Applied Machine Learning

Perceptron and Support Vector Machines

Reihaneh Rabbany

School of Computer Science, McGill

COMP 551 (winter 2020)

Learning objectives

geometry of linear classification

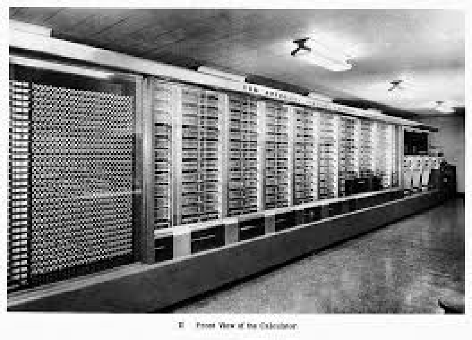
Perceptron learning algorithm

margin maximization and support vectors

hinge loss and relation to logistic regression

Perceptron

old implementation (1960's)



historically a significant algorithm

(first neural network, or rather just a neuron)

biologically motivated model

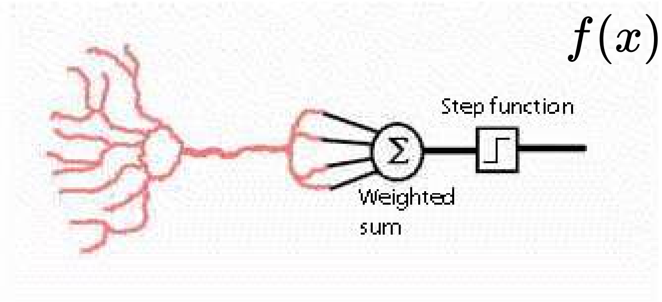
simple learning algorithm

convergence proof

beginning of *connectionist* AI

it's criticism in the book "Perceptrons" was a factor in AI winter

Model



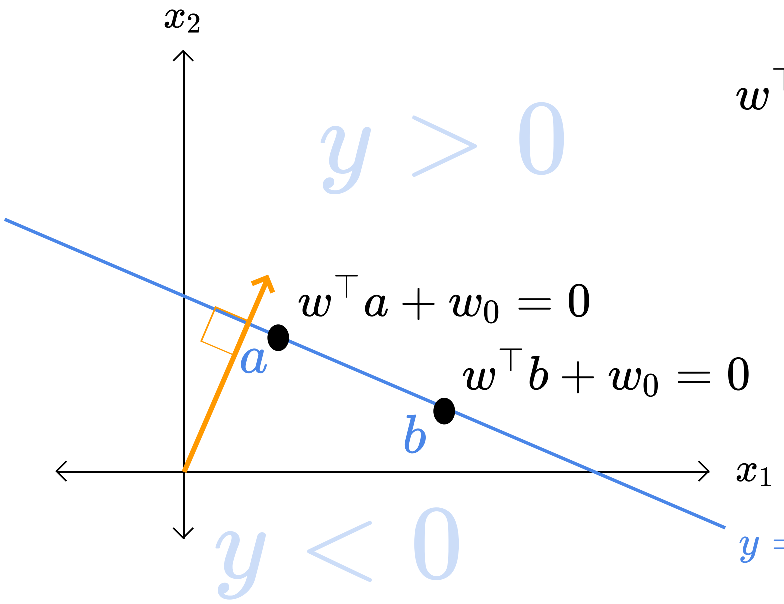
$$f(x) = \boxed{\text{sign}}(w^\top x + w_0)$$

note that we're using +1/-1 for labels rather than 0/1.

image:https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/Neuron/index.html geometry of the **separating hyperplane**

this hyperplane has one dimension lower than D (number of features)

for any two points **a** and **b** on the line



$$w^\top (a - b) + w_0 - w_0 = 0$$

$$y = w^\top x + w_0 = w_2 x_2 + w_1 x_1 + w_0 = 0$$

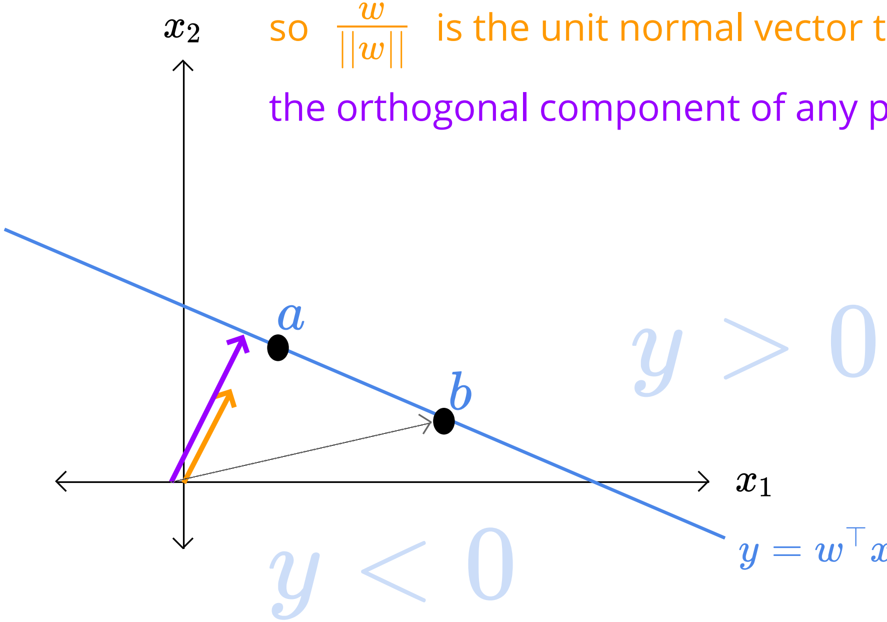
geometry of the **separating hyperplane**

this hyperplane has one dimension lower than D (number of features)

for any two points **a** and **b** on the line $w^\top(a-b) + w_0 - w_0 = 0$

so $\frac{w}{||w||}$ is the unit normal vector to the line

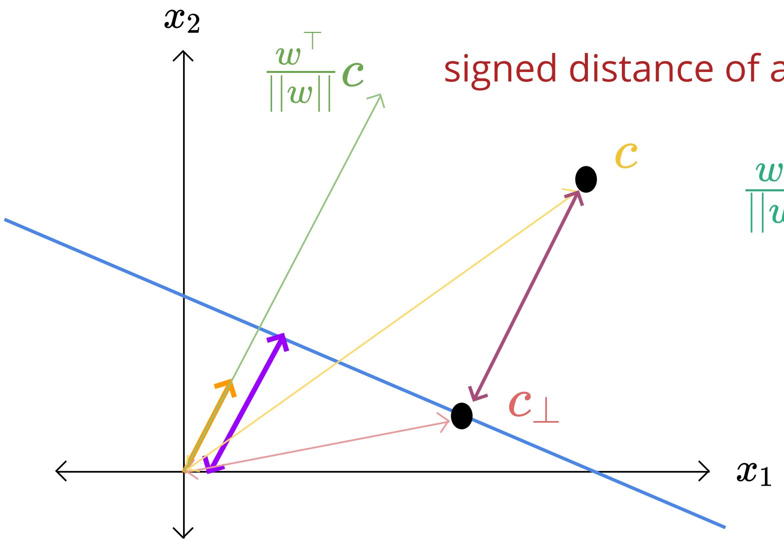
the orthogonal component of any point on the line $\frac{w^\top}{||w||}b = -\frac{w_0}{||w||}$



$y = w^\top x + w_0 = w_2 x_2 + w_1 x_1 + w_0 = 0$

geometry of the **separating hyperplane**

the orthogonal component of any point on the line $\frac{w^\top}{||w||} b = -\frac{w_0}{||w||}$



signed distance of any point (c) from the line

$$\frac{w^\top}{||w||} c - \frac{w^\top}{||w||} c_\perp = \frac{1}{||w||}(w^\top c + w_0)$$

$\frac{w}{||w||}$

$c_\perp$

$\frac{w^\top}{||w||} c_\perp$

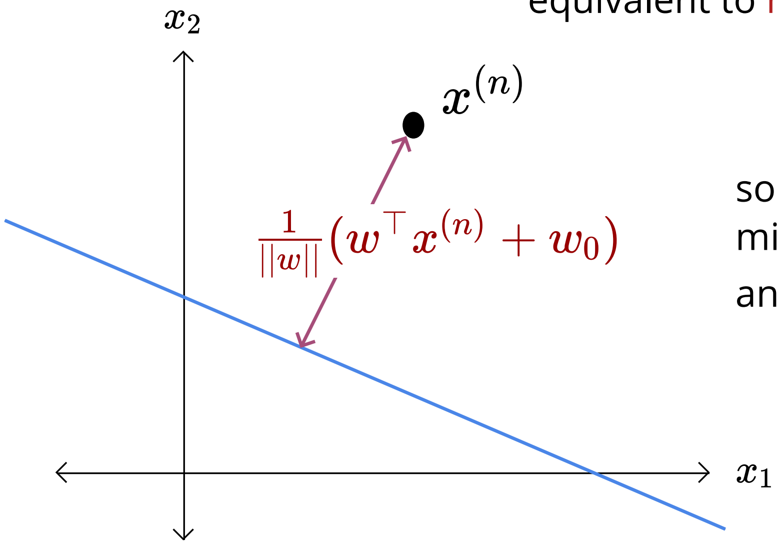
$\frac{w^\top}{||w||} c$

Perceptron: objective

if $y^{(n)}\hat{y}^{(n)} < 0$ try to increase it

label and prediction have different signs

equivalent to minimizing $-y^{(n)}\overbrace{(w^\top x^{(n)} + w_0)}^{\hat{y}^{(n)}}$



so perceptron tries to minimize the distance of misclassified points from the decision boundary

and push them to the right side

revisiting

Perceptron: optimization

if $y^{(n)}\hat{y}^{(n)} < 0$ minimize $J_n(w) = -y^{(n)}(w^\top x^{(n)})$

now we included bias in w

otherwise, do nothing

use stochastic gradient descent $\nabla J_n(w) = -y^{(n)}x^{(n)}$

$$w^{\{t+1\}} \leftarrow w^{\{t\}} - \alpha \nabla J_n(w) = w^{\{t\}} + \alpha y^{(n)}x^{(n)}$$

Perceptron uses learning rate of 1

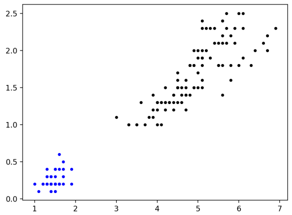
this is okay because scaling w does not affect prediction

$$\mathrm{sign}(w^\top x) = \mathrm{sign}(\alpha w^\top x)$$

Perceptron convergence theorem

the algorithm is guaranteed to converge in finite steps if linearly separable

Perceptron: example

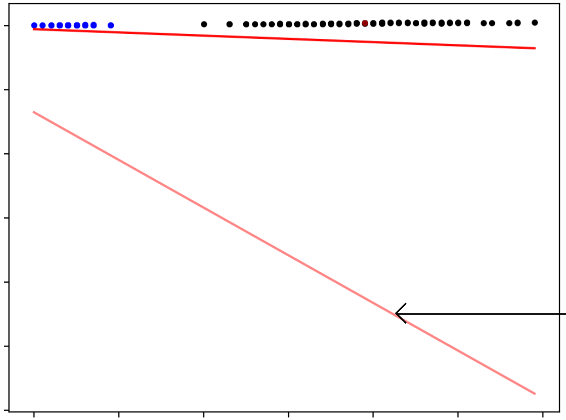


Iris dataset
(linearly separable case)

```
def Perceptron(X, y, max_iters):
    N,D = X.shape
    w = np.random.rand(D)
    for t in range(max_iters):
        n = np.random.randint(N)
        yh = np.sign(np.dot(X[n,:], w))
        if yh != y[n]:
                w = w + y[n]*X[n,:]
    return w
```

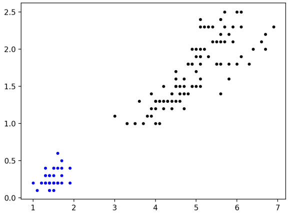
note that the code is not chacking for convergence

iteration 1



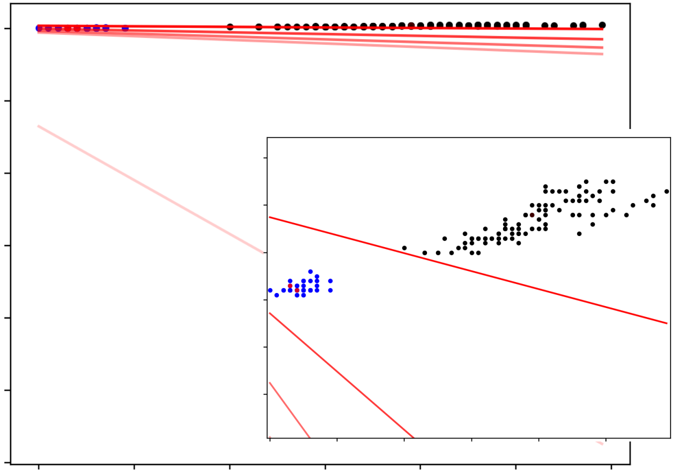
initial decision boundary $w^\top x = 0$

Perceptron: example



Iris dataset
(linearly separable case)

iteration 10



```python
def Perceptron(X, y, max_iters):
    N,D = X.shape
    w = np.random.rand(D)
    for t in range(max_iters):
        n = np.random.randint(N)
        yh = np.sign(np.dot(X[n,:], w))
        if yh != y[n]:
                w = w + y[n]*X[n,:]
    return w
```

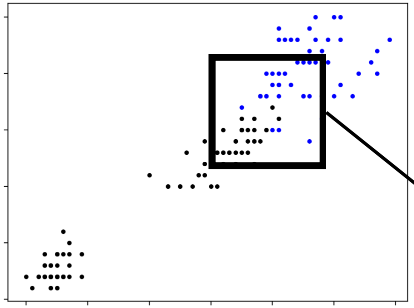
note that the code is not chacking for convergence

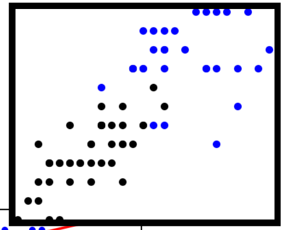
observations:

after finding a linear separator no further updates happen

the final boundary depends on the order of instances
(different from all previous methods)

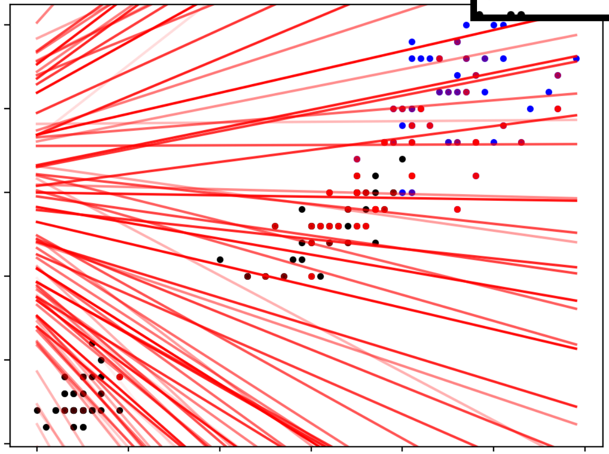
Perceptron: example





Iris dataset

(**NOT** linearly separable case)



```
def Perceptron(X, y, max_iters):
    N,D = X.shape
    w = np.random.rand(D)
    for t in range(max_iters):
        n = np.random.randint(N)
        yh = np.sign(np.dot(X[n,:], w))
        if yh != y[n]:
            w = w + y[n]*X[n,:]
    return w
```

note that the code is not chacking for convergence

the algorithm does not converge

there is always a wrong prediction and the weights will be updated

Perceptron: issues

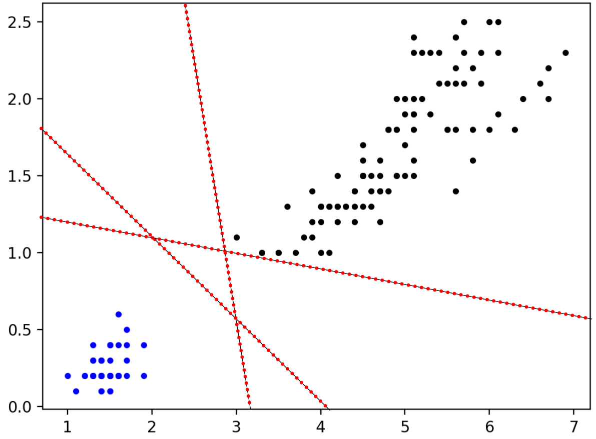
cyclic updates if the data is not linearly separable?

- try make the data separable using additional features?
- data may be inherently noisy

even if linearly separable

convergence could take many iterations

the decision boundary may be suboptimal



Perceptron: issues

cyclic updates if the data is not linearly separable?

- try make the data separable using additional features?
- data may be inherently noisy

even if linearly separable

convergence could take many iterations

let's fix this problem first

assume linear separability

the decision boundary may be suboptimal

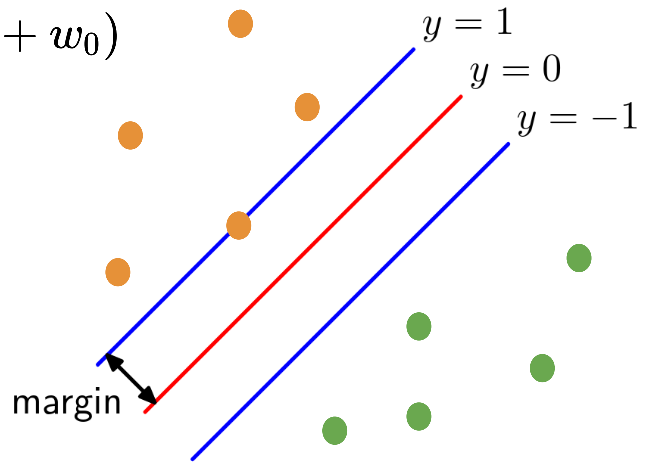
Margin

the margin of a classifier (assuming correct classification)

is the distance of the closest point to the decision boundary

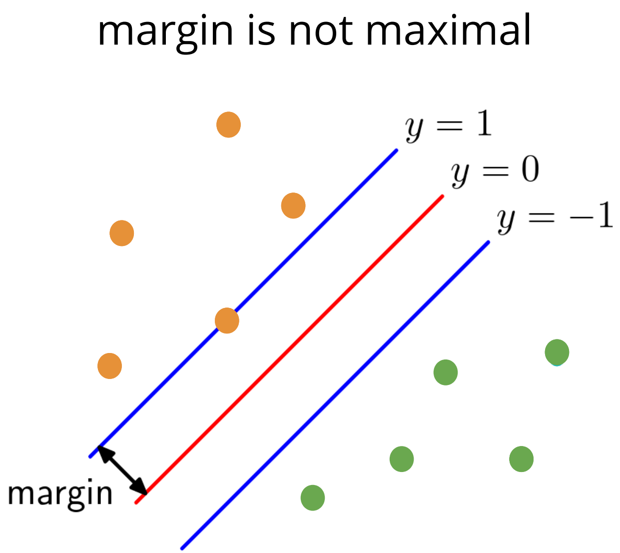
signed distance is $\frac{1}{||w||}(w^\top x^{(n)} + w_0)$

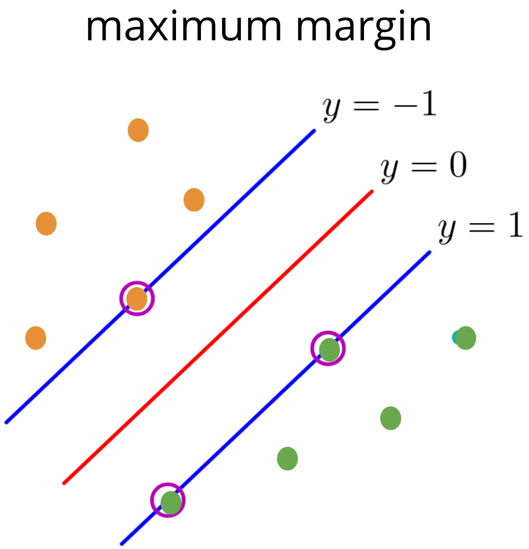
correcting for sign (margin) $\frac{1}{||w||}y^{(n)}(w^\top x + w_0)$



Max margin classifier

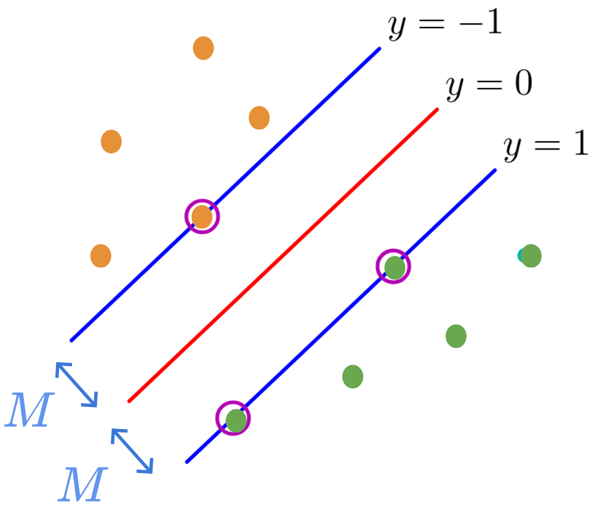
find the decision boundary with maximum margin





Max margin classifier

find the decision boundary with maximum margin



$$\begin{cases} \max_{w,w_0} M \\ M \leq \frac{1}{||w||_2} y^{(n)}(w^\top x^{(n)} + w_0) \quad \forall n \end{cases}$$

only the points (n) with

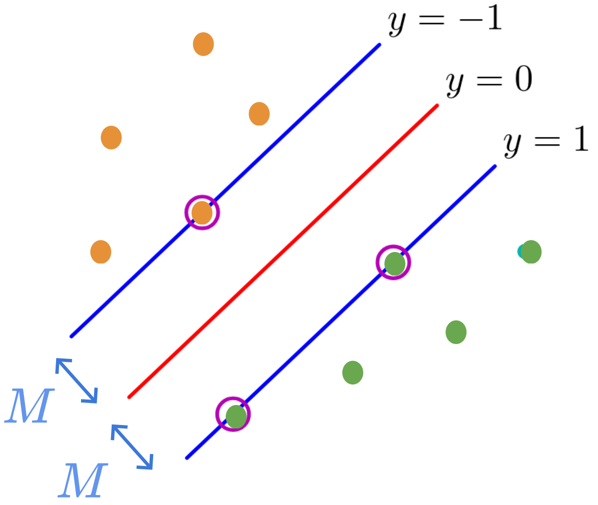
$M = \frac{1}{||w||_2} y^{(n)}(w^\top x^{(n)} + w_0)$ matter in finding the boundary

these are called **support vectors**

max-margin classifier is called **support vector machine** (SVM)

Support Vector Machine

find the decision boundary with maximum margin



$$\begin{cases} \max_{w,w_0} M \\ M \leq \frac{1}{||w||_2} y^{(n)}(w^\top x^{(n)} + w_0) \quad \forall n \end{cases}$$

observation

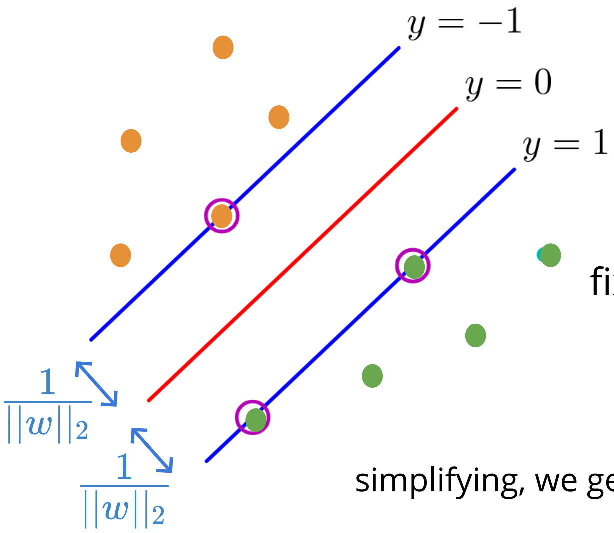
if w^*, w_0^* is an optimal solution then

cw^*, cw_0^* is also optimal (same margin)

fix the norm of w to avoid this $||w||_2 = \frac{1}{M}$

Support Vector Machine

find the decision boundary with maximum margin



$$\begin{cases} \max_{w,w_0} M \\ M \leq \frac{1}{||w||_2} y^{(n)}(w^\top x^{(n)} + w_0) \quad \forall n \end{cases}$$

fixing $||w||_2 = \frac{1}{M}$

$$\begin{cases} \max_{w,w_0} \frac{1}{||w||_2} \\ \frac{1}{||w||_2} \leq \frac{1}{||w||_2} y^{(n)}(w^\top x^{(n)} + w_0) \quad \forall n \end{cases}$$

simplifying, we get hard margin SVM objective

$$\begin{cases} \min_{w,w_0} ||w||_2^2 \\ y^{(n)}(w^\top x^{(n)} + w_0) \geq 1 \quad \forall n \end{cases}$$

Perceptron: issues

cyclic updates if the data is not linearly separable?

- try make the data separable using additional features?
- data may be inherently noisy

now lets fix this problem

maximize a **soft** margin

even if linearly separable

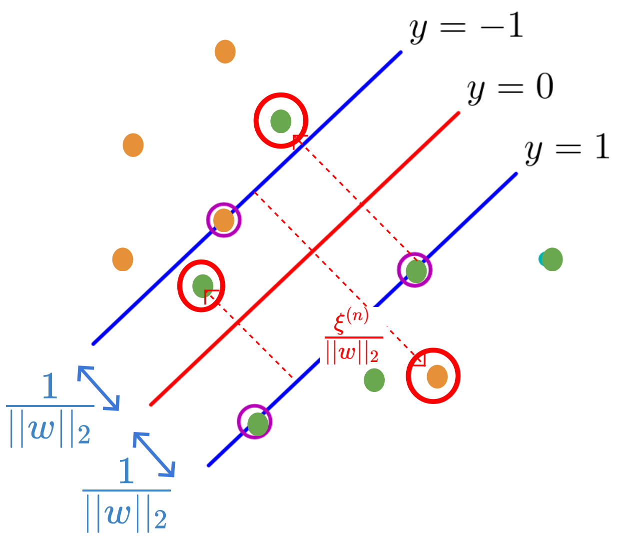
convergence could take many iterations

the decision boundary may be suboptimal

maximize the **hard** margin

Soft margin constraints

allow points inside the margin and on the wrong side
but penalize them



instead of hard constraint $y^{(n)}(w^\top x^{(n)} + w_0) \geq 1 \quad \forall n$

use $y^{(n)}(w^\top x^{(n)} + w_0) \geq 1 - \xi^{(n)} \quad \forall n$

$\xi^{(n)} \geq 0$ slack variables (one for each n)

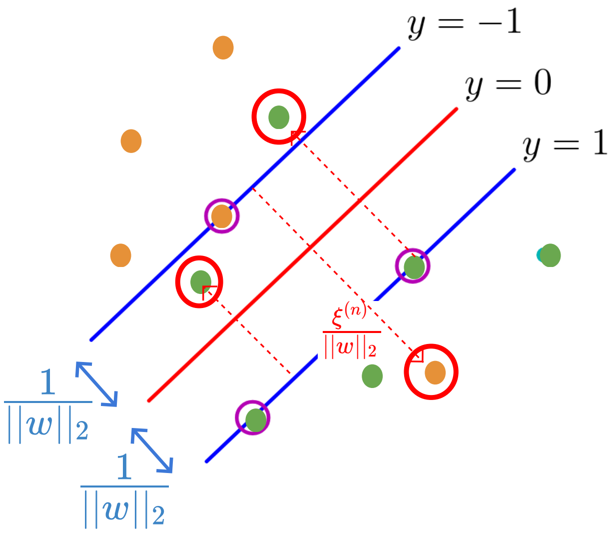
$\xi^{(n)} = 0$ zero if the point satisfies original margin constraint

$0 < \xi^{(n)} < 1$ if correctly classified but inside the margin

$\xi^{(n)} > 1$ incorrectly classified

Soft margin constraints

allow points inside the margin and on the wrong side
but penalize them



soft-margin objective

$$\min_{w,w_0} \frac{1}{2}||w||_2^2 + \gamma \sum_n \xi^{(n)}$$

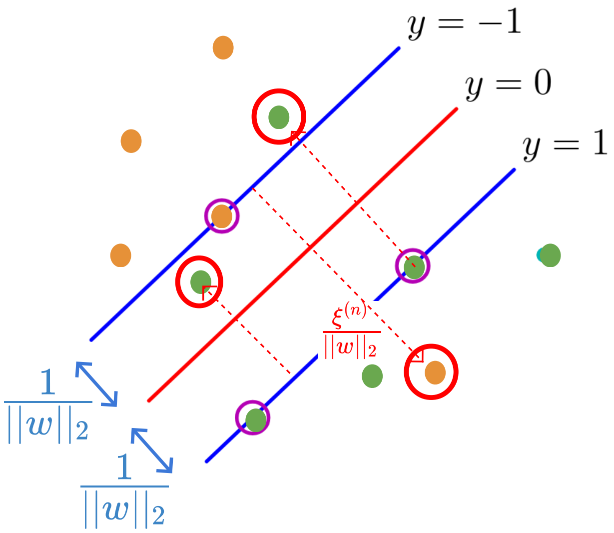
$$y^{(n)}(w^\top x^{(n)} + w_0) \geq 1 - \xi^{(n)} \quad \forall n$$

$$\xi^{(n)} \geq 0 \quad \forall n$$

γ is a hyper-parameter that defines the importance of constraints
for very large γ this becomes similar to hard margin svm

Hinge loss

would be nice to turn this into an unconstrained optimization



$$\min_{w,w_0} \frac{1}{2}||w||_2^2 + \gamma \sum_n \xi^{(n)}$$

$$y^{(n)}(w^\top x^{(n)} + w_0) \geq 1 - \xi^{(n)}$$

$$\xi^{(n)} \geq 0 \quad \forall n$$

if point satisfies the margin $y^{(n)}(w^\top x^{(n)} + w_0) \geq 1$

minimum slack is $\xi^{(n)} = 0$

otherwise $y^{(n)}(w^\top x^{(n)} + w_0) < 1$

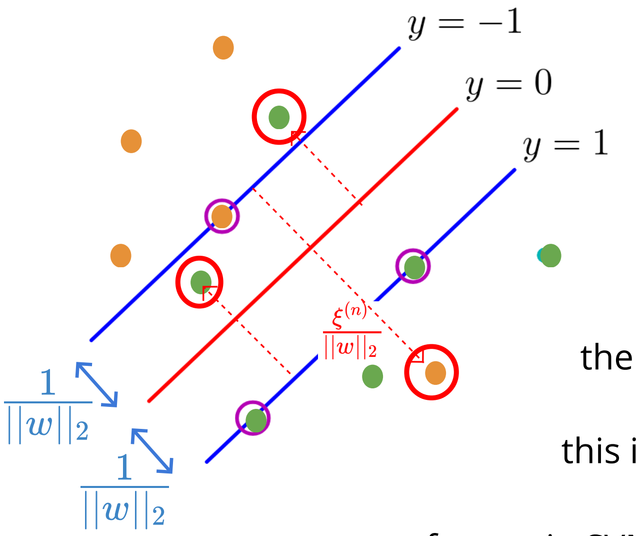
the smallest slack is $\xi^{(n)} = 1 - y^{(n)}(w^\top x^{(n)} + w_0)$

so the optimal slack satisfying both cases

$$\xi^{(n)} = \max(0, 1 - y^{(n)}(w^\top x^{(n)} + w_0))$$

Hinge loss

would be nice to turn this into an unconstrained optimization



$$\min_{w,w_0} \tfrac{1}{2}||w||_2^2 + \gamma \textstyle\sum_n \xi^{(n)}$$

$$y^{(n)}(w^\top x^{(n)} + w_0) \geq 1 - \xi^{(n)}$$

$$\xi^{(n)} \geq 0 \quad \forall n$$

replace $\xi^{(n)} = \max(0, 1 - y^{(n)}(w^\top x^{(n)} + w_0))$

we get $\min_{w,w_0} \tfrac{1}{2}||w||_2^2 + \gamma \sum_n \max(0, 1 - y^{(n)}(w^\top x^{(n)} + w_0))$

the same as $\min_{w,w_0} \sum_n \max(0, 1 - y^{(n)}(w^\top x^{(n)} + w_0)) + \tfrac{1}{2\gamma}||w||_2^2$

this is called the hinge loss $L_{hinge}(y, \hat{y}) = \max(0, 1 - y\hat{y})$

soft-margin SVM is doing L2 regularized hinge loss minimization

Perceptron vs. SVM

Perceptron

if correctly classified evaluates to zero
otherwise it is $\min_{w,w_0} -y^{(n)}(w^\top x^{(n)} + w_0))$

can be written as

$\sum_n \max(0, -y^{(n)}(w^\top x^{(n)} + w_0))$

finds some linear decision boundary if exists

stochastic gradient descent with fixed learning rate

SVM

$\sum_n \max(0, 1 - y^{(n)}(w^\top x^{(n)} + w_0)) + \frac{\lambda}{2}||w||_2^2$

so this is the difference!
(plus regularization)

for small lambda finds the max-marging decision boundary

depending on the formulation we have many choices

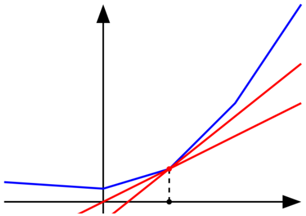
Perceptron vs. SVM

cost $J(w) = \sum_n \max(0, 1 - y^{(n)} w^\top x^{(n)}) + \frac{\lambda}{2}||w||_2^2$

now we included bias in w

```
def cost(X,y,w, lamb=1e-3):
    yh = np.dot(X, w)
    J = np.mean(np.maximum(0, 1 - y*yh)) + lamb * np.dot(w[:-1],w[:-1])/2
    return J
```

check that the cost function is convex in w(?)



hinge loss is not smooth (piecewise linear)

if we use **"stochastic" sub-gradient** descent

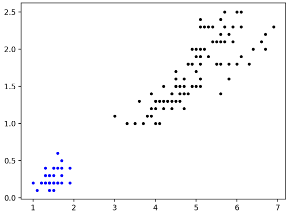
the update will look like Perceptron

if $y^{(n)}\hat{y}^{(n)} < 1$ minimize $-y^{(n)}(w^\top x^{(n)}) + \frac{\lambda}{2}||w||_2^2$

otherwise, do nothing

```
def subgradient(X, y, w, lamb):
    N,D = X.shape
    yh = np.dot(X, w)
    violations = np.nonzero(yh*y < 1)[0]
    grad = -np.dot(X[violations,:].T, y[violations])/N
    grad[:-1] += lamb2 * w[:-1]
    return grad
```

Example

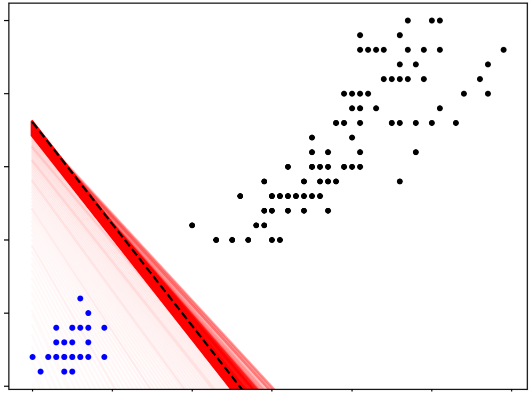


Iris dataset (D=2)

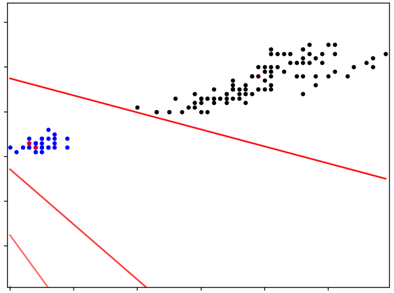
(linearly separable case)

```
def SubGradientDescent(X,y,lr=1,eps=1e-18, max_iters=1000, lamb=1e-8):
    N,D = X.shape
    w = np.zeros(D)
    t = 0
    w_old = w + np.inf
    while np.linalg.norm(w - w_old) > eps and t < max_iters:
        g = subgradient(X, y, w, lamb=lamb)
        w_old = w
        w = w - lr*g/np.sqrt(t+1)
        t += 1
    return w
```

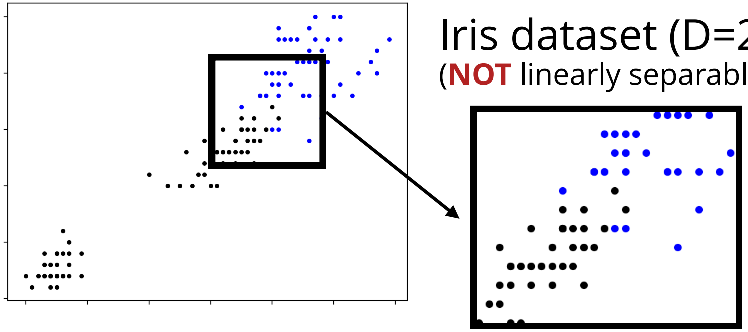
max-margin boundary (using small lambda $\lambda = 10^{-8}$)



compare to Perceptron's decision boundary



Example

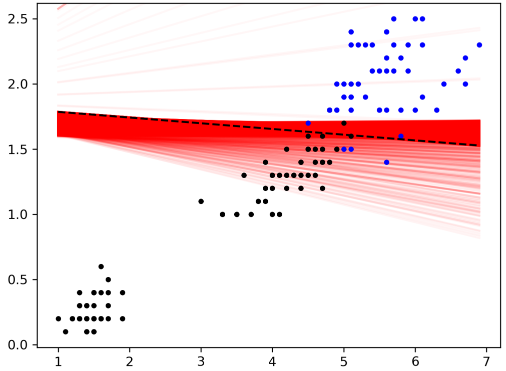


Iris dataset (D=2)

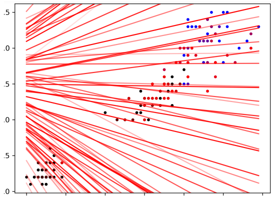
(**NOT** linearly separable case)

```
def SubGradientDescent(X,y,lr=1,eps=1e-18, max_iters=1000, lamb=1e-8):
    N,D = X.shape
    w = np.zeros(D)
    g = np.inf
    t = 0
    while np.linalg.norm(g) > eps and t < max_iters:
        g = subgradient(X, y, w, lamb=lamb)
        w = w - lr*g/np.sqrt(t+1)
        t += 1
    return w
```

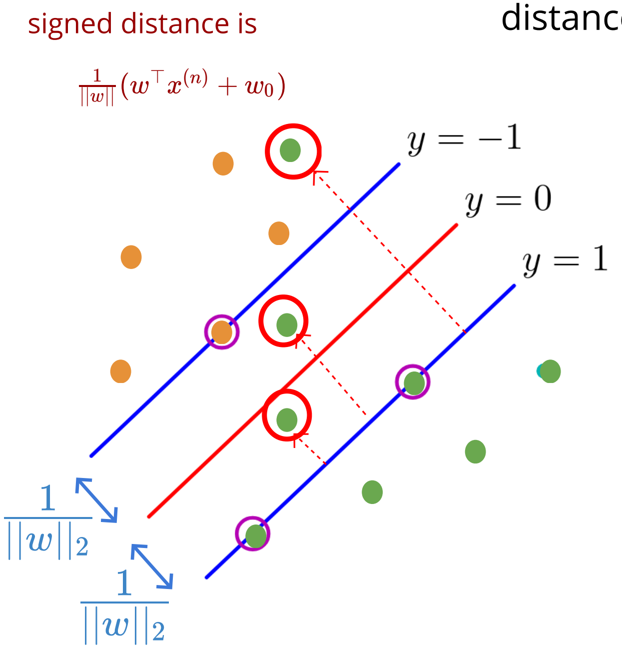
soft margins using small lambda $\lambda = 10^{-8}$



Perceptron does not converge



SVM recap



distance: $\frac{1}{||w||} y^{(n)} (w^\top x + w_0)$

$y^{(n)} \hat{y}^{(n)} < 0$ wrong side

Perceptron $\min_{w,w_0} \sum_n \max(0, -y^{(n)}(w^\top x^{(n)} + w_0))$

minimize the number of points on the wrong side

$$\max_{w,w_0} M = \max_{w,w_0} \frac{1}{||w||_2} = \min_{w,w_0} ||w||_2$$

$\min_{w,w_0} ||w||_2^2$ subject to

hard margin $y^{(n)}(w^\top x^{(n)} + w_0) \geq 1 \quad \forall n$

soft margin $y^{(n)}(w^\top x^{(n)} + w_0) \geq 1 - \xi^{(n)} \quad \forall n$

$$\xi^{(n)} = 1 - y^{(n)}(w^\top x^{(n)} + w_0)$$

SVM $\min_{w,w_0} \sum_n \max(0, 1 - y^{(n)}(w^\top x^{(n)} + w_0)) + \frac{1}{2\gamma} ||w||_2^2$

SVM vs. logistic regression

recall: **logistic regression** simplified cost for $y \in \{0,1\}$

$$L_{CE}(y,\hat{y}) = -y\log(\hat{y}) - (1-y)\log(1-\hat{y})$$

cross entropy loss

$$\hat{y} = \sigma(w^T x) = \frac{1}{1+e^{-w^T x}} = \frac{1}{1+e^{-z}} \Rightarrow log(\hat{y}) = -log(1+e^{-z})$$

$$1-\hat{y} = 1 - \frac{1}{1+e^{-z}} = \frac{e^{-z}}{1+e^{-z}} = \frac{1}{1+e^{z}} \Rightarrow log(1-\hat{y}) = -log(1+e^{z})$$

$$J(w) = \sum_{n=1}^{N} y^{(n)} \log\left(1+e^{-z^{(n)}}\right) + (1-y^{(n)}) \log\left(1+e^{z^{(n)}}\right)$$ where $z^{(n)} = w^\top x^{(n)}$

includes the bias

SVM vs. logistic regression

recall: **logistic regression** simplified cost for $y \in \{0,1\}$

$$J(w) = \sum_{n=1}^{N} y^{(n)} \log\left(1 + e^{-z^{(n)}}\right) + (1 - y^{(n)}) \log\left(1 + e^{z^{(n)}}\right) \quad \text{where} \quad z^{(n)} = w^\top x^{(n)}$$

includes the bias

for $y \in \{-1,+1\}$ we can write this as

$$J(w) = \sum_{n=1}^{N} \log\left(1 + e^{z^{(n)}}\right) \text{ for } y = +1$$

$$J(w) = \sum_{n=1}^{N} \log\left(1 + e^{-z^{(n)}}\right) \text{ for } y = -1$$

$$J(w) = \sum_{n=1}^{N} \log\left(1 + e^{-y^{(n)} z^{(n)}}\right)$$

$$J(w) = \sum_{n=1}^{N} \log\left(1 + e^{-y^{(n)} z^{(n)}}\right) + \frac{\lambda}{2}||w||_2^2$$

also added some regularization

compare to **SVM cost** for $y \in \{-1,+1\}$

$$J(w) = \sum_{n} \max(0, 1 - y^{(n)} z^{(n)}) + \frac{\lambda}{2}||w||_2^2$$

SVM vs. logistic regression

for $y \in \{-1, +1\}$ we can write this as

$$J(w) = \sum_{n=1}^{N} \log\left(1 + e^{-y^{(n)} z^{(n)}}\right) + \frac{\lambda}{2}||w||_2^2$$

also added some regularization

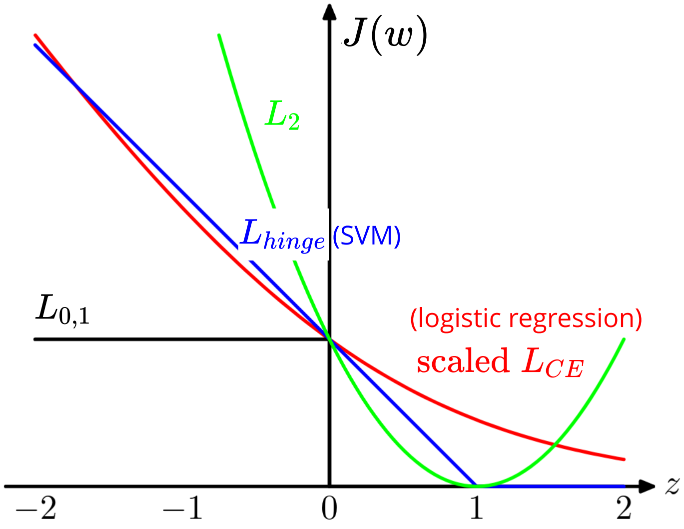
compare to **SVM cost** for $y \in \{-1, +1\}$

$$J(w) = \sum_n \max(0, 1 - y^{(n)}(z^{(n)})) + \frac{\lambda}{2}||w||_2^2$$

they both try to approximate 0-1 loss (accuracy)

$$L_{0-1}(y, w^T x) = \mathbb{I}(y = \text{sign}(w^T x))$$

Ideal loss function



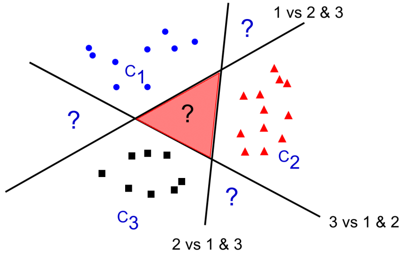
Multiclass classification

can we use multiple binary classifiders?

one versus the rest

training:

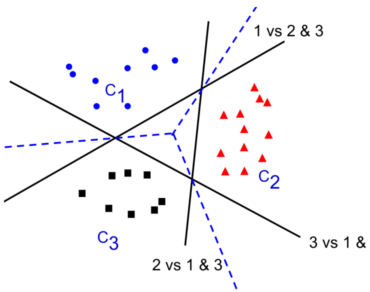
train C different 1-vs-(C-1) classifiers $y_c(x) = w_{[c]}^\top x$



test time:

choose the class with the highest score

$$c^* = \arg\max_c y_c(x)$$



problems:

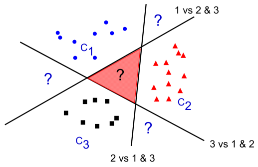
class imbalance

not clear what it means to compare $y_c(x)$ values

image credit: Andrew Zisserman

Multiclass classification

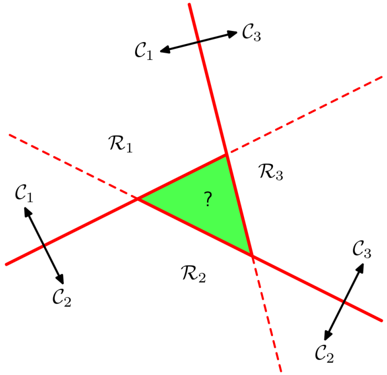
can we use multiple binary classifiders?



one versus one

training:
train $\frac{C(C-1)}{2}$ classifiers for each class pair

test time:
choose the class with the highest vote



problems:
computationally more demanding for large C
ambiguities in the final classification

Summary

- geometry of linear classification
- Perceptron algorithm
- distance to the decision boundary (margin)
- max-margin classification
- support vectors
- hard vs soft SVM
- relation to perceptron
- hinge loss and its relation to logistic regression
- some ideas for max-margin multi-class classification