Computer Science

CSC380: Principles of Data Science

Basic machine learning 3

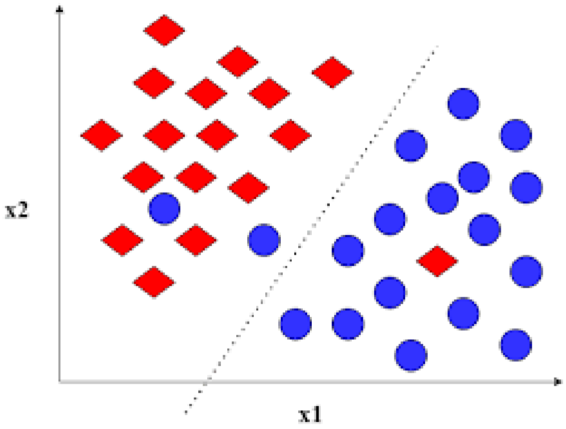
Chicheng Zhang

Outline

- Support Vector Machines
- Nonlinear models
  - Basis functions, kernels
  - Neural networks
- Unsupervised learning: clustering

Support vector machines

Classification



For this section (SVMs):

- We will focus on classification with binary labels

- We will use the convention that the labels of examples are in $\{-1, +1\}$

Linear classifier is a hyperplane

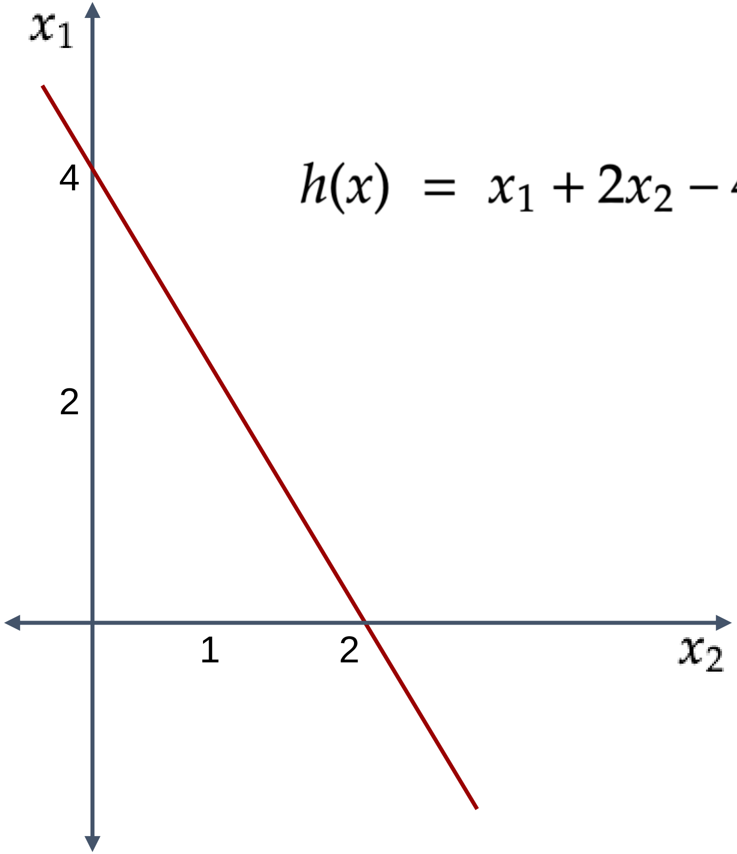
A linear classifier in d dimensions is given by a hyperplane, defined as follows:

Notation: inner product

$$h(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$$
$$= w_1x_1 + w_2x_2 + \cdots + w_dx_d + b$$

For points that lie on the hyperlane, we have:

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

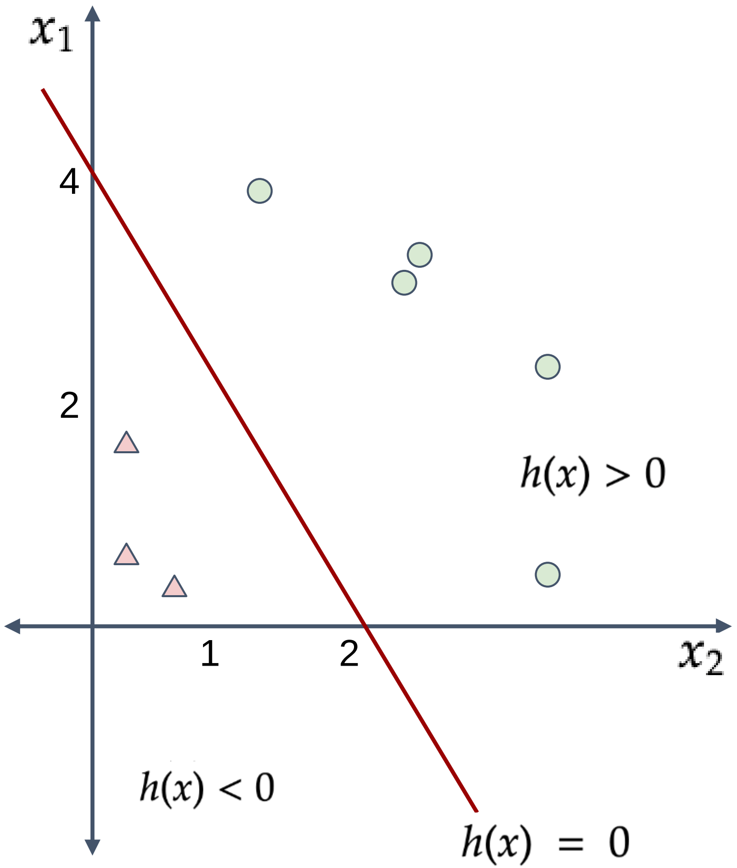


$$h(x) = x_1 + 2x_2 - 4$$

Separating Hyperplane

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

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



Separating Hyperplane: weight vector

Fact The weight vector **w** is orthogonal to the hyperplane.

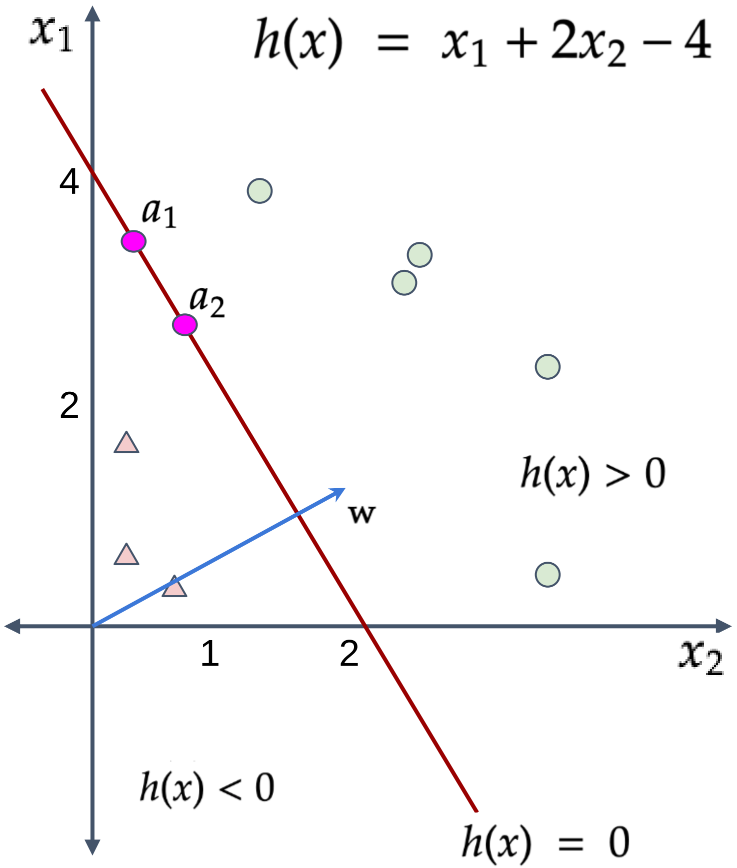
w also known as the *normal vector*

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

$$h(\mathbf{a}_1) = \mathbf{w}^T\mathbf{a}_1 + b = 0$$
$$h(\mathbf{a}_2) = \mathbf{w}^T\mathbf{a}_2 + b = 0$$

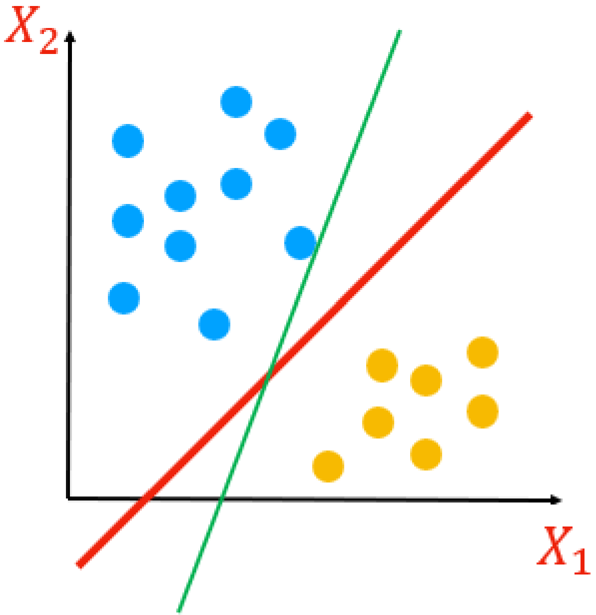
Subtracting one from the other:

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



Linear Decision Boundary

Any boundary that separates classes is equally good on training data

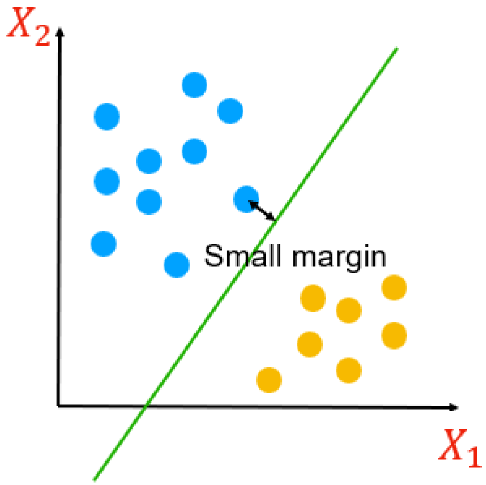


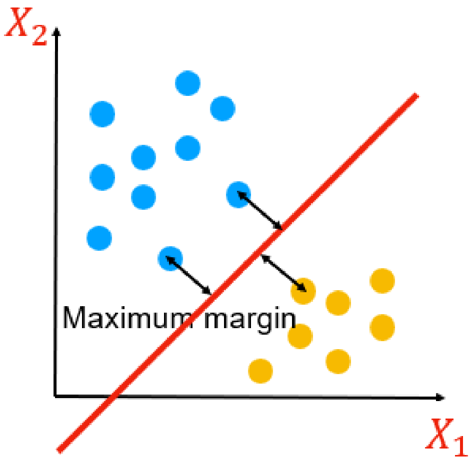
But are they equally good on unseen test data?

Which boundary is better, red or green?

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

Classifier Margin





*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

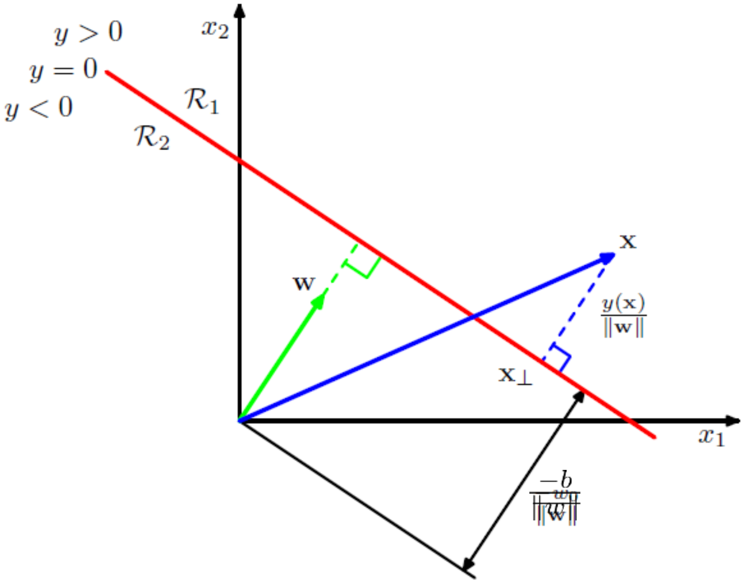
Idea Learn the classifier with the largest margin that still separates the data…

…we call this a *max-margin classifier*

Background: distance of a point to decision boundary

A linear classifier is given by

$$f(x) = w^T x + b$$



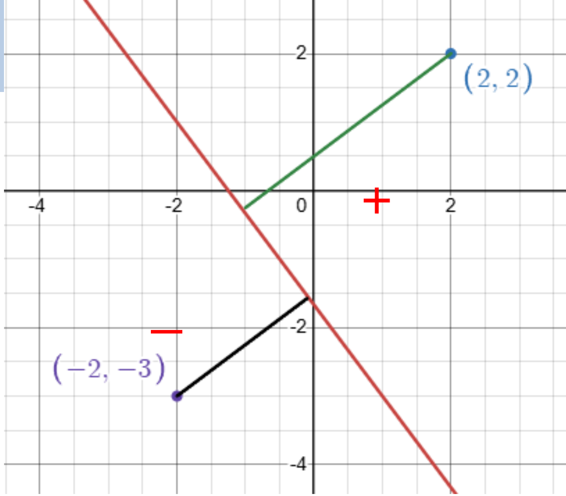
Decision boundary is now at $f(x) = 0$ and distance of x to it is:

$$\frac{f(x)}{\|w\|}$$

Where the norm of the weights is $\|w\| = \sqrt{w^T w} = \sqrt{\sum_i w_i^2}$

Known as the *distance from a point to a plane* equation: [wiki/Distance_from_a_point_to_a_plane]

Example



Linear classifier: $f(x) = 0.8x_1 + 0.6x_2 + 1$

Decision boundary: $0.8x_1 + 0.6x_2 + 1 = 0$

Distance of (2,2) to the boundary?

$$\frac{0.8 \times 2 + 0.6 \times 2 + 1}{\sqrt{0.8^2 + 0.6^2}} = 3.8$$

Distance of (-2,-3) to the boundary?

$$\frac{0.8 \times (-2) + 0.6 \times (-3) + 1}{\sqrt{0.8^2 + 0.6^2}} = -2.4$$

Here distances are *signed:*

sign represents which side the point is at

i.e, the predicted label

Classification margin

Given linear classifier $w \cdot x + b$, its *classification margin* on *labeled example* (x, y) is defined as $\frac{y(w \cdot x + b)}{||w||_2}$

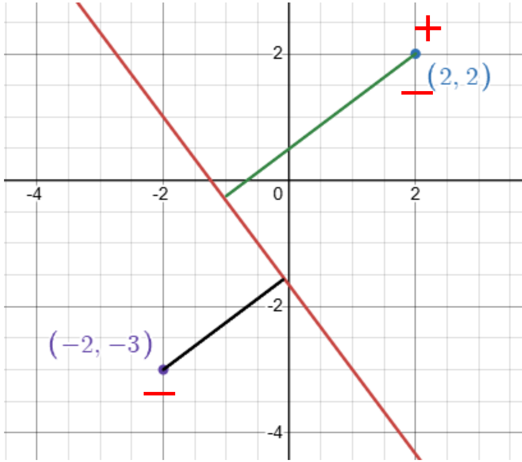
Example $f(x) = 0.8x_1 + 0.6x_2 + 1,$ $||w||_2 = 1$

x	y
$(2,2)$	$+$
$(-2,-3)$	$-$
$(2,2)$	$-$

margin = $+1 \times 3.8 = 3.8$

margin = $-(-2.4) = 2.4$

margin = $-1 \times 3.8 = -3.8$



Margin > 0 ⇔ correct classification

Larger margin: correct with higher confidence

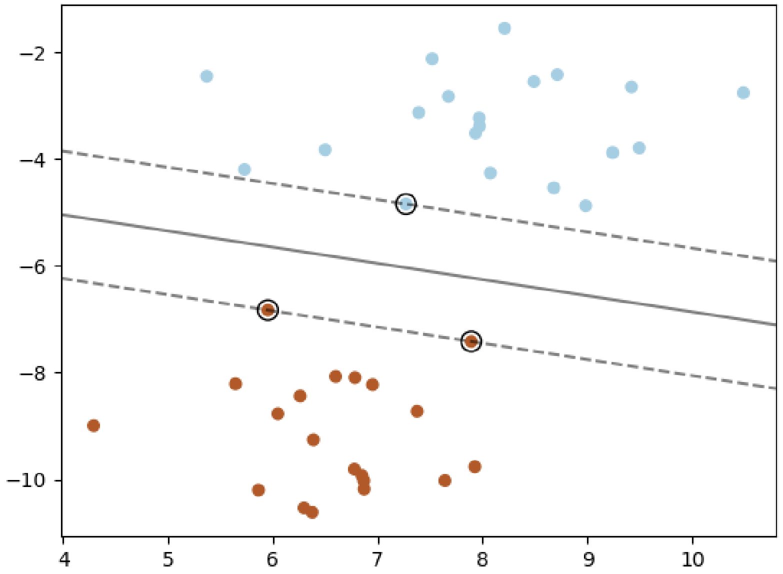
Margin and Support Vectors

Over all 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***.

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



Maximum margin classifier

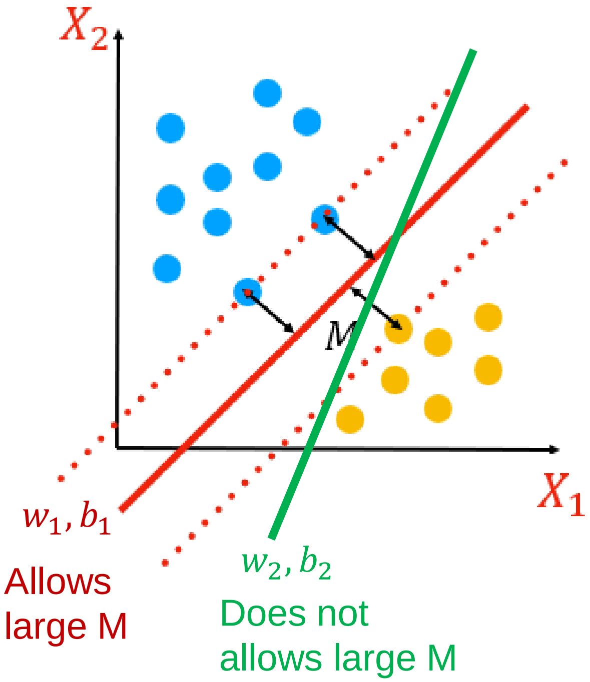
We can formulate finding a maximum margin classifier as an *optimization problem*:

Find $w, b, M \geq 0$ such that

$$\text{maximize } M$$

with the constraints that

$$\frac{y_i(w \cdot x_i + b)}{||w||_2} \geq M \quad \text{for all } i$$



Math Interlude: optimization problems

- The above falls to the general form of

$$\text{maximize } f(x)$$

subject to

$$g_i(x) \le 0,\ i = 1, \dots, m$$

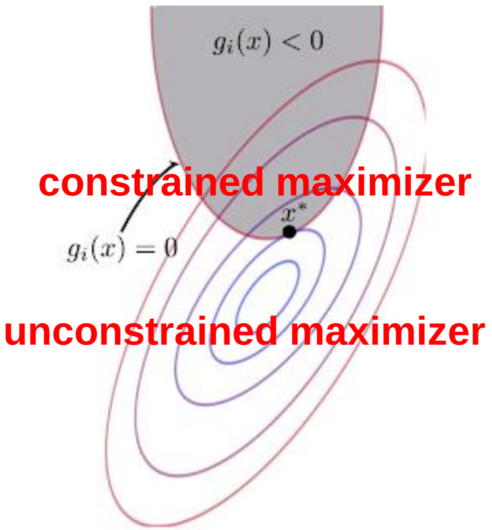
x: Optimization variables

constraints

- These are called constrained optimization problems

- Due to the constraints, finding the maximizer requires more care..

- Still, solvable by many standard packages

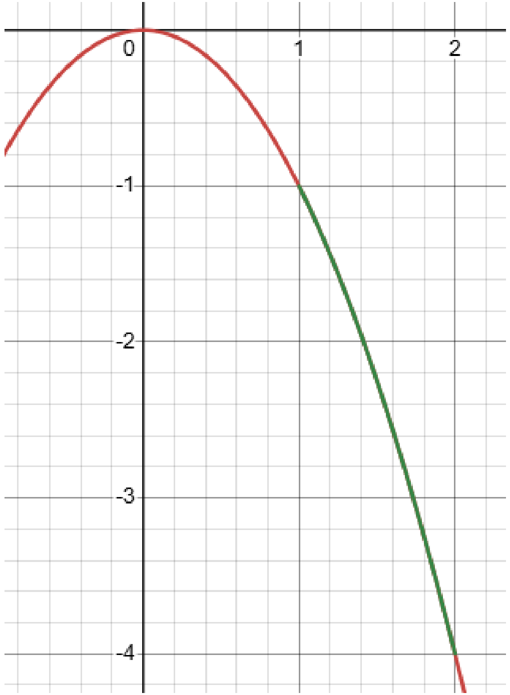


Math Interlude: optimization problems

Example Find the solution of

$$\text{maximize } -x^2 \text{ subject to } x \geq 1 \text{ and } x \leq 3$$

Solution We can draw a picture..

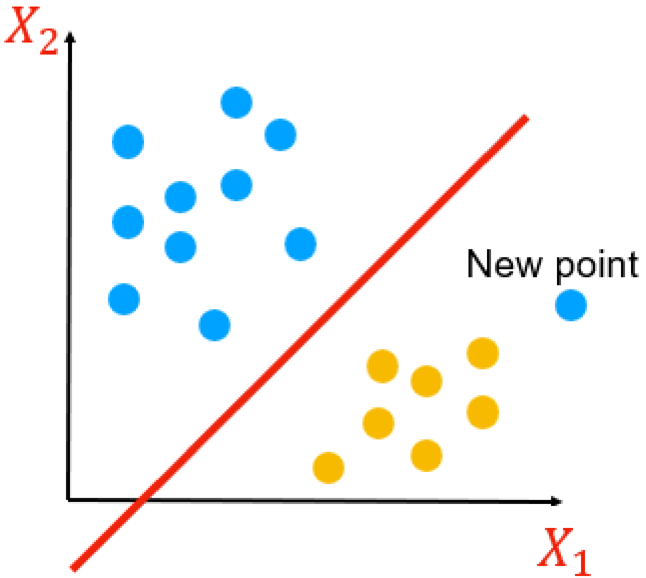


The objective is maximized at $x = 1$

Note: the constrained maximizer is **not** the vertex of the parabola

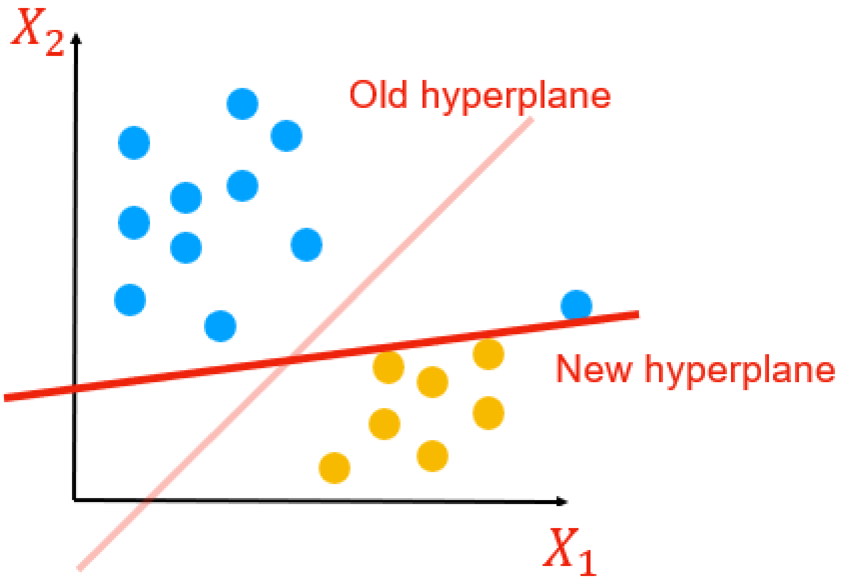
Support vector machine: extension

The maximum margin solution can be sensitive to outliers



Support vector machine: extension

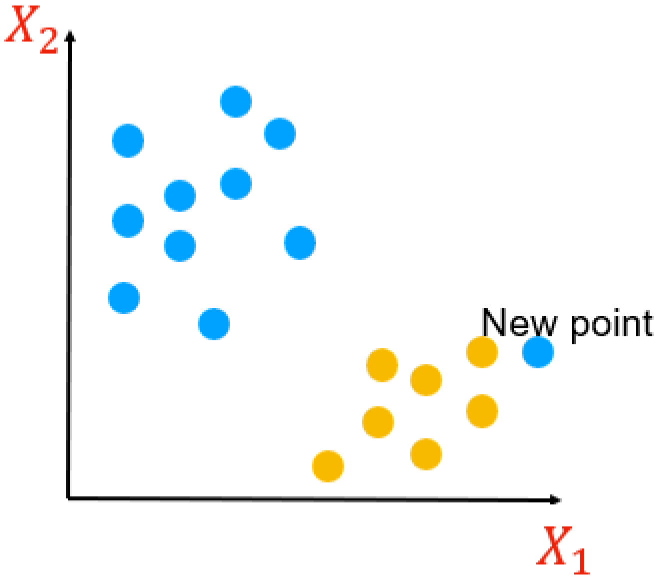
The maximum margin solution can be sensitive to outliers



Maybe prone to overfitting!

Support vector machine: extension

- The maximum margin solution may not even exist



No separating hyperplane (line in 2D)

Perhaps requiring the output classifier to predict every example correctly is too strict?

requirement of “hard margins”

Solution: soft margins – allow mistakes on some training examples

Soft margin support vector machines

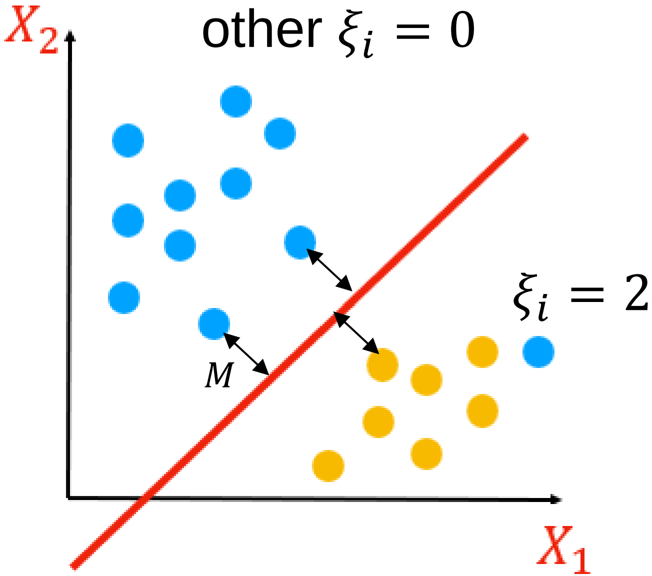
Find w, b, M, such that

$$\text{maximize } M$$

with the constraints that

$$\frac{y_i(w \cdot x_i + b)}{||w||_2} \geq M(1 - \xi_i) \quad \text{for all } i$$

and $\xi_i \geq 0$, $\sum_i \xi_i \leq C$



ξ_i: slack variables

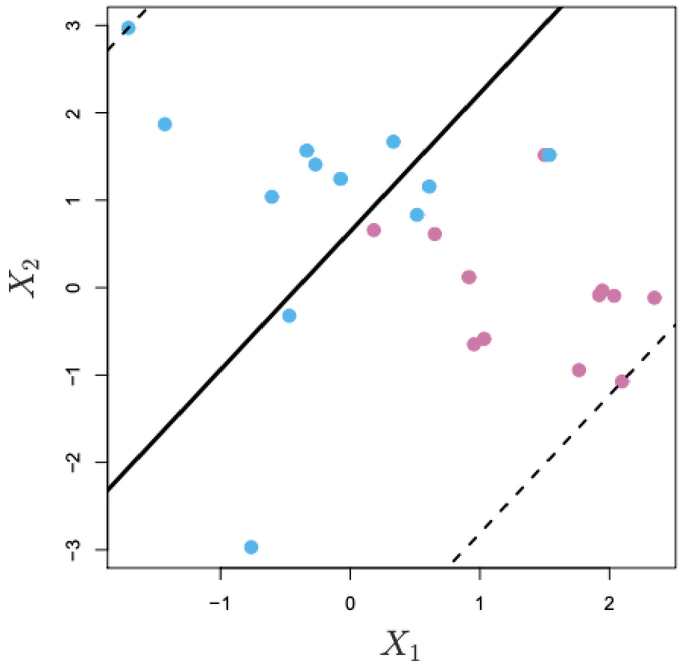
allows some examples to be on the wrong side

C: # in-margin examples allowed

Soft margin support vector machines

- Large C

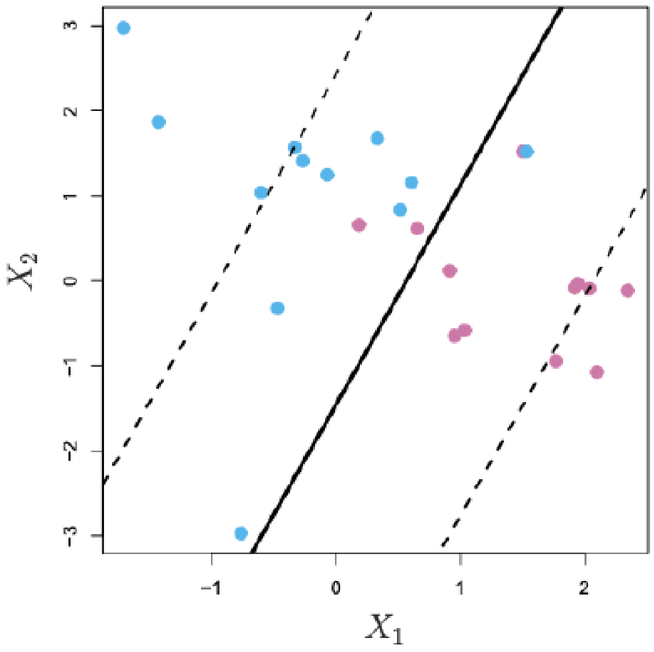
Many points inside the margin, many points on the wrong side of the line



Soft margin support vector machines

- Smaller C

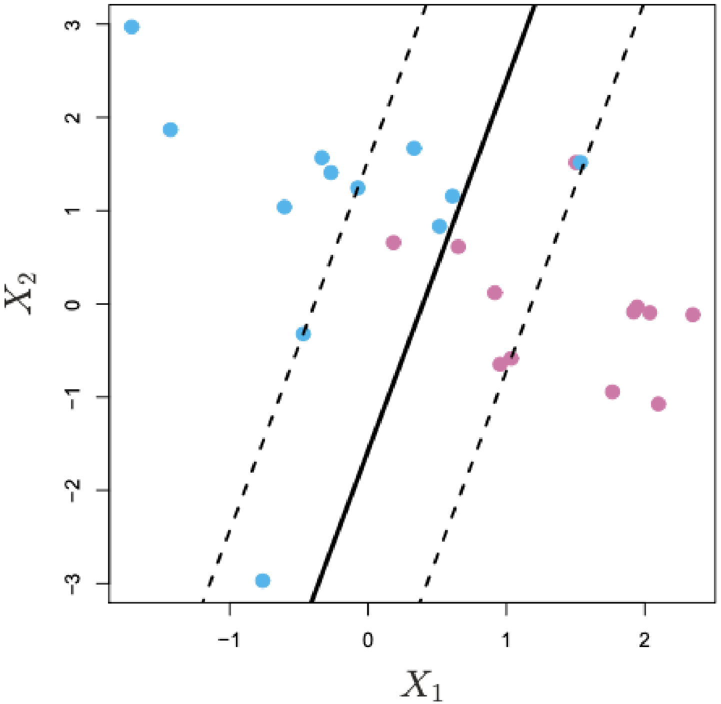
Fewer points inside the margin,
Fewer points on the wrong side of the line



Soft margin support vector machines

- Even smaller C

Even fewer points inside the margin,
Very few points on the wrong side
of the line



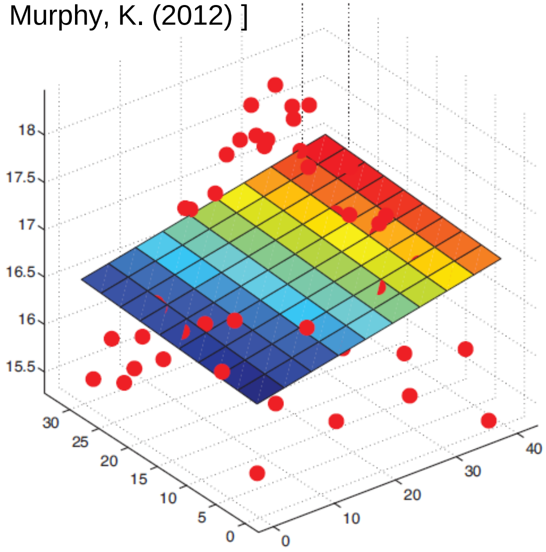
Smaller C => More overfitting => Lower bias, higher complexity
As usual, we can choose C by cross validation

Nonlinear classification models

Nonlinear basis functions; kernels

Linear Models

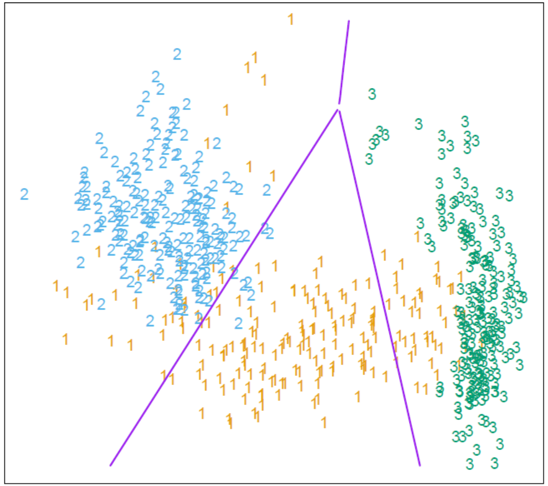
[Image: Murphy, K. (2012)]



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

$$y = w^T x + b$$

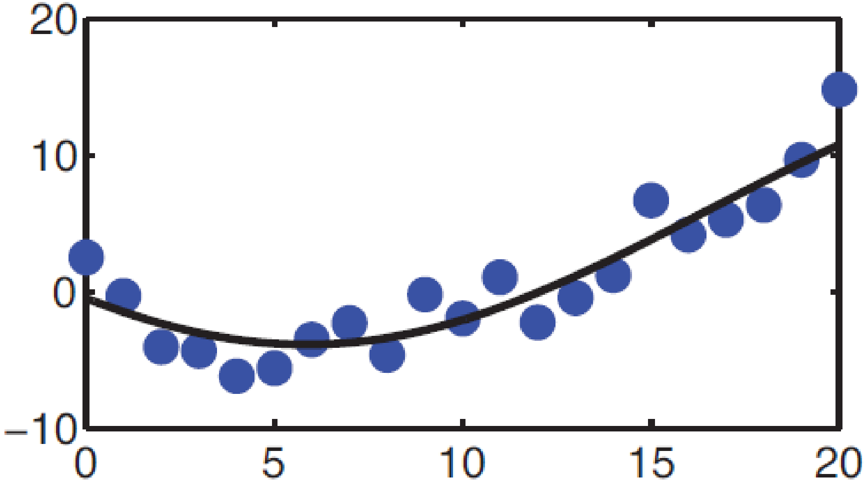
[Image: Hastie et al. (2001)]

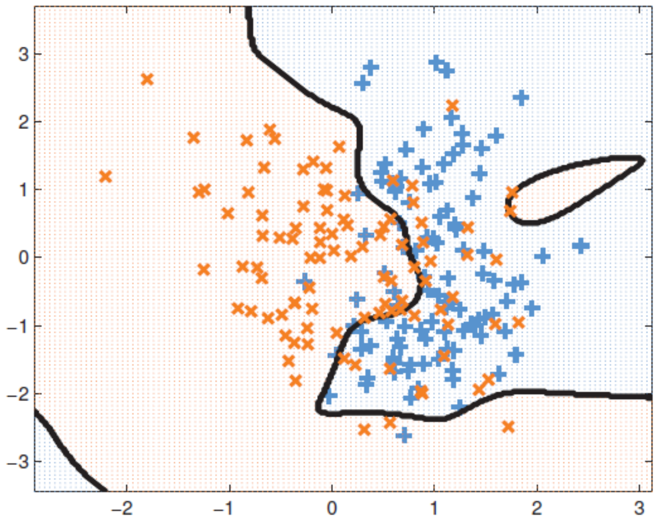


Logistic Regression Learn a decision boundary that is *linear in the data*,

$$P(y = 1 \mid w, x) = \sigma(w^T x)$$

Nonlinear Data





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

What if classes are *not* *linearly-separable*?

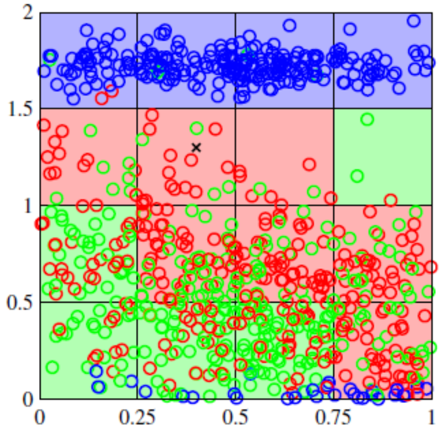
[Source: Murphy, K. (2012)]

Nonlinear prediction problems

- Nearest neighbor methods are OK, but they suffer from *the curse of dimensionality*

In high dimensions, all points are (kind-of) far from each other

For high-dimensional data, most cells are empty!



Alternative approach:

We can *reduce* learning nonlinear models

to learning linear models

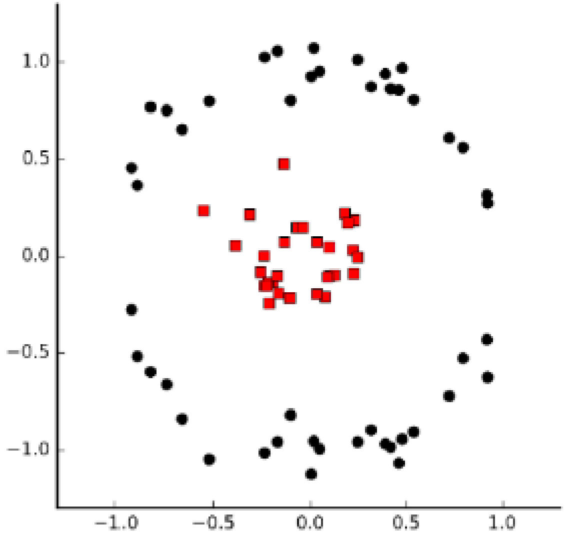
Basis Functions

- A **basis function** can be any function of the input features X
- Define a set of m basis functions $\phi_1(x), \ldots, \phi_m(x)$
- Fit a linear model in terms of basis functions,

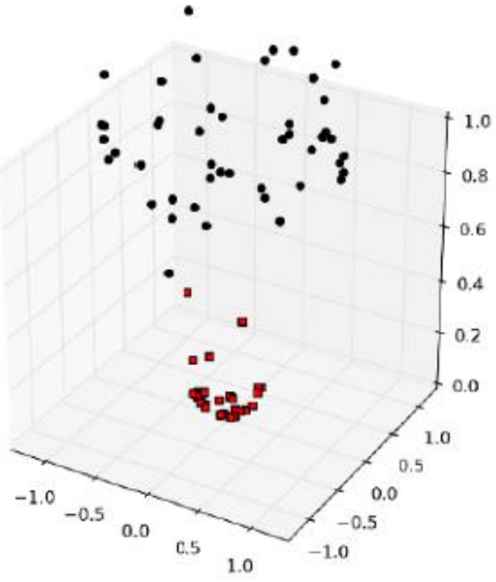
$$f(x) = \sum_{i=1}^{m} w_i \phi_i(x) = w^T \phi(x)$$

- Model is *linear in the basis transformations*
- Model is *nonlinear in the data X*

Why do Basis Functions help?



$$\phi(x_1, x_2) = (x_1, x_2, x_1^2 + x_2^2)$$



Not Linearly separable

Linearly separable

Common "All-Purpose" Basis Functions

- Linear basis functions recover the original linear model,

$$\phi_m(x) = x_m$$

Returns m^{th} dimension of X

- Quadratic $\phi_m(x) = x_j^2$ or $\phi_m(x) = x_j x_k$ capture 2nd order interactions
- An order p polynomial $\phi \to x_d, x_d^2, \ldots, x_d^p$ captures higher-order nonlinearities (but requires O(d^p) parameters)
- Nonlinear transformation of single inputs,

$$\phi \to (\log(x_j), \sqrt{x_j}, \ldots)$$

- An indicator function specifies a region of the input,

$$\phi_m(x) = I(L_m \leq x_k < U_m)$$

`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 1: Polynomial Basis Functions

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 $(1, x_1, x_2, x_1^2, x_1 x_2, x_2^2)$,

```
>>> 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 2: Polynomial Regression

Create a 3rd order polynomial (cubic) regression data,

```
from sklearn.preprocessing import PolynomialFeatures
x = np.arange(5)
y = 3 - 2 * x + x ** 2 - x ** 3
y
```

```
array([  3,   1,  -5, -21, -53])
```

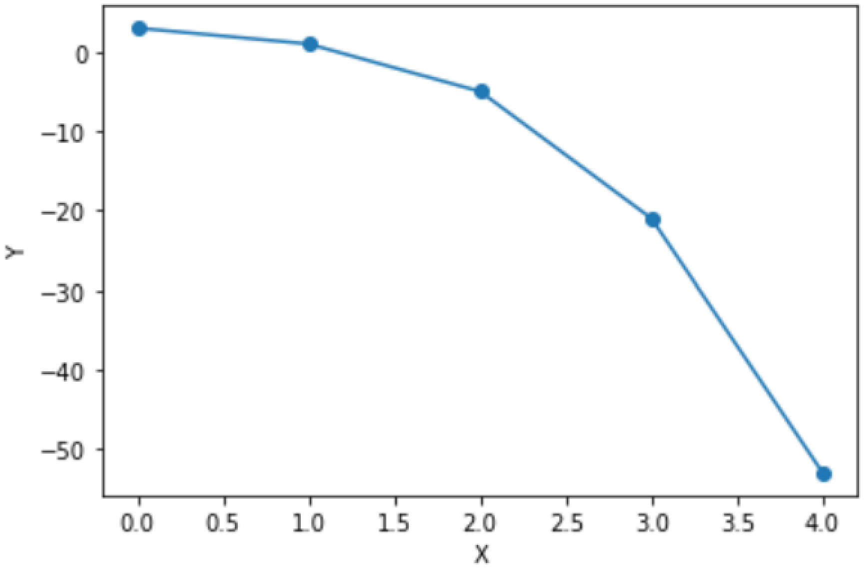
Create cubic features $(1, x, x^2, x^3)$,

```
from sklearn.linear_model import LinearRegression
poly = PolynomialFeatures(degree=3)
x_new = poly.fit_transform(x[:,np.newaxis])
x_new
```

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

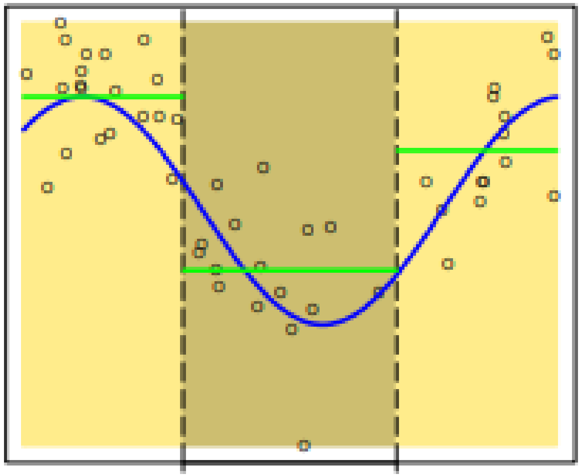
Example: Polynomial Regression

```
model = LinearRegression(fit_intercept=False).fit(x_new, y)
ypred = model.predict(x_new)
plt.scatter(x,y)
plt.plot(x,ypred,'-')
plt.xlabel('X')
plt.ylabel('Y')
plt.show()
```



Example: Piecewise Constant Regression

[Source: Hastie et al. (2001)]



Decompose the input space into 3 regions with indicator basis functions,

$$\phi_1(x) = I(x < \xi_1)$$
$$\phi_2(x) = I(\xi_1 \leq x < \xi_2)$$
$$\phi_3(x) = I(\xi_2 \leq x)$$

Fit linear regression model,

$$y = w_1\phi_1(x) + w_2\phi_2(x) + w_3\phi_3(x)$$

Effectively fits 3 constant functions to data in each region

Kernels

Fact Many machine learning algorithms output linear models of the form $w = \sum_i \alpha_i x_i$ and thus makes prediction by

Sometimes called 'dual variables'

$$\sum_i \alpha_i x_i \cdot x + b$$

Examples: SVM, logistic regression

when learning with basis functions, the trained models make prediction by

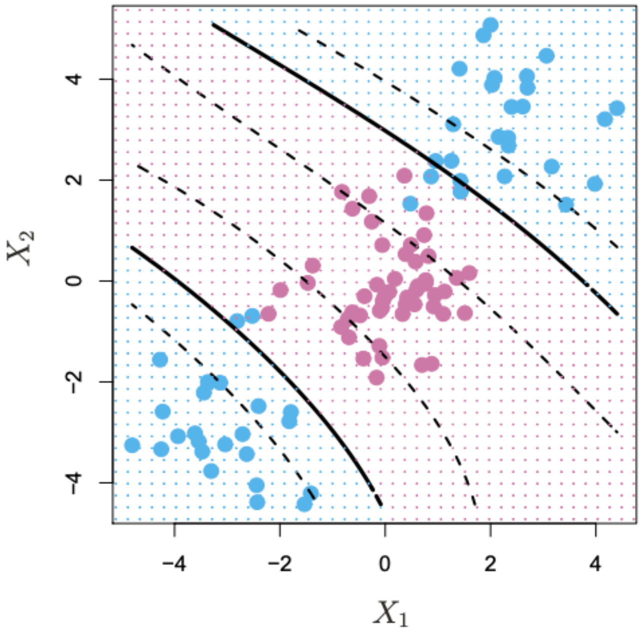
$$\sum_i \alpha_i \phi(x_i) \cdot \phi(x) + b$$

kernel: generalizes inner products;
captures similarity between examples

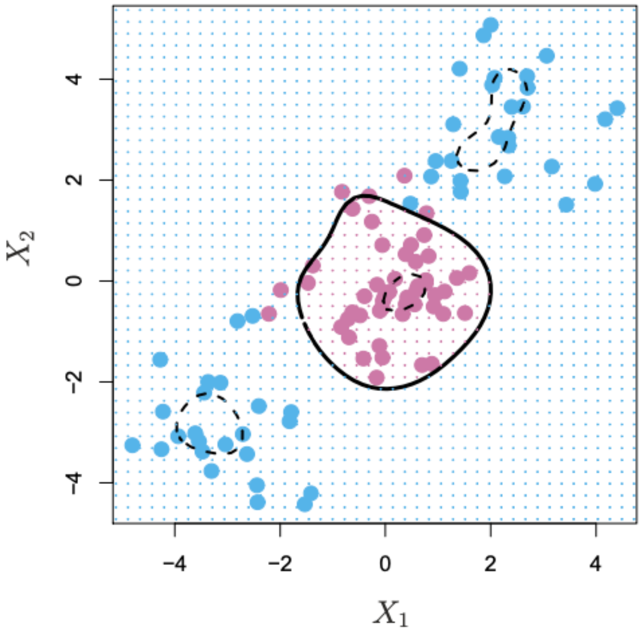
popular kernels: polynomial, radial

Kernel SVM

Applying kernel SVMs to nonlinear data



polynomial (d=3) kernel



radial kernel

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 / (n_features * X.var()) as value of gamma,
- if 'auto', uses 1 / n_features.

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 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_samples / (n_classes * np.bincount(y))`.

Example: Fisher's Iris Dataset

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,

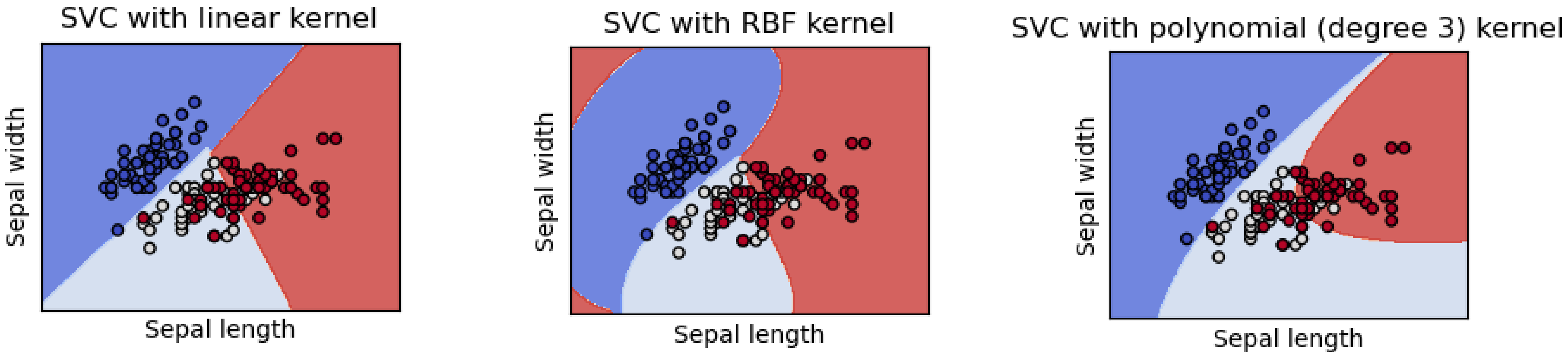
```
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
[[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/]

Kernel SVM in Scikit Learn



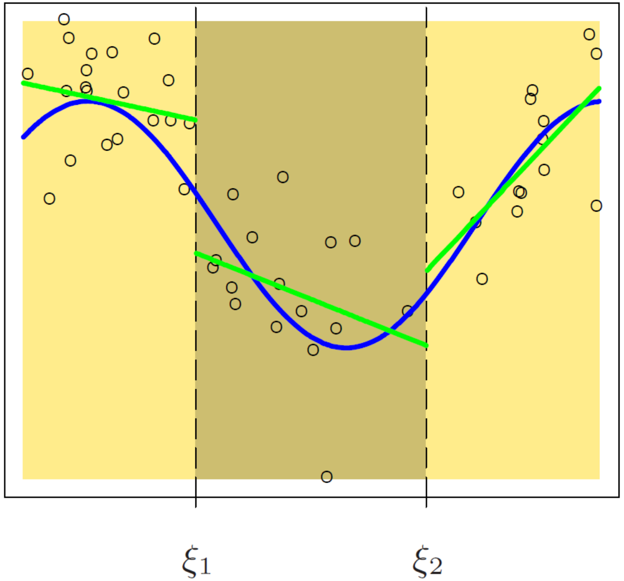
- General kernel-based SVM lives in:

  `sklearn.svm.svc(kernel='kernel_name')`

Backup

Example: Piecewise Linear Regression

[Source: Hastie et al. (2001)]



Regression lines are discontinuous at boundary points

Decompose the input space into 3 regions with basis functions,

$$\phi_1(x) = I(x < \xi_1) \qquad \phi_4(x) = xI(x < \xi_1)$$

$$\phi_2(x) = I(\xi_1 \leq x < \xi_2) \qquad \phi_5(x) = xI(\xi_1 \leq x < \xi_2)$$

$$\phi_3(x) = I(\xi_2 \leq x) \qquad \phi_6(x) = xI(\xi_2 \leq x)$$

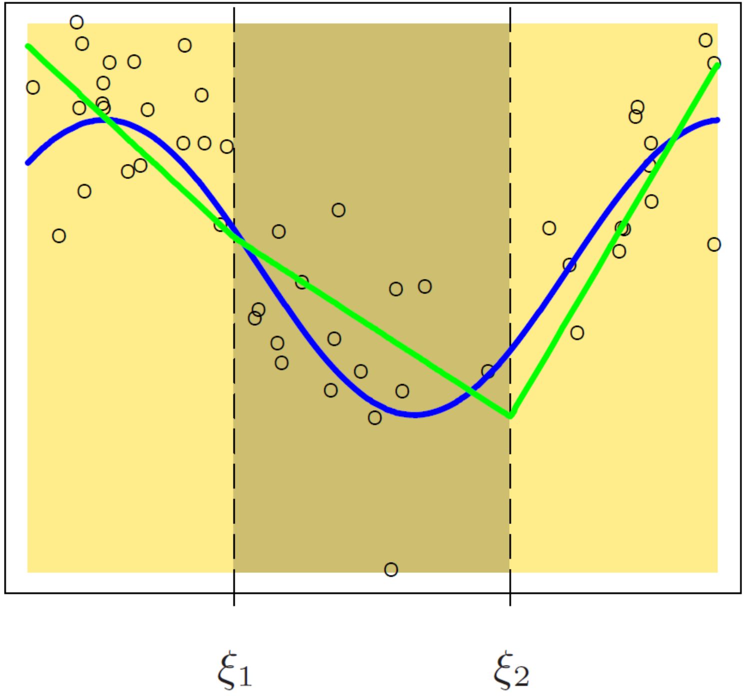
Fit linear regression model,

$$y = \sum_{i=1}^{6} w_i \phi_i(x)$$

Effectively fits 3 linear regressions independently to data in each region

Example: Piecewise Linear Regression

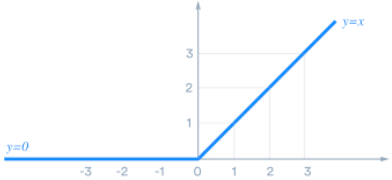
[Source: Hastie et al. (2001)]



Enforce constraint that lines agree at boundary points,

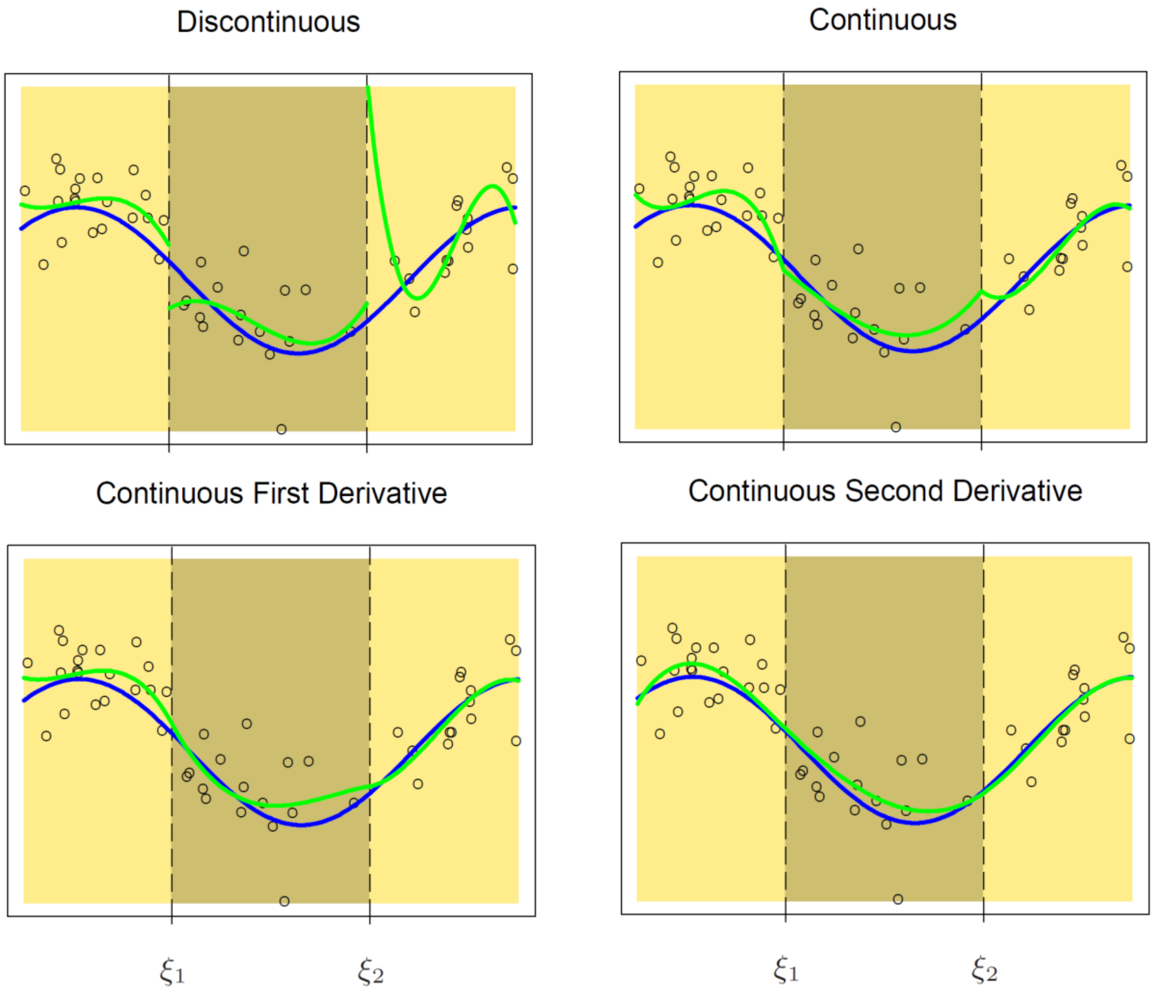
$$\phi_1(x) = 1$$
$$\phi_2(x) = x$$
$$\phi_3(x) = (x - \xi_1)_+$$
$$\phi_4(x) = (x - \xi_2)_+$$

Where $(z)_+ := \max(z, 0)$



An improvement, but generally prefer *smoother* functions…

[Source: Hastie et al. (2001)]



Replace linear basis functions with polynomial,

$$\phi_1(x) = 1 \quad \phi_2(x) = x$$

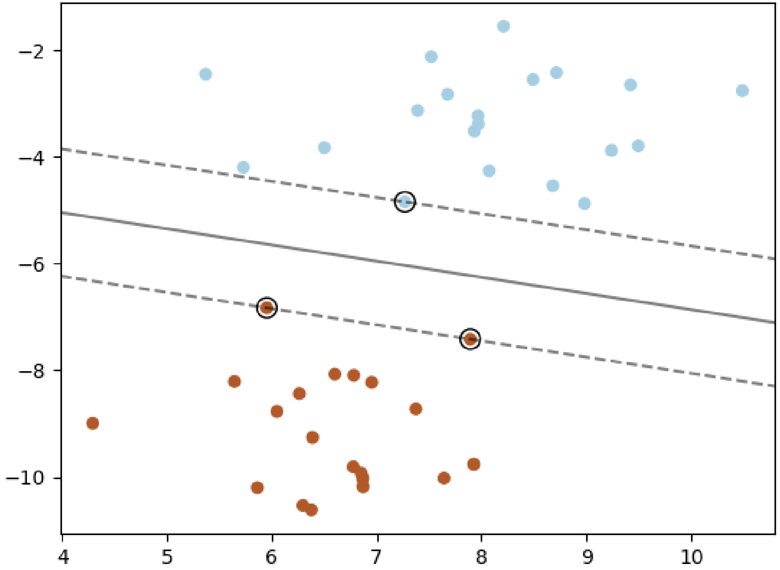
$$\phi_3(x) = x^2 \quad \phi_4(x) = x^3$$

$$\phi_5(x) = (x - \xi_1)_+^3$$

$$\phi_6(x) = (x - \xi_2)_+^3$$

Additional constraints ensure smooth 1st and 2nd derivatives at boundaries

Max-Margin Classifier (Linear Separable Case)



Maximize the minimum margin

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

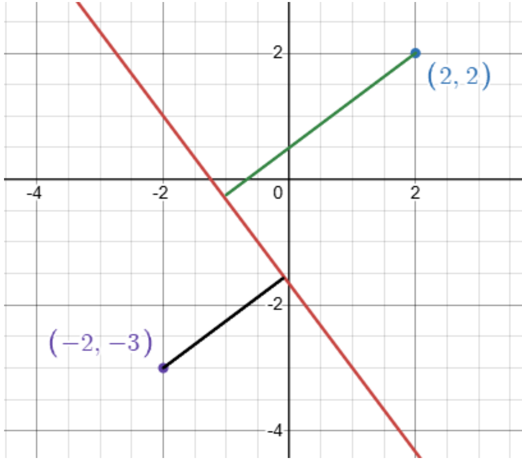
Minimum margin over all training data

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

Normalized margin

Given linear classifier $w \cdot x + b$, its *normalized classification margin* on *labeled example* (x, y) is defined as

$$\frac{y(w \cdot x + b)}{||w||_2}$$

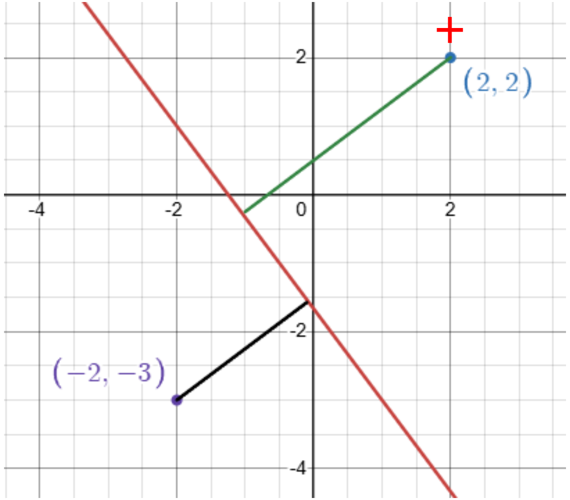


Interpretation how correct

Classification margin

Given linear classifier $w \cdot x + b$, its *classification margin* on *labeled example* (x_i, y_i) is defined as $y_i\,(w \cdot x_i + b)$

$0.8x_1 + 0.6x_2 + 1$



Example for example (2,2) with label +,

margin = $+(0.8 \times 2 + 0.6 \times 2 + 1) = 3.8$

for example (-2,-3) with label -,

margin = $-(0.8 \times (-2) + 0.6 \times (-3) + 1) = 2.4$

for example (2,2) with label -:

margin = $-(0.8 \times 2 + 0.6 \times 2 + 1) = -3.8$