



# CSC380: Principles of Data Science

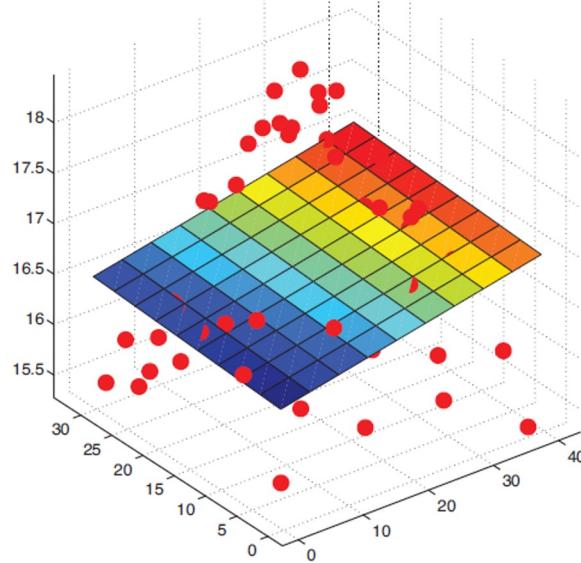
**Nonlinear Models 1**

**Alon Efrat**

- Basis Functions
- Support Vector Machine
- Neural Networks

# Linear Models

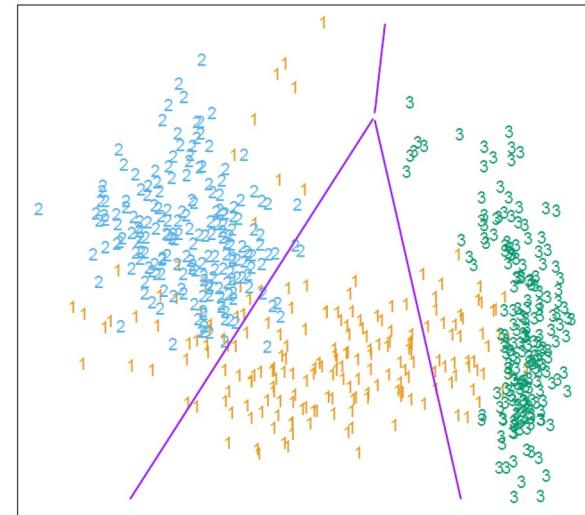
[ Image: Murphy, K. (2012) ]



**Linear Regression** Fit a *linear function* to the data,

$$y = w^T x$$

[ Image: Hastie et al. (2001) ]

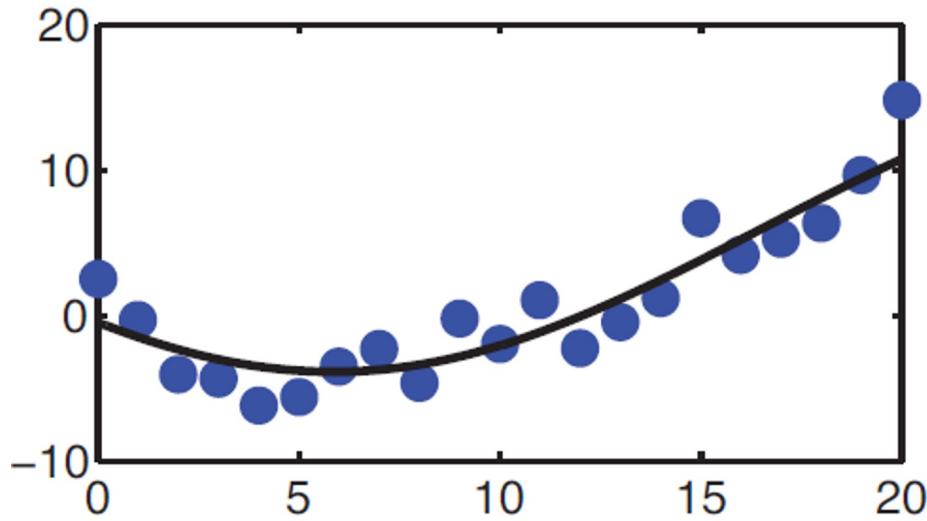


**Logistic Regression** Learn a decision boundary that is *linear in the data*,

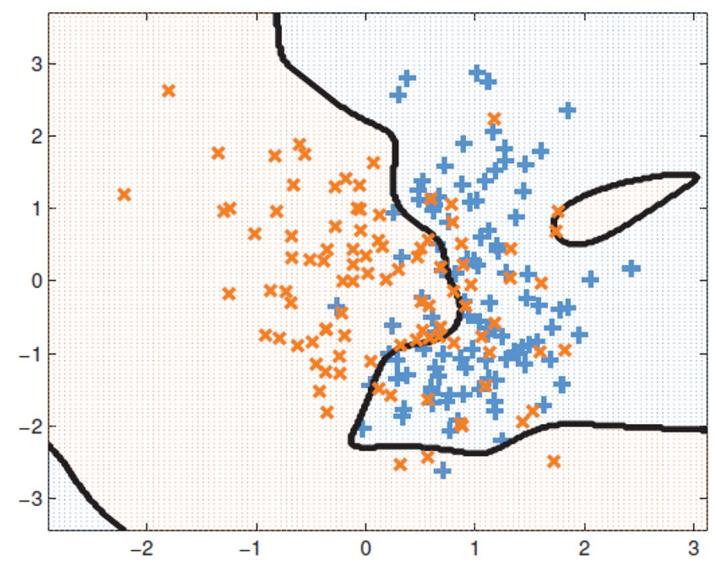
$$y = \mathbf{I}\{w^T x \geq 0\}$$

# Nonlinear Data

5



What if our data are *not* well-described by a linear function?



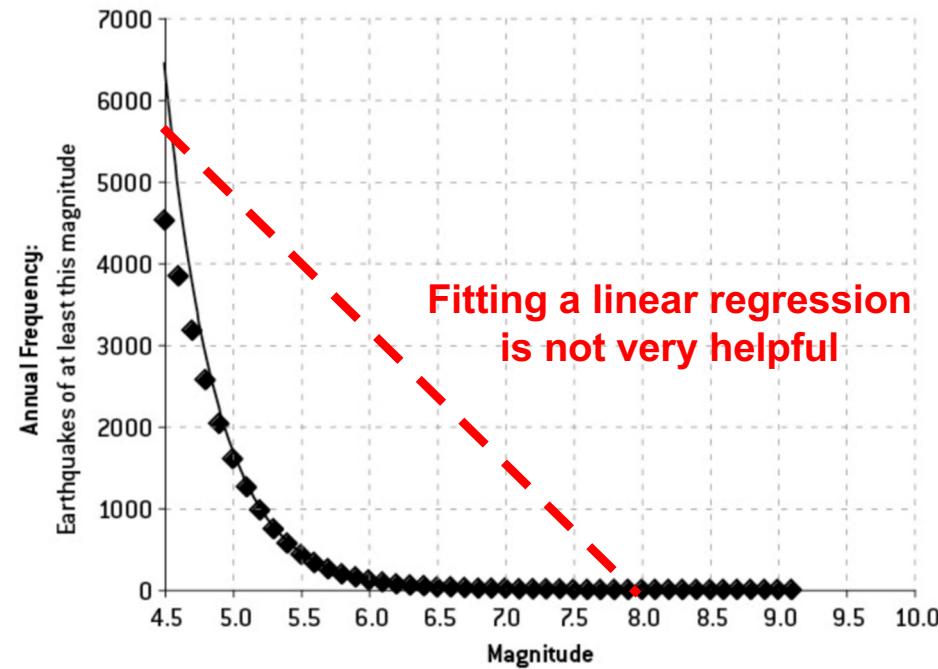
What if classes cannot be well-distinguished by a linear function?

## Example: Earthquake Prediction

6

Suppose that we want to predict the number of earthquakes that occur of a certain magnitude. Our data are given by,

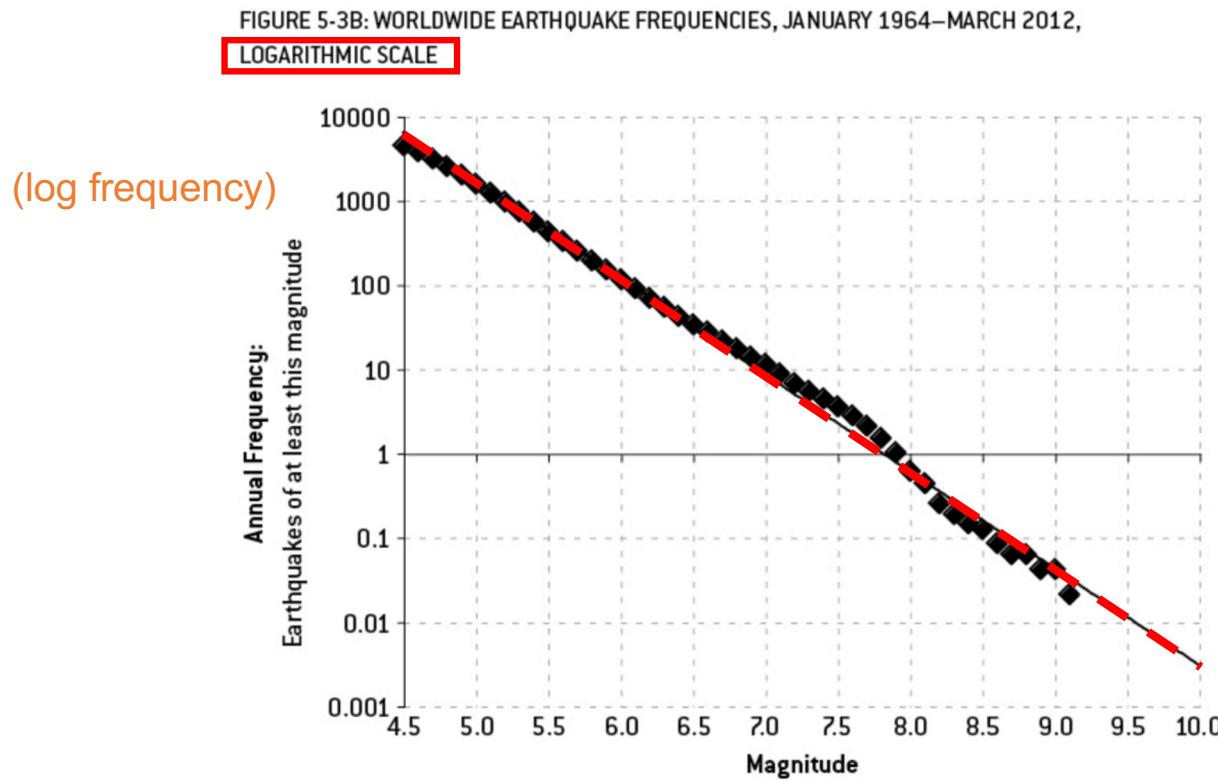
FIGURE 5-3A: WORLDWIDE EARTHQUAKE FREQUENCIES, JANUARY 1964–MARCH 2012



## Example: Earthquake Prediction

7

Suppose that we want to predict the number of earthquakes that occur of a certain magnitude. Our data are given by,



But plotting outputs on a logarithmic scale reveals a strong linear relationship...

it's like  $y = e^{-ax+b}$

- Recall: for 1d problem, we embedded the feature:  $x' = (x, 1) \in \mathbb{R}^2$  so we can encode the intercept term.

$$\phi_0(x) = 1 \quad \phi_1(x) = x \quad y = \mathbf{w}^\top \Phi_{\text{lin}}(x) = \phi_0(x)w_0 + \phi_1(x)w_1 = w_0 + w_1 x$$

- Recall: for 1d problem, we embedded the feature:  $x' = (x, 1) \in \mathbb{R}^2$  so we can encode the intercept term.

$$\phi_0(x) = 1 \quad \phi_1(x) = x \quad y = \mathbf{w}^\top \Phi_{\text{lin}}(x) = \phi_0(x)w_0 + \phi_1(x)w_1 = w_0 + w_1 x$$

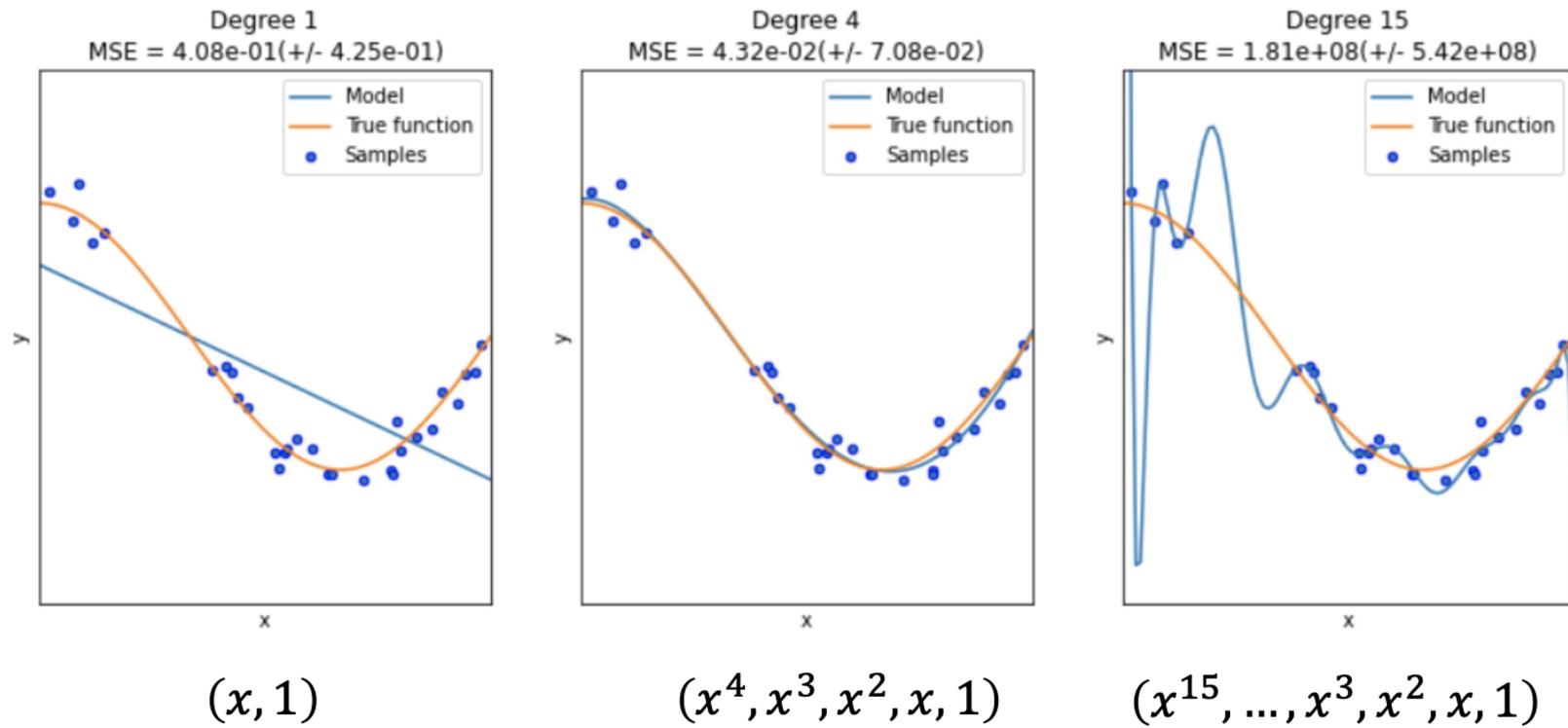
- Actually, the embedding trick is stronger.

- $(x^2, x, 1)$ : 2<sup>nd</sup> order polynomial with respect to  $x$
- $(x^d, x^{d-1}, \dots, 1)$ : d-th order polynomial (= degree d)

$$\phi_0(x) = 1 \quad \phi_1(x) = x \quad \phi_2(x) = x^2$$

$$y = \mathbf{w}^\top \Phi_{\text{lin}}(x) = \phi_0(x)w_0 + \phi_1(x)w_1 + \phi_2(x)w_2 = w_0 + w_1 x + w_2 x^2$$

# Feature embedding trick



higher-order polynomial = higher complexity = prone to overfitting!

- A **basis function** can be any function of the input features  $\mathbf{X}$
- Define a set of  $B$  basis functions  $\phi_1(x), \dots, \phi_B(x)$
- Fit a linear regression model in terms of basis functions,

$$y = \sum_{b=1}^B w_b \phi_b(x) = w^T \phi(x)$$

notation:  
 $\phi(x) := [\phi_1(x), \dots, \phi_B(x)]$

- The model is *linear in the transformed basis/induced features  $\phi(x)$* .
- The model is *nonlinear in the data  $\mathbf{X}$*

# Linear Regression

12

Recall the ordinary least squares solution is given by,

$$\mathbf{X} = \begin{pmatrix} 1 & x_{11} & \dots & x_{1D} \\ 1 & x_{21} & \dots & x_{2D} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{m1} & \dots & x_{mD} \end{pmatrix} \quad \mathbf{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix} \quad w^{\text{OLS}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

**Design Matrix**  
( each training input on a column )

**Vector of  
Training labels**

Can similarly solve in terms of basis functions,

$$\Phi = \begin{pmatrix} 1 & \phi_1(x_1) & \dots & \phi_B(x_1) \\ 1 & \phi_1(x_2) & \dots & \phi_B(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ 1 & \phi_1(x_m) & \dots & \phi_B(x_m) \end{pmatrix} \quad w^{\text{OLS}} = (\Phi^T \Phi)^{-1} \Phi^T \mathbf{y}$$

## sklearn.preprocessing.PolynomialFeatures

### **degree : int or tuple (min\_degree, max\_degree), default=2**

If a single int is given, it specifies the maximal degree of the polynomial features. If a tuple `(min_degree, max_degree)` is passed, then `min_degree` is the minimum and `max_degree` is the maximum polynomial degree of the generated features. Note that `min_degree=0` and `min_degree=1` are equivalent as outputting the degree zero term is determined by `include_bias`.

### **interaction\_only : bool, default=False**

If `True`, only interaction features are produced: features that are products of at most `degree` *distinct* input features, i.e. terms with power of 2 or higher of the same input feature are excluded:

- included: `x[0]`, `x[1]`, `x[0] * x[1]`, etc.
- excluded: `x[0] ** 2`, `x[0] ** 2 * x[1]`, etc.

### **include\_bias : bool, default=True**

If `True` (default), then include a bias column, the feature in which all polynomial powers are zero (i.e. a column of ones - acts as an intercept term in a linear model).

### **order : {'C', 'F'}, default='C'**

Order of output array in the dense case. `'F'` order is faster to compute, but may slow down subsequent estimators.

## Example: Polynomial Basis Functions

14

Create three two-dimensional data points [0,1], [2,3], [4,5]:

```
>>> X = np.arange(6).reshape(3, 2)
>>> X
array([[0, 1],
       [2, 3],
       [4, 5]])
```

Compute quadratic features  $(1, x_1, x_2, x_1^2, x_1x_2, x_2^2)$  ,

```
>>> poly = PolynomialFeatures(degree=2)
>>> poly.fit_transform(X)
array([[ 1.,  0.,  1.,  0.,  0.,  1.],
       [ 1.,  2.,  3.,  4.,  6.,  9.],
       [ 1.,  4.,  5., 16., 20., 25.]])
```

These are now our new data and ready to fit a model...

# Example: Polynomial Basis Functions

15

Create a 3-rd order polynomial (cubic) function,

```
f = lambda x: (x-1)*(x-2)*(x-3)
import numpy.random as ra
ra.seed(20)
train_x = np.arange(5)
train_y = f(train_x) + 1*ra.randn(len(train_x))
train_y
```

✓ 0.3s

```
array([-5.11610689,  0.19586502,  0.35753652, -2.34326191,  4.91516741])
```

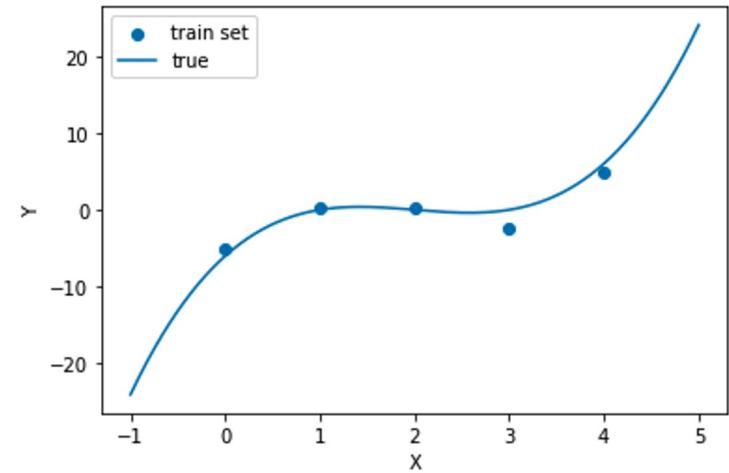
Plot train set and the actual function

```
test_x = np.linspace(-1,5,400)

from matplotlib import pyplot as plt
plt.scatter(train_x,train_y)

plt.plot(test_x, f(test_x))
plt.legend(['train set', 'true'])
plt.xlabel('X')
plt.ylabel('Y')
plt.show()
```

✓ 0.4s



# Example: Polynomial Basis Functions

Create cubic features  $(1, x, x^2, x^3)$

```
poly = PolynomialFeatures(degree=3)
train_xx = poly.fit_transform(train_x[:, np.newaxis])
train_xx
```

✓ 0.4s turns train\_x (length 5 array) into a matrix (5 by 1 matrix)

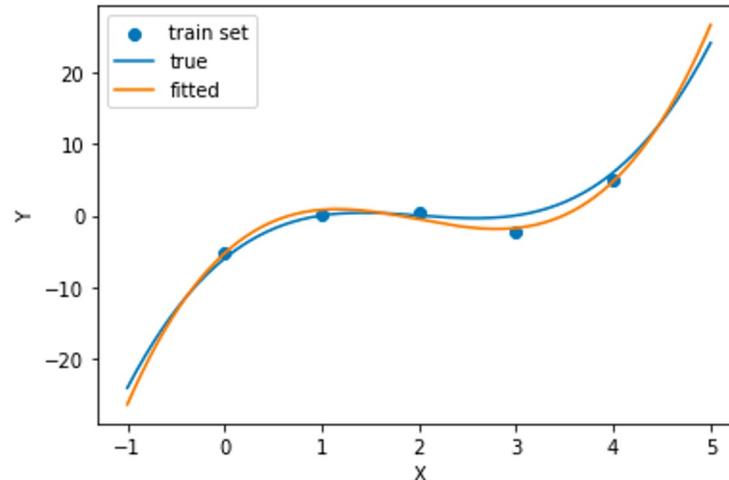
```
array([[ 1.,  0.,  0.,  0.],
       [ 1.,  1.,  1.,  1.],
       [ 1.,  2.,  4.,  8.],
       [ 1.,  3.,  9., 27.],
       [ 1.,  4., 16., 64.]])
```

Perform linear regression; plot it

```
from matplotlib import pyplot as plt
from sklearn.linear_model import LinearRegression
model = LinearRegression().fit(train_xx, train_y)
test_x = np.linspace(-1, 5, 400)
test_xx = poly.fit_transform(test_x[:, np.newaxis])
pred_y = model.predict(test_xx)

plt.scatter(train_x, train_y)
plt.plot(test_x, f(test_x))
plt.plot(test_x, pred_y)
plt.legend(['train set', 'true', 'fitted'])
plt.xlabel('X')
plt.ylabel('Y')
plt.show()
```

✓ 0.2s



- Fit  $x_i$  to  $\log(y_i)$  (note that the sum of error is not so easily explained)
- Assume data in 3D. Allow fitting a function of the form
$$z_i = w_0 + w_1 x_i + w_2 x_i^2 + w_3 y_i$$
(so quadratic in x but linear in y)

- Generally the first step in data science involves *preprocessing* or transforming data in some way
  - Filling in missing values (imputation)
  - Centering / normalizing / standardizing
  - Etc.
- We then fit our models to this preprocessed data
- One way to view preprocessing is simply as computing some basis function  $\phi(x)$ , nothing more

## PROs

- More flexible modeling that is nonlinear in the original data
- Increases model expressivity

## CONs

- Typically requires **more parameters** to be learned
- More sensitive to **overfitting** training data (due to expressivity)
- Requires more **regularization** to avoid overfitting
- Need to find good basis functions (feature engineering)

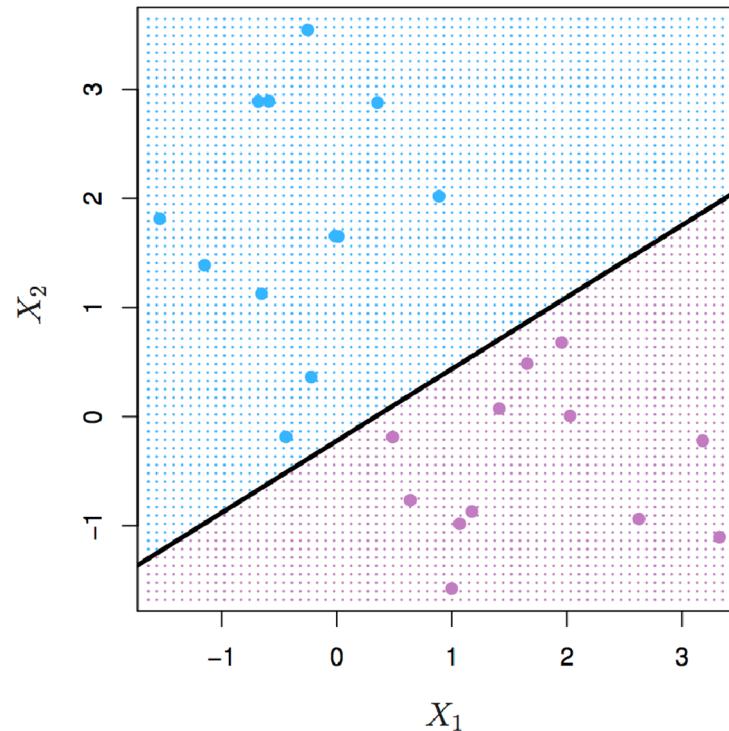
- Basis Functions
- Support Vector Machine
- Neural Networks

# Linear Decision Boundary

21

*Forget about the ‘regression’ point of view for now..*

*At the end of the day, we just want a line that separates the two classes well.*

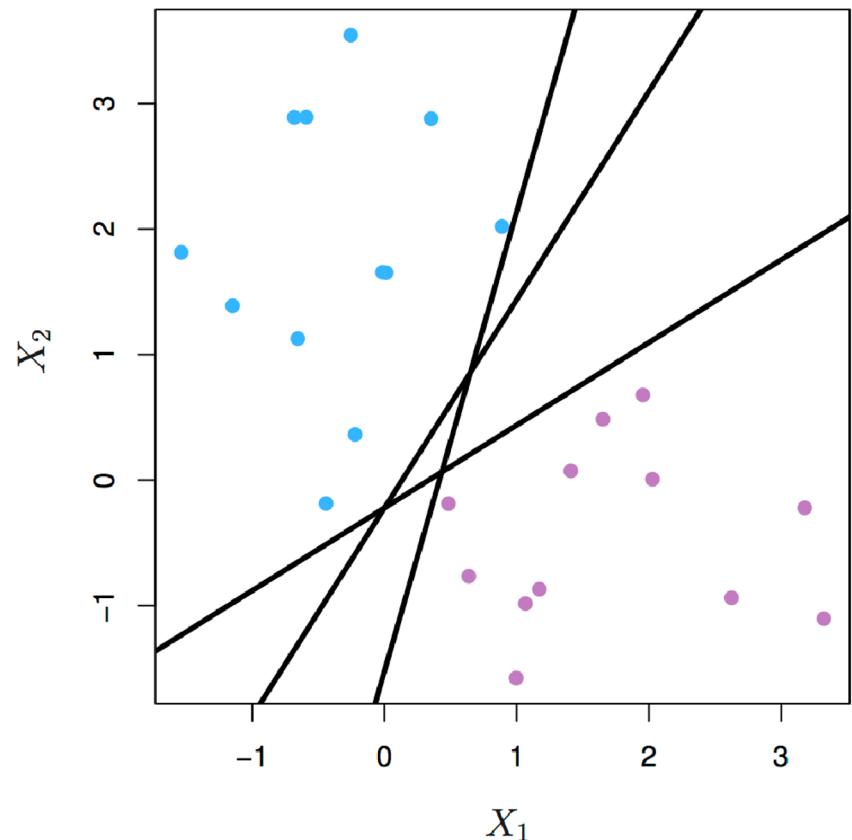


[ Source: <http://www-bcf.usc.edu/~gareth/ISL/> ]

# Linear Decision Boundary

22

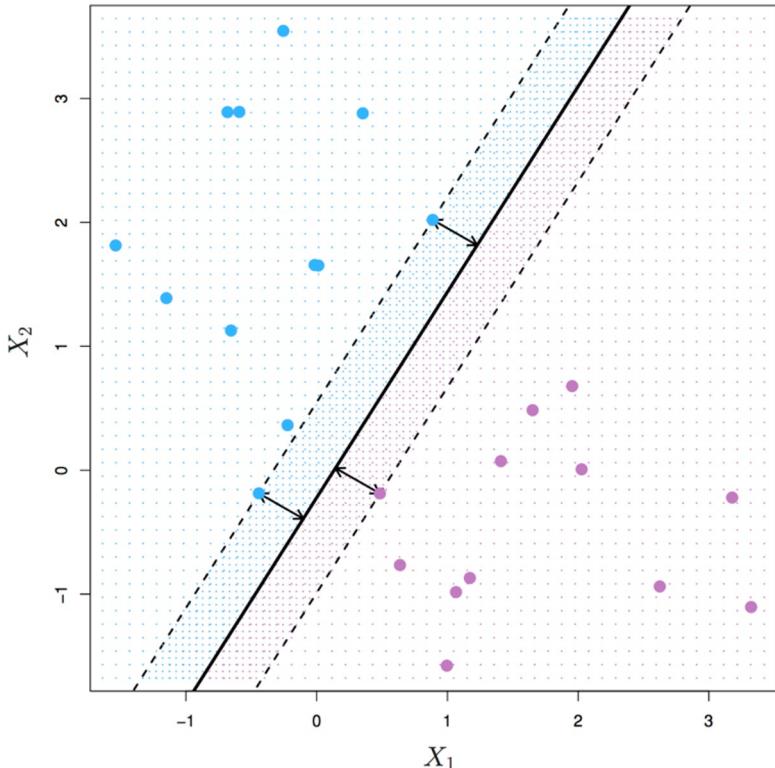
*Note: Any boundary that separates classes is equivalently good on training data*



Q: but if you have to choose one,  
which one will you choose?

# Classifier Margin

23



*The **margin** measures minimum distance between each class and the decision boundary*

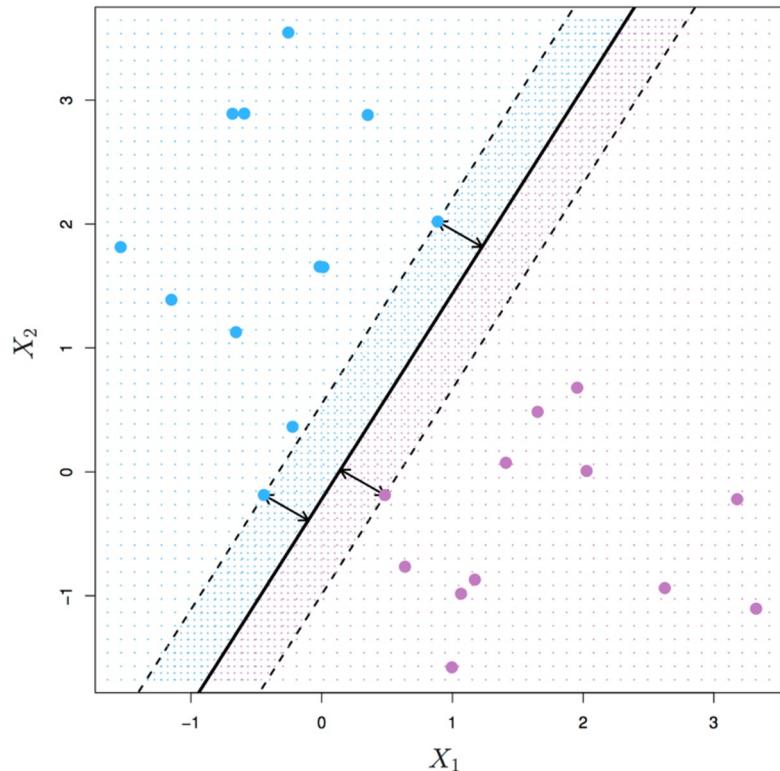
**Observation** Decision boundaries with larger margins are more likely to generalize to unseen data

**Idea** Learn the classifier with the largest margin that still separates the data...

...we call this a **max-margin classifier**

# Max-Margin Classifier (Linear Separable Case)

24



For now, let's focus on the case where the data is **linearly separable**

(Otherwise, there is no margin to talk about!)

# Hyperplane

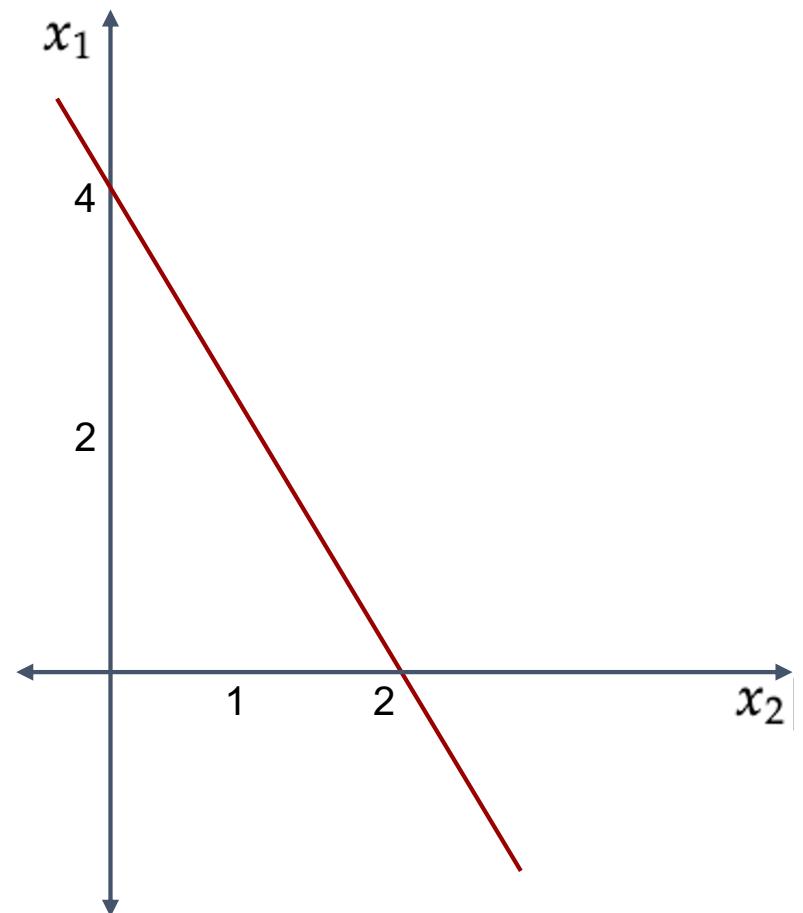
25

A linear discriminant function in D dimensions is given by a hyperplane, defined as follows:

$$\begin{aligned} h(\mathbf{x}) &= \mathbf{w}^T \mathbf{x} + b \\ &= w_1 x_1 + w_2 x_2 + \cdots + w_d x_d + b \end{aligned}$$

For points that lie on the hyperlane, we have:

$$h(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b = 0$$



# Separating Hyperplane

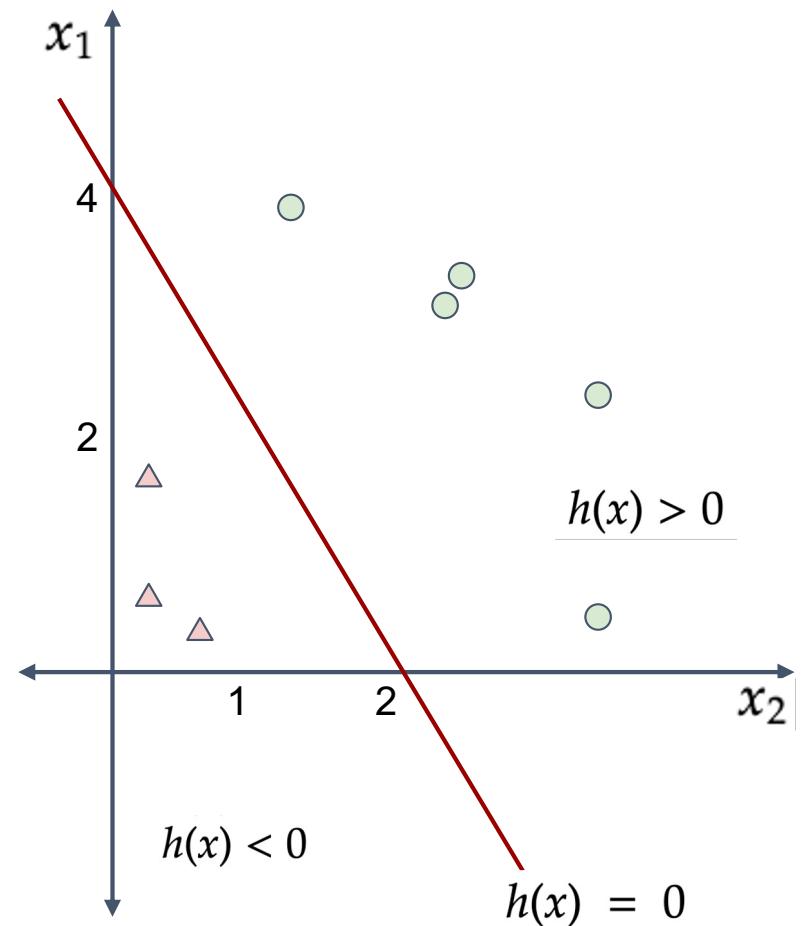
26

A hyperplane  $h(\mathbf{x})$  splits the original  $d$ -dimensional space into two half-spaces.  
If the input dataset is linearly separable:

$$y = \begin{cases} +1 & \text{if } h(\mathbf{x}) > 0 \\ -1 & \text{if } h(\mathbf{x}) < 0 \end{cases}$$

Example:

$$h(x) = x_1 + 2x_2 - 4$$



# Separating Hyperplane: weight vector

27

Let  $a_1$  and  $a_2$  be two arbitrary points that lie on the hyperplane, we have:

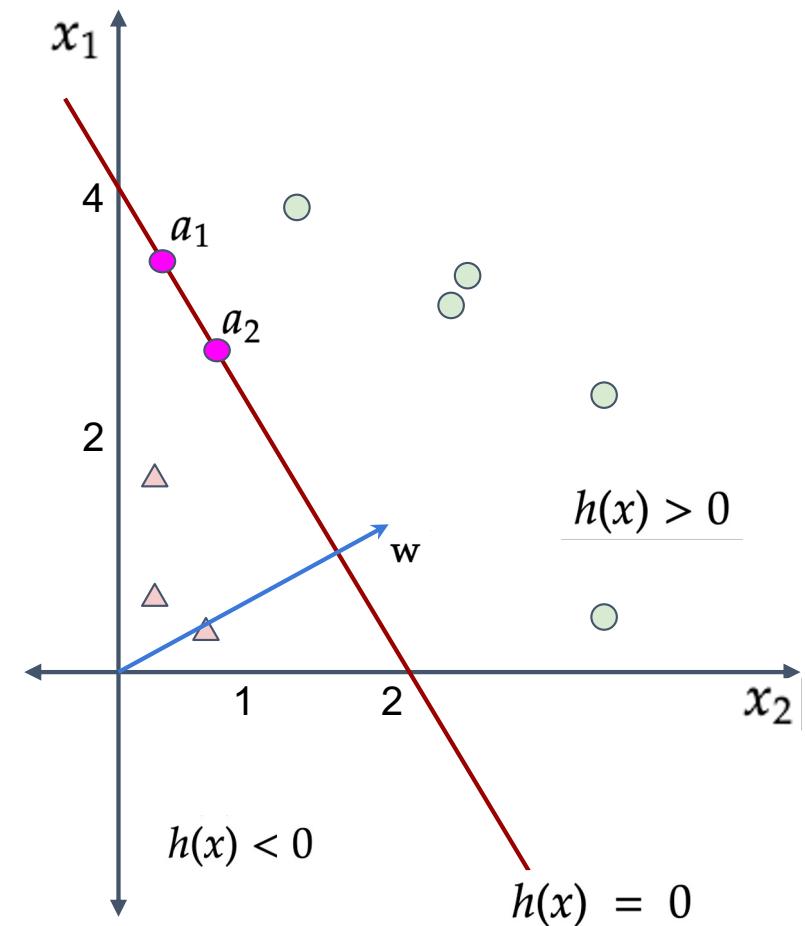
$$h(\mathbf{a}_1) = \mathbf{w}^T \mathbf{a}_1 + b = 0$$

$$h(\mathbf{a}_2) = \mathbf{w}^T \mathbf{a}_2 + b = 0$$

Subtracting one from the other:

$$\mathbf{w}^T (\mathbf{a}_1 - \mathbf{a}_2) = 0$$

The weight vector  $\mathbf{w}$  is orthogonal to the hyperplane.



# Distance of a Point to the Hyperplane

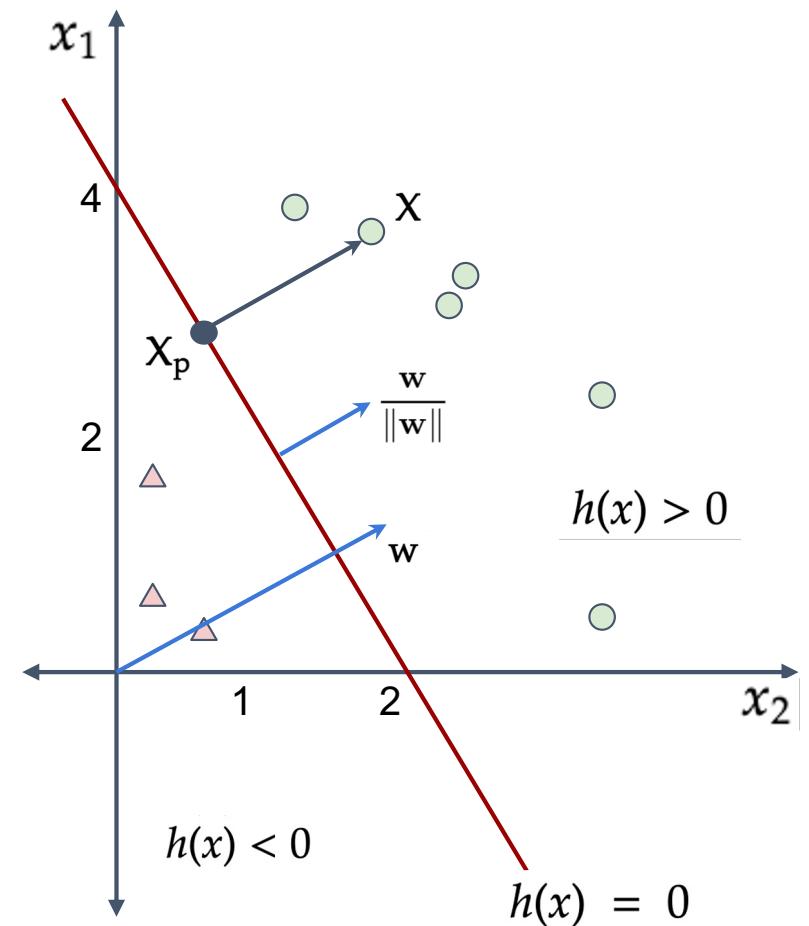
28

Consider a point  $X$  not on the hyperplane. Let  $X_p$  be the projection of  $X$  on the hyperplane.

Let  $r$  be the steps need to walk from  $X_p$  to  $X$ .

$$\mathbf{x} = \mathbf{x}_p + r \frac{\mathbf{w}}{\|\mathbf{w}\|}$$

Q: how many steps/direct distance do we need to walk?



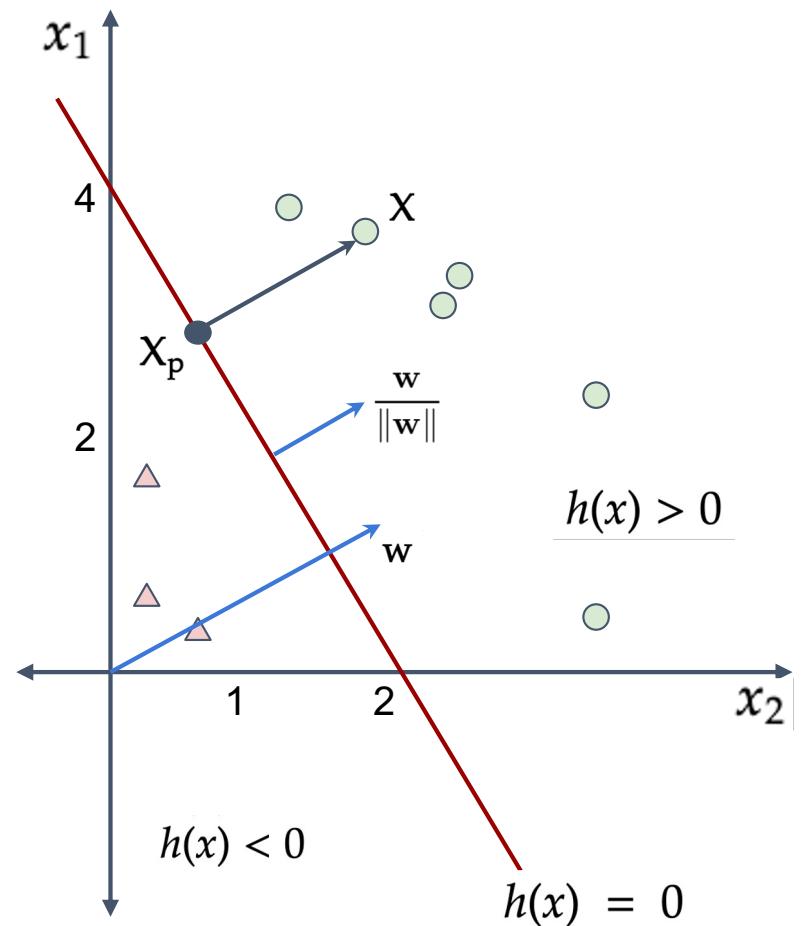
# Distance of a Point to the Hyperplane

29

Consider a point  $X$  not on the hyperplane. Let  $X_p$  be the projection of  $X$  on the hyperplane.

Let  $r$  be the steps need to walk from  $X_p$  to  $X$ .

$$\begin{aligned}
 h(\mathbf{x}) &= h(\mathbf{x}_p + r \frac{\mathbf{w}}{\|\mathbf{w}\|}) \\
 &= \mathbf{w}^T \left( \mathbf{x}_p + r \frac{\mathbf{w}}{\|\mathbf{w}\|} \right) + b \\
 &= \underbrace{\mathbf{w}^T \mathbf{x}_p + b}_{h(\mathbf{x}_p)} + r \frac{\mathbf{w}^T \mathbf{w}}{\|\mathbf{w}\|} \\
 &= \underbrace{h(\mathbf{x}_p)}_0 + r \|\mathbf{w}\| \\
 &= r \|\mathbf{w}\| \quad r = \frac{h(\mathbf{x})}{\|\mathbf{w}\|}
 \end{aligned}$$



# Distance of a Point to the Hyperplane

30

Q: What is the direct distance from origin ( $x=0$ ) to the hyperplane?

$$r = \frac{h(\mathbf{x})}{\|\mathbf{w}\|} \quad r = \frac{h(\mathbf{0})}{\|\mathbf{w}\|} = \frac{\mathbf{w}^T \mathbf{0} + b}{\|\mathbf{w}\|} = \frac{b}{\|\mathbf{w}\|}$$

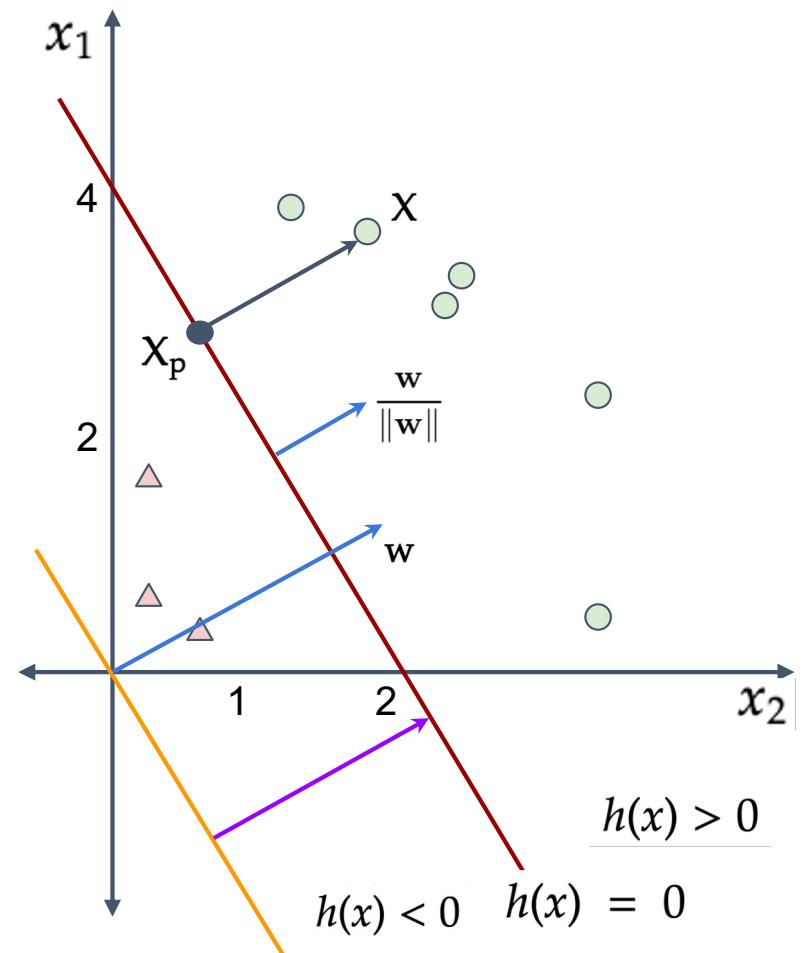
Example:

$$h(x) = x_1 + 2x_2 - 4$$

$$\mathbf{w}^T \mathbf{x} + b = (1 \ 2) \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} - 4$$

$$\frac{b}{\|\mathbf{w}\|} = -\frac{4}{\sqrt{5}}$$

Q: how to deal with negative distance?



# Distance of a Point to the Hyperplane

31

Q: How to deal with negative distance?

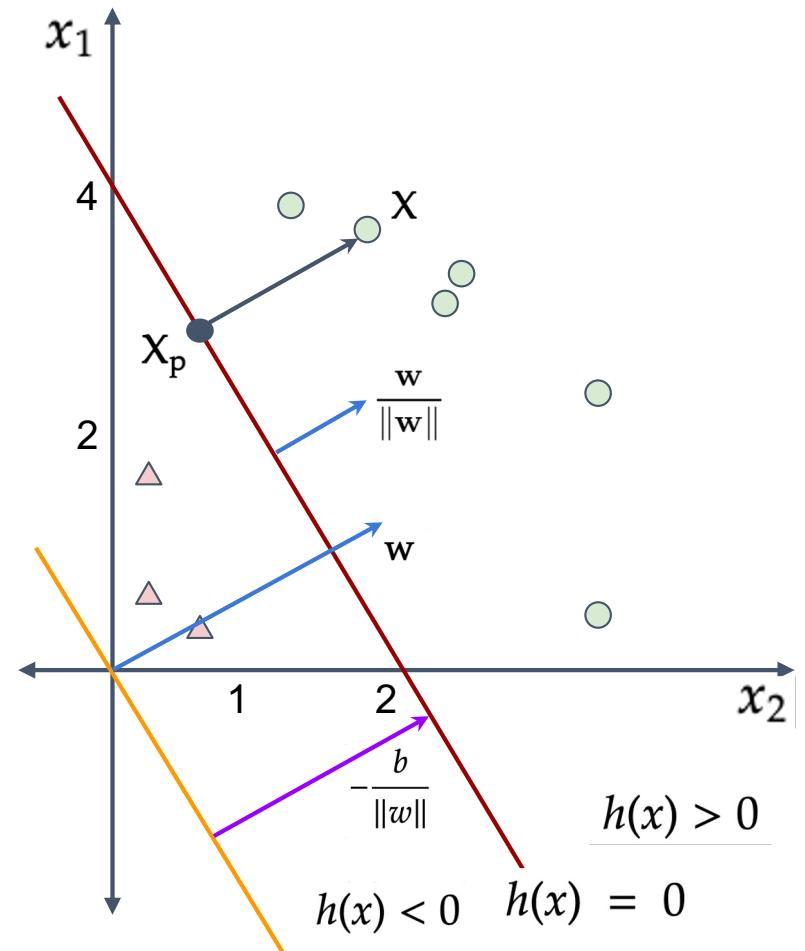
$$r = \frac{h(\mathbf{x})}{\|\mathbf{w}\|}$$

$$y = \begin{cases} +1 & \text{if } h(\mathbf{x}) > 0 \\ -1 & \text{if } h(\mathbf{x}) < 0 \end{cases}$$

$$\delta = y r = \frac{y h(\mathbf{x})}{\|\mathbf{w}\|}$$

Example (when point is the origin):

$$(-1) \cdot \frac{b}{\|\mathbf{w}\|} = \frac{4}{\sqrt{5}}$$



# Margin and Support Vectors

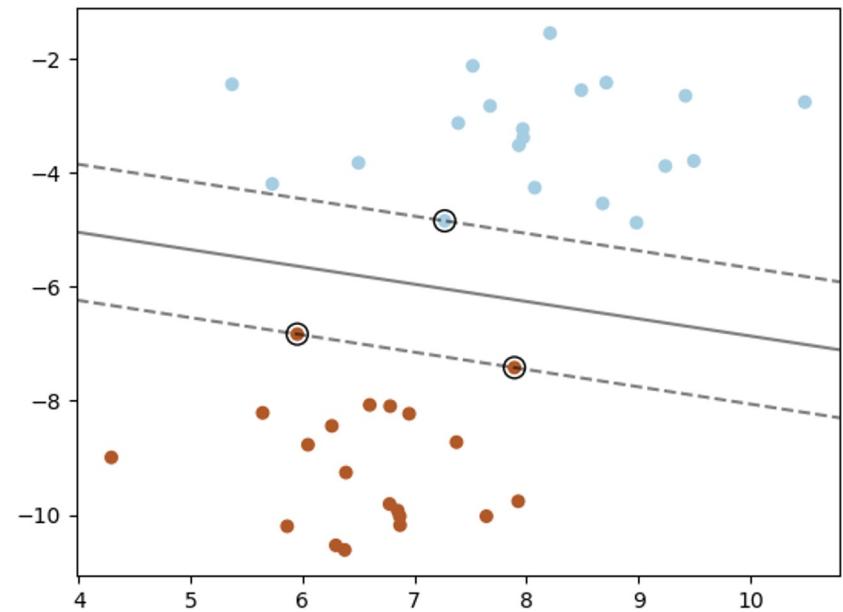
32

Over all the n points, the **margin** of the linear classifier is the minimum distance of a point from the separating hyperplane:

$$\delta^* = \min_{\mathbf{x}_i} \left\{ \frac{y_i(\mathbf{w}^T \mathbf{x}_i + b)}{\|\mathbf{w}\|} \right\}$$

All the points that achieve this minimum distance are called **support vectors**.

$$\delta^* = \frac{y^*(\mathbf{w}^T \mathbf{x}^* + b)}{\|\mathbf{w}\|}$$



## Max-Margin Classifier (Linear Separable Case) 33

For training data  $\{(x^{(i)}, y^{(i)})\}_{i=1}^m$ , a classifier  $f(x) = w^\top x + b$  with 0 train error will satisfy

$$y^{(i)}f(x^{(i)}) = y^{(i)}(w^\top x^{(i)} + b) > 0$$

↓ negative margin when misclassifying it!

The distance for  $(x^{(i)}, y^{(i)})$  to separating hyperplane

$$\frac{y^{(i)}(w^\top x^{(i)} + b)}{\|w\|}$$

The margin of a classifier  $f(x)$  is

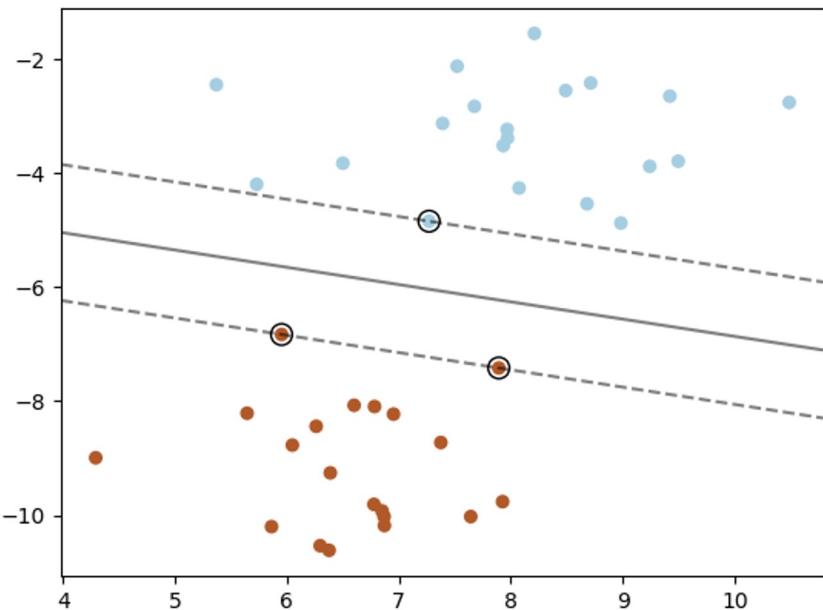
$$\min_i \frac{y^{(i)}(w^\top x^{(i)} + b)}{\|w\|}$$

Find  $f$  that maximize margin

$$\arg \max_{w,b} \min_i \frac{y^{(i)}(w^\top x^{(i)} + b)}{\|w\|}$$

# Max-Margin Classifier (Linear Separable Case)

34



Maximize the minimum margin

$$\arg \max_{w,b} \min_i \frac{y^{(i)}(w^\top x^{(i)} + b)}{\|w\|}$$

Minimum margin over all training data

Find the parameters  $(w, b)$  that **maximize** the **smallest margin** over all the training data

# Canonical Hyperplane

35

**Issue:** infinite equivalent hyperplanes result in infinite solutions:

- Multiplying on both sides by some scalars yields an equivalent hyperplane

$$s h(\mathbf{x}) = s \mathbf{w}^T \mathbf{x} + s b$$

Example of equivalent hyperplanes:

$$h(x) = x_1 + 2x_2 - 4$$

$$h(x) = 2x_1 + 4x_2 - 8$$

$$\arg \max_{w,b} \min_i \frac{y^{(i)}(w^T x^{(i)} + b)}{\|w\|}$$

Maximize the minimum margin

Minimum margin over all training data

# Canonical Hyperplane

36

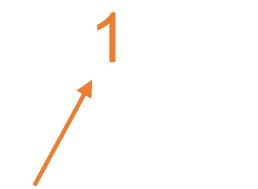
Way to solve this issue:

- Choose the scalar  $s$  such that the absolute distance of a **support vector** from the hyperplane is 1.

$$sy^*(\mathbf{w}^T \mathbf{x}^* + b) = 1$$

$$s = \frac{1}{y^*(\mathbf{w}^T \mathbf{x}^* + b)} = \frac{1}{y^* h(\mathbf{x}^*)}$$

$$y_i (\mathbf{w}^T \mathbf{x}_i + b) \geq 1, \text{ for all points } \mathbf{x}_i \in \mathbf{D}$$

$$\arg \max_{w,b} \min_i \frac{y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + b)}{\|\mathbf{w}\|}$$


$$\text{Margin: } \delta^* = \frac{1}{\|\mathbf{w}\|}$$

$$\text{Max margin: } h^* = \arg \max_h \{\delta_h^*\} = \arg \max_{\mathbf{w}, b} \left\{ \frac{1}{\|\mathbf{w}\|} \right\}$$

# Canonical Hyperplane: an example

37

$$h'(\mathbf{x}) = \begin{pmatrix} 5 \\ 2 \end{pmatrix}^T \mathbf{x} - 20 = 0$$

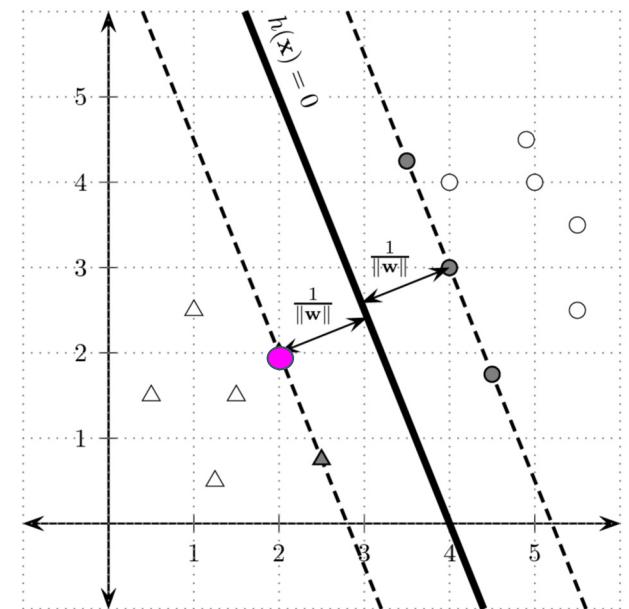
support vector  $\mathbf{x}^* = (2, 2)$ , with class  $y^* = -1$

$$s = \frac{1}{y^* h'(\mathbf{x}^*)} = \frac{1}{-1 \left( \begin{pmatrix} 5 \\ 2 \end{pmatrix}^T \begin{pmatrix} 2 \\ 2 \end{pmatrix} - 20 \right)} = \frac{1}{6}$$

$$\mathbf{w} = \frac{1}{6} \begin{pmatrix} 5 \\ 2 \end{pmatrix} = \begin{pmatrix} 5/6 \\ 2/6 \end{pmatrix} \quad b = \frac{-20}{6}$$

$$h(\mathbf{x}) = \begin{pmatrix} 5/6 \\ 2/6 \end{pmatrix}^T \mathbf{x} - 20/6 = \begin{pmatrix} 0.833 \\ 0.333 \end{pmatrix}^T \mathbf{x} - 3.33$$

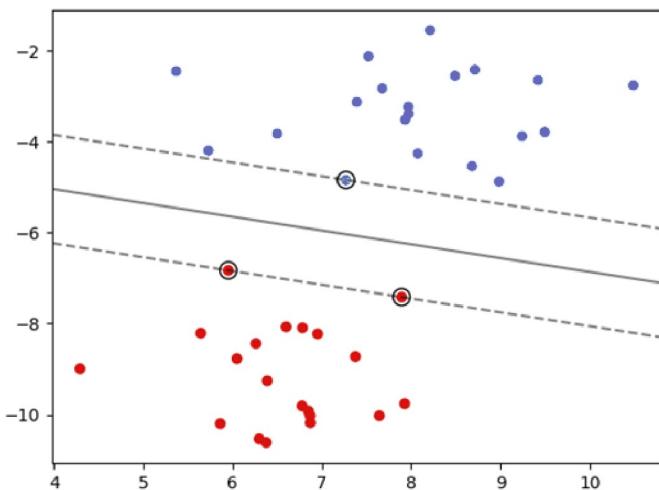
$$\delta^* = \frac{y^* h(\mathbf{x}^*)}{\|\mathbf{w}\|} = \frac{-1 \left( \begin{pmatrix} 5/6 \\ 2/6 \end{pmatrix}^T \begin{pmatrix} 2 \\ 2 \end{pmatrix} - 20/6 \right)}{\sqrt{(\frac{5}{6})^2 + (\frac{2}{6})^2}} = \frac{1}{\frac{\sqrt{29}}{6}} = 1.114$$



# Support Vector Machine (Hard Margin)

38

... it leads to



$$\min_{w,b} \frac{1}{2} \|w\|^2$$

subject to

$$y^{(i)}(w^\top x^{(i)} + b) \geq 1 \quad \text{for } i = 1, \dots, m$$

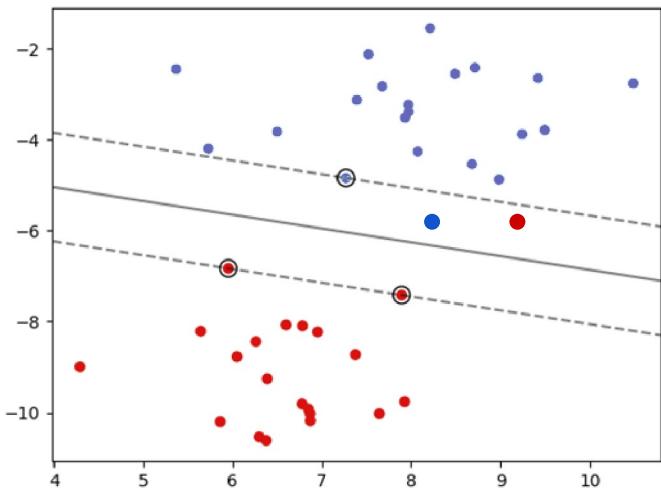
This is a convex (quadratic) optimization problem  
that can be solved efficiently

- Data are D-dimensional vectors
- Margins determined by nearest data points called *support vectors*
- We call this a *support vector machine* (SVM)

# Support Vector Machine (Soft Margin)

39

If the data is linearly not separable,



$$\begin{aligned} & \min_{w,b} \frac{1}{2} \|w\|^2 + C \cdot \sum_{i=1}^m \xi_i \\ & \text{subject to} \\ & y^{(i)}(w^\top x^{(i)} + b) \geq 1 - \xi_i \\ & \xi_i \geq 0 \quad \text{for } i = 1, \dots, m \end{aligned}$$

error

Tradeoff between margin and the error!

# Support Vector Machine

$$\text{minimize} \frac{1}{2} \|w\|^2 + C \cdot \sum_{i=1}^m \xi_i$$

subject to

$$y^{(i)}(w^\top x^{(i)} + b) \geq 1 - \xi_i$$

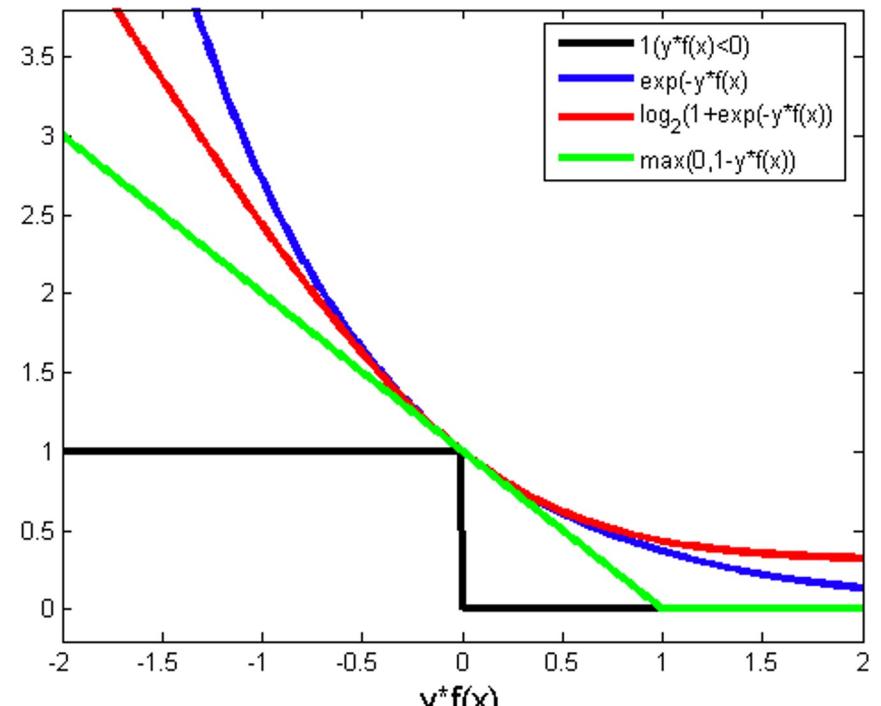
$$\xi_i \geq 0 \quad \text{for } i = 1, \dots, m$$

Equivalent formulation

$$\min_{w,b} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^m (1 - y^{(i)}(w^\top x^{(i)} + b))_+$$

$$\ell(f; x^{(i)}, y^{(i)}) = (1 - y^{(i)}f(x^{(i)}))_+$$

$(X)_+ := \max(X, 0)$



# General Principle

$$\arg \min_{w,b} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^m (1 - y^{(i)}(w^\top x^{(i)} + b))_+$$

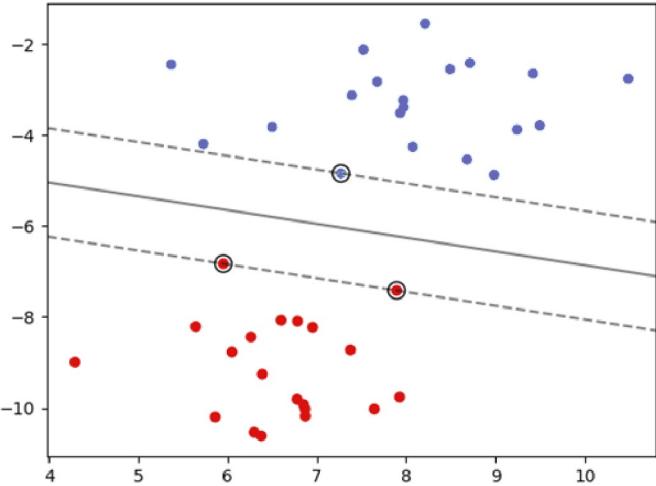
=> by setting  $C = 1/\lambda$ , it's equivalent to solve

$$\arg \min_{w,b} \frac{\lambda}{2} \|w\|^2 + \sum_{i=1}^m (1 - y^{(i)}(w^\top x^{(i)} + b))_+$$

SVM belongs to the general loss-oriented formulation!

$$\text{Model} = \arg \min_{\text{model}} \text{Loss}(\text{Model}, \text{Data}) + \lambda \cdot \text{Regularizer}(\text{Model})$$

# Support Vectors



$$\begin{aligned} & \min_{w,b} \frac{1}{2} \|w\|^2 + C \cdot \sum_{i=1}^m \xi_i \\ & \text{subject to} \\ & y^{(i)}(w^\top x^{(i)} + b) \geq 1 - \xi_i \\ & \xi_i \geq 0 \quad \text{for } i = 1, \dots, m \end{aligned}$$

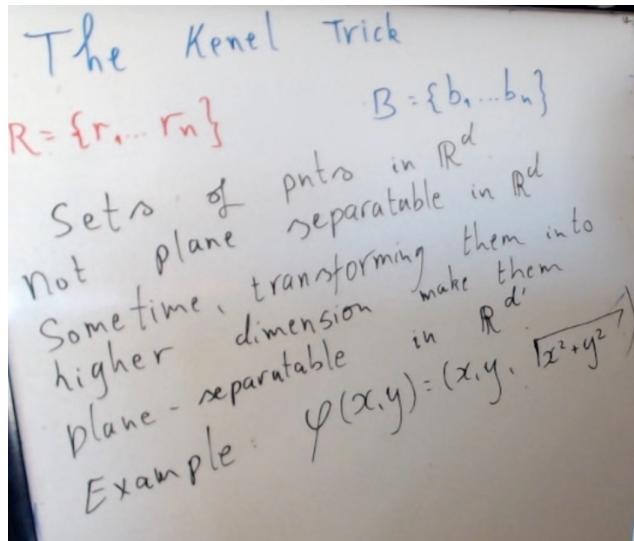
Those data points achieving equality  $y^{(i)}(w^\top x^{(i)} + b) = 1 - \xi_i$  are called **support vectors**.

Turns out, if you knew support vectors already, solving the optimization problem above with **just the support vectors as train set** leads to the same solution.

⇒ Leave-one-out cross validation can be done fast!

# Using polynomials, and the Kernel Trick

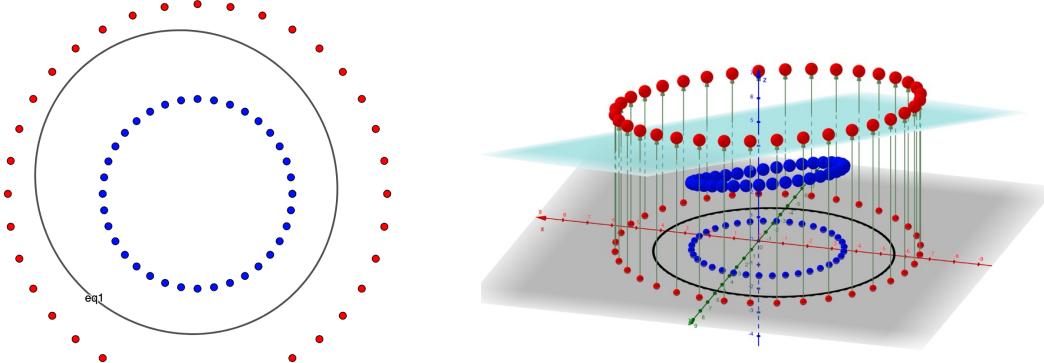
44



1. Let  $d$  denote the number of features, and let  $n$  denote the number of data points
2. When  $n \gg d$ , use LinearSVC to solve the **primal** problem. Much faster (only convex programming in low dimension). However still doable with polynomial description (but takes dimension up)
3. Use Dual-SVN if  $n \ll d$  or if want to use the kernel trick. E.g In python  
**model = SVC(kernel='rbf', C=1.0)**

Where Radial Basis Function as Kernel uses

$$K(x, x') = \exp(-\gamma |x - x'|^2)$$



[ggb](#) [video clip](#)

SVM with linear decision boundaries,

sklearn.svm.LinearSVC

Call options include...

**penalty : {‘l1’, ‘l2’}, default=‘l2’**

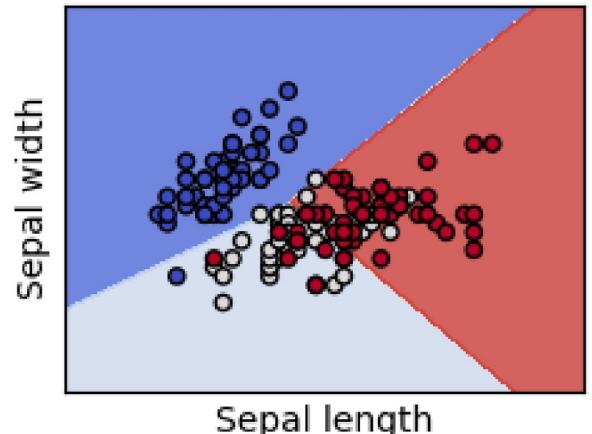
Specifies the norm used in the penalization. The ‘l2’ penalty is the standard used in SVC. The ‘l1’ leads to `coef_` vectors that are sparse.

**dual : bool, default=True**

Select the algorithm to either solve the dual or primal optimization problem. Prefer dual=False when `n_samples > n_features`.

**C : float, default=1.0**

Regularization parameter. The strength of the regularization is inversely proportional to C. Must be strictly positive.



## sklearn.svm.SVC

### **kernel : {'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'}, default='rbf'**

Specifies the kernel type to be used in the algorithm. It must be one of 'linear', 'poly', 'rbf', 'sigmoid', 'precomputed' or a callable. If none is given, 'rbf' will be used. If a callable is given it is used to pre-compute the kernel matrix from data matrices; that matrix should be an array of shape (n\_samples, n\_samples).

### **gamma : {'scale', 'auto'} or float, default='scale'**

Kernel coefficient for 'rbf', 'poly' and 'sigmoid'.

- if `gamma='scale'` (default) is passed then it uses  $1 / (\text{n\_features} * \text{X.var()})$  as value of gamma,
- if 'auto', uses  $1 / \text{n\_features}$ .

for RBF,  
small  $\gamma$ : complex decision boundary  
large  $\gamma$ : more like linear decision boundary

### **max\_iter : int, default=-1**

Hard limit on iterations within solver, or -1 for no limit.

### **verbose : bool, default=False**

Enable verbose output. Note that this setting takes advantage of a per-process runtime setting in libsvm that, if enabled, may not work properly in a multithreaded context.

### **class\_weight : dict or 'balanced', default=None**

Set the parameter C of class i to  $\text{class\_weight}[i] * C$  for SVC. If not given, all classes are supposed to have weight one. The "balanced" mode uses the values of y to automatically adjust weights inversely proportional to class frequencies in the input data as  $n_{\text{samples}} / (n_{\text{classes}} * \text{np.bincount}(y))$ .

# Example: Fisher's Iris Dataset

47

*Classify among 3 species of Iris flowers...*



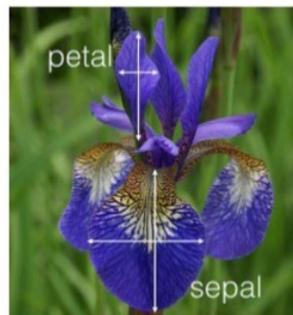
Iris setosa



Iris versicolor



Iris virginica

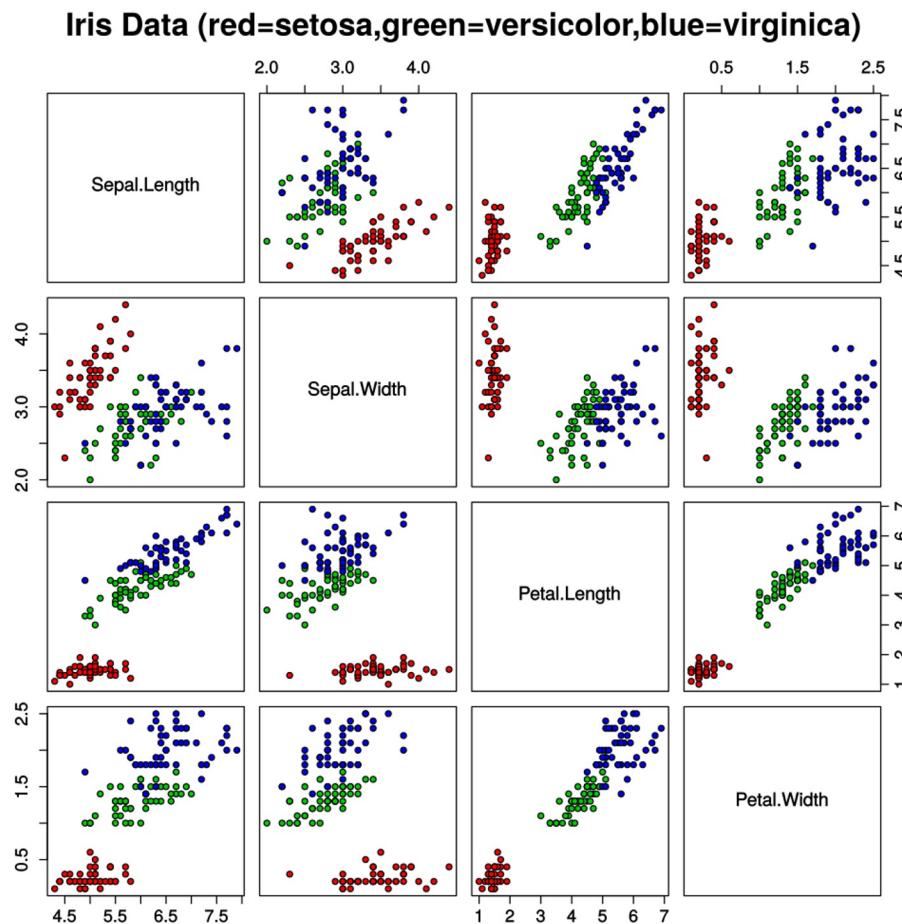


Four features (in centimeters)

- Petal length / width
- Sepal length / width

# Example: Fisher's Iris Dataset

48



*Fairly easy to separate  
setosa from others using a  
linear classifier*

*Need to use nonlinear basis /  
kernel representation to  
better separate other classes*

# Example: Fisher's Iris Dataset

49

Train 8-degree polynomial kernel SVM classifier,

```
from sklearn.svm import SVC  
svclassifier = SVC(kernel='poly', degree=8)  
svclassifier.fit(X_train, y_train)
```

Generate predictions on held-out test data,

```
y_pred = svclassifier.predict(X_test)
```

Show confusion matrix and classification accuracy,

```
print(confusion_matrix(y_test, y_pred))  
print(classification_report(y_test, y_pred))
```

```
[[11  0  0]  
 [ 0 12  1]  
 [ 0  0  6]]
```

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	11
Iris-versicolor	1.00	0.92	0.96	13
Iris-virginica	0.86	1.00	0.92	6
avg / total	0.97	0.97	0.97	30

[ Source: <https://stackabuse.com/implementing-svm-and-kernel-svm-with-pythons-scikit-learn/> ]

# Trick for Multi-Class

50

- Recall: logistic regression had a very natural extension to multi-class.
- What about SVM?

... Researchers have found a few, but it was not any better than a simple trick below.

$$\text{binary: } p(y = 1 | x) = \frac{1}{1 + e^{-w^T x}}$$

$$\text{multi-class: } p(y = j | x) = \frac{\exp(w^{(j)^T} x)}{\sum_{c=1}^C \exp(w^{(c)^T} x)}$$

## [One-vs-the-rest trick]

- Given: dataset  $D = \{(x^{(i)}, y^{(i)})\}_{i=1}^m$
- For each class  $c \in \{1, \dots, C\}$ 
  - Define label  $z^{(i)} \in \{-1, 1\}$  where 1 for class  $c$  and -1 for other classes, for all  $i=1, \dots, m$ .
  - Train a classifier  $f_c$  with  $\{(x^{(i)}, z^{(i)})\}_{i=1}^m$
- To classify  $x^*$ , compute  $\hat{y} = \arg \max_{c \in \{1, \dots, C\}} \text{decision\_value}(f_c(x^*))$