CPSC 340:
Machine Learning and Data Mining

Kernel Trick

Last Time: Feature Engineering

- We discussed feature engineering:
  - Designing a set of features to achieve good performance on a problem.
- We discussed various issues:
  - Feature aggregation/discretization to address coupon counting.
  - Feature scaling to address features of different scales.
  - Non-linear transforms to make relationships more linear.
- We started discussing feature engineering on text data:
  - Universal representation, Bag of words, n-gram (capture local contex), Part-of-speech features
  - Personalized features

Part 3 Key Ideas: Linear Models, Least Squares

- Focus of Part 3 is linear models:
  - Supervised learning where prediction is linear combination of features:

$$\hat{y}_i = w_1 x_{i1} + w_2 x_{i2} + \cdots + w_d x_{id}$$
$$= w^T x_i$$

- Regression:
  - Target y_i is numerical, testing ($\hat{y}_i == y_i$) doesn't make sense.

- Squared error: $\frac{1}{2}\sum_{i=1}^{n}(w^T x_i - y_i)^2$ or $\frac{1}{2}\|Xw - y\|^2$
  - Can find optimal 'w' by solving "normal equations".

Part 3 Key Ideas: Change of Basis, Gradient Descent

- Change of basis: replaces features x_i with non-linear transforms z_i:
  - Add a bias variable (feature that is always one).
  - Polynomial basis.
  - Other basis functions (logarithms, trigonometric functions, etc.).

- For large 'd' we often use gradient descent:
  - Iterations only cost O(nd).
  - Converges to a critical point of a smooth function.
  - For convex functions, it finds a global optimum.

Part 3 Key Ideas: Error Functions, Smoothing

- Error functions:
  - Squared error is sensitive to outliers.
  - Absolute (L_1) error and Huber error are more robust to outliers.
  - Brittle (L_∞) error is more sensitive to outliers.
- L_1 and L_∞ error functions are convex but non-differentiable:
  - Finding 'w' minimizing these errors is harder than squared error.
- We can approximate these with differentiable functions:
  - L_1 can be approximated with Huber.
  - L_∞ can be approximated with log-sum-exp.
- With these smooth (convex) approximations,
  we can find global optimum with gradient descent.

Part 3 Key Ideas: Regularization

- L0-regularization (AIC, BIC):
  - Adds penalty on the number of non-zeros to select features (non-convex, hard to optimize).

$$f(w) = \|Xw - y\|^2 + \lambda \|w\|_0$$

- L2-regularization (ridge regression):
  - Adding penalty on the L2-norm of 'w' to decrease overfitting (unique solution).

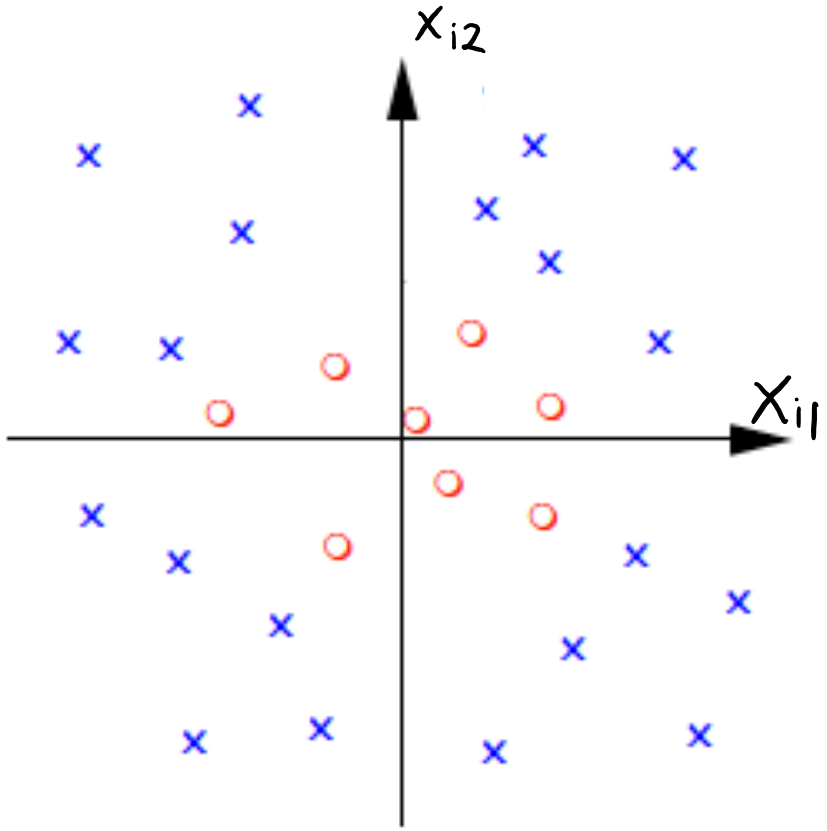
$$f(w) = \|Xw - y\|^2 + \frac{\lambda}{2}\|w\|^2$$

- L1-regularization (LASSO):
  - Adding penalty on the L1-norm decreases overfitting and selects features.

$$f(w) = \|Xw - y\|^2 + \lambda \|w\|_1$$

Support Vector Machines for Non-Separable

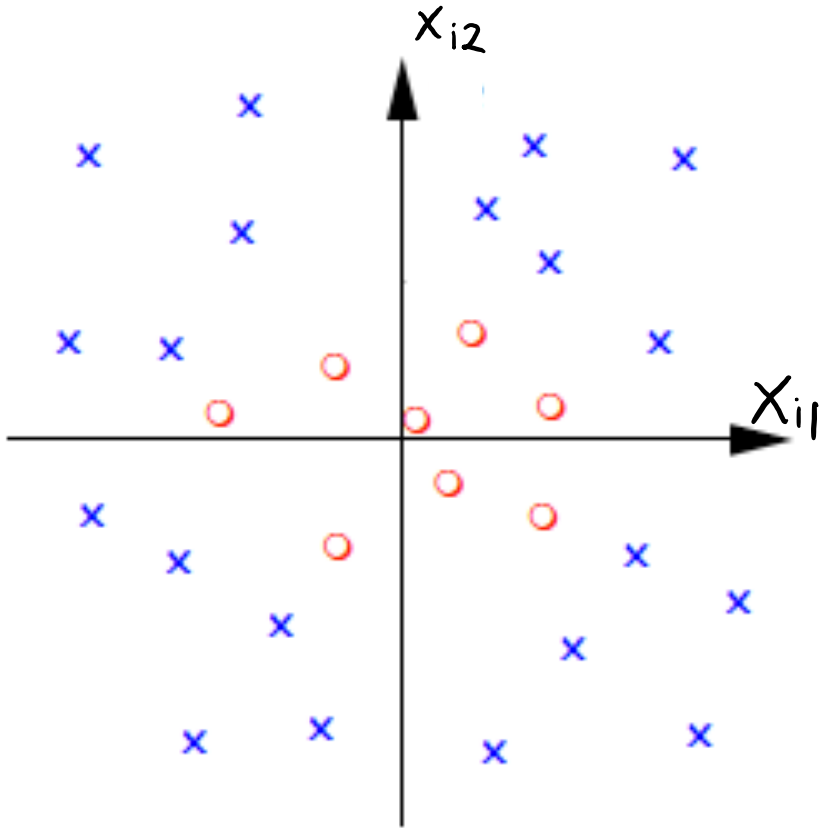
- Can we use linear models for data that is not close to separable?

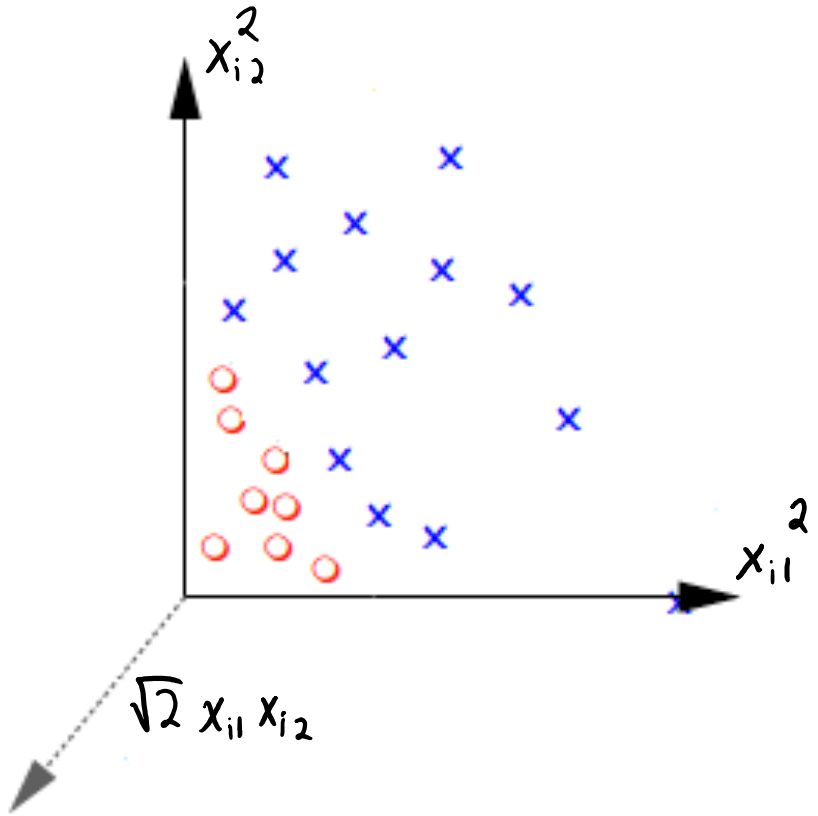


http://math.stackexchange.com/questions/353607/how-do-inner-product-space-determine-half-planes

Support Vector Machines for Non-Separable

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

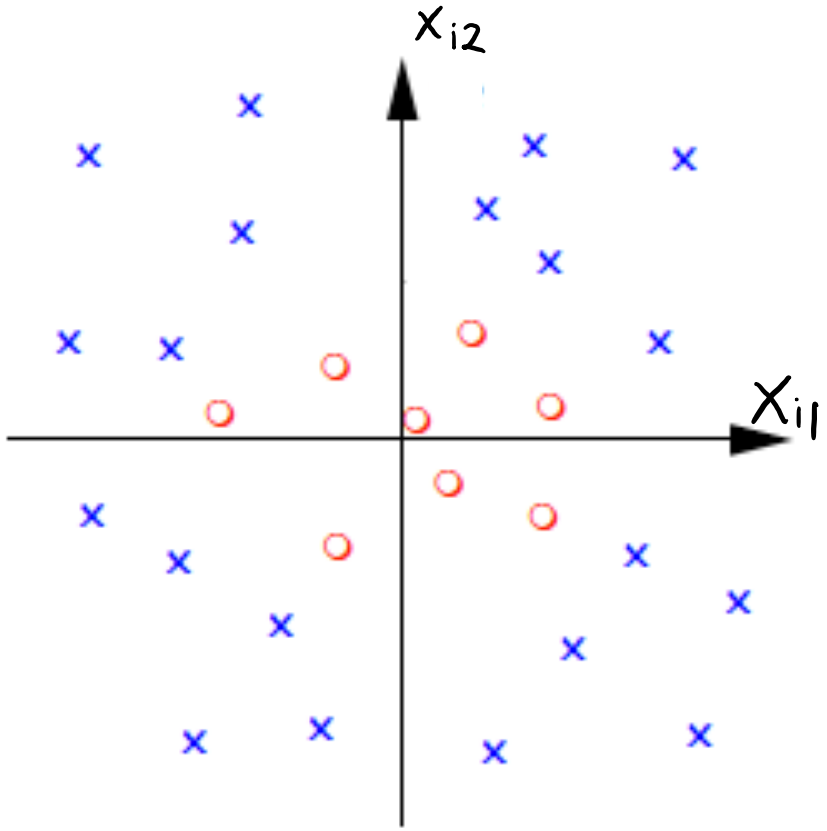


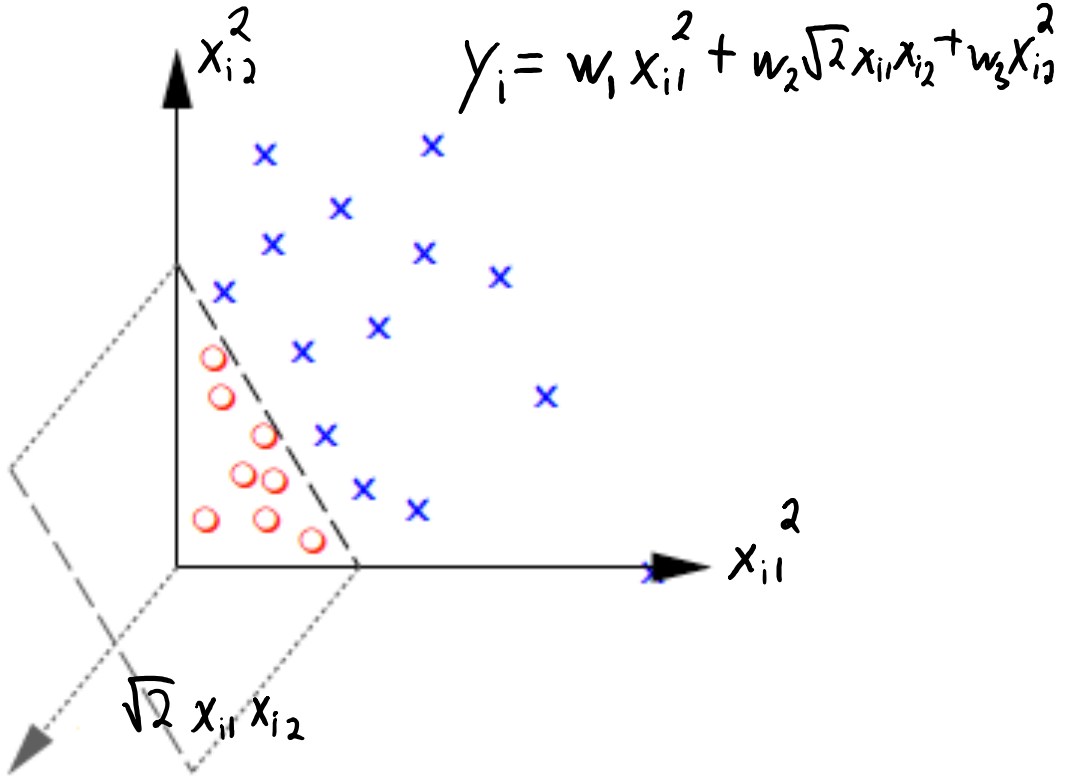


http://math.stackexchange.com/questions/353607/how-do-inner-product-space-determine-half-planes

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).

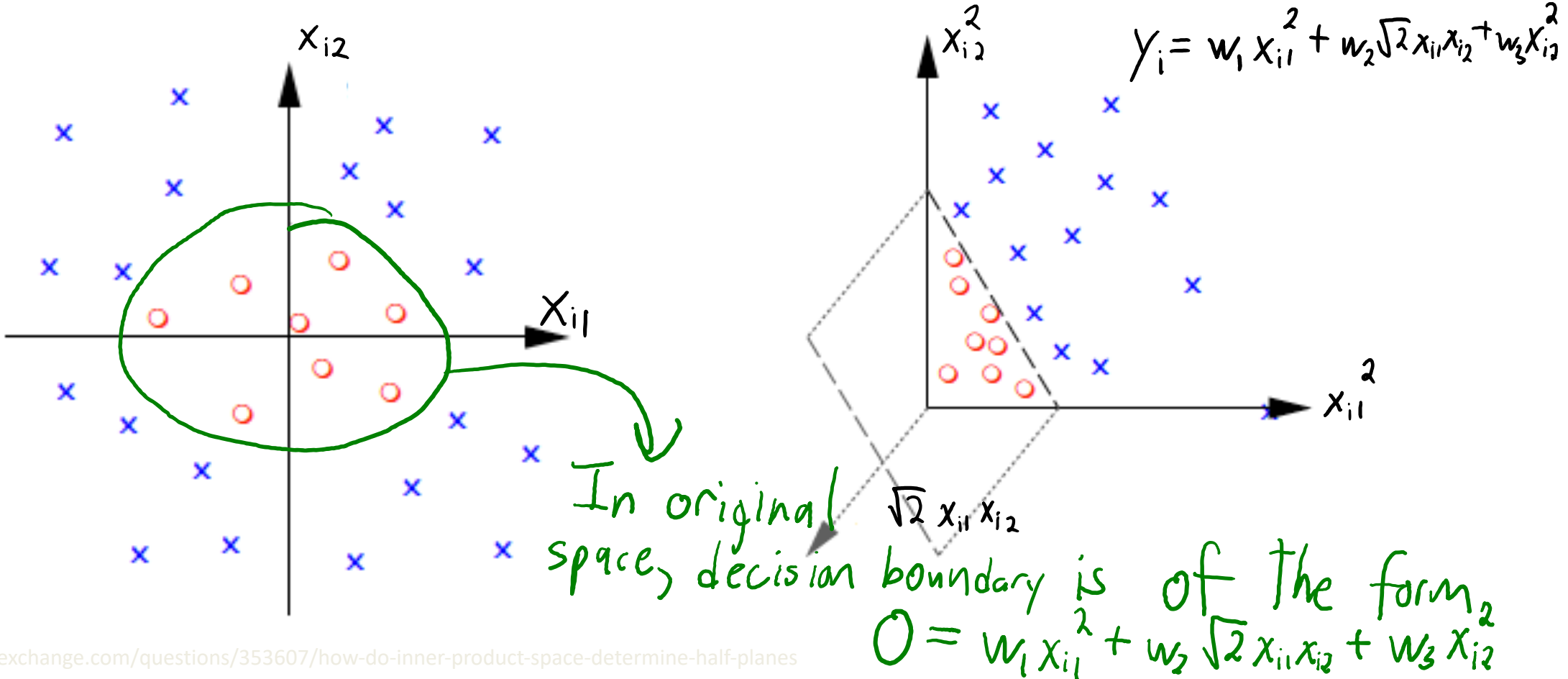




http://math.stackexchange.com/questions/353607/how-do-inner-product-space-determine-half-planes

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).



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:

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

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

Multi-Dimensional Polynomial Basis

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

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

bias x_{i1} x_{i2} $(x_{i1})^2$ $(x_{i2})^2$ $(x_{i1})(x_{i2})$

- 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 $O(d^5)$ quintic terms:

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

- 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.

How can you use an exponential-sized basis?

- Which of these two expressions would you rather compute?

$x^9 + 9x^8 + 36x^7 + 84x^6 + 126x^5 + 126x^4 + 84x^3 + 36x^2 + 9x + 1$ or $(x+1)^9$

  - Expressions are equal, but left way costs O(p) while right costs O(1).

- Which of these two expressions would you rather compute?

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

  - Expressions are equal, but left way has infinite terms and right costs O(1).

- Can we add weights to the terms in sum, and use these tricks?

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

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

($k \times k$)

(in practice you still solve the linear system, since inverse is less numerically unstable – see CPSC 302)

- With some work (bonus slide), this can equivalently be written as:

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

($n \times n$)

- 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:

$$\hat{y} = \tilde{Z} v$$

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

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

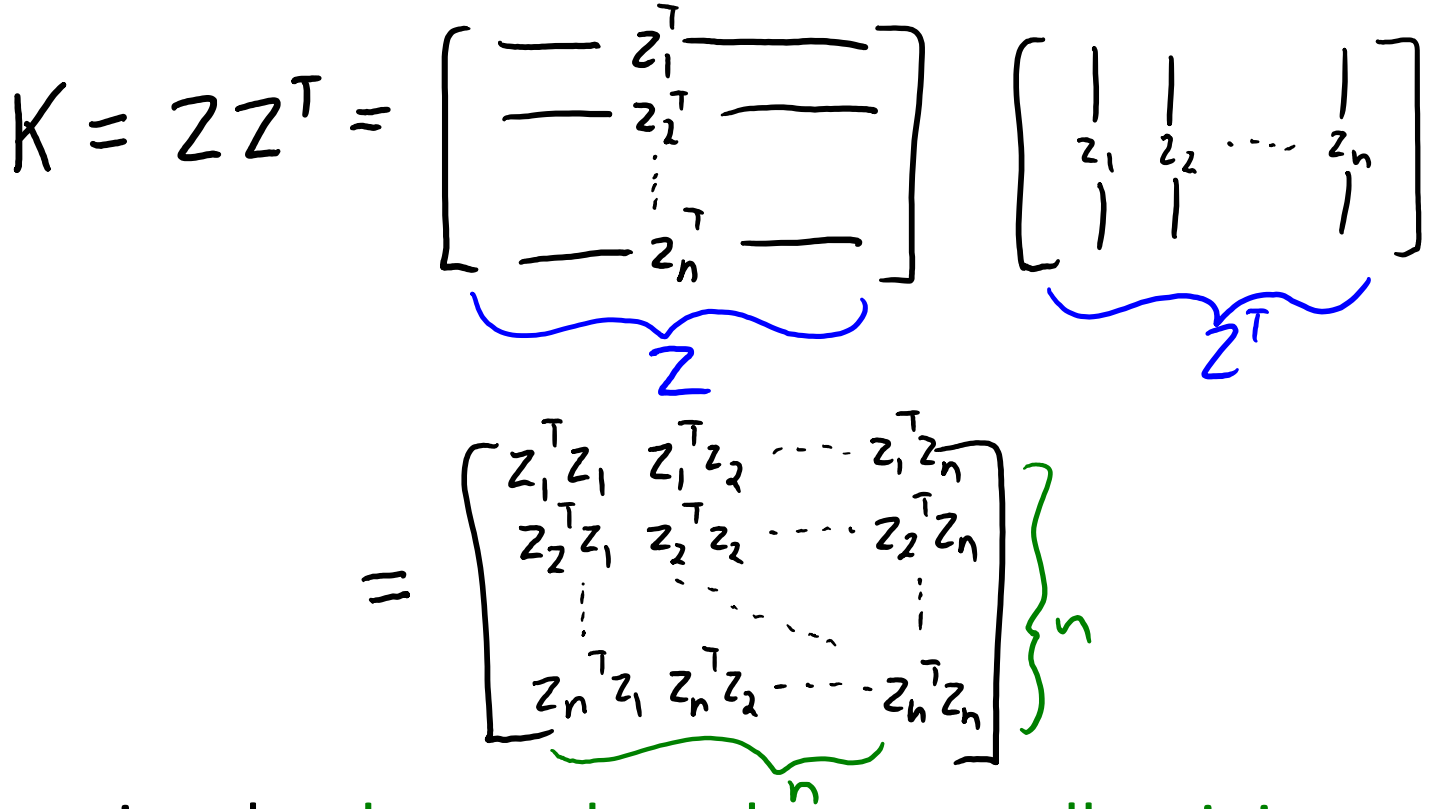
$t \times 1 = t \times n \quad n \times n \quad n \times 1$

$n \times 1$ vector of "kernel weights" that we learn

- Notice that if we can from K and $\tilde{K}$ then we do not need Z and $\tilde{Z}$.
- Key idea behind "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 compute $(x+1)^9$ instead of $x^9 + 9x^8 + 36x^7 + 84x^6$…

Gram Matrix

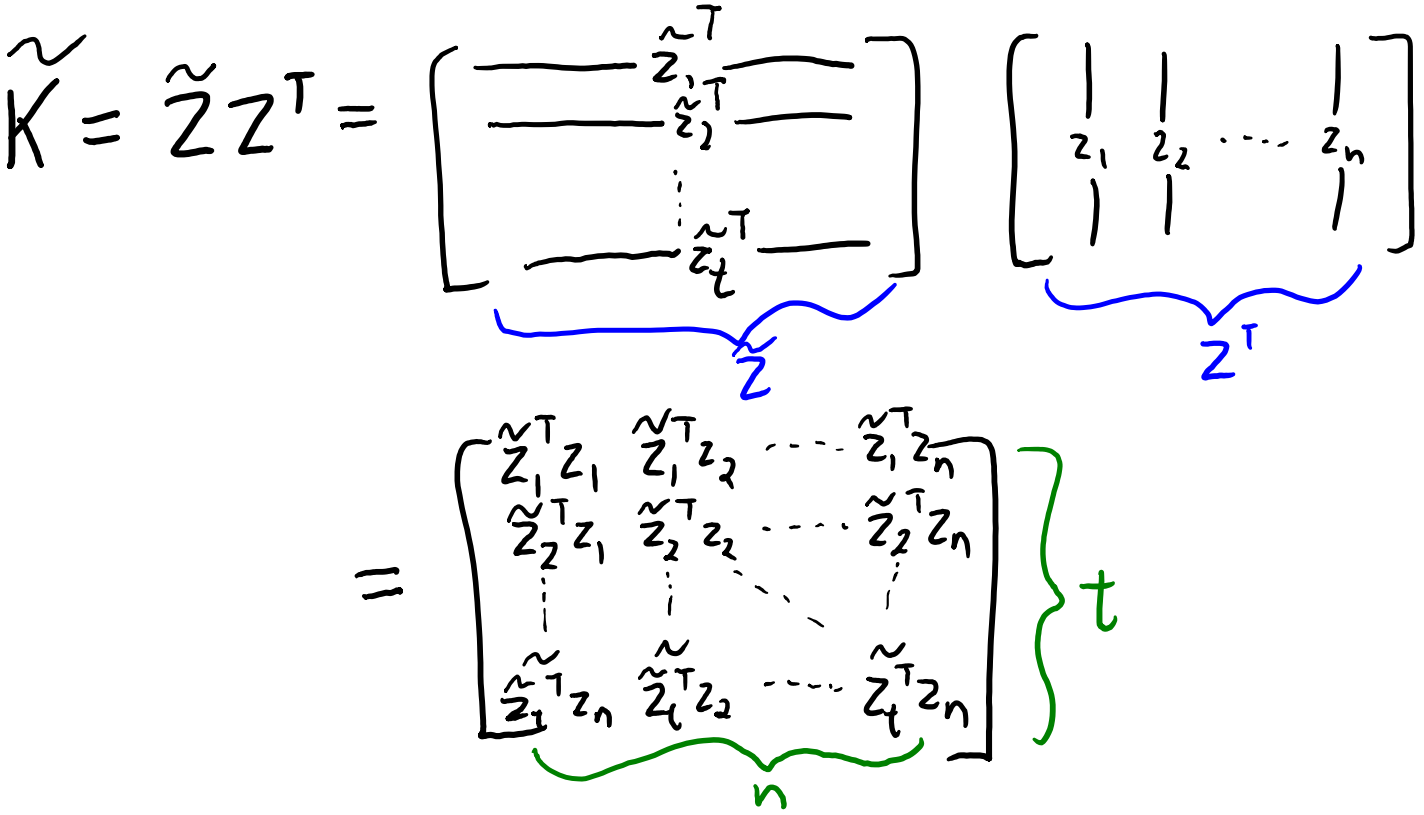
- The matrix $K = ZZ^T$ is called the Gram matrix K.



- 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

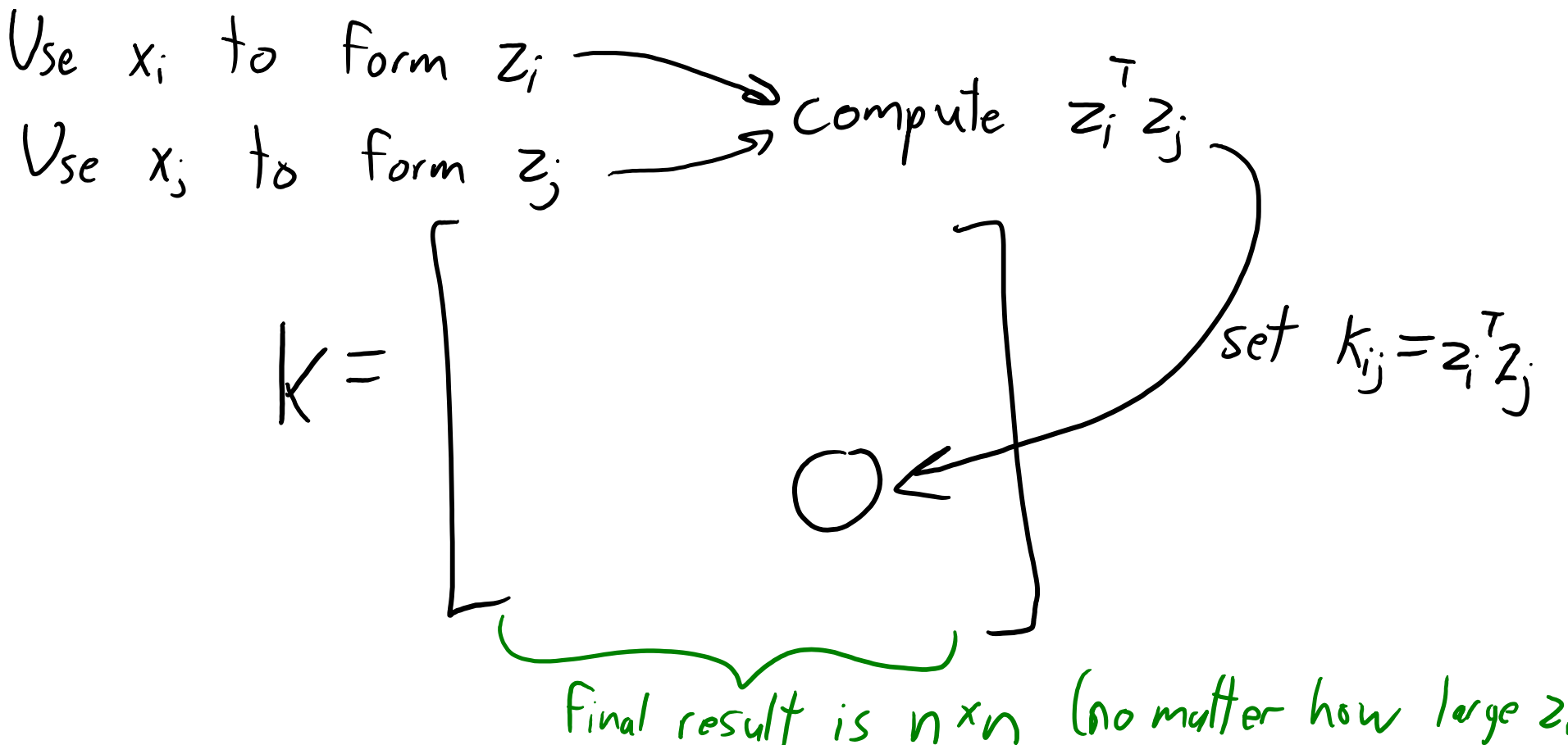
- The matrix $\tilde{K} = \tilde{Z}Z^T$ has dot products between train and test examples:



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

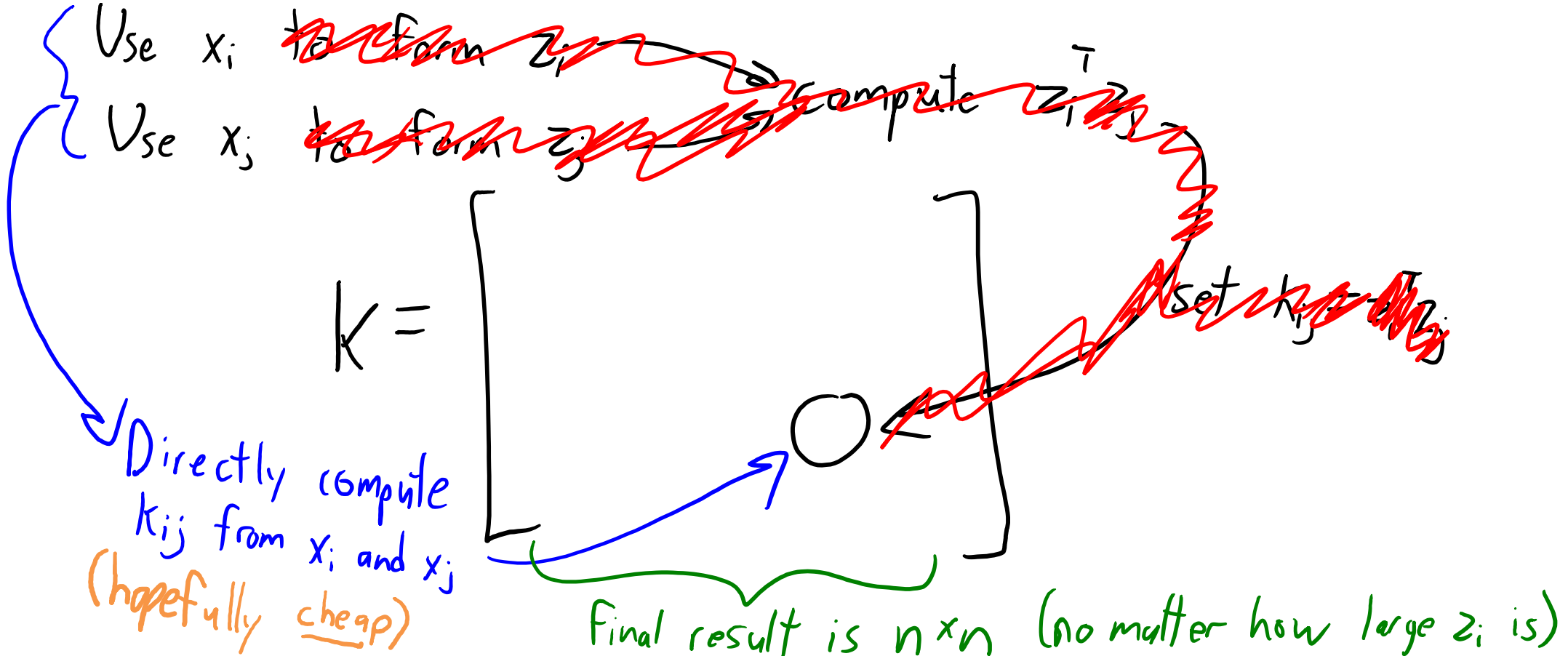
The Kernel Trick

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



The Kernel Trick

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



Linear Regression vs. Kernel Regression

Linear Regression

Training

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

Testing

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

Both methods make the same predictions.

Kernel Regression

Training:

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

Non-parametric

Testing:

1. Form inner products $\tilde{K}$ from X and $\tilde{X}$
2. Compute $\hat{y} = \underbrace{\tilde{K}}_{t\times n} \underbrace{u}_{n\times 1}$

If you want explicit feature weights 'v' from kernel regression, you can use $v = Z^T u$

Degenerate Example: "Linear Kernel"

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

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

- As an example kernel, the "linear kernel" just uses original features:

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

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

$$z_i^T z_j = x_i^T x_j$$

(z_i: x_i, z_j: x_j)

  - But in this case model is still a linear function of original features.

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 $k(x_i, x_j) = (x_i^T x_j)^2$:

$$\begin{aligned} z_i^T z_j &= x_{i1}^2 x_{j1}^2 + (\sqrt{2}\, x_{i1} x_{i2})(\sqrt{2}\, x_{j1} x_{j2}) + x_{i2}^2 x_{j2}^2 \\ &= x_{i1}^2 x_{j1}^2 + 2 x_{i1} x_{i2} x_{j1} x_{j2} + x_{i2}^2 x_{j2}^2 \\ &= (x_{i1} x_{j1} + x_{i2} x_{j2})^2 \qquad \text{"completing the square"} \\ &\qquad (x_{i1} x_{j1} + x_{i2} x_{j2}) = x_i^T x_j \\ &= (x_i^T x_j)^2 \leftarrow \text{No need for } z_i \text{ to compute } z_i^T z_j \end{aligned}$$

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 $k(x_i,x_j) = (1 + x_i^T x_j)^2$:

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

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

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

$$= z_i^T z_j$$

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 $O(d)$ instead of $O(d^p)$ 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:

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

test example; train example

  - Make predictions using:

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

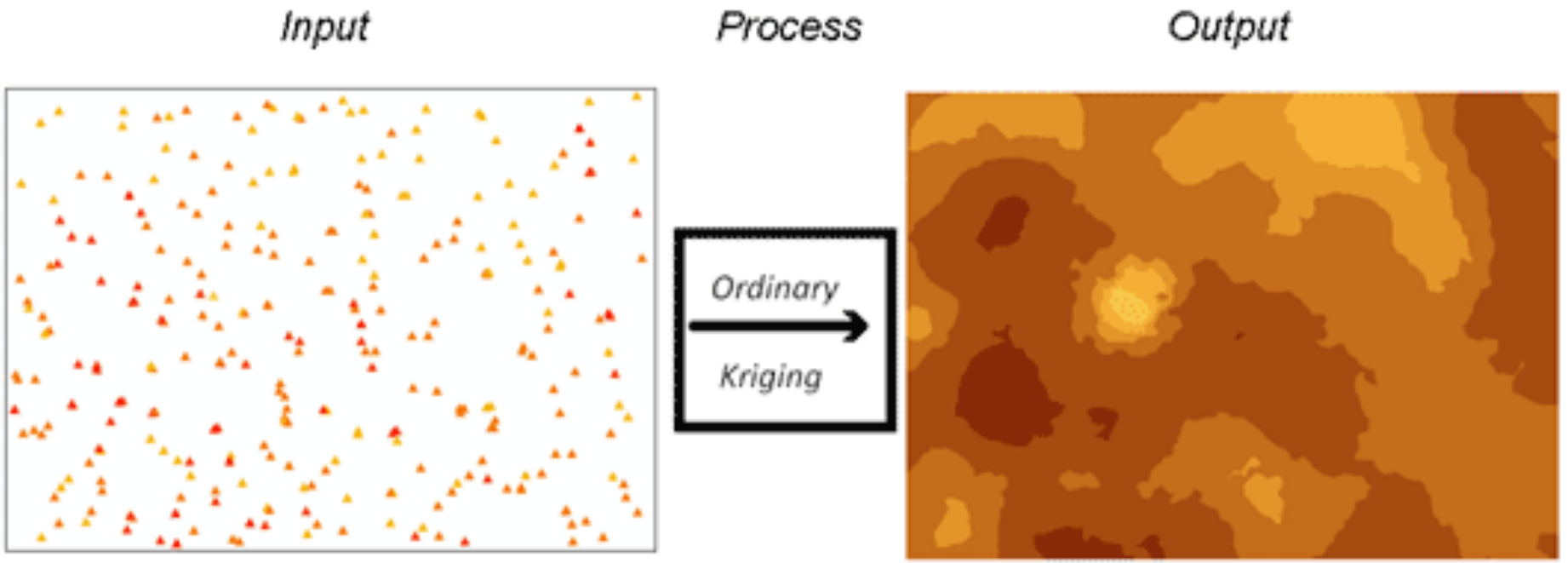
$\hat{y}$: $t \times 1$; $\tilde{K}$: $t \times n$; $(K + \lambda I)^{-1}$: $n \times n$; y: $n \times 1$

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

- Training cost is only $O(n^2 d + n^3)$, despite using $k = O(d^p)$ features.
  - We can form 'K' in $O(n^2 d)$, and we need to "invert" an 'n x n' matrix.
  - Testing cost is only O(ndt), cost to form $\tilde{K}$.

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.

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”.

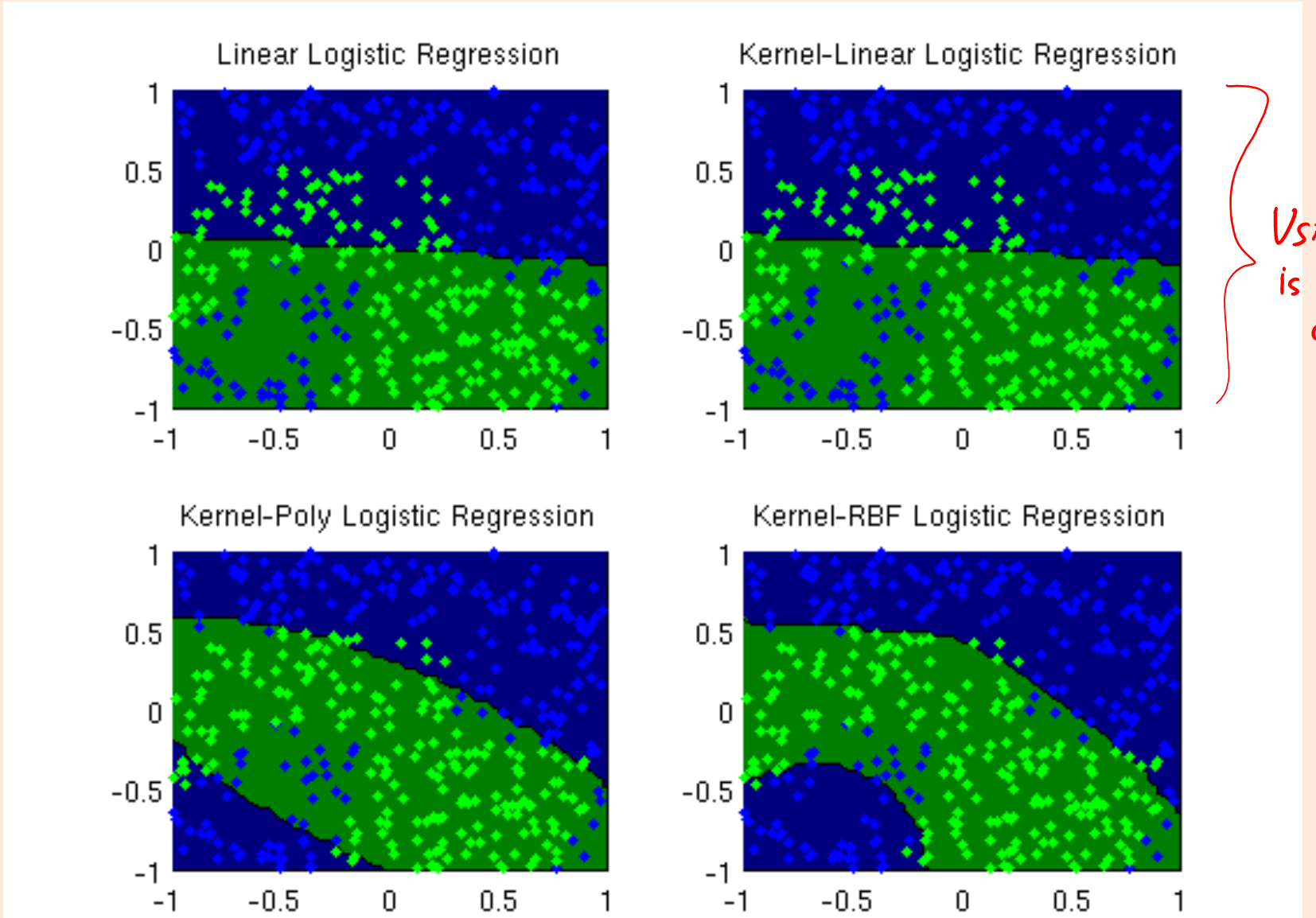
Kernel Trick for Other Methods

- 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 (see bonus):
    - Kernel robust regression with L2-regularization.
    - Kernel brittle regression with L2-regularization.
    - Kernel hinge loss (SVM) or logistic loss with L2-regularization.
    - Kernel multi-class SVM or multi-class logistic with L2-regularization.

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

m: Number of support vectors.

Logistic Regression with Kernels



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

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.
- Kernels let us use similarity between objects, rather than features.
  - Allows some exponential-sized feature sets.

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

An Infinite-Dimensional Basis?

- Suppose d=1 and I want to use this infinite set of features (d = ∞):

$$z_i = \exp\left(-\tfrac{1}{2}x_i^2\right)\left[1 \quad \sqrt{\tfrac{1}{1!}}x_i \quad \sqrt{\tfrac{1}{2!}}x_i^2 \quad \sqrt{\tfrac{1}{3!}}x_i^3 \quad \sqrt{\tfrac{1}{4!}}x_i^4 \quad \sqrt{\tfrac{1}{5!}}x_i^5 \quad \cdots\right]$$

- The kernel function has a simple form:

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

$$= \exp\left(-\tfrac{1}{2}x_i^2\right)\exp\left(-\tfrac{1}{2}x_j^2\right)\left(1 + \tfrac{1}{1!}x_i x_j + \tfrac{1}{2!}x_i^2 x_j^2 + \tfrac{1}{3!}x_i^3 x_j^3 + \tfrac{1}{4!}x_i^4 x_j^4 + \tfrac{1}{5!}x_i^5 x_j^5 + \cdots\right)$$

$$= \exp(x_i x_j)$$

$$= \exp\left(-\tfrac{1}{2}x_i^2 - \tfrac{1}{2}x_j^2 + x_i x_j\right)$$

$$= \exp\left(-\tfrac{1}{2}(x_i - x_j)^2\right)$$

- For these features, even though d=∞, cost of kernel is O(1).

Gaussian-RBF Kernel

- Previous slide is a special case of the Gaussian RBF kernel:

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

  - Where we have introduced a variance hyper-parameter σ^2.
  - This is the most popular kernel function.

- Same formula as Gaussian RBF features, but not equivalent:
  - Before we used Gaussian RBFs as a set of 'n' features.
  - Now we are using Gaussian RBFs as a dot product (for infinite features).
    - In practice, Gaussian RBFs as features or as kernels gives similar performance.

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.

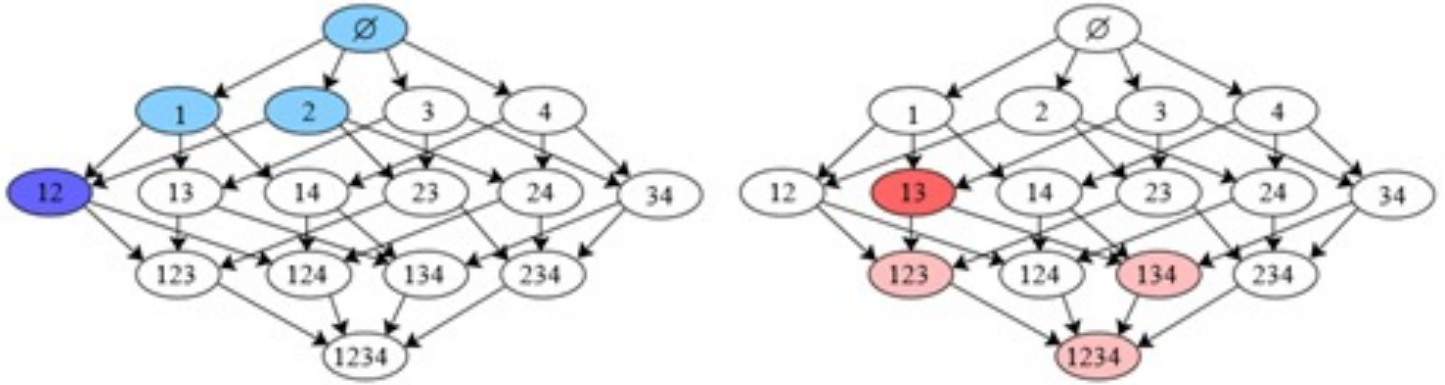


Fig 9: Power set of the set $\{1, \ldots, 4\}$: in blue, an authorized set of selected subsets. In red, an example of a group used within the norm (a subset and all of its descendants in the DAG).

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

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

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

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

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 $G = X$:

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

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

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

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.

question

stop following 83 views

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)?

Chenliang Zhou 8 months ago Oh so is my following reasoning correct?:

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

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

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,
$\tilde{Z} Z^T (Z Z^T + \lambda I)^{-1} (\text{for G-RBF basis}) \neq \tilde{Z}(Z + \lambda I)^{-1} (\text{for G-RBF kernel})$.

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:

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

  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,

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

- 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

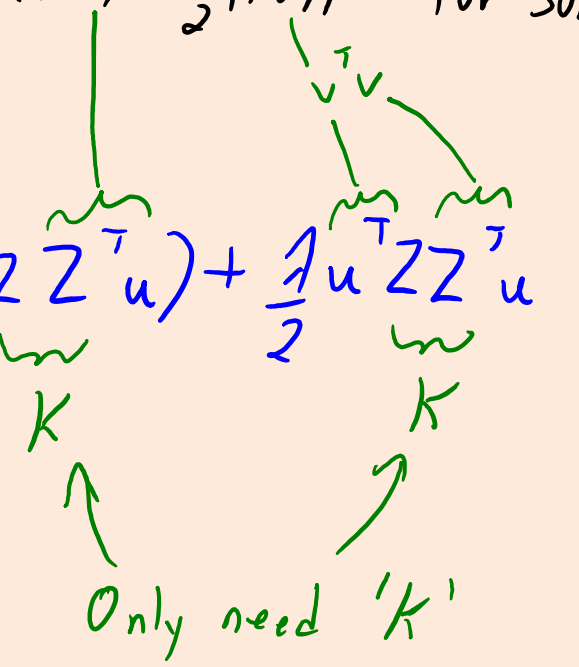
- Besides L2-regularized least squares, when can we use kernels?
  - "Representer theorems have shown that any L2-regularized linear model can be kernelized:

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

then under weak conditions ("representer theorem")

we can re-parameterize in terms of $v = Z^T u$

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



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

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 gradient descent on $f(Zv)$ can be written as $v = Z^T u$
which lets us re-parameterize as $f(ZZ^T u)$

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

($X^T u$ under v; $\tilde{K}$ under $\tilde{Z}Z^T$)