COMP 652: Machine Learning

Lecture 12

Today

- Perceptrons
  - Definition
  - Perceptron learning rule
  - Convergence
- (Linear) support vector machines
  - Margin & max margin classifier
  - Formulation as optimization problem
  - Generalized Lagrangian and dual

Perceptrons

- Consider a binary classification problem with data $\{\mathbf{x_i}, y_i\}_{i=1}^m$
- Let the two classes be $y = -1$ and $y = +1$
- A *perceptron* is a classifier of the form:

$$h_{\mathbf{w},w_0}(\mathbf{x}) = \text{sgn}(\mathbf{w} \cdot \mathbf{x} + w_0) = \begin{cases} +1 & \text{if } \mathbf{w} \cdot \mathbf{x} + w_0 \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

  Here, $\mathbf{w}$ is a vector of weights, "$\cdot$" denotes the dot product, and w_0 is a constant offset.
- The decision boundary is $\mathbf{w} \cdot \mathbf{x} + w_0 = 0$.

- How do we measure goodness of fit w.r.t. a data set?

The 0-1 loss function

- Perceptrons do not output class probabilities, just a guess at the class.
- An example $\langle \mathbf{x}, y \rangle$ is classified correctly if (and pretty much only if):

$$y(\mathbf{w} \cdot \mathbf{x} + w_0) > 0$$

- For a data set $\{\mathbf{x_i}, y_i\}_{i=1}^{m}$, the *0-1 loss* for a particular perceptron given by parameters $\mathbf{w}, w_0$ is simply the number of examples classified incorrectly.
- The $0 - 1$ loss function:
  - Takes values in the set $\{0, 1, \ldots, m\}$.
  - Is piecewise constant.
  - The derivatives with respect to $\mathbf{w}$, w_0 are zero everywhere that they are defined.

  $\Rightarrow$ Not good for analytical solution or for gradient descent.

A gradient descent-like learning rule

- Condisder the following procedure:
  1. Initialize $\mathbf{w}$ and w_0 randomly
  2. While any training examples remain incorrecty classified
     (a) Loop through all misclassified examples
     (b) For misclassified example i, perform the updates:

$$\mathbf{w} \leftarrow \mathbf{w} + \gamma y_i \mathbf{x}_i, \qquad w_0 \leftarrow w_0 + \gamma y_i$$

where γ is a step-size parameter.

- The update equation, or sometimes the whole procedure, is called the *perceptron learning rule*.
- Does this rule make sense?

A gradient descent-like learning rule

- Condisder the following procedure:
  1. Initialize $\mathbf{w}$ and w_0 randomly
  2. While any training examples remain incorrecty classified
     (a) Loop through all misclassified examples
     (b) For misclassified example i, perform the updates:

$$\mathbf{w} \leftarrow \mathbf{w} + \gamma y_i \mathbf{x}_i, \qquad w_0 \leftarrow w_0 + \gamma y_i$$

     where γ is a step-size parameter.

- The update equation, or sometimes the whole procedure, is called the *perceptron learning rule*.
- Does this rule make sense? Yes, because for examples misclassified as negative, it increases $\mathbf{w} \cdot \mathbf{x_i} + w_0$, and for examples misclassified as positive, it decreases $\mathbf{w} \cdot \mathbf{x_i} + w_0$.

Gradient descent interpretation

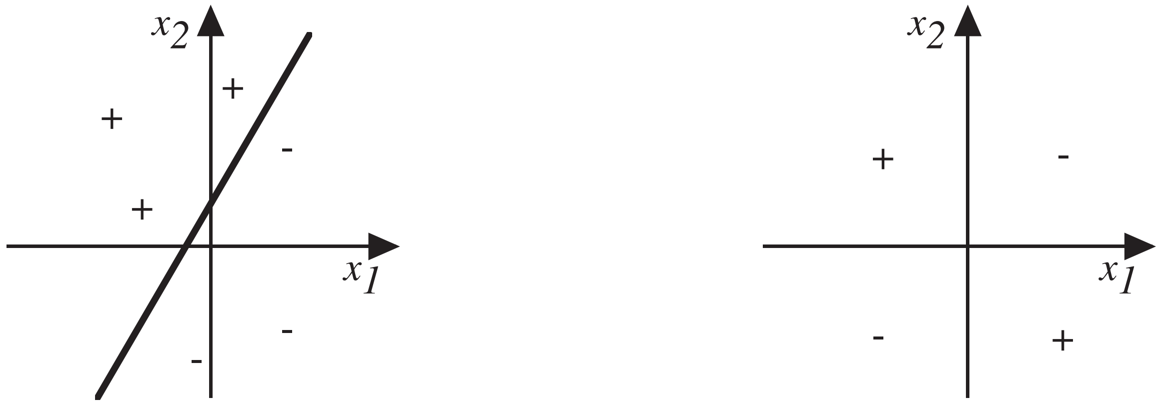
- The perceptron learning rule can be interpreted as a gradient descent procedure, but with a different error function than the 0-1 loss.
- Consider the *perceptron criterion function*

$$J(\mathbf{w}, w_0) = \sum_{i=1}^{m} \begin{cases} 0 & \text{if } y_i(\mathbf{w} \cdot \mathbf{x_i} + w_0) \geq 0 \\ -y_i(\mathbf{w} \cdot \mathbf{x_i} + w_0) & \text{if } y_i(\mathbf{w} \cdot \mathbf{x_i} + w_0) < 0 \end{cases}$$

- For correctly classified examples, the error is zero.
- For incorrectly classified examples, the error is by how much $\mathbf{w} \cdot \mathbf{x_i} + w_0$ is on the wrong side of the decision boundary.
- J is piecewise linear, and its gradient gives the perceptron learning rule.
- J is zero if (and pretty much only iff) all examples are classified correctly – just like the 0-1 loss function.
- Do we get convergence?

Linear separability

- Consider a (binary classifiacdtion) data set $\{\mathbf{x_i}, y_i\}_{i=1}^m$.
- The data set is <u>*linearly separable*</u> if and only if there exists $\mathbf{w}$, w_0 such that:
  - For all i, $y_i(\mathbf{w} \cdot \mathbf{x_i} + w_0) > 0$.
  - Or equivalently, the 0-1 loss is zero.

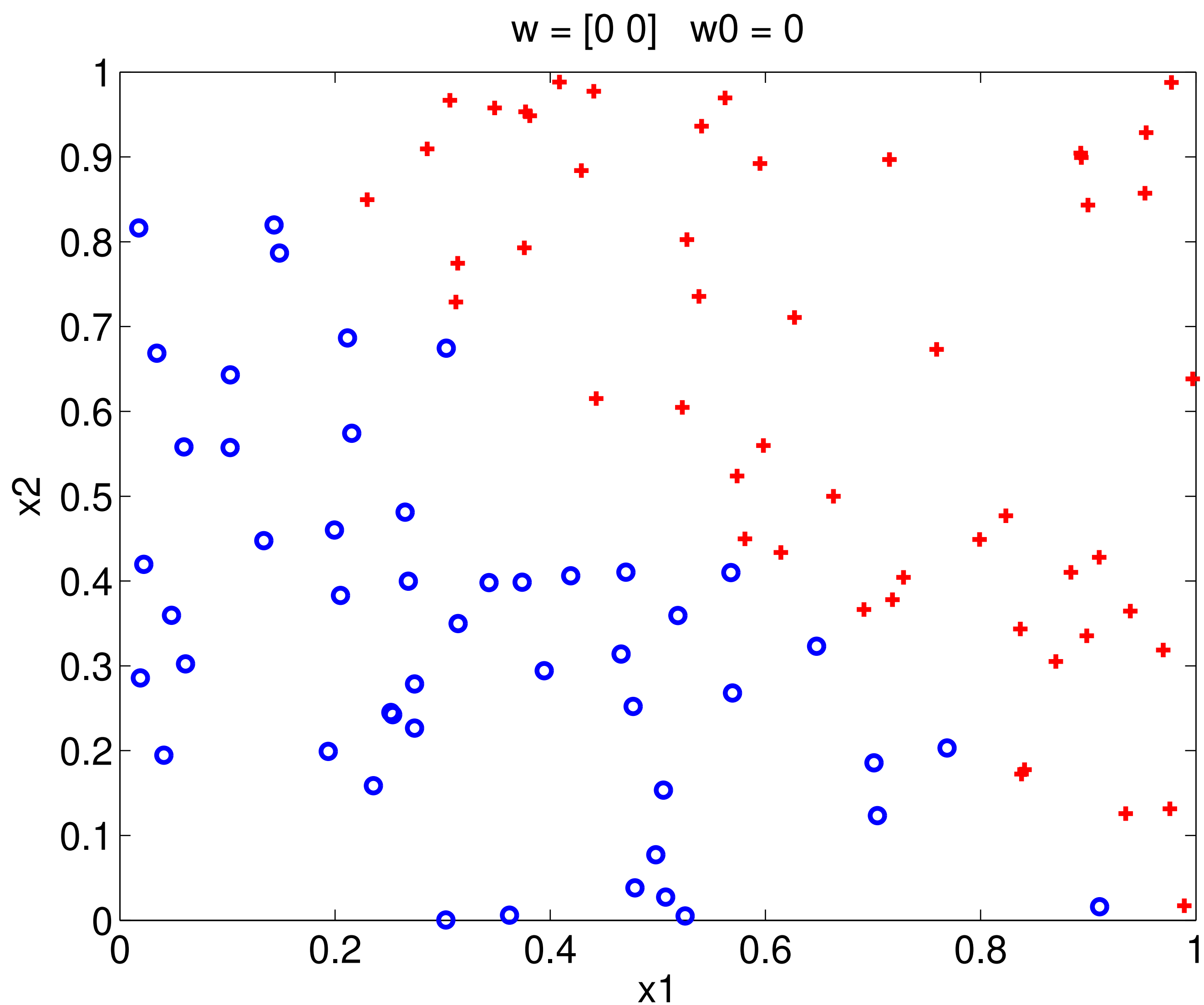


(a) (b)

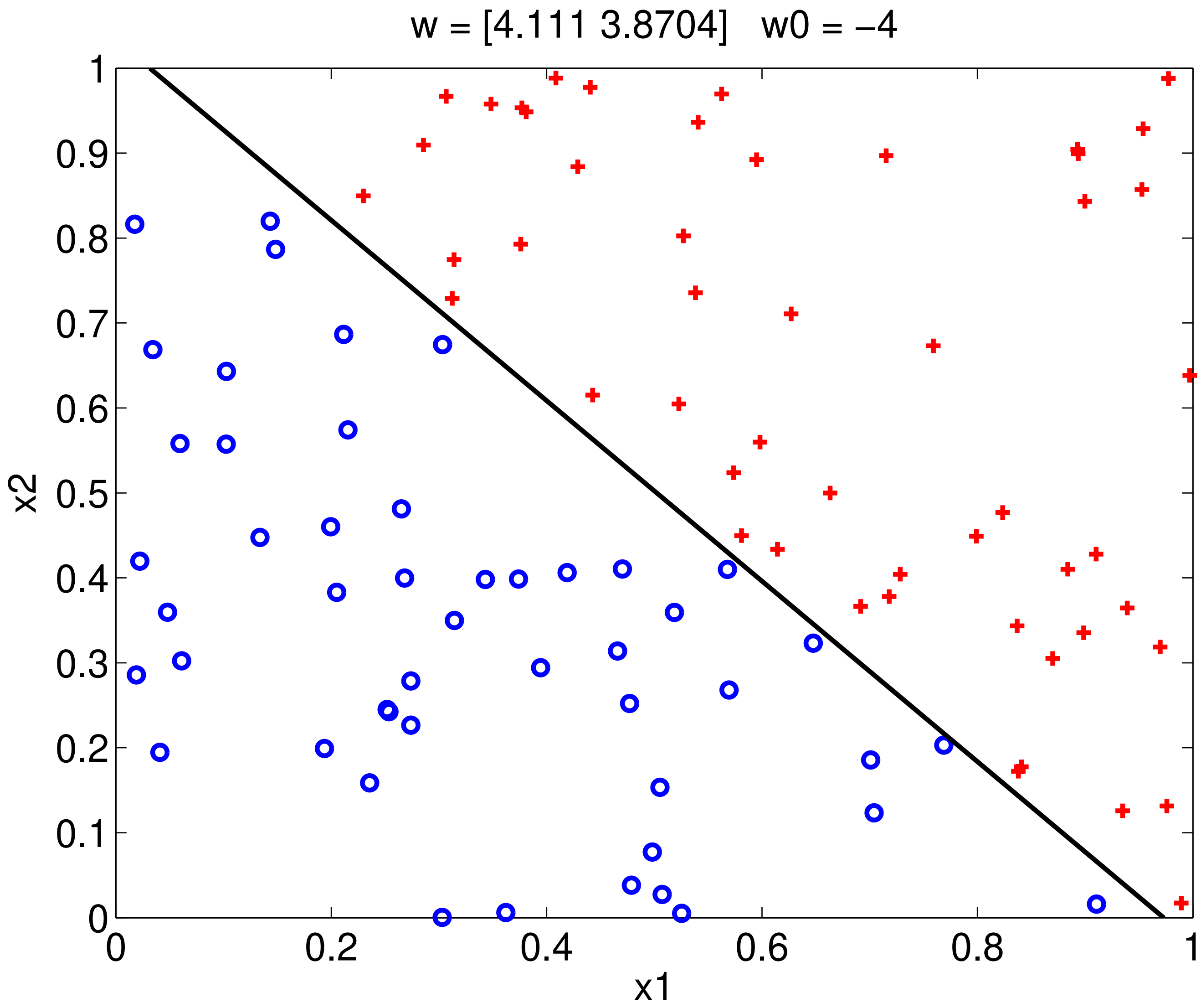
Perceptron convergence theorem

- The *perceptron convergence theorem* states that if the perceptron learning rule is applied to a linearly separable data set, a solution will be found after some finite number of updates.
- The number of updates depends on the data set, and also on the step size parameter. Step size $= 1$ can be used.
- See Duda, Hart & Stork, Chapter 5, for proof.

Perceptron learning example–separable data



Perceptron learning example–separable data



Weight as a combination of input vectors

- Recall percepton learning rule:

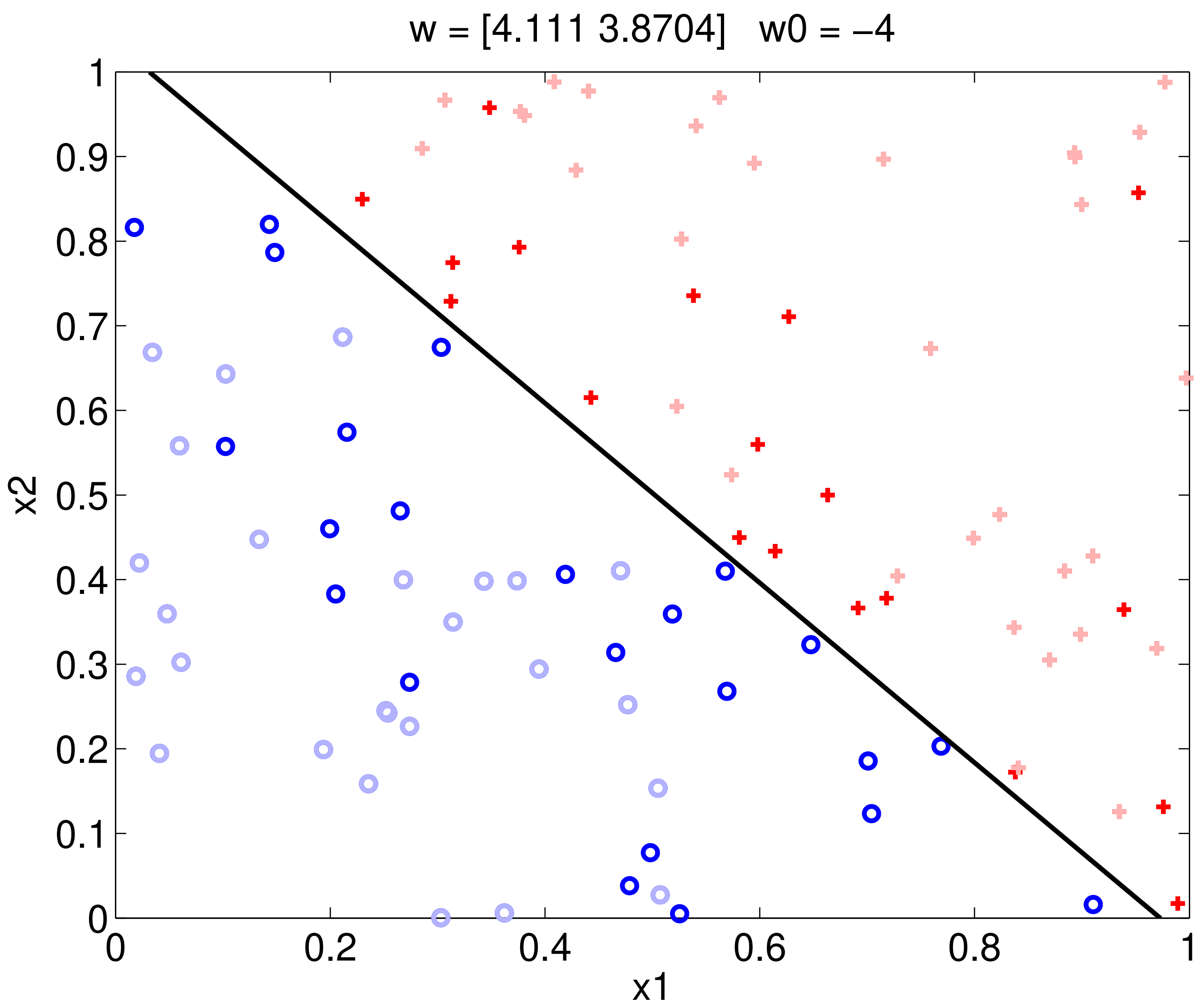
$$\mathbf{w} \leftarrow \mathbf{w} + \gamma y_i \mathbf{x}_i, \qquad w_0 \leftarrow w_0 + \gamma y_i$$

- If initial weights are zero, then at any step, the weights are a linear combination of feature vectors.
  That is, at any time we can write

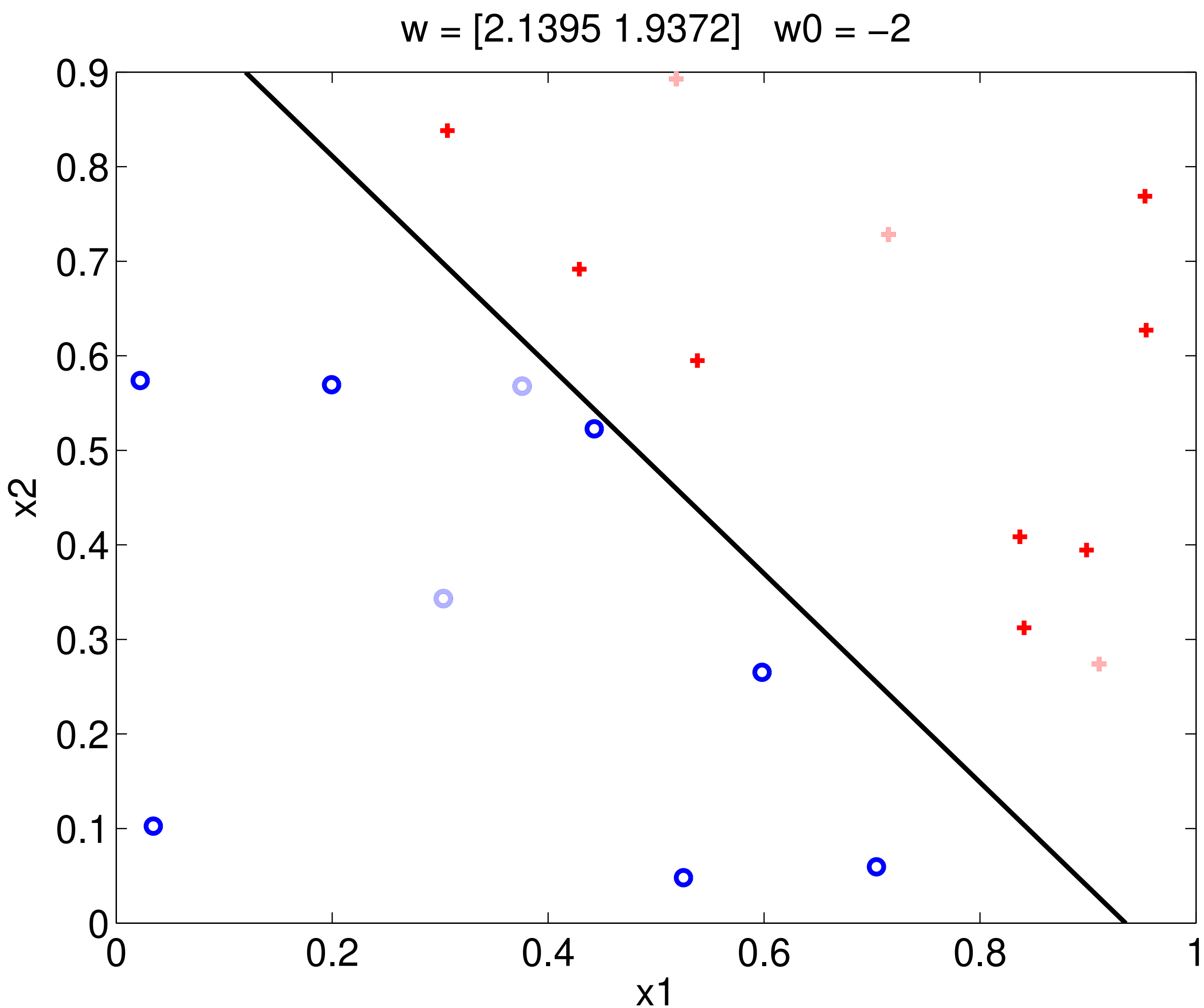
$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i \mathbf{x_i}, \qquad w_0 = \sum_{i=1}^{m} \alpha_i y_i$$

  where α_i is the sum of step sizes used for all updates based on example i.
- This is called the *dual representation* of the classifier.
- Even by the end of training, some example may have never participated in an update.

Example used (bold) and not used (faint) in updates



Comment: Solutions are nonunique



Perceptron comments

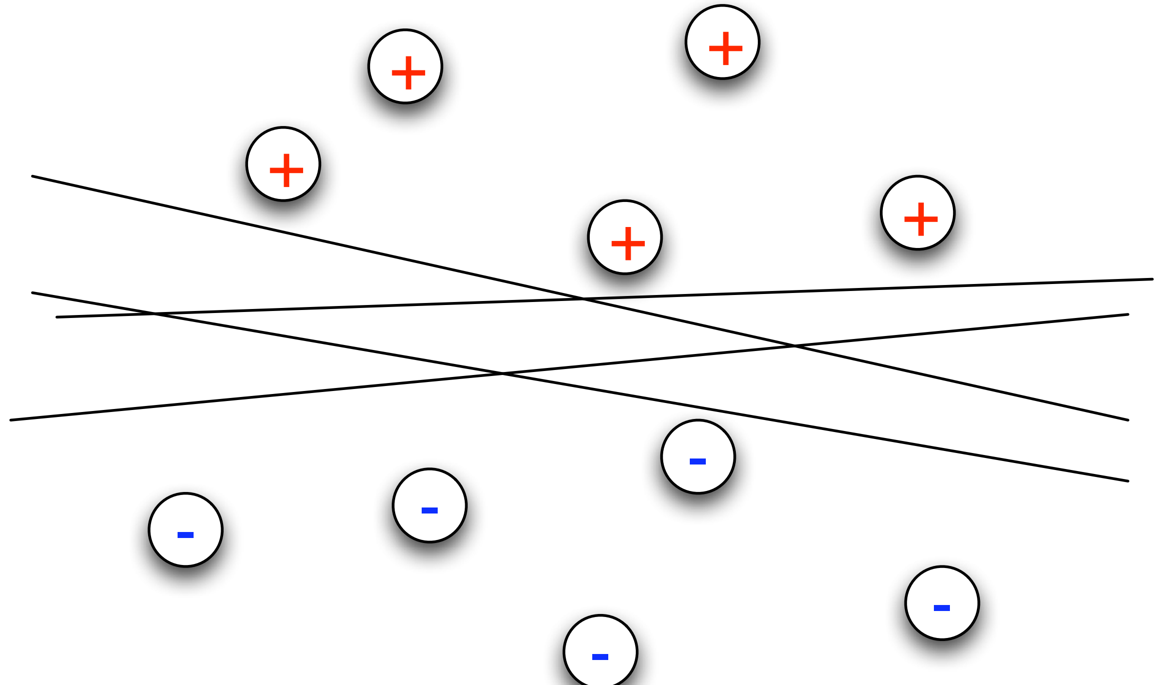
- Perceptrons can be learned to fit linearly separable data, using a gradient descent rule.
- There are other fitting approaches – e.g., formulation as a linear constraint satisfaction problem / linear program.
- Solutions are non-unique.
- What about non-linearly seperable data?
  - Training by perceptron learning rule doesn't terminate / need not converge.
  - Perhaps data can be linearly separated in a different feature space?
  - Perhaps we can relax the criterion of separating all the data?

Support Vector Machines

- Support vector machines (SVMs) for binary classification are a way of training perceptrons (pretty much)
- There are three main ideas they add:
  - An alternative optimization criterion (the "margin"), which eliminates the non-uniqueness of solutions and has certain theoretical advantages
  - An efficient way of operating in expanded feature spaces – the "kernel trick"
  - A way of handling nonseparable data
- SVMs can also be used for multiclass classification and regression.

Returning to the non-uniqueness issue

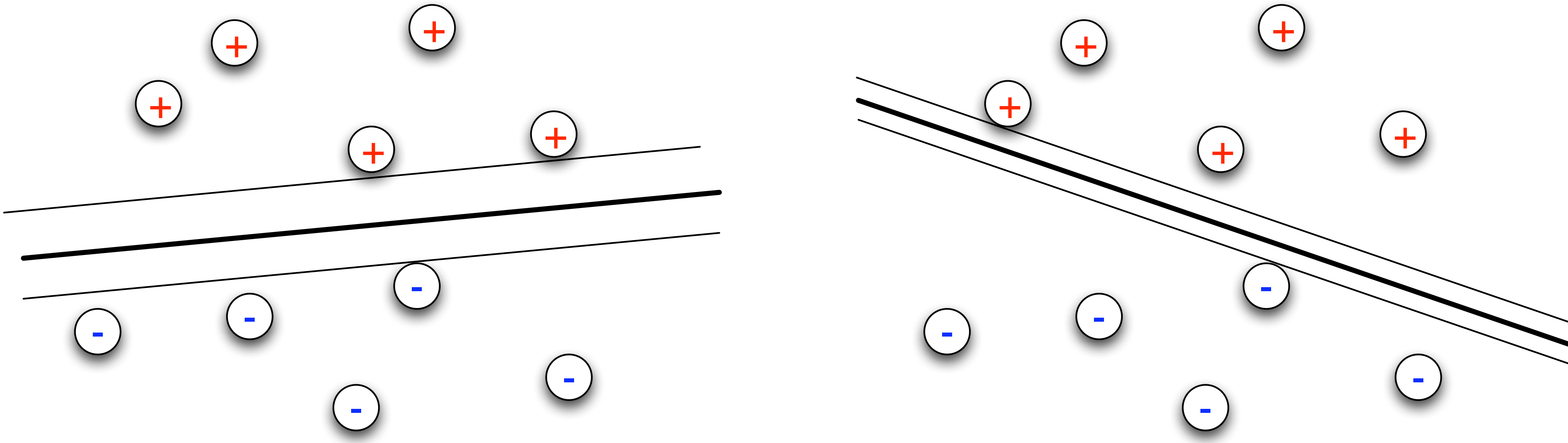
- Consider a linearly separable binary classification data set $\{\mathbf{x_i}, y_i\}_{i=1}^{m}$.
- There is an infininte number of hyperplanes that separate the two classes:



- Which plane is best?
- Relatedly, for a given plane, for which points should we be most confident in the classification?

The margin, and linear SVMs

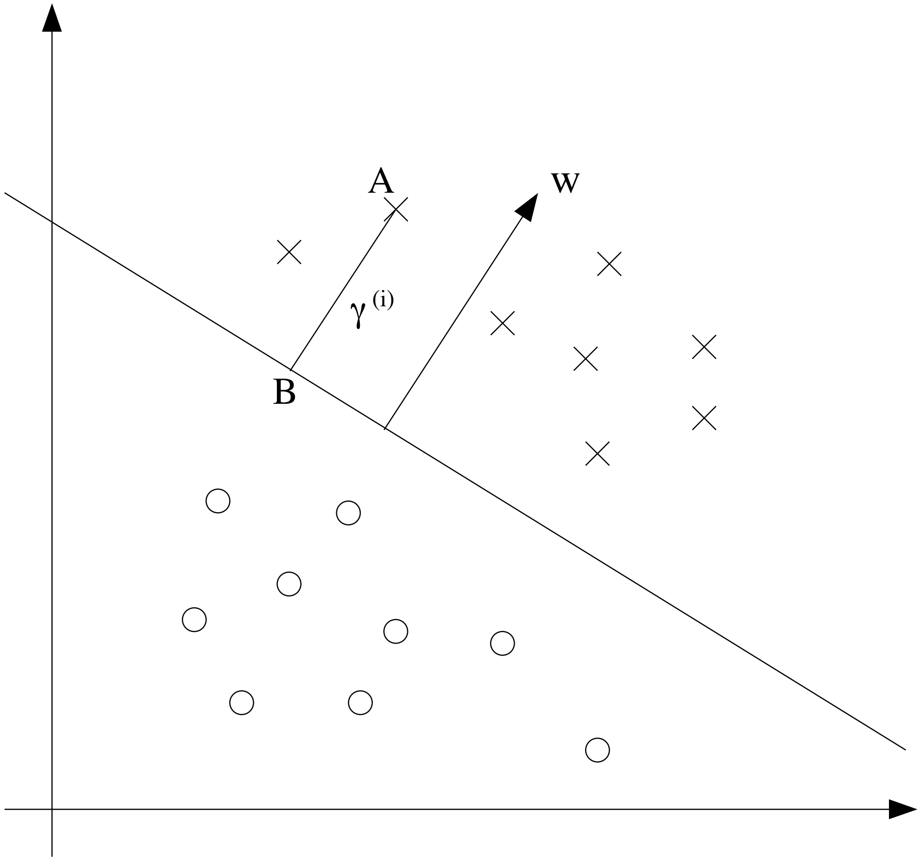
- For a given separating hyperplane, the *margin* is two times the (Euclidean) distance from the hyperplane to the nearest training example.



- It is the width of the "strip" around the decision boundary containing no training examples.
- A linear SVM is a perceptron for which we choose $\mathbf{w}, w_0$ so that margin is maximized

Distance to the decision boundary

- Suppose we have a decision boundary that separates the data.



- Let γ_i be the distance from instance $\mathbf{x_i}$ to the decision boundary.
- How can we write γ_i in term of $\mathbf{x_i}, y_i, \mathbf{w}, w_0$?

Distance to the decision boundary (II)

- The vector $\mathbf{w}$ is normal to the decision boundary. Thus, $\frac{\mathbf{w}}{||\mathbf{w}||}$ is the unit normal.
- The vector from the B to A is $\gamma_i \frac{\mathbf{w}}{||\mathbf{w}||}$.
- B, the point on the decision boundary nearest $\mathbf{x_i}$, is $\mathbf{x_i} - \gamma_i \frac{\mathbf{w}}{||\mathbf{w}||}$.
- As B is on the decision boundary,

$$\mathbf{w} \cdot (\mathbf{x_i} - \gamma_i \frac{\mathbf{w}}{||\mathbf{w}||}) + w_0 = 0$$

- Solving for γ_i yields

$$\gamma_i = \frac{\mathbf{w}}{||\mathbf{w}||} \cdot \mathbf{x_i} + \frac{w_0}{||\mathbf{w}||}$$

- That's for a positive example. For a negative example, we get that negative of that. In either case, we can write:

$$\gamma_i = y_i \left(\frac{\mathbf{w}}{||\mathbf{w}||} \cdot \mathbf{x_i} + \frac{w_0}{||\mathbf{w}||} \right)$$

The margin

- From the previous definition, the margin of the hyperplane is $2M$, where

$$M = \min_i \gamma_i$$

- The most direct statement of the problem of finding a maximum margin separating hyperplane is thus

$$\begin{aligned} & \max_{\mathbf{w}, w_0} \min_i \gamma_i \\ \equiv \quad & \max_{\mathbf{w}, w_0} \min_i y_i \left(\frac{\mathbf{w}}{||\mathbf{w}||} \cdot \mathbf{x_i} + \frac{w_0}{||\mathbf{w}||} \right) \end{aligned}$$

- This turns out to be inconvenient for optimization, however. . .
- Aside: The "distances" γ_i were computed assuming that $\mathbf{w}, w_0$ define a separating hyperline. If they do not–i.e., some data is misclassified–then at least one γ_i will be negative. Thus, maximizing $\min_i \gamma_i$ not only gives us a separating hyperplane (assuming one exists), it also gives us one with maximum margin.

Treating the γ_i as constraints

- From the definition of margin, we have:

$$M \leq \gamma_i = y_i \left(\frac{\mathbf{w}}{||\mathbf{w}||} \cdot \mathbf{x_i} + \frac{w_0}{||\mathbf{w}||} \right) \qquad \forall i$$

- This suggests:

$$\begin{aligned} \text{maximize} \quad & M \\ \text{with respect to} \quad & \mathbf{w}, w_0 \\ \text{subject to} \quad & y_i \left(\frac{\mathbf{w}}{\|\mathbf{w}\|} \cdot \mathbf{x}_i + \frac{w_0}{\|\mathbf{w}\|} \right) \geq M \text{ for all } i \end{aligned}$$

Treating the γ_i as constraints

- From the definition of margin, we have:

$$M \leq \gamma_i = y_i \left(\frac{\mathbf{w}}{\|\mathbf{w}\|} \cdot \mathbf{x_i} + \frac{w_0}{\|\mathbf{w}\|} \right) \qquad \forall i$$

- This suggests:

$$\begin{array}{rl} \text{maximize} & M \\ \text{with respect to} & \mathbf{w}, w_0 \\ \text{subject to} & y_i \left(\frac{\mathbf{w}}{\|\mathbf{w}\|} \cdot \mathbf{x}_i + \frac{w_0}{\|\mathbf{w}\|} \right) \geq M \text{ for all } i \end{array}$$

- Problems:
  - $\mathbf{w}$ appears nonlinearly in the constraints.
  - This problem is underconstrained. If $(\mathbf{w}, w_0, M)$ is an optimal solution, then so is $(\beta\mathbf{w}, \beta w_0, M)$ for any $\beta > 0$.

Adding a constraint

- Let's try adding the constraint that $\|\mathbf{w}\|M = 1$.
- This allows us to rewrite the objective function and constraints as:

$$\begin{aligned} \text{min} \quad & \|\mathbf{w}\| \\ \text{w.r.t.} \quad & \mathbf{w}, w_0 \\ \text{s.t.} \quad & y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) \geq 1 \end{aligned}$$

- This is really nice because the constraints are linear.
- The objective function $\|\mathbf{w}\|$ is still a bit awkward.

Final formulation

- Let's maximize $\|\mathbf{w}\|^2$ instead of $\|\mathbf{w}\|$.
  (Taking the square is a monotone transformation, as $\|weights\|$ is postive, so this doesn't change the optimal solution.)
- This gets us to:

$$\begin{aligned} \text{min} \quad & \|\mathbf{w}\|^2 \\ \text{w.r.t.} \quad & \mathbf{w}, w_0 \\ \text{s.t.} \quad & y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) \geq 1 \end{aligned}$$

- This we can solve! How?

Final formulation

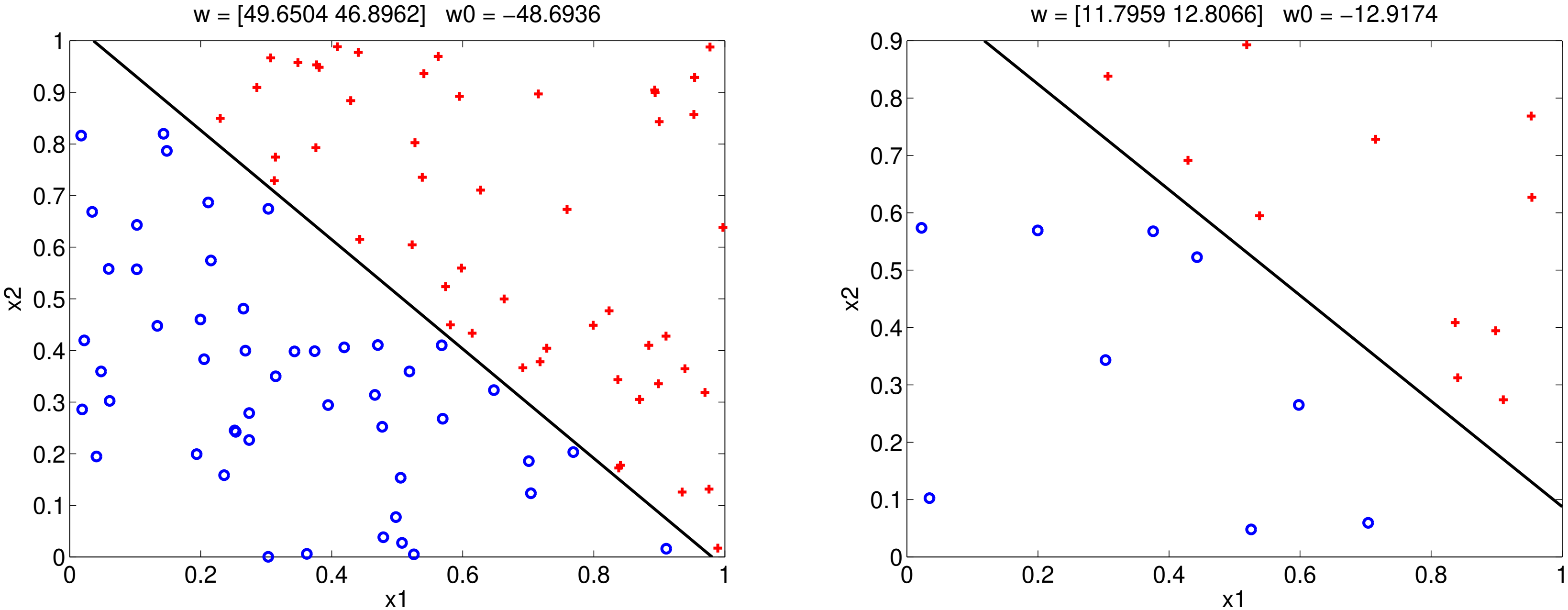
- Let's maximize $\|\mathbf{w}\|^2$ instead of $\|\mathbf{w}\|$.
  (Taking the square is a monotone transformation, as $\|weights\|$ is postive, so this doesn't change the optimal solution.)
- This gets us to:

$$\begin{aligned} \text{min} \quad & \|\mathbf{w}\|^2 \\ \text{w.r.t.} \quad & \mathbf{w}, w_0 \\ \text{s.t.} \quad & y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) \geq 1 \end{aligned}$$

- This we can solve! How?
  - It is a *quadratic programming* (QP) problem—a standard type of optimization problem for which many efficient softwares are available.
  - Better yet, it's a convex (positive semidefinite) QP
  - Computational complexity?

A pause to think

- We could stop here. This QP gives us a unique solution to finding a maximum margin perceptron / linear SVM. E.g., from MATLAB:



- However, we're going to look at solving the same optimization problem another way. The formulation we've arrived at:
  - Doesn't show us the "kernel trick"
  - Doesn't show us the "support vectors"

Lagrange multipliers for inequality constraints

- Suppose we have the following optimization problem, called **primal**:

$$\min_{\mathbf{w}} f(\mathbf{w})$$
$$\text{such that } g_i(\mathbf{w}) \quad \leq \quad 0, \; i = 1 \ldots k$$

- We define the **generalized Lagrangian**:

$$L(\mathbf{w}, \alpha) = f(\mathbf{w}) + \sum_{i=1}^{k} \alpha_i g_i(\mathbf{w}), \tag{1}$$

where $\alpha_i, \; i = 1 \ldots k$ are the Lagrange multipliers.

A different optimization problem

- Consider $\mathcal{P}(\mathbf{w}) = \max_{\alpha:\alpha_i \geq 0} L(\mathbf{w}, \alpha)$
- Observe that the follow is true. Why?

$$\mathcal{P}(\mathbf{w}) = \begin{cases} f(\mathbf{w}) & \text{if all constraints are satisfied} \\ +\infty & \text{otherwise} \end{cases}$$

- Hence, instead of computing $\min_{\mathbf{w}} f(\mathbf{w})$ subject to the original constraints, we can compute:

$$p^* = \min_{\mathbf{w}} \mathcal{P}(\mathbf{w}) = \min_{\mathbf{w}} \max_{\alpha:\alpha_i \geq 0} L(\mathbf{w}, \alpha)$$

Dual optimization problem

- Let $d^* = \max_{\alpha:\alpha_i \geq 0} \min_{\mathbf{w}} L(\mathbf{w}, \alpha)$ (max and min are reversed)
- We can show that $d^* \leq p^*$.
  - Let $p^* = L(w^p, \alpha^p)$
  - Let $d^* = L(w^d, \alpha^d)$
  - Then $d^* = L(w^d, \alpha^d) \leq L(w^p, \alpha^d) \leq L(w^p, \alpha^p) = p^*$.)

Dual optimization problem

- If f, g_i are convex and the g_i can all be satisfied simultaneously for some $\mathbf{w}$, then we have equality: $d^* = p^* = L(\mathbf{w}^*, \alpha^*)$
- Moreover $\mathbf{w}^*, \alpha^*$ solve the primal and dual if and only if they satisfy the following conditions (called Karush-Kuhn-Tucker):

$$
\begin{aligned}
\frac{\partial}{\partial w_i} L(\mathbf{w}^*, \alpha^*) &= 0,\ i = 1 \ldots n & (2)\\
\alpha_i^* g_i(\mathbf{w}^*) &= 0,\ i = 1 \ldots k & (3)\\
g_i(\mathbf{w}^*) &\leq 0,\ i = 1 \ldots k & (4)\\
\alpha_i^* &\geq 0,\ i = 1 \ldots k & (5)
\end{aligned}
$$

Back to maximum margin perceptron

- We wanted to solve (rewritten slightly):

$$\begin{array}{rl} \text{min} & \frac{1}{2}\|\mathbf{w}\|^2 \\ \text{w.r.t.} & \mathbf{w}, w_0 \\ \text{s.t.} & 1 - y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) \leq 0 \end{array}$$

- The Lagrangian is:

$$L(\mathbf{w}, w_0, \alpha) = \frac{1}{2}\|\mathbf{w}\|^2 + \sum_i \alpha_i (1 - y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0))$$

- The *primal problem* is: $\min_{\mathbf{w}, w_0} \max_{\alpha : \alpha_i \geq 0} L(\mathbf{w}, w_0, \alpha)$
- We will solve the *dual problem*: $\max_{\alpha : \alpha_i \geq 0} \min_{\mathbf{w}, w_0} L(\mathbf{w}, w_0, \alpha)$
- In this case, the optimal solutions coincide, because we have a quadratic objective and linear constraints (both of which are convex).

Solving the dual

- From KKT condition (2), we know that the derivatives of $L(\mathbf{w}, w_0, \alpha)$ wrt $\mathbf{w}, w_0$ should be 0
- The condition on the derivative wrt w_0 gives $\sum_i \alpha_i y_i = 0$
- The condition on the derivative wrt $\mathbf{w}$ gives:

$$\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x_i}$$

⇒ Just like for the perceptron with zero initial weights, the optimal solution for $\mathbf{w}$ is a linear combination of the $\mathbf{x_i}$, and likewise for w_0.

- The output is

$$h_{\mathbf{w},w_0}(\mathbf{x}) = \mathrm{sgn}\left(\sum_{i=1}^{m} \alpha_i y_i (\mathbf{x_i} \cdot \mathbf{x}) + w_0\right)$$

⇒ Output depends on weighted dot product of input vector with training examples

Solving the dual (II)

- By plugging these back into the expression for L, we get:

$$\max_{\alpha} \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} y_i y_j \alpha_i \alpha_j (\mathbf{x_i} \cdot \mathbf{x_j})$$

with constraints: $\alpha_i \geq 0$ and $\sum_i \alpha_i y_i = 0$

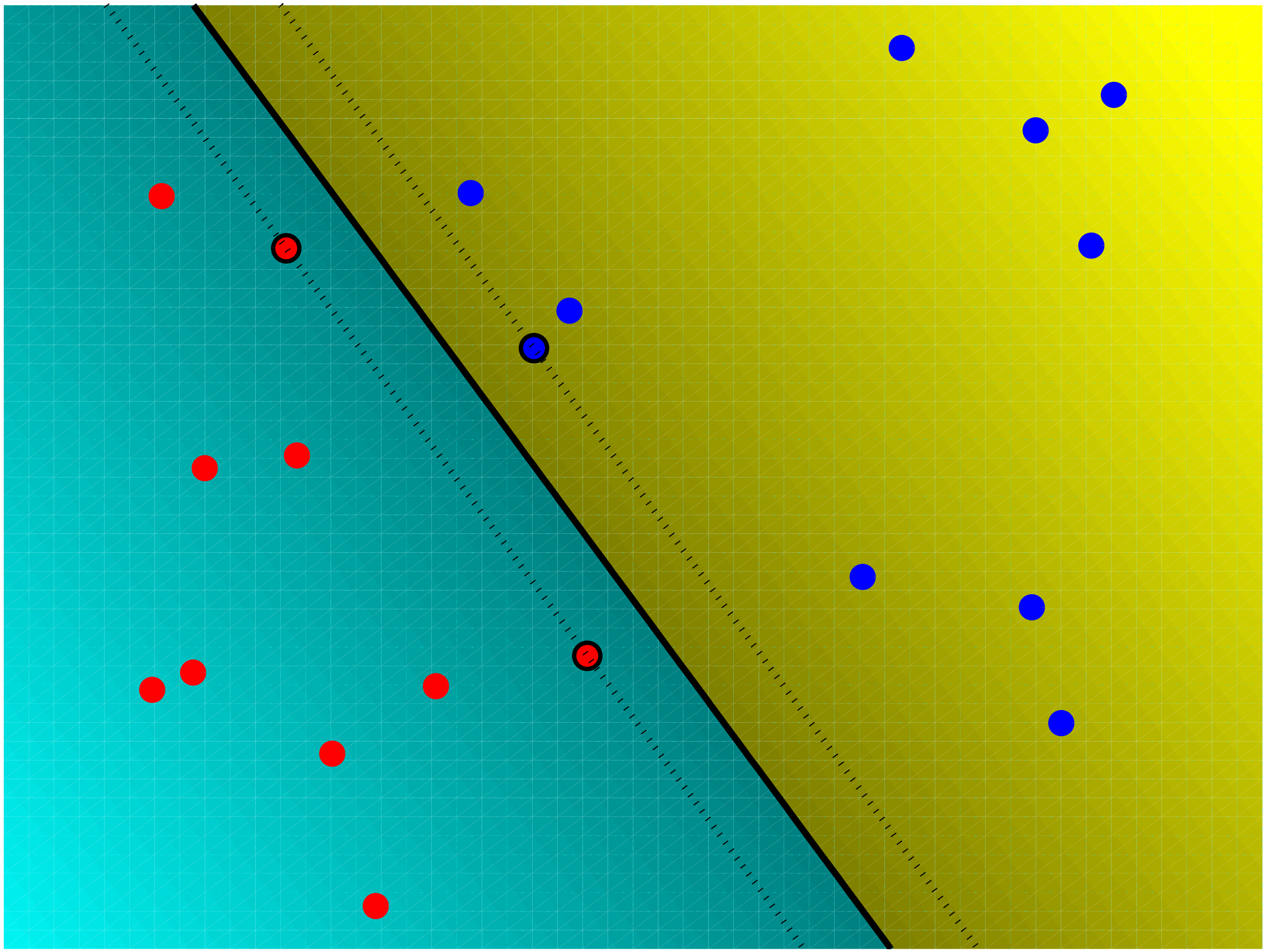
The support vectors

- Suppose we find optimal α's (e.g., obtained by a standard QP package)
- The α_i will be > 0 only for the points for which $1 - y_i(\mathbf{w} \cdot \mathbf{x_i} + w_0) = 0$
- These are the points lying on the edge of the margin, and they are called **support vectors**, because they define the decision boundary
- The output of the classifier for query point $\mathbf{x}$ is computed as:

$$\text{sgn}\left(\sum_{i=1}^{m} \alpha_i y_i (\mathbf{x_i} \cdot \mathbf{x}) + w_0\right)$$

  Hence, the output is determined by computing the dot product of the point with the support vectors!

Example



Support vectors are in bold

Summary of perceptrons and linear SMVs

- Perceptron is essentially just a linear decision boundary, with positive class on one side, negative on the other
- Perceptron learning rule allows finding a decision boundary if the data is linearly separable... but it's non-unique
- The margin is twice the distance from the decision boundary to the nearest instance
- A linear support vector machine is a perceptron that maximizes the margin
- It can be found by quadratic programming
- The solution weights are a linear combination of support vectors—the training instances closest to the decision boundary (a fact made clear by the transformation to the dual)