{\mathbf{w}, w_0} \frac{\|\mathbf{w}\|^2}{2}$$

subject to:

$$yg(\mathbf{x}) - 1 \geq 0\ \forall (\mathbf{x}, y) \in \mathbf{L} \tag{4.13}$$

(4.13) is now a convex optimization problem (actually quadratic optimization) with linear inequality constraints. There are well known methods to solve such problems [5]. The KKT (Karush, Kuhn, Tucker) conditions are necessary conditions for optimality. In some cases like the present one they are also sufficient. In particular we use the KKT (some call it Lagrange) multipliers method to solve it. A Lagrangian embeds the objective function and the constraints in the original problem and converts it into an unconstrained problem. The Lagrangian is:

$$\begin{aligned} L(\mathbf{w}, w_0, \alpha) &= \frac{\|\mathbf{w}\|^2}{2} - \sum_{i=1}^{n} \alpha_i y_i g(\mathbf{x}_i) - 1) \\ &= \frac{\mathbf{w}^T\mathbf{w}}{2} - \sum_{i=1}^{n} \alpha_i [y_i(\mathbf{w}^T\mathbf{x}_i + w_0) - 1] \end{aligned} \tag{4.14}$$

The KKT conditions are:

Stationarity:

$$\frac{\partial L}{\partial \mathbf{w}} = 0 \tag{4.15}$$

$$\frac{\partial L}{\partial w_0} = 0 \tag{4.16}$$

[5]S Boyd, L Vandenberghe, Convex Optimization, Cambridge Univ. Press, 2004.

Dual feasibility:

$$\alpha_1 \geq 0 \tag{4.17}$$

Complementary slackness:

$$\alpha_i g_i(\mathbf{w}) = 0 \tag{4.18}$$

The solution to the optimization problem is obtained by minimizing L with respect to the *primal* variables $\mathbf{w}$, w_0 and maximizing with respect to the *dual* variables - the multipliers α_i. Equation 4.16 yields

$$\sum_{i=1}^{n} \alpha_i y_i = 0 \tag{4.19}$$

and 4.15 gives

$$\mathbf{w} - \sum_{i=1}^{n} \alpha_i y - i\mathbf{x}_i = 0$$

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \tag{4.20}$$

Equation (4.20) gives $\mathbf{w}$ as a linear combination of vectors from the learning set $\mathbf{L}$ — those for which $\alpha_i \neq 0$ are called *support vectors*. Equation (4.14) can be written as:

$$L(\mathbf{w}, w_0, \alpha_i) = \frac{1}{2}\langle \mathbf{w}, \mathbf{w} \rangle - [\sum_{i=1}^{n} \alpha_i y_i \langle \mathbf{w}, \mathbf{x}_i \rangle + w_0 \sum_{i=1}^{n} \alpha_i y_i - \sum_{i=1}^{n} \alpha_i] \tag{4.21}$$

Note that $\langle ., . \rangle$ stands for the scalar/dot/inner product.

Using (4.20) we can substitute for $\mathbf{w}$ in (4.21) to get:

$$L(\mathbf{w}, w_0, \overline{\alpha} = \frac{1}{2}\langle \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i, \sum_{j=1}^{n} \alpha_j y_j \mathbf{x}_j \rangle - \sum_{i=1}^{n} \alpha_i y_i \langle \sum_{j=1}^{n} \alpha_j y_j \mathbf{x}_j, \mathbf{x}_i \rangle + \sum_{i=1}^{n} \alpha_i \tag{4.22}$$

$$W(\overline{\alpha}) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \tag{4.23}$$

Note that the first and second terms in equation (4.22) are the same (by properties of inner product) except for the $\frac{1}{2}$.

Equation (4.23) is called the Wolf dual. The dual problem becomes:

$$\max_{\overline{\alpha}} W(\overline{\alpha})$$

Subject to

$$\alpha_i \geq 0, \ \sum_{i=1}^{n} \alpha_i y_i = 0 \tag{4.24}$$

The optimization problem (4.24) is solved to get the optimal α_i^*s (note that it is a quadratic programming proglem). For $\alpha_i^* \neq 0$ we can use the KKT complementary slackness condition $\alpha_i^*[y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + w_0) - 1] = 0$ to get w_0 since all the other values are known. This means we have w_0 and $\mathbf{w}$ from (4.20) giving us the complete hyper-plane.

The decision function for an unknown vector $\mathbf{x}$ is: $\text{sign}(\sum_{i=1}^{n} \alpha_i y_i \langle \mathbf{x}, \mathbf{x}_i \rangle + w_0)$. If the sign is positive it the predicted label is ω_1 otherwise ω_2.

A fairly complete discussion on SVMs can be found in Cristianini and Shaw-Taylor's book.[6]

[6]N Cristianini, J Shawe-Taylor, An introduction to support vector machines and other kernel based learning methods, Cambridge Univ. Press, 2000.