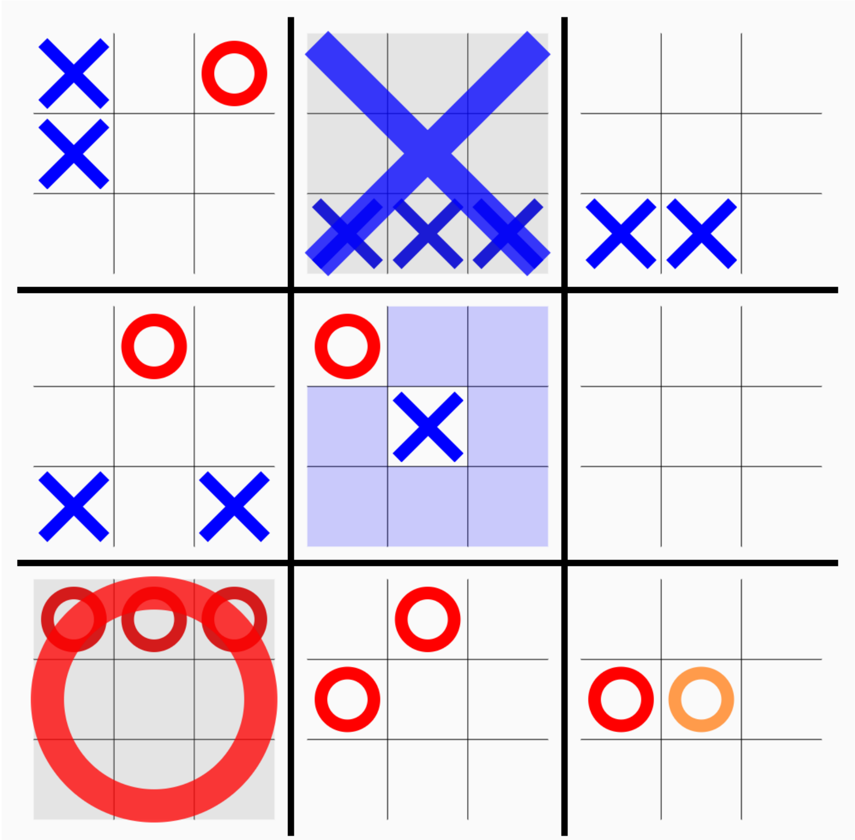
CS2109S: Introduction to AI and Machine Learning

Lecture 7: Regularization, Kernels, and Support Vector Machines

11 March 2025

Announcements

Mini-Project



- Develop an agent to play **Ultimate Tic-Tac-Toe**
- Can use search, machine learning, or both!
- Compete against our agents.
  - If you win against all of our agents then full mark – 10%.
  - Developed only using techniques in class.
  - Calibrated to be beatable by reasonable agents.
- Constraints:
  - Minimax-family only
  - No state representation modifications
- Due Date: **12 Apr 23:59** (~1 Month from now)
- See announcements for more details.

Plagiarism

- First batch of cases found. Will submit to UG by this week.
- We will continue investigation for the rest of the cases.
- Mini Project:
  - Any form of cheating—such as plagiarism, hacking Coursemology test cases, or any other dishonest conduct—will be treated as an academic offense.
  - The mini project constitutes 10% of the final grade; therefore, any academic misconduct will be classified as a **Moderate Offense**, with the maximum penalty being an **'F' grade** for the module.
  - Submission history will be closely monitored.

Survey Results: Lectures

2.

How do you find the lectures?

** Required*

S/N	Multiple Choice Option	Count	Percentage
1	I have no clue what the lecturer is talking about most of the time	2	0.8%
2	I have no idea what is happening half the time	42	16.7%
3	Lectures are okay	99	39.4%
4	Lectures are clear and I am able to follow the material quite well	99	39.4%
5	Lectures are way cool. Easily the best class that I've taken at NUS (not like I've taken a lot or even any other, but I'm guessing)	9	3.6%

Sort By Percentage

3.

How do you find the content of the lectures?

** Required*

S/N	Multiple Choice Option	Count	Percentage
1	Too handwavy	25	10.0%
2	Just right	193	76.9%
3	Too rigorous	33	13.1%

Sort By Percentage

Survey Results: Midterm

Time: The time allocated to answer all the questions.

** Required*

S/N	Multiple Choice Option	Count	Percentage	Sort By Percentage
1	Way too little. Too long, too little time.	7	2.8%	
2	Time is somewhat short	45	17.9%	
3	Time allocated is just nice	190	75.7%	
4	Too much time, too little to do	8	3.2%	
5	I can nap for an hour during the midterm and still finish every question	1	0.4%	

Relevance: The content reflected what was taught in the course.

** Required*

S/N	Multiple Choice Option	Count	Percentage	Sort By Percentage
1	Strongly Agree	43	17.1%	
2	Agree	148	59.0%	
3	Neutral	46	18.3%	
4	Disagree	12	4.8%	
5	Strongly Disagree	2	0.8%	

Breadth of Topics: The exam covered the right range of topics.

** Required*

S/N	Multiple Choice Option	Count	Percentage	Sort By Percentage
1	Strongly Agree	38	15.1%	
2	Agree	140	55.8%	
3	Neutral	58	23.1%	
4	Disagree	12	4.8%	
5	Strongly Disagree	3	1.2%	

Clarity: The questions were clear and easy to understand.

** Required*

S/N	Multiple Choice Option	Count	Percentage	Sort By Percentage
1	Strongly Agree	34	13.5%	
2	Agree	113	45.0%	
3	Neutral	70	27.9%	
4	Disagree	29	11.6%	
5	Strongly disagree	5	2.0%	

Difficulty Level: The exam difficulty was appropriate.

** Required*

S/N	Multiple Choice Option	Count	Percentage	Sort By Percentage
1	Too Easy	4	1.6%	
2	Easy	13	5.2%	
3	Just Right	141	56.2%	
4	Difficult	83	33.1%	
5	Too Difficult	10	4.0%	

Survey Results: Overall

What is your overall impression of CS2109S thus far?

** Required*

S/N	Multiple Choice Option	Count	Percentage
1	This is a horrible class. Truly regret choosing it.	4	1.6%
2	Just like any other module.	83	33.1%
3	CS2109S is cool.	145	57.8%
4	CS2109S rocks. Coolest class I have taken in my life.	19	7.6%

Has CS2109S been able to arouse your interest in AI/ML?

** Required*

S/N	Multiple Choice Option	Count	Percentage
1	Yes	138	55.0%
2	No	35	13.9%
3	I was already interested in AI/ML before CS2109S!	70	27.9%
4	I was once interested in AI/ML, but CS2109S killed it :-'(	8	3.2%

Would you recommend CS2109S to other students?

** Required*

S/N	Multiple Choice Option	Count	Percentage
1	Yes	215	85.7%
2	No	36	14.3%

Survey Results: Other Things

- Lectures/tutorials
  - More examples
  - Less math and proofs (but majority says everything is just right)
- Workload:
  - Just nice (majority), leaning towards somewhat heavy (2^{nd} most votes)
- Tutorials:
  - Helpful for most of people

Materials

Recap

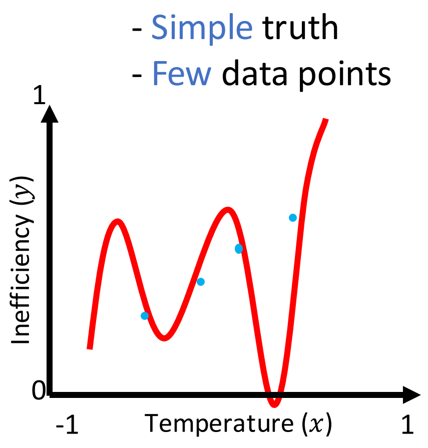
- Logistic Regression: compute the probability of an input belonging to a class
  - Model: d dimensional input features: $h_w(x) = \sigma(\sum_{j=0}^{d} w_j x_j) = \sigma(w^T x)$
  - Loss: Binary Cross Entropy (BCE) Loss
  - Non-linearly separable data: use feature transformations
- Learning via Gradient Descent: derivative is the same with linear regression!
- Multi-Class classification: One vs One, One vs Rests
- Advanced Topics in Supervised Learning
  - Generalization
  - Model Complexity
  - Overfitting & Underfitting
  - Hyperparameter Tuning

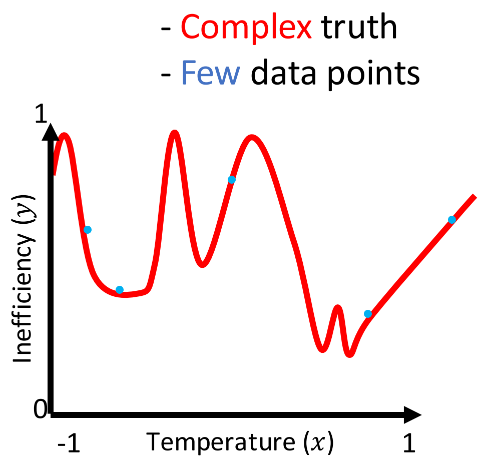
Outline

- Regularization
  - The problem of overfitting
  - Regularization
  - Linear regression with regularization
- Kernel Method
  - Dual formulation of linear regression
  - Transformed features
  - Kernel functions
- Support Vector Machines

The Problem of **Overfitting**

Complex model fits all data points including the noise in the data.

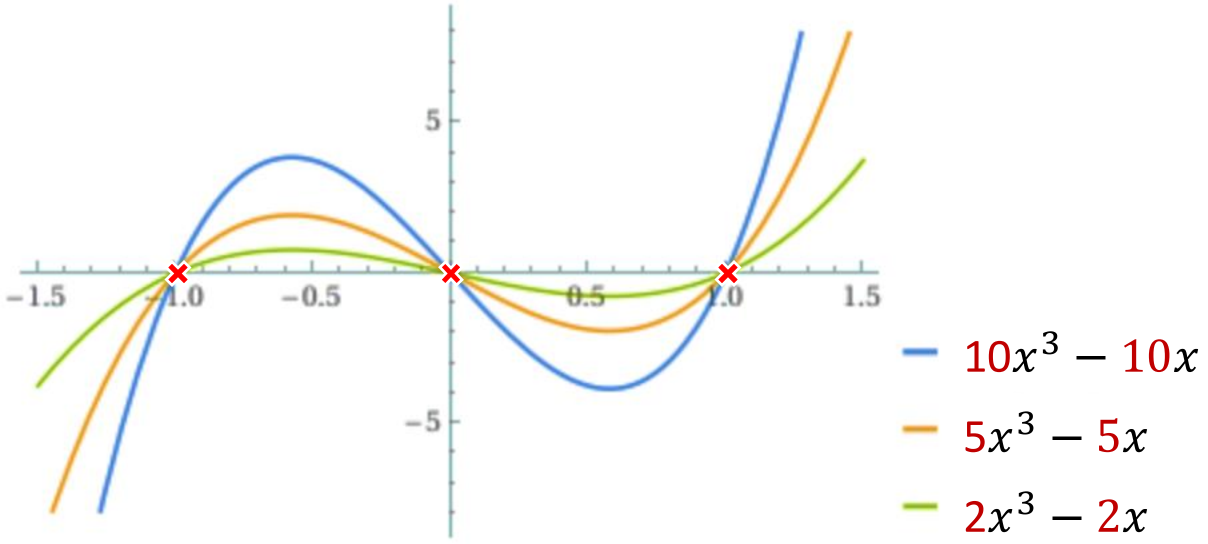




The learned model often badly generalizes to unseen data as it does not capture the underlying ground truth.

A View on **Overfitting:** Large Weights

Consider a model $h(x) = wx^3 - wx$ and a dataset $\{(x^{(i)}, y^{(i)})\} = \{(-1,0), (0,0), (1,0)\}$



Increasing w increases the oscillation (complexity) while "fitting" the same three points.

Overfitting is often associated with large weights! What to do?

Key Idea: **Penalize** Large Weights

What can we do to prevent overfitting?

➢ Keep weights small during optimization of the weights.

How can we keep them small?

➢ Put a cost on having large weights.

How do we measure cost so far in machine learning?

➢ Loss function!

Regularization: Main Idea

Given a loss function (e.g., MSE, BCE):

$$J(w)$$

Add a penalty function/regularizer $P(w)$ to the loss function with a penalty strength $\lambda \geq 0$:

$$J_{reg}(w) = J\,(w) + \lambda\, P(w)$$

Optimization goal:

$$\min_{w} J_{reg}(w)$$

Regularization: Penalty Functions

- Square: $P(w) = \sum_{j=0}^{d} \frac{1}{2} w_j^2$

Also known as: L2 penalty/regularizer
Ridge regression in Linear Regression

- Absolute: $P(w) = \sum_{j=0}^{d} |w_j|$

Also known as: L1 penalty/regularizer
Lasso regression in Linear Regression

- Max: $P(w) = \max(w_0, w_1, w_2, \ldots)$

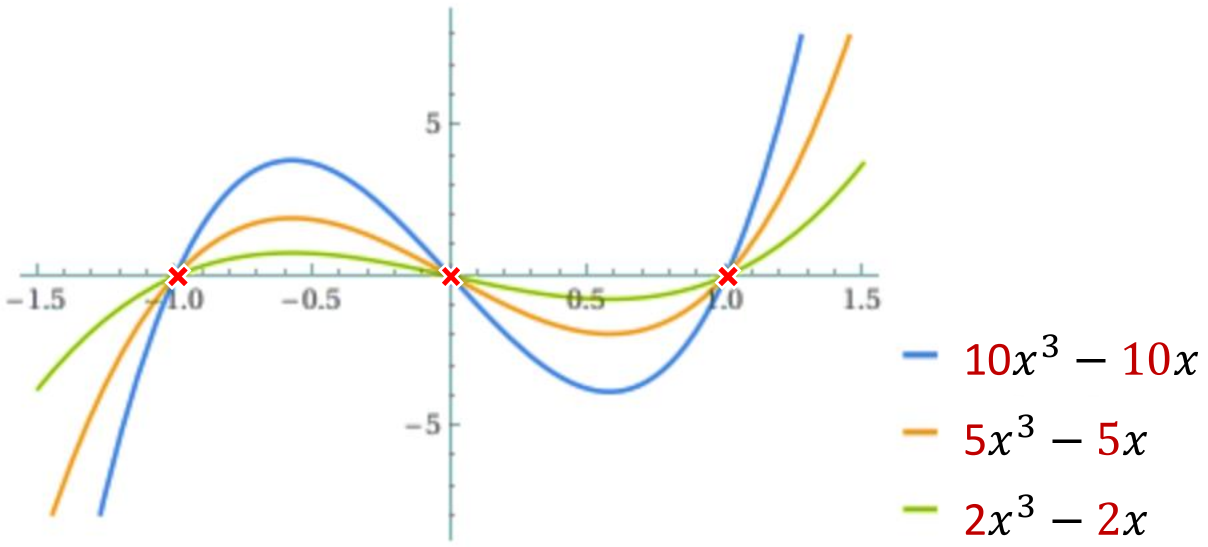
Also known as: Max-norm penalty

- Others: entropy, …

Regularization: An Example

Consider a model $h(x) = wx^3 - wx$ and a dataset $\{(\mathrm{x}^{(i)}, y^{(i)})\} = \{(-1,0), (0,0), (1,0)\}$.

Suppose, we use the L1 penalty/regularizer.



$$J_{reg}(w) = J\,(w) + \lambda|w|$$

Regularization and Learning Algorithms

- Gradient descent: apply gradient to both terms

$$\frac{\partial}{\partial w_j} J_{reg}(w) = \frac{\partial}{\partial w_j} J(w) + \lambda \frac{\partial}{\partial w_j} P(w)$$

- Normal equation: can derive for linear regression with square penalty (Ridge regression).

Background: Identity matrix

$$\mathbb{I} = \begin{bmatrix} 1 & 0 & \ddots & 0 \\ 0 & 1 & \ddots & \ddots \\ \ddots & \ddots & \ddots & \\ 0 & 0 & \ddots & 1 \end{bmatrix}$$

Let λ be a scalar. What is $\lambda\ \mathbb{I}$?

Let A be a matrix. What is $\mathrm{A}\mathbb{I}$?

Recall: Normal Equation

Goal: find w that minimizes J_{MSE}

$$\frac{\partial J_{MSE}(w)}{\partial w_j} = \frac{1}{N}\sum_{i=1}^{N}\left(w^T x^{(i)} - y^{(i)}\right)x_j^{(i)} = 0$$

Express with **vectors** and **matrices**

$$X^T(Xw - Y) = 0$$

$$X = \begin{bmatrix} 1 & x_1^{(1)} & & x_d^{(1)} \\ 1 & x_1^{(2)} & \dots & x_d^{(2)} \\ 1 & \vdots & & \vdots \\ 1 & x_1^{(N)} & & x_d^{(N)} \end{bmatrix} \quad w = \begin{bmatrix} w_0 \\ w_1 \\ \dots \\ w_d \end{bmatrix} \quad Y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \dots \\ y^{(N)} \end{bmatrix}$$

Assume **invertible**

$$\boxed{w = (X^T X)^{-1} X^T Y}$$

Normal Equation with **Regularization**

Find w that minimizes $\frac{\partial J_{reg}(w)}{\partial w_j}$

$$\frac{\partial J_{reg}(w)}{\partial w_j} = \frac{1}{N}\sum_{i=1}^{N}\left(w^T x^{(i)} - y^{(i)}\right)x_j^{(i)} + \lambda w_j = 0$$

Performing the same steps as before, we will arrive at:

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

Theorem: For all $\lambda > 0$ the matrix $X^T X + \lambda \mathbb{I}$ is invertible.

- Normal equation with regularization works no matter whether there is (almost) linear dependency among the features or insufficient number of observations.

Regularization: Summary

- Overfitting is (often) due to a complex model with large weights.
- Main idea: penalize large weights by adding a term in the loss function.
  - Learning algorithm takes the penalty into account via the gradient of the loss function.
  - Result: weight vector tends to have smaller weights.
- The trade-off between fitting the data and keeping the weights small is determined by a hyperparameter λ (penalty strength).

Outline

- Regularization
    - The problem of overfitting
    - Regularization
    - Linear regression with regularization

- Kernel Method
    - Dual formulation of linear regression
    - Transformed features
    - Kernel functions

- Support Vector Machine

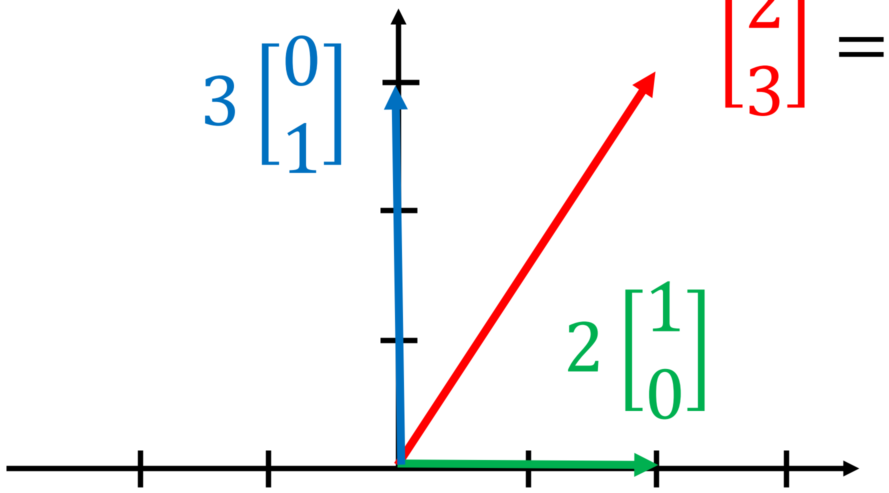
Background: Linear Combination of Vectors

Let $x^{(1)}, x^{(2)}, \dots, x^{(N)}$ be N vectors. Let $\alpha_1, \alpha_2, \dots, \alpha_N$ be N real numbers. The corresponding linear combination v is defined by:

$$v = \alpha_1\, x^{(1)} + \alpha_2\, x^{(2)} + \cdots + \alpha_N\, x^{(N)}$$

Example:

$$\begin{bmatrix}2\\3\end{bmatrix} = 2\begin{bmatrix}1\\0\end{bmatrix} + 3\begin{bmatrix}0\\1\end{bmatrix}$$



Linear Model: Weights and Training Data

Recall: data

$$X = \begin{bmatrix} 1 & x_1^{(1)} & & x_d^{(1)} \\ 1 & x_1^{(2)} & \dots & x_d^{(2)} \\ 1 & \vdots & & \vdots \\ 1 & x_1^{(N)} & & x_d^{(N)} \end{bmatrix} = \left[x^{(1)}, x^{(2)}, \dots, x^{(N)}\right]^T \qquad Y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \dots \\ y^{(N)} \end{bmatrix}$$

It can be shown that normal equation can be re-written as a weighted combination of training data:

$$\begin{aligned} w &= (X^T X)^{-1} X^T Y \\ &= (X^T X)^{-1} \sum_{j=1}^{N} y^{(j)} x^{(j)} \\ &= \sum_{j=1}^{N} \alpha_j x^{(j)} \end{aligned}$$

Here, α is a vector of real numbers of dimension N.

Linear Model: Dual Formulation

Since weights can be rewritten as a linear combination of training data:

$$w = \sum_{j=1}^{N} \alpha_j x^{(j)}$$

We can rewrite our linear model as follows:

$$h_w(x) = w^T x = \sum_{j=1}^{N} \alpha_j \, {x^{(j)}}^T x = h_\alpha(x)$$

Here, we obtain a *dual* hypothesis $h_\alpha(x)$ which contains a sum of dot products between all $x^{(j)}$ and x, and parameters/weights α_j.

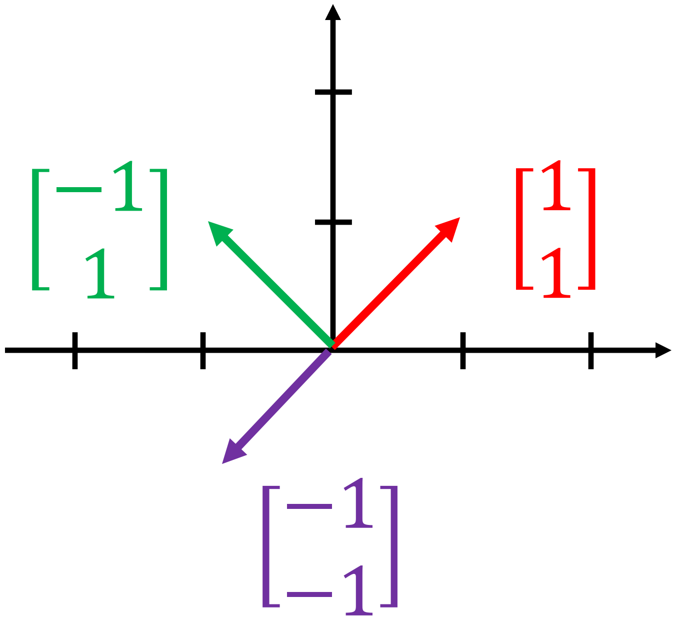
Background: Dot Product and Similarity

Suppose that we have two vectors, u and v.

$$u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \quad v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

Dot product:

$$u \cdot v = u^T v = u_1 v_1 + u_2 v_2$$



Dot product gives the **<u>similarity</u>** of two vectors:

$$\begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = 2 \qquad \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \end{bmatrix} = 0 \qquad \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} -1 \\ -1 \end{bmatrix} = -2$$

Linear Model: Dual Formulation with Kernel

Dual hypothesis of linear model:

$$h_\alpha(x) = \sum_{j=1}^{N} \alpha_j \, {x^{(j)}}^T x$$

Let k be the function that defines the dot product:

$$k(u, v) = u^T v$$

We can rewrite the dual hypothesis using $k(u, v)$ as follows:

$$h_\alpha(x) = \sum_{j=1}^{N} \alpha_j k(x^{(j)}, x)$$

Instead of $k(u, v) = u^T v$, we can also use a different function and obtain a different linear model.

Kernels and Kernel Trick

- Moral of the story: when a dot product shows up, we can replace it with other similarity functions $k(u, v)$.
  - These functions are called kernel functions or simply kernels.
  - This replacement is called the kernel trick.
  - A hypothesis function that uses kernel trick is called a kernel machine.
- A kernel is a valid kernel if it satisfies the kernel validity conditions.
  - Valid kernel = *continuous symmetric positive-definite* kernel.
- Why do we want to do this?

Kernels and **Transformed Features**

There is a relation between kernels and feature transformations.

Let’s have a look!

Transformed Features: Single Variable

So far, we have created a new feature from a single existing features. Examples:

- Monomials: $x_j \rightarrow x_j$ and x_j^5.
- Log: $x_j \rightarrow x_j$ and $\log(x_j)$.
- Exponential: $x_j \rightarrow x_j$ and $\exp(x_j)$.
- …

Such transformations allow us to make the models more complex/expressive.

Can you think of more general transformations?
Multi-variable!

Transformed Features: Multi-Variable

Multi-variable transformed features take several features and create a new feature. Examples:

- Monomials: x_k and $x_j \rightarrow x_k$ and x_j and $x_k x_j$.
- Log: x_k and $x_j \rightarrow x_k$ and x_j and $\log(x_k + x_j)$.
- Exponential: x_k and $x_j \rightarrow x_k$ and x_j and $\exp(x_k x_j)$.
- …

What is the most general case?

$x \in \mathbb{R}^d \rightarrow \phi(x) \in \mathbb{R}^M$

Transformed Features: Multi-Variable

The most general case is when we take a d-dimensional feature vector and transform it into a M-dimensional feature vector. Usually $M \geq d$.

$$x \in \mathbb{R}^d \quad \rightarrow \quad \phi(x) \in \mathbb{R}^M$$

Here, the function ϕ is called a feature transformation or feature map, and the vector $\phi(x)$ is called transformed features.

We know that feature transformations helps with non-linear data, but…

Example: **Polynomial Features**

Feature transformation to all monomials of degree up to 2.

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \rightarrow \phi_{M2}(x) = [x_1, x_2, x_1^2, x_2^2, x_1 x_2]^T$$

Feature transformation to all monomials of degree up to 2, starting with 3 features.

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \rightarrow \phi_{M2}(x) = [x_1, x_2, x_3, x_1^2, x_2^2, x_3^2, x_1 x_2, x_1 x_3, x_2 x_3]^T$$

For 100 features and transformation to all monomials of degree 6, the feature vector dimension is about 1.6 billion!

What can we do?

Linear Model with Transformed Features

Given an input vector x of dimension d and a feature map $\phi(x)$ of dimension M, the hypothesis class of linear models with transformed features is defined as the set of functions:

$$h_w^{\phi}(x) = w_0\phi(x)_0 + w_1\phi(x)_1 + w_2\phi(x)_2 + \cdots + w_M\phi(x)_M$$

where $w_0, \dots, w_M$ are **parameters/weights**, with dummy feature $\phi(x)_0 = 1$.

We shorthand this function by using the dot product:

$$h_w(x) = w^T\phi(x)$$

Dual Hypothesis with Transformed Features

The dual hypothesis of a linear model with ϕ feature map is as follows:

$$h_{\alpha}^{\phi}(x) = \sum_{j=1}^{N} \alpha_j \phi^T(x^{(j)})\, \phi(x)$$

Notice that we have a dot product between transformed features:

$$\phi^T(x^{(j)})\, \phi(x)$$

This dot product defines a new valid kernel function:

$$k_{\phi}(u, v) := \phi^T(u)\phi(v)$$

Dual Hypothesis with Kernel

The dual hypothesis of a linear model with kernel k_ϕ (based on feature map ϕ) is as follows:

$$h_{\alpha}^{\phi}(x) = \sum_{j=1}^{N} \alpha_j k_\phi(x^{(j)}, x)$$

Notice that we don't have to compute $\phi(x)$ explicitly. Is it beneficial?

Polynomial Kernel

Polynomial degree = 1:

$$k_{P1}(u,v) = \phi_{P1}(u)^T \phi_{P1}(v) = [u_1, u_2] \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = u^T v$$

Polynomial degree = 2:

$$k_{P2}(u,v) = \phi_{P2}(u)^T \phi_{P2}(v) = \left[u_1^2, \sqrt{2}u_1 u_2, u_2^2\right] \begin{bmatrix} v_1^2 \\ \sqrt{2}v_1 v_2 \\ v_2^2 \end{bmatrix} = (u^T v)^2$$

Polynomial degree = s (d^s terms):

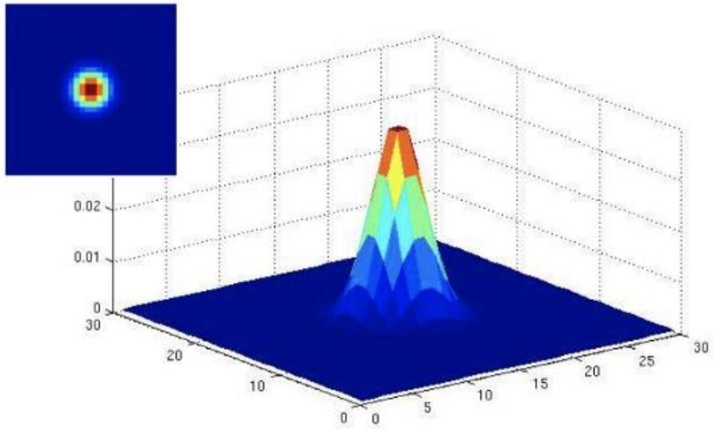
$$k_{Ps}(u,v) = \phi_{Ps}(u)^T \phi_{Ps}(v) = (u^T v)^s$$

Computing kernel is very efficient!

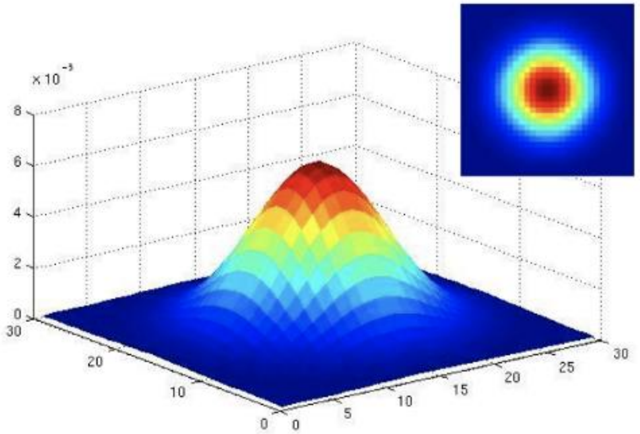
Gaussian Kernel

A popular kernel function is the Gaussian Kernel or Radial Basis Function (RBF) kernel:

$$k_{RBF}(u,v) := e^{-\frac{\|u-v\|^2}{2\sigma^2}}$$



$\sigma^2 = small$



$\sigma^2 = large$

It can be shown that k_{RBF} corresponds to a feature map ϕ_{RBF} that maps to an **infinite-dimensional** feature vector.

Can't even practically compute $\phi_{RBF}(x)$ explicitly!

Source: https://medium.com/jun94-devpblog/cv-2-gaussian-and-median-filter-separable-2d-filter-2d11ee022c66

Other Kernels

- String kernel
- Chi-squared kernel
- tanh kernel
- …

Kernels: Chicken and Egg

What comes first? Feature transformation or Kernel function?

- For some kernels (e.g., polynomial kernel), we know the feature map and we formulate the kernel based on that.
- For some other kernels (e.g., RBF kernel), we can't even practically compute the feature map.

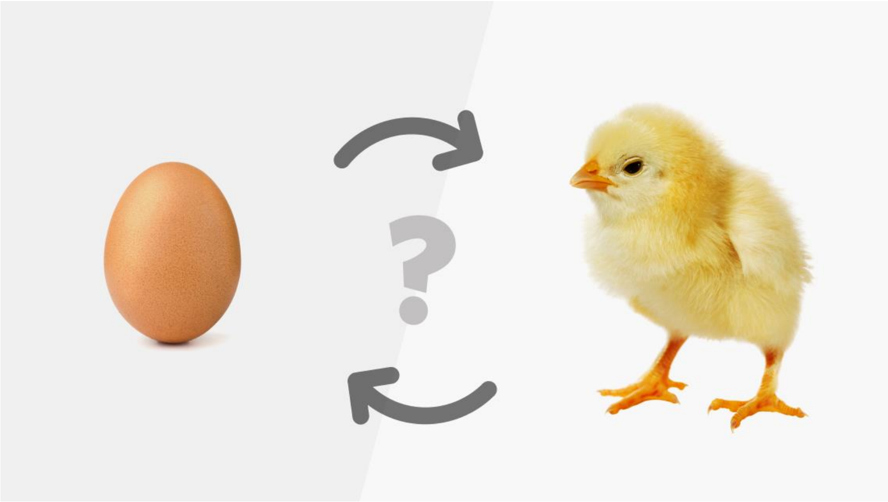


Image credit: LinkedIn

Kernels and **Transformed Features**

There is a relation between kernels and feature transformations.

Theorem (informal):

- For any valid kernel $k(u, v)$, there exists a corresponding feature transformation ϕ (which may be infinite-dimensional), where $k(u, v) = \phi^T(u)\, \phi(v)$.
- Conversely, every feature transformation ϕ induces a valid kernel k.

Conclusion to Kernels

- Dual formulation of linear model.
  - An example of a kernel machine.
- Kernel: a similarity function between vectors.
  - There are conditions for valid kernel functions.
- Replace dot product with other kernel function: Kernel trick!
- Feature transformations ↔ kernels
- Kernel machine utilizes kernel to compute feature transformation implicitly and very efficiently.

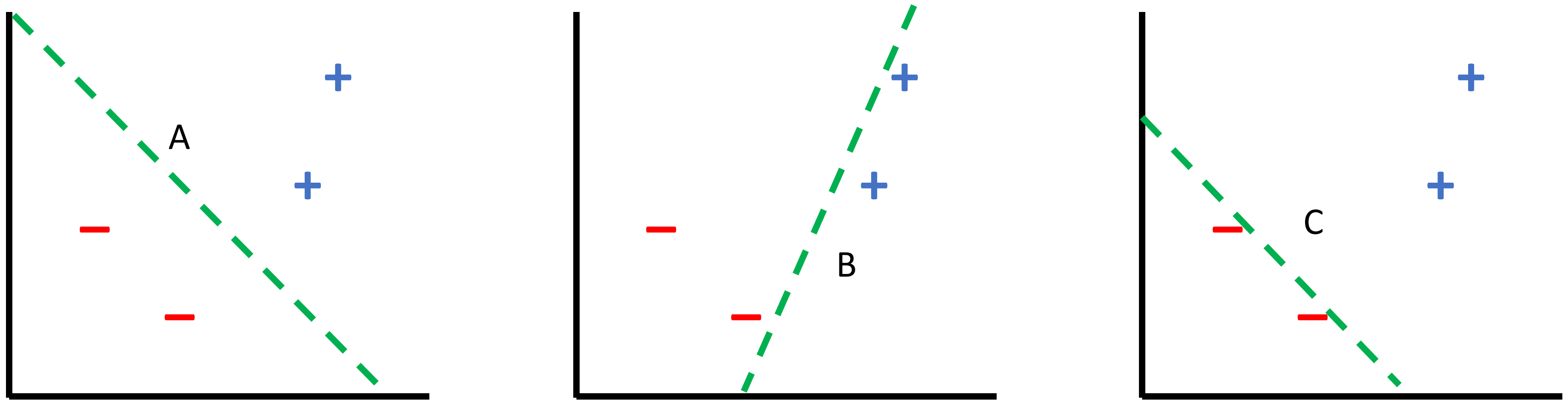
Outline

- Regularization
  - The problem of overfitting
  - Regularization
  - Linear regression with regularization
- Kernel Method
  - Dual formulation of linear regression
  - Transformed features
  - Kernel functions
- Support Vector Machine

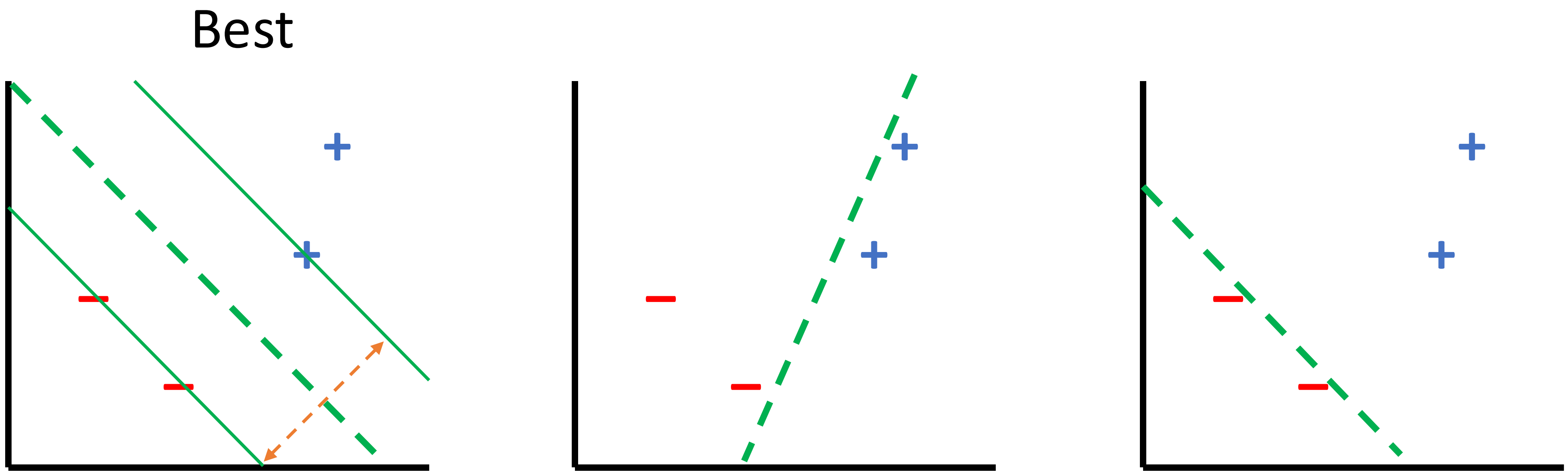
Poll Everywhere

Which one is the best decision boundary?

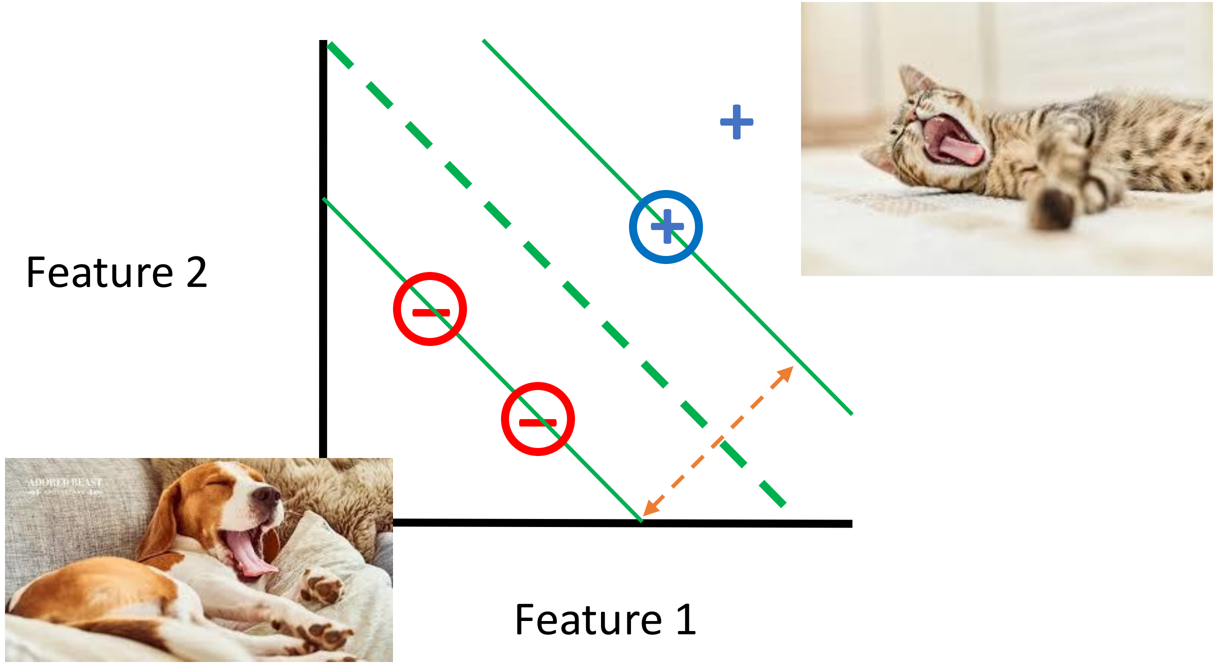
- A
- B
- C



Decision Boundaries



Support Vector Machine (SVM)



Based on constructed “optimal” decision boundary, classify a new data point.

Support Vector Machine (SVM)

Key aspects:

- SVM is a classifier (at least the one that we discuss in this lecture).
- Based on training data, we construct an "optimal" separating decision boundary.
  - "Fat margin" classifier that maximizes performance on unseen data.
  - Robustness to noise.
- Kernel trick can be applied naturally.
  - Leads to non-linear decision boundaries.
  - SVM is the most famous kernel machine.

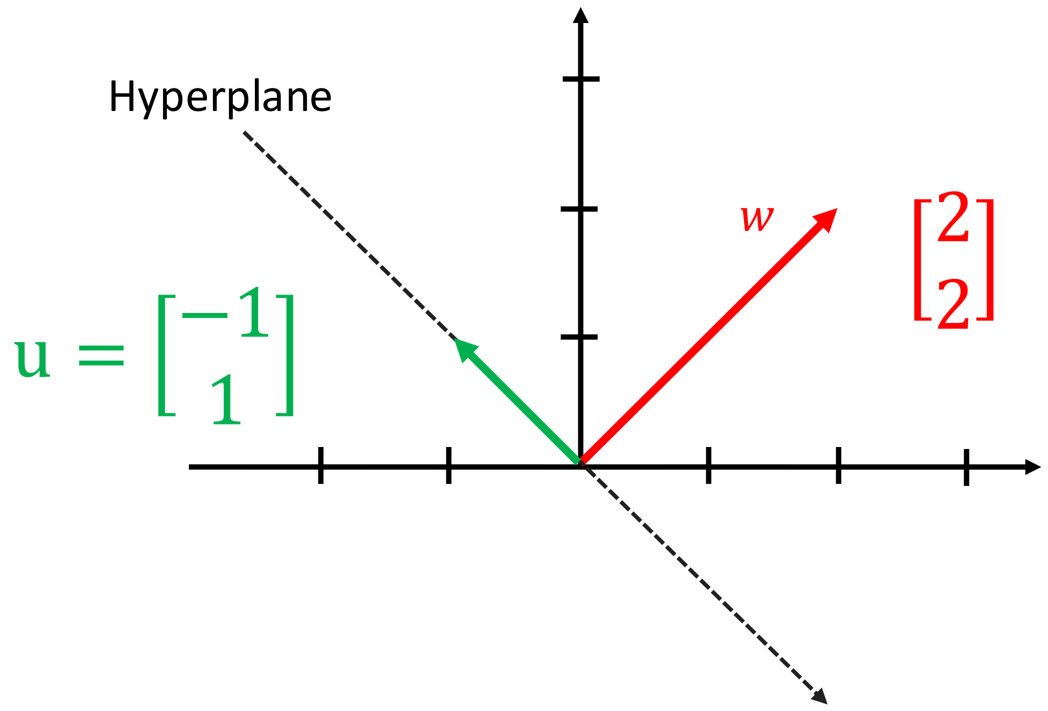
SVM: Disclaimer

Due to the necessary mathematical background required, a comprehensive discussion of Support Vector Machines (SVM) is beyond the scope of CS2109S. For those interested in a deeper exploration, we recommend optional further reading.

In this course, we will focus on introducing a select set of foundational concepts that lead to an understanding of SVM, specifically:

- Hyperplanes
- Hyperplane-based decision rules

Hyperplanes – Definition

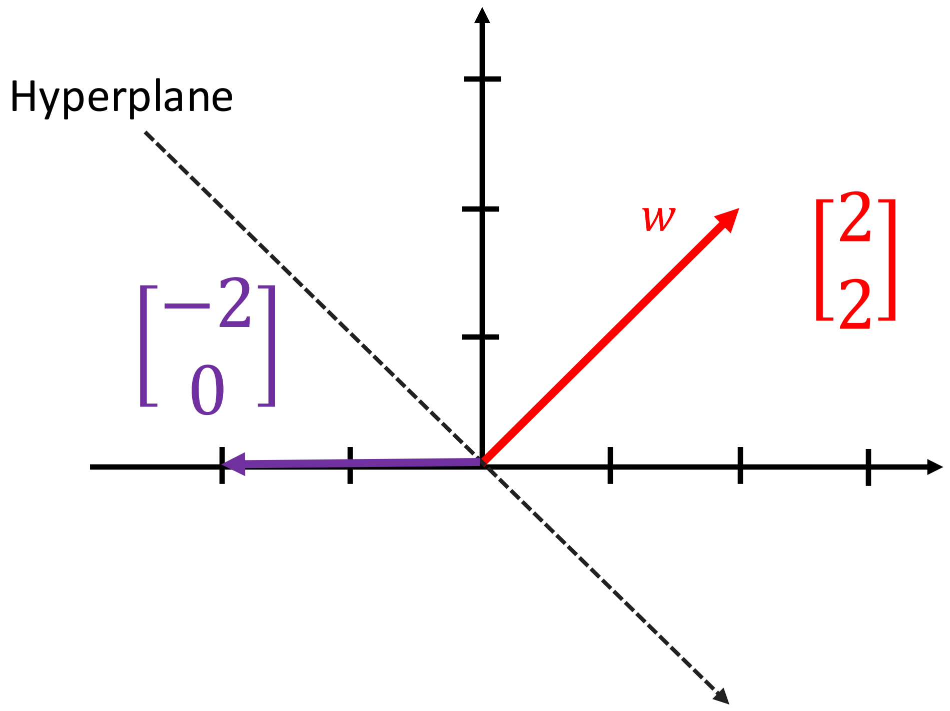


Dot product:

$[2\ 2]\begin{bmatrix}-1\\1\end{bmatrix}=0$

Definition: The zero-offset hyperplane is defined by a **normal vector** w and contains all vectors u for which $w^T u = 0$

Hyperplanes – Which side?

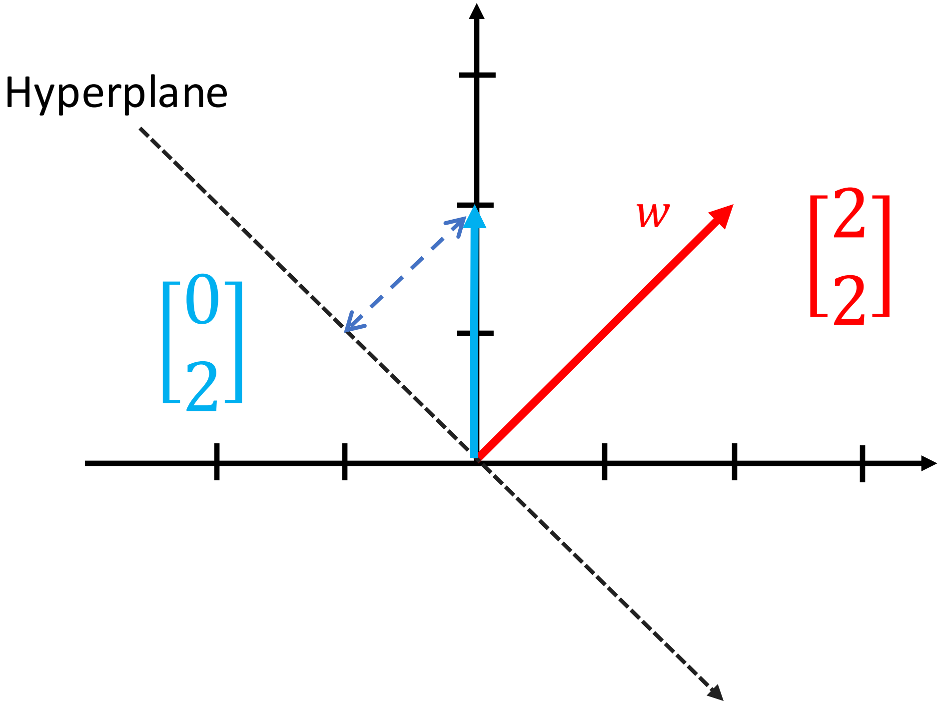


Dot product:

$$[2\ 2]\begin{bmatrix} -2 \\ 0 \end{bmatrix} = -4$$

The sign of $w^T u$ tells us which side a point u is with respect to the zero-offset hyperplane.

Hyperplanes – Distance?



Dot product:

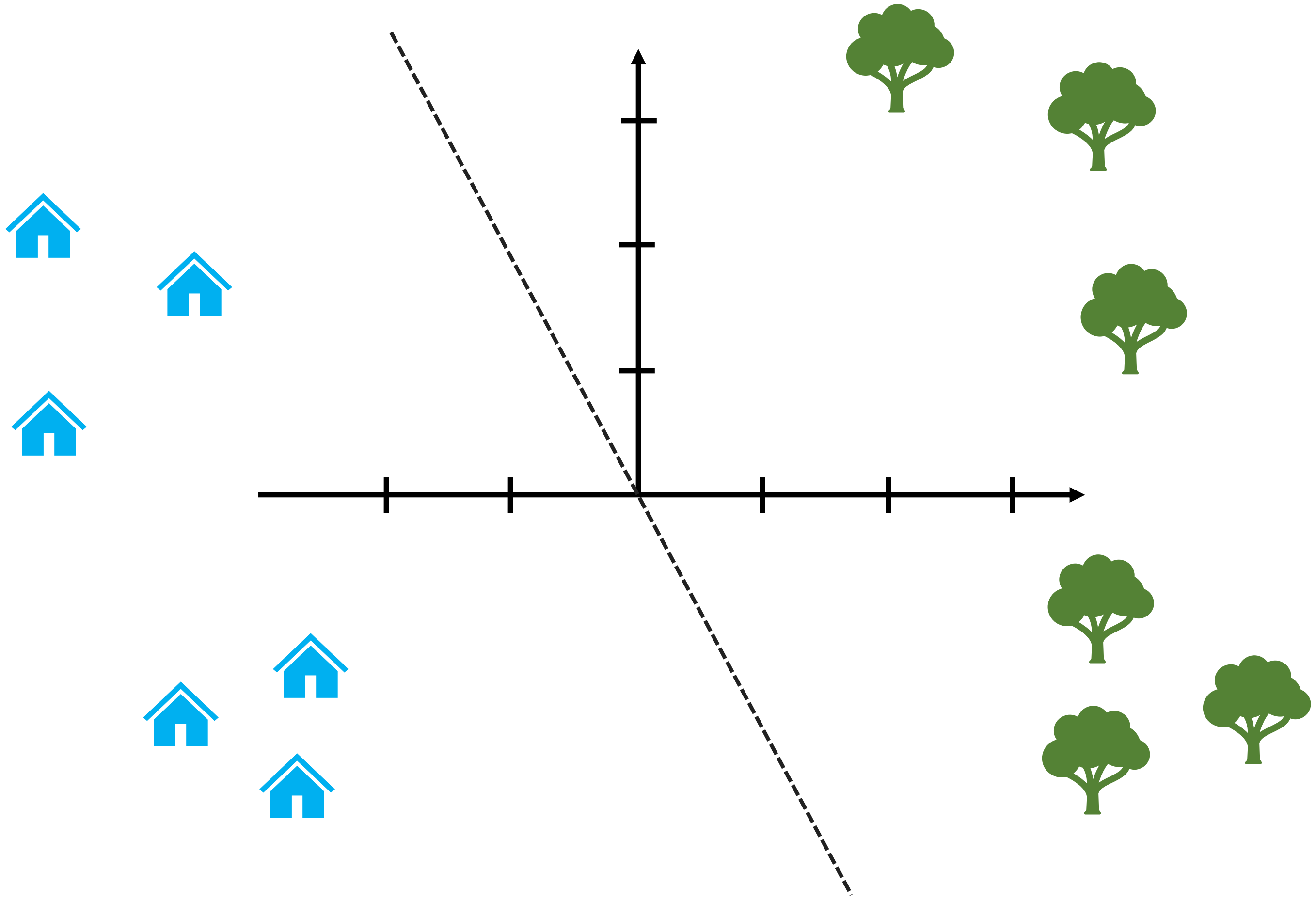
$$[2\ 2]\begin{bmatrix}0\\2\end{bmatrix} = 4$$

Distance:

$$\frac{4}{\sqrt{8}} = \sqrt{2} \approx 1.41$$

Distance of a point u to a zero-offset hyperplane: Compute $|w^T u|$ and divide the result by the length of w: $\frac{|w^T u|}{||w||}$. (We can think of $\frac{|w^T u|}{||w||}$ as the length of the projection of u onto w.)

SVM: Houses, Trees, and The Widest Street

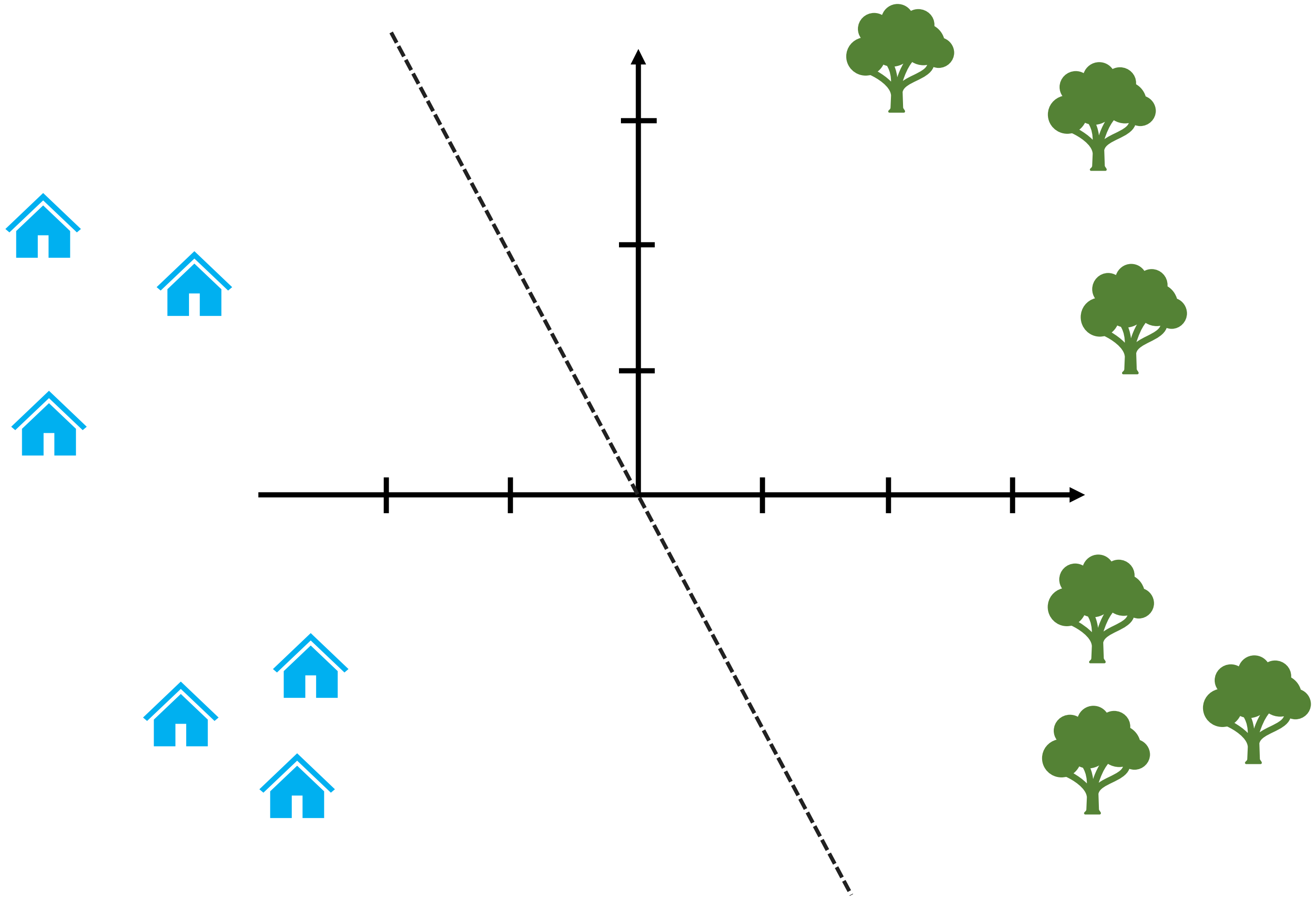


Houses: class 1
Trees: class -1

Street direction: hyperplane
Street width: 2*margin

Assume the houses and trees are linearly separable.

SVM: Houses, Trees, and The Widest Street



- Optimization of the street means changing the direction of the street and maximizing the street width (2*margin)
- This optimization is done by changing the hyperplane normal vector
  - We will not discuss the offset.
- At the same time: maintain the constraints that the houses are on one side, and the trees are on the other side of the hyperplane.
- Some of the houses and trees will be on the edge of the street. These data points are called “support vectors”.

SVM: Data, Model, Objective

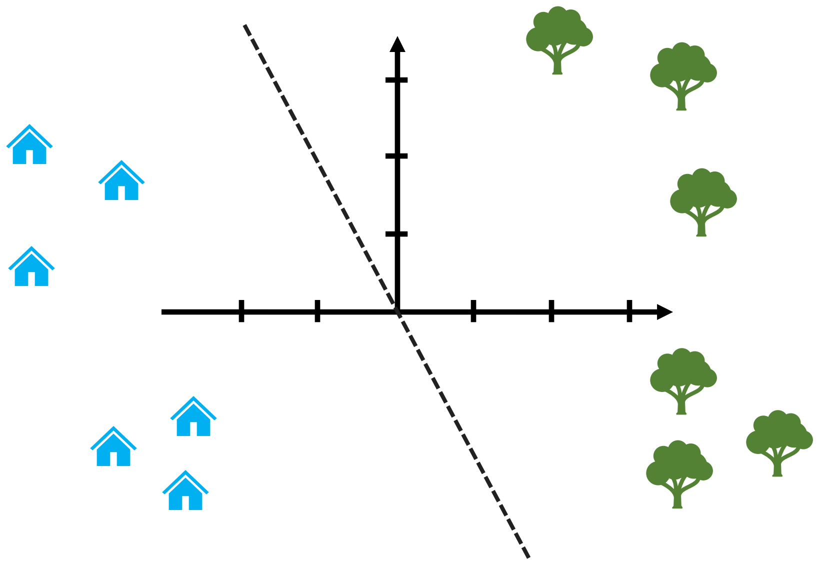
- We are given N data points. Each data point consists of features $x^{(i)}$ and a target variable $y^{(i)}$.
  - The features are described by a vector of real numbers in dimension d.
  - The target is {-1,1}, where -1 is “negative” class and 1 is “positive” class.
- The SVM classifier in the dual formulation is a model that depends on parameters $\alpha^{(i)}$. The model is:

$$SVM_\alpha(x) = \text{sign}(\sum_{i=1}^{N} \alpha^{(i)} k(x^{(i)}, x))$$

- The model is optimized, via optimizing all $\alpha^{(i)}$, to maximize the margin subject to the constraints that data points are on the correct side of the decision boundary.
  - Learning algorithm: quadratic programming, etc

SVM: Sparsity

- Often many of the $\alpha^{(1)}, \alpha^{(2)}, \ldots, \alpha^{(N)}$ will be zero.
- This is because usually only a few data points are exactly on the margin. These data points are called **support vectors**.
- Only a few data points contribute to the classifier (others can be thrown out!)



$$SVM_{\alpha}(x) = \text{sign}\Big(\sum_{\substack{i \in \text{Support} \\ \text{vectors}}} \alpha^{(i)} k(x^{(i)}, x)\Big)$$

SVM: Implementation

You probably don't want to implement SVM from scratch.

```
from sklearn import svm

X = [[0, 0], [1, 1]]
y = [0, 1]

model = svm.SVC()
model.fit(X, y)

model.support_vectors_ # get support vectors

model.predict([[2., 2.]]) # predict unseen data

model.dual_coef_ # obtains the α^(i)
```

Summary

- Regularization
  - The problem of overfitting
  - Regularization
  - Linear regression with regularization
- Kernel Method
  - Dual formulation of linear regression
  - Kernel functions: ~similarity function
  - Feature map ↔ kernels
- Support Vector Machine
  - Classifier with optimal decision boundary

Logistic Regression / SVM
With x as features

Logistic Regression / SVM
With $\boldsymbol{\phi}(\boldsymbol{x})$ as features

SVM with Kernel Trick
With $\boldsymbol{\phi}$(x) mapping to
finite-dimensional
features

SVM with Kernel Trick
With $\boldsymbol{\phi}$(x) mapping to
infinite-dimensional
features

Coming Up Next Lecture

- Unsupervised Learning
  - K-means.
  - Principal component analysis.
  - …

To Do

- **Lecture Training 7**
  - +250 Free EXP
  - +100 Early bird bonus
- **PS3**
- **Mini-project**
  - Due in ~1 month