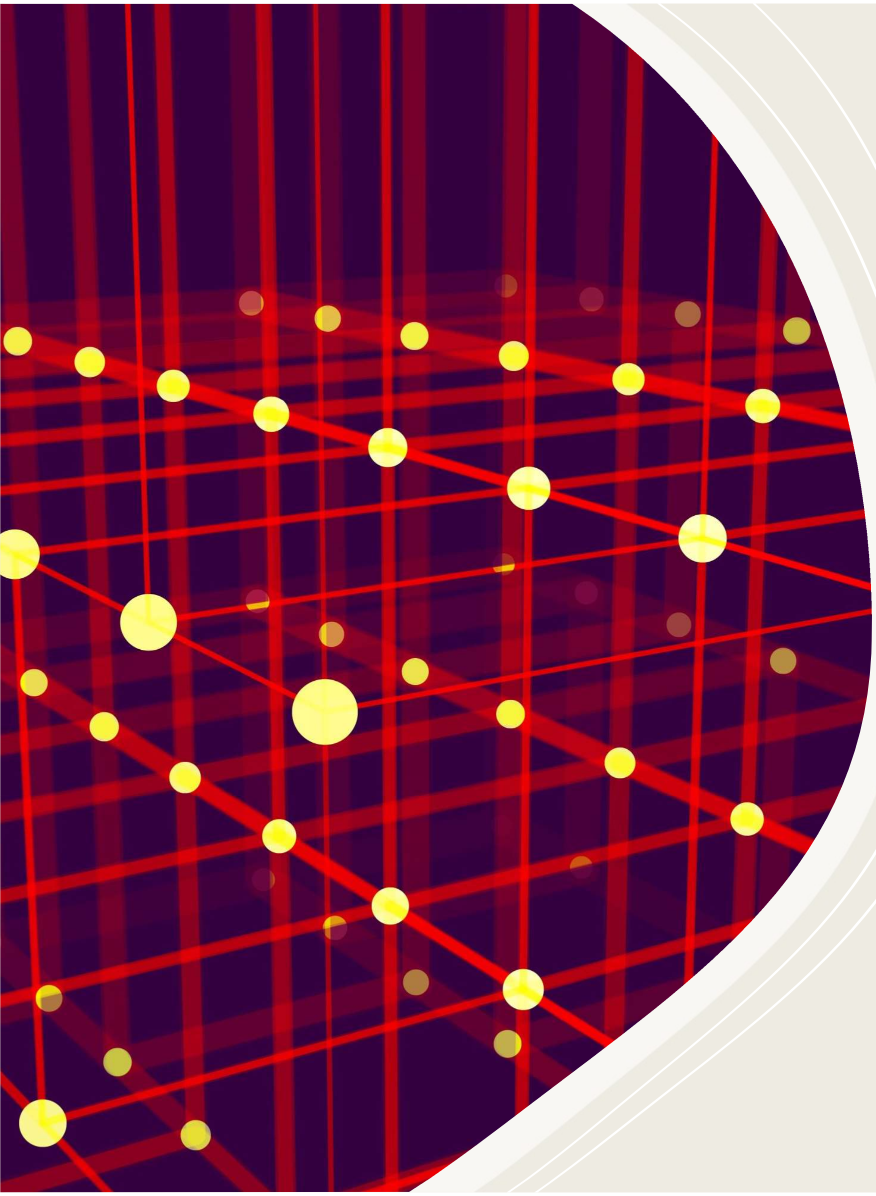


CPSC 340 – Tutorial 5

Michael Liu
mfliu@students.cs.ubc.ca

Template from Lironne Kurzman

Some slides from Nam Hee Kim

University of British Columbia

November 1st, 2021

Agenda

- Multi-class classification
- Kernel Trick

Multi-class Classification

Motivation

- Our linear classifiers only work for binary classification
  - E.g. "important" vs "not important" emails
  - E.g. "cat" vs "no cat" in picture
- We would like to build a classifier that works for $k > 2$ classes
  - E.g. "cat" vs "dog" vs "bird" in picture

Notation Review

- X is an $n \times d$ matrix of **training examples**
- y is an $n \times 1$ vector of **training labels**
- x_i is a $d \times 1$ vector of **feature values** for example i
- y_i is a *scalar* in the range $[1, k]$, where there are k classes
- W is a $k \times d$ matrix of **weights** for each classifier $1 \leq c \leq k$
- w_c is a $d \times 1$ vector of **weights** for the classifier for class c
  - Can also be written as w_{y_i} where $y_i = c$

One-vs-all Linear Classification

- Idea: Independently train a linear classifier for each class c
  - Each classifier predicts how likely x_i is in class c
    - *E.g.* *Classifier 1: x_i is probably not a dog*
      Classifier 2: x_i is probably not a cat
      Classifier 3: x_i is probably a bird
      One-vs-all classifier: Predict x_i is a bird
  - At test time, use argmax to find “best class” for example $\tilde{x}_i$
- Problem: classifiers may operate with different scales
  - How do we compare $w_c^T x_i$ with $w_{c'}^T x_i$?

Multi-class SVM

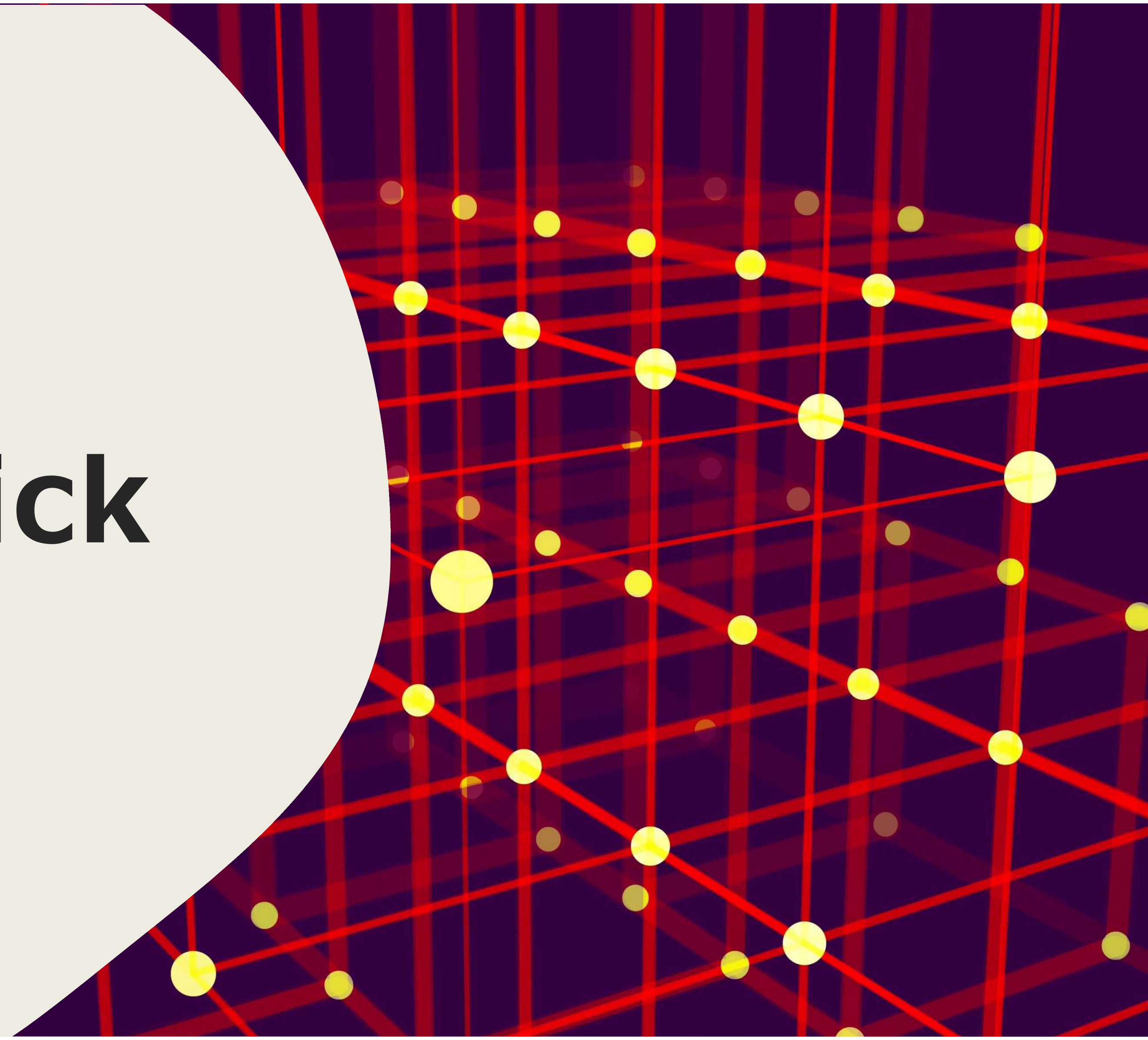
- Goal: Every classifier outputs a value on the same scale
  - ☹ Now we must train all k classifiers together
  - ☹ W is no longer a simple vector
  - ☺ It performs better than one-vs-all linear classification
  - ☺ We only need to implement it once
- We want to make $w_{y_i}^T x_i$ larger than $w_c^T x_i$ for all $c \neq y_i$

Multi-class SVM Loss

$$\sum_{c \neq y_i} \max\left\{0, 1 - w_{y_i}^T x_i + w_c^T x_i\right\}$$

- This is the loss for a **single** example x_i
  - To get the loss over all examples, we can take the sum or average
- The 1 term avoid degeneracy (see binary SVM loss)
- The $-w_{y_i}^T x_i$ term **rewards** large values for the correct class
- The $w_c^T x_i$ term **penalizes** large values for incorrect classes

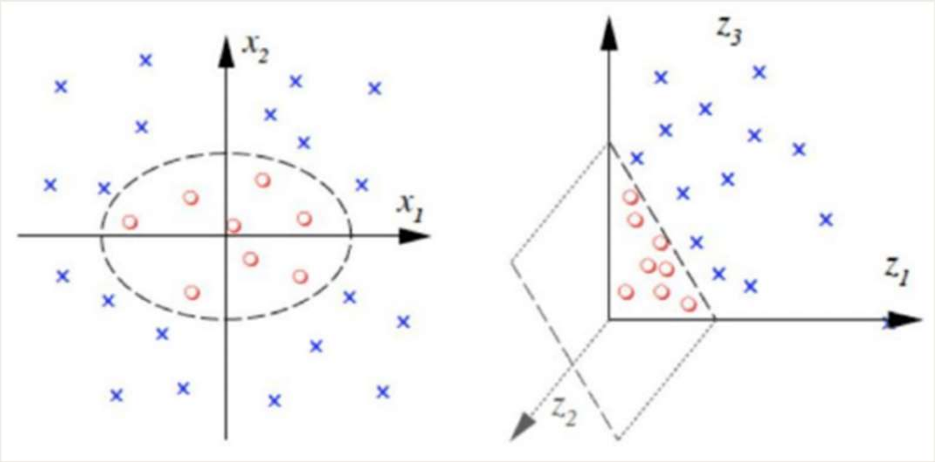
Kernel Trick



Motivation

- We've already learned about linear models
  - Easy to train
  - Easy to test
- We've already learned about change of basis
  - Training and testing are simple (just use linear model)
  - Creating Z matrix is easy, but can take a long time
    - $O(d^p)$ *values for **each** example*

Motivation as a Picture



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

“Other” Normal Equations

- Assume L2-regularized least squares objective with basis Z:

$$f(v) = \frac{1}{2}||Zv - Y||^2 + \frac{\lambda}{2}||v||^2$$

- Normal equations to find minimum v:

$$v = (Z^T Z + \lambda I)^{-1} Z^T Y$$

- Other normal equations to find minimum v:

$$v = Z^T (Z Z^T + \lambda I)^{-1} Y$$

Making Predictions

- If we want to make predictions $\hat{y}$ for test data $\tilde{X}$ by forming $\tilde{Z}$ using "other normal" equations:

$$\begin{aligned}\hat{y} &= \tilde{Z}v \\ &= \overset{\tilde{K}}{\boxed{\tilde{Z}Z^T}}(\overset{K}{\boxed{ZZ^T}} + \lambda I)^{-1}Y \\ &= \tilde{K}(K + \lambda I)^{-1}Y\end{aligned}$$

- Efficiently compute K and $\tilde{K}$ even though forming Z and $\tilde{Z}$ is intractable.

Kernel Functions

- Kernel function does not calculate Z explicitly. Returns similarity between transformed points Z_i, Z_j :

$$K(X_i, X_j) = Z_i^T Z_j$$

using only untransformed points X_i, X_j.

Example Kernels

- Linear Kernel:

$$K(X_i, X_j) = Z_i^T Z_j = X_i^T X_j$$

- Degree P polynomial Kernel:

$$K(X_i, X_j) = Z_i^T Z_j = (1 + X_i^T X_j)^P$$

- Gaussian RBF Kernel:

$$K(X_i, X_j) = Z_i^T Z_j = \exp(-\frac{||X_i - X_j||^2}{2\sigma^2})$$