# More Large-Scale Machine Learning

## Perceptrons
## Support-Vector Machines

Jeffrey D. Ullman
Stanford University

# The Perceptron

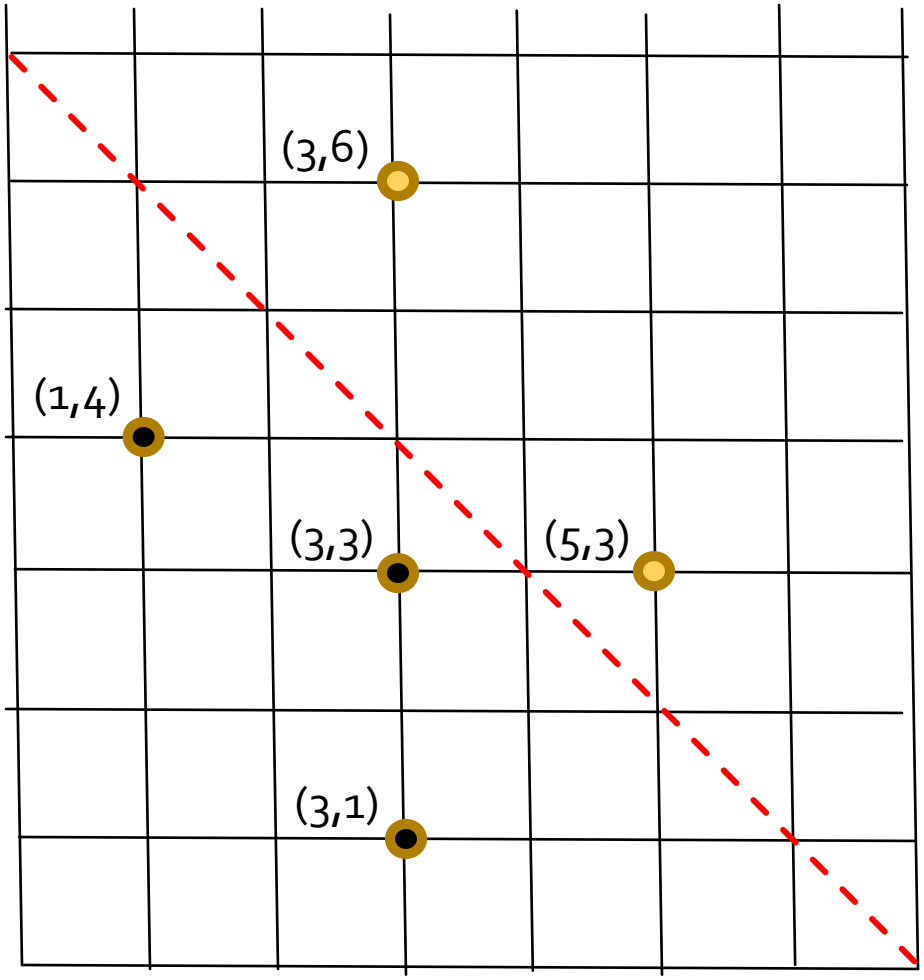
- Given a set of training points (**x**, y), where:
  1. **x** is a real-valued vector of d dimensions (i.e., a point in a Euclidean space), and
  2. y is a binary decision +1 or -1,

a perceptron tries to find a linear separator between the positive and negative inputs.

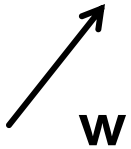
# Linear Separators

- A *linear separator* is a d-dimensional vector **w** and a *threshold* $\theta$ such that the *hyperplane* defined by **w** and $\theta$ separates the positive and negative examples.
- More precisely: given input **x**, the separator returns +1 if **x**.**w** > $\theta$ and returns -1 if not.
  - I.e., the hyperplane is the set of points whose dot product with **w** is $\theta$.

# Example: Linear Separator



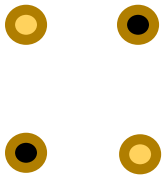

Black points = -1
Gold points = +1
**w** = (1,1)
$\theta = 7$




Hyperplane **x**.**w** = $\theta$
If **x** = (a,b),
then a+b = 7

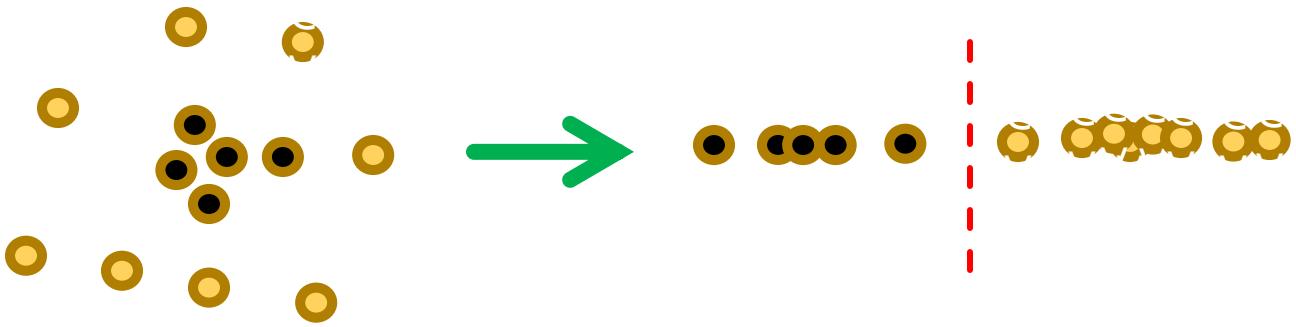
# Goal: Finding w and θ

- Possibly **w** and θ do not exist, since there is no guarantee that the points are linearly separable.
- Example:



# Kernel Functions Can Linearize

- Sometimes, we can transform points that are not linearly separable into a space where they are linearly separable.
- Example: Remember the clustering problem of concentric circles?
- Mapping points to their radii gives us a 1-dimensional space where they are separable.



# Making the Threshold Zero

- A simplification: we can arrange that $\theta = 0$.
- Replace each d-dimensional training point **x** by (**x**,-1), a (d+1)-dimensional vector with -1 as its last component.
- Replace unknown vector **w** (the normal to the separating hyperplane) by (**w**, $\theta$).
  - I.e., add a (d+1)st unknown component, which effectively functions as the threshold.
- Then **x**.**w** > $\theta$ if and only if (**x**,-1).(**w**, $\theta$) > 0.

# Previous Example, Continued

- The positive training points (3,6) and (5,3) become (3,6,-1) and (5,3,-1).
- The negative training points (1,4), (3,3), and (3,1) become (1,4,-1), (3,3,-1), and (3,1,-1).
- Since we know **w** = (1,1) and $\theta = 7$ separated the original points, then **w'** = (1,1,7) and $\theta = 0$ will separate the new points.
- Example: (3,6,-1).(1,1,7) > 0 and (1,4,-1).(1,1,7) $\leq$ 0.

# Training a Perceptron

- Assume threshold = 0.
- Pick a *learning rate* $\eta$, typically a small fraction.
- Start with **w** = (0, 0,…, 0).
- Consider each training example (**x**,y) in turn, until there are no misclassified points.
  - Use y = +1 for positive examples, y = -1 for negative.
- If **x**.**w** has a sign different from y, then this is a misclassified point.
  - Special case: also misclassified if **x**.**w** = 0.

# Training – (2)

- If (**x**,y) is misclassified, adjust **w** to accommodate **x** slightly.
- Replace **w** by **w'** = **w** + $\eta y\mathbf{x}$.
- Note **x**.**w'** = **x**.**w** + $\eta y|\mathbf{x}|^2$.
- That is, if y = +1, then the dot product of **x** with **w'**, which was negative, has been increased by $\eta$ times the square of the length of **x**.
  - Similarly, if y = -1, the dot product has decreased.
  - May still have the wrong sign, but we're headed in the right direction.

# Example: Training

| Name | x | y |
|---|---|---|
| A | (1,4,-1) | -1 |
| B | (3,3,-1) | -1 |
| C | (3,1,-1) | -1 |
| D | (3,6,-1) | +1 |
| E | (5,3,-1) | +1 |

Let $\eta = 1/3$.

$\mathbf{w} = (0, 0, 0)$

Use A: misclassified. New $\mathbf{w}$ =
$(0, 0, 0) + (1/3)(-1)(1,4,-1) = (-1/3, -4/3, 1/3)$.

Use B: OK; Use C: OK.

Use D: misclassified. New $\mathbf{w}$ =
$(-1/3, -4/3, 1/3) + (1/3)(+1)(3,6,-1) = (2/3, 2/3, 0)$.

Use E: OK.

Use A: misclassified. New $\mathbf{w}$ =
$(2/3, 2/3, 0) + (1/3)(-1)(1,4,-1) = (1/3, -2/3, -1/3)$.

. . .

# Parallelization

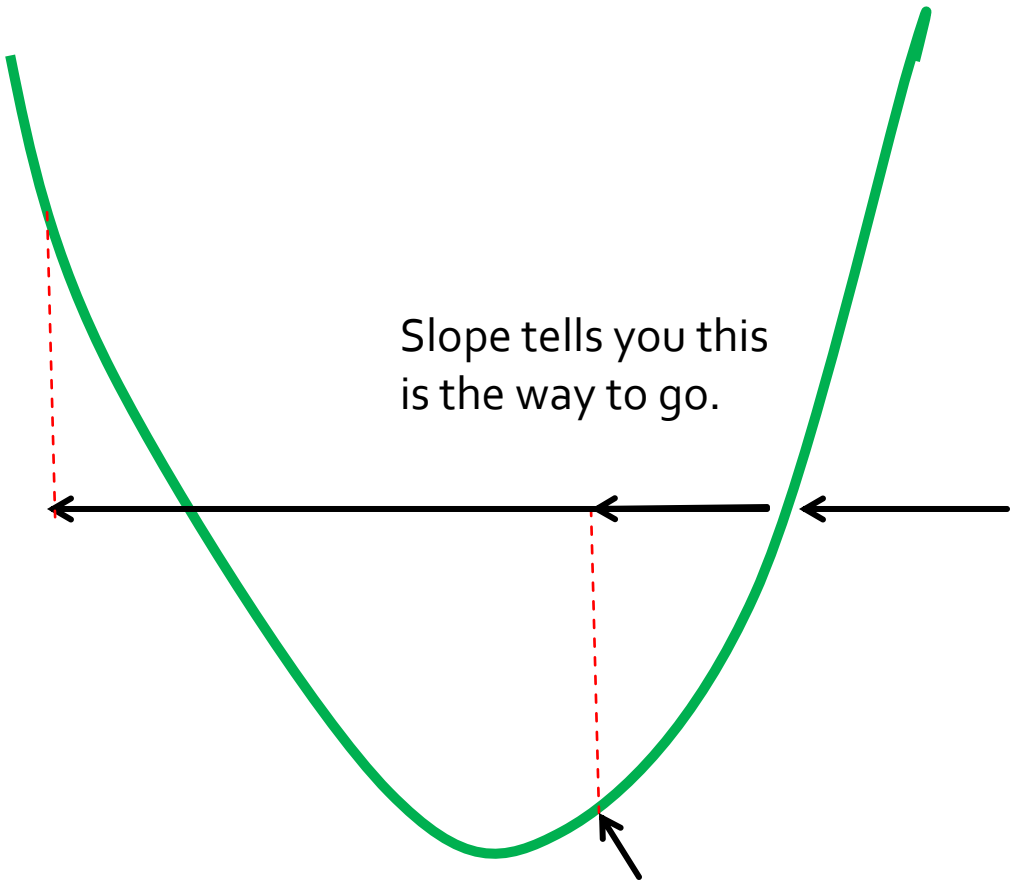
- Convergence is an inherently sequential process.
- We change **w** at each step, which can change:
  1. Which training points are misclassified.
  2. What the next vector **w'** is.
- However, if the learning rate is small, these changes are not great at each step.
- It is generally safe to process many training points at once, obtain the increments to **w** for each, and add them all at once.

# Picking the Training Rate

- A very small training rate causes convergence to be slow.
- Too large a training rate can cause oscillation and may make convergence impossible, even if the training points are linearly separable.

# The Problem With High Training Rate

But if you travel too far in the right direction, you actually can make things worse.



# The Winnow Algorithm

- Perceptron learning for binary training examples.
  - I.e., assume components of input vector **x** are 0 or 1; outputs y are -1 or +1.
- Uses a threshold $\theta$, usually the number of dimensions of the input vector or half that number.
- Select a training rate $0 < \eta < 1$.
- Initial weight vector **w** is (1, 1,…, 1).

# Winnow Algorithm – (2)

- Visit each training example (**x**,y) in turn, until convergence.
- If **x**.**w** > $\theta$ and y = +1, or **x**.**w** < $\theta$ and y = -1, we're OK, so make no change to **w**.
- If **x**.**w** $\geq$ $\theta$ and y = -1, lower each component of **w** where **x** has value 1.
  - More precisely: IF $x_i = 1$ THEN replace $w_i$ by $\eta w_i$.
- If **x**.**w** $\leq$ $\theta$ and y = +1, raise each component of **w** where **x** has value 1.
  - More precisely: IF $x_i = 1$ THEN replace $w_i$ by $w_i/\eta$.

# Example: Winnow Algorithm

| Viewer | Star Wars | Martian | Aveng-ers | Titanic | Lake House | You've Got Mail | y |
|---|---|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 | 1 | 0 | +1 |
| B | 1 | 1 | 1 | 0 | 0 | 0 | +1 |
| C | 0 | 1 | 0 | 1 | 1 | 0 | -1 |
| D | 0 | 0 | 0 | 1 | 0 | 1 | -1 |
| E | 1 | 0 | 1 | 0 | 0 | 1 | +1 |

Goal is to classify "Scifi" viewers (+1) versus "Romance" (-1).

Initial w = (1, 1, 1, 1, 1, 1).

Threshold: $\theta = 6$.

Use $\eta = 1/2$.

# Example: Winnow – (2)

| | S | M | A | T | L | Y | y |
|---|---|---|---|---|---|---|---|
| A | 0 | 1 | 1 | 1 | 1 | 0 | +1 |
| B | 1 | 1 | 1 | 0 | 0 | 0 | +1 |
| C | 0 | 1 | 0 | 1 | 1 | 0 | -1 |
| D | 0 | 0 | 0 | 1 | 0 | 1 | -1 |
| E | 1 | 0 | 1 | 0 | 0 | 1 | +1 |

**w** = (1, 1, 1, 1, 1, 1).

Use A: misclassified. **x**.**w** = 4 ≤ 6.
New **w** = (1, 2, 2, 2, 2, 1).

Use B: misclassified. **x**.**w** = 5 ≤ 6.
New **w** = (2, 4, 4, 2, 2, 1).

Use C: misclassified. **x**.**w** = 8 > 6.
New **w** = (2, 2, 4, 1, 1, 1).

Now, D, E, A, B, C are all OK, so done.

Question for thought:
Would this work if inputs were arbitrary reals, not just 0, 1?

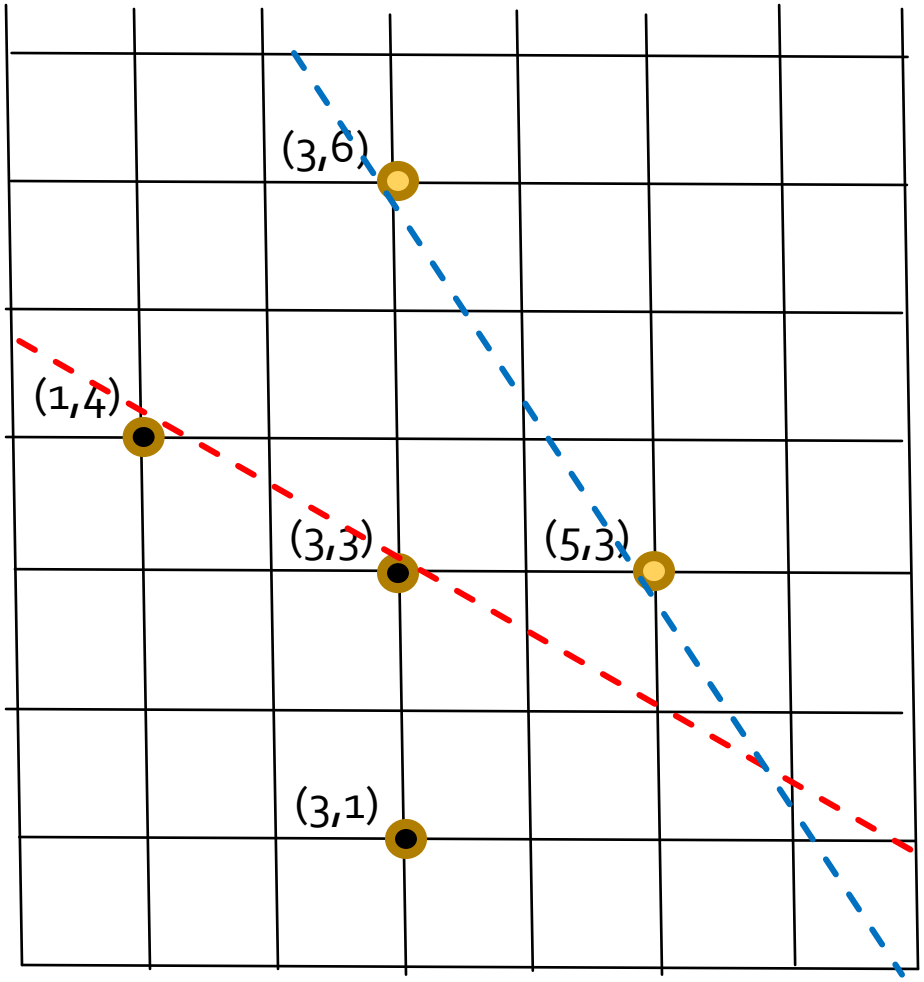
# Support-Vector Machines

- Problem with Perceptrons
- Linearly Separable Data
- Dealing with Nonseparable Data

# Problems With Perceptrons

1. Not every dataset is linearly separable.
   - More common: a dataset is "almost" separable, but with a small fraction of the points on the wrong side of the boundary.
2. Perceptron design stops as soon as a linear separator is found.
   - May not be the best boundary for separating the data to which the perceptron is applied, even if the training data is a random sample from the full dataset.

# Example: Problem



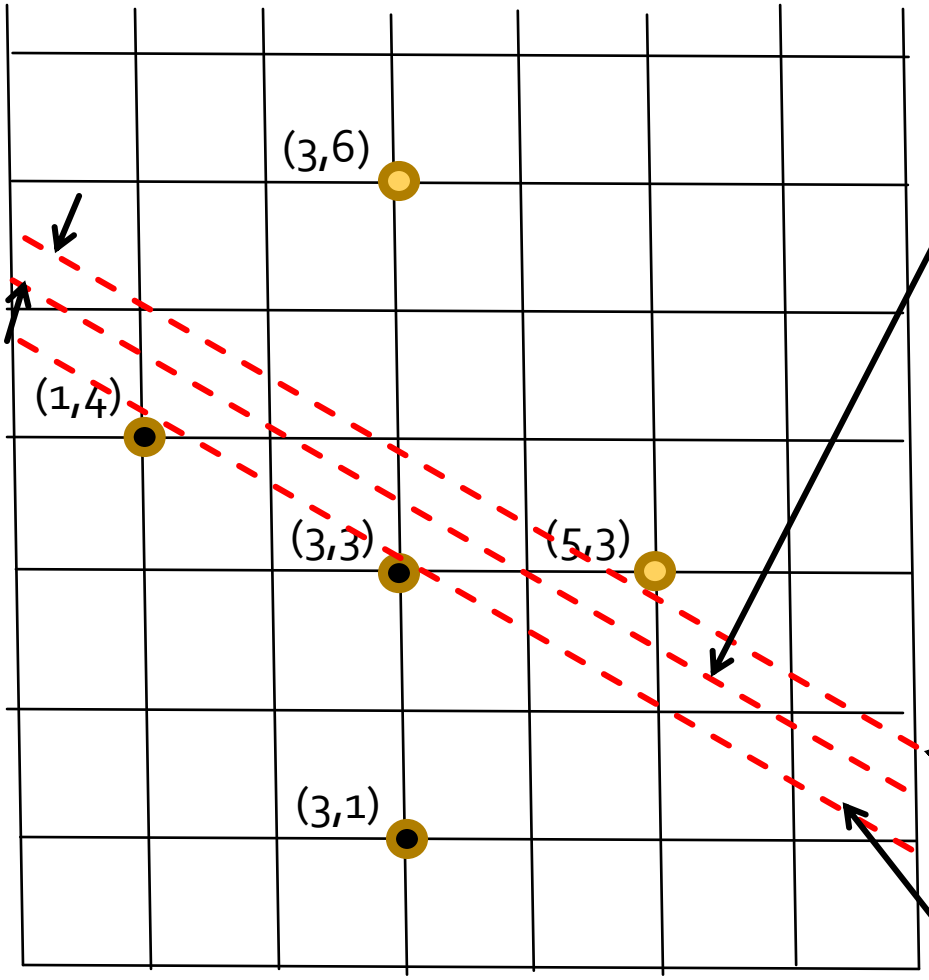

Either red or blue line separates training points. Can give different answers for many points.

# Intuition Behind SVM

- By designing a better cost function, we can force the separating hyperplane to be as far as possible from the points in either class.
  - Reduces the likelihood that points in the test or validation sets will be misclassified.
- Later, we'll also consider picking a hyperplane for nonseparable data, in a way that minimizes the "damage."

# Example: One Candidate

Margin $\gamma$

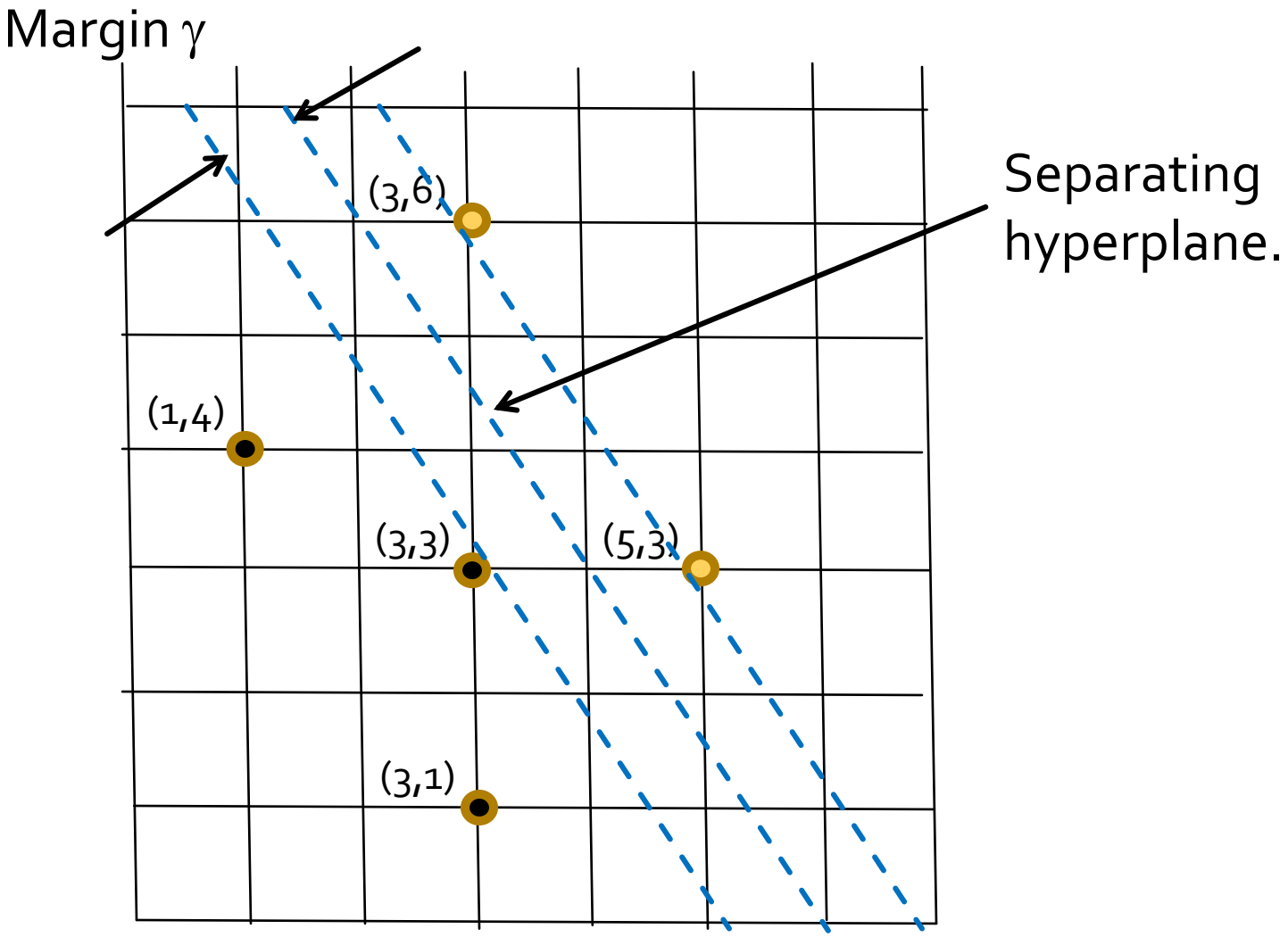



Separating hyperplane.

(1,4), (3,3), and (5,3) are the *support vectors*, limiting the margin for this choice of hyperplane.

Call these the "upper" and "lower" hyperplanes.

# Example: Hyperplane With Larger γ



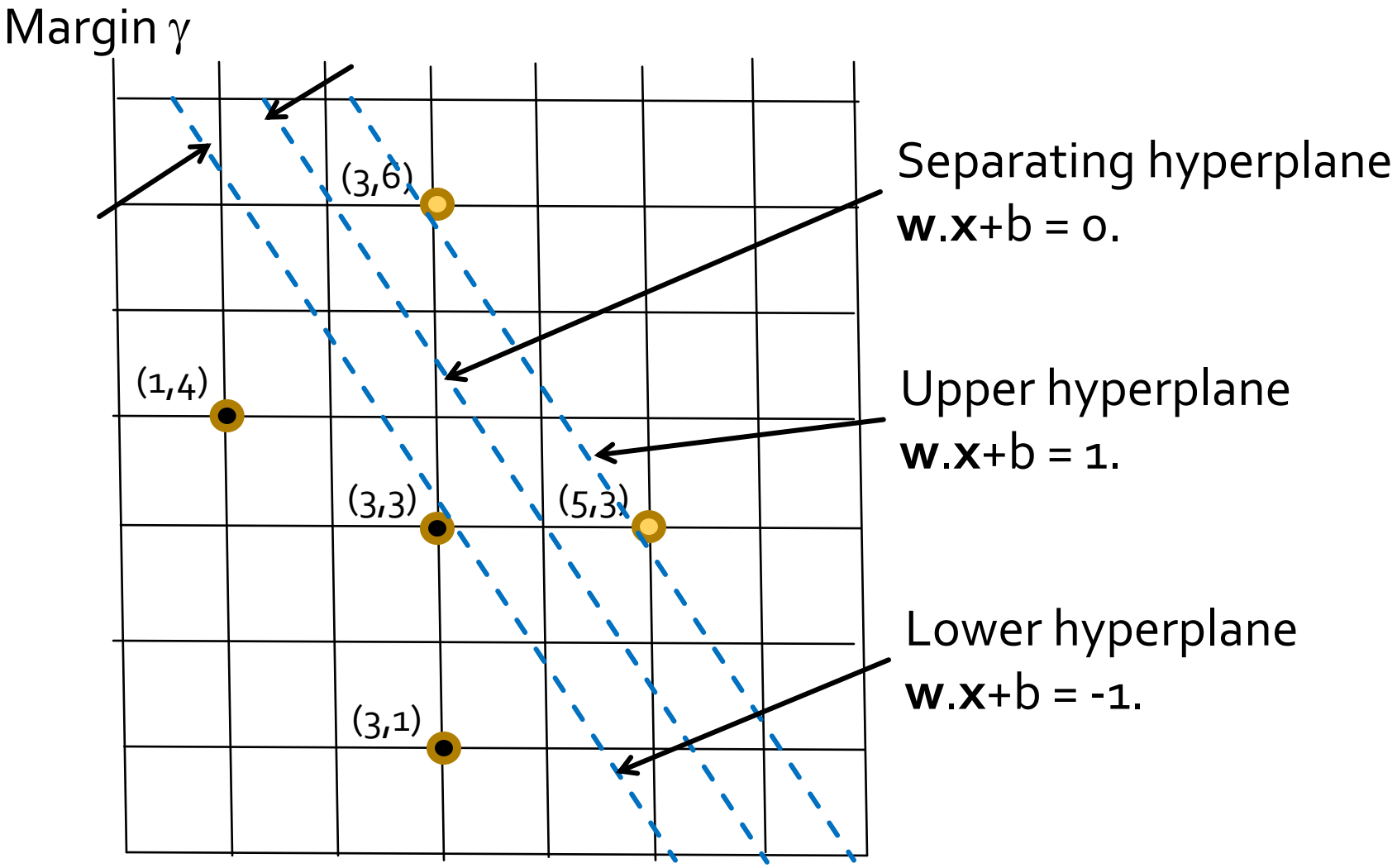
# Maximizing $\gamma$

- Goal: find **w** (the normal to the separating hyperplane) and b (the constant that positions the separating hyperplane) to maximize $\gamma$, subject to the constraints that for each training example (**x**,y), we have y(**w**.**x** + b) $\geq \gamma$.
  - That is, if y = +1, then point **x** is at least $\gamma$ above the separating hyperplane, and if y = -1, then **x** is at least $\gamma$ below.
- Problem: scale of **w** and b.
  - Double **w** and b and we can double $\gamma$.

# Maximizing $\gamma$ – (2)

- Solution: require $|\mathbf{w}|$ to be the unit of length for $\gamma$.
- Equivalent formulation: require that the constant terms in the upper and lower hyperplanes (those that are parallel to the separating hyperplanes, but just touch the support vectors) be b+1 and b-1.
- The problem of maximizing $\gamma$, computed in units of $|\mathbf{w}|$, is equivalent to minimizing $|\mathbf{w}|$ subject to the constraint that all points are outside the upper and lower hyperplanes.
  - Why? We forced the margin to be 1, so the smaller $\mathbf{w}$ is, the larger $\gamma$ looks in units of $|\mathbf{w}|$.

# Example: Unit Separation



# Example: Constraints

- Consider the running example, with positive points (3,6) and (5,3), and with negative points (1,4), (3,3), and (3,1).
- Let **w** = (u,v).
- Then we must minimize |**w**| subject to:
  - $3u + 6v + b \geq 1$.
  - $5u + 3v + b \geq 1$.
  - $u + 4v + b \leq -1$.
  - $3u + 3v + b \leq -1$.
  - $3u + v + b \leq -1$.

# Solving the Constraints

- This is almost a linear program.
- Difference: the objective function sqrt($u^2+v^2$) is not linear.
- Cheat: if we believe the blue hyperplane with support vectors (3,6), (5,3), and (3,3) is the best we can do, then we know that the normal to this hyperplane has v = 2u/3, and we only have to minimize u.

# Solving the Constraints if v = 2u/3

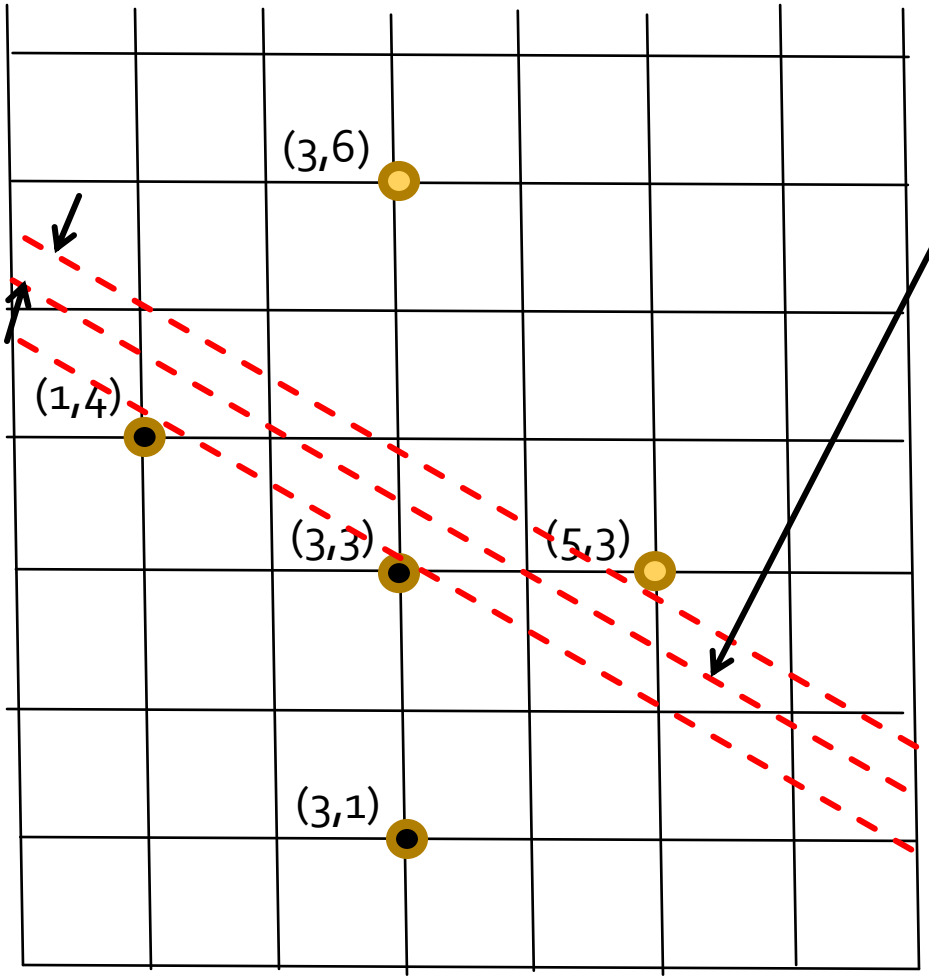
| Point | Constraint | If v = 2u/3 |
|---|---|---|
| (3,6) | $3u + 6v + b \geq 1$ | **$7u + b \geq 1$** |
| (5,3) | $5u + 3v + b \geq 1$ | **$7u + b \geq 1$** |
| (1,4) | $u + 4v + b \leq -1$ | $11u/3 + b \leq -1$ |
| (3,3) | $3u + 3v + b \leq -1$ | **$5u + b \leq -1$** |
| (3,1) | $3u + v + b \leq -1$ | $11u/3 + b \leq -1$ |

Constraints of support vectors are hardest to satisfy.
Smallest u is when u = 1, v = 2/3, b = -6.

$|\mathbf{w}| = \text{sqrt}(1^2 + (2/3)^2) = 1.202.$

# Remember This Hyperplane With a Smaller Margin?

Margin $\gamma$




Separating hyperplane.

The normal to the hyperplane, **w**, has slope 2, so v = 2u.

# Here's What Happens if v = 2u

| Point | Constraint | If v = 2u |
|---|---|---|
| (3,6) | $3u + 6v + b \geq 1$ | $15u + b \geq 1$ |
| (5,3) | $5u + 3v + b \geq 1$ | **$11u + b \geq 1$** |
| (1,4) | $u + 4v + b \leq -1$ | **$9u + b \leq -1$** |
| (3,3) | $3u + 3v + b \leq -1$ | **$9u + b \leq -1$** |
| (3,1) | $3u + v + b \leq -1$ | $5u + b \leq -1$ |

Constraints of support vectors are hardest to satisfy.
Smallest u is when
u = 1, v = 2, b = -10.

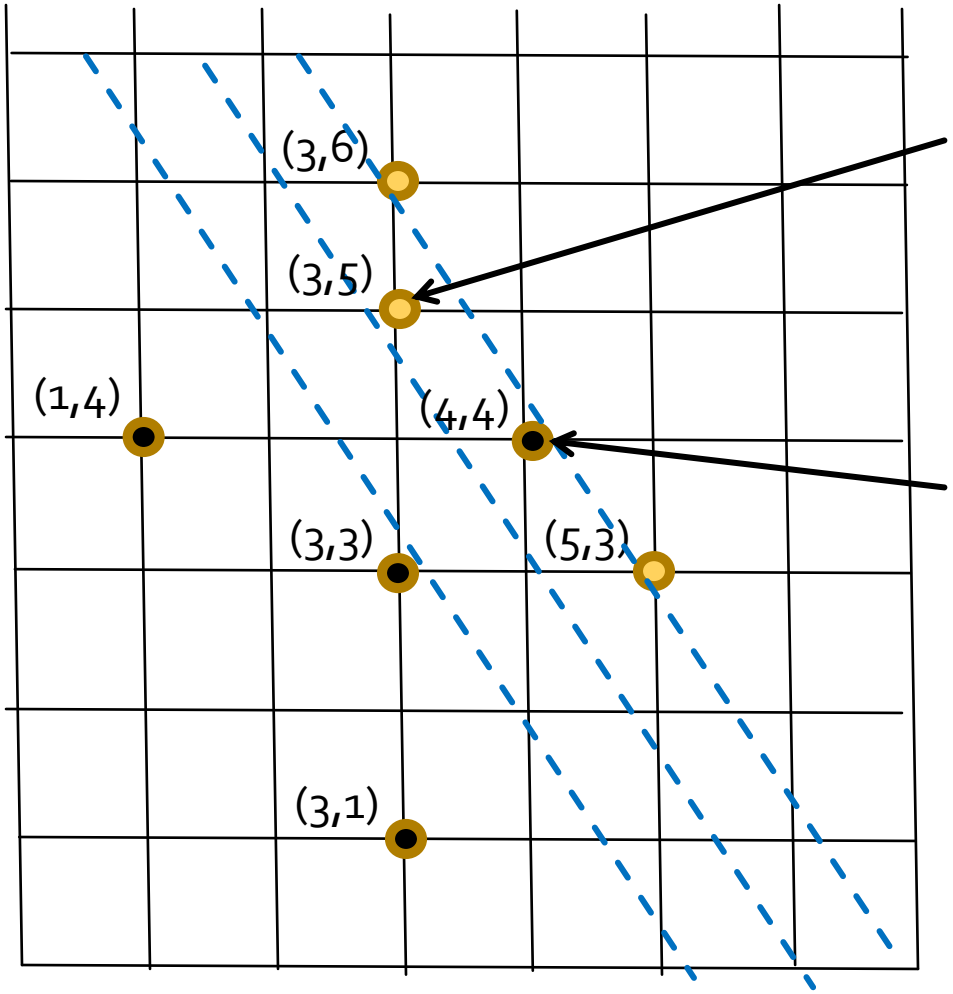
$|\mathbf{w}| = \text{sqrt}(1^2 + 2^2) = 2.236.$

Since we want the minimum $|\mathbf{w}|$, we prefer the previous hyperplane.

# Did That Look Too Easy?

- 2 dimensions is not that hard.
- In general there are d+1 support vectors for d-dimensional data.
- Support vectors must lie on the convex hulls of the sets of positive and negative points.
- Once you find a candidate separating hyperplane and its parallel upper and lower hyperplanes, you can calculate $|\mathbf{w}|$ for that candidate.
- But there is a more general approach, next.

# Nonseparable Data




Correctly classified, but too close to the separating hyperplane

Misclassified

# New Goal

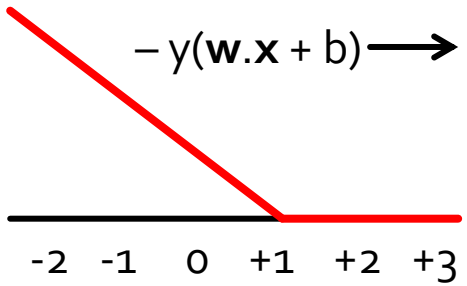
- We'll still assume that we want a "separating" hyperplane $\mathbf{w}.\mathbf{x} + b = 0$ defined by normal vector $\mathbf{w}$ and constant b.
- And to establish the length of $\mathbf{w}$, we take the upper and lower hyperplanes to be $\mathbf{w}.\mathbf{x} + b = +1$ and $\mathbf{w}.\mathbf{x} + b = -1$.
- Allow points to be inside the upper and lower hyperplanes, or even entirely on the wrong side of the separator.

# New Goal – (2)

- Minimize a cost function that includes:
  1. The square of the length of **w** (to encourage a small |**w**|), and
  2. A term that penalizes points that are either:
     a. On the right side of the separator, but on the wrong side of the upper or lower hyperplanes.
     b. On the wrong side of the separator.
- The term (2) is *hinge loss* =
  - 0 if point is on the right side of the upper or lower hyperplane.
  - Otherwise linear in the amount of "wrong."

# Hinge Loss Function

- Let **w**.**x** + b = 0 be the separating hyperplane, and let (**x**, y) be a training example.
- The hinge loss for this point is
  max(0, 1 – y(**w**.**x** + b)).




- Example: If y = +1 and **w**.**x** + b = 2, loss = 0.
  - Point **x** is properly classified and beyond the upper hyperplane.
- Example: If y = +1 and **w**.**x** + b = 1/3, loss = 2/3.
  - Point **x** is properly classified but not beyond the upper hyperplane.
- Example: If y = -1 and **w**.**x** + b = 2, loss = 3.
  - Point **x** is completely misclassified.

# Expression to Be Minimized

- Let there be n training examples ($\mathbf{x}_i$, $y_i$).
- The cost expression:

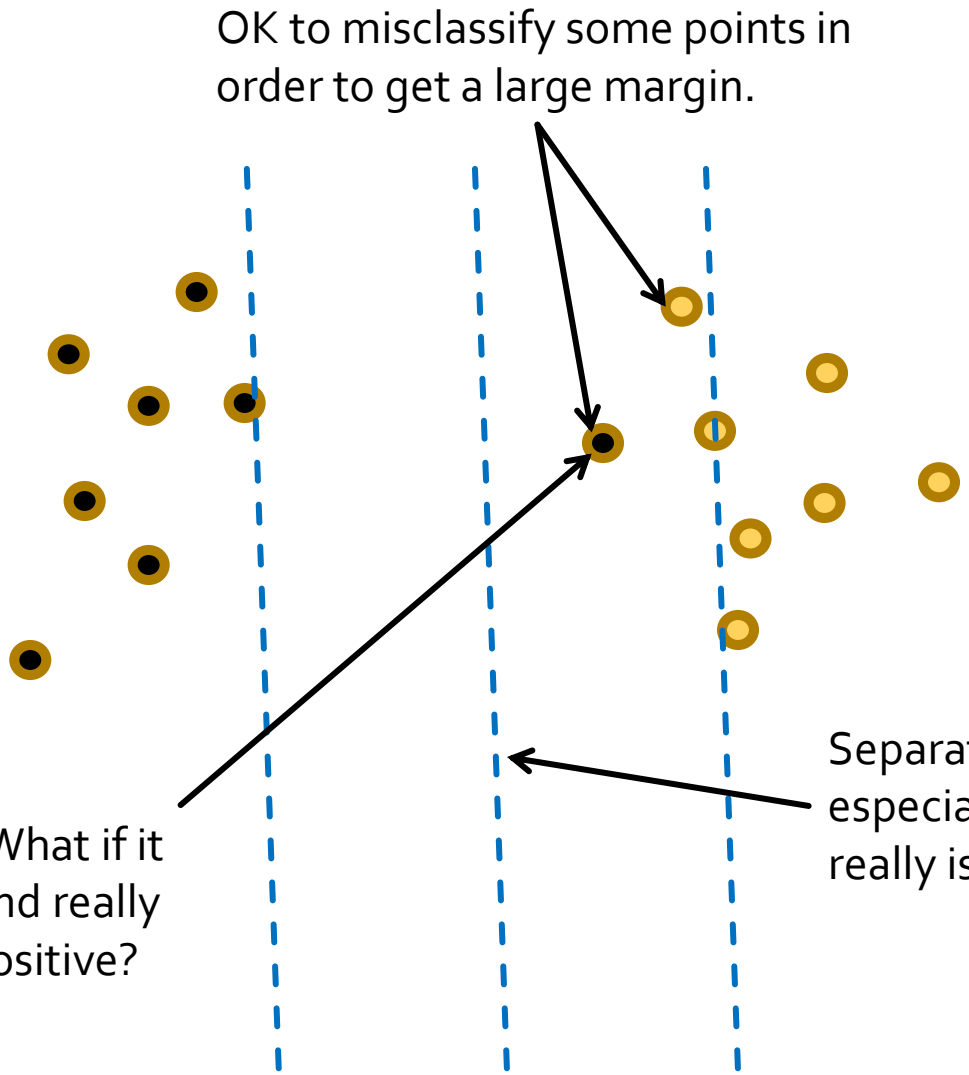
$$f(\mathbf{w}, b) = |\mathbf{w}|^2/2 + C \sum_{j=1,\ldots,n} \max(0, 1 - y_j(\mathbf{w}.\mathbf{x}_j + b))$$

  - C is a constant to be chosen.
- Solve by gradient descent.
- Remember, $\mathbf{w} = (w_1, w_2, \ldots, w_d)$ and each $\mathbf{x}_j = (x_{j1}, x_{j2}, \ldots, x_{jd})$.
- Take partial derivatives with respect to each $w_i$.
- First term has derivative $w_i$.
  - Which, BTW, is why we divided by 2 for convenience.

# Gradient Descent – (2)

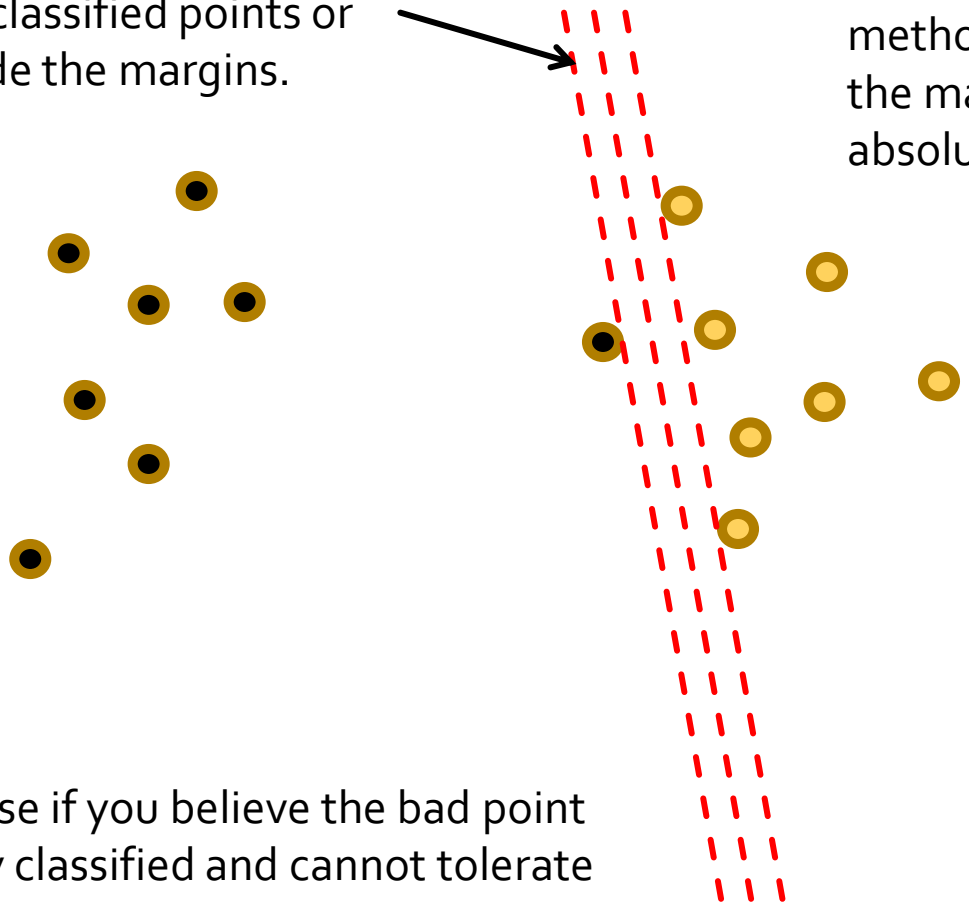
- The second term $C \sum_{j=1,\ldots,n} \max(0, 1 - y_j(\mathbf{w}.\mathbf{x}_j + b))$ is trickier.
- There is one term in the partial derivative with respect to $w_i$ for each j.
- If $y_j(\mathbf{w}.\mathbf{x}_j + b) \geq 1$, then this term is 0.
- But if not, then this term is $-Cy_jx_{ji}$.
- So given the current **w**, you need first to sort out which $\mathbf{x}_j$'s give 0 and which give $-Cy_jx_{ji}$ before you can compute the partial derivatives.

# When C is Small



# When C is Large

Margin must be small so there are no misclassified points or points inside the margins.



Note also: If you use the first method, where points inside the margins are forbidden absolutely, this is what you get.

Makes sense if you believe the bad point is correctly classified and cannot tolerate even a few errors.