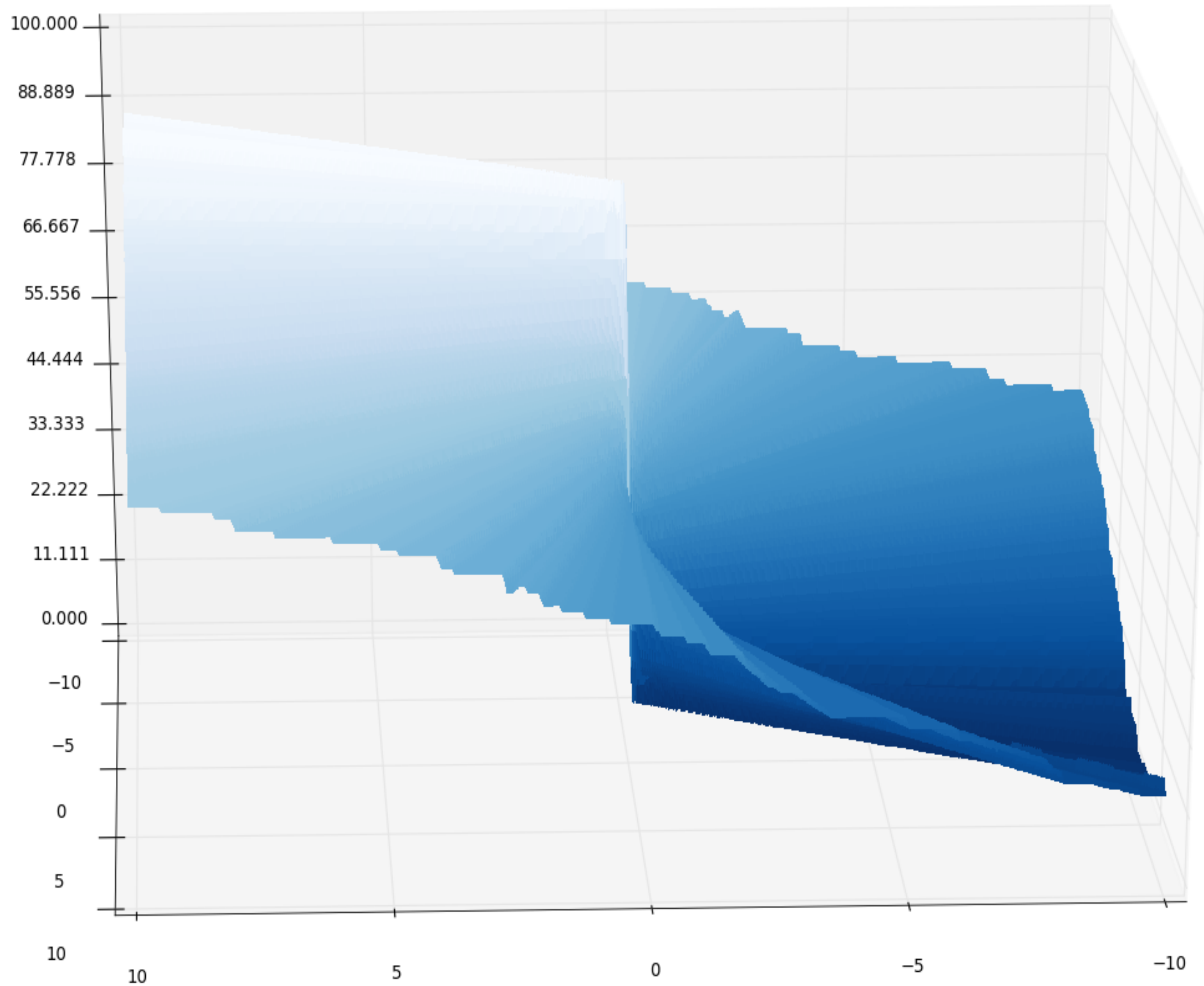
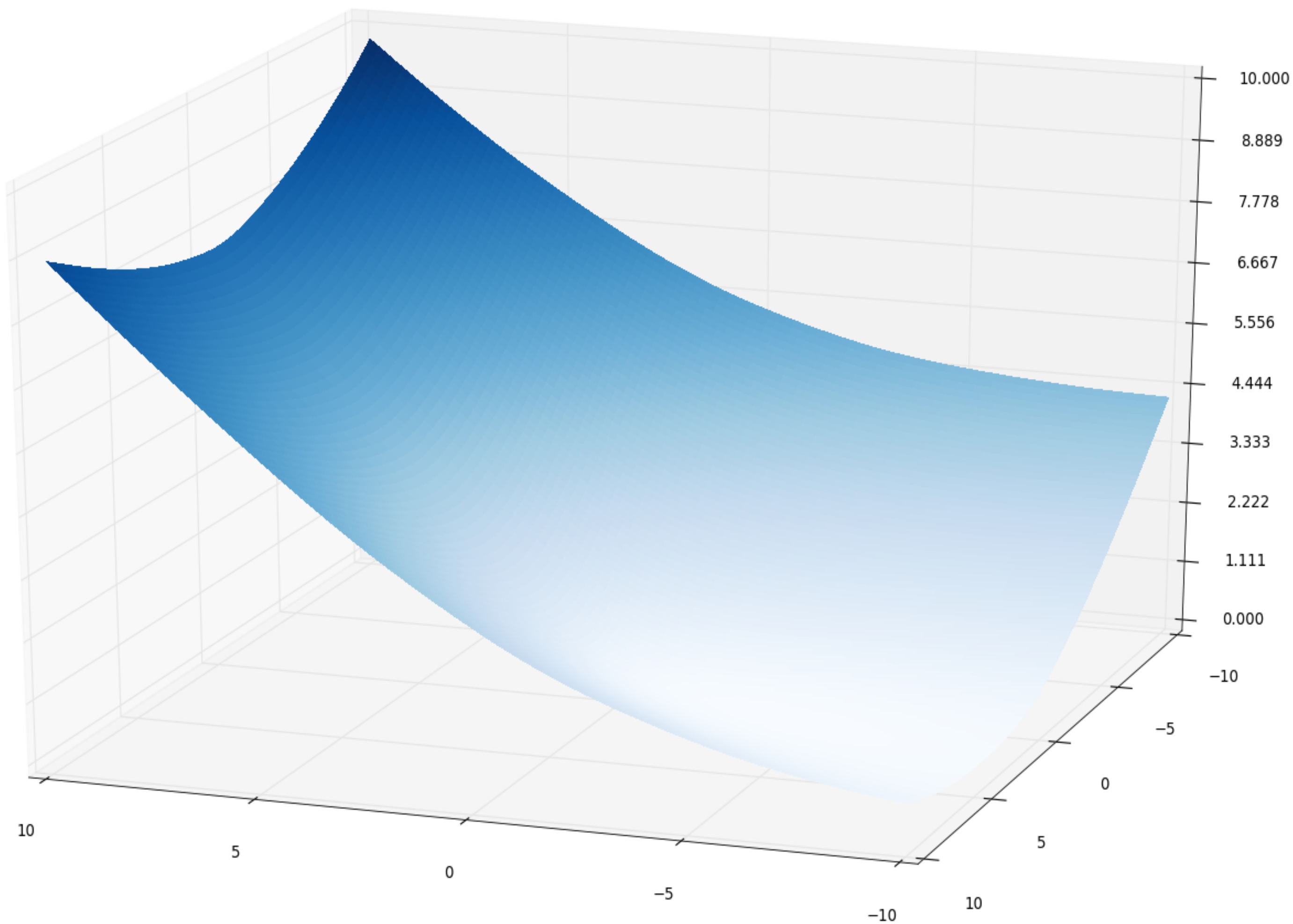
Intro to Machine Learning for Data Science

charles nainan

0-1 loss from our lab



logistic loss



surrogate loss

- Notice the 0-1 loss has lots of small flat places. Its is not convex, so using an general purpose optimizer will be very difficult.

- Logistic loss on the other hand is smooth and convex. Gradient based optimizers work well with logistic loss!

- When we do the lab on gradient descent this will make more sense.

- Because of this we use a "surrogate loss" for 0-1loss i.e even though we may be interested in 0-1 loss, we use a smooth convex loss like logistic because it can be solved with existing techniques.

loss functions

model	loss function
linear regression	Quadratic loss : $\sum \ (f (x) - y)^2$
logistic regression	logistic loss : $\sum \ \log (1 + e^{- f (x) \cdot y})$
SVM	Hinge loss : $\sum \ \max(0, \ 1 - f (x) \cdot y)$
decision tree(CART)	Information gain :
boosting	exponential loss : $\sum \ e^{- f(x) \cdot y}$

loss functions

All using $y \in \{-1,+1\}$

Perceptron

$$\mathbf{w} = \underset{w}{argmin} \sum_i Max(0\ ,\ -y_i <\mathbf{w},\mathbf{x}_i>)$$

$$gradient = \sum_{misclassified} y_i\, \mathbf{x}_i$$

Boosting

$$\mathbf{w} = \underset{w}{argmin} \sum_i exp(-y_i <\mathbf{w},\mathbf{x}_i>)$$

Logistic regression

$$\mathbf{w} = \underset{w}{argmin} \sum_i log(1 + exp(-y_i <\mathbf{w},\mathbf{x}_i>))$$

$$gradient = \sum_i \frac{y_i\, \mathbf{w}}{(1 + exp(-y_i <\mathbf{w},\mathbf{x}_i>)}$$

Quadratic loss

$$\mathbf{w} = \underset{w}{argmin} \sum_i (y_i - <\mathbf{w},\mathbf{x}_i>)^2$$

Svm : hinge loss

$$\mathbf{w} = \underset{w}{argmin} \sum_i max(0\ ,\ 1 - y_i <\mathbf{w},\mathbf{x}_i>)$$

linear models

- Predictions are based on the dot product of weight vector $\boldsymbol{w}$ and feature vector $\boldsymbol{x}$:

model weight vector

$$\boldsymbol{w} \times \boldsymbol{x} = \sum_{k=1}^{K} w_k \, x_k$$

feature vector

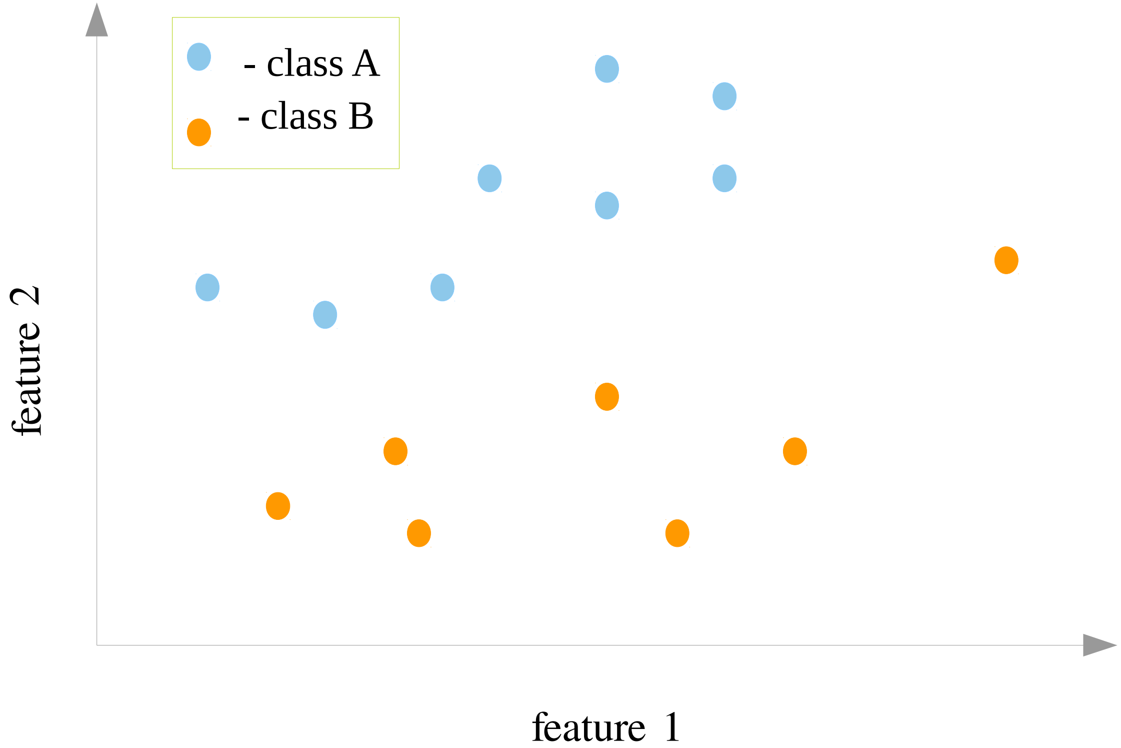
Sum over feature values

Linear Models for Classifiction

linear classification

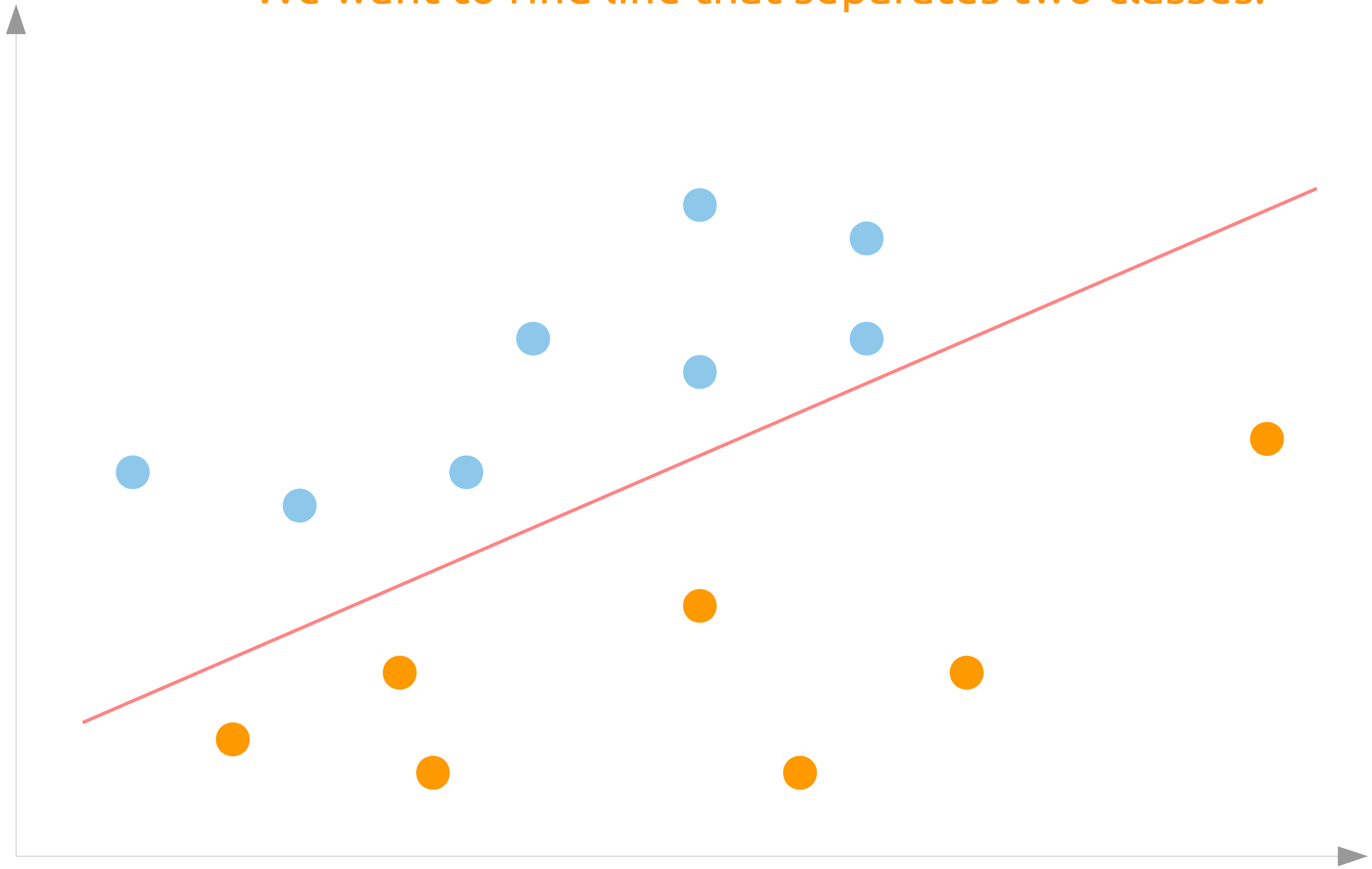
Binary classification problem where we want to predict the label:

feature1	feature2	label
3.44	2.66	A
1.47	-1.26	A
1.38	2.24	B
1.09	5.8	B
1.28	-3.22	B
2.43	6.72	A
3.16	0.04	A
.	.	.
.	.	.
.	.	.
2.13	-0.22	B



linear classification

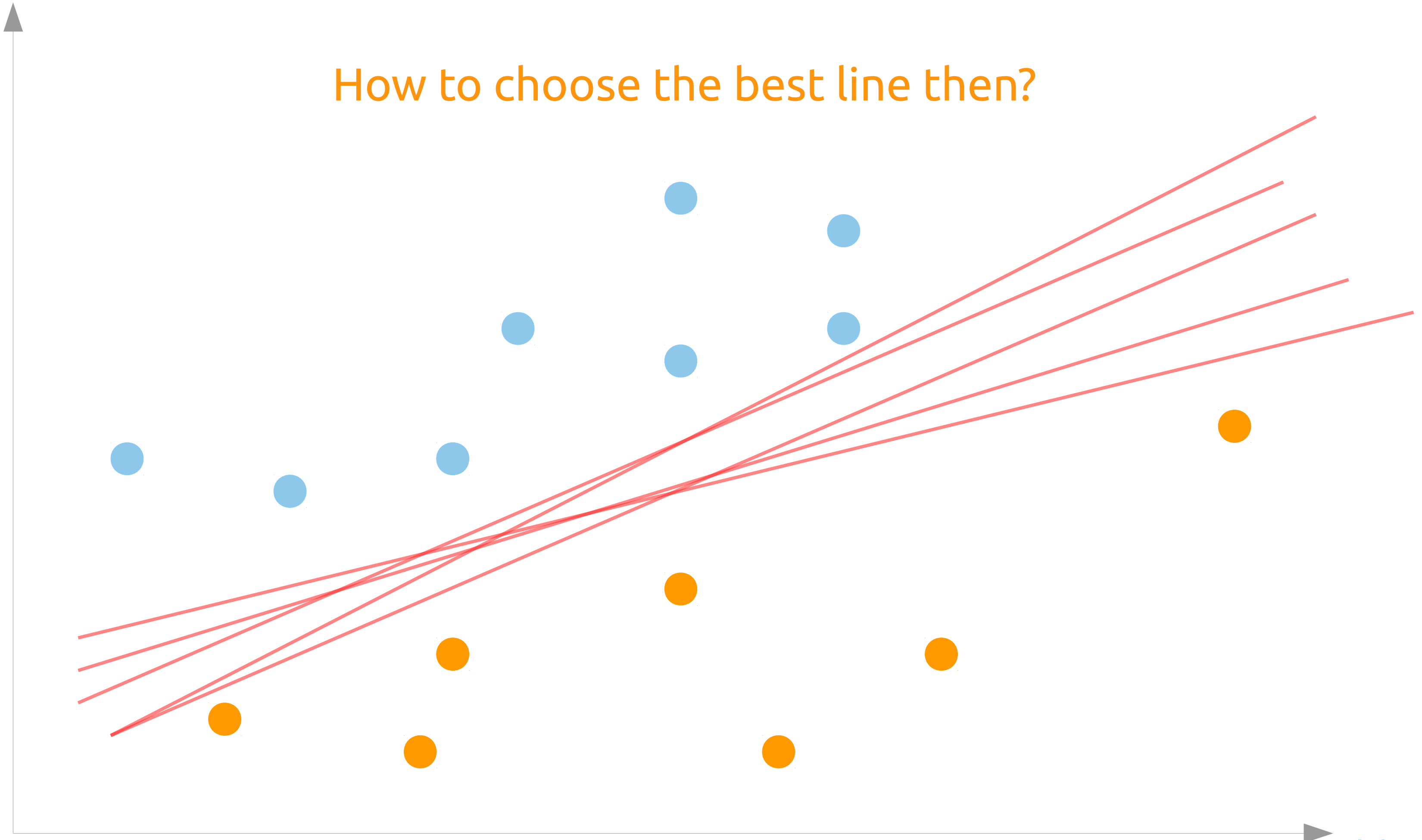
We want to find line that separates two classes:



linear classification

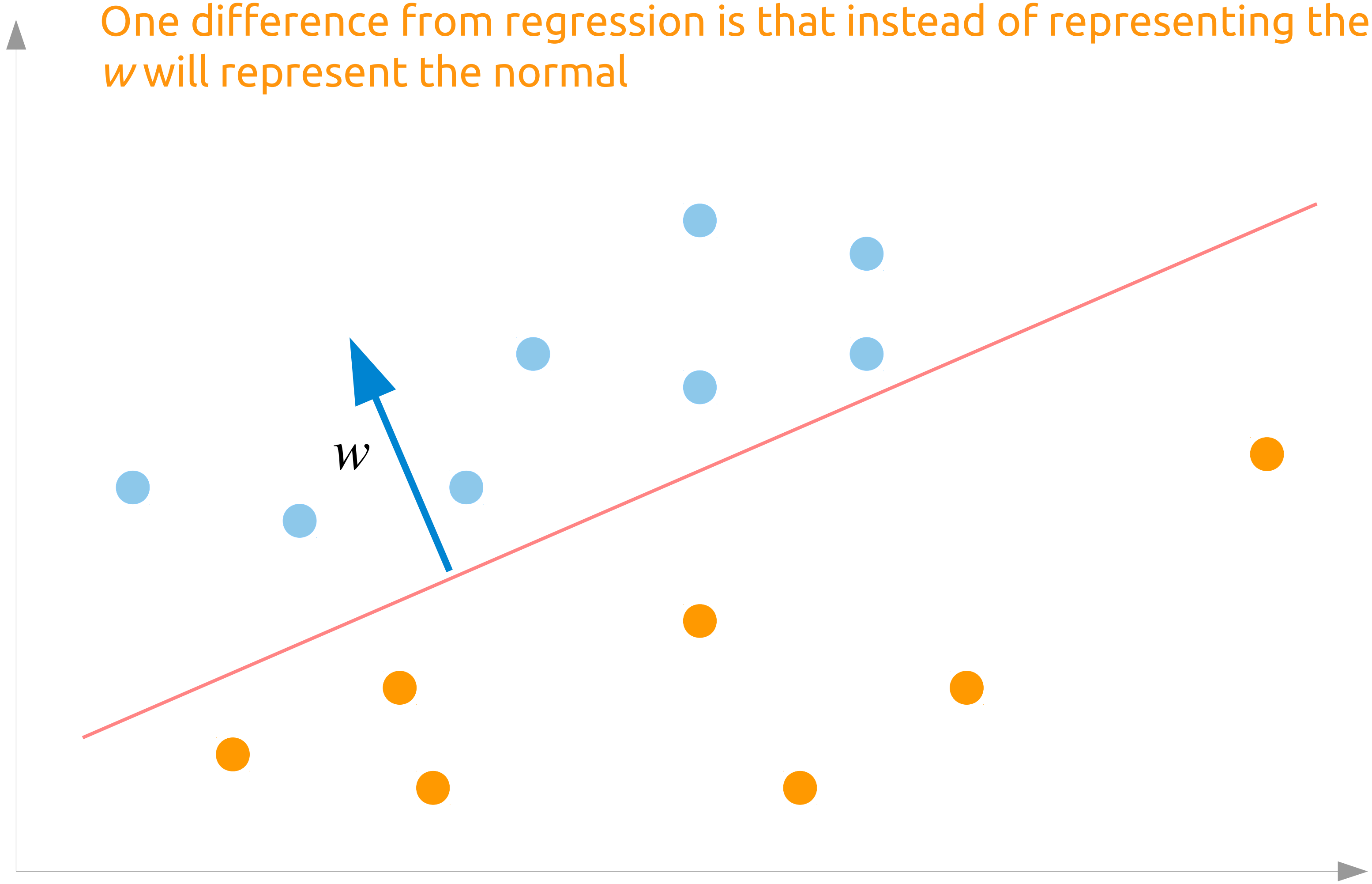
As in regression, there are lots of options.

How to choose the best line then?



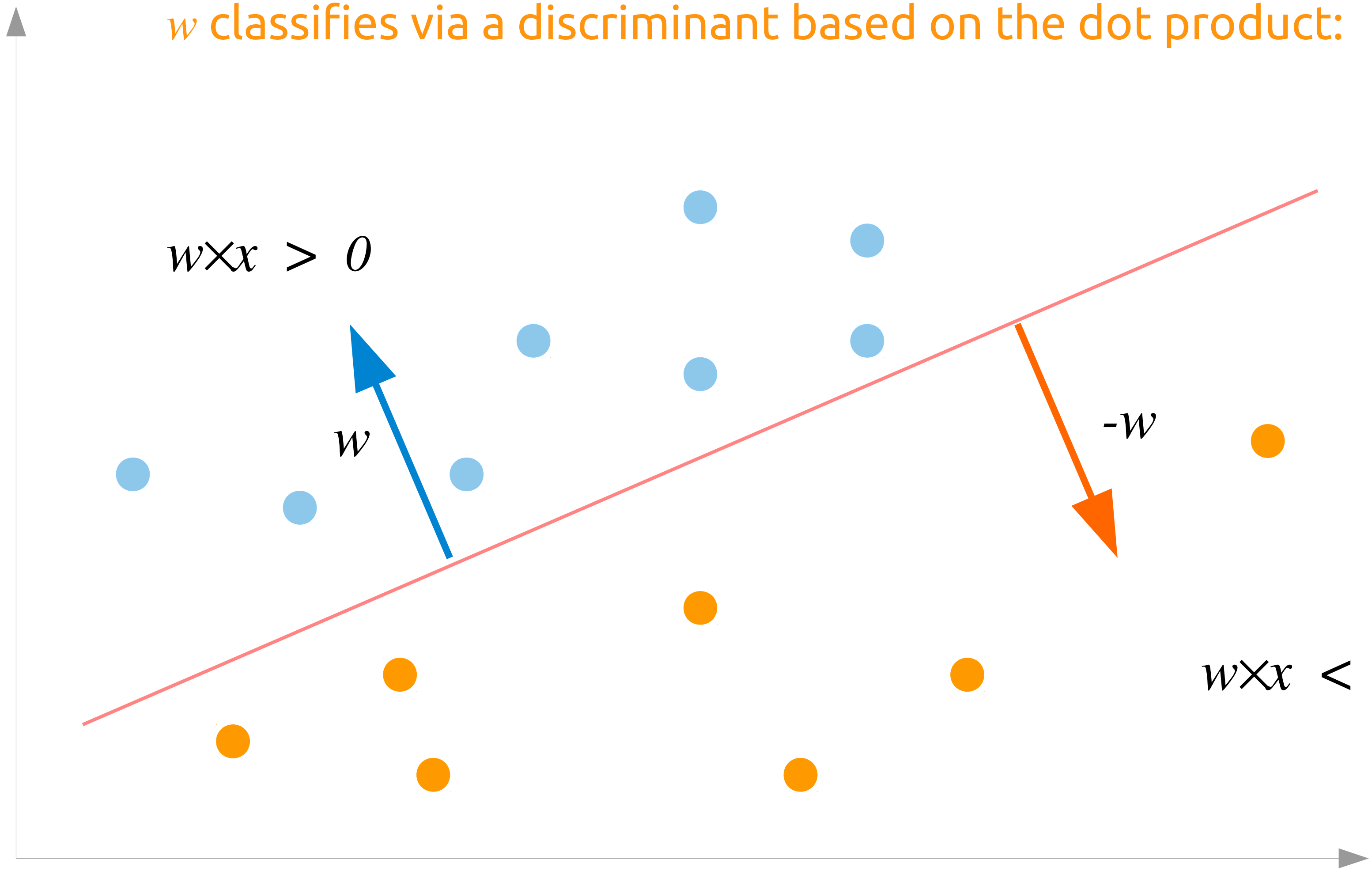
linear classification

One difference from regression is that instead of representing the line itself, *w* will represent the normal



linear classification

w classifies via a discriminant based on the dot product:



linear classification

- **Logistic regression** – estimates w via a conditional probabilistic model of classification : $p(y|x)$
- **Support vector machine(svm)** – uses the geometric notion of margin and estimates w by maximizing the margin
- **Perceptron** – estimates w via a simplified version of svm loss

Logistic Regression

logistic regression

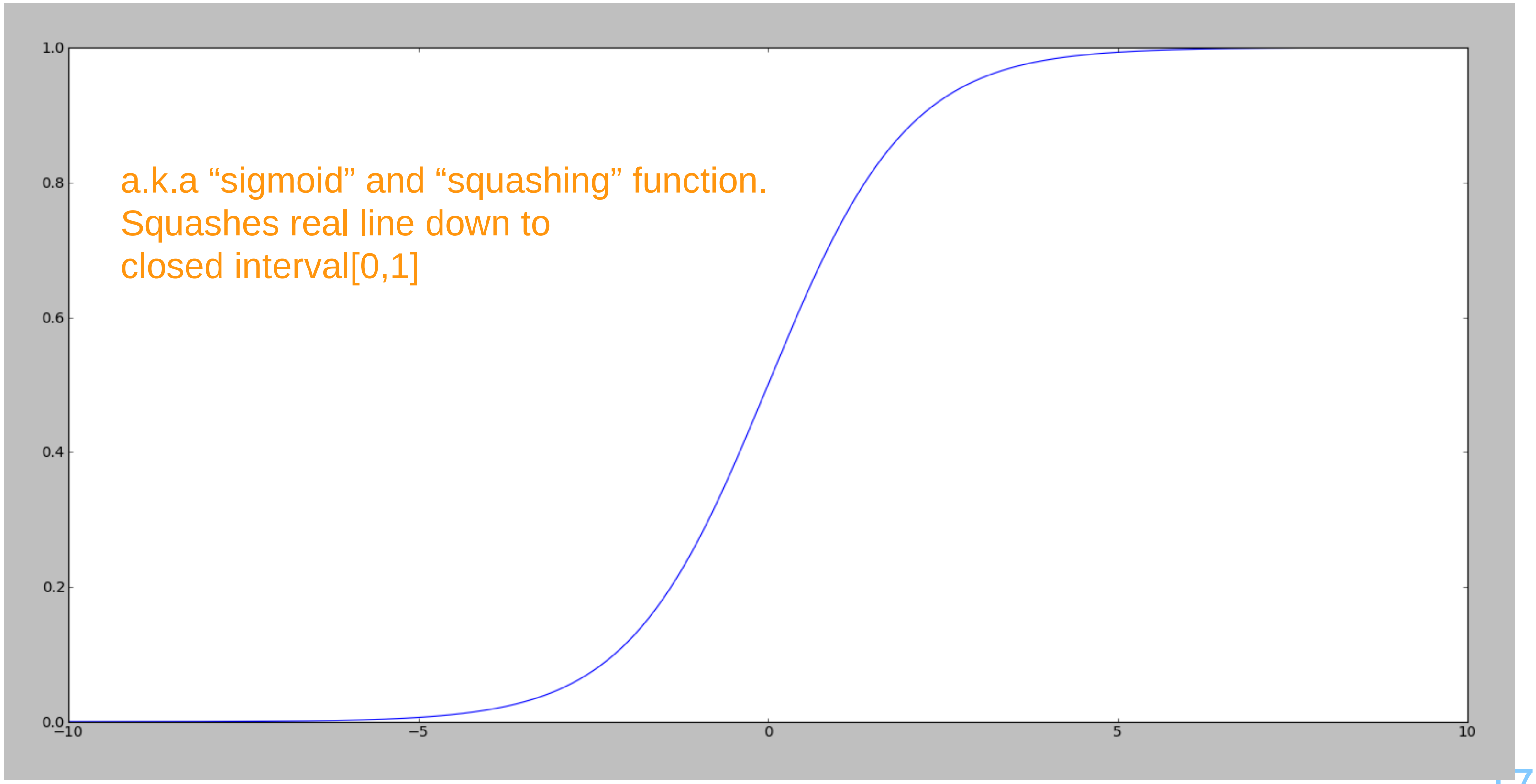
- As mentioned before, for classification we can not use $w \cdot x$ directly since it can occur anywhere on the real line and we need to discrete values.

- We would like to be able to have a probabilistic interpretation so we need to map $w \cdot x$ onto a probability value in [0,1]

- Enter the **logistic** function....

logistic function

$$logistic(x) = \frac{1}{1+e^{-x}}$$



logistic regression $y \in \{0,1\}$

$$P(y=1|\,x) = logistic(\,w \times x) \;=\; \frac{1}{1+e^{-w \cdot x}}$$

$$P(y=0|\,x) = 1 - P(y=1|\,x) \;=\; \frac{1}{1+e^{w \cdot x}}$$

$$likelihood = \prod_{i=1}^{N} (1 + e^{-wx_i})^{-y_i} (1 + e^{wx_i})^{-(1-y_i)}$$

$$-\log(likelihood) = \sum_{i=1}^{N} (y_i log(1 + e^{-wx_i}) + (1- y_i) log(1 + e^{wx_i}))$$

logistic regression $y \in \{0,1\}$

Loss(w) = -log(likelihood) + regularization

$$= \sum_{i=1}^{N} (y_i log(1 + e^{-wx}) + (1 - y_i) log(1 + e^{wx})) + \lambda \|w\|^2$$

Logistic regression {0,1}

$$\nabla \mathbf{Loss}(w) = \sum_{i=1}^{N} 2x_i\left(\, logistic(w \times x_i) - y_i\right) + 2\lambda w$$

$$= \sum_{i=1}^{N} 2x_i\left(\, \hat{y}_i - y_i\right) + 2\lambda w$$

$$\nabla \mathbf{Loss}(w) = 2(X^t(\hat{y} - y) + \lambda w)$$

$$\hat{y} = \text{logistic}(X \times w)$$

logistic vs linear regression

Logistic regression

$$\nabla \mathbf{Loss}(w) = 2(X^t(\hat{y} - y) + \lambda w)$$
$$\hat{y} = \mathit{logistic}(X \times w)$$

Linear regression

$$\nabla \mathbf{Loss}(w) = 2(X^t(\hat{y} - y) + \lambda w)$$
$$\hat{y} = X \times w$$

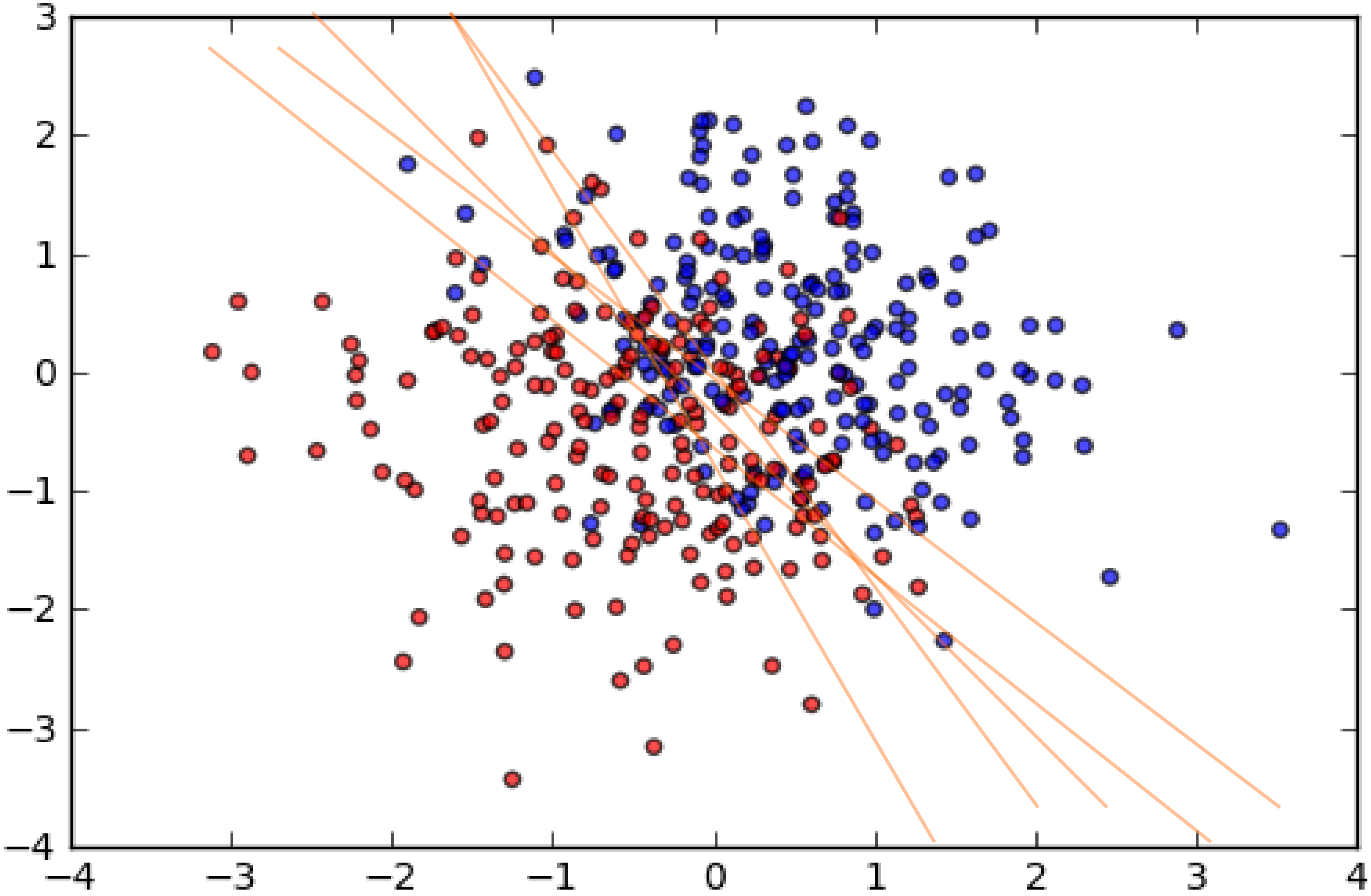
gradient descent

Now just as before, we can use gradient descent to estimate the model weights :

$$w = w - \eta \left(X^t(\hat{y} - y) + \lambda w\right)$$

example

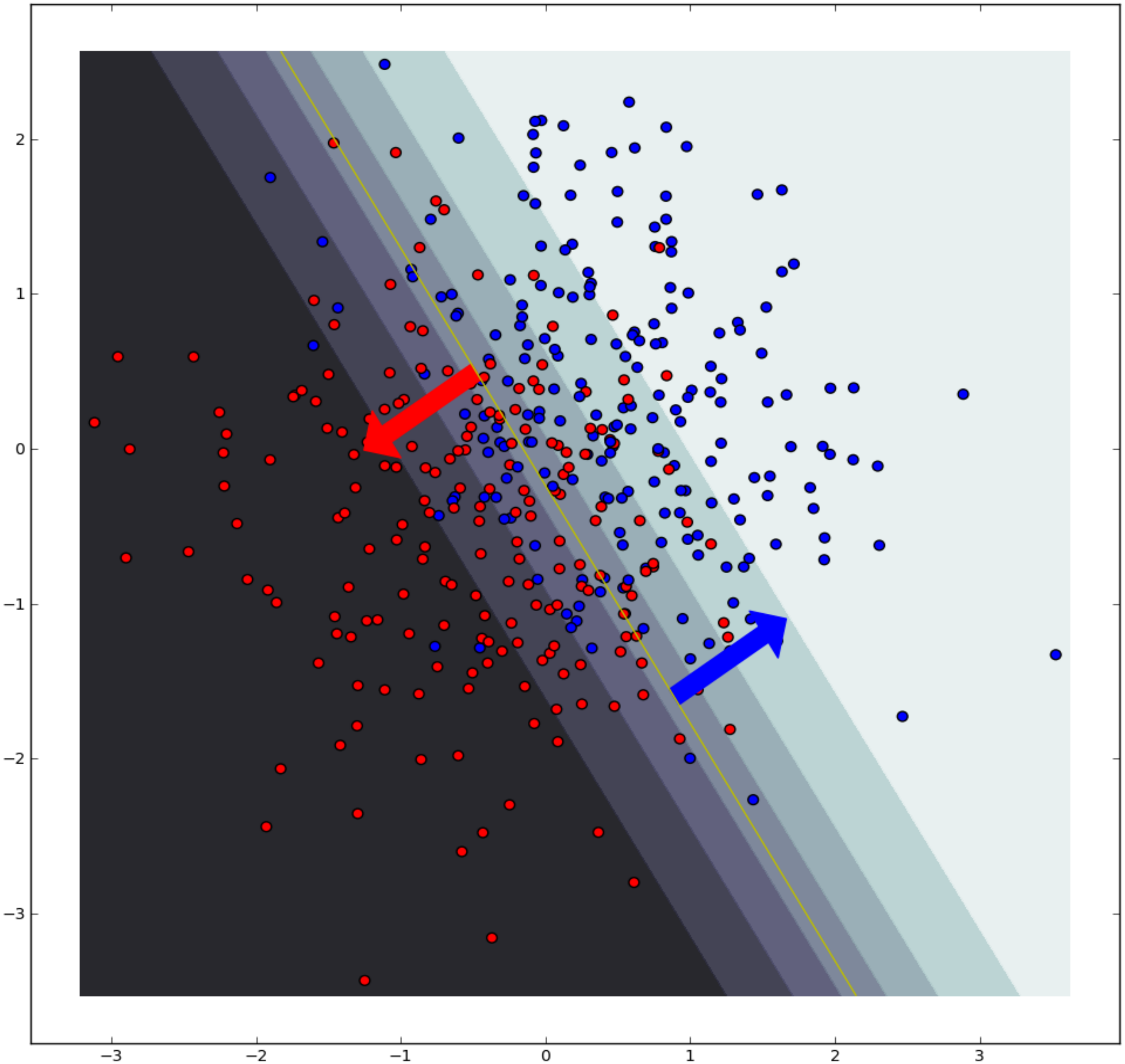
Lets use logistic regression to find a separating plane for this synthetic data set:



simplified logistic regression

```
def logreg_cost(w,X,y,C=0.1):
    norm = C*0.5*(w.dot(w))
    Xw =X.dot(w)
    nll =  ( y.dot(np.logaddexp(0,-Xw))  + (1.0-y).dot(np.logaddexp(0,Xw)))
    return nll + norm

def logreg_gradient(w,X,y,C=0.1):
    p   = logistic(X.dot(w))
    Xt  = X.T
    res = Xt.dot(p - y)
    return res + C*w

C = 1.0
cost     = lambda w: logreg_cost(w,X_train,y_train,C)
gradient = lambda w: logreg_gradient(w,X_train, y_train ,C)

N, M   = X_train.shape
w     = np.zeros(M)
print "initial cost: ", cost(w)

iters = 0
MAX_ITERS = 10
step   = 1.0

# gradient descent
while (iters < MAX_ITERS):
    iters += 1
    w = w - step * gradient(w)
```

logistic regression solution



logistic regression $y \in \{-1,+1\}$

$$P(y = 1 | x) = logistic(w \times x) = \frac{1}{1+e^{-w \cdot x}}$$

$$P(y = -1 | x) = 1 - P(y = 1 | x) = \frac{1}{1+e^{w \cdot x}}$$

$$\boxed{P(y | x) = logistic(y\, w \times x) = \frac{1}{1+e^{-y\, w \cdot x}}}$$

logistic regression $y \in \{-1,+1\}$

$$P(y \mid x) = logistic(\, y\, w \times x) = \frac{1}{1+e^{-y\, w \cdot x}}$$

$$likelihood = \prod_{i=1}^{N} (1 + e^{-y_i w x})^{-1}$$

$$-\log(likelihood) = \sum_{i=1}^{N} (log(1 + e^{-y_i w x}))$$

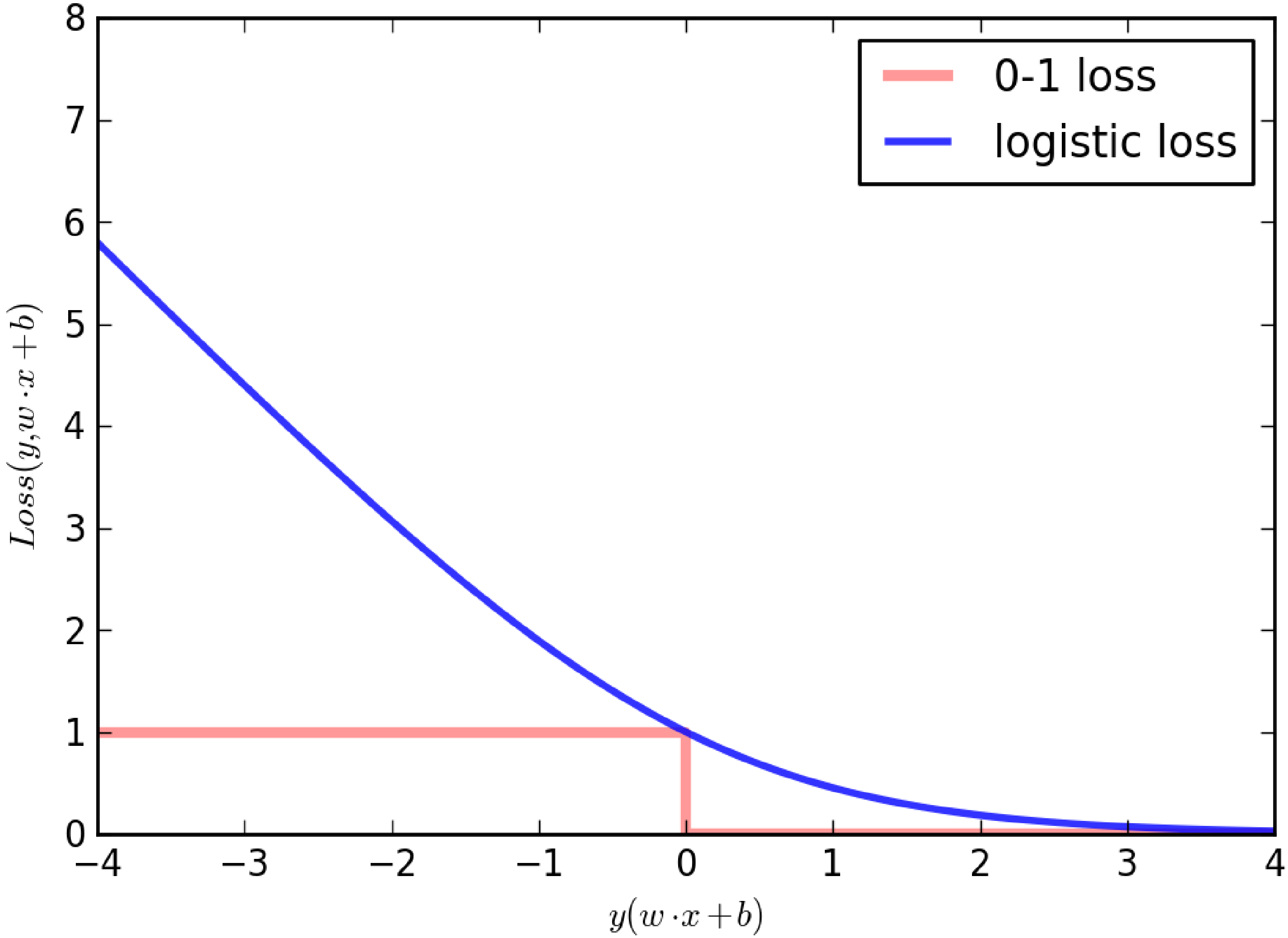
logistic regression $y \in \{-1,1\}$

$$\textbf{Loss}(w) = \sum_{i=1}^{N} (log(1 + e^{-y_i wx}) + \lambda \|w\|^2$$

$$\nabla\textbf{Loss}(w) = \sum_{i=1}^{N} \frac{e^{-y_i wx}}{(1 + e^{-y_i wx})} y_i x_i + 2\lambda w$$

$$\nabla^2\textbf{Loss}(w) = \sum_{i=1}^{N} \frac{e^{-y_i wx}}{(1 + e^{-y_i wx})^2} x_i x_i^{\mathrm{T}}$$

logistic loss



Support Vector Machines(SVM)

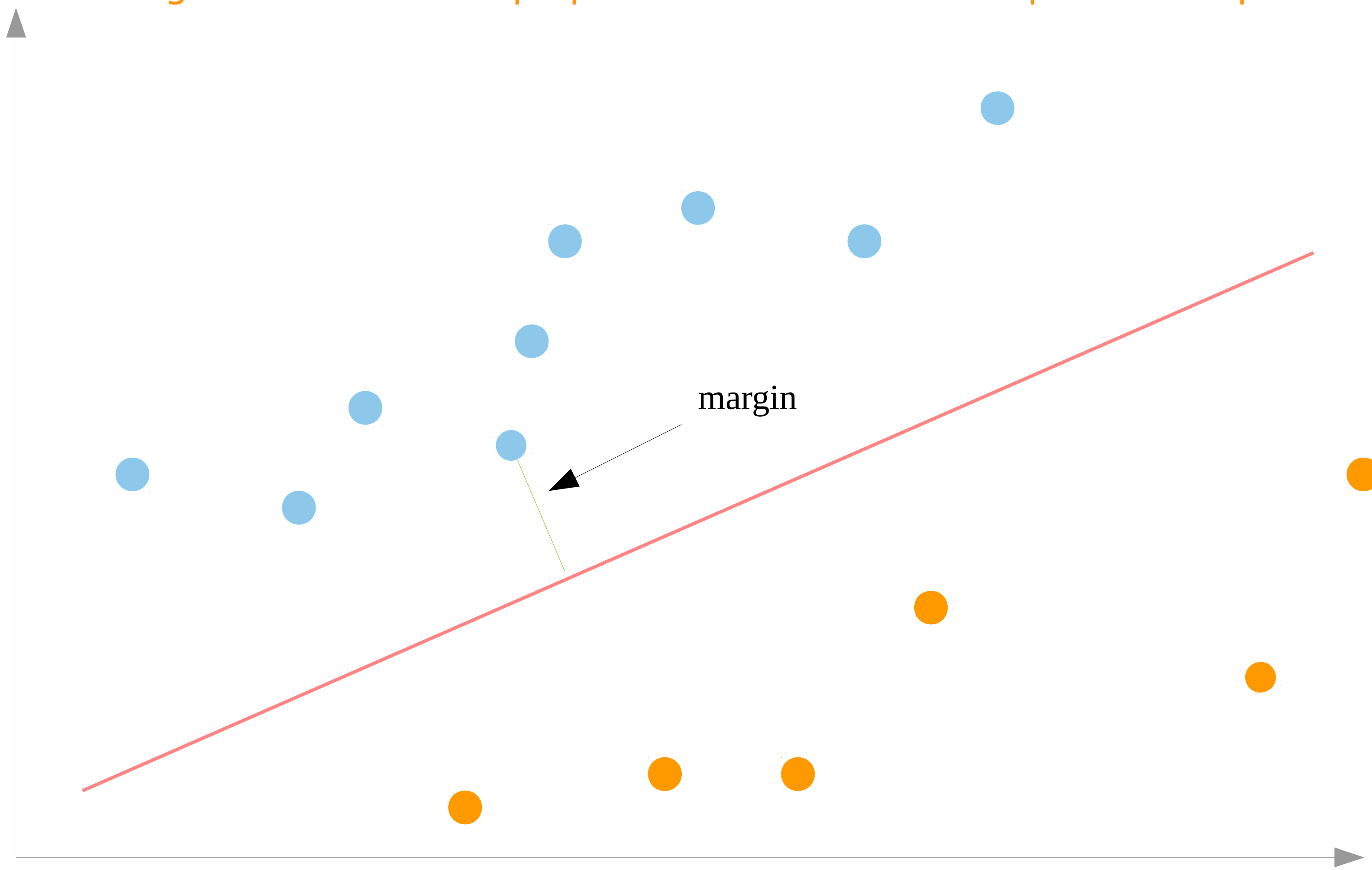
support vector machines(svm)

- SVMs came about in the mid 1990s :
  - Based on the work of Vapnik and Chernovikas in the 70s and 80s
  - Linear classifier based on hinge loss, but non-linear extension based on theory of Reproducing Kernel Hilbert Space (RKHS) – a.k.a the "kernel trick"
  - Structural risk minimization(SRM) supported by a sophisticated theory of statistical learning with error bounds measured by the VC dimension.
  - Because of the ease of use and strong performance numbers, they displaced the dominant ml paradigm of the 80s and early 90s : feed-forward neural networks

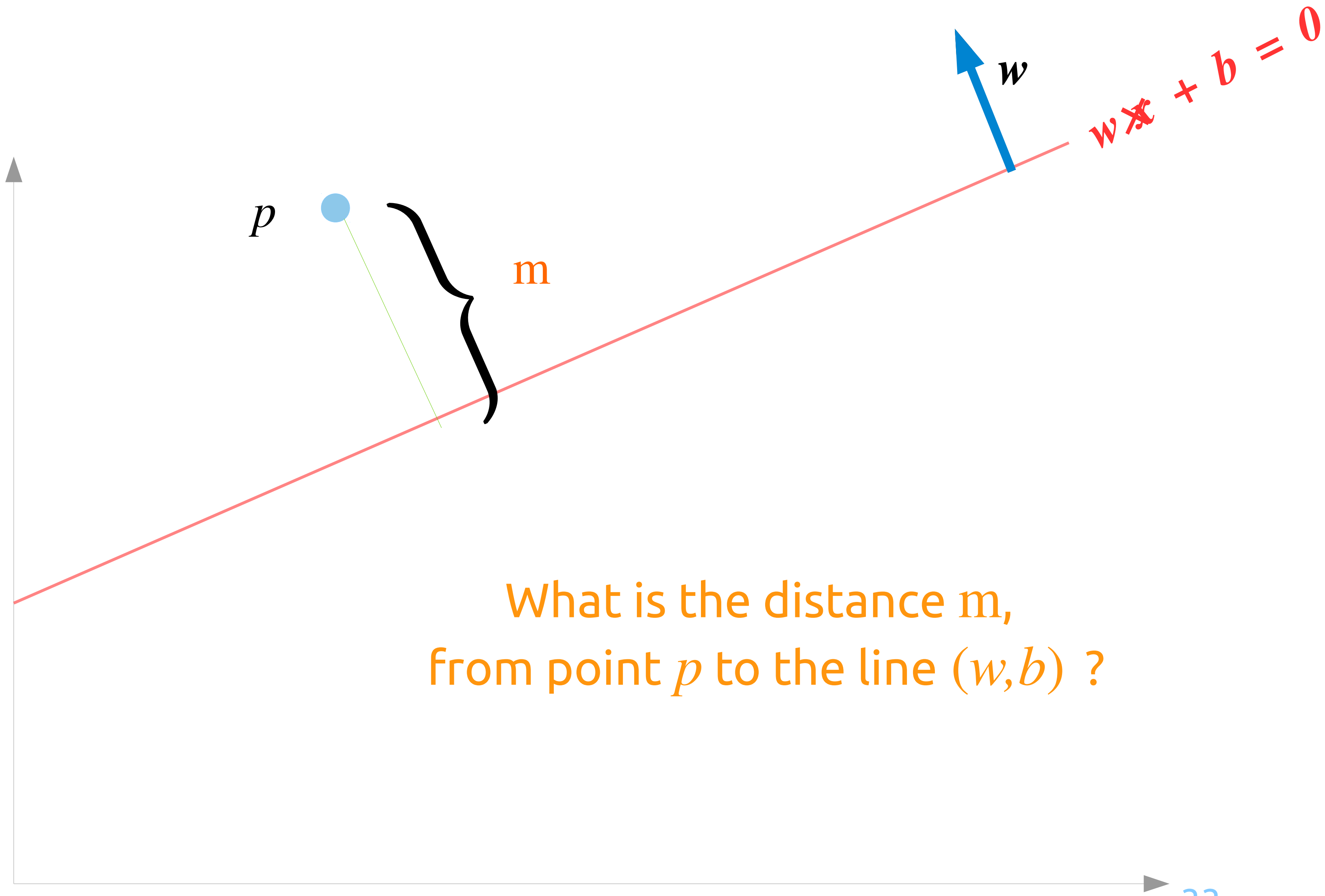
- Basic idea is to find the best separating hyperplane by maximizing the geometric **margin**.

margin

Margin is defined as the perpendicular distance from a point to the plane:

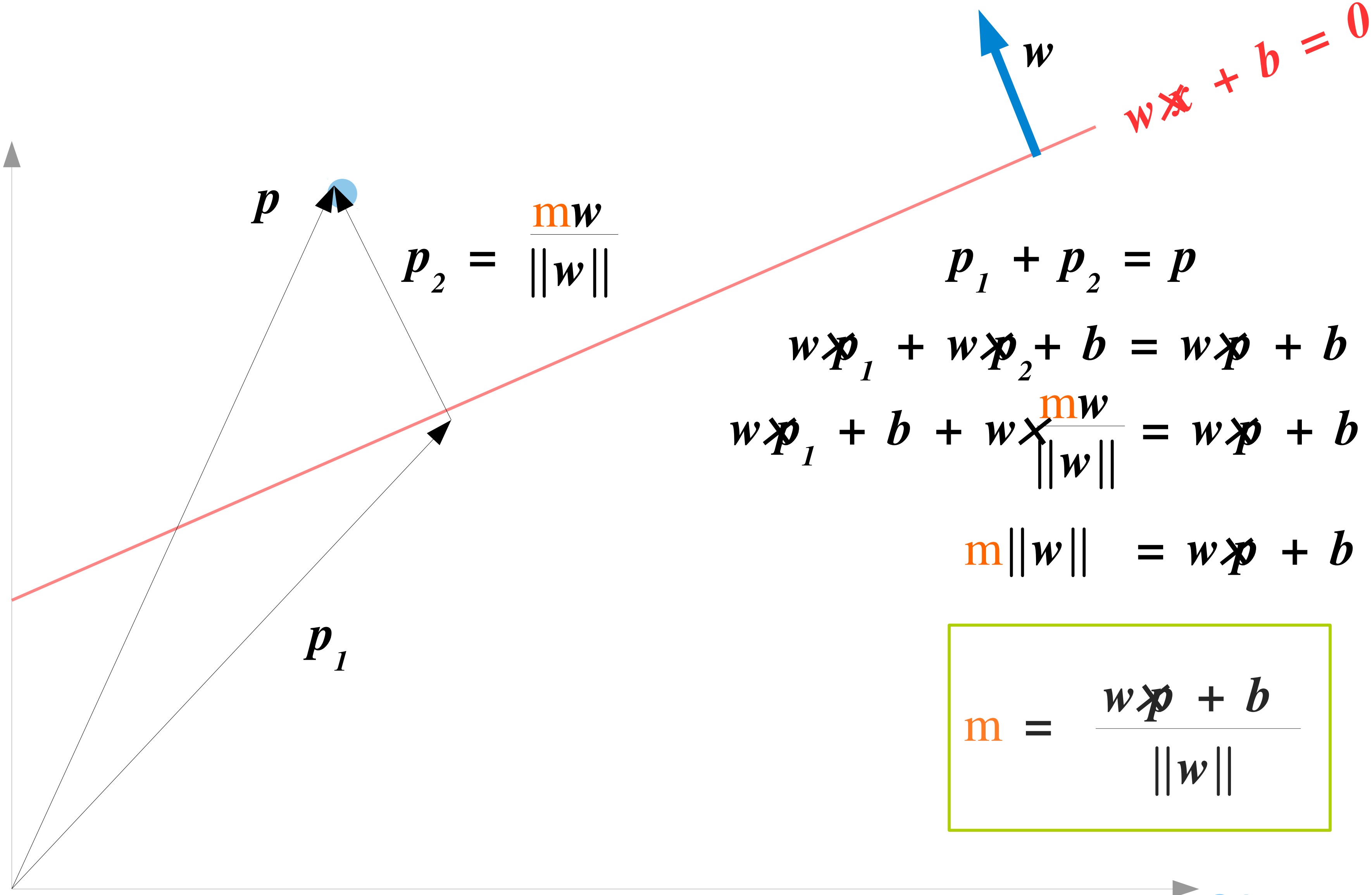


perpendicular distance to line



What is the distance m,
from point p to the line (w, b) ?

distance to line



$$p_1 + p_2 = p$$

$$wp_1 + wp_2 + b = wp + b$$

$$wp_1 + b + w\frac{mw}{||w||} = wp + b$$

$$m||w|| = wp + b$$

$$\boxed{m = \frac{wp + b}{||w||}}$$

svm

- Notice that m will not change if we rescale:

$$(w,b) \Rightarrow (\alpha w, \alpha b)$$

- So we may choose a scaling such that:

$$w\cdot x + b = 1 \text{ at the margin hyperplane}$$

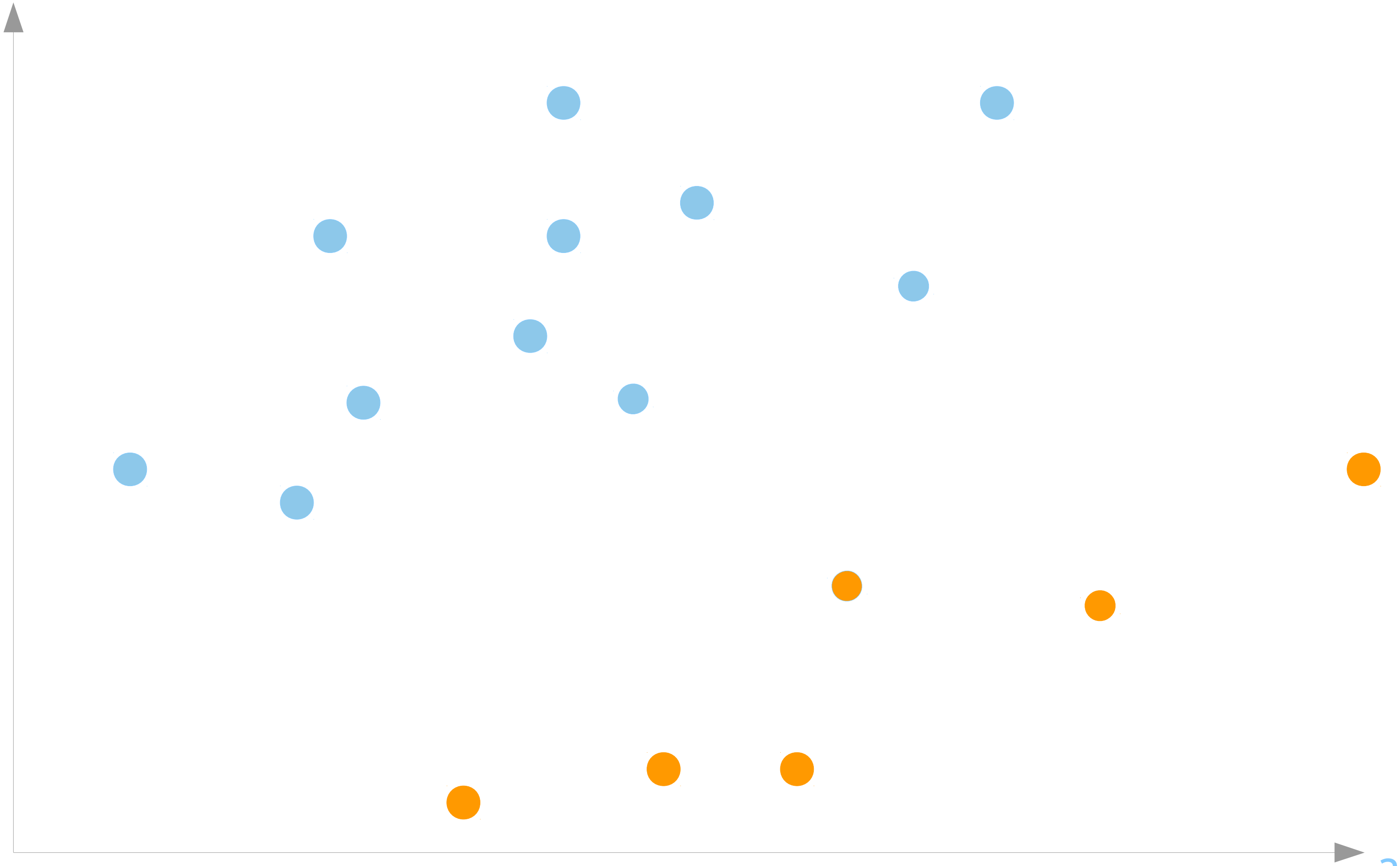
- Then:

$$m = \frac{1}{||w||}$$

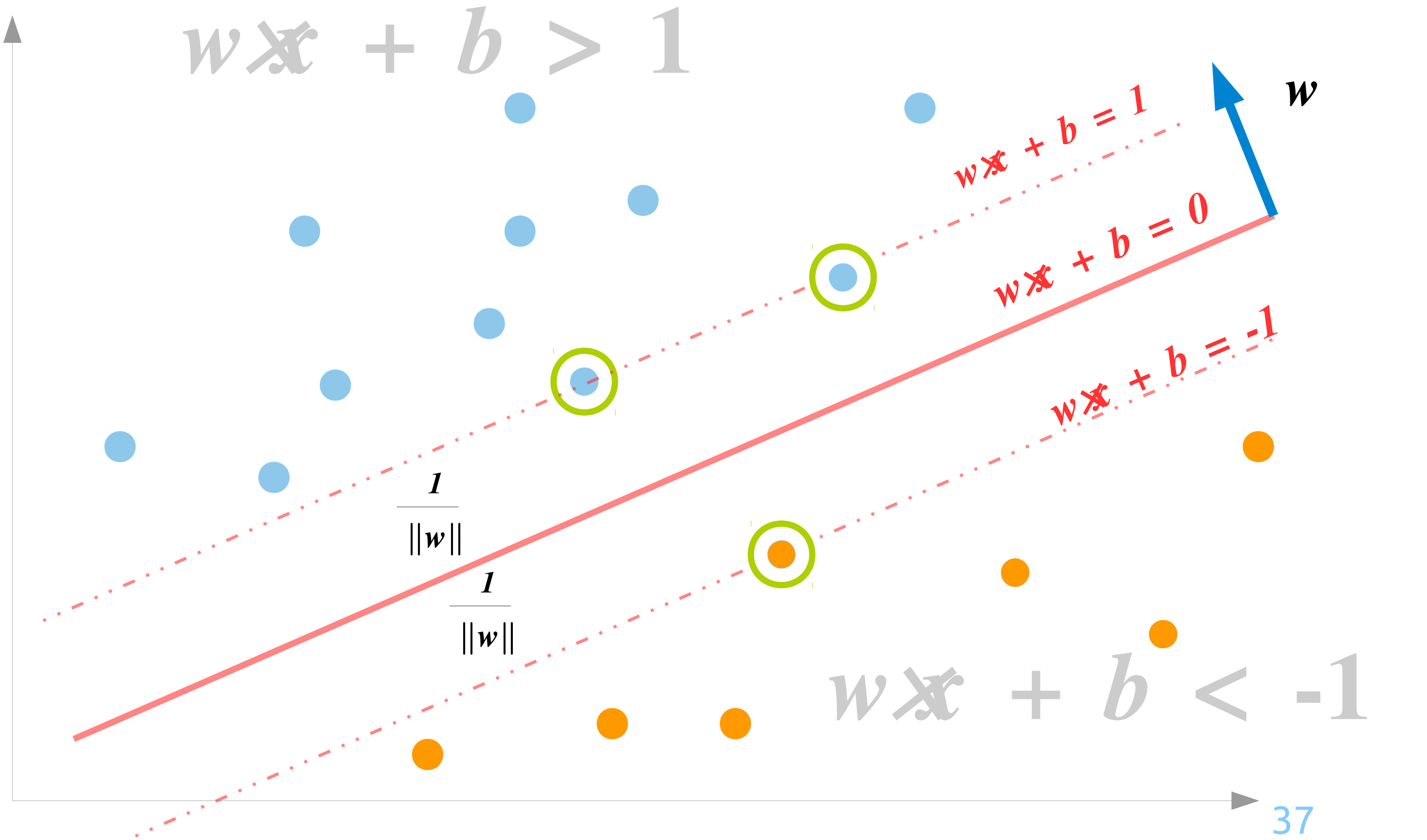
- Maximizing margin for *separable* data $y \in \{-1,+1\}$ means:

$$\boxed{\textit{minimize} \quad ||w||^2 \quad \text{with} \quad y(w\cdot x + b) > 1}$$

svm: separable data



support vectors



svm: inseparable data

- In general, a perfect separating hyperplane will not exist. We will have points x_i where

$$y_i(w x_i + b) < 1$$

- So we introduce slack variables :

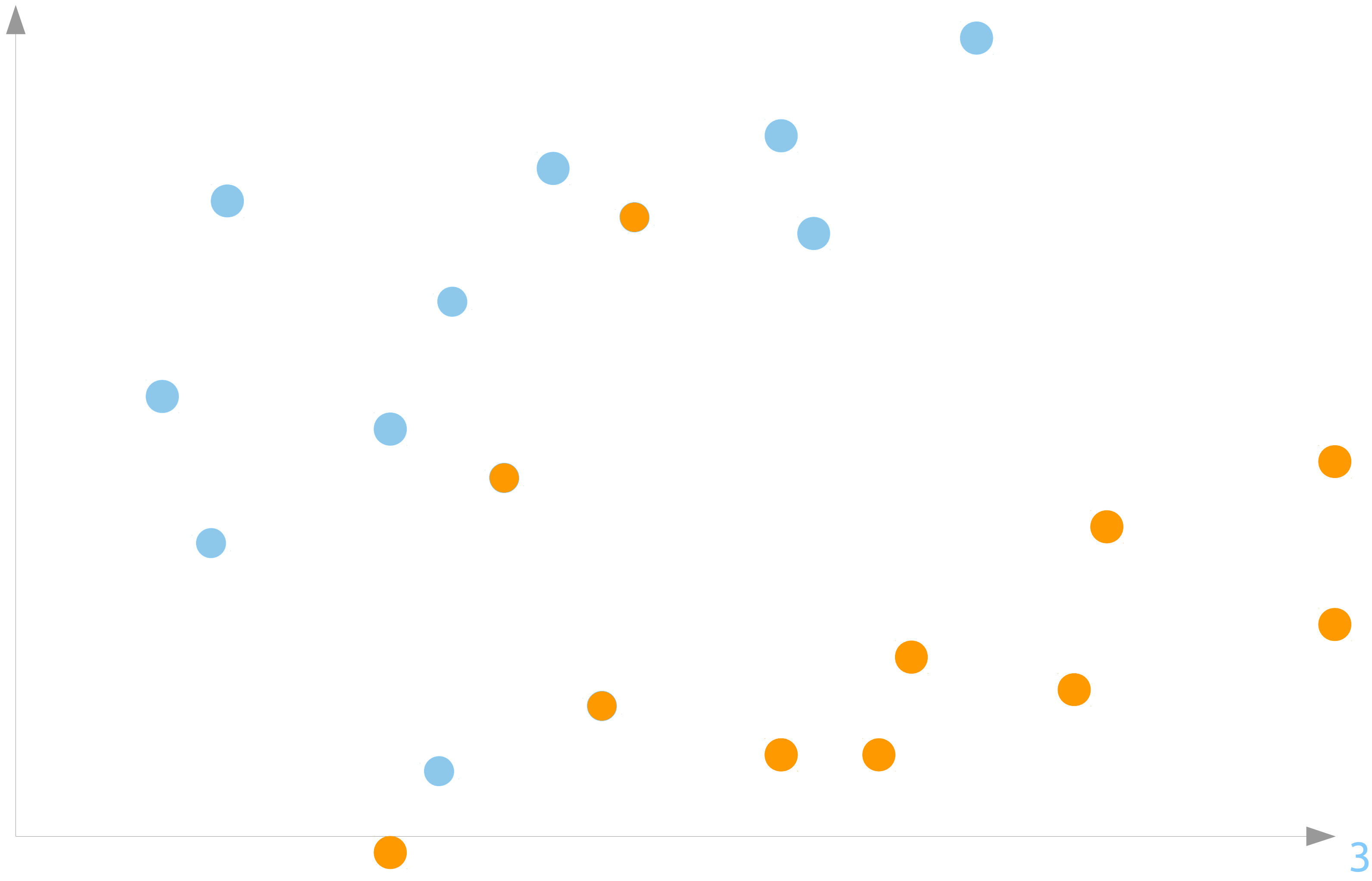
$$\beta_i = 1 - y_i(w x_i + b) > 0$$

- Then we want to minimize:

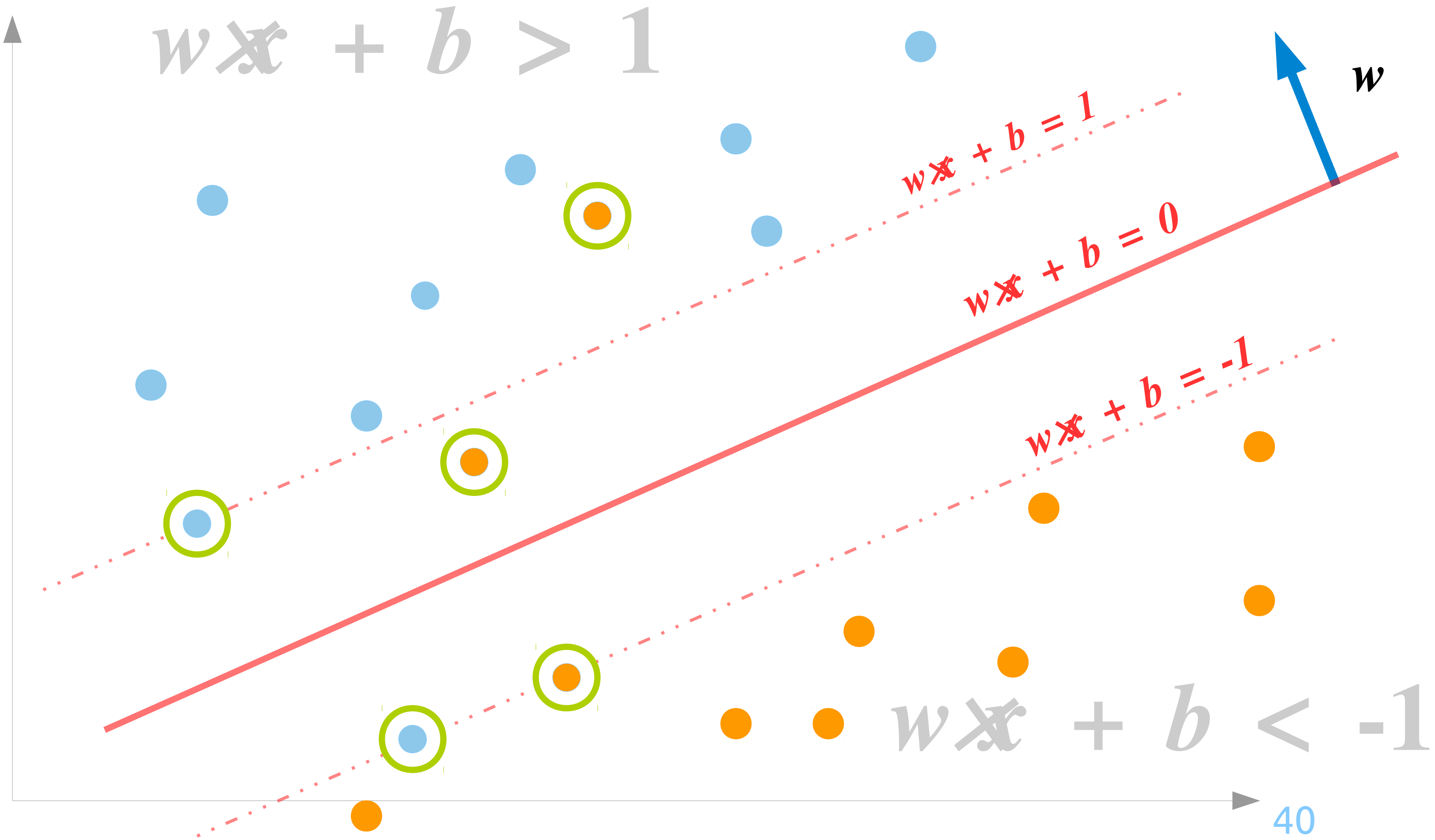
$$minimize \quad \|w\|^2 + C \sum_{y_i(w x_i + b) < 1} \beta_i$$

$$\boxed{minimize \quad \|w\|^2 + C \sum \max(0, 1 - y_i(w x_i + b))}$$

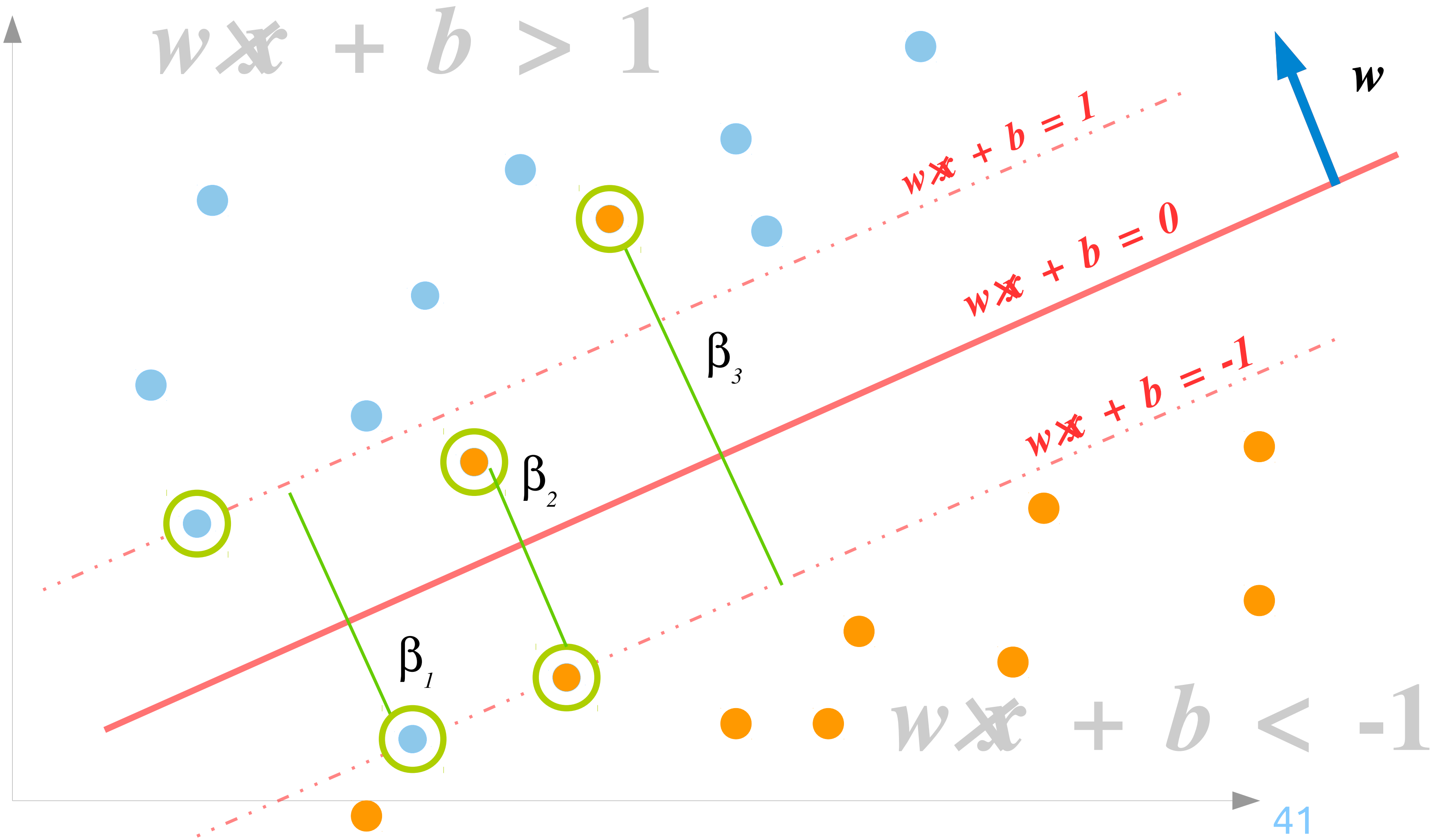
svm: inseparable data



support vectors



svm: inseparable data



svm: hinge loss

$$\textit{minimize} \quad \|w\|^2 + C \sum \max(0, 1 - y_i(w x_i + b))$$

margin becomes regularization

slack becomes loss term: **hinge loss**

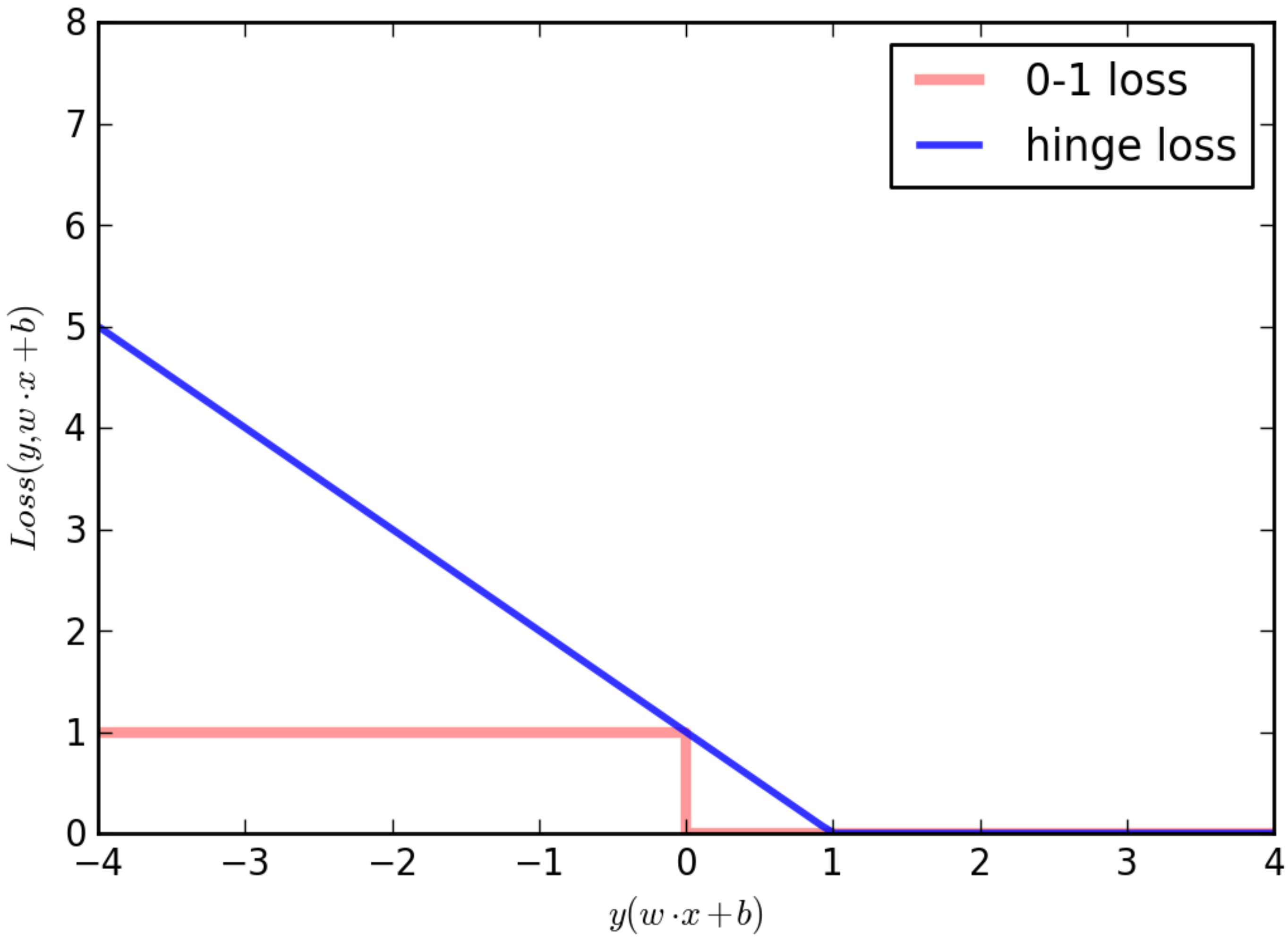
hinge loss and subgradient

$$\mathbf{Loss}(\mathrm{w}) = \|w\|^2 + C\sum \max(0, 1 - y_i(w x_i + b))$$

$$\nabla\mathbf{Loss}(\mathrm{w}) = 2w - C\sum_{y_i(w x_i + b) < 1} y_i x_i$$

Not a true gradient : actually a **subgradient**
because Loss is not everywhere differentiable!!

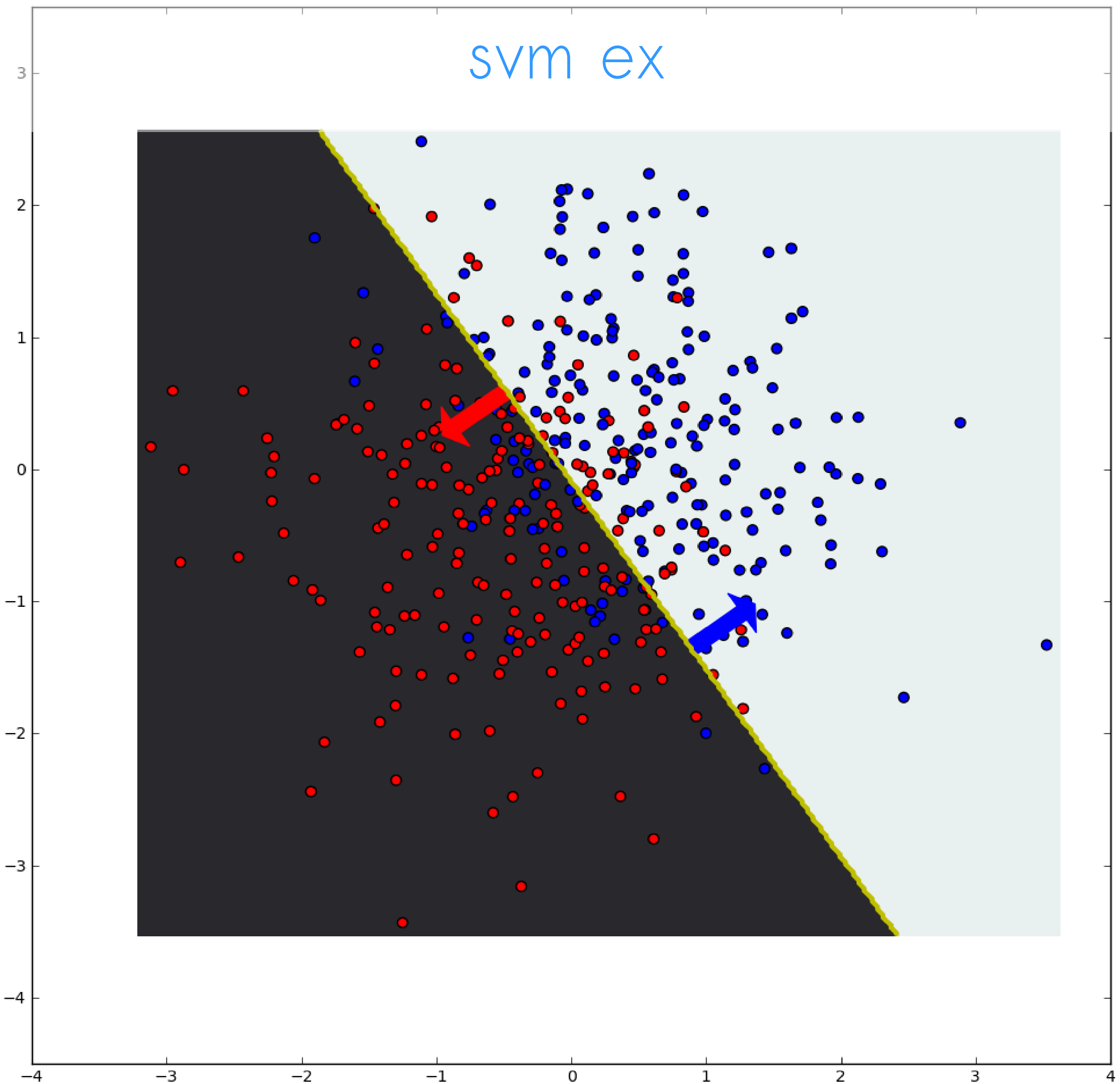
hinge loss

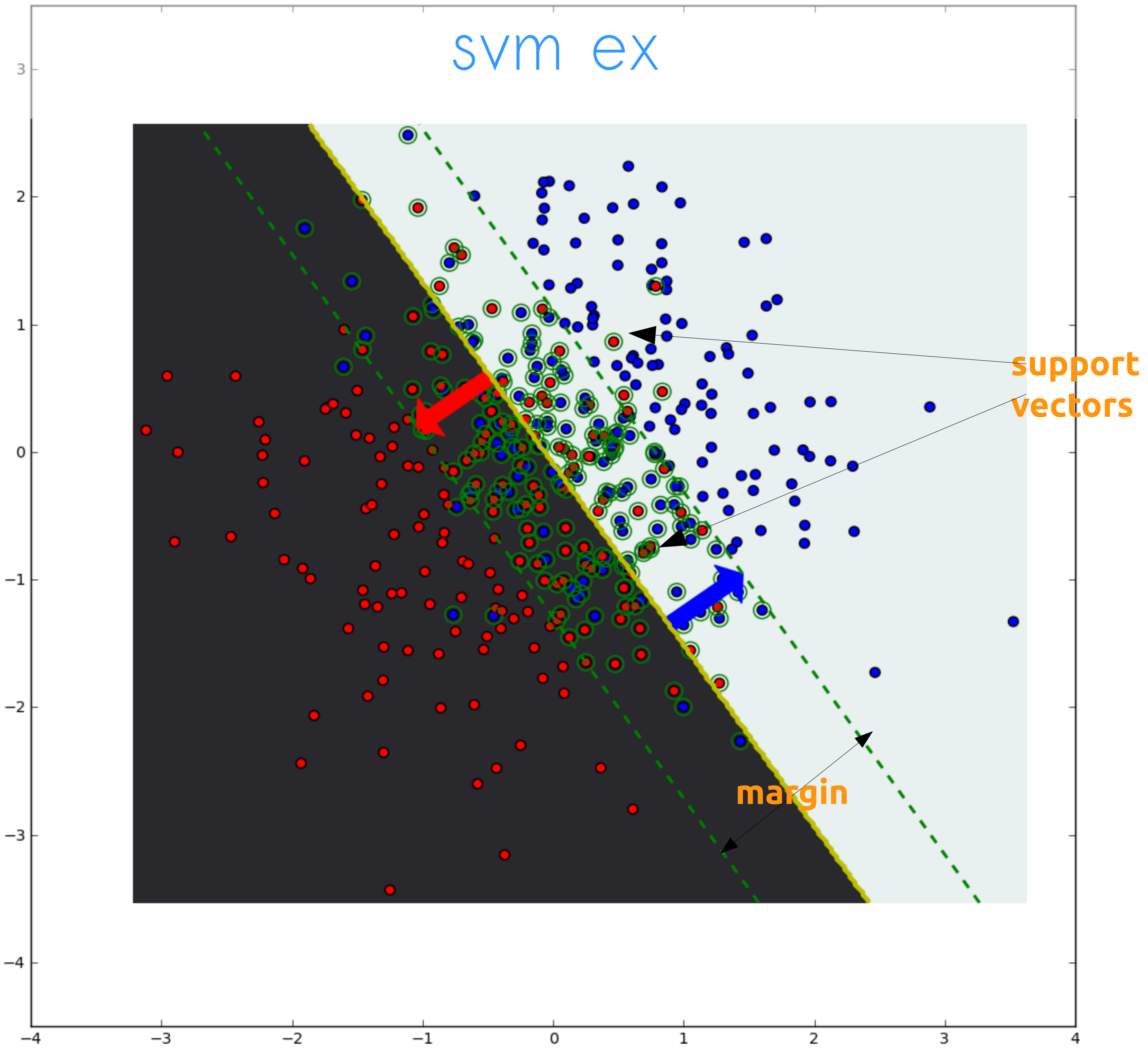


0-1 loss
hinge loss
$Loss(y, w \cdot x + b)$
$y(w \cdot x + b)$
8
7
6
5
4
3
2
1
0
−4
−3
−2
−1
0
1
2
3
4

svm: (primal) subgradient descent

```
def svm_cost(weights,XData,yData,C):
    margin = yData * (np.dot(XData, weights) )
    indxs = np.where(1.0 - margin > 0)
    m = margin[indxs]
    loss = ( 1 - m)
    l2_cost = 0.5 * np.dot(weights[1:3], weights[1:3])
    loss = C*np.sum(loss) + l2_cost
    return loss


def svm_gradient(weights,XData,yData,C):
    w = weights.copy()
    margin = yData * (np.dot(XData, w) )
    indxs = np.where(1.0 - margin > 0)
    X = XData[indxs]
    y = yData[indxs]
    m = margin[indxs]
    w[0] = 0.0
    return w - C*np.dot( X.T,y )

def predict_svm(w, X):
    return np.array([1.0 if p > 0. else 0.0 for p in X.dot(w)])


N, M = X_train.shape
MAX_ITERS  = 10
w = np.zeros(M)
STEP_SIZE = 1.0
C         = .1

cost_func = lambda w: svm_cost(w,X_train,y_train_svm,C)
grad_func = lambda w: svm_gradient(w,X_train, y_train_svm ,C)

while (iters < MAX_ITERS):
    iters += 1
    w = w - step * grad_func(w)  # sub-gradient descent
```


svm ex
3
2
1
0
−1
−2
−3
−4
−4
−3
−2
−1
0
1
2
3
4


svm ex
support
vectors
margin

perceptron

perceptron

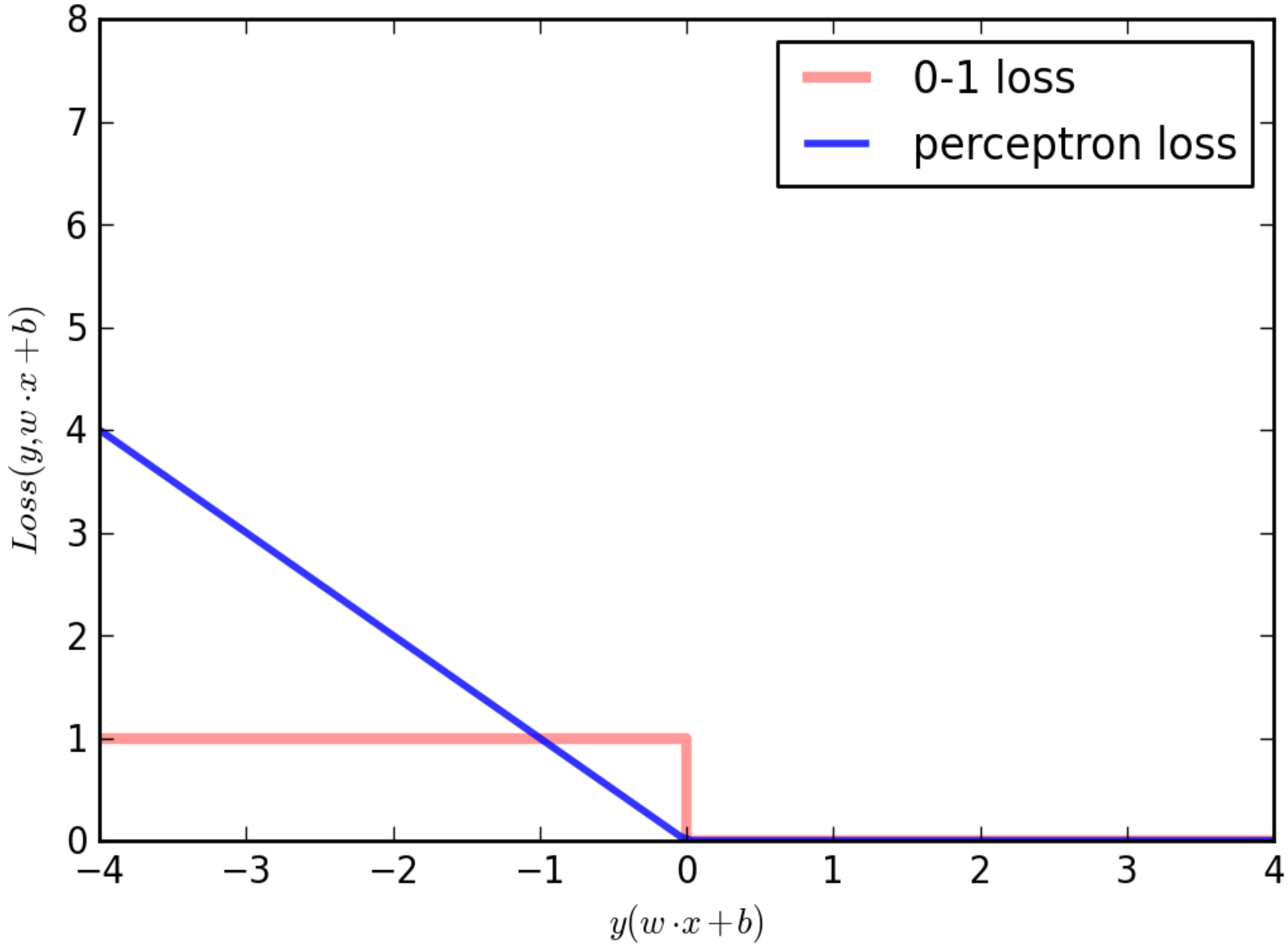
- Invented in 1950s (Rosenblatt)

- One of the simplest classifiers, uses a modified hinge loss:

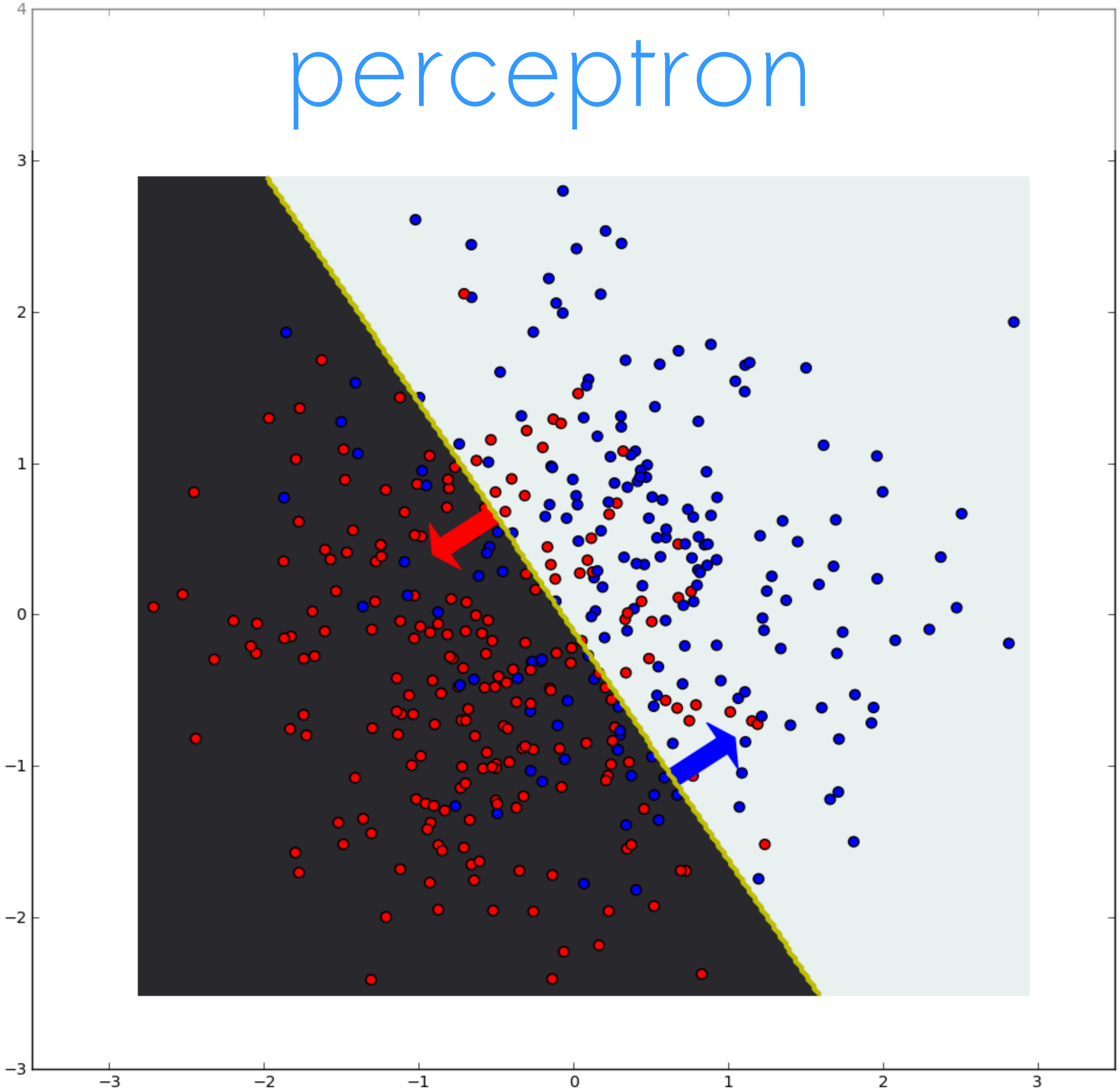
$$\mathbf{Loss}(\mathrm{w}) = \|w\|^2 + C\sum \max(0, -y_i(w \cdot x_i + b))$$

$$\nabla \mathbf{Loss}(\mathrm{w}) = 2w - C\sum_{y_i(w \cdot x_i + b) < 0} y_i x_i$$

perceptron loss



perceptron



loss functions compared

Define **functional margin** $m_i = y_i(w \cdot x_i + b)$

$$\textbf{Logistic Loss}(w) = \sum_{i=1}^{N} log(1 + e^{-m_i}) + \lambda \|w\|^2$$

$$\textbf{SVM Loss}(w) = \sum_{i=1}^{N} max(0, 1 - m_i) + \lambda \|w\|^2$$

$$\textbf{Perceptron Loss}(w) = \sum_{i=1}^{N} max(0, -m_i) + \lambda \|w\|^2$$

loss functions compared