Machine Learning

CS 165B

Prof. Matthew Turk

Monday, May 9, 2016

T
o
d
a
y

- Linear learning models (cont.)

Notes

- HW#3 due 4:30pm today
  - I’m available for questions after class until noon today
- Midterm
  - Average = 63.5/76 (84%)
  - Median = 66/76 (87%)
  - Available for pickup (and questions) in tomorrow’s discussion session
- Schedule
  - We’re a little behind the (somewhat optimistic) schedule – I’ll update the Schedule page (topics, reading) later today.

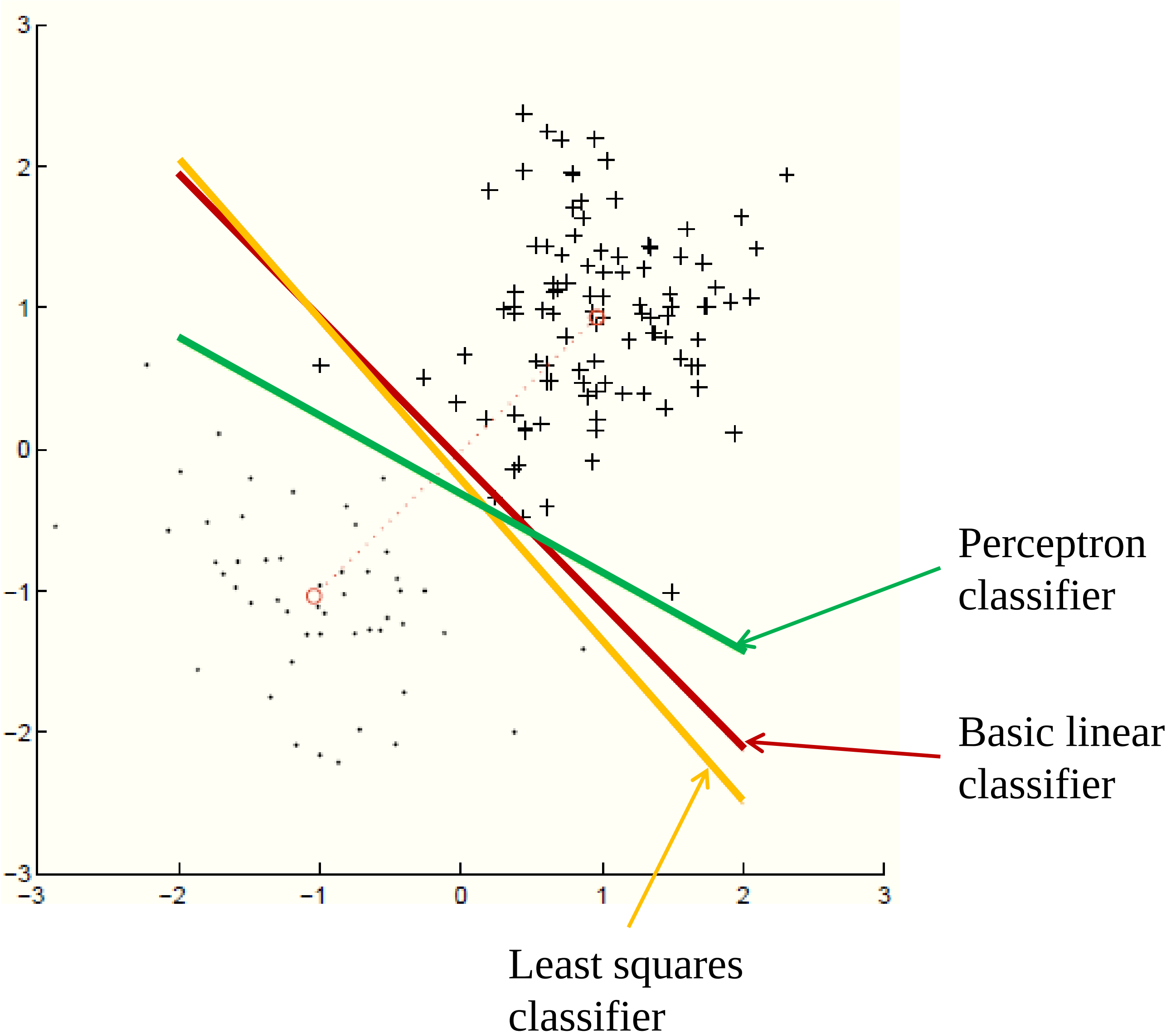
Notes

- Midterm, problem 6

  6. Someone proposes this loss function for a regression problem: $L(x) = {}^{1}/_{(R(x))^2}$, where $R(x)$ is the residual. Explain why this is a good or bad loss function for regression.

  – Should have read $L(R(x)) = {}^{1}/_{(R(x))^2}$

Linear classifier comparison



The perceptron training algorithm

$D = \{ (\mathbf{x}_i, y_i) \}$

Algorithm Perceptron(D, η) – train a perceptron for linear classification.

Input : labelled training data D in homogeneous coordinates; learning rate η.
Output : weight vector $\mathbf{w}$ defining classifier $\hat{y} = \text{sign}(\mathbf{w} \cdot \mathbf{x})$.

```
w ← 0 ;                    // Other initialisations of the weight vector are possible
converged ← false;
while converged = false do
    converged ← true;
    for i = 1 to |D| do
        if  y_i w · x_i ≤ 0          // i.e., ŷ_i ≠ y_i      Misclassified
        then
            w ← w + η y_i x_i ;
            converged ← false;       // We changed w so haven't converged yet
        end
    end
end
```

If a positive example is misclassified, add it to $\boldsymbol{w}$
If a negative example is misclassified, subtract it from $\boldsymbol{w}$

All components of homogeneous $\boldsymbol{w}$ are updated (including $\boldsymbol{w}_{k+1} = -t$)

Perceptron duality

- Every time a training example $\boldsymbol{x}_i$ is misclassified, the amount $\eta y_i \boldsymbol{x}_i$ is added to the weight vector $\boldsymbol{w}$
- After training is completed, each example $\boldsymbol{x}_i$ has been misclassified α_i times
- Thus the weight vector can be written as

$$\boldsymbol{w} = \eta \sum_{\boldsymbol{i}} \alpha_i y_i \boldsymbol{x}_i$$

Assuming the initial value of $\boldsymbol{w}$ was initialize to $\boldsymbol{0}$

So the weight vector is a linear combination of the training instances

- So, alternatively, we can view perceptron learning as learning the α_i coefficients and then, when finished, constructing $\boldsymbol{w}$
  - This perspective comes up again (soon) in support vector machines

Perceptron training in dual form

Algorithm DualPerceptron(D) – perceptron training in dual form.

Input : labelled training data D in homogeneous coordinates.

Output : coefficients α_i defining weight vector $\mathbf{w} = \sum_{i=1}^{|D|} \alpha_i y_i \mathbf{x}_i$.

$\alpha_i \leftarrow 0$ for $1 \le i \le |D|$;
converged←false;
while *converged* = false **do**
  converged←true;
  for $i = 1$ to $|D|$ **do**
    if $y_i \sum_{j=1}^{|D|} \alpha_j y_j \boxed{\mathbf{x}_i \cdot \mathbf{x}_j} \le 0$ **then**
      $\alpha_i \leftarrow \alpha_i + 1$;
      converged←false;
    end
  end
end

Misclassified (from regular Perceptron algorithm)

$\boxed{\text{if } y_i \mathbf{w} \cdot \mathbf{x}_i \le 0 \qquad // \text{ i.e., } \hat{y}_i \ne y_i}$

$$\text{if } y_i \underbrace{\sum_{j=1}^{|D|} \alpha_j y_j \boldsymbol{x}_j}_{\boldsymbol{w}} \cdot \boldsymbol{x}_i \le 0$$

Dot products of training examples

Contained in the Gram matrix $\mathbf{G} = \mathbf{X}^{\mathrm{T}}\mathbf{X}$

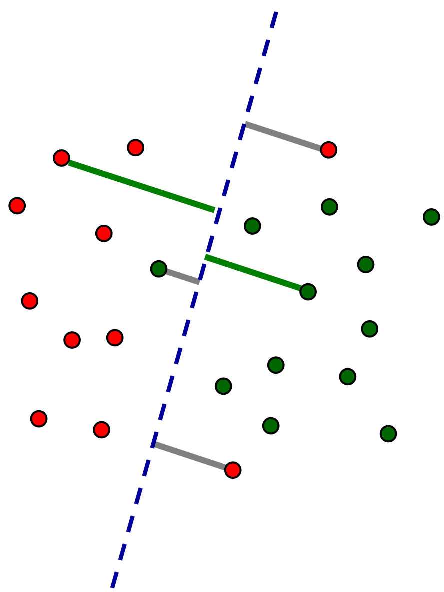
$\mathbf{G}_{ij} = \boldsymbol{x}_i^{\mathrm{T}} \boldsymbol{x}_j$

The Gram matrix is typically computed in advance for computational efficiency

This notation assumes $\mathbf{X}$ is a matrix in which each column is a vector $\boldsymbol{x}_i$

Classifier margin

- The margin ($\boldsymbol{z}$) of a sample is its distance from the classification boundary
  - Positive if it's correctly classified
  - Negative if it's incorrectly classified



Perceptron margin for point $\boldsymbol{x}$:

$$\boldsymbol{z}(\boldsymbol{x}) = \frac{y(\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x} - t)}{||\boldsymbol{w}||} = \frac{m}{||\boldsymbol{w}||}$$

Non-homogeneous representation

Note: m is not the margin; it's the result of plugging $\boldsymbol{x}_i$ into $y(\boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x} - t)$

Margin, distance, and *m*

- Note that on p. 211 in the textbook (beginning of Section 7.3), *m*, margin, and distance are confused

- Ignore the book's terminology here. We use the following:

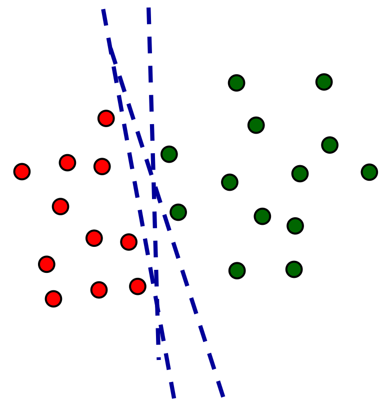
$$m = y(\boldsymbol{w}^T\boldsymbol{x} - t)$$ (not a measure of distance)

margin: $$\boldsymbol{z}(\boldsymbol{x}) = \frac{y(\boldsymbol{wTx} - t)}{||\boldsymbol{w}||} = \frac{m}{||\boldsymbol{w}||}$$ (a measure of distance)

(I think the book is relatively consistent with this terminology elsewhere)

Classifier margin

- The margin of a classifier on a training set is the minimum margin of the data points for that classifier
- The version space of a linear classifier applied to linearly separable data is infinite

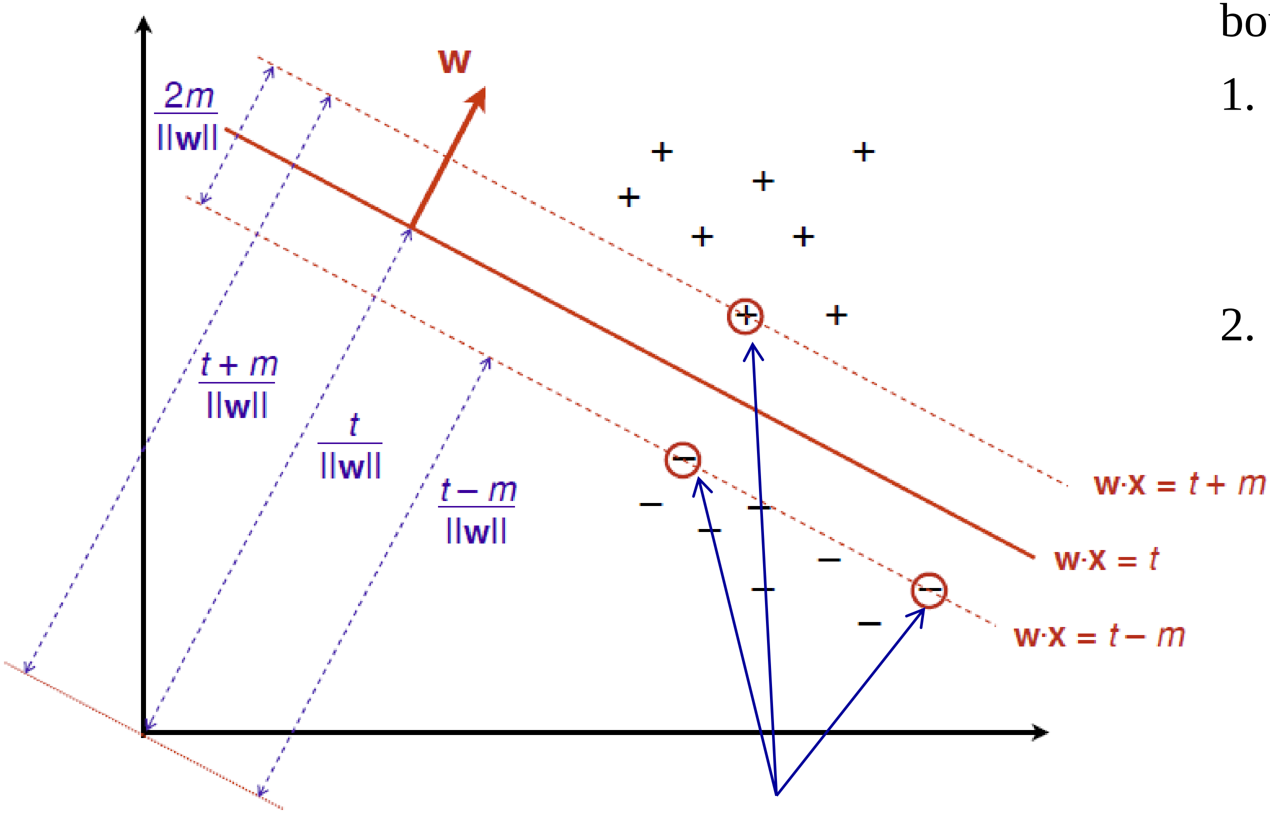


Infinitely many lines can separate the two classes

- What is the best linear classifier?
- Perhaps the one that maximizes some function of the margins
  - E.g., maximize the sum of the smallest margin from each class

Classifier margin

Let's look at the margins for a given training set and decision boundary:



These training samples nearest to the decision boundary are call support vectors

Choose the decision boundary $\boldsymbol{w}^T\boldsymbol{x} - t = 0$ that:

1. Maximizes the sum of the minimum positive class margin and the minimum negative class margin
2. Makes them equal

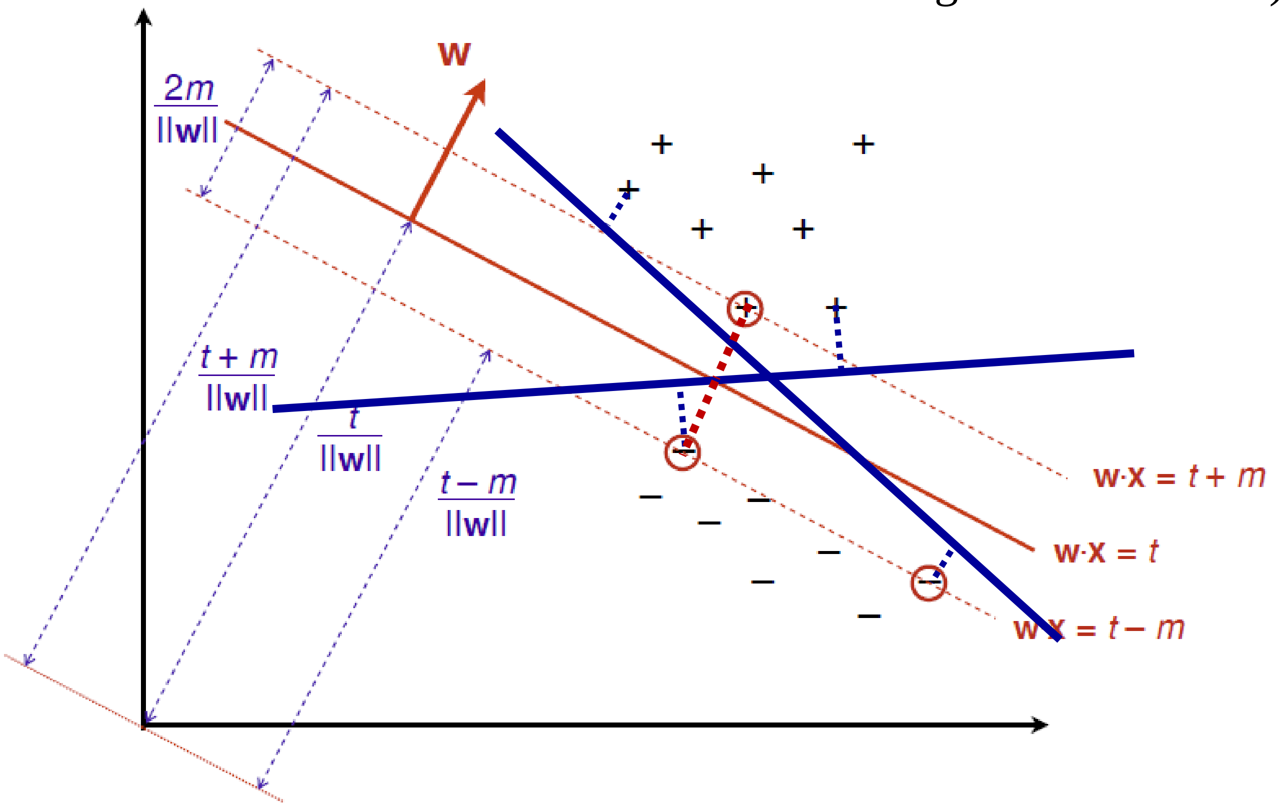
$$\text{Margin}(\boldsymbol{x}) = \frac{y(\boldsymbol{w}^T\boldsymbol{x}-t)}{||\boldsymbol{w}||} = \frac{m}{||\boldsymbol{w}||}$$

or sometimes defined as $\frac{2m}{||\boldsymbol{w}||}$

(the sum of pos. and neg. margins)

Classifier margin

Choose a classification line that maximizes the sum of minimum margins (i.e., maximizes the margin for each class)



Support vector machine (SVM)

- A support vector machine (SVM) is a linear classifier whose decision boundary is a linear combination of the support vectors
- In an SVM, we find classifier parameters $(\boldsymbol{w}, t)$ to maximize the margin
- Since $m = y(\boldsymbol{w}^T\boldsymbol{x} - t)$ and we wish to maximize the margin $\frac{m}{||\boldsymbol{w}||}$, we can instead fix $m = 1$ and minimize $||w||$
  - Provided that none of the training points fall inside the margin
- This leads to a constrained optimization problem:

$$\mathbf{w}^*, t^* = \underset{\mathbf{w},t}{\arg\min} \frac{1}{2}||\mathbf{w}||^2 \qquad \text{subject to } y_i(\mathbf{w}\cdot\mathbf{x}_i - t) \geq 1, 1 \leq i \leq n$$

  - Then, after some magical quadratic optimization (p. 212-214)….

Support vector machine (SVM)

…we get the following result:

$$\boldsymbol{w} = \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i \quad \text{where } \alpha_i \text{ are non-negative reals s.t. } \sum_{i=1}^{n} \alpha_i y_i = 0$$

$\alpha_i > 0$ only for the support vectors!

Other examples $\boldsymbol{x}_i$ for which $\alpha_i = 0$ can be removed from the training set without affecting the learned decision boundary

I.e., the decision boundary is defined only by the (typically few) support vectors from the training set – those that are nearest to the decision boundary (at the margin)

And the weight vector $\boldsymbol{w}$ is merely a linear combination of the (typically few) support vectors

The threshold t can be found by solving $m = 1 = \boldsymbol{w}^T\boldsymbol{x} - t$ for any support vector $\boldsymbol{x}$

Support vector machine (SVM)

- How do we find the α_i values?
  - Via a quadratic optimization solver!
  - But in some simple problems we can do them by hand

Note the pairwise dot products between training instances

$$\alpha_1^*, \ldots, \alpha_n^* = \underset{\alpha_1, \ldots, \alpha_n}{\arg\max} \left[-\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j + \sum_{i=1}^{n} \alpha_i \right]$$

$$\text{subject to } \alpha_i \geq 0, 1 \leq i \leq n \text{ and } \sum_{i=1}^{n} \alpha_i y_i = 0$$

1. Quadratic optimization to solve for $\alpha_1, \ldots, \alpha_n$
   - Non-zero α_i corresponds to support vector $\boldsymbol{x}_i$
2. Create $\boldsymbol{w} = \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i$
3. Solve for t by plugging in for any support vector $\boldsymbol{x}_i$

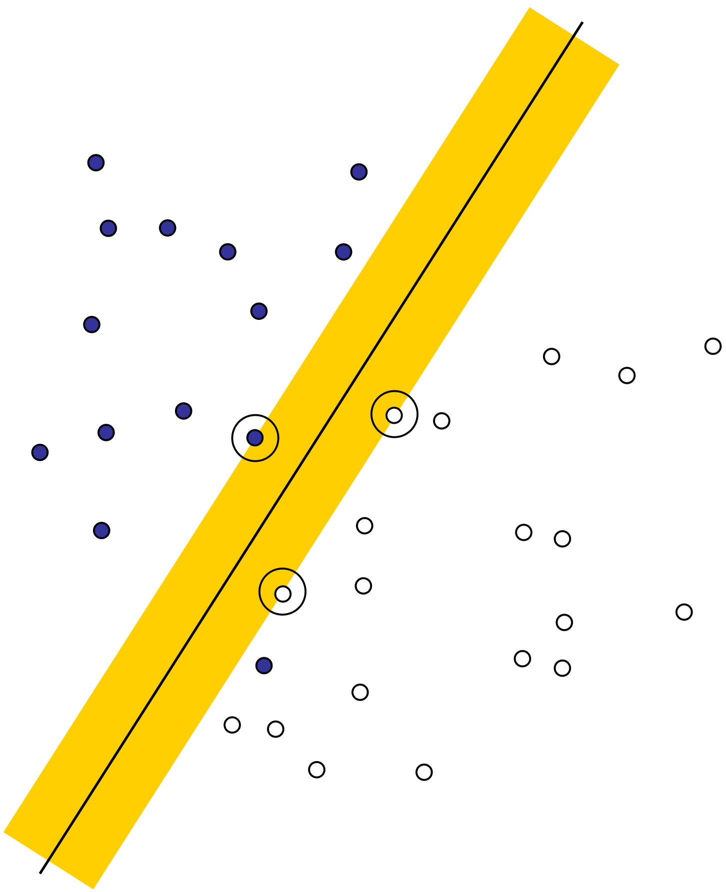
$$m = 1 = \boldsymbol{w}^T \boldsymbol{x}_i - t$$

The support vectors $\boldsymbol{x}_i$ fully determine the decision boundary!

Support vector machine (SVM)

What if the data is not linearly separable?

Or a data point "strays" into an otherwise nice margin?



This can be solved with a Soft Margin SVM

Soft margin SVM

ξ_i = "xi"

- We introduce a slack variable ξ_i for each training example to account for margin errors
  - Points that are inside the margin
  - Points that are on the wrong side of the decision boundary

$$\boldsymbol{w}^T \boldsymbol{x}_i - t \geq 1 - \xi_i \qquad \xi_i > 0 \rightarrow \boldsymbol{x}_i \text{ is not a support vector}$$

- Results in the soft margin optimization problem:

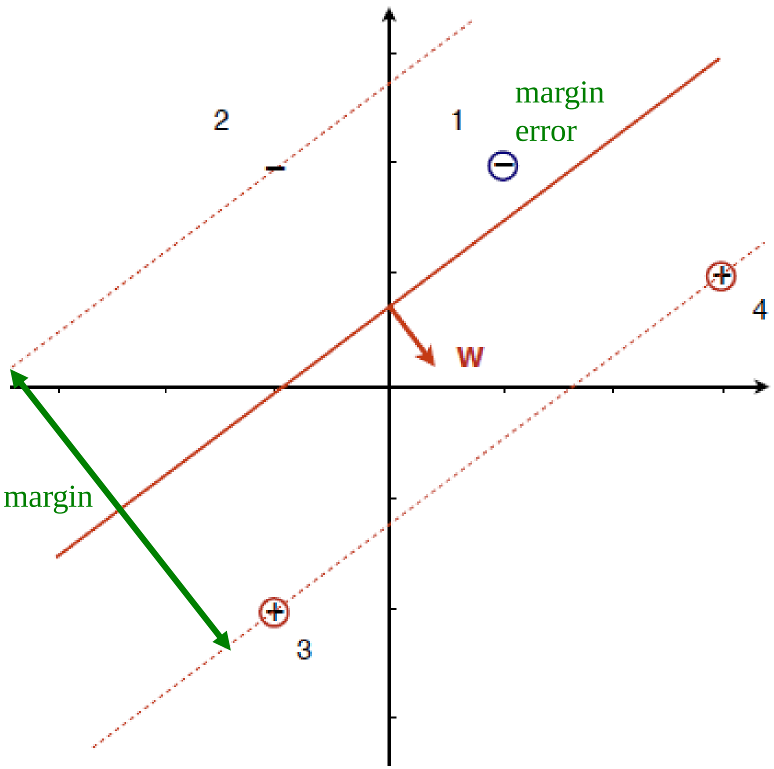
$$\mathbf{w}^*, t^*, \xi_i^* = \underset{\mathbf{w}, t, \xi_i}{\arg\min} \left[\frac{1}{2} ||\mathbf{w}||^2 + C \sum_{i=1}^{n} \xi_i \right]$$

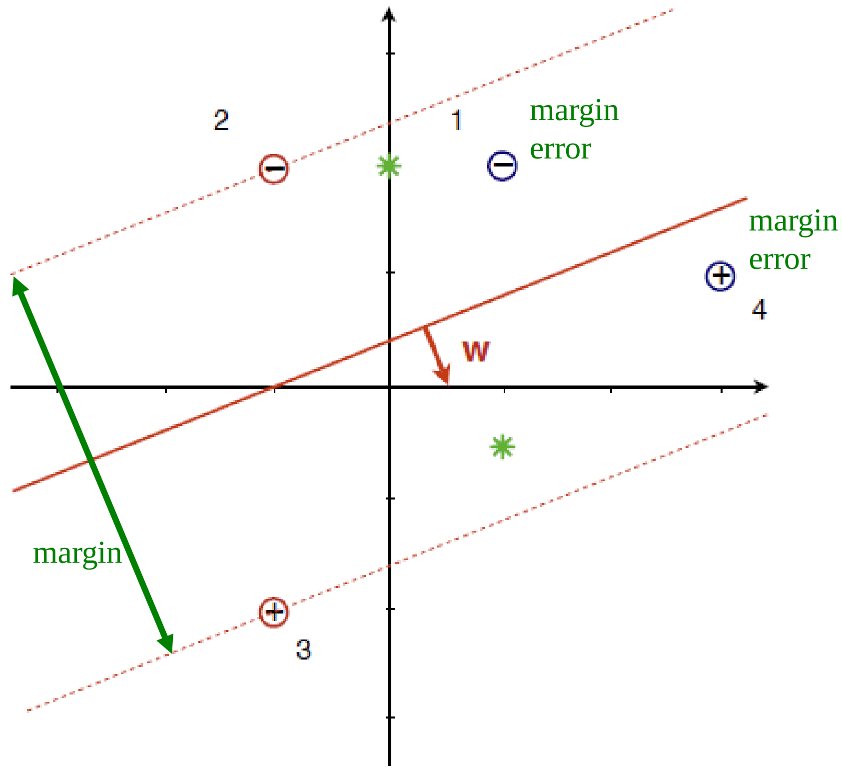
$$\text{subject to } y_i(\mathbf{w} \cdot \mathbf{x}_i - t) \geq 1 - \xi_i \text{ and } \xi_i \geq 0, 1 \leq i \leq n$$

The complexity parameter C is a user-defined parameter that allows for a tradeoff between maximizing the margin (lower C) and minimizing the margin errors (higher C)

- *Note that when C = 0, this gives no penalty to outliers – which makes it equivalent to our basic linear classifier!*

Soft margin SVM



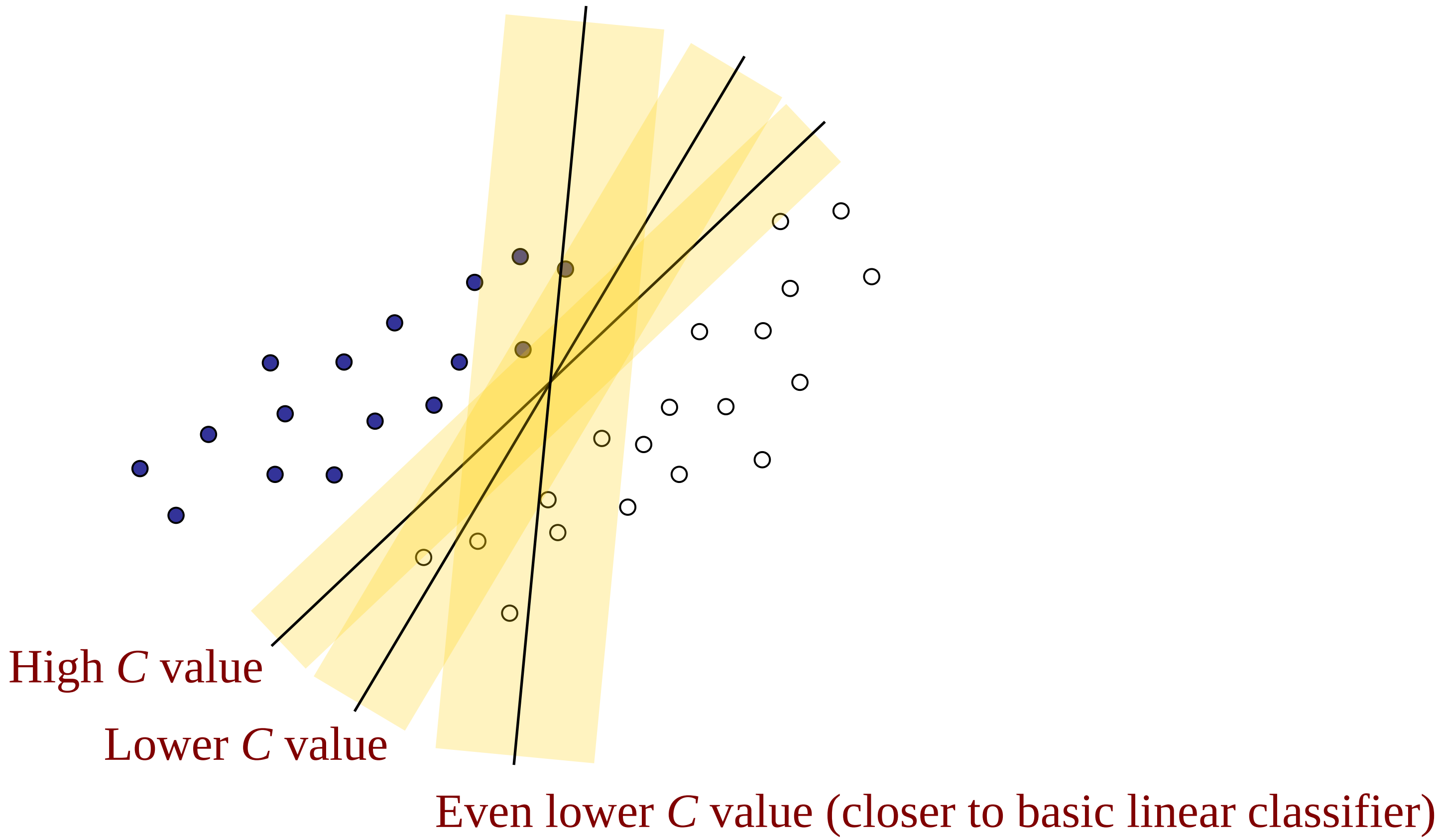


$C = \frac{5}{16}$ Smaller margin
Fewer margin errors

$C = \frac{1}{10}$ Larger margin
More margin errors

Soft margin SVM

A minimal-complexity (low *C*) soft margin classifier summarizes the classes by their class means in a way very similar to the basic linear classifier



Perceptron and SVM binary classifiers – summary

- In the perceptron model, we iteratively learn the linear discriminant $\boldsymbol{w}$, which is a linear combination of the misclassified input vectors $\boldsymbol{x}_i$

$$\boldsymbol{w} = \eta \sum_{i} \alpha_i y_i \boldsymbol{x}_i$$

α_i – # of times $\boldsymbol{x}_i$ was misclassified
y_i – class label of $\boldsymbol{x}_i$ {+1, –1}

  – After training, a new input is classified as a member of the positive class if $\boldsymbol{w}^T \boldsymbol{x} > 0$ (using homogeneous coordinate representation)

- In SVM learning, we solve a constrained optimization problem:

$$\alpha_1^*, \ldots, \alpha_n^* = \underset{\alpha_1, \ldots, \alpha_n}{\operatorname{argmax}} \left[-\frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j + \sum_{i=1}^{n} \alpha_i \right]$$

subject to $\alpha_i \geq 0, 1 \leq i \leq n$ and $\sum_{i=1}^{n} \alpha_i y_i = 0$

which leads us to $\boldsymbol{w} = \sum_{i=1}^{n} \alpha_i y_i \boldsymbol{x}_i$ where $\alpha_i = 0$ except for the support vectors

Perceptron and SVM binary classifiers – summary

- In both perceptron and SVM learning, the linear decision boundary is a linear combination of the training data points
  - In the perceptron, just the ones that get misclassified in the iterative training
  - In the SVM, just the (few) support vectors

- Both learning methods have a dual form in which the dot product of training data points $\boldsymbol{x}_i^T \boldsymbol{x}_j$ is part of the main computation
  - All values of $\boldsymbol{x}_i^T \boldsymbol{x}_j$ are contained in the Gram matrix

$$\boldsymbol{G} = \boldsymbol{X}^T \boldsymbol{X} = [\boldsymbol{x}_1 \quad \boldsymbol{x}_2 \quad \dots \quad \boldsymbol{x}_k]^T [\boldsymbol{x}_1 \quad \boldsymbol{x}_2 \quad \dots \quad \boldsymbol{x}_k]$$

  so it's often efficient to compute the Gram matrix in advance and index into it, rather than computing the dot products over and over again

Perceptron and SVM binary classifiers – summary

- Perceptron and (basic) SVM learning only converge to a solution if the training data is linearly separable
- If the data is not linearly separable, we can employ a soft margin SVM, where we introduce a *slack variable* ξ_i for each training data point, allowing for margin errors:

$$\boldsymbol{w}^T \boldsymbol{x}_i - t \geq 1 - \xi_i \qquad \xi_i > 0 \ \rightarrow \ \boldsymbol{x}_i \text{ is not a support vector}$$

and leading to this optimization problem:

$$\mathbf{w}^*, t^*, \xi_i^* = \underset{\mathbf{w}, t, \xi_i}{\arg\min} \left[\frac{1}{2} ||\mathbf{w}||^2 + C \sum_{i=1}^{n} \xi_i \right]$$

$$\text{subject to } y_i(\mathbf{w} \cdot \mathbf{x}_i - t) \geq 1 - \xi_i \text{ and } \xi_i \geq 0, 1 \leq i \leq n$$

where the complexity parameter C is a user-defined parameter that allows for a tradeoff between maximizing the margin (lower C) and minimizing the margin errors (higher C)