CPSC 340: Machine Learning and Data Mining

Kernel Trick

Summer 2021

In This Lecture

1. Kernel Trick (30 minutes)
2. Stochastic Gradient Descent Intro (15 minutes)

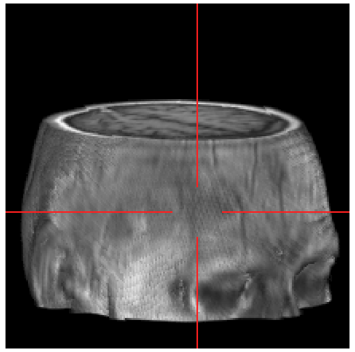
Coming Up Next

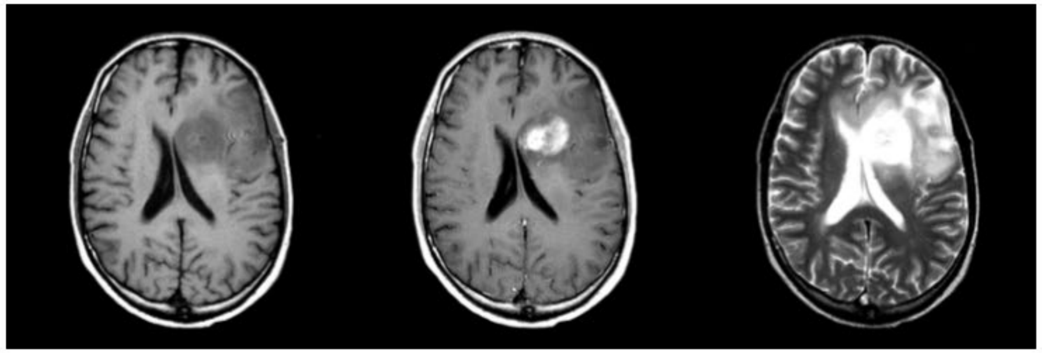
PROBLEM WITH HIGH-DIMENSIONAL BASIS

Motivation: Automatic Brain Tumor Segmentation

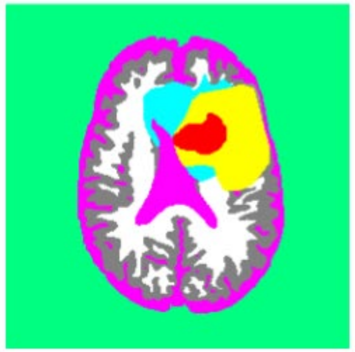
- Task: segmentation tumors and normal tissue in multi-modal MRI data.
  - We previously discussed using convolutions to engineer features.

Input:





Output:

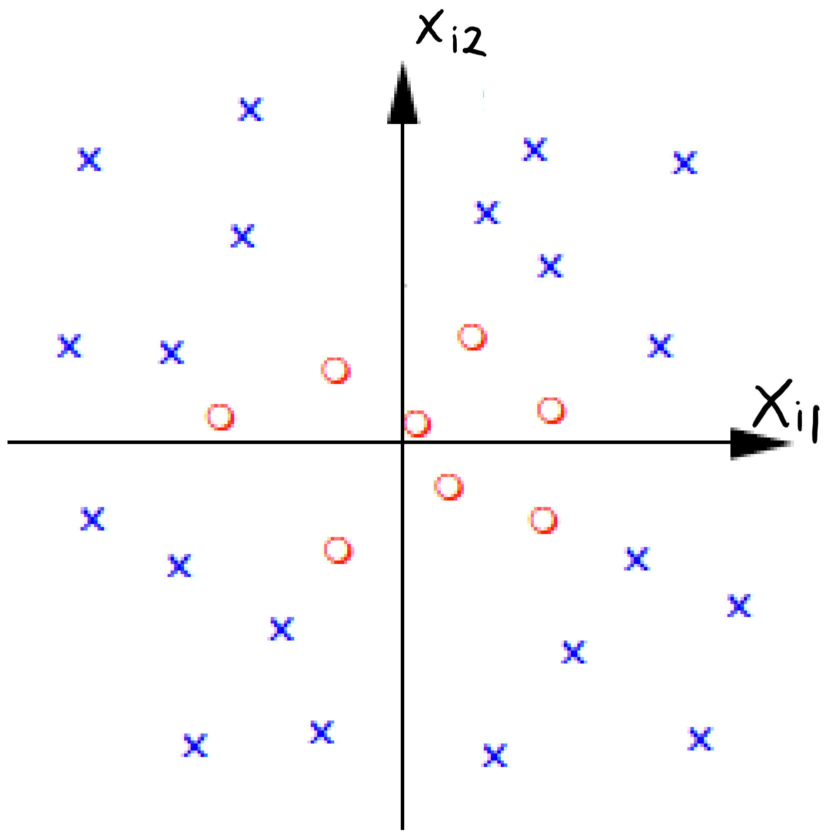


- Best performance was obtained with linear classifiers (SVMs/logistic).
  - Provided you did feature selection or used regularization.

- One of the only methods that worked better:
  - Regularized linear classifier with a low-order polynomial basis ($p=2$ or $p=3$).
    - Makes the data "closer to separable" in the higher-dimensional space.

Support Vector Machines for Non-Separable

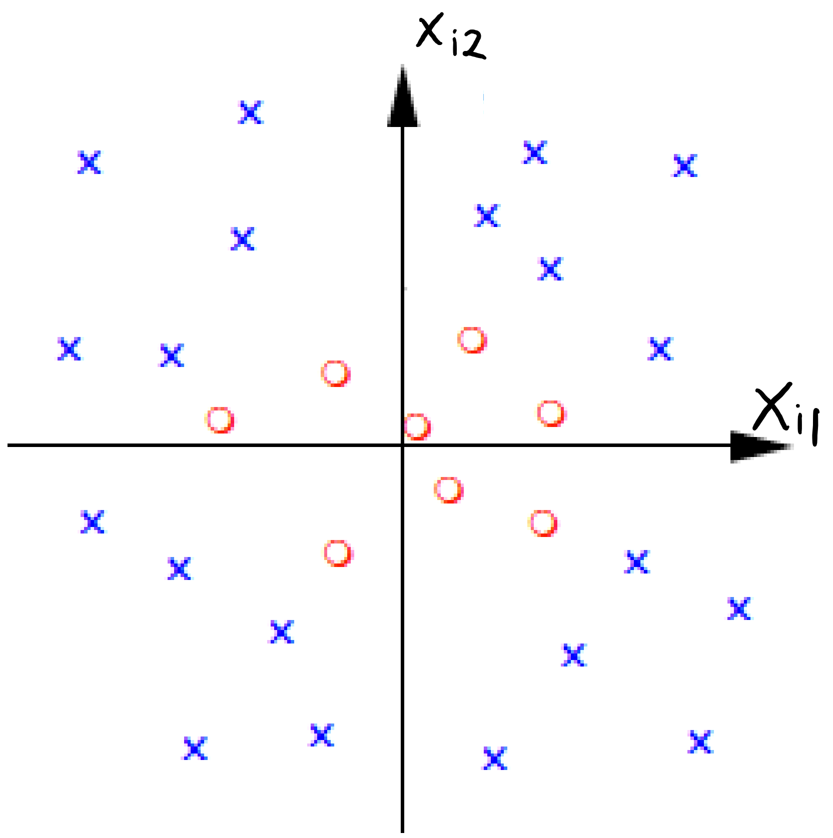
- Can we use linear models for data that is not close to separable?

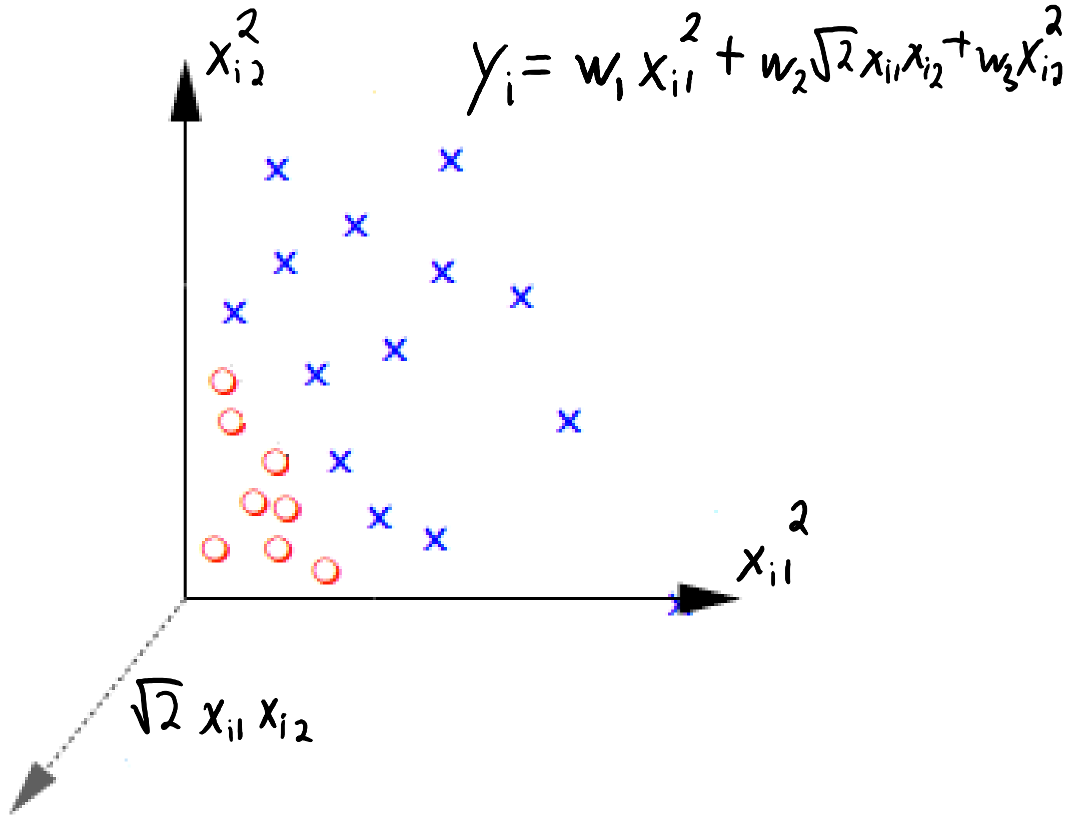


http://math.stackexchange.com/questions/353607/how-do-inner-product-space-determine-half-planes

Support Vector Machines for Non-Separable

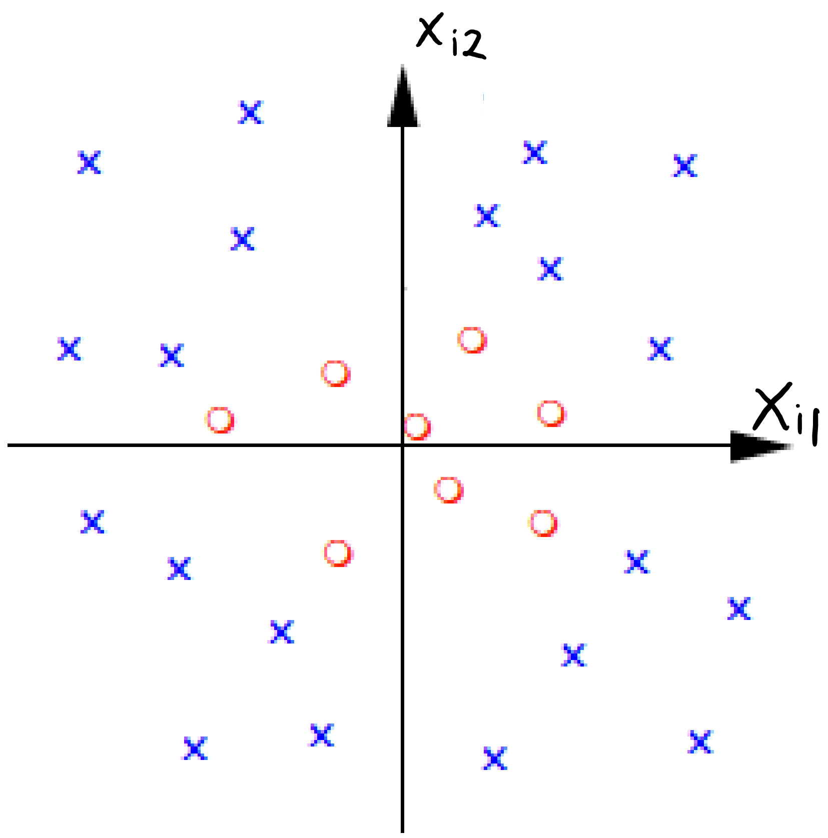
- Can we use linear models for data that is not close to separable?
  - It may be separable under change of basis (or closer to separable).

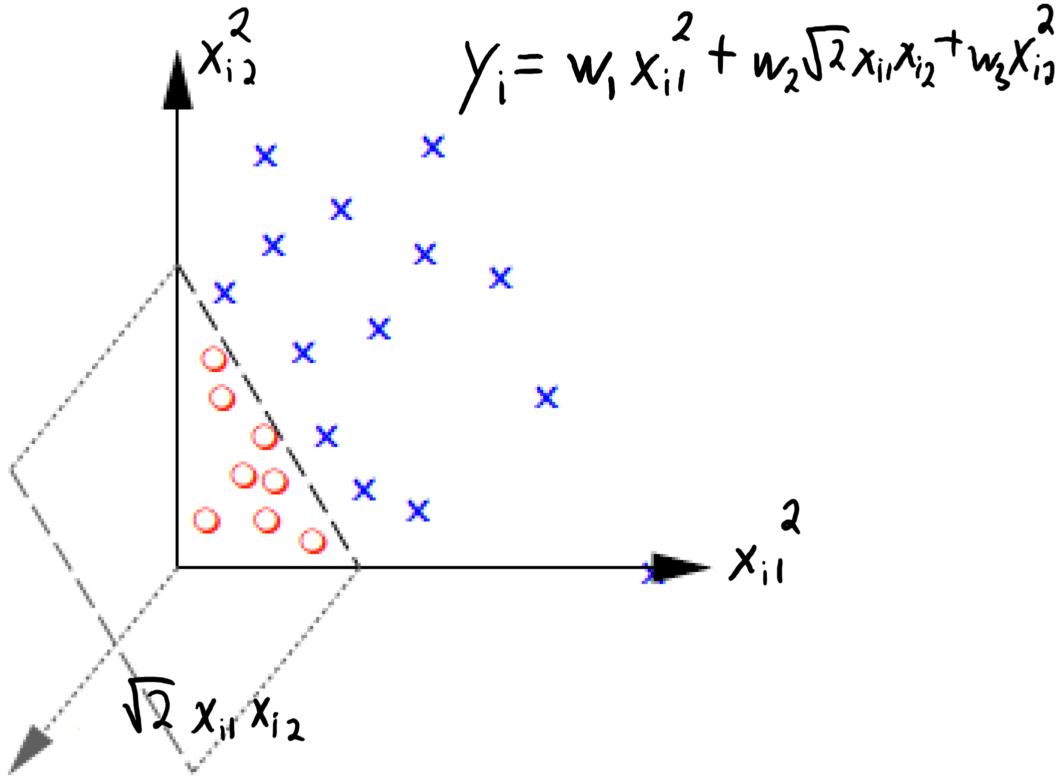




Support Vector Machines for Non-Separable

- Can we use linear models for data that is not close to separable?
  - It may be separable under change of basis (or closer to separable).

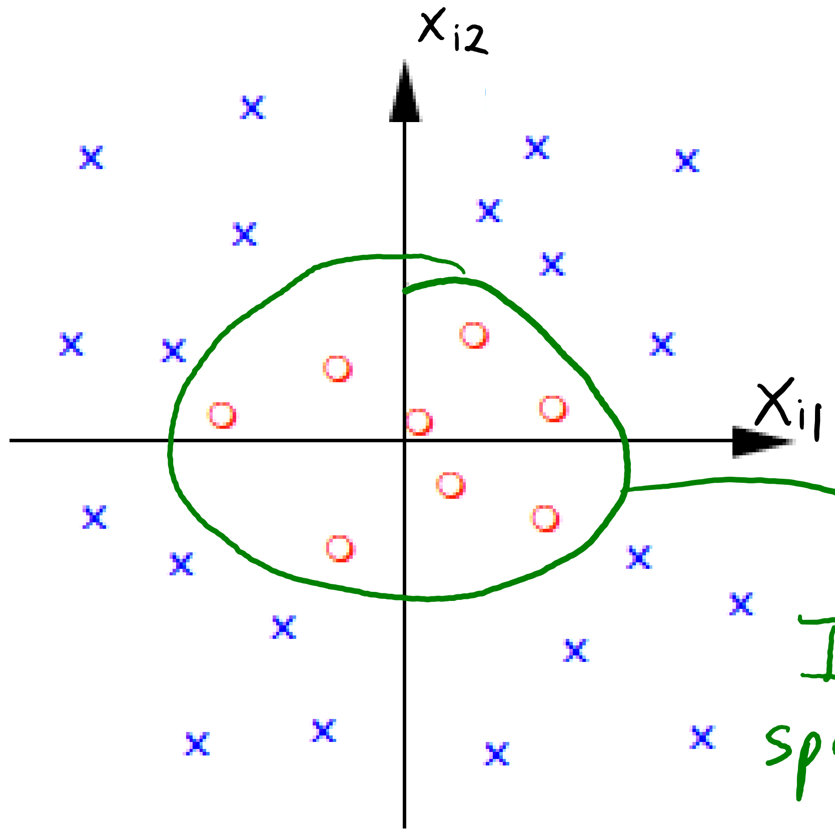


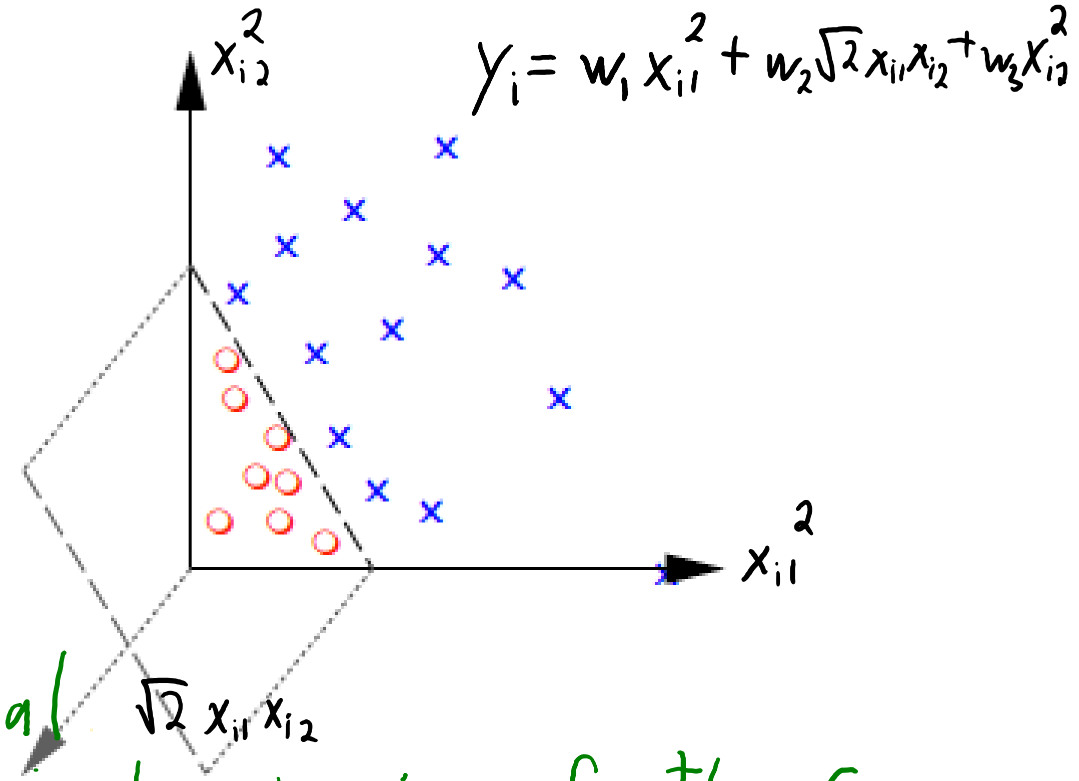


http://math.stackexchange.com/questions/353607/how-do-inner-product-space-determine-half-planes

Support Vector Machines for Non-Separable

- Can we use linear models for data that is not close to separable?
  - It may be separable under change of basis (or closer to separable).





In original space, decision boundary is of the form

$$0 = w_1 x_{i1}^2 + w_2 \sqrt{2} x_{i1} x_{i2} + w_3 x_{i2}^2$$

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

Multi-Dimensional Polynomial Basis

- Recall fitting polynomials when we only have 1 feature:

$$\hat{y}_i = w_0 + w_1 x_i + w_2 x_i^2$$

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

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

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

Multi-Dimensional Polynomial Basis

- Polynomial basis for d=2 and p=2:

$$X = \begin{bmatrix} 0.2 & 0.3 \\ 1 & 0.5 \\ -0.5 & -0.1 \end{bmatrix} \longrightarrow Z = \begin{array}{c} \begin{bmatrix} 1 & 0.2 & 0.3 & (0.2)^2 & (0.3)^2 & (0.1)(0.3) \\ 1 & 1 & 0.5 & (1)^2 & (0.5)^2 & (1)(0.5) \\ 1 & 0.5 & -0.1 & (0.5)^2 & (-0.1)^2 & (-0.5)(-0.1) \end{bmatrix} \\ \begin{array}{cccccc} \underbrace{\quad}_{bias} & \underbrace{\quad}_{x_{i1}} & \underbrace{\quad}_{x_{i2}} & \underbrace{\quad}_{(x_{i1})^2} & \underbrace{\quad}_{(x_{i2})^2} & \underbrace{\qquad}_{(x_{i1})(x_{i2})} \end{array} \end{array}$$

- With d=4 and p=3, the polynomial basis would include:
  - Bias variable and the x_{ij}: 1, x_{i1}, x_{i2}, x_{i3}, x_{i4}.
  - The x_{ij} squared and cubed: $(x_{i1})^2$, $(x_{i2})^2$, $(x_{i3})^2$, $(x_{i4})^2$, $(x_{i1})^3$, $(x_{i2})^3$, $(x_{i3})^3$, $(x_{i4})^3$.
  - Two-term interactions: $x_{i1}x_{i2}$, $x_{i1}x_{i3}$, $x_{i1}x_{i4}$, $x_{i2}x_{i3}$, $x_{i2}x_{i4}$, $x_{i3}x_{i4}$.
  - Cubic interactions: $x_{i1}x_{i2}x_{i3}$, $x_{i2}x_{i3}x_{i4}$, $x_{i1}x_{i3}$,x_{i4}, $x_{i1}x_{i2}x_{i4}$,
    $x_{i1}^2x_{i2}$, $x_{i1}^2x_{i3}$, $x_{i1}^2x_{i4}$, $x_{i1}x_{i2}^2$, $x_{i2}^2x_{i3}$, $x_{i2}^2x_{i4}$, $x_{i1}x_{i3}^2$, $x_{i2}x_{i3}^2$,$x_{i3}^2x_{i4}$, $x_{i1}x_{i4}^2$, $x_{i2}x_{i4}^2$, $x_{i3}x_{i4}^2$.

Kernel Trick

- If we go to degree p=5, we'll have O(d⁵) quintic terms:

$$x_{i1}^5, x_{i1}^4 x_{i2}, x_{i1}^4 x_3, \ldots, x_{i1}^4 x_{id}, x_{i1}^3 x_{i2}^2, x_{i1}^3 x_{i3}^2, \ldots, x_{i1}^3 x_{id}^2, \ldots, x_{i2}^5, x_{i2}^4 x_{i3}, \ldots \ldots, x_{id}^5$$

- For large 'd' and 'p', storing a polynomial basis is intractable!
  - 'Z' has $k=O(d^p)$ columns, so it does not fit in memory.

- Could try to search for a good subset of these.
  - "Hierarchical forward selection" (bonus).

- Alternating, you can use all of them with the "kernel trick".
  - For special case of L2-regularized linear models.

Coming Up Next

GRAM MATRIX AND THE KERNEL TRICK

How can you use an exponential-sized basis?

- Which of these two expressions would you rather compute?

$$x^9+9x^8+36x^7+84x^6+126x^5+126x^4+84x^3+36x^2+9x+1 \quad \text{or} \quad (x+1)^9$$

  - Expressions are equal, but left costs O(_) while right costs O(_).

- Which of these two expressions would you rather compute?

$$1+x+\frac{x^2}{2!}+\frac{x^3}{3!}+\frac{x^4}{4!}+\frac{x^5}{5!}+\frac{x^6}{6!}\dots \quad \text{or} \quad e^x$$

  - Expressions are equal, but left has infinite terms and right costs O(_).

- Maybe we can somehow add weights to the expressions on the left, and formulate least squares to use tricks like on the right?

The “Other” Normal Equations

- Recall the L2-regularized least squares objective with basis ‘Z’:

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

- We showed that the minimum is given by

$$v = (\underbrace{Z^TZ + \lambda I}_{k \times k})^{-1} Z^T y$$

- (in practice you still solve the linear system, since inverse is less numerically unstable – see CPSC 302)
- With some work (bonus), this can equivalently be written as:

$$v = Z^T(\underbrace{ZZ^T + \lambda I}_{n \times n})^{-1} y$$

- This is faster if n << k:
  - After forming ‘Z’, cost is $O(n^2k + n^3)$ instead of $O(nk^2 + k^3)$.
  - But for the polynomial basis, this is still too slow since $k = O(d^p)$.

The "Other" Normal Equations

- With the "other" normal equations we have $v = Z^T(ZZ^T + \lambda I)^{-1}y$
- Given test data $\tilde{X}$, predict $\hat{y}$ by forming $\tilde{Z}$ and then using:

$$\begin{aligned} \hat{y} &= \tilde{Z}v \\ &= \underbrace{\tilde{Z}Z^T}_{\tilde{K}}(\underbrace{ZZ^T}_{K} + \lambda I)^{-1}y \\ \underset{t\times 1}{} &= \underbrace{\tilde{K}}_{t\times n}\underbrace{(K + \lambda I)^{-1}}_{n\times n}\underbrace{y}_{n\times 1} \end{aligned}$$

- Notice that if you have K and $\tilde{K}$ then you do not need Z and $\tilde{Z}$.

The "Other" Normal Equations

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

$$\begin{aligned}\hat{y} &= \tilde{Z}v \\ &= \underbrace{\tilde{Z}Z^T}_{\tilde{K}}(\underbrace{ZZ^T}_{K} + \lambda I)^{-1}y \\ \underset{t \times 1}{} &= \underbrace{\tilde{K}}_{t \times n}\underbrace{(K + \lambda I)^{-1}}_{n \times n}\underbrace{y}_{n \times 1}\end{aligned}$$

- "kernel trick": for certain bases (like polynomials), We can efficiently compute K and $\tilde{K}$ even though forming Z and $\tilde{Z}$ is intractable.
  - In the same way we can comptue $(x+1)^9$ instead of $x^9 + 9x^8 + 36x^7 + 84x^6$…

Gram Matrix: Training

- The matrix K = ZZ^T is called the Gram matrix K.

$$K = ZZ^T = \underbrace{\begin{bmatrix} — & z_1^T & — \\ — & z_2^T & — \\ & \vdots & \\ — & z_n^T & — \end{bmatrix}}_{Z} \underbrace{\begin{bmatrix} | & | & & | \\ z_1 & z_2 & \cdots & z_n \\ | & | & & | \end{bmatrix}}_{Z^T}$$

$$= \underbrace{\left.\begin{bmatrix} z_1^Tz_1 & z_1^Tz_2 & \cdots & z_1^Tz_n \\ z_2^Tz_1 & z_2^Tz_2 & \cdots & z_2^Tz_n \\ \vdots & & \ddots & \vdots \\ z_n^Tz_1 & z_n^Tz_2 & \cdots & z_n^Tz_n \end{bmatrix}\right\} n}_{n}$$

- K contains the dot products between all training examples.
  - Similar to ‘Z’ in RBFs, but using dot product as “similarity” instead of distance.

Gram Matrix: Prediction

- The matrix $\tilde{K} = \tilde{Z}Z^T$ has dot products between train and test examples:

$$\tilde{K} = \tilde{Z}Z^T = \underbrace{\begin{bmatrix} \text{---} \; \tilde{z}_1^T \; \text{---} \\ \text{---} \; \tilde{z}_2^T \; \text{---} \\ \vdots \\ \text{---} \; \tilde{z}_t^T \; \text{---} \end{bmatrix}}_{\tilde{Z}} \underbrace{\begin{bmatrix} | & | & & | \\ z_1 & z_2 & \cdots & z_n \\ | & | & & | \end{bmatrix}}_{Z^T}$$

$$= \left.\underbrace{\begin{bmatrix} \tilde{z}_1^T z_1 & \tilde{z}_1^T z_2 & \cdots & \tilde{z}_1^T z_n \\ \tilde{z}_2^T z_1 & \tilde{z}_2^T z_2 & \cdots & \tilde{z}_2^T z_n \\ \vdots & \vdots & \ddots & \vdots \\ \tilde{z}_t^T z_n & \tilde{z}_t^T z_2 & \cdots & \tilde{z}_t^T z_n \end{bmatrix}}_{n}\right\} t$$

- Kernel function: $k(x_i, x_j) = z_i^T z_j$.
  - Computes dot product between in basis ($z_i^T z_j$) *using original features* x_i and x_j.

Interpreting Kernel Function

$$\underset{d\times 1}{x_i} \longrightarrow \underset{k\times 1}{z_i}$$

$$\underset{d\times 1}{x_j} \longrightarrow \underset{k\times 1}{z_j}$$

Change of basis: project examples into another (more complex) feature space

$$\underset{1\times k}{z_i^T}\,\underset{k\times 1}{z_j} \neq \underset{1\times d}{x_i^T}\,\underset{d\times 1}{x_j}$$

Dot product (“similarity”) between examples i and j in the new feature space

$$K(x_i, x_j) = z_i^T z_j$$

Gram Matrix as "Change of Basis"

$$K = n\begin{bmatrix} k(x_1,x_1) & k(x_1,x_2) & \cdots & k(x_1,x_n) \\ k(x_2,x_1) & k(x_2,x_2) & \cdots & k(x_2,x_n) \\ \vdots & & & \\ k(x_n,x_1) & \cdots & & k(x_n,x_n) \end{bmatrix}_{n}$$

- A row of the matrix K: k_i
  - Results of kernel function between x_i and other examples
  - Kernel regression: let's use these as features!

Linear Regression vs. Kernel Regression

Linear Regression (L2-reg)

Training

1. Form basis Z from X.
2. Compute $\underbrace{v}_{k\times 1} = (Z^TZ + \lambda I)^{-1}(Z^Ty)$

Testing

1. Form basis $\tilde{Z}$ from $\tilde{X}$
2. Compute $\hat{y} = \underbrace{\tilde{Z}}_{t\times k}\underbrace{v}_{k\times 1}$

Kernel Regression (L2-reg)

Training:

1. Form inner products K from X.
2. Compute $\underbrace{u}_{n\times 1} = (K + \lambda I)^{-1} y$

Testing:

1. Form inner products $\tilde{K}$ from X and $\tilde{X}$

Non-parametric

2. Compute $\hat{y} = \underbrace{\tilde{K}}_{t\times n}\underbrace{u}_{n\times 1}$

(Everything you need to know about Z and $\tilde{Z}$ is contained within K and $\tilde{K}$)

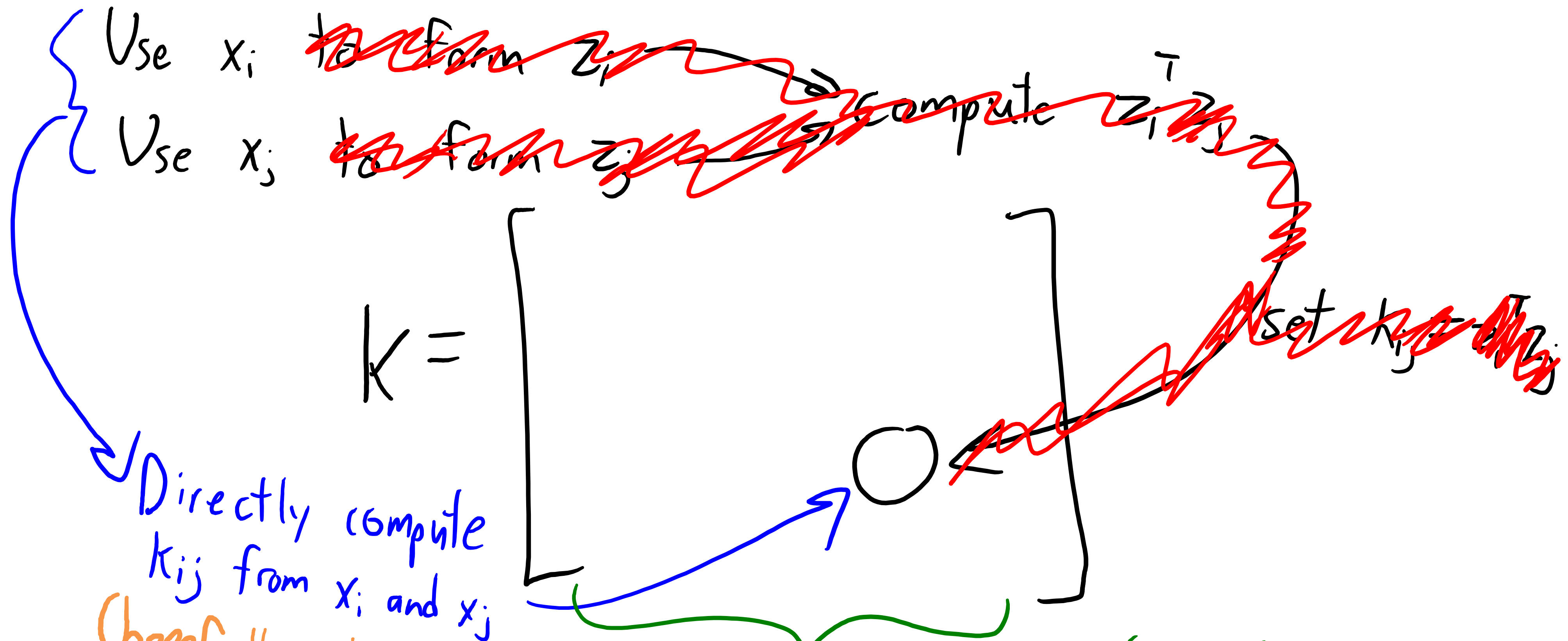
Linear Regression vs. Kernel Regression

To apply linear regression, I only need to know K and $\tilde{K}$

$$\begin{tikzcd}
\text{Use } x_i \text{ to form } z_i \arrow[dr, bend left] & \\
& \text{compute } z_i^T z_j \arrow[d, bend left, "\text{set } k_{ij} = z_i^T z_j"] \\
\text{Use } x_j \text{ to form } z_j \arrow[ur] & K = \underbrace{\begin{bmatrix} \quad & \quad & \quad \\ & & \bigcirc \\ & & \end{bmatrix}}_{\text{Final result is } n \times n \text{ (no matter how large } z_i \text{ is)}}
\end{tikzcd}$$

Linear Regression vs. Kernel Regression

To apply linear regression, I only need to know K and $\tilde{K}$



Use x_i ~~to form z_i~~

Use x_j ~~to form z_j~~

~~compute $z_i^T z_j$~~

~~set $k_{ij} = z_i^T z_j$~~

$$K = \begin{bmatrix} & & \\ & & \bigcirc \\ & & \end{bmatrix}$$

Directly compute k_{ij} from x_i and x_j

(hopefully cheap)

Final result is $n \times n$ (no matter how large z_i is)

Degenerate Example: "Linear 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 our standard ("linear") basis:

$$z_i = (x_{i1}, x_{i2}) \qquad z_j = (x_{i1}, x_{i2})$$

- In this case the inner product $z_i^T z_j$ is $k(x_i, x_j) = x_i^T x_j$:

$$\underset{\overset{\uparrow}{x_i}}{z_i^T}\,\underset{\overset{\uparrow}{x_j}}{z_j} = x_i^T x_j$$

Example: Degree-2 Kernel

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

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

- Now consider a particular degree-2 basis:

$$z_i = (x_{i1}^2, \sqrt{2}\, x_{i1} x_{i2}, x_{i2}^2) \qquad z_j = (x_{j1}^2, \sqrt{2}\, x_{j1} x_{j2}, x_{j2}^2)$$

- In this case the inner product $z_i^T z_j$ is $k(x_i,x_j) = (x_i^T x_j)^2$:

[1] $$z_i^T z_j = x_{i1}^2 x_{j1}^2 + (\sqrt{2}\, x_{i1} x_{i2})(\sqrt{2}\, x_{j1} x_{j2}) + x_{i2}^2 x_{j2}^2$$

[2] $$= x_{i1}^2 x_{j1}^2 + 2 x_{i1} x_{i2} x_{j1} x_{j2} + x_{i1}^2 x_{i2}^2$$

[3] $$= (\underbrace{x_{i1} x_{j1} + x_{i2} x_{j2}}_{x_i^T x_j})^2$$ "completing the square"

[4] $$= (x_i^T x_j)^2 \leftarrow$$ No <u>need</u> for z_i to compute $z_i^T z_j$

Coming Up Next

POLYNOMIAL AND GAUSSIAN RBF KERNELS

Polynomial Kernel with Higher Degrees

- Let's add a bias and linear terms to our degree-2 basis:

$$z_i = \begin{bmatrix} 1 & \sqrt{2}x_{i1} & \sqrt{2}x_{i2} & x_{i1}^2 & \sqrt{2}x_{i1}x_{i2} & x_{i2}^2 \end{bmatrix}^T$$

- In this case the inner product $z_i^T z_j$ is $k(x_i,x_j) = (1 + x_i^T x_j)^2$:

[1] $(1 + x_i^T x_j)^2 = 1 + 2x_i^T x_j + (x_i^T x_j)^2$

[2] $= 1 + 2x_{i1}x_{j1} + 2x_{i2}x_{j2} + x_{i1}^2 x_{j1}^2 + 2x_{i1}x_{i2}x_{j1}x_{j2} + x_{i2}^2 x_{j2}^2$

[3] $$= \underbrace{\begin{bmatrix} 1 & \sqrt{2}x_{i1} & \sqrt{2}x_{i2} & x_{i1}^2 & \sqrt{2}x_{i1}x_{i2} & x_{i2}^2 \end{bmatrix}}_{z_i^T} \underbrace{\begin{bmatrix} 1 \\ \sqrt{2}x_{j1} \\ \sqrt{2}x_{j2} \\ x_{j2}^2 \\ \sqrt{2}x_{j1}x_{j2} \\ x_{j2}^2 \end{bmatrix}}_{z_j}$$

[4] $= z_i^T z_j$

Polynomial Kernel with Higher Degrees

- To get all degree-4 "monomials" I can use:

$$k(x_i, x_j) = (x_i^T x_j)^4$$

Equivalent to using a z_i with weighted versions of $x_{i1}^4, x_{i1}^3 x_{i2}, x_{i1}^2 x_{i2}^2, x_{i1} x_{i2}^3, x_{i2}^4, \ldots$

- To also get lower-order terms use $k(x_i,x_j) = (1 + x_i^T x_j)^4$
- The general degree-p polynomial kernel function:

$$k(x_i, x_j) = (1 + x_i^T x_j)^p$$

  - Works for any number of features 'd'.
  - But cost of computing one $k(x_i,x_j)$ is O(_) instead of O(__) to compute $z_i^T z_j$.
  - Take-home message: I can compute dot-products without the features.

Kernel Trick with Polynomials

- Using polynomial basis of degree ‘p’ with the kernel trick:
  - Compute K and $\tilde{K}$ using:

$$K_{ij} = (1 + x_i^T x_j)^p \qquad \tilde{K}_{ij} = (1 + \underbrace{\tilde{x}_i}_{\text{test example}}{}^T \underbrace{x_j}_{\text{train example}})^p$$

  - Make predictions using:

$$\underbrace{\hat{y}}_{t\times 1} = \underbrace{\tilde{K}}_{t\times n}\underbrace{(K+\lambda I)^{-1}}_{n\times n}\underbrace{y}_{n\times 1} = \tilde{K}u \qquad u = (K+\lambda I)^{-1}y$$

- Training cost is only O(__________), despite using k=O(__) features.
  - We can form ‘K’ in O(n²d), and we need to “invert” an ‘n x n’ matrix.
  - Testing cost is only O(___), cost to form $\tilde{K}$.

Gaussian-RBF Kernel

- Most common kernel is the Gaussian RBF kernel:

$$K(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$$

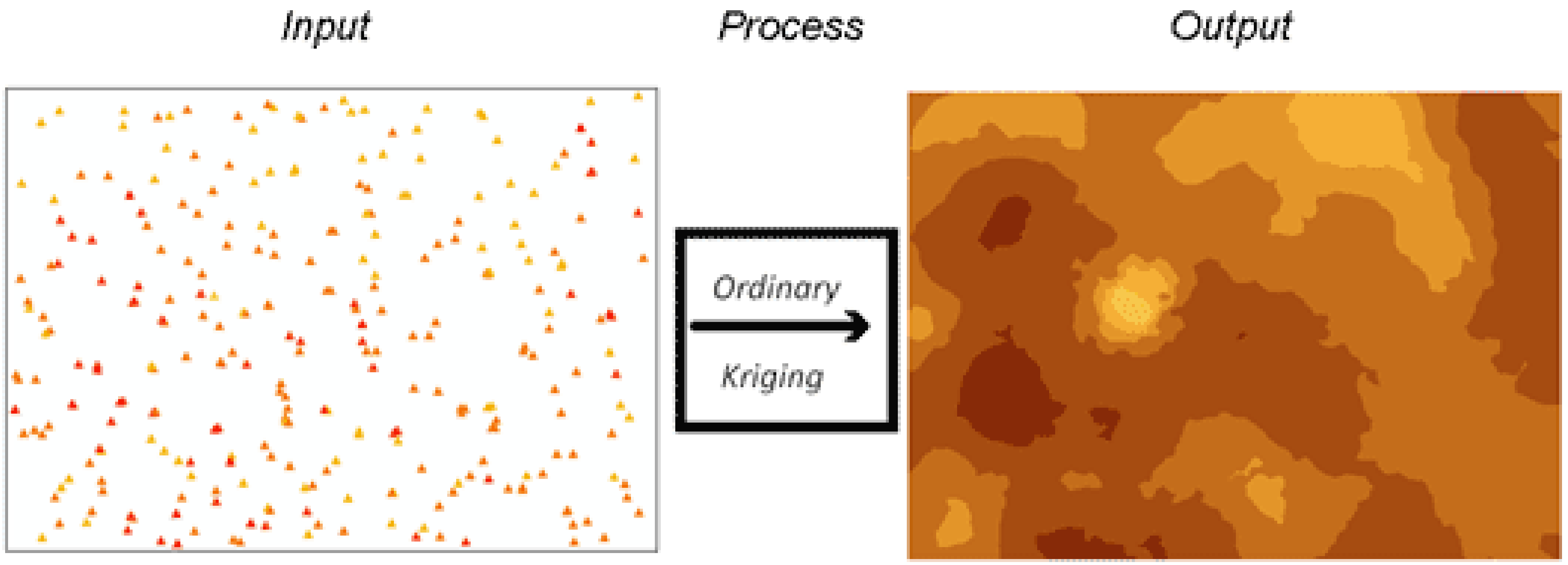
- Same formula and behaviour as RBF basis, but not equivalent:
  - Before we used RBFs as a basis, now we're using them as inner-product.

- Basis z_i giving Gaussian RBF kernel is infinite-dimensional.
  - If d=1 and σ=1, it corresponds to using this basis (bonus slide):

$$z_i = \exp(-x_i^2)\left[1 \quad \sqrt{\tfrac{2}{1!}}\,x_i \quad \sqrt{\tfrac{2^2}{2!}}\,x_i^2 \quad \sqrt{\tfrac{2^3}{3!}}\,x_i^3 \quad \sqrt{\tfrac{2^4}{4!}}\,x_i^4 \quad \cdots\cdots\right]$$

Motivation: Finding Gold

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



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

Kernel Trick for Non-Vector Data

- Consider data that doesn’t look like this:

$$X = \begin{bmatrix} 0.5377 & 0.3188 & 3.5784 \\ 1.8339 & -1.3077 & 2.7694 \\ -2.2588 & -0.4336 & -1.3499 \\ 0.8622 & 0.3426 & 3.0349 \end{bmatrix}, \quad y = \begin{bmatrix} +1 \\ -1 \\ -1 \\ +1 \end{bmatrix},$$

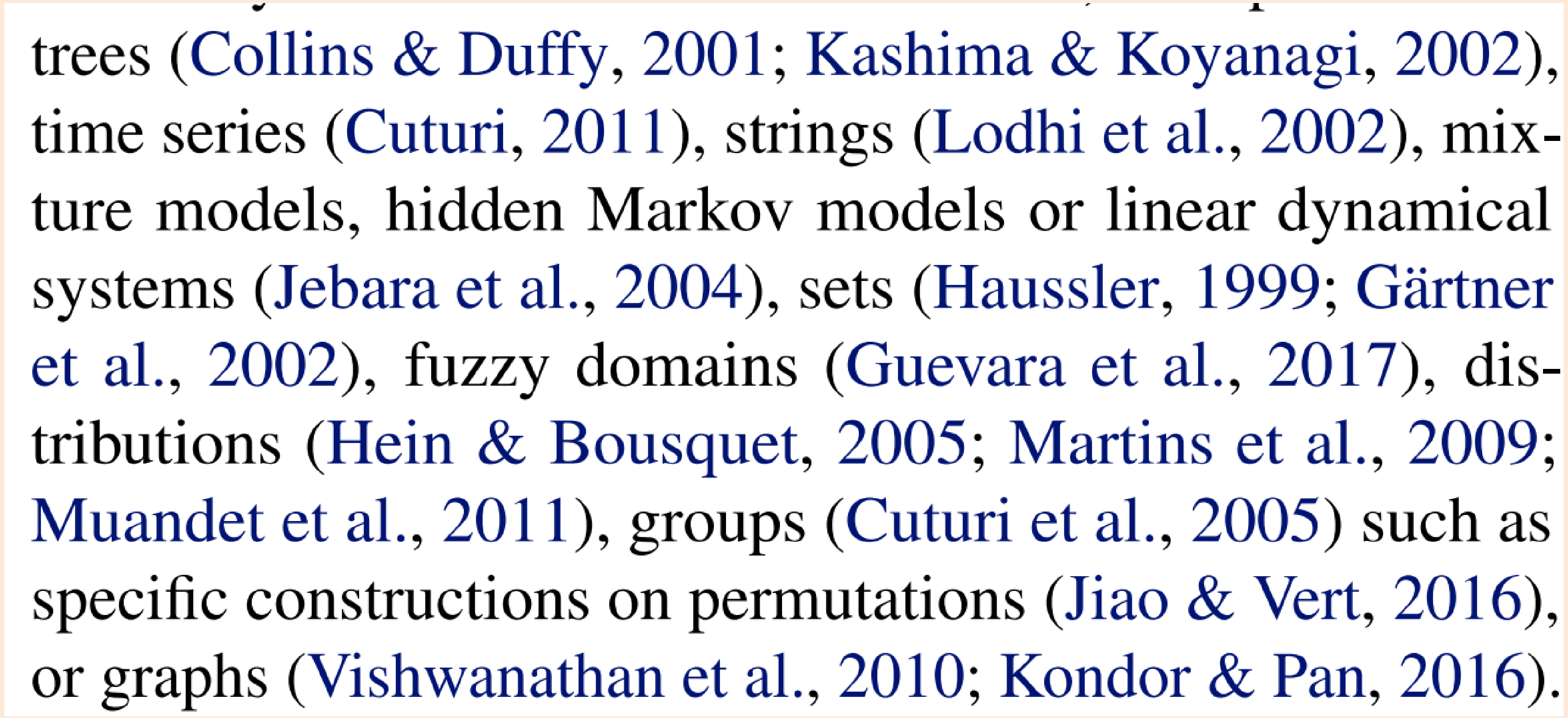
- But instead looks like this:

$$X = \begin{bmatrix} \text{Do you want to go for a drink sometime?} \\ \text{J'achète du pain tous les jours.} \\ \text{Fais ce que tu veux.} \\ \text{There are inner products between sentences?} \end{bmatrix}, y = \begin{bmatrix} +1 \\ -1 \\ -1 \\ +1 \end{bmatrix}.$$

- We can interpret $k(x_i,x_j)$ as a “similarity” between objects xi and xj.
  - We don’t need features if we can compute “similarity” between objects.
  - Kernel trick lets us fit regression models without explicit features.
  - There are “string kernels”, “image kernels”, “graph kernels”, and so on.

Kernel Trick for Non-Vector Data

- Recent list of types of data where people have defined kernels:



trees (Collins & Duffy, 2001; Kashima & Koyanagi, 2002), time series (Cuturi, 2011), strings (Lodhi et al., 2002), mixture models, hidden Markov models or linear dynamical systems (Jebara et al., 2004), sets (Haussler, 1999; Gärtner et al., 2002), fuzzy domains (Guevara et al., 2017), distributions (Hein & Bousquet, 2005; Martins et al., 2009; Muandet et al., 2011), groups (Cuturi et al., 2005) such as specific constructions on permutations (Jiao & Vert, 2016), or graphs (Vishwanathan et al., 2010; Kondor & Pan, 2016).

- Bonus slide overviews a particular “string” kernel.

https://arxiv.org/pdf/1802.04784.pdf

Valid Kernels

- What kernel functions $k(x_i,x_j)$ can we use?

- Kernel 'k' must be an inner product in some space:
  - There must exist a mapping from the x_i to some z_i such that $k(x_i,x_j) = z_i^T z_j$.

- It can be hard to show that a function satisfies this.
  - Infinite-dimensional eigenfunction problem.

- But like convex functions, there are some simple rules for constructing "valid" kernels from other valid kernels (bonus slide).

Kernel Trick for Other Methods

- Besides L2-regularized least squares, when can we use kernels?
  - We can compute Euclidean distance with kernels:

$$\|z_i - z_j\|^2 = z_i^T z_i - 2 z_i^T z_j + z_j^T z_j = k(x_i, x_i) - 2k(x_i, x_j) + k(x_j, x_j)$$

  - All of our distance-based methods have kernel versions:
    - Kernel k-nearest neighbours.
    - Kernel clustering k-means (allows non-convex clusters)
    - Kernel density-based clustering.
    - Kernel hierarchical clustering.
    - Kernel distance-based outlier detection.
    - Kernel "Amazon Product Recommendation".

Kernel Trick for Other Methods

- Besides L2-regularized least squares, when can we use kernels?
  - "Representer theorems" (bonus slide) have shown that any L2-regularized linear model can be kernelized:

If learning can be written in the form $\min_v \underbrace{f(Zv)}_{} + \frac{1}{2}\underbrace{\|v\|^2}_{v^Tv}$ for some 'Z'

then under weak conditions ("representer theorem") we can re-parameterize in terms of $v = Z^Tu$

giving $\min_u f(\underbrace{ZZ^T}_{K}u) + \frac{1}{2}u^T\underbrace{ZZ^T}_{K}u$

Only need 'K'

At test time you would use $\tilde{Z}\underbrace{v}_{Z^Tu} = \underbrace{\tilde{Z}Z^T}_{\tilde{K}}u = \tilde{K}u$

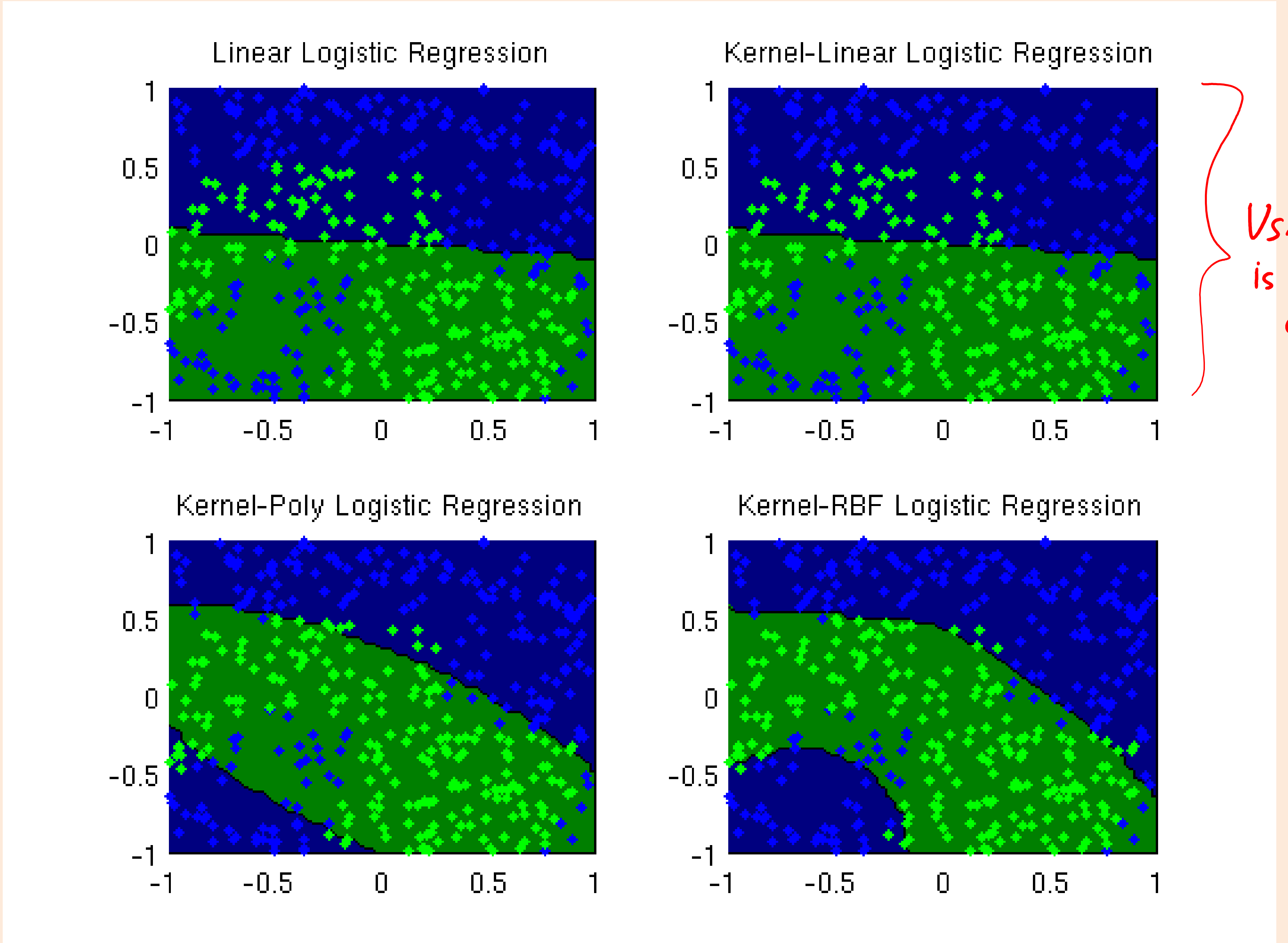
Kernel Trick for Other Methods

- Besides L2-regularized least squares, when can we use kernels?
  - "Representer theorems" (bonus slide) have shown that any L2-regularized linear model can be kernelized:
    - L2-regularized robust regression.
    - L2-regularized brittle regression.
    - L2-regularized logistic regression.
    - L2-regularized hinge loss (SVMs).

With a particular implementation, can reduce prediction cost from $O(ndt)$ to $O(mdt)$.

Number of support vectors.

Logistic Regression with Kernels



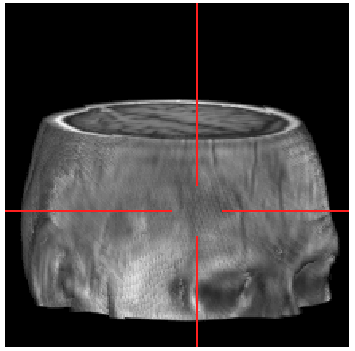
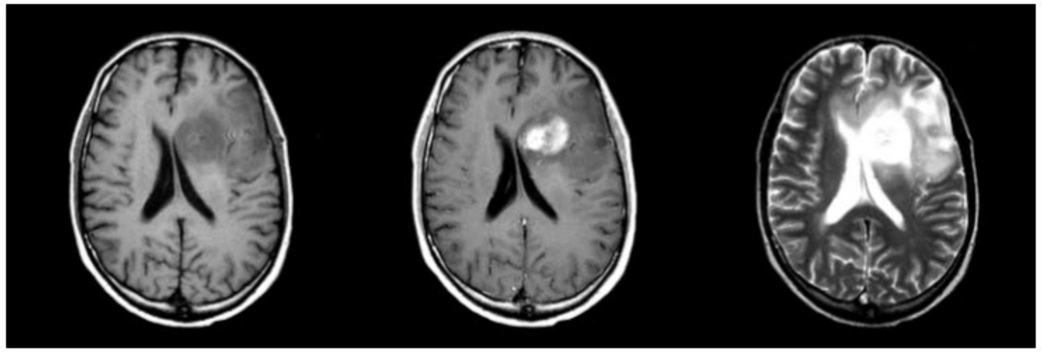
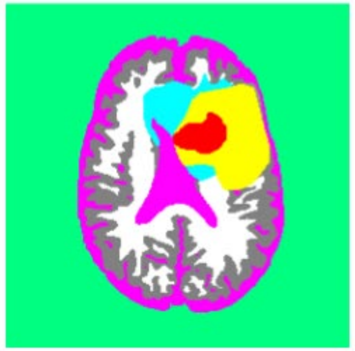
Using "linear" Kernel is the same as using original features

Coming Up Next

STOCHASTIC GRADIENT DESCENT INTRO

Motivation: Big-n Problems

- Recall the automatic brain tumour segmentation problem:

- MRI scanners at the time produced 200x200x200 volumes.
  - So one volume givens 8 million training examples.
  - And you need to train on more than one volume!

- Similar issues arise in the Gmail application:
  - If every e-mail is a training example, you have LOTS of training examples.

Motivation: Big-n Problems

- Consider fitting a least squares model:

$$f(w) = \frac{1}{2}\sum_{i=1}^{n}(w^Tx_i - y_i)^2 \propto \frac{1}{n}\sum_{i=1}^{n}(w^Tx_i - y_i)^2$$

- Gradient methods are effective when 'd' is very large.
  - O(__) per iteration instead of O(_________) to solve as linear system.

- But what if number of training examples 'n' is very large?
  - All Gmails, all products on Amazon, all homepages, all images, etc.

Gradient Descent vs. Stochastic Gradient

- Recall the gradient descent algorithm:

$$w^{t+1} = w^t - \alpha^t \nabla f(w^t)$$

- For least squares, our gradient has the form:

$$\nabla f(w) = \sum_{i=1}^{n} \underbrace{(w^T x_i - y_i)}_{\text{scalar}} \underbrace{x_i}_{d \times 1}$$

- So the cost of computing this gradient is linear in 'n'.
  - As 'n' gets large, gradient descent iterations become expensive.

Gradient Descent vs. Stochastic Gradient

- Common solution to this problem is stochastic gradient algorithm:

$$w^{t+1} = w^t - \alpha^t \nabla f_i(w^t)$$

- Uses the gradient of a randomly-chosen training example:

$$\nabla f_i(w) = \underbrace{(w^T x_i - y_i)}_{\text{scalar}}\underbrace{x_i}_{d \times 1}$$

- Cost of computing this one gradient is independent of 'n'.
  - Iterations are 'n' times faster than gradient descent iterations.
  - With 1 billion training examples, this iteration is 1 billion times faster.

Stochastic Gradient (SG)

- Stochastic gradient is an iterative optimization algorithm:
  - We start with some initial guess, w^0.
  - Generate new guess by moving in the negative gradient direction:

$$w^1 = w^0 - \alpha^0 \nabla f_i(w^0)$$

    - For a random training example 'i'.
  - Repeat to successively refine the guess:

$$w^{t+1} = w^t - \alpha^t \nabla f_i(w^t) \quad \text{for } t = 1, 2, 3, \dots$$

    - For a random training example 'i'.

Problem where we can use Stochastic Gradient

- Stochastic gradient applies when minimizing averages:

$$f(w) = \frac{1}{n}\sum_{i=1}^{n}(w^Tx_i - y_i)^2 \quad \text{(squared error)}$$

$$f(w) = \frac{1}{n}\sum_{i=1}^{n}\log(1+\exp(-y_iw^Tx_i)) \quad \text{(logistic regression)}$$

$$f(w) = \frac{1}{n}\sum_{i=1}^{n}\left[\log(1+\exp(-y_iw^Tx_i)) + \frac{\lambda}{2}\|w\|^2\right] \quad \text{(}L_2\text{-regularized logistic)}$$

$$f(w) = \frac{1}{n}\sum_{i=1}^{n}f_i(w) \quad \text{(our }\underline{\text{notation}}\text{ for the general case)}$$

- Basically, all our regression losses except "brittle" regression.
  - Recall: multiplying by positive constant doesn't change location of optimal 'w'.

Why Does Stochastic Gradient Work / Not Work?

- Main problem with stochastic gradient:
  - Gradient of random example might point in the wrong direction.

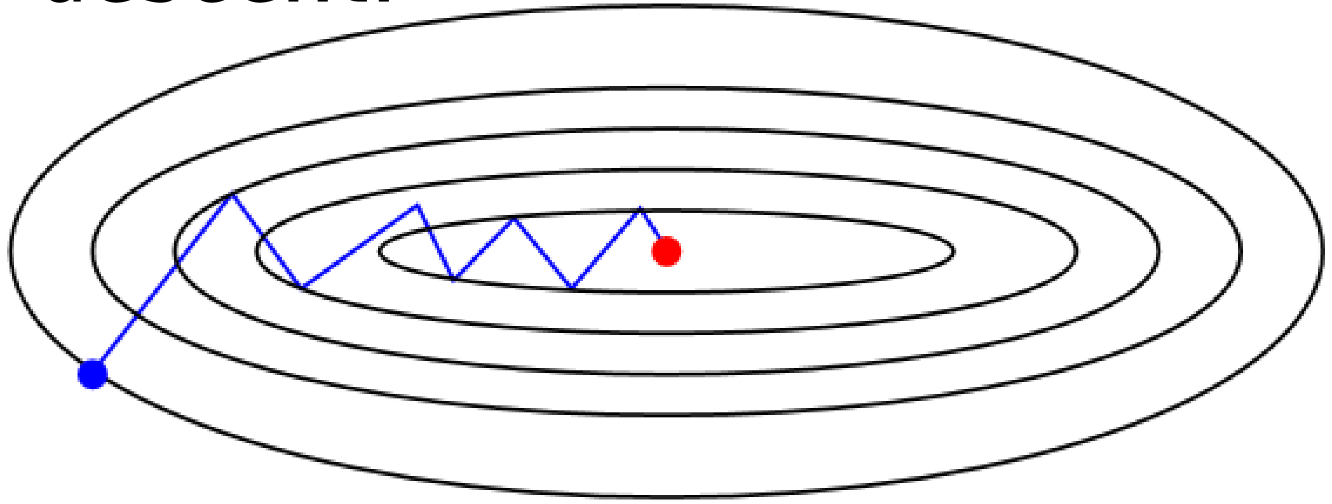
- Does this have any hope of working?
  - The expected direction is the full gradient.

$$\underbrace{E[\nabla f_i(w^k)]}_{\text{expectation over choice of random example 'i'}} = \underbrace{\sum_{i=1}^{n} p(i)\nabla f_i(w^k)}_{\text{definition of expectation}} = \sum_{i=1}^{n} \underbrace{\tfrac{1}{n}}_{\text{if each example is equally likely}} \nabla f_i(w^k) = \frac{1}{n}\sum_{i=1}^{n} \nabla f_i(w^k) = \underbrace{\nabla f(w^k)}_{\text{gradient over \underline{all} examples}}$$

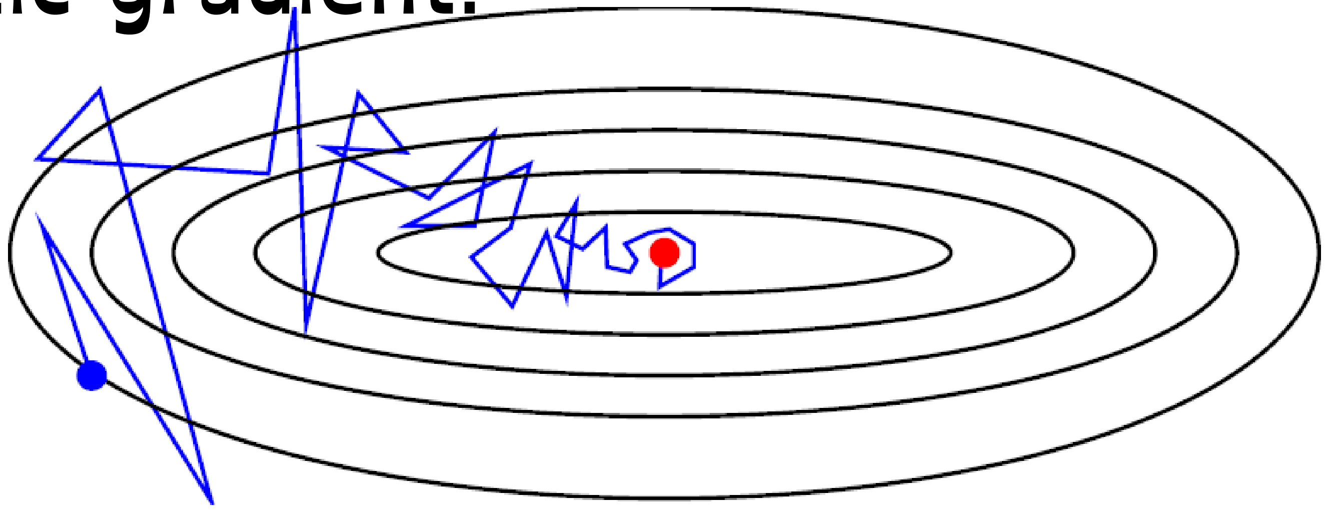
  - The algorithm is going in the right direction on average.

Gradient Descent vs. Stochastic Gradient (SG)

- Gradient descent:



- Stochastic gradient:



Summary

- High-dimensional bases allows us to separate non-separable data.
- "Other" normal equations are faster when n < d.
- Kernel trick allows us to use high-dimensional bases efficiently.
  - Write model to only depend on inner products between features vectors.

$$\hat{y} = \underbrace{\tilde{K}}_{t \times n \text{ matrix } \tilde{Z}Z^T \text{ containing inner products between test examples and training examples.}}(\underbrace{K}_{n \times n \text{ matrix } ZZ^T \text{ containing inner products between all training examples.}} + \lambda I)^{-1} y$$

- Kernels let us use similarity between objects, rather than features.
  - Allows some exponential- or infinite-sized feature sets.
  - Applies to distance-based and linear models with L2-regularization.
- Stochastic gradient methods let us use huge datasets.

- Next time:
  - How do we train on all of Gmail?

Review Questions

- Q1: What is the signature of a kernel function?
- Q2: In what scenarios would using the kernel trick be too expensive?
- Q3: How does polynomial and Gaussian RBF kernels affect the shape of decision boundaries in linear classifiers?
- Q4: Why do “deep learning” models often use stochastic gradient descent?

Feature Selection Hierarchy

- Consider a linear models with higher-order terms,

$$\hat{y}_i = w_0 + w_1 x_{i1} + w_2 x_{i2} + w_3 x_{i3} + w_{12} x_{i1} x_{i2} + w_{13} x_{i1} x_{i3} + w_{23} x_{i2} x_{i3} + w_{123} x_{i1} x_{i2} x_{i3}$$

- The number of higher-order terms may be too large.
  - Can't even compute them all.
  - We need to somehow decide which terms we'll even consider.

- Consider the following hierarchical constraint:
  - You only allow $w_{12} \neq 0$ if $w_1 \neq 0$ and $w_2 \neq 0$.
  - "Only consider feature interaction if you are using both features already."

Hierarchical Forward Selection

- Hierarchical Forward Selection:
  - Usual forward selection, but consider interaction terms obeying hierarchy.
  - Only consider $w_{12} \neq 0$ once $w_1 \neq 0$ and $w_2 \neq 0$.
  - Only allow $w_{123} \neq 0$ once $w_{12} \neq 0$ and $w_{13} \neq 0$ and $w_{23} \neq 0$.
  - Only allow $w_{1234} \neq 0$ once all threeway interactions are present.

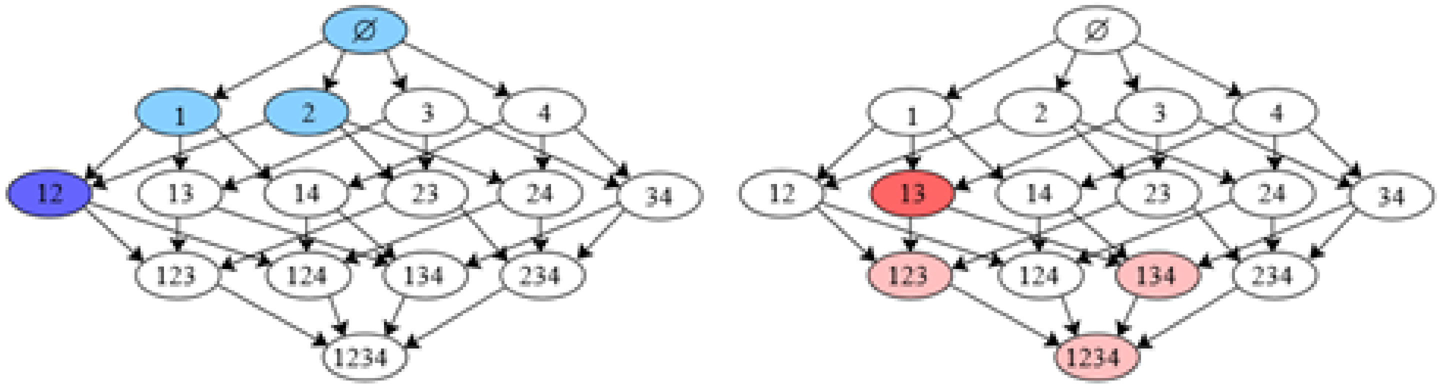


Fig 9: Power set of the set $\{1, \ldots, 4\}$: in blue, an authorized set of selected subsets. In red, an example of a group used within the norm (a subset and all of its descendants in the DAG).

http://arxiv.org/pdf/1109.2397v2.pdf

Bonus Slide: Equivalent Form of Ridge Regression

Note that $\hat{X}$ and Y are the same on the left and right side, so we only need to show that

$$(X^TX + \lambda I)^{-1}X^T = X^T(XX^T + \lambda I)^{-1}. \tag{1}$$

A version of the matrix inversion lemma (Equation 4.107 in MLAPP) is

$$(E - FH^{-1}G)^{-1}FH^{-1} = E^{-1}F(H - GE^{-1}F)^{-1}.$$

Since matrix addition is commutative and multiplying by the identity matrix does nothing, we can re-write the left side of (1) as

$$(X^TX+\lambda I)^{-1}X^T = (\lambda I+X^TX)^{-1}X^T = (\lambda I+X^TIX)^{-1}X^T = (\lambda I-X^T(-I)X)^{-1}X^T = -(\lambda I-X^T(-I)X)^{-1}X^T(-I)$$

Now apply the matrix inversion with $E = \lambda I$ (so $E^{-1} = \left(\frac{1}{\lambda}\right) I$), $F = X^T$, $H = -I$ (so $H^{-1} = -I$ too), and $G = X$:

$$-(\lambda I - X^T(-I)X)^{-1}X^T(-I) = -(\frac{1}{\lambda})IX^T(-I - X\left(\frac{1}{\lambda}\right)X^T)^{-1}.$$

Now use that $(1/\alpha)A^{-1} = (\alpha A)^{-1}$, to push the $(-1/\lambda)$ inside the sum as $-\lambda$,

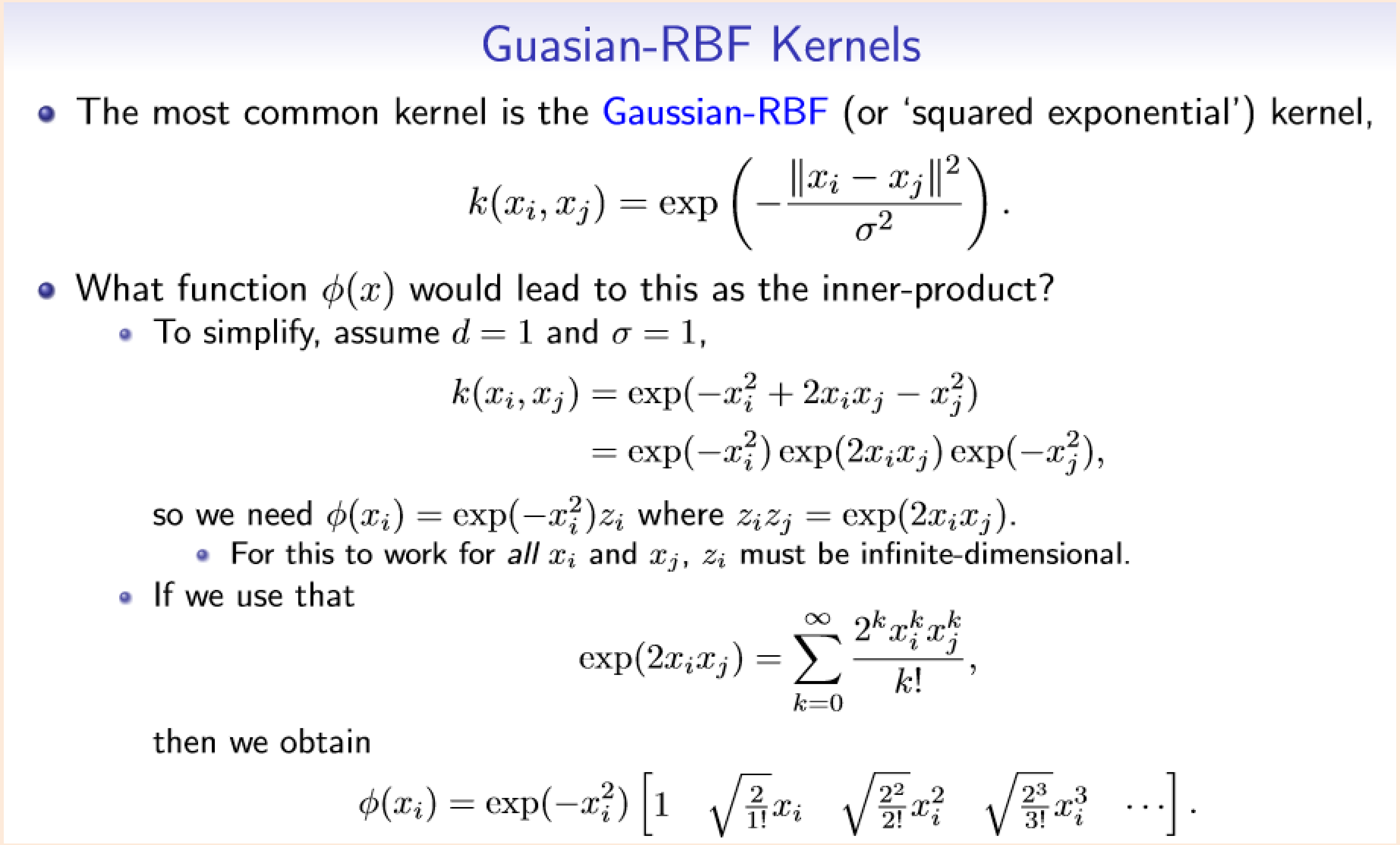
$$-(\frac{1}{\lambda})IX^T(-I - X\left(\frac{1}{\lambda}\right)X^T)^{-1} = X^T(\lambda I + XX^T)^{-1} = X^T(XX^T + \lambda I)^{-1}.$$

Why is inner product a similarity?

- It seems weird to think of the inner-product as a similarity.
- But consider this decomposition of squared Euclidean distance:

$$\frac{1}{2}\|x_i - x_j\|^2 = \frac{1}{2}\|x_i\|^2 - x_i^T x_j + \frac{1}{2}\|x_j\|^2$$

- If all training examples have the same norm, then minimizing Euclidean distance is equivalent to maximizing inner product.
  - So "high similarity" according to inner product is like "small Euclidean distance".
  - The only difference is that the inner product is biased by the norms of the training examples.
  - Some people explicitly normalize the x_i by setting $x_i = (1/\|x_i\|)x_i$, so that inner products act like the negation of Euclidean distances.
    - E.g., Amazon product recommendation.



Guasian-RBF Kernels

- The most common kernel is the Gaussian-RBF (or ‘squared exponential’) kernel,

$$k(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{\sigma^2}\right).$$

- What function $\phi(x)$ would lead to this as the inner-product?
  - To simplify, assume $d = 1$ and $\sigma = 1$,

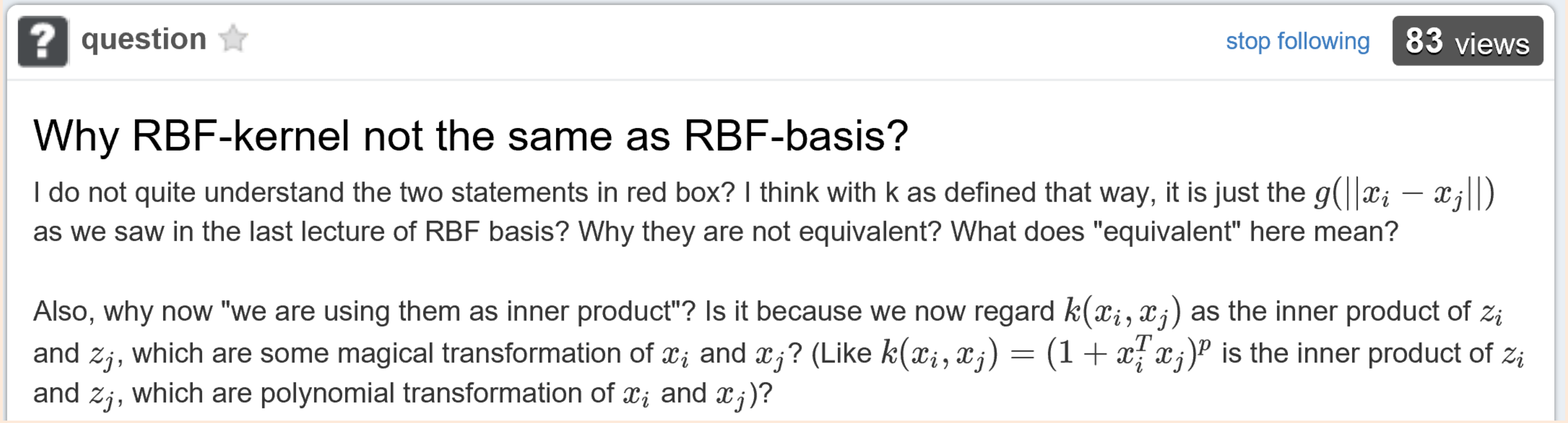
$$\begin{aligned} k(x_i, x_j) &= \exp(-x_i^2 + 2x_ix_j - x_j^2) \\ &= \exp(-x_i^2)\exp(2x_ix_j)\exp(-x_j^2), \end{aligned}$$

    so we need $\phi(x_i) = \exp(-x_i^2)z_i$ where $z_iz_j = \exp(2x_ix_j)$.
    - For this to work for *all* x_i and x_j, z_i must be infinite-dimensional.
  - If we use that

$$\exp(2x_ix_j) = \sum_{k=0}^{\infty} \frac{2^k x_i^k x_j^k}{k!},$$

    then we obtain

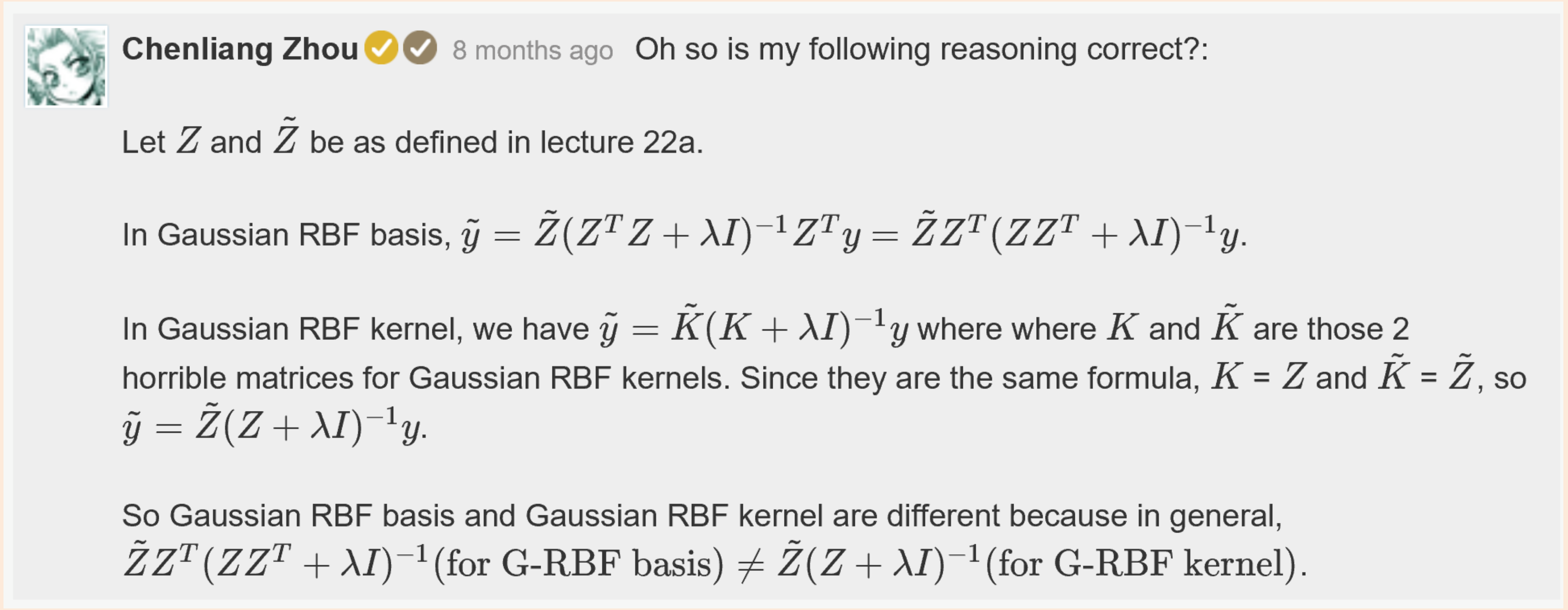
$$\phi(x_i) = \exp(-x_i^2)\left[1 \quad \sqrt{\tfrac{2}{1!}}x_i \quad \sqrt{\tfrac{2^2}{2!}}x_i^2 \quad \sqrt{\tfrac{2^3}{3!}}x_i^3 \quad \cdots\right].$$



Why RBF-kernel not the same as RBF-basis?

I do not quite understand the two statements in red box? I think with k as defined that way, it is just the $g(||x_i - x_j||)$ as we saw in the last lecture of RBF basis? Why they are not equivalent? What does "equivalent" here mean?

Also, why now "we are using them as inner product"? Is it because we now regard $k(x_i, x_j)$ as the inner product of z_i and z_j, which are some magical transformation of x_i and x_j? (Like $k(x_i, x_j) = (1 + x_i^T x_j)^p$ is the inner product of z_i and z_j, which are polynomial transformation of x_i and x_j)?



Oh so is my following reasoning correct?:

Let Z and $\tilde{Z}$ be as defined in lecture 22a.

In Gaussian RBF basis, $\tilde{y} = \tilde{Z}(Z^TZ + \lambda I)^{-1}Z^Ty = \tilde{Z}Z^T(ZZ^T + \lambda I)^{-1}y$.

In Gaussian RBF kernel, we have $\tilde{y} = \tilde{K}(K + \lambda I)^{-1}y$ where where K and $\tilde{K}$ are those 2 horrible matrices for Gaussian RBF kernels. Since they are the same formula, K = Z and $\tilde{K}$ = $\tilde{Z}$, so $\tilde{y} = \tilde{Z}(Z + \lambda I)^{-1}y$.

So Gaussian RBF basis and Gaussian RBF kernel are different because in general, $\tilde{Z}Z^T(ZZ^T + \lambda I)^{-1}(\text{for G-RBF basis}) \neq \tilde{Z}(Z + \lambda I)^{-1}(\text{for G-RBF kernel})$.

A String Kernel

- A classic “string kernel”:
  - We want to compute k(“cat”, “cart”).
  - Find all common subsequences: ‘c’, ‘a’, ‘t’, ‘ca’, ‘at’, ‘ct’, ‘cat’.
  - Weight them by total length in original strings:
    - ‘c’ has length (1,1), ‘ca’ has lengths (2,2), ‘ct’ has lengths (3,4), and so on.
  - Add up the weighted lengths of common subsequences to get a similarity:

$$\text{k}(\text{“cat”}, \text{“cart”}) = \underbrace{\gamma^1\gamma^1}_{\text{‘c’}} + \underbrace{\gamma^1\gamma^1}_{\text{‘a’}} + \underbrace{\gamma^1\gamma^1}_{\text{‘t’}} + \underbrace{\gamma^2\gamma^2}_{\text{‘ca’}} + \underbrace{\gamma^2\gamma^3}_{\text{‘at’}} + \underbrace{\gamma^3\gamma^4}_{\text{‘ct’}} + \underbrace{\gamma^3\gamma^4}_{\text{‘cat’}},$$

  - where γ is a hyper-parameter controlling influence of length.

- Corresponds to exponential feature set (counts/lengths of all subsequences).
  - But kernel can be computed in polynomial time by dynamic programming.
- Many variations exist.

Constructing Valid Kernels

- If $k_1(x_i, x_j)$ and $k_2(x_i, x_j)$ are valid kernels, then the following are valid kernels:
  - $k_1(\phi(x_i), \phi(x_j))$.
  - $\alpha k_1(x_i, x_j) + \beta k_2(x_i, x_j)$ for $\alpha \geq 0$ and $\beta \geq 0$.
  - $k_1(x_i, x_j) k_2(x_i, x_j)$.
  - $\phi(x_i) k_1(x_i, x_j) \phi(x_j)$.
  - $\exp(k_1(x_i, x_j))$.
- Example: Gaussian-RBF kernel:

$$k(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{\sigma^2}\right)$$

$$= \underbrace{\exp\left(-\frac{\|x_i\|^2}{\sigma^2}\right)}_{\phi(x_i)} \underbrace{\exp\left(\underbrace{\frac{2}{\sigma^2}}_{\alpha \geq 0} \underbrace{x_i^T x_j}_{\text{valid}}\right)}_{\exp(\text{valid})} \underbrace{\exp\left(-\frac{\|x_j\|^2}{\sigma^2}\right)}_{\phi(x_j)}.$$

Representer Theorem

- Consider linear model differentiable with losses f_i and L2-regularization,

$$\operatorname*{argmin}_{w\in\mathbb{R}^d} \sum_{i=1}^{n} f_i(w^T x_i) + \frac{\lambda}{2}\|w\|^2.$$

- Setting the gradient equal to zero we get

$$0 = \sum_{i=1}^{n} f_i'(w^T x_i)x_i + \lambda w.$$

- So any solution w^* can written as a linear combination of features x_i,

$$\begin{aligned} w^* &= -\frac{1}{\lambda}\sum_{i=1}^{n} f_i'((w^*)^T x_i)x_i = \sum_{i=1}^{n} z_i x_i \\ &= X^T z. \end{aligned}$$

- This is called a representer theorem (true under much more general conditions).

Kernel Trick for Other Methods

- Besides L2-regularized least squares, when can we use kernels?
  - "Representer theorems" have shown that
    any L2-regularized linear model can be kernelized.
  - Linear models without regularization fit with gradient descent.
    - If you starting at v=0 or with any other value in span of rows of 'Z'.

Iterations of <u>gradient descent</u> on $f(Zv)$ can be written as $v = Z^T u$

which lets us re-parameterize as $f(ZZ^T u)$

At <u>test</u> time you would use $\tilde{Z}v = \underbrace{\tilde{Z}Z^T}_{\tilde{K}} u = \tilde{K}u$ $\quad \underbrace{v}_{X^T u}$