CSCE 421: Machine Learning

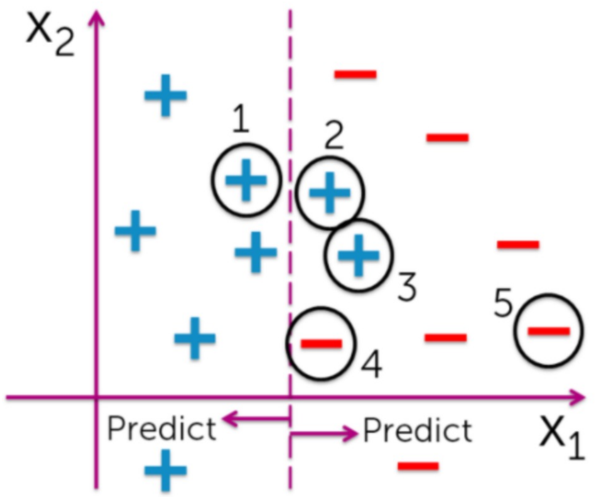
Lecture 21: Perceptron and Support Vector Machines

Texas A&M University

Bobak Mortazavi

Review

1. What is AdaBoost?
2. What is Gradient Boosting?
3. We train an AdaBoost classifier. A a certain iteration, the classifier produces a decision boundary like this:



Which of the five circled point will get higher weight the next iteration?

Goals for This Week

- Learn about the Perceptron
- Introduce even more loss functions
- Hinge loss, Softmax Cost
- Support Vector Machines

Linear Decision Boundaries

- A large number of our classification models are of the form:

$$F(x) = sign(\sum w * f)$$

Where, even if f is non-linear, this forms a linear decision boundary.

For example, in Logistic Regression, that boundary is:

$$X^T w = 0$$

What if a model tried to learn this boundary, outright, instead of a function that results in this boundary?

Perceptron

- A method to learn linear decision boundary (as the one found by logistic regression)
- Provides new insight into two-class classification
- Geometric interpretation helps explain regularization further
- Assume, again, the following notation:

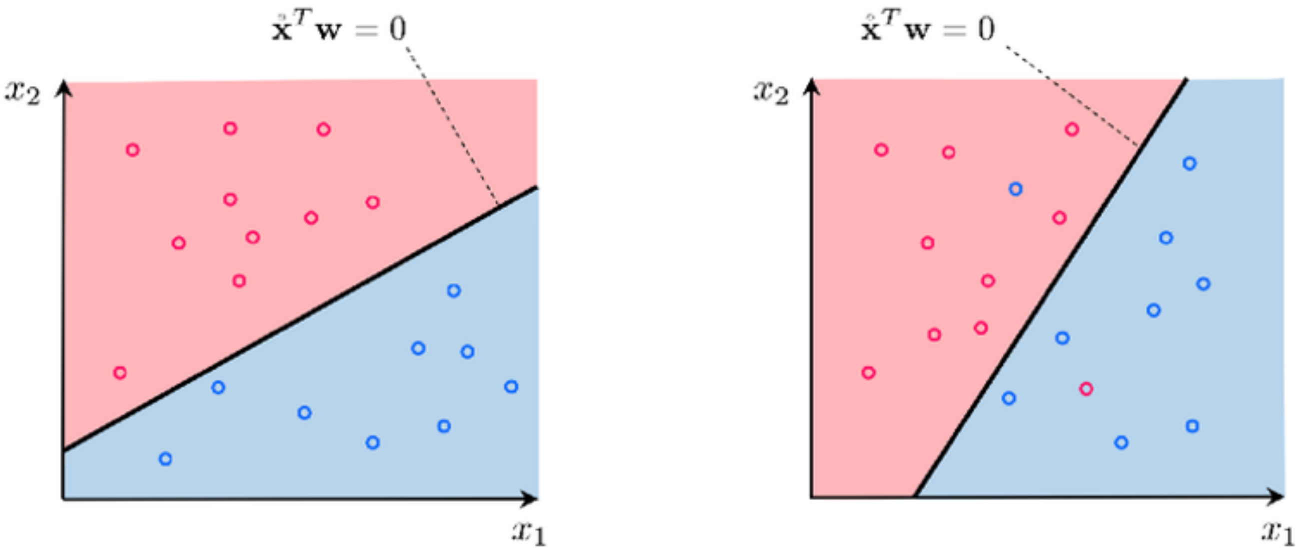
$$D = \{(x_i, y_i)\}_{i=1}^{N}$$

Where

$$y_i \in \{-1, +1\}$$

In the simplest of machine learning classification tasks, the two classes can be separated by a linear decision boundary

Perceptron: Cost Function



In this case, we find a linear decision boundary that divides the input space into two half-spaces

Perceptron: Cost Functions

- So, the desired set would be

$$y_i = +1 \; if \; x_i^T w > 0$$

$$y_i = -1 \; if \; x_i^T w < 0$$

- Which is equivalent to the following (think back to the 0-1 loss discussion in logistic regression):

$$-y_i x_i^T w < 0$$

- Which results in the cost function (known as the perceptron cost, the rectified linear unit, or the hinge loss):

$$g(w) = \frac{1}{N} \sum_{i=1}^{N} \max(0, -y_i x_i^T w)$$

Optimization of Hinge Loss

- This cost function is always convex
- In each input dimension, there is only one (discontinuous) derivative
- We can only use zero and first order optimization schemes, and must avoid the trivial solution of all w = 0
- ReLU limits our optimization capability:
  - No Newton's Method (why not?)
  - Poor choice of learning rate results in trivial solution!
  - So we should approximate this!

Softmax Approximation

- Introduce a new function

$$soft(s_0, s_1, \ldots, s_{C-1}) = \log(e^{s_0} + e^{s_1} + \cdots + e^{s_{C-1}})$$

- Which approximates

$$soft(s_0, s_1, \ldots, s_{C-1}) \approx max(s_0, s_1, \ldots, s_{C-1})$$

- Used in our cost function

Softmax Approximation

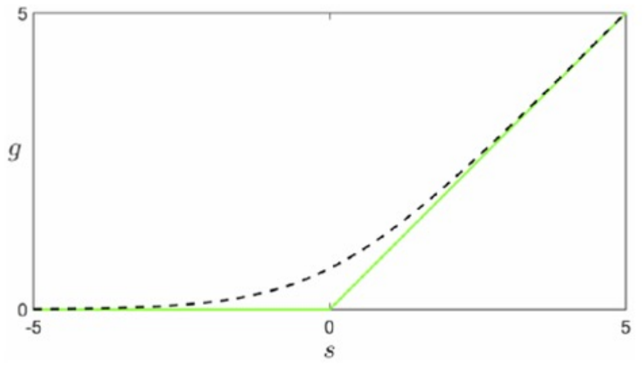
- Introduce a new function

$$soft(s_0, s_1, \ldots, s_{C-1}) = \log(e^{s_0} + e^{s_1} + \cdots + e^{s_{C-1}})$$

- Which approximates

$$soft(s_0, s_1, \ldots, s_{C-1}) \approx max(s_0, s_1, \ldots, s_{C-1})$$

- Used in our cost function



Softmax Approximation

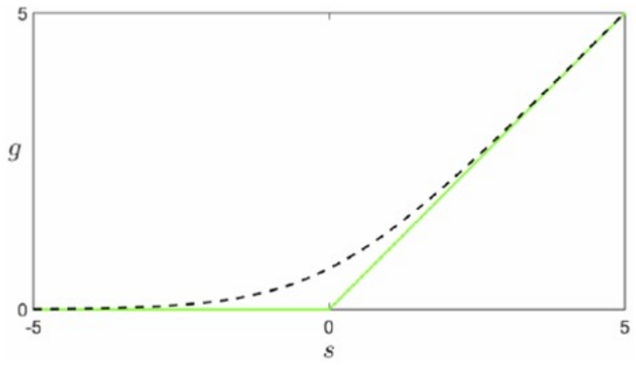
- Introduce a new function

$$soft(s_0, s_1, \dots, s_{C-1}) = \log(e^{s_0} + e^{s_1} + \cdots + e^{s_{C-1}})$$

- Which approximates

$$soft(s_0, s_1, \dots, s_{C-1}) \approx max(s_0, s_1, \dots, s_{C-1})$$

- Used in our cost function
- Twice differentiable!
- Broader set of optimization tools
- Closely approximates ReLU



The Softmax Cost

$$g(w) = \frac{1}{N}\sum_{i=1}^{N} max(0, -y_i x_i^T w) \approx \frac{1}{N}\sum_{i=1}^{N} \log\left(1 + e^{-y_i x_i^T w}\right)$$

- Differentiable
- No trivial solution
- So we can use this, right? It should work well?

The Softmax Cost

$$g(w) = \frac{1}{N}\sum_{i=1}^{N} max(0, -y_i x_i^T w) \approx \frac{1}{N}\sum_{i=1}^{N} \log\left(1 + e^{-y_i x_i^T w}\right)$$

- If there is a linearly separable dataset (rare)
- Our first guess of w will result in perfect separation
- ReLU will stop!
- But Softmax Cost will not be 0, so it will keep iterating (causing instability in solution)

The Softmax Cost

$$g(w) = \frac{1}{N}\sum_{i=1}^{N} max(0, -y_i x_i^T w) \approx \frac{1}{N}\sum_{i=1}^{N} \log\left(1 + e^{-y_i x_i^T w}\right)$$

- If there is a linearly separable dataset (rare)
- Our first guess of w will result in perfect separation
- ReLU will stop!
- But Softmax Cost will not be 0, so it will keep iterating (causing instability in solution)
  - Take fewer steps?
  - Halt if weights become too big?
  - Regularization!

Modify notation a bit

Take

$$w = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_D \end{bmatrix}$$

And separate into bias b = w_0 and the normal vector, which has feature-touching weights

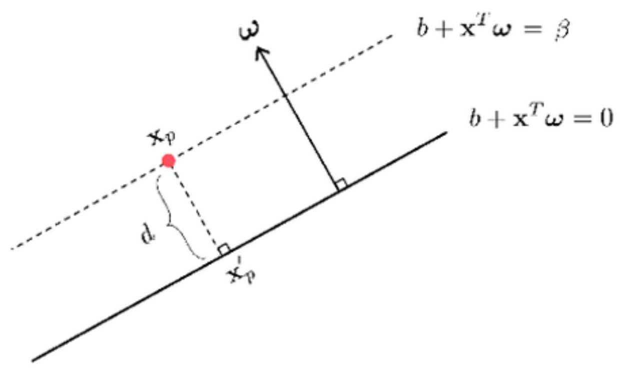
$$w = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_D \end{bmatrix}$$

So we express the decision boundary as

$$b + \ x^T w = 0$$

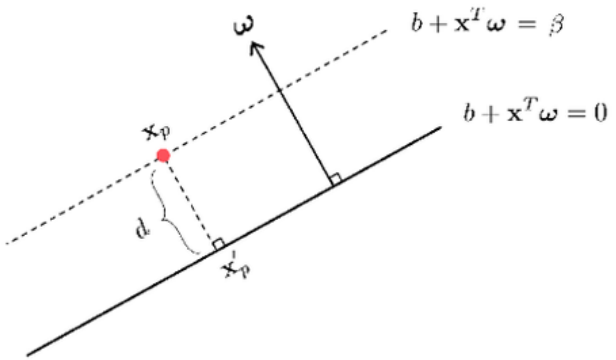
Normal Vector of Decision Boundary

- The feature-touching weights w define the normal vector of the decision boundary
- The error (signed distance) of a point x to a linear decision boundary in terms of the normal vector can then be computed



Normal Vector of Decision Boundary

- The feature-touching weights w define the normal vector of the decision boundary
- The error (signed distance) of a point x to a linear decision boundary in terms of the normal vector can then be computed



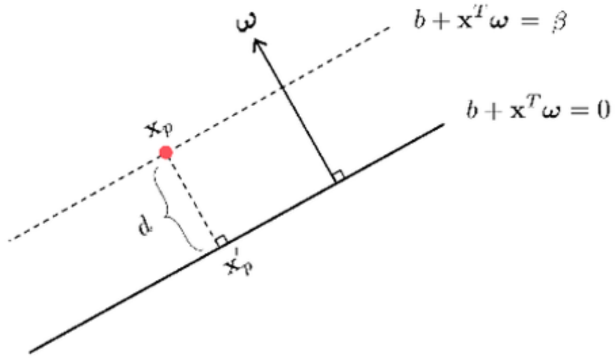
- To compute the signed distance between x_i' and x_i:

$$d = \|x_i' - x_i\|_2 * sign(\beta)$$

- Since it is parallel to the normal vector w, we can compute

$$(x_i' - x_i)^T w = \|x_i' - x_i\|_2 \|w\|_2 = d * \|w\|_2$$

Normal Vector of Decision Boundary



So the difference between the decision boundary and its translated version would be:

$$\beta - 0 = \left(b + \; {x'}_i^T w\right) - \left(b + \; x_i^T w\right) = (x_i' - x_i)^{\,T} w$$

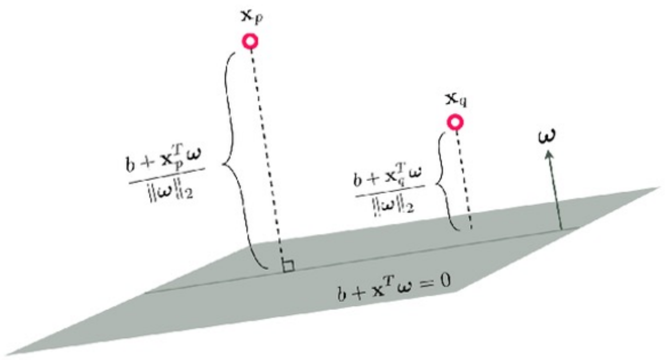
Which results in

$$d \, * \, \|w\|_2 = \, \beta$$

Which then yields

$$d = \frac{\beta}{\|w\|_2} = \frac{\left(b + \; x_i^T w\right)}{\|w\|_2}$$

Normal Vector of Decision Boundary



So any linear decision boundary

$$\frac{\left(b + \ x_i^T w\right)}{\|w\|_2} = \frac{(b)}{\|w\|_2} + \frac{\left(x_i^T w\right)}{\|w\|_2} = 0$$

Can be scaled by C = $1/\|w\|_2$, and if we force $\|w\|_2 = 1$ as a regularization scheme we can

- Create a decision boundary that perfectly separates classes with normalized feature weights
- Prevents diverging to infinity
- And the normalization happens while optimizing loss

Perception: Two class classification

- So our optimization problem becomes

$$\min_{\beta,w} \frac{1}{N} \sum_{i=1}^{N} \log\left(1 + e^{-y_i(b + x_i^T w)}\right)$$

subject to

$$\|w\|_2^2 = 1$$

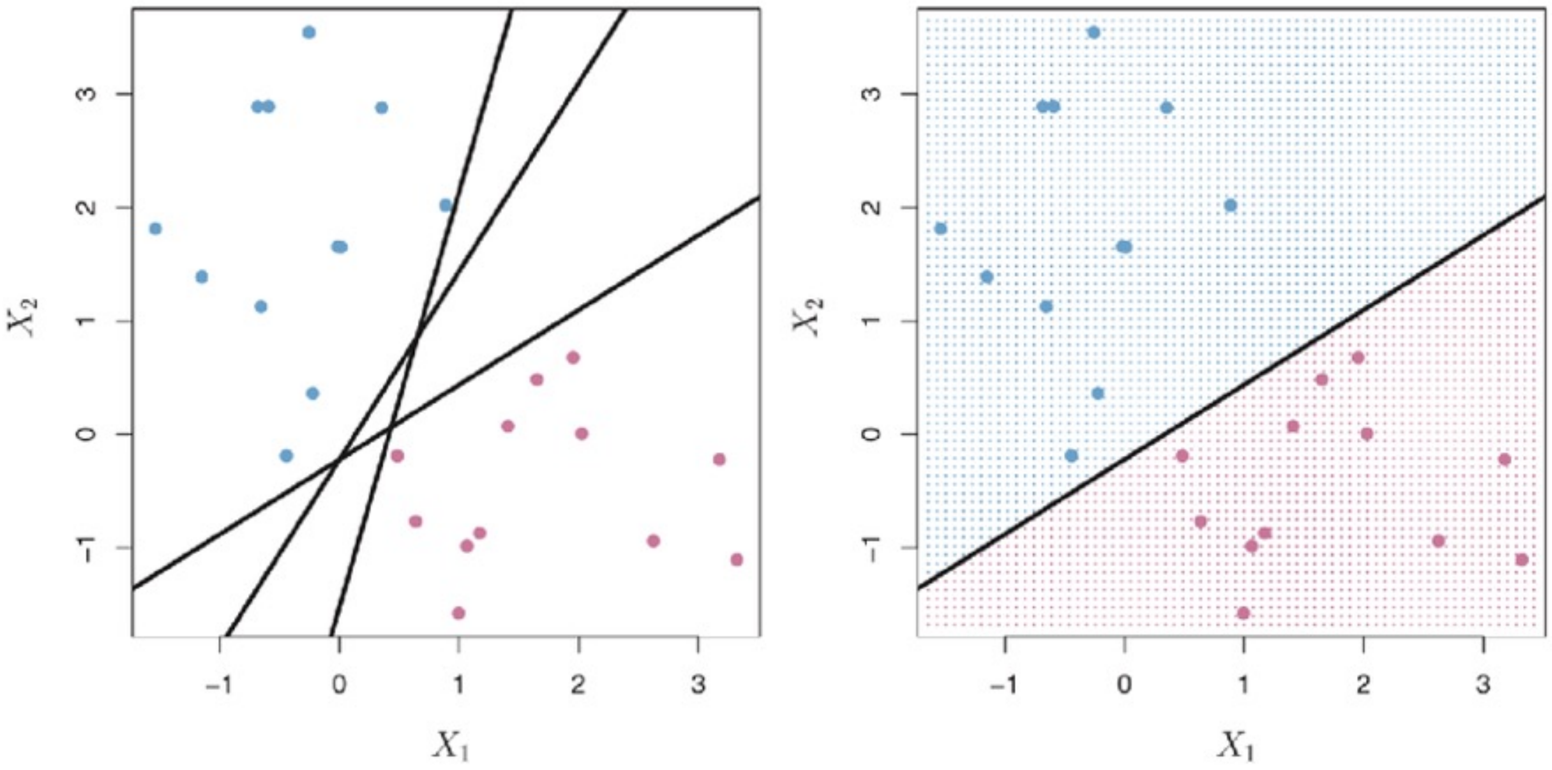
- Which learns a linear decision boundary to perfectly separate two classes of data
- That avoids divergence through feature-weight normalization on the unconstrained version:

$$\min_{b,w} g(b,w) = \frac{1}{N} \sum_{i=1}^{N} log\left(1 + e^{-y_i(b + x_i^T w)}\right) + \lambda\|w\|_2^2$$

What exactly is that decision boundary?

- In D dimensional space, what do we call a flat subspace of D-1?

Hyperplanes in Separable Data



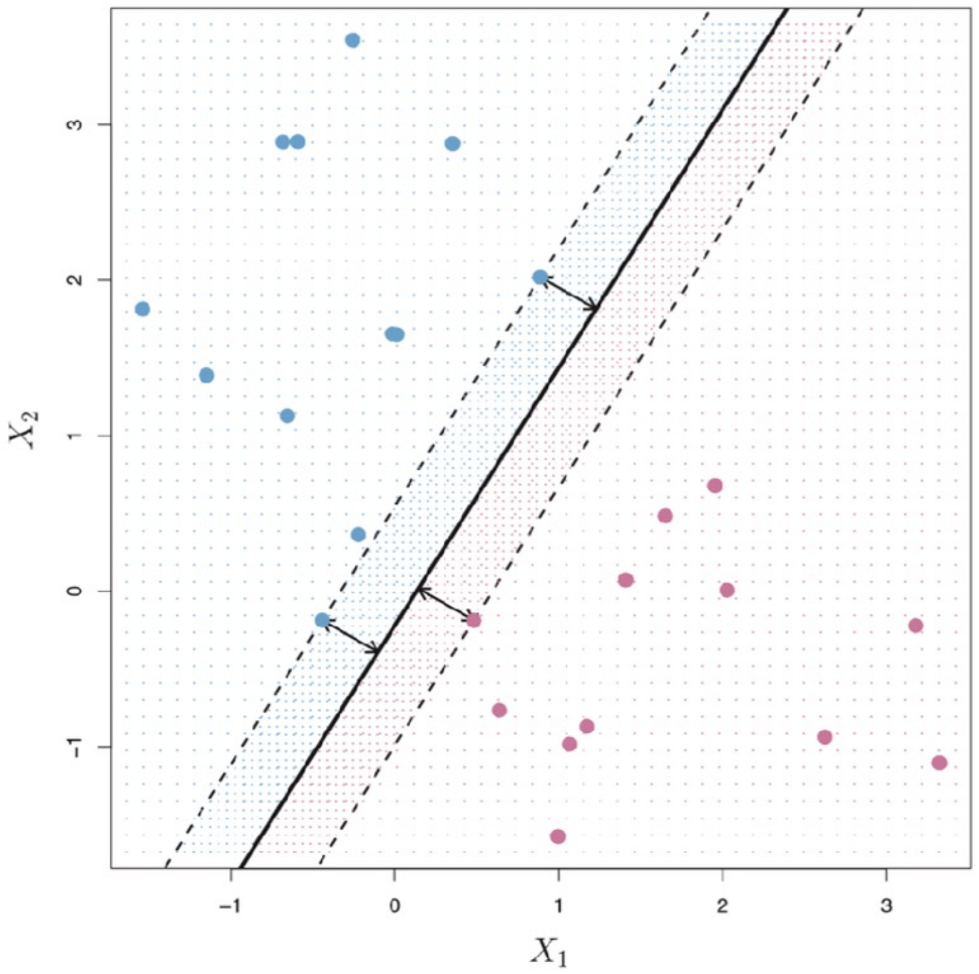
Picking the key hyperplane

- If a separating hyperplane exists, classification is easy!
- f(x) > 0 implies y = 1
- f(x) < 0 implies y = -1
- So our classifier is

$$f = H = \{x \mapsto sign(w * x + b): w \in \mathbb{R}^D, b \in \mathbb{R}\}$$

- Where we can see the magnitude of f to determine just how far we are from the hyperplane. Farther = more confident
- So, pick the hyperplane that is then farthest from all the training set points

Optimal Hyperplane



Maximal Marginal Classifier

- Find the maximal marginal hyperplane – which depends directly on the points that lie on the margin
- These points are called “support vectors”

$$x_1, \dots, x_N \in \mathbb{R}^D$$
$$y_1, \dots, y_N \in \{-1, +1\}$$

Then

$$\max_{w_0, w_1, \dots, w_D, M} M$$

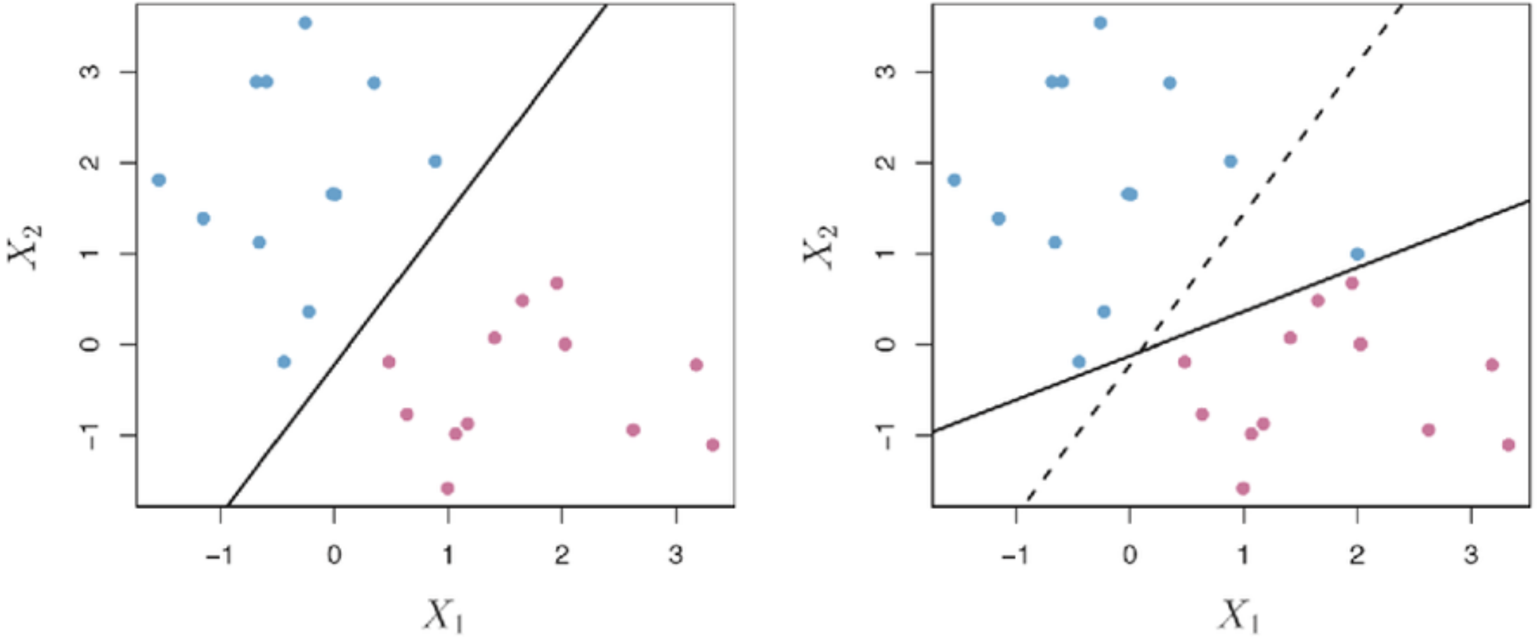
Subject to

$$\sum_{q=1}^{D} w_q^2 = 1$$

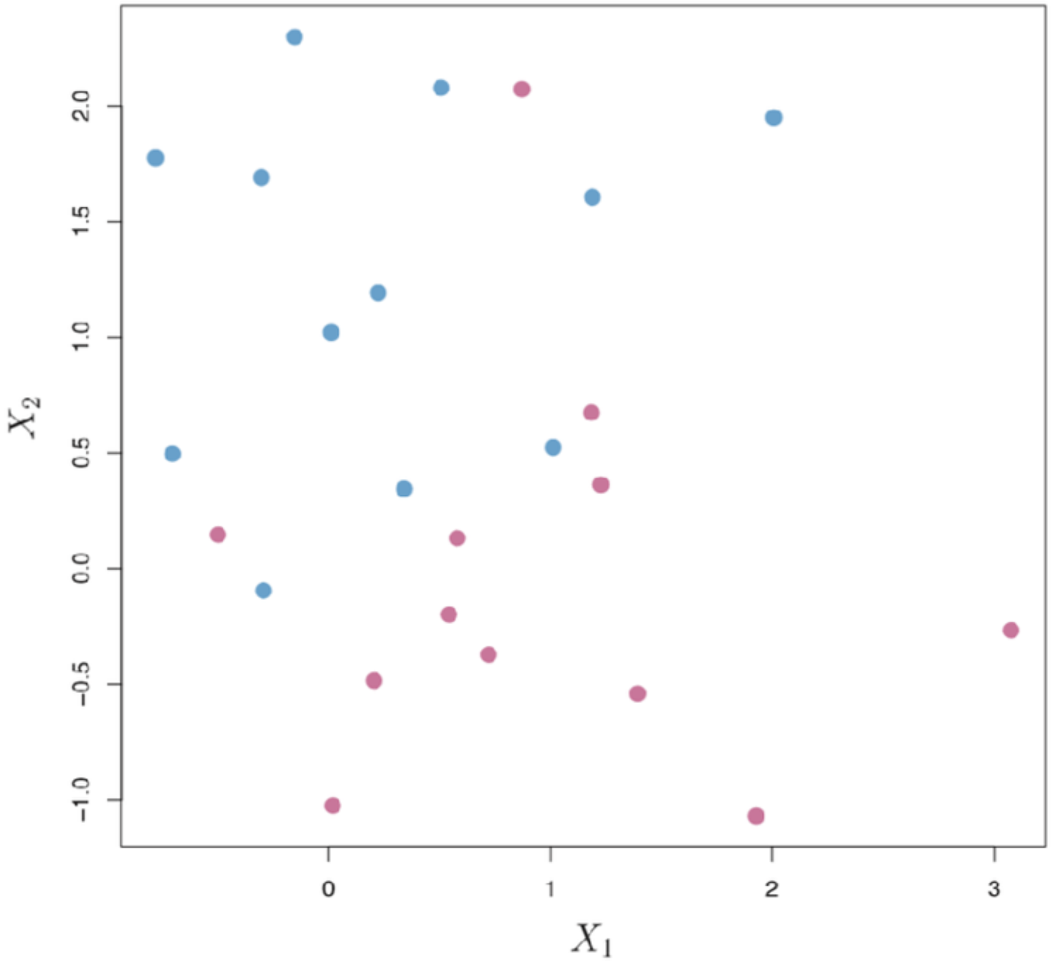
And

$$y_i(w_o + w_1 x_{i1} + \dots + w_D x_{iD}) \geq M \; \forall \; i = 1, 2, \dots, N$$

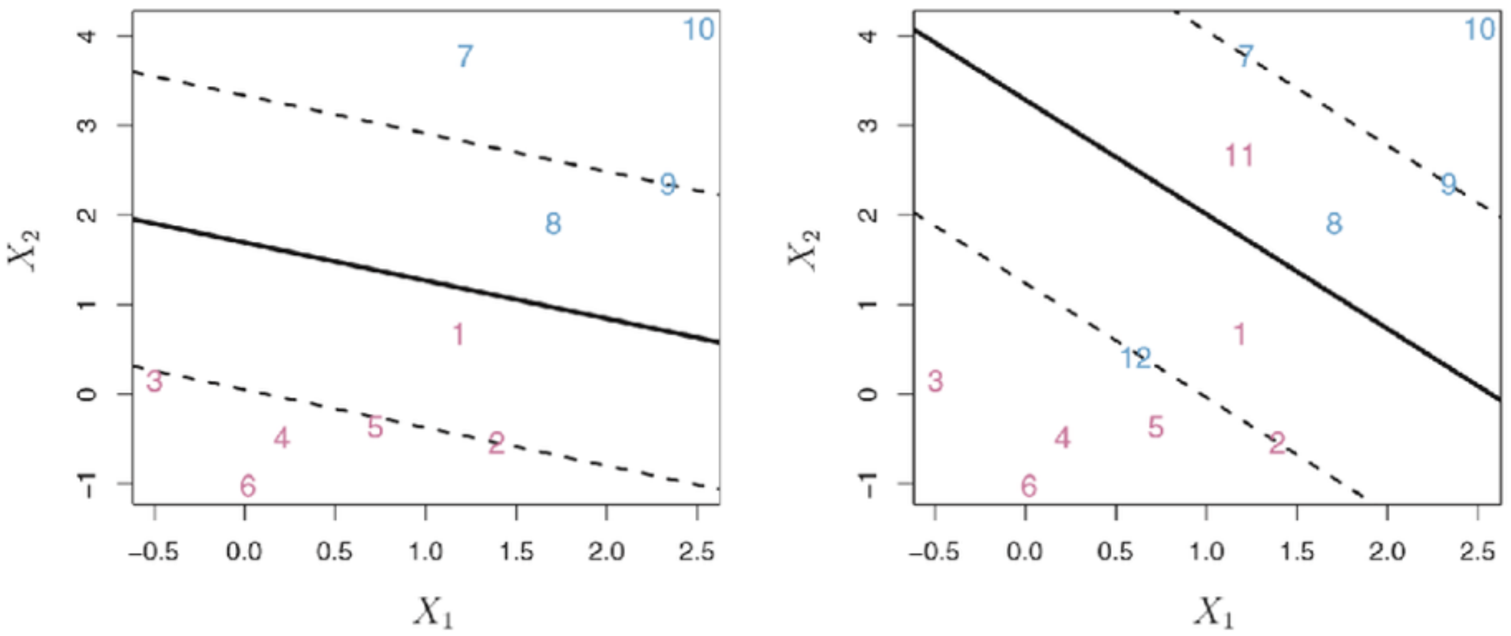
Examples



What happens when we can’t separate?



What if we allow for some “slack”?



The Support Vector Classifier

$$x_1, \dots, x_N \in \mathbb{R}^D$$
$$y_1, \dots, y_N \in \{-1, +1\}$$

Then

$$\max_{w_{0}, w_{1}, \dots, w_{N}, M} M$$

Subject to

$$\sum_{q=1}^{D} w_q^2 = 1$$

And

$$y_i(w_o + w_1 x_{i1} + \dots + w_D x_{iD}) \geq M(1 - \varepsilon_i) \; \forall \, i = 1, 2, \dots, N$$

The Support Vector Classifier

$$y_i(w_o + w_1 x_1 + \ \dots + w_D x_D) \geq M(1 - \varepsilon_i)\ \forall\ i = 1, 2, \dots, N$$

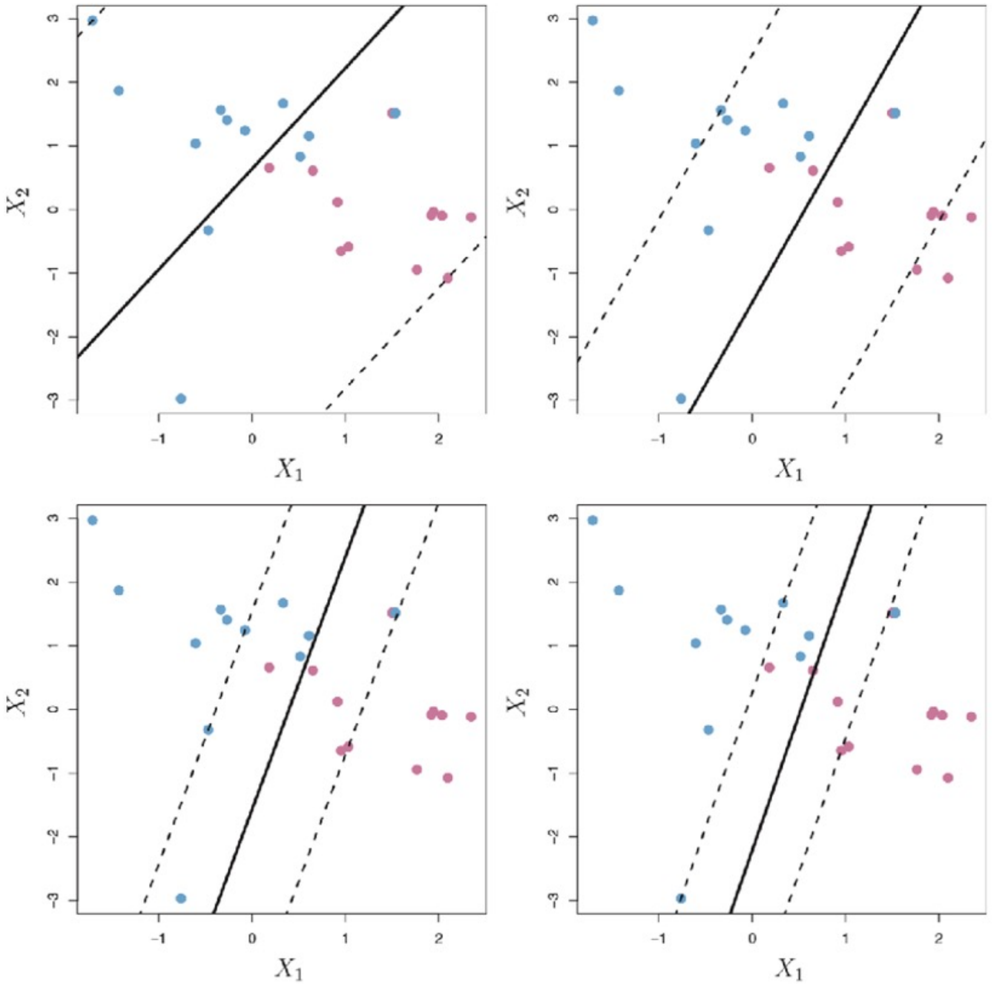
Where

$$\varepsilon_i \geq 0$$

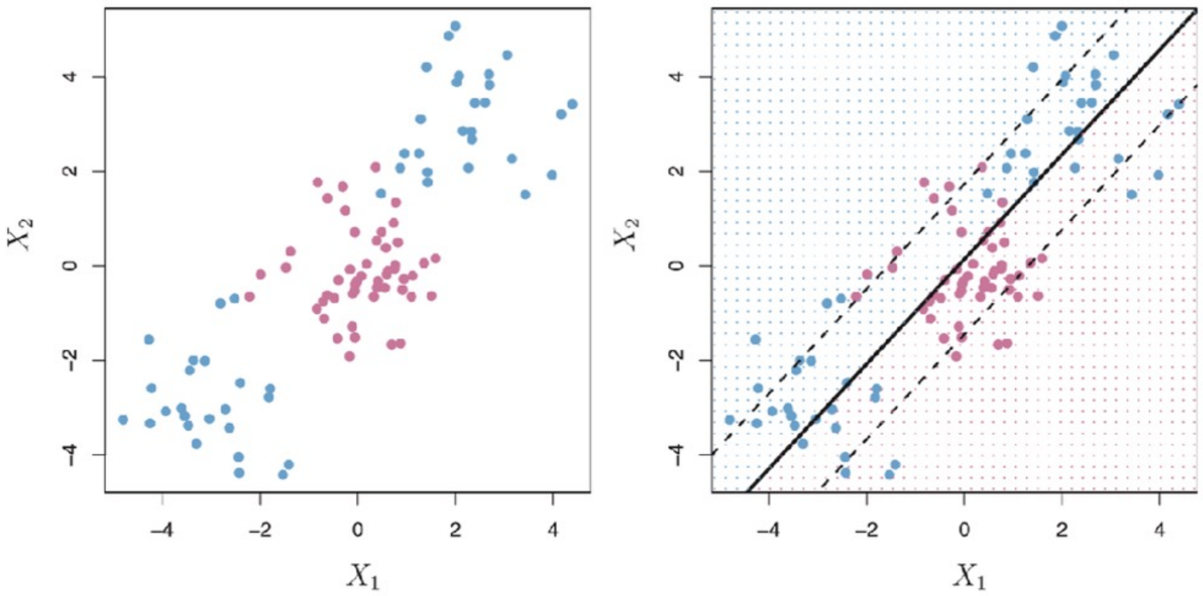
$$\sum_{i=1}^{N} \varepsilon_i \leq C$$

- Where C is a non-negative tuning parameter
- M is the width of the margin
- ε_i are the slack variables. When $\varepsilon_i > 1$ the object is on the wrong side of the hyperplane, when $\varepsilon_i > 0$ the object violates the margin
- Therefore – C determines the number and severity of margin violations allowed
- C picked through cross-validation. What happens with small C? large C?

Tuning C



But... what if...?



The Support Vector Machine

- The support vector classifier is natural for 2-class decisions
- What if we add more interaction terms?

$$x_1, \dots, x_D, x_1^2, \dots, x_D^2 \in \mathbb{R}^N$$
$$y_1, \dots, y_D \ \in \{-1, +1\}$$

Then

$$\max_{w_{0,} w_{1,} \dots, w_{N,} M} M$$

Subject to

$$y_i \left(w_o + \sum_{q=1}^{D} w_{j1} x_{ij} + \sum_{q=1}^{D} w_{j2} x_{ij}^2 \right) \geq M(1 - \varepsilon_i) \ \forall \ i = 1, 2, \dots, N$$

And

$$\sum_{q=1}^{D} \sum_{k=1}^{2} w_{qk}^2 = 1$$

SVMs: Enlarge parameters through kernel space

- SVC is solved through the kernel space
- If we define an inner product as $\langle x_i, x_{i'} \rangle = \sum_{q=1}^{D} x_{qi} x_{qi'}$
- Then the linear SVC finds

$$f(x) = w_0 + \sum_{i=1}^{N} \alpha_i \langle x, x_i \rangle$$

- Where there are P α_p per training example
- For this we need $\binom{N}{2}$ inner products (or $\mathrm{N}(N-1)/2$)
- But it turns out that $\alpha_i \neq 0$ only for the support vectors, so we can rewrite f as

$$f(x) = w_0 + \sum_{i \in S} \alpha_i \langle x, x_i \rangle$$

Kernels in Higher Dimension

- To project data into higher dimensions and still use SVM, we don't need to know how to formulate data in this high dimension – just need to know how to calculate the inner product $K(x_p, x_{p'})$
- This kernel K quantifies the similarity between two observations
- When

$$K(x_i, x_{i'}) = \sum_{q=1}^{D} x_{iq} x_{i'q}$$

We get the linear SVC – so this is a linear kernel

- Similarly

$$K(x_i, x_{i'}) = \left(1 + \sum_{q=1}^{D} x_{iq} x_{i'q}\right)^d$$

Yields the polynomial kernel of degree d, which is a much more flexible decision boundary (but needs more data to train, why?)

Radial Basis Function (RBF) Kernel

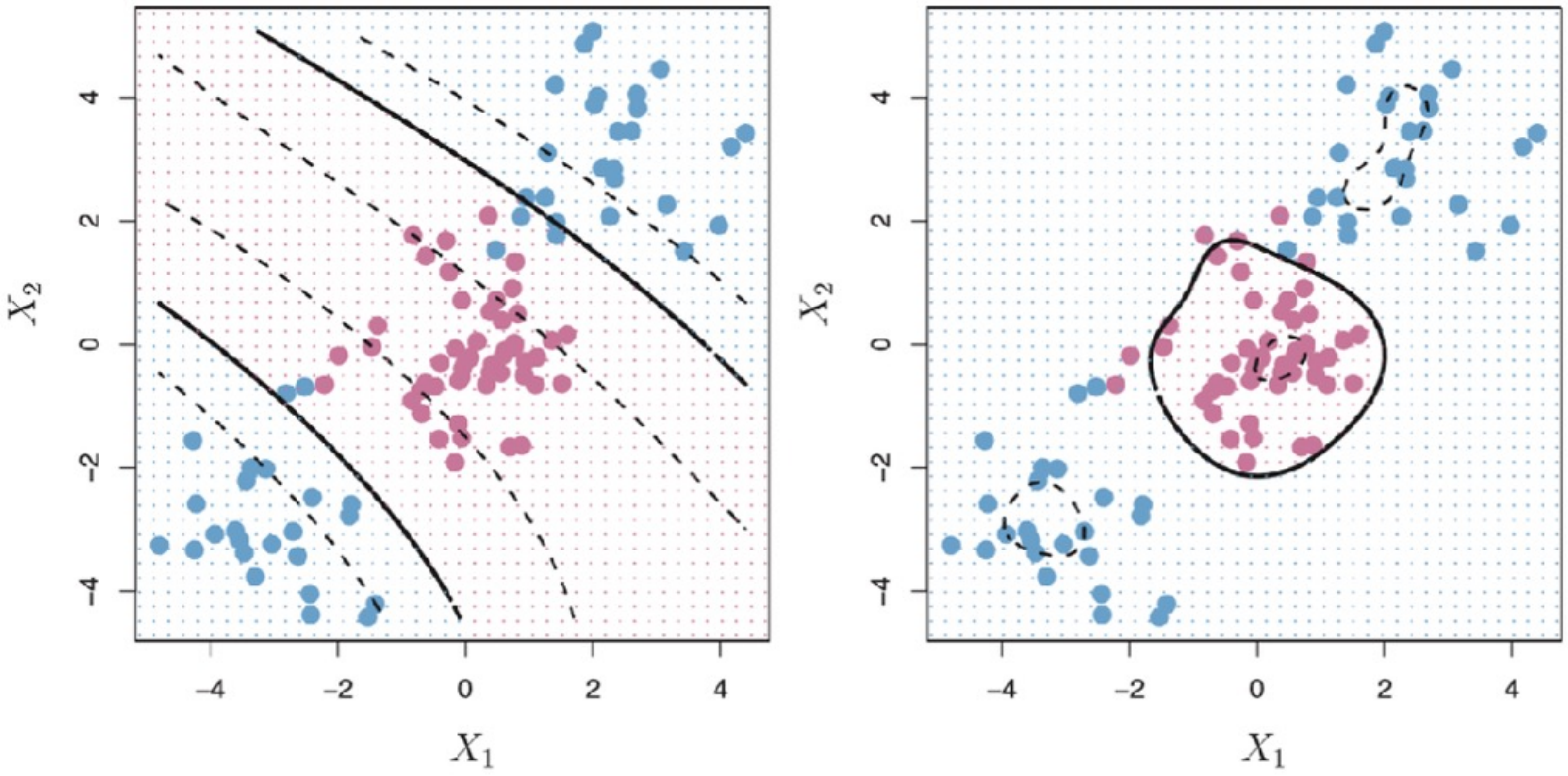
$$K(x_i, x_{i'}) = exp\left(-\gamma \sum_{q=1}^{D} \left(x_{iq} - x_{i'q}\right)^2\right)$$

- Very popular kernel (RBF or Gaussian kernel)
- Very local behavior, because of the sum of squared differences
- Rather than simply adding additional features, using these kernels is better computationally
- Feature space is implicit and infinite dimensional, so we could never compute the full model projected into these spaces

- In general, then , we say SVM defines a function

$$f(x) = w_0 + \sum_{i \in S} \alpha_i K(x, x_i)$$

Decision boundary with RBF



Takeaways and Next Time

- Understanding Perceptron
- Understanding Hinge Loss
- Motivating the Maximal Margin Hyperplane
- Introducing the Support Vector Classifier
- Understanding SVM and Kernels (and their hyperparameters)
- Next time: The underlying Math that makes this all work!