CPSC 340:
Machine Learning and Data Mining

Kernel Methods

Fall 2015

Admin

- Assignment 3 due Friday:
  - Submit as a single PDF file.
- Practice midterm coming this weekend.
- Monday tutorials:
  - Go through practice midterm.
- Midterm next Friday, October 30.
  - In class, 55 minutes, closed-book, cheat sheet: 2-pages each double-sided.

Regression Framework: (Loss) + (Regularizer)

- Framework for regression models:

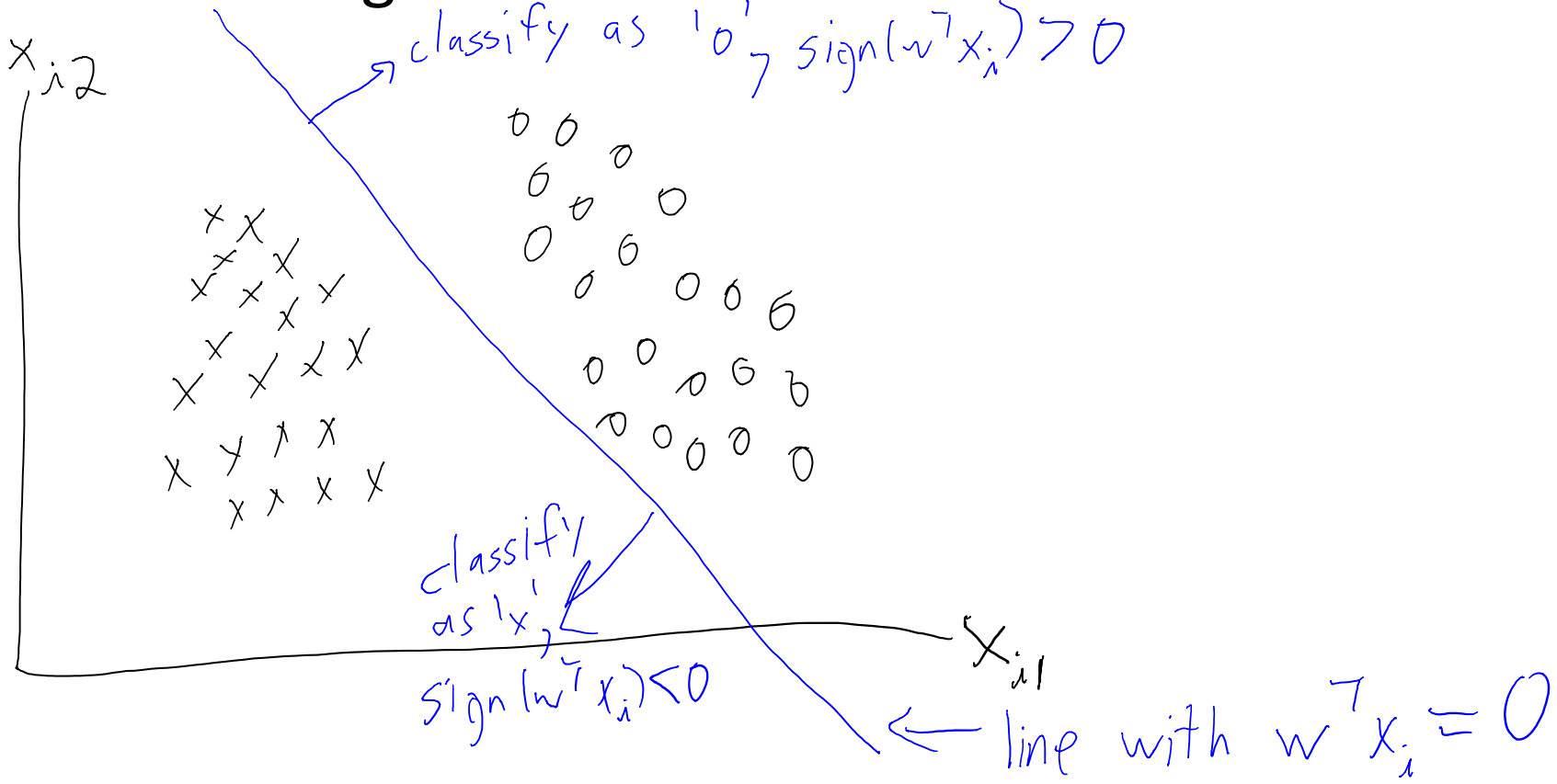
$$\underset{w \in \mathbb{R}^d}{\operatorname{argmin}} \sum_{i=1}^{n} g(y_i w^T x_i) + \lambda r(w).$$

- **Loss function** 'g':
  - Squared error (default choice): $(y_i - w^T x_i)^2$
  - Absolute error (robust to outliers): $|y_i - w^T x_i|$
    - Smooth approximation is Huber loss.
  - 0-1 loss (classification – not convex) $\mathbb{I}[y_i \neq \text{sign}(w^T x_i)]$
    - Convex approximation is hinge loss): $\max\{0, 1 - y_i w^T x_i\}$
    - Smooth approximation is logistic loss.
- **Regularization function** 'r':
  - L2-regularization (default choice): $\frac{1}{2}\|w\|^2$
  - L1-regularization (feature selection): $\|w\|_1$

We've also discussed choosing features x_i:
- standardization
- adding a bias
- polynomial basis
- RBFs (consistent)
- global vs. local
- feature selection

2D View of Linear Classifiers

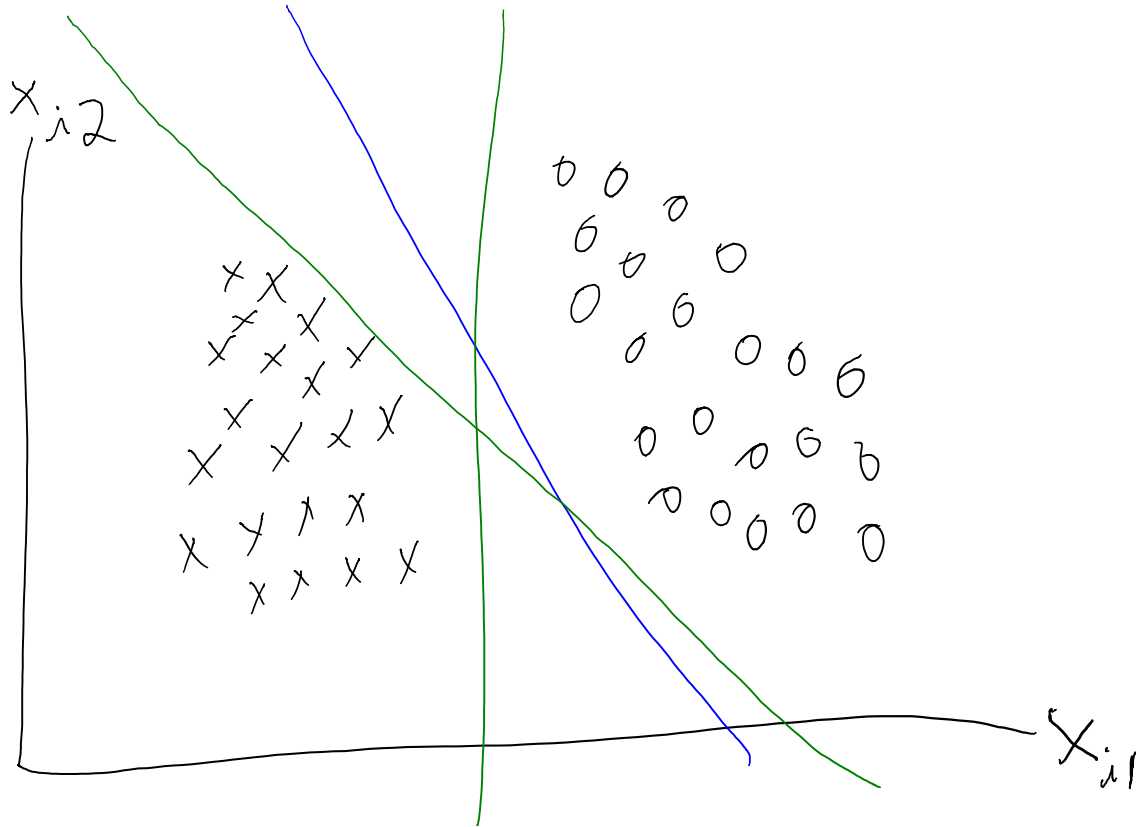
- 2D Visualization of linear regression for classification:



- Linearly separable: a perfect linear classifier exists.

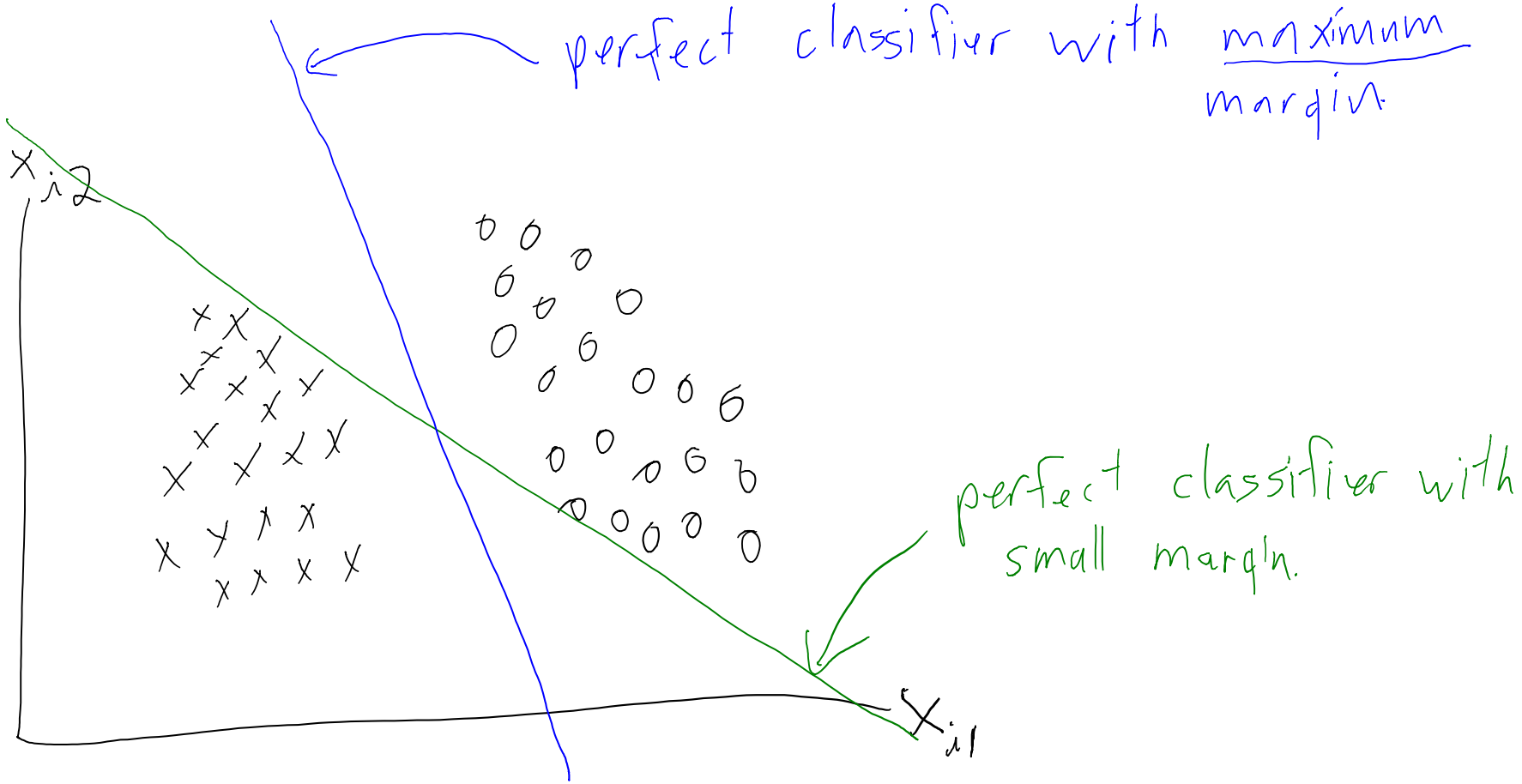
Maximum-Margin Classifier

- Consider a linearly-separable dataset.
  - 'Perceptron' algorithm finds *some* classifier with zero error.
  - But are all zero-error classifiers equally good?



Maximum-Margin Classifier

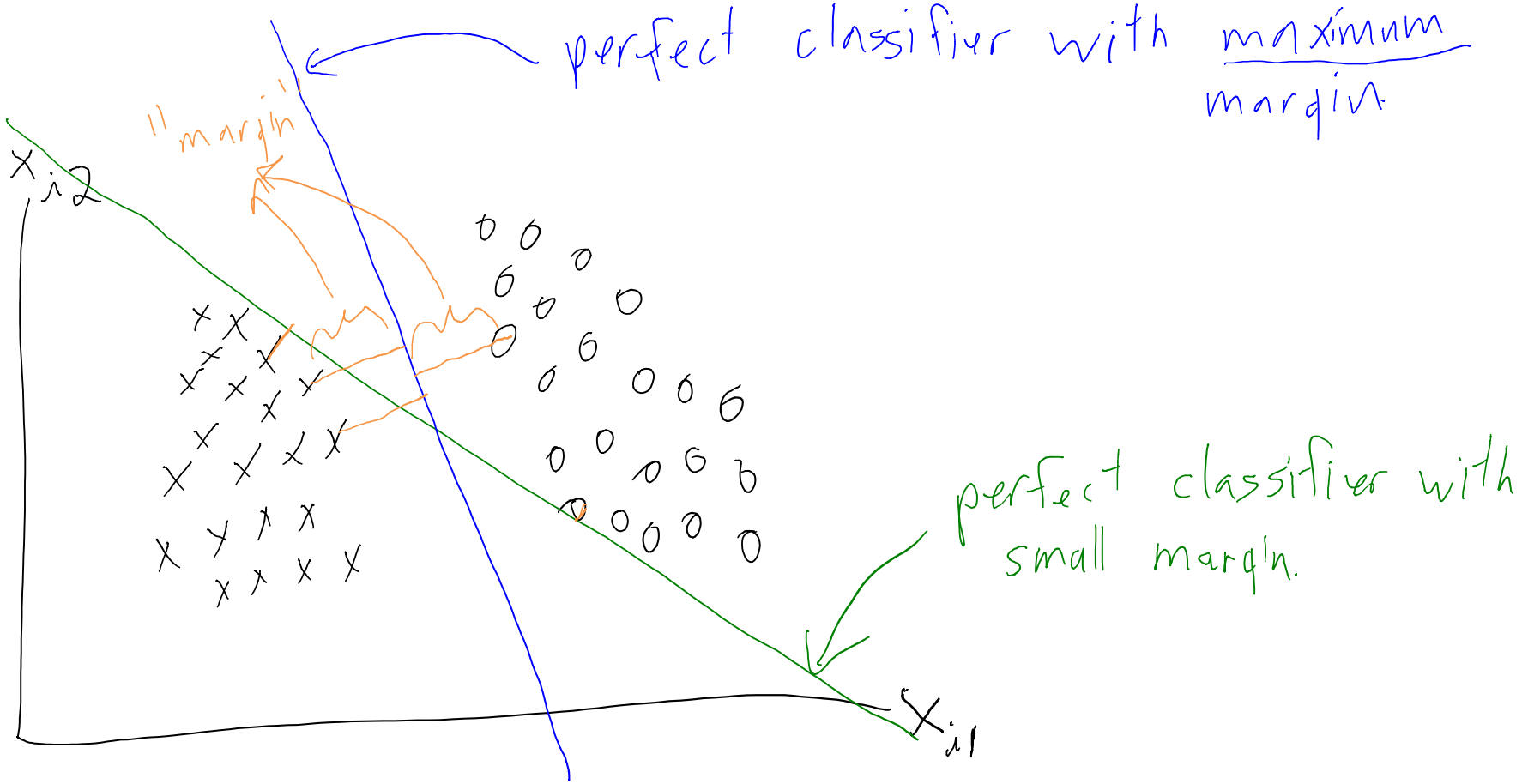
- Consider a linearly-separable dataset.
  - Maximum-margin classifier: choose the farthest from both classes.



Maximum-Margin Classifier

- Consider a linearly-separable dataset.
  - Maximum-margin classifier: choose the farthest from both classes.

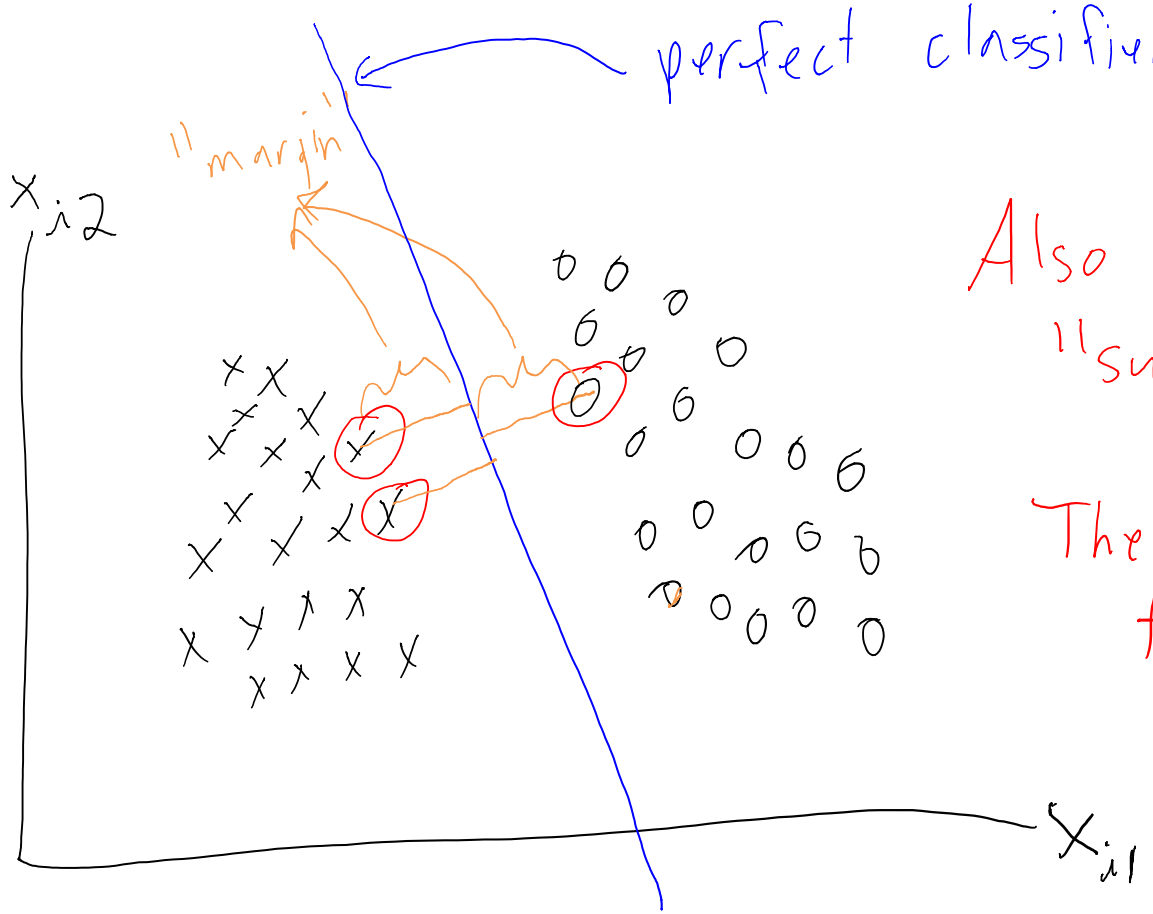
If new data is close to old data, expect max-margin to do better.



Maximum-Margin Classifier

- Consider a linearly-separable dataset.
  - Maximum-margin classifier: choose the farthest from both classes.

If new data is close to old data, expect max-margin to do better.



Also known as "support vector machine" (SVM).

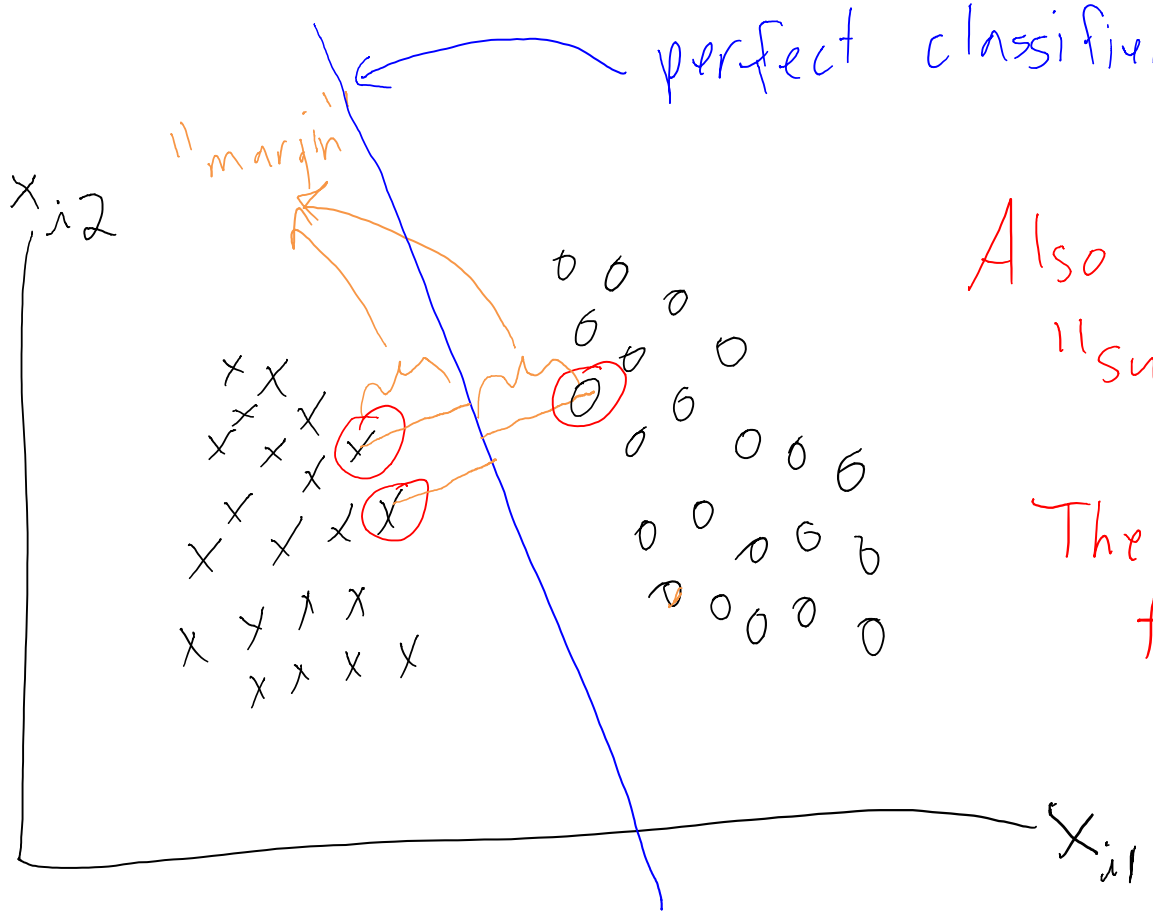
The closest examples are the "support vectors."

Maximum-Margin Classifier

- Consider a linearly-separable dataset.
  - Maximum-margin classifier: choose the farthest from both classes.

If new data is close to old data, expect max-margin to do better.

Max-margin classifier only depends on support vectors.



Also known as "support vector machine" (SVM).

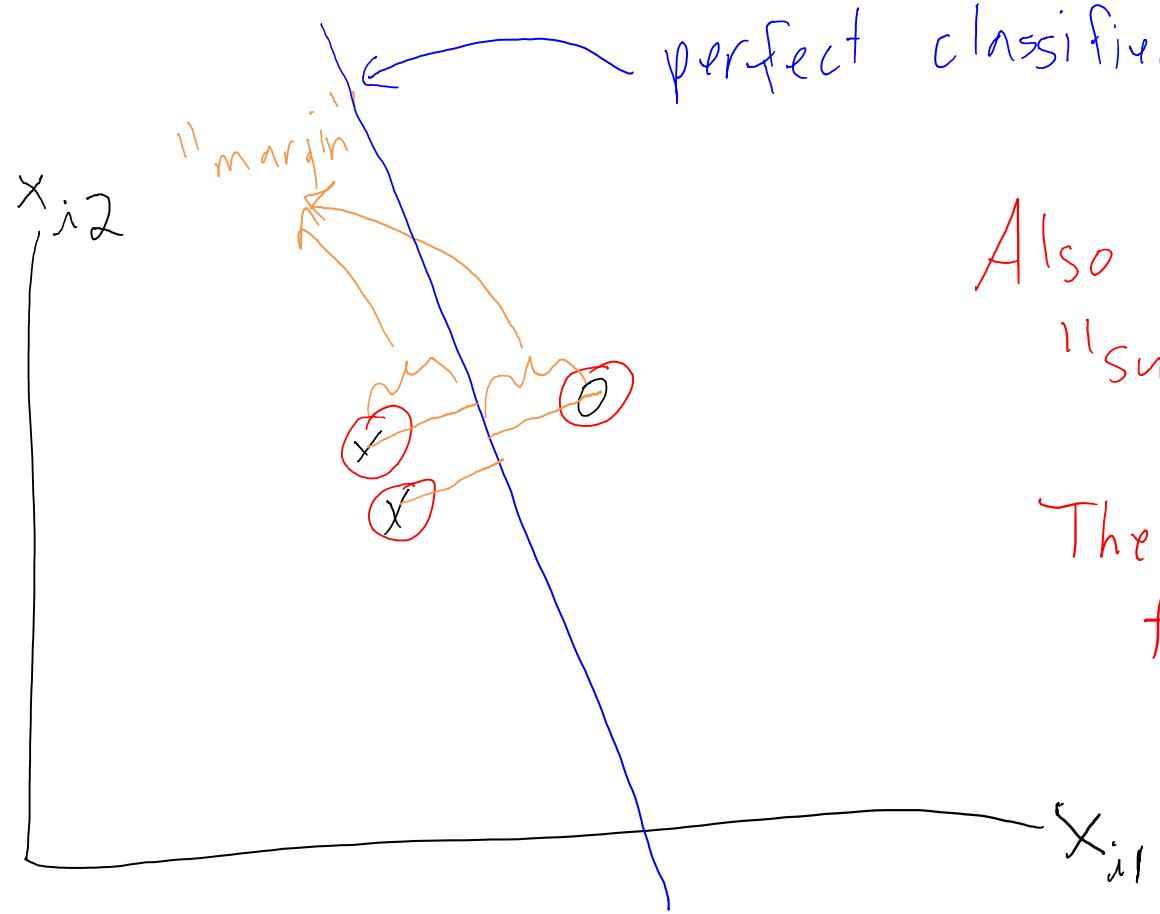
The closest examples are the "support vectors".

Maximum-Margin Classifier

- Consider a linearly-separable dataset.
  - Maximum-margin classifier: choose the farthest from both classes.

If new data is close to old data, expect max-margin to do better.

Max-margin classifier only depends on support vectors.



Also known as "support vector machine" (SVM).

The closest examples are the "support vectors".

Support Vector Machine

- For linearly-separable data, max-margin classifier is solution of

$$\underset{w \in \mathbb{R}^d}{\operatorname{argmin}} \sum_{i=1}^{n} \mathbb{I}[y_i \neq \operatorname{sign}(w^T x_i)] + \frac{\lambda}{2}\|w\|^2$$

(for sufficiently small λ)

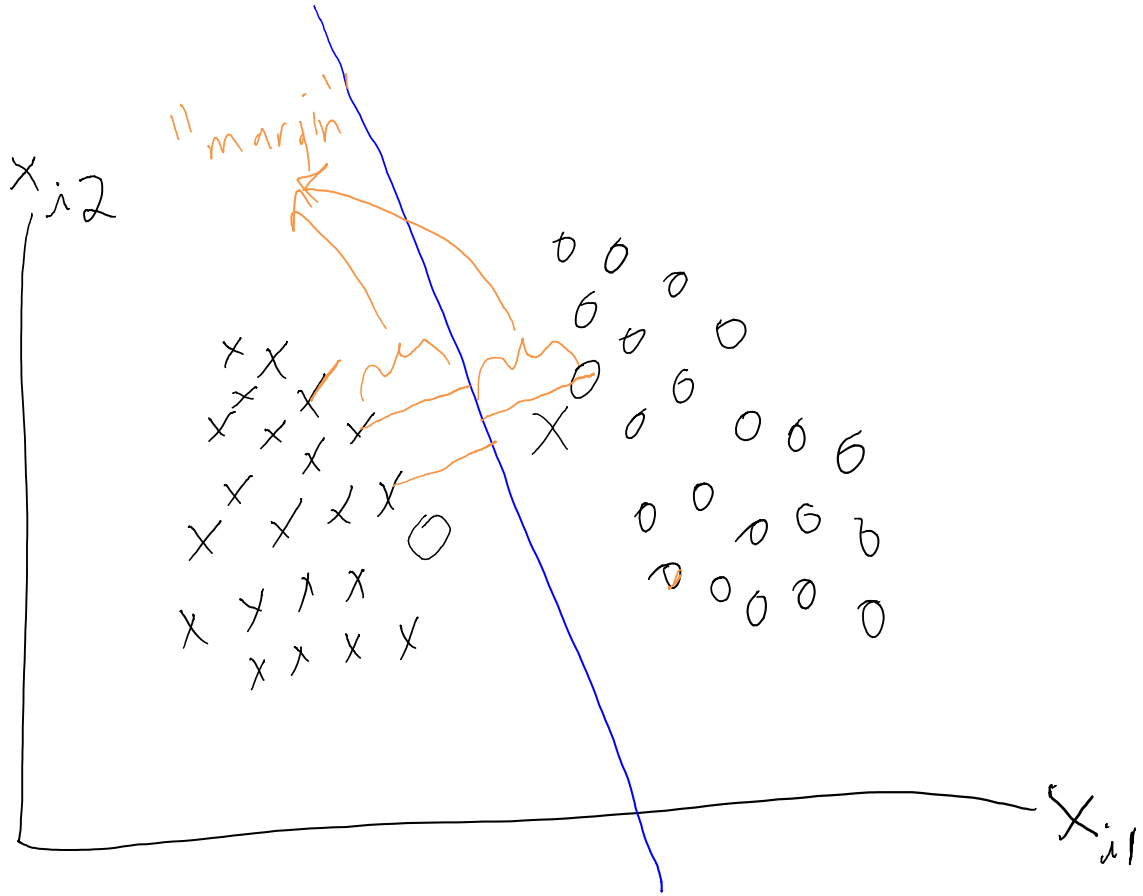
- Maximum margin classifier (SVM) is 0-1 loss with L2-regularization.
- For non-separable data, usually use L2-regularized hinge loss:

$$\underset{w \in \mathbb{R}^d}{\operatorname{argmin}} \sum_{i=1}^{n} \max\{0, 1 - y_i w^T x_i\} + \frac{\lambda}{2}\|w\|^2$$

- This is the standard SVM formulation.
  - Can approximate it with L2-regularized logistic regression.

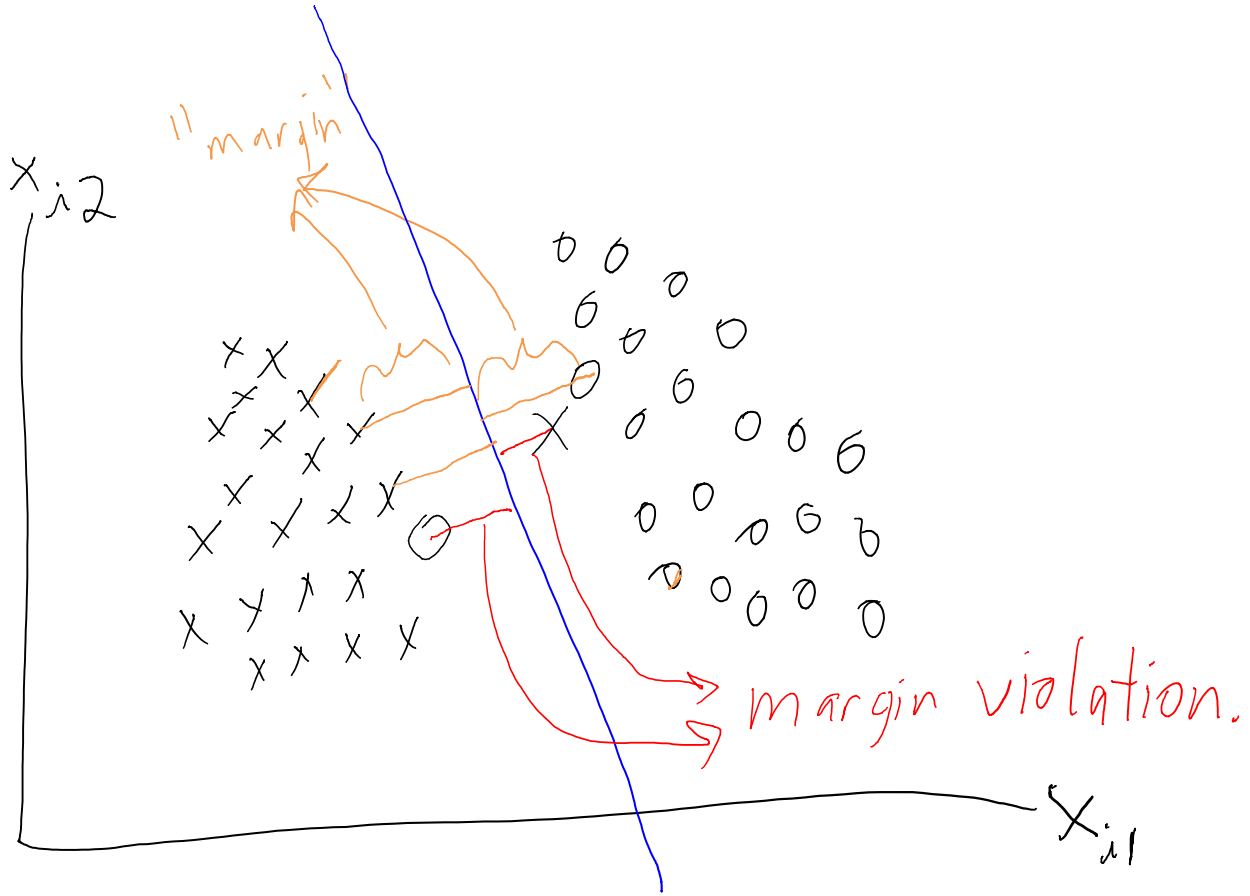
Support Vector Machines for Non-Separable

- Non-separable case:



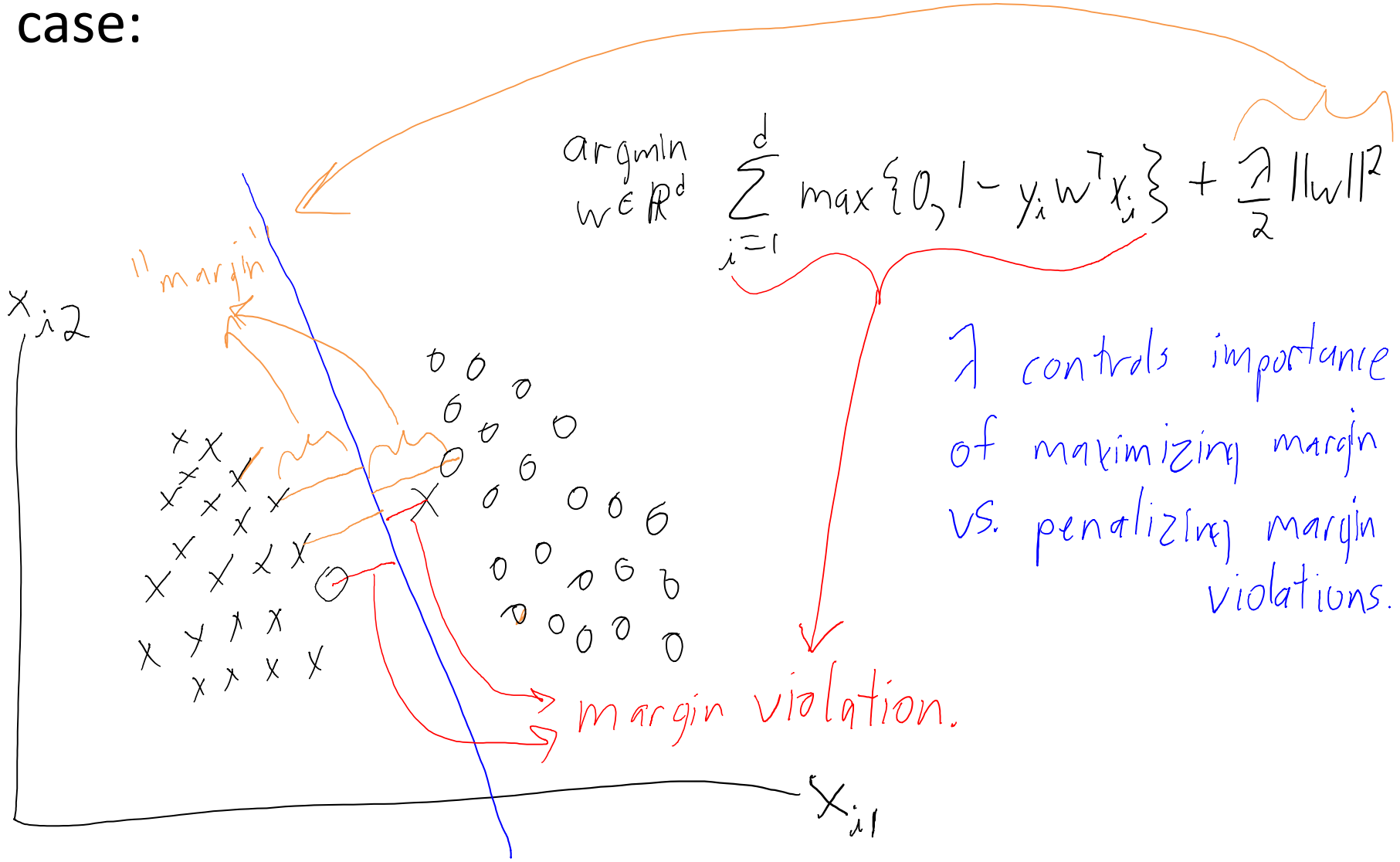
Support Vector Machines for Non-Separable

- Non-separable case:



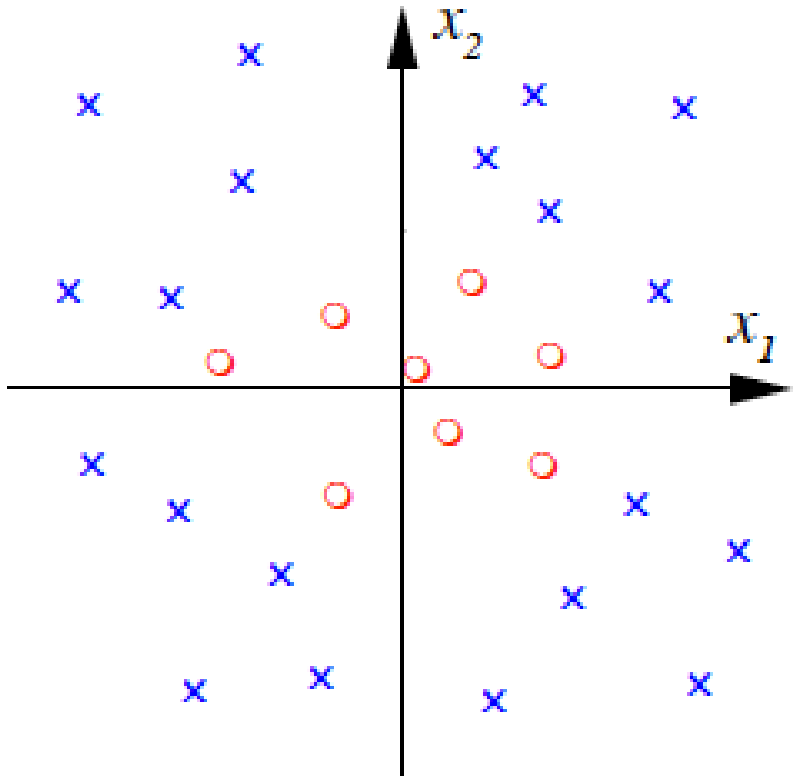
Support Vector Machines for Non-Separable

- Non-separable case:



Support Vector Machines for Non-Separable

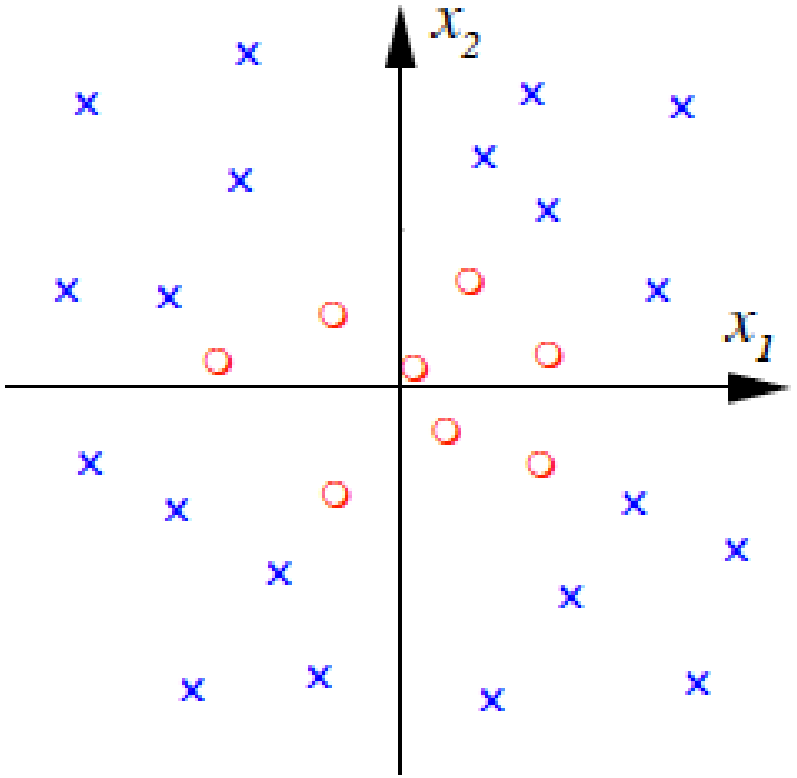
- What about data that is not even close to separable?

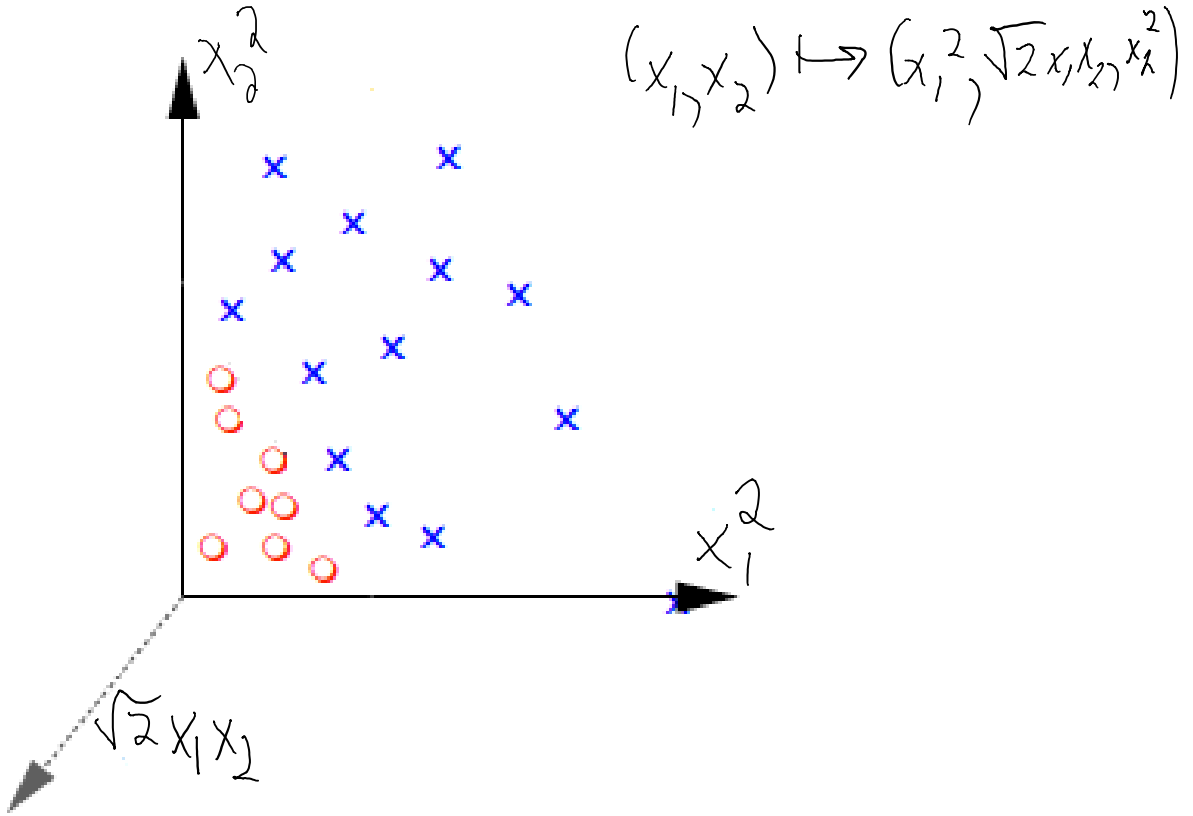


http://math.stackexchange.com/questions/353607/how-do-inner-product-space-determine-half-planes

Support Vector Machines for Non-Separable

- What about data that is not even close to separable?
  - It may be separable under change of basis (or almost separable).

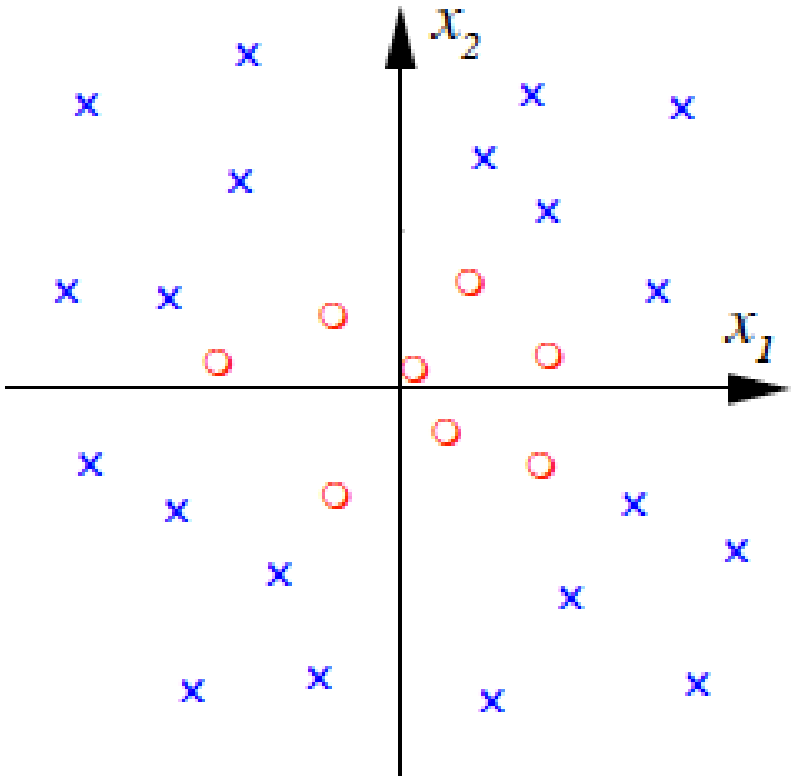


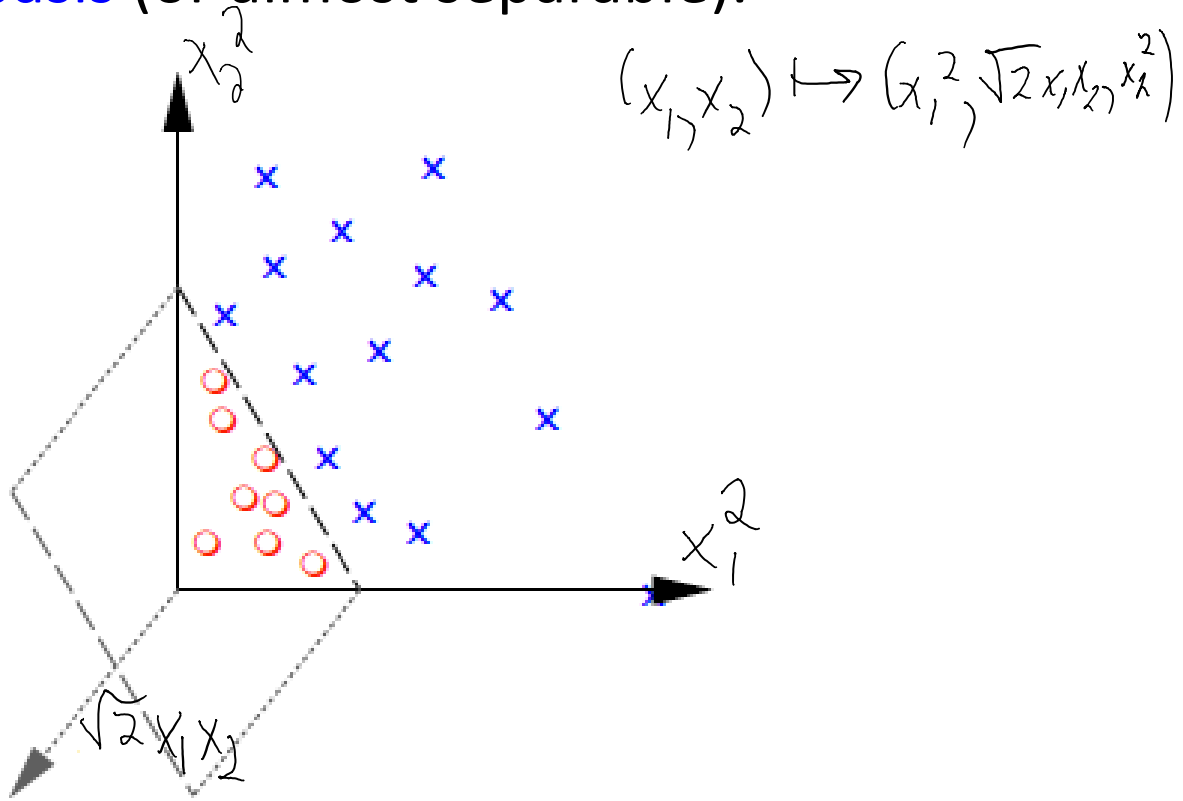


http://math.stackexchange.com/questions/353607/how-do-inner-product-space-determine-half-planes

Support Vector Machines for Non-Separable

- What about data that is not even close to separable?
  - It may be separable under change of basis (or almost separable).

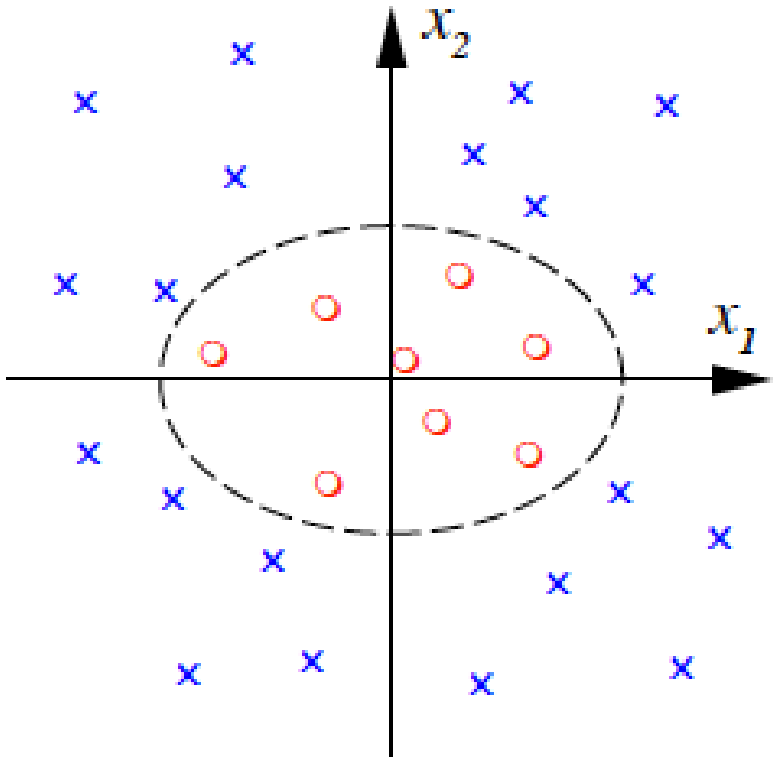


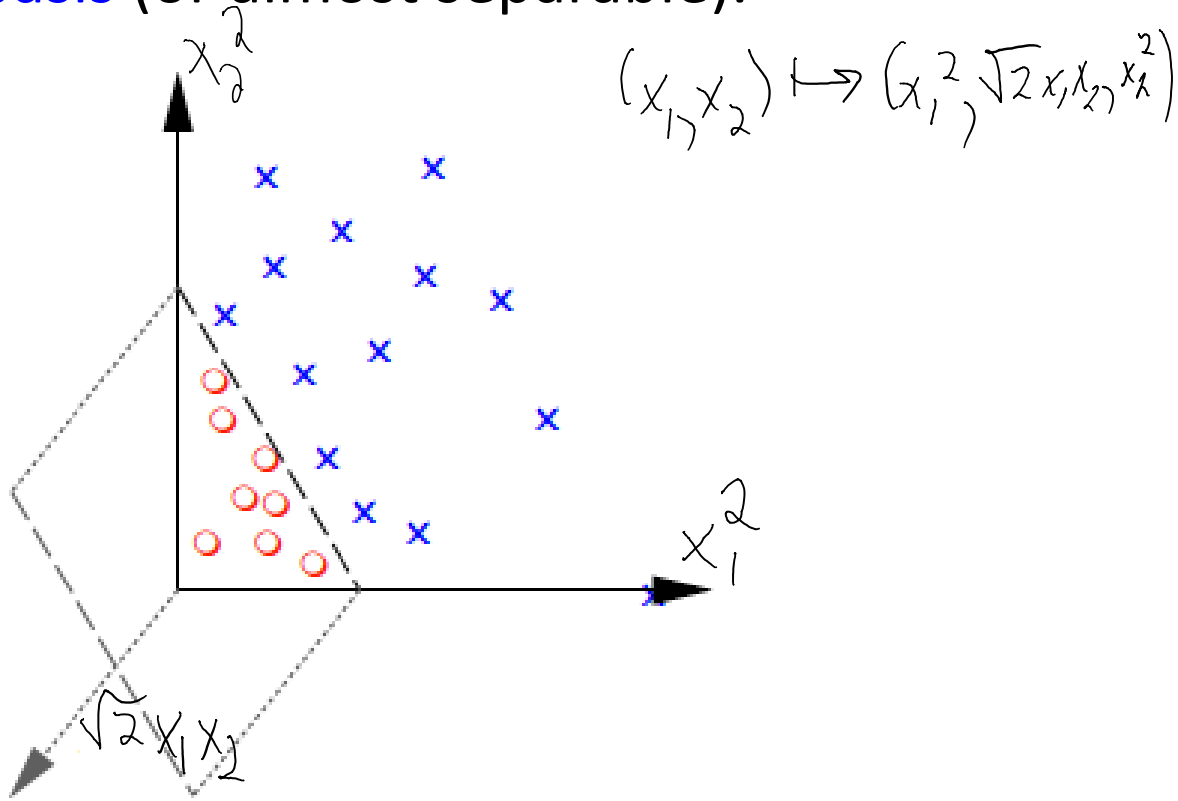


http://math.stackexchange.com/questions/353607/how-do-inner-product-space-determine-half-planes

Support Vector Machines for Non-Separable

- What about data that is not even close to separable?
  - It may be separable under change of basis (or almost separable).





http://math.stackexchange.com/questions/353607/how-do-inner-product-space-determine-half-planes

Multi-Dimensional Polynomial Basis

- Recall fitting polynomials:

$$y_i = w_0 + w_1 x_i + w_2 x_i^2$$

- We can fit these models using a change of basis:

$$X = \begin{bmatrix} 0.2 \\ -0.5 \\ 1 \\ 4 \end{bmatrix} \qquad X_{poly} = \begin{bmatrix} 1 & 0.2 & (0.2)^2 \\ 1 & -0.5 & (-0.5)^2 \\ 1 & 1 & (1)^2 \\ 1 & 4 & (4)^2 \end{bmatrix}$$

- How can we do this when we have a lot of features?

Multi-Dimensional Polynomial Basis

- Approach 1: use polynomial basis for each variable.

$$X = \begin{bmatrix} 0.2 & 0.3 \\ 1 & 0.5 \\ -0.5 & -0.1 \end{bmatrix} \longmapsto X_{poly} = \begin{bmatrix} 1 & 0.2 & (0.2)^2 & 0.3 & (0.3)^2 \\ 1 & 1 & (1)^2 & 0.5 & (0.5)^2 \\ 1 & -0.5 & (-0.5)^2 & -0.1 & (-0.1)^2 \end{bmatrix}$$

- But this is restrictive:
  - We should allow terms like 'x_1x_2' that depend on feature interaction.
  - But number of terms in X_{poly} is huge:
    - Degree-5 polynomial basis has $O(d^5)$ terms:

$$x_1^5, x_1^4x_2, x_1^4x_3, \ldots, x_1^4x_d, x_1^3x_2^2, x_1^3x_3^2, \ldots, x_1^3x_d^2, \ldots, x_2^5, x_2^4x_3, \ldots, x_d^5$$

- If 'n' is not too big, we can do this efficiently using the kernel trick.

Equivalent Form of Ridge Regression

- Recall L2-regularized least squares model:

$$\underset{w \in \mathbb{R}^d}{\operatorname{argmin}} \ \frac{1}{2}\|y - Xw\|^2 + \frac{\lambda}{2}\|w\|^2$$

$X: n \times d$

- We showed that the solution is given by:

$$w = (X^T X + \lambda I)^{-1} X^T y$$

$X^T X: d \times d$

- An equivalent way to write solution is:

$$w = X^T (XX^T + \lambda I)^{-1} y$$

$d \times n \quad n \times n \quad n \times 1$

$XX^T: n \times n$

- Computing w with second formula is faster if n << d:
  - since XX^T is 'n' by 'n' while X^TX is 'd' by 'd'.

Predictions using Equivalent Form

- Given test data $\hat{X}$, predict $\hat{y}$ using:

$$\hat{y} = \hat{X}w$$
$$= \hat{X}X^T(XX^T + \lambda I)^{-1}y$$
$$= \hat{K}(K + \lambda I)^{-1}y$$

where $K = XX^T$ and $\hat{K} = \hat{X}X^T$

w

$X: n \times d$

$X^TX: d \times d$

$XX^T: n \times n$

$\hat{X}X^T: t \times n$

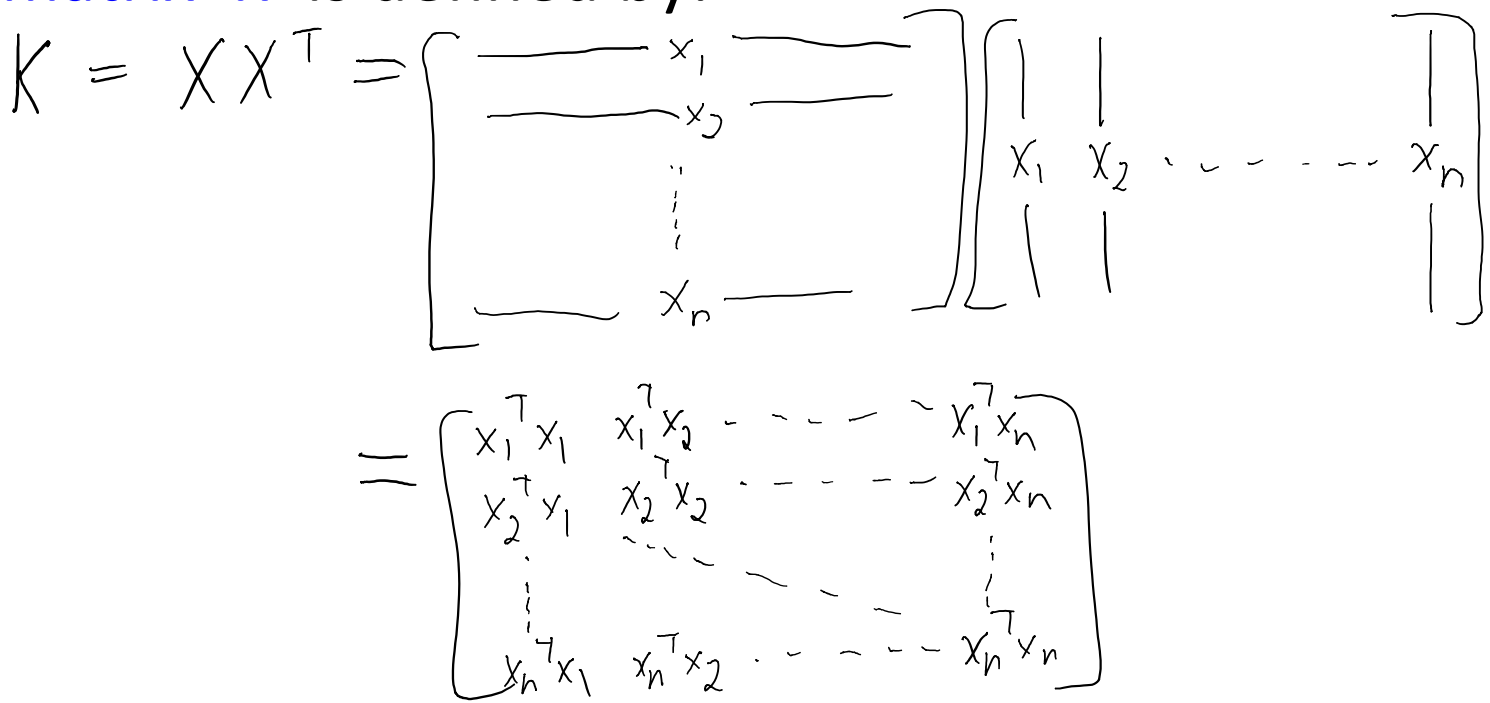
$\hat{X}: t \times d$

number of test examples

- Key observation behind kernel trick:
  - If we have K and $\hat{K}$, we don't need the features.

Gram Matrix

- The Gram matrix ‘K’ is defined by:



- ‘K’ contains the inner products between all training examples.
- ‘$\widehat{K}$ contains the inner products between training and test examples.
  - If we have some way to compute $x_i^T x_j$, we don’t need x_i and x_j.

Polynomial Kernel

- Consider two examples x_i and x_j for a 2-dimensional dataset:

$$x_i = (x_{i1}, x_{i2}) \qquad x_j = (x_{j1}, x_{j2})$$

- And consider a particular degree-2 basis:

$$(x_{poly})_i = (x_{i1}^2, \sqrt{2}\, x_{i1} x_{i2}, x_{i2}^2) \qquad (x_{poly})_j = (x_{j1}^2, \sqrt{2}\, x_{j1} x_{j2}, x_{j2}^2)$$

- We can compute inner product without forming $(x_{poly})_i$ and $(x_{poly})_j$:

$$\begin{aligned}
(x_{poly})_i^T (x_{poly})_j &= (x_{i1}^2, \sqrt{2}\, x_{i1} x_{i2}, x_{i2}^2)^T (x_{j1}^2, \sqrt{2}\, x_{j1} x_{j2}, x_{j2}^2) \\
&= x_{i1}^2 x_{j1}^2 + 2 x_{i1} x_{i2} x_{j1} x_{j2} + x_{i2}^2 x_{j2}^2 \\
&= (x_{i1} x_{j1} + x_{i2} x_{j2})^2 \qquad \text{"completing the square"} \\
&= (x_i^T x_j)^2
\end{aligned}$$

Polynomial Kernel with Higher Degrees

- If we want all degree-4 monomials, raise to 4th power:

$$(x_i^T x_j)^4 = (x_{poly})_i^T (x_{poly})_j$$

Corresponds to x_{poly} having weighted versions of $x_{i1}^4, x_{i1}^3 x_{i2}, x_{i1}^2 x_{i2}^2, x_{i1} x_{i2}^3, x_{i2}^4$.

- For lower-order terms like x_{i1} or bias, add constant inside power:

$$(1 + x_i^T x_j)^2 = 1 + x_i^T x_j + (x_i^T x_j)^2$$
$$= (1, x_{i1}, x_{i2}, x_{i1}^2, \sqrt{2}\, x_{i1} x_{i2}, x_{i2}^2)^T (1, x_{j1}, x_{j2}, x_{j1}^2, \sqrt{2}\, x_{j1} x_{j2}, x_{j2}^2)$$
$$= (x_{poly})_i^T (x_{poly})_j \quad \text{where } (x_{poly})_i = (1, x_{i1}, x_{i2}, x_{i1}^2, \sqrt{2}\, x_{i1} x_{i2}, x_{i2}^2)$$

- These formula still work for any dimension of the x_i

Kernel Trick

- Using polynomial basis of degree 'p' with the kernel trick:
  - Compute K and $\hat{K}$:

$$K(i,j) = (1 + x_i^T x_j)^p \qquad \hat{K}(i,j) = (1 + \hat{x}_i^T x_j)^p$$

test example 'i'

training example 'j'

  - Make predictions using:

$$\hat{y} = \hat{K}(K + \lambda I)^{-1} y$$

$t \times 1$ $t \times n$ $n \times n$ $n \times 1$

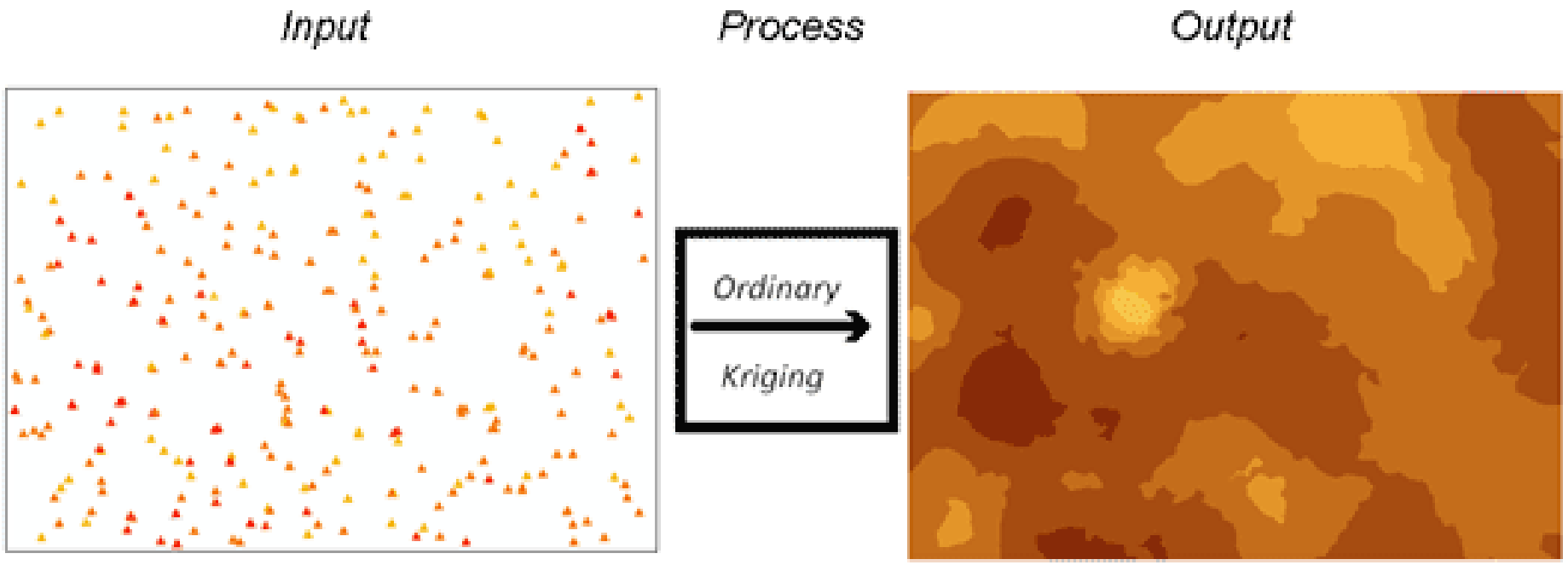
- Cost is $O(n^2 d + n^3)$, even though number of features is $O(d^p)$.
- Many algorithms have kernelized versions: SVMs, logistic, KNN, etc.

Kernel Trick

- Kernel trick lets us fit regression models without explicit features:
  - We can interpret K(i,j) as a similarity measure between objects.
  - We don't need x_i and x_j if we can compute 'similarity' between objects:
    - 'String' kernels, 'graph' kernels, 'image' kernels, etc.
- We call a kernel 'valid' if there exists feature-space representation.
  - This might be very high-dimensional or even infinite-dimensional.
  - Characterizing valid kernels theoretically: "Mercer's theorem".
  - In practice, there are a few tricks to memorize.
- Most common non-linear kernels:
  - Polynomial kernel: $(\beta + x^T z)^p$
  - RBF kernel: $\exp\left(-\frac{\|x - z\|^2}{2\sigma^2}\right)$ "universal kernel"

Motivation: Finding Gold

- Kernel methods first came from mining engineering ('Kriging'):
  - Mining company wants to find gold.
  - Drill holes, measure gold content.
  - Build a regression model (typically RBF kernel).

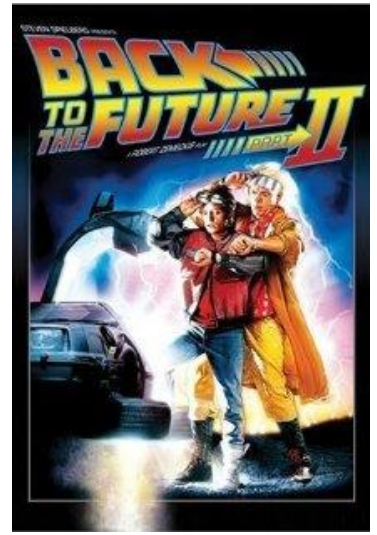


http://www.bisolutions.us/A-Brief-Introduction-to-Spatial-Interpolation.php

Summary

- Loss plus regularizer describes a huge number of ML models.
- Support vector machines maximize margin to nearest data points.
- High-dimensional bases allows us to separate non-separable data.
- Kernel trick allows us to use high-dimensional bases efficiently.
- Kernels let us use similarity between objects, rather than features.

- Next time:
  - Fitting linear models with huge number of training examples.
  - Predicting the future part 2.



http://www.imdb.com/title/tt0096874/
http://www.october212015.com/