The Perceptron Algorithm

CSE 446: Machine Learning
Slides by Emily Fox (mostly)
Presented by Anna Karlin

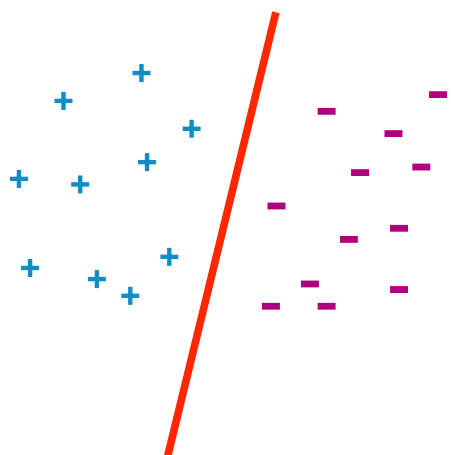
April 29, 2019

Can we do binary classification without a model?

Before we used the logistic model:

$P(Y = y \mid x, w) = 1/ (1+\exp(-ywTx))$

i.e. separate the data into two data groups.
Find a hyperplane that divides the two.



The perceptron algorithm

The perceptron algorithm [Rosenblatt '58, '62]

For simplicity in this lecture we will assume that b=0

- Classification setting: y in {-1,+1}

Each data point is in one of two groups

- Linear model

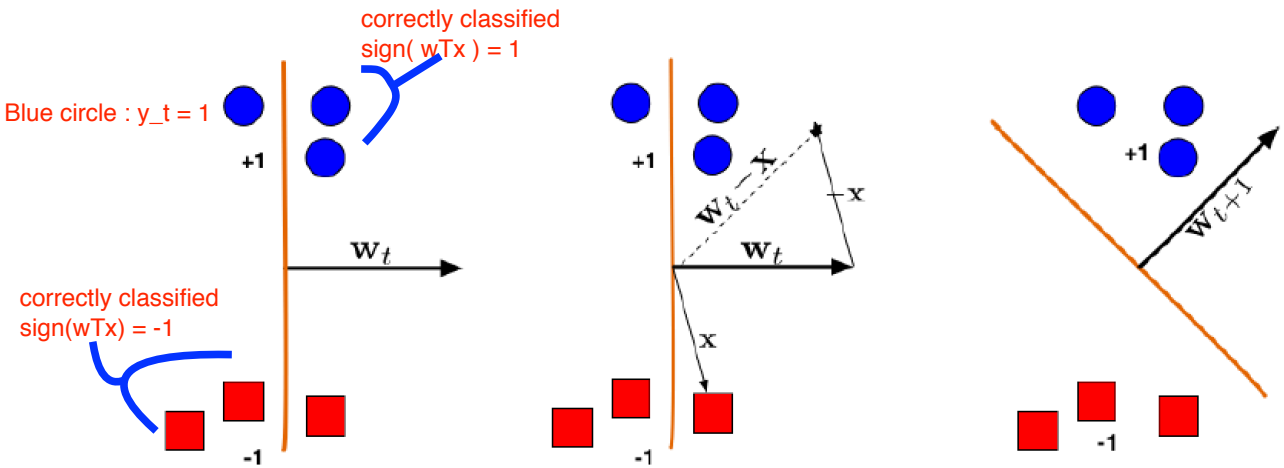
using basic thresholding —> Find a w and b based on training set. if wTx + b > 0 choose y = +1, otherwise choose -1
y_guess = sign(wTx + b)

  - Prediction:

- Training:
  - Initialize weight vector:
  - At each time step:
    - Observe features: find the x_t of a datapoint in training set (features)
    - Make prediction: y_guess = sign(wTx + b)
    - Observe true class: y_t in {-1, 1}

    - Update model: ONLINE LEARNING (but could be used in batch setting just one data point at a time)
      - If prediction is not equal to truth w_{t+1} = w_t + y_t * x_t

Picture due to Killian Weinberger



- Initialize $\mathbf{w}_0$.
- for $t := 1$ to T
  - Observe feature vector $\mathbf{x}_t$.
  - Make a prediction: $\hat{y} = \text{sign}(\mathbf{x}_t^T \mathbf{w}_t)$
  - Observe the true label $y_t \in \{-1, 1\}$.
  - Update the model:
    - If $\hat{y} = y_t$, then $\mathbf{w}_{t+1} := \mathbf{w}_t$.
    - Otherwise ($\hat{y} \neq y_t$), then $\mathbf{w}_{t+1} := \mathbf{w}_t + y_t \mathbf{x}_t$.

if we are incorrect:
w_t + x_t if y_t = 1
w_t - x_t if y_t = -1

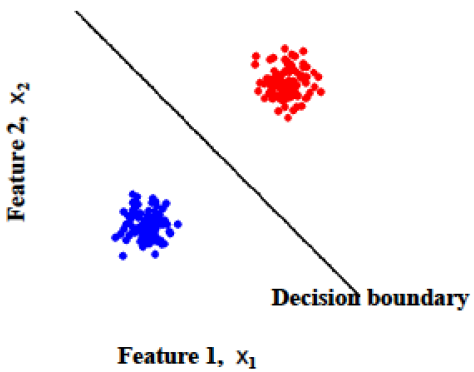
Why move like this? If w_t had BEEN x_t when y_t = 1, then we would have classified correctly.
If w_t had BEEN -x_t when y_t = -1, then we would have classified correctly.

How many mistakes can a perceptron make?

Assumption: data is linearly separable

linearly separable: hyperplane such that the data is perfectly separated in two.

Linearly separable data



Linearly non-separable data

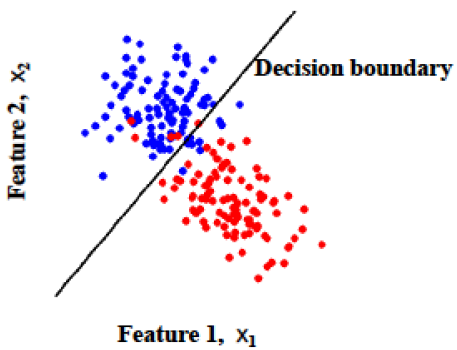


Figure by Alexander Ihler

Linear separability using margin

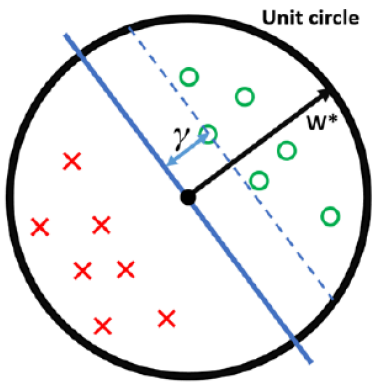


Figure by Kilian Weinberger

Data linearly separable, if there exists

- a vector — w* that defines a hyperplane (orthogonal to it) which perfectly separates the data into two groups
- a margin — Buffer around the hyperplane of width 2*gamma that separates the plain from the data.
  i..e gamma = min distance from any point in training set to hyperplane

such that

distance from x to hyperplane w*v = 0 is >= gamma
THUS
if y_t = 1 then w* dot x_t >= gamma
if y_t = -1 then w* dot x_t <= -gamma

l w* dot x_t l >= gamma

Perceptron analysis: Linearly separable case

Theorem [Block, Novikoff]:

- Given a sequence of labeled examples:

  (x_1, y_1) , (x_2, y_2), …, (x_n, y_n)

- Each feature vector has bounded norm:

  ll x ll <= R

- If dataset is linearly separable:

  and there exists a gamma > 0 such that there exists a w* with ll w* ll = 1
  such that l x_i dot w*) >= gamma for all i

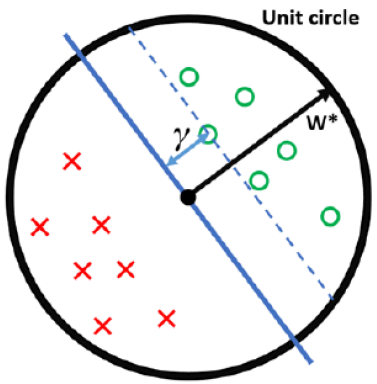


Figure by Kilian Weinberger

Then the # mistakes made by the online perceptron on this sequence is bounded by

mistakes <= R^2 / gamma^2

Perceptron proof for linearly separable case

w* is the perfect vector; defines a hyperplane that perfectly separates data into two classes.

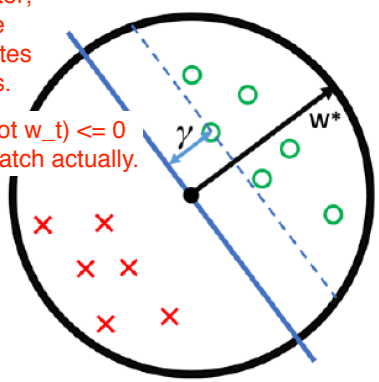


Figure by Kilian Weinberger

- Every time we make a mistake, we get w closer to w^*:
  - Mistake at time t:
  - Taking dot product with w^*:
  - Thus after m mistakes:

$$\mathbf{w}_{t+1} := \mathbf{w}_t + y_t \mathbf{x}_t$$

mistake if y_t * (x_t dot w_t) <= 0
predicted does not match actually.

w* w_{t+1}
= w* w_t + y_t w*x_t
>= w*w_t + gamma

w* w_{t+1} >= gamma * m

- On the other hand, norm of w_{t+1} doesn't grow too fast:

$$||\mathbf{w}_{t+1}||^2 = ||\mathbf{w}_t||^2 + 2y_t(\mathbf{w}_t \cdot \mathbf{x}_t) + ||\mathbf{x}_t||^2$$

<= R^2 by assumption that each feature has bounded norm

<= 0 as we misclassfied

  - Thus, after m mistakes:

|| w_{t+1} || <= m * R^2

- Putting all together:

gamma * m <= w* w_{t+1} <= || w* || || w_{t+1} || <= sqrt(m) R
gamma * m <= sqrt(m) R
m <= R^2 / gamma^2

Beyond linearly separable case

- Perceptron algorithm is super cool!
  - No assumption about data distribution!
    - Could be generated by an adversary, no need to be iid
  - Makes a fixed number of mistakes, and it's done forever!
    - Even if you see infinite data

Rosenblatt 1957

"the embryo of an electronic computer that [the Navy] expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence."

The New York Times, 1958

NEW NAVY DEVICE LEARNS BY DOING

Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser

WASHINGTON, July 7 (UPI) —The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.

The embryo—the Weather Bureau's $2,000,000 "704" computer—learned to differentiate between right and left after fifty attempts in the Navy's demonstration for newsmen.

The service said it would use this principle to build the first of its Perceptron thinking machines that will be able to read and write. It is expected to be finished in about a year at a cost of $100,000.

Dr. Frank Rosenblatt, de-

Beyond linearly separable case

- Perceptron algorithm is super cool!
  - No assumption about data distribution!
    - Could be generated by an adversary, no need to be iid
  - Makes a fixed number of mistakes, and it's done for ever!
    - Even if you see infinite data

- Possible to motivate the perceptron algorithm as an implementation of SGD for minimizing a certain loss function. (see next homework)

Beyond linearly separable case

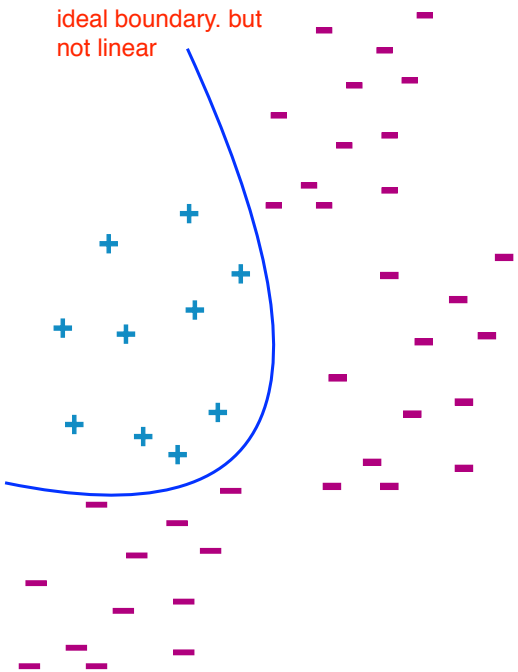
- Perceptron algorithm is super cool!
  - No assumption about data distribution!
    - Could be generated by an adversary, no need to be iid
  - Makes a fixed number of mistakes, and it's done for ever!
    - Even if you see infinite data

- However, real world is not linearly separable
  - Can't expect never to make mistakes again
  - If data not separable, cycles forever and hard to detect.
  - Even if separable, may not give good generalization accuracy (make have small margin).

i.e. small gamma… This means
if I get future data theres a higher
likelihood one of the points slips over the line

The kernelized perceptron

What to do if data are not linearly separable?



Use feature maps…

$$\phi(x) : \mathbb{R}^d \rightarrow F$$

- Start with set of inputs for each observation **x** = (x[1],x[2],…, x[d])

and training set: $\{(\mathbf{x}_i, y_i)\}_{i=1..n}$

- Define feature map that transforms each input vector $\mathbf{x}_i$ to higher dimensional feature vector $\phi(\mathbf{x}_i)$

Example: $(x_i[1], x_i[2], x_i[3])^T$

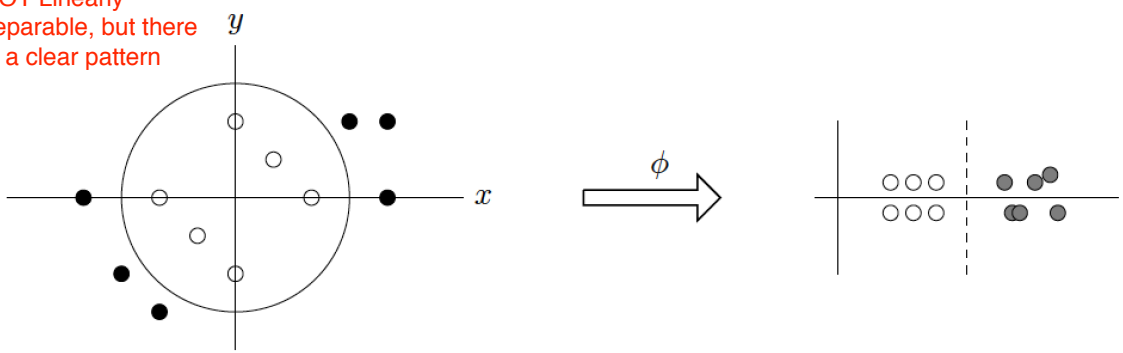
3 dimensional data. Suppose not linearly separable. Map to a higher dimensional space where it will be linear separable.

$\phi(\mathbf{x}_i) = (1,\ x_i[1],\ x_i[1]^2,\ x_i[1]x_i[2],\ x_i[2],\ x_i[2]^2,\ \cos(\pi\, x_i[3]/6))^T$

$\vec{x} = \begin{pmatrix} 2 \\ 3 \end{pmatrix}$ $\phi(\vec{x}) = \begin{pmatrix} 1 \\ \sqrt{2}\cdot 2 \\ \sqrt{2}\cdot 3 \\ 4 \\ \sqrt{2}\cdot 6 \\ 9 \end{pmatrix}$

black = 1
white = -1

NOT Linearly separable, but there is a clear pattern



or map to x^2 + y^2

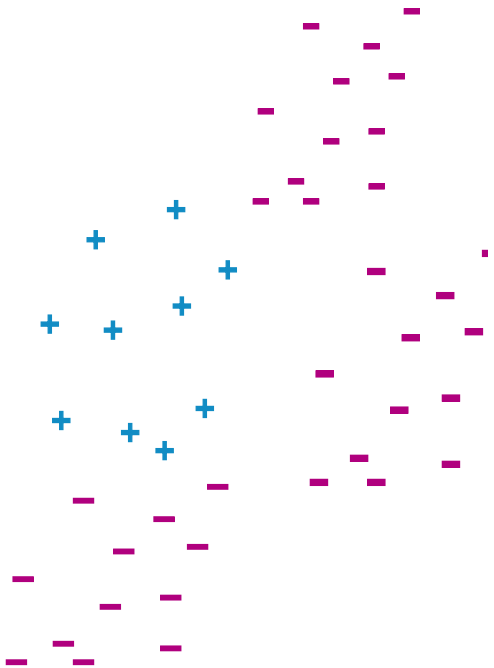
Figure 5.2: Data that is not linearly separable in the input space $\mathbb{R}^2$ but that is linearly separable in the "ϕ-space," $\phi(\mathbf{x}) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, \sqrt{2}x_1x_2, x_2^2)$.

our hyperplane that linearly separates the data in the higher dimensional space.

For instance, the hyperplane $\phi(\mathbf{x})^T\mathbf{w}^* = 0$ for $\mathbf{w}^* = (-4, 0, 0, 1, 0, 1)$ circle $x_1^2 + x_2^2 = 4$ in the original space, such as in Figure 5.2.

From "Foundations of Data Science" by Blum, Kannan, Hopcroft

What to do if data are not linearly separable?



Use feature maps…

$$\phi(x) : \mathbb{R}^d \rightarrow F$$

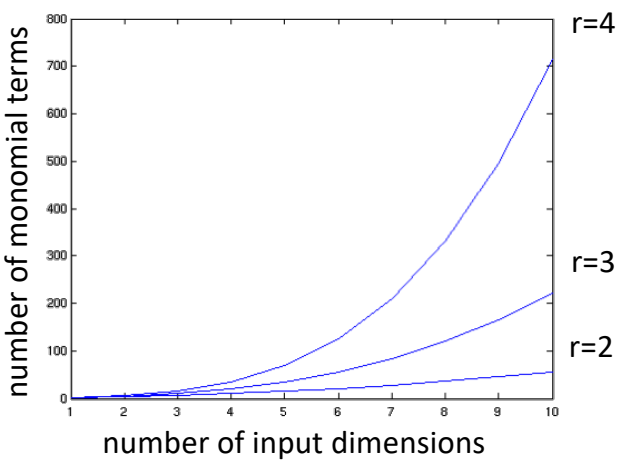
$$\Phi(\mathbf{x}) = \begin{pmatrix} 1 \\ x[1] \\ x[2] \\ \dots \\ x[1]^2 \\ x[2]^2 \\ \dots \\ x[1]x[2] \\ x[1]x[3] \\ \dots \end{pmatrix} \quad \text{where} \quad \mathbf{x} = (x[1], \dots, x[d])$$

$$\Phi(\mathbf{x}) = \begin{pmatrix} 1 \\ x[1] \\ \dots \\ \dots \\ x[1]e^{\sin(x[3])} \\ e^{-x[2]^2/\sigma^2} \\ \dots \end{pmatrix}$$

Could even be infinite dimensional….

Example: Higher order polynomials

Feature space can get really large really quickly!



$$\phi(\mathbf{u}) = \begin{pmatrix} 1 \\ u_1 \\ u_2 \\ \dots \\ u_1^2 \\ u_1 u_2 \\ \dots \\ u_1^5 u_3 u_{17}^4 \\ \dots \end{pmatrix}$$

d – input dimension
r – degree of polynomial

$$\binom{r+d-1}{r} \text{ terms}$$

grows fast!
r = 6, d = 100
about 1.6 billion terms

Kernel trick

- Allows us to do these apparently intractable calculations efficiently!

Perceptron revisited

Initialize $\mathbf{w}_0$.
for $t := 1$ to T
- Observe feature vector $\mathbf{x}_t$.
- Make a prediction: $\hat{y} = \text{sign}(\mathbf{x}_t^T \mathbf{w}_t)$
- Observe the true label $y_t \in \{-1, 1\}$.
- Update the model:
  - If $\hat{y} = y_t$, then $\mathbf{w}_{t+1} := \mathbf{w}_t$.
  - Otherwise ($\hat{y} \neq y_t$), then $\mathbf{w}_{t+1} := \mathbf{w}_t + y_t \mathbf{x}_t$.

When using high dimensional features:

sign (sum(y_j PHI(x_t) PHI(x_j))

Write weight vector in terms of mistaken data points only.

Let $M^{(t)}$ be time steps up to *t* when mistakes were made:

w_{t+1} = sum(y_j x_j) over all mistakes

Prediction rule becomes:

sign (x_t dot sum (y_j x_j) for mistakes j)
= sign (sum (y_j x_t dot x_j) for mistakes j)

Classification only depends on inner products!

Why does dependence on inner products help?

- Because sometimes they can be computed **much much much more efficiently**.

$$\phi(\mathbf{u}) = \begin{pmatrix} 1 \\ u_1 \\ u_2 \\ \dots \\ u_1^2 \\ u_1 u_2 \\ \dots \\ u_1^5 u_3 u_{17}^4 \\ \dots \end{pmatrix} \qquad \phi(u) \cdot \phi(v) = \begin{pmatrix} 1 \\ u_1 \\ u_2 \\ \dots \\ u_1^2 \\ u_1 u_2 \\ \dots \\ u_1^5 u_3 u_{17}^4 \\ \dots \end{pmatrix} \cdot \begin{pmatrix} 1 \\ v_1 \\ v_2 \\ \dots \\ v_1^2 \\ v_1 v_2 \\ \dots \\ v_1^5 v_3 v_{17}^4 \\ \dots \end{pmatrix}$$

$$\binom{r+d-1}{r}$$

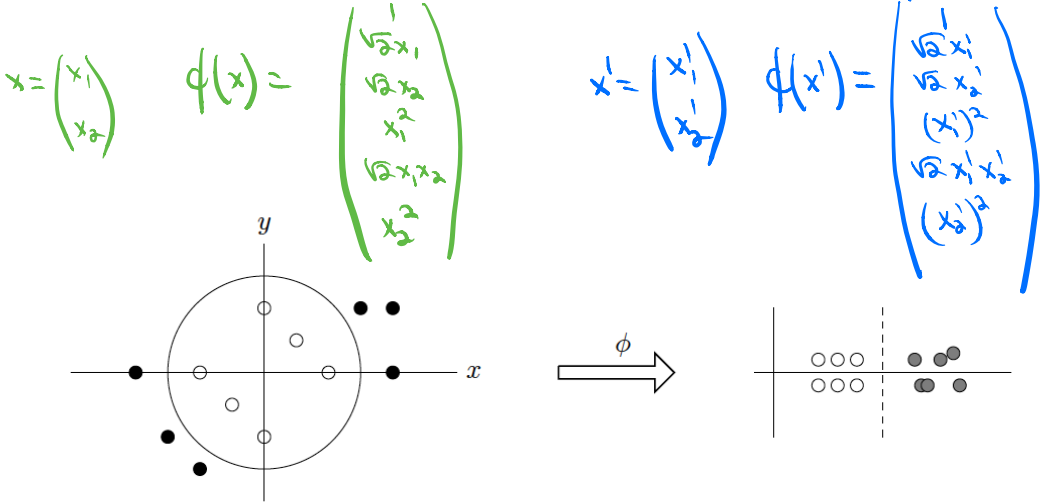


Figure 5.2: Data that is not linearly separable in the input space $\mathbb{R}^2$ but that is linearly separable in the "ϕ-space," $\phi(\mathbf{x}) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, \sqrt{2}x_1x_2, x_2^2)$, corresponding to the kernel function $K(\mathbf{x}, \mathbf{x}') = (1 + x_1x_1' + x_2x_2')^2$.

For instance, the hyperplane $\phi(\mathbf{x})^T\mathbf{w}^* = 0$ for $\mathbf{w}^* = (-4, 0, 0, 1, 0, 1)$ circle $x_1^2 + x_2^2 = 4$ in the original space, such as in Figure 5.2.

From "Foundations of Data Science" by Blum, Kannan, Hopcroft

Dot-product of polynomials: why useful?

- Because sometimes they can be computed much much much more efficiently.

$$\phi(\mathbf{u}) = \begin{pmatrix} 1 \\ u_1 \\ u_2 \\ \dots \\ u_1^2 \\ u_1 u_2 \\ \dots \\ u_1^5 u_3 u_{17}^4 \\ \dots \end{pmatrix} \qquad \phi(u) \cdot \phi(v) = \begin{pmatrix} 1 \\ u_1 \\ u_2 \\ \dots \\ u_1^2 \\ u_1 u_2 \\ \dots \\ u_1^5 u_3 u_{17}^4 \\ \dots \end{pmatrix} \cdot \begin{pmatrix} 1 \\ v_1 \\ v_2 \\ \dots \\ v_1^2 \\ v_1 v_2 \\ \dots \\ v_1^5 v_3 v_{17}^4 \\ \dots \end{pmatrix}$$

So with appropriate constants in front

u and v are only dimension d! MUCH smaller

$$\phi(\mathbf{u}) \cdot \phi(\mathbf{v}) = (1 + \mathbf{u} \cdot \mathbf{v})^r$$

$$(1 + \mathbf{u} \cdot \mathbf{v})^r = [1 + u_1 v_1 + u_2 v_2 + \dots u_d v_d]^r$$

Typical term: $C \cdot (u_1 v_1)^{i_1} (u_2 v_2)^{i_2} \cdots (u_d v_d)^{i_d}$ where $\sum_{j=1}^{d} i_j \le r$

constants are unimportant…

Kernel trick

- Allows us to do these apparently intractable calculations efficiently!
- If we can compute inner products efficiently, can run the algorithm (e.g., Perceptron) efficiently.
- Hugely important idea in ML.

Finally, the kernel trick
Kernelized perceptron

A Kernel function K(x, x') is a function that implements inner products in the feature space

K(x, x') = phi(x) dot phi(x')
where phi: R^d —> F

where phi maps from the features to some higher dimensional feature space.

- Every time you make a mistake, remember $(\mathbf{x}_t, y_t)$

- Kernelized perceptron prediction for x:

$$\begin{aligned}\text{sign}(\mathbf{w}_t \cdot \phi(\mathbf{x})) &= \sum_{i \in M^{(t)}} y_i (\phi(\mathbf{x}_i) \cdot \phi(\mathbf{x})) \\ &= \sum_{i \in M^{(t)}} y_i K(\mathbf{x}_i, \mathbf{x}).\end{aligned}$$

$$K(\mathbf{u}, \mathbf{v}) = \phi(\mathbf{u}) \cdot \phi(\mathbf{v}) = (1 + \mathbf{u} \cdot \mathbf{v})^r$$

Common kernels

- Polynomials of degree exactly p

$$K(\mathbf{u}, \mathbf{v}) = (\mathbf{u} \cdot \mathbf{v})^p$$

- Polynomials of degree up to p

$$K(\mathbf{u}, \mathbf{v}) = (\mathbf{u} \cdot \mathbf{v} + 1)^p$$

- **Gaussian (squared exponential) kernel**

$$K(\mathbf{u}, \mathbf{v}) = \exp\left(-\frac{||\mathbf{u} - \mathbf{v}||^2}{2\sigma^2}\right)$$

- Many many many others

Kernelizing a machine learning algorithm

- Prove that the solution is always in span of training points. I.e.,
- Rewrite the algorithm so that all training or test inputs are accessed only through inner products with other inputs.
- Choose (or define) a kernel function and substitute $K(\mathbf{x}_i, \mathbf{x}_j)$ for $\mathbf{x}_i^T \mathbf{x}_j$

So with appropriate constants in front in definition of $\phi(\cdot)$

$$K(\mathbf{u}, \mathbf{v}) = \phi(\mathbf{u}) \cdot \phi(\mathbf{v}) = (1 + \mathbf{u} \cdot \mathbf{v})^r$$

What you need to know

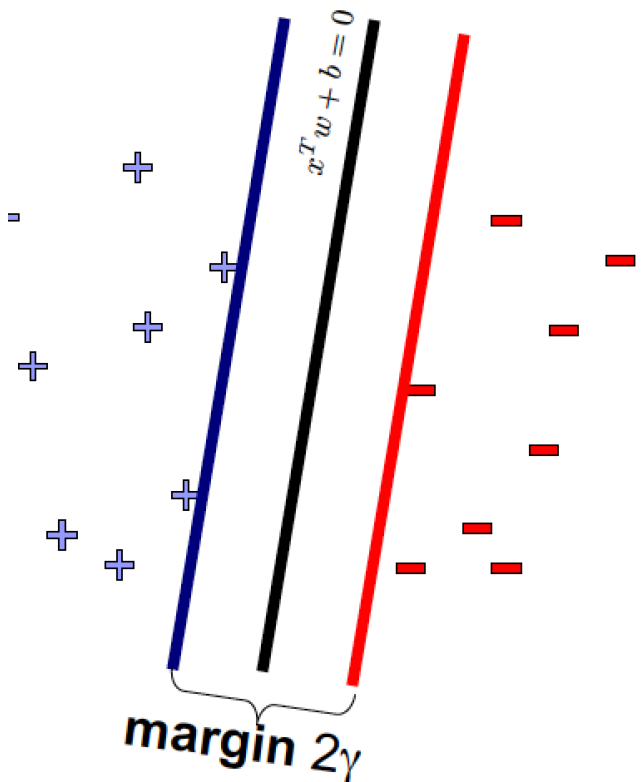
- Linear separability in higher-dim feature space
- The kernel trick
- Kernelized perceptron
- Polynomial and other common kernels (will see more later)

Support Vector Machines

CSE 446: Machine Learning
Slides by Emily Fox + Kevin Jamieson + others
Presented by Anna Karlin

May 1, 2019

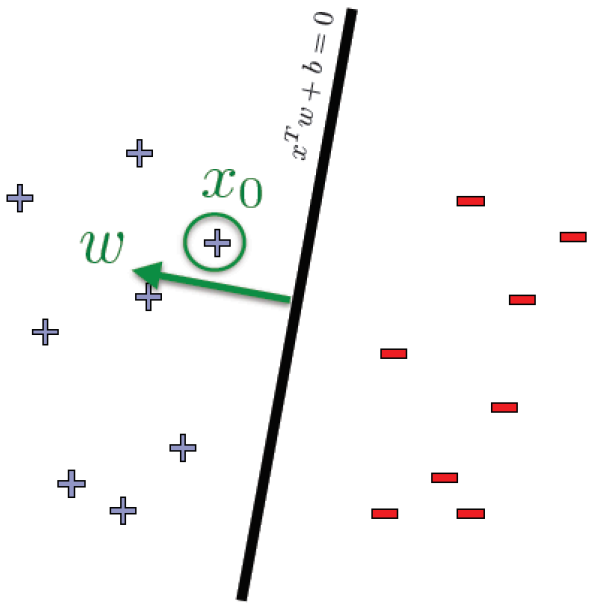
Linear classifiers—Which line is better?



©2018 Kevin Jamieson

Maximizing the margin for linearly separable data

Given hyperplane, what is margin?

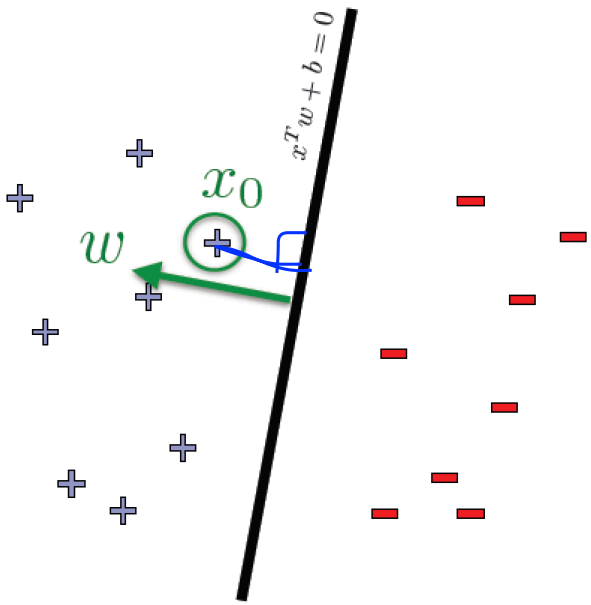


Distance from x_0 to hyperplane defined by $x^T w + b = 0$?

larger margin between hyperplane and data = larger margin of error in forming a classification.

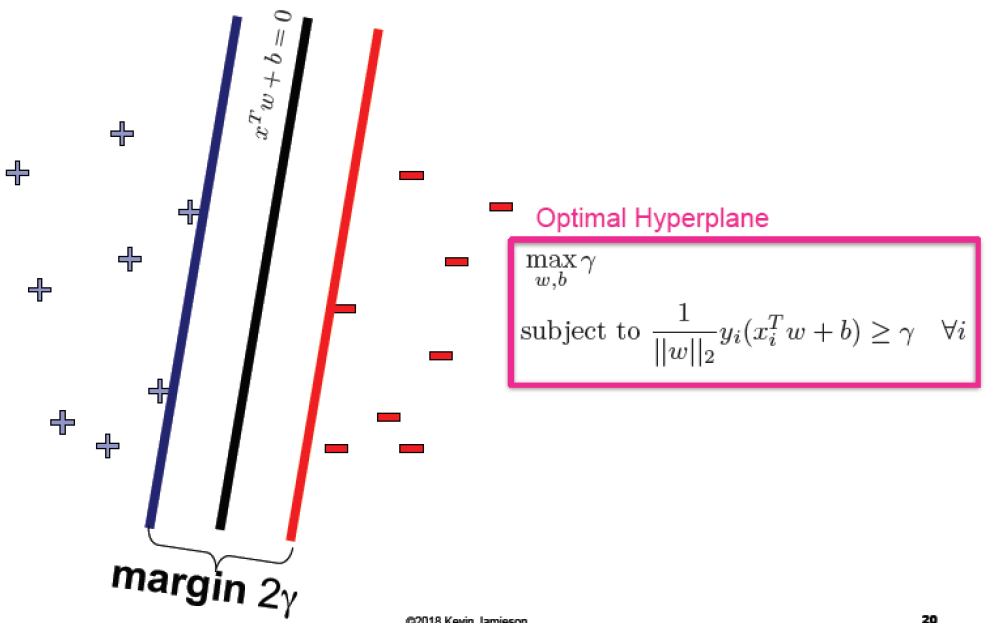
Choose hyperplane that is as far away from the nearest data point

Given hyperplane, what is margin?



Line perpendicular

Our optimization problem



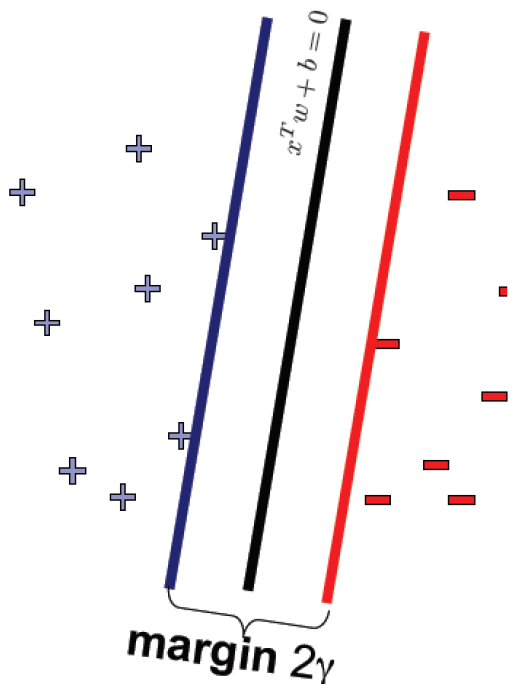
Optimal Hyperplane

$$\max_{w,b} \gamma$$

$$\text{subject to } \frac{1}{||w||_2} y_i (x_i^T w + b) \geq \gamma \quad \forall i$$

END LECTURE WED MAY 1

Final version



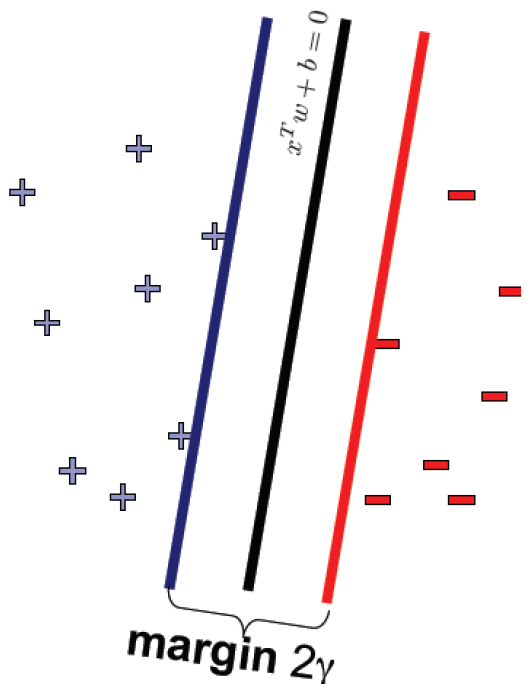
Solvable efficiently –
quadratic programming problem

Optimal Hyperplane (reparameterized)

$$\min_{w,b} ||w||_2^2$$

$$\text{subject to } y_i(x_i^T w + b) \geq 1 \quad \forall i$$

What are support vectors?



Optimal Hyperplane (reparameterized)

$$\min_{w,b} ||w||_2^2$$

$$\text{subject to } y_i(x_i^T w + b) \geq 1 \quad \forall i$$

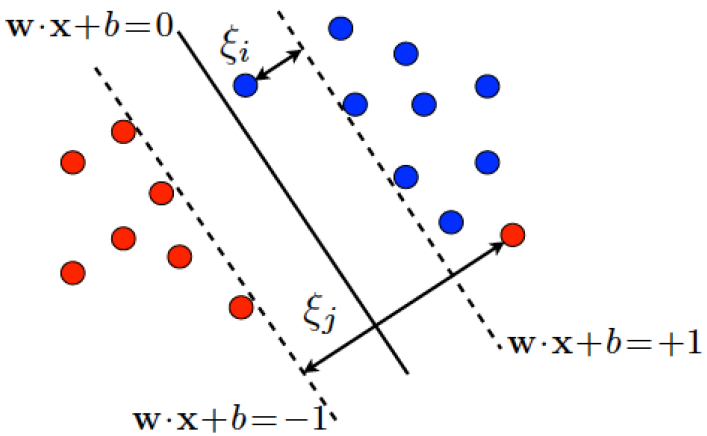
What if the data are not linearly separable?

What if data are not linearly separable?

Use feature maps…

SVMs can be kernelized!!!!

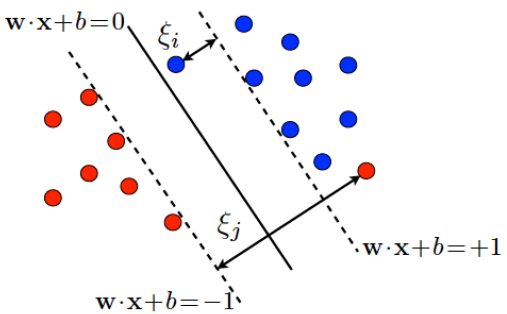
What if data are still not linearly separable?



$$y_i(\mathbf{w}\cdot\mathbf{x}_i+b)\geq 1-\xi_i$$

Courtesy Mehryar Mohri

What if data are still not linearly separable?



Courtesy Mehryar Mohri

$$\min_{\mathbf{w},b,\xi} ||\mathbf{w}||^2 + C\sum_i \xi_i$$

$$y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1 - \xi_i \quad \forall i$$
$$\xi_i \geq 0 \quad \forall i.$$

Final objective

$$\frac{1}{n}\sum_{i=1}^{n}(1 - y_i((\mathbf{w}^T\mathbf{x}_i + b))_+ + \lambda||\mathbf{w}||_2^2$$

Gradient descent for SVMs

Minimizing regularized hinge loss (aka SVMs)

- Given a dataset: $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)$
- Minimize regularized hinge loss: $\frac{1}{n} \sum_i (1 - y_i((\mathbf{w}^T \mathbf{x}_i + b))_+ + \lambda ||\mathbf{w}||_2^2$

Subgradient of hinge loss

- Hinge loss: $\ell((\mathbf{x}, y), \mathbf{w}) = (1 - y((\mathbf{w}^T\mathbf{x} + b))_+$

- Subgradient of hinge loss:

$$\partial_{\mathbf{w}}\ell((\mathbf{x}, y), \mathbf{w}) = \begin{cases} & y(\mathbf{w}^T\mathbf{x} + b) > 1 \\ & y(\mathbf{w}^T\mathbf{x} + b) < 1 \\ & y(\mathbf{w}^T\mathbf{x} + b) = 1 \end{cases}$$

Subgradient of hinge loss

- Hinge loss: $\ell((\mathbf{x}, y), \mathbf{w}) = (1 - y((\mathbf{w}^T\mathbf{x} + b))_+$

- Subgradient of hinge loss:

$$\partial_{\mathbf{w}}\ell((\mathbf{x}, y), \mathbf{w}) = \begin{cases} \mathbf{0} & y(\mathbf{w}^T\mathbf{x} + b) > 1 \\ -y\mathbf{x} & y_(\mathbf{w}^T\mathbf{x} + b) < 1 \\ [-y\mathbf{x}, \mathbf{0}] & y(\mathbf{w}^T\mathbf{x} + b) = 1 \end{cases}$$

In one line:

$$\partial_{\mathbf{w}}\ell((\mathbf{x}, y), \mathbf{w}) = \mathbb{I}\{y(\mathbf{w}^T\mathbf{x} + b) \leq 1\}(-y\mathbf{x})$$

Subgradient descent for hinge minimization

- Given data: $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)$

- Want to minimize:

$$\frac{1}{n}\sum_{i=1}^{n} \ell((\mathbf{x}_i, y_i), \mathbf{w}) + \lambda||\mathbf{w}||_2^2 = \frac{1}{n}\sum_{i}(1 - y_i((\mathbf{w}^T\mathbf{x}_i + b))_+ + \lambda||\mathbf{w}||_2^2$$

- As we've discussed, subgradient descent works like gradient descent:
  - But if there are multiple subgradients at a point, just pick (any) one:

$$\begin{aligned}
\mathbf{w}_{t+1} &:= \mathbf{w}_t - \eta\left(\frac{1}{n}\sum_{i=1}^{n} \partial_{\mathbf{w}}\ell((\mathbf{x}_i, y_i), \mathbf{w}) + 2\lambda\mathbf{w}_t\right) \\
&= \mathbf{w}_t - \eta\left(\frac{1}{n}\sum_{i=1}^{n} \mathbb{I}\{y(\mathbf{w}_t \cdot \mathbf{x}_i + b) \leq 1\}(-y_i\mathbf{x}_i) + 2\lambda\mathbf{w}_t\right) \\
&= \mathbf{w}_t + \eta\frac{1}{n}\sum_{i=1}^{n} \mathbb{I}\{y(\mathbf{w}_t \cdot \mathbf{x}_i + b) \leq 1\}(y_i\mathbf{x}_i) - \eta 2\lambda\mathbf{w}_t.
\end{aligned}$$

SVM

- **Gradient Descent Update**

$$\begin{aligned}\mathbf{w}_{t+1} &:= \mathbf{w}_t - \eta\left(\frac{1}{n}\sum_{i=1}^{n}\partial_{\mathbf{w}}\ell((\mathbf{x}_i, y_i), \mathbf{w}) + 2\lambda\mathbf{w}_t\right) \\ &= \mathbf{w}_t - \eta\left(\frac{1}{n}\sum_{i=1}^{n}\mathbb{I}\{y(\mathbf{w}_t \cdot \mathbf{x}_i + b) \leq 1\}(-y_i\mathbf{x}_i) + 2\lambda\mathbf{w}_t\right) \\ &= \mathbf{w}_t + \eta\frac{1}{n}\sum_{i=1}^{n}\mathbb{I}\{y(\mathbf{w}_t \cdot \mathbf{x}_i + b) \leq 1\}(y_i\mathbf{x}_i) - \eta 2\lambda\mathbf{w}_t.\end{aligned}$$

- **SGD update**

$$\mathbf{w}_{t+1} := \mathbf{w}_t + \eta\,\mathbb{I}\{y(\mathbf{w}_t \cdot \mathbf{x}_i + b) \leq 1\}(y_i\mathbf{x}_i) - \eta 2\lambda\mathbf{w}_t.$$

Machine learning problems

- Given i.i.d. data set:

$$(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)$$

- Find parameters **w** to minimize average loss (or regularized version):

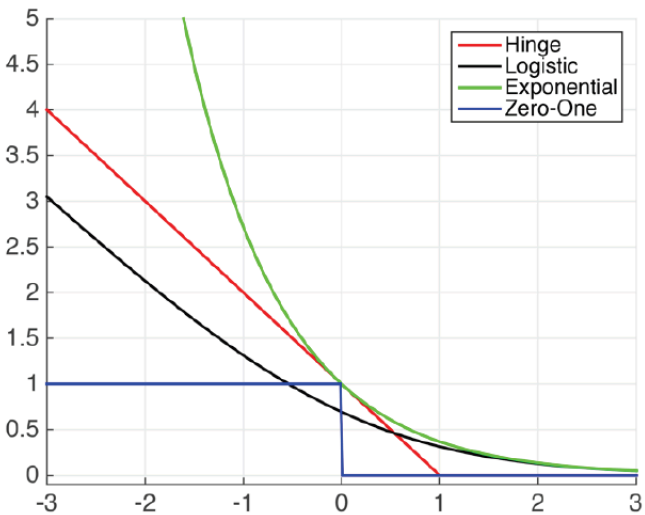
$$\frac{1}{n} \sum_{i=1}^{n} \ell_i(\mathbf{w})$$

Squared loss: $\ell_i(\mathbf{w}) = (y_i - x_i^T \mathbf{w})^2$

Logistic loss: $\ell_i(\mathbf{w}) = \log(1 + \exp(-y_i \mathbf{x}_i^T \mathbf{w}))$

Hinge loss: $\ell_i(\mathbf{w}) = \max\{0, 1 - y_i \mathbf{x}_i^T \mathbf{x}\}$

Courtesy Killian Weinberger



What you need to know…

- Maximizing margin
- Derivation of SVM formulation
- Non-linearly separable case
  - Hinge loss
  - a.k.a. adding slack variables
- Can optimize SVMs with SGD
  - Many other approaches possible