CPSC 340: Machine Learning and Data Mining

Kernel Trick

Fall 2018

Admin

- Assignment 4:
  - Due Friday of next week.

- Midterm:
  - Grades posted.
  - Can view exam during Mike or my office hours this week and next week.

- 532M Projects:
  - “No news is good news”.

Last Time: Other Normal Equations and Kernel Trick

- We discussed the "other" normal equations (under basis 'Z'):

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

  - Faster if n < d.
- Predictions only depend on features through inner-product matrices 'K' and $\tilde{K}$.
  - So everything we need to know about z_i is summarized by the $z_i^T z_j$.

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

$$= \tilde{K} u$$

we have 'n' parameters, independent of the size of 'Z'

- Kernel trick:
  - If you have a kernel function $k(x_i,x_j)$ that computes $z_i^T z_j$,
    then you don't ever need to compute the basis z_i explicitly to use the model.

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:

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

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

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

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

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

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

$$\begin{aligned}
(1 + x_i^T x_j)^2 &= 1 + 2x_i^T x_j + (x_i^T x_j)^2 \\
&= 1 + 2x_{i1}x_{j1} + 2x_{i2}x_{j2} + x_{i1}^2 x_{j1}^2 + 2x_{i1}x_{i2}x_{j1}x_{j2} + x_{i2}^2 x_{j2}^2 \\
&= \underbrace{\begin{bmatrix} 1 & \sqrt{2}x_{i1} & \sqrt{2}x_{i2} & x_{i1}^2 & \sqrt{2}x_{i1}x_{i2} & x_{i2}^2 \end{bmatrix}}_{z_i^T} \underbrace{\begin{bmatrix} 1 \\ \sqrt{2}x_{j1} \\ \sqrt{2}x_{j2} \\ x_{j2}^2 \\ \sqrt{2}x_{j1}x_{j2} \\ x_{j2}^2 \end{bmatrix}}_{z_j} \\
&= z_i^T z_j
\end{aligned}$$

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 $\widetilde{K}$ 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:

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

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

Gaussian-RBF Kernel

- Most common kernel is the Gaussian RBF kernel:

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

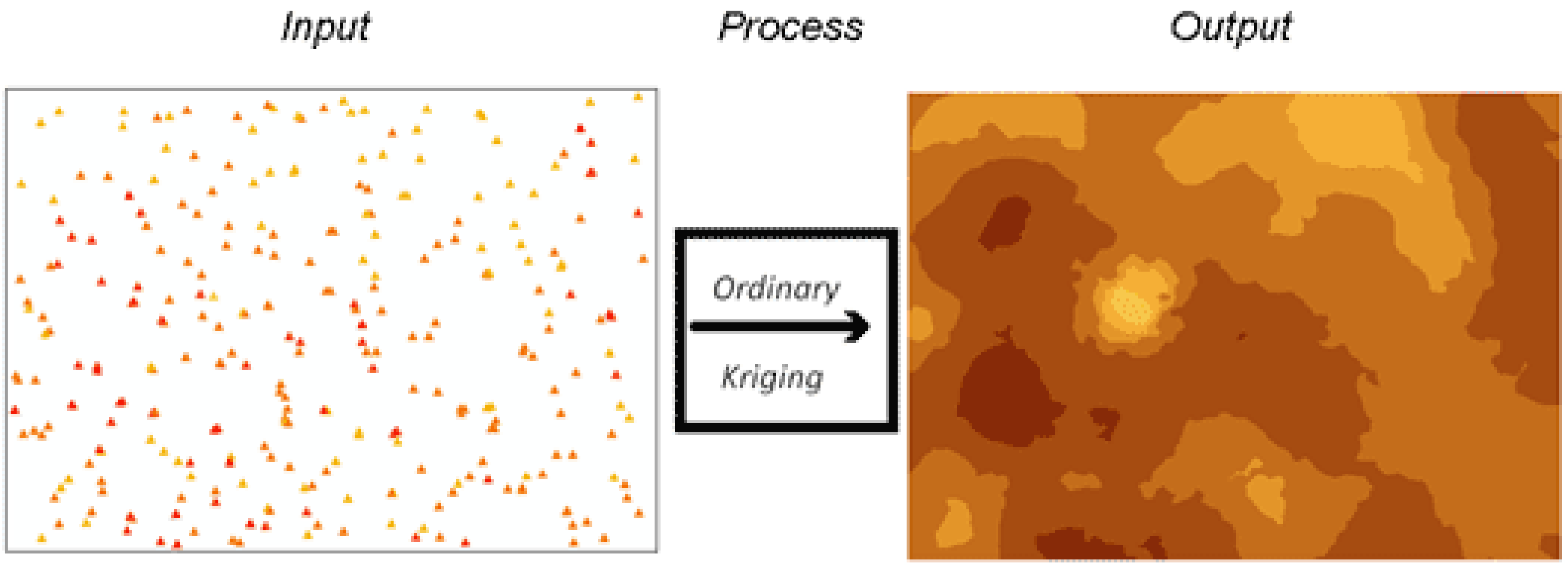
- Same formula and behaviour as RBF basis, but not equivalent:
  - Before we used RBFs as a basis, now we're using them as inner-product.

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

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

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:

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

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

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 $\min_v f(Zv) + \frac{1}{2}\|v\|^2$ for some 'Z'

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

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

Only need 'K'

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

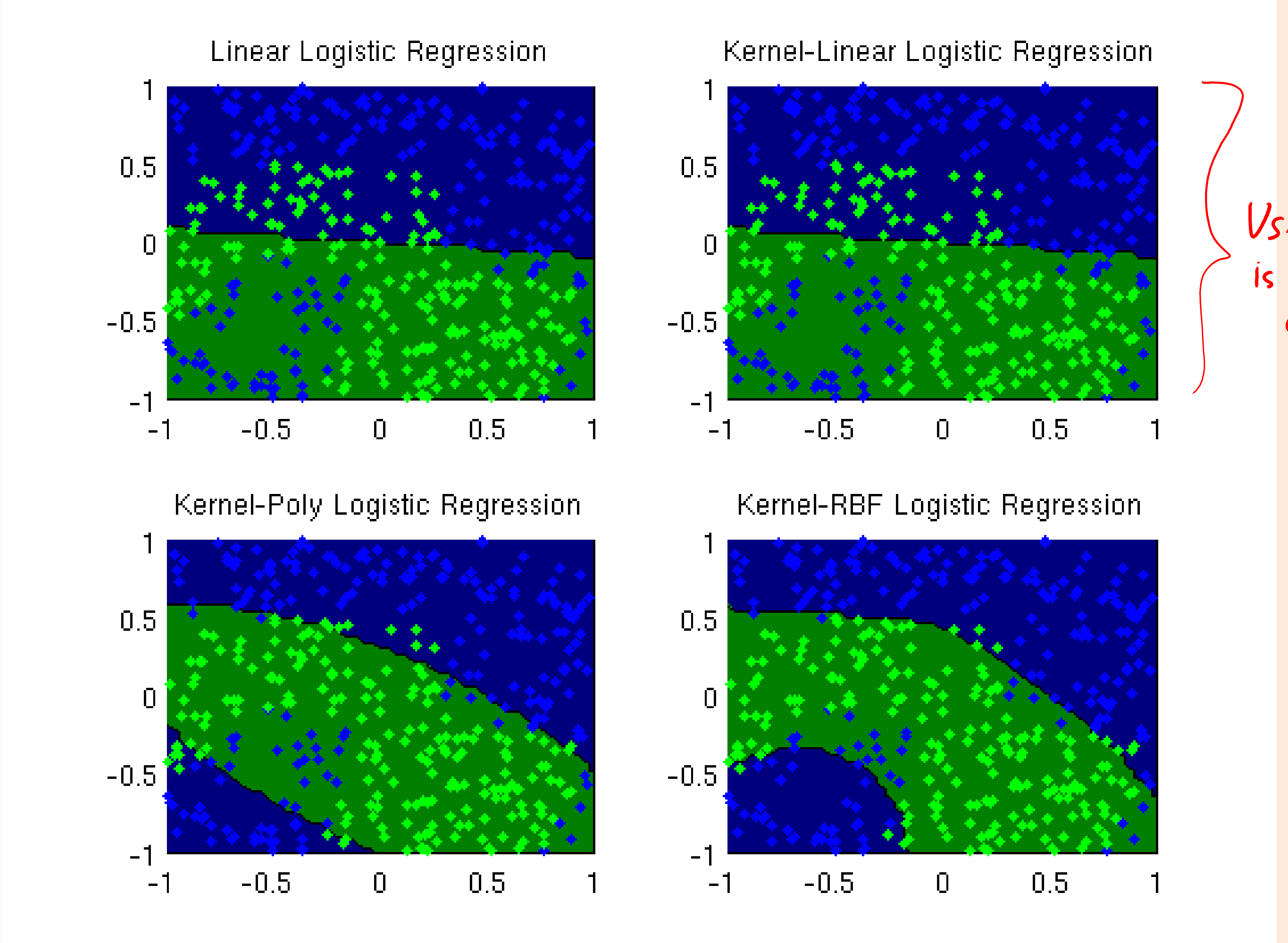
Kernel Trick for Other Methods

- 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

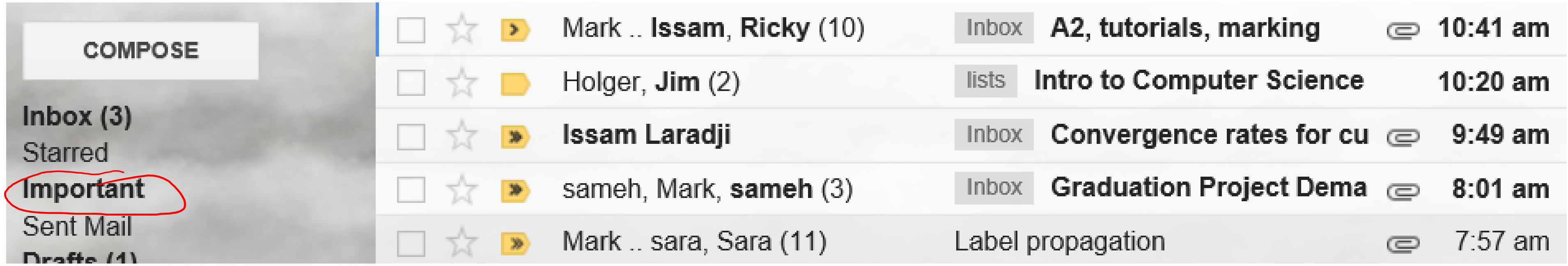


Using "linear" kernel is the same as using original features

(pause)

Motivation: "Personalized" Important E-mails

- Recall that we discussed identifying 'important' e-mails?



- There might be some "globally" important messages:
  - "This is your mother, something terrible happened, give me a call ASAP."

- But your "important" message may be unimportant to others.
  - Similar for spam: "spam" for one user could be "not spam" for another.

Digression: Linear Models with Binary Features

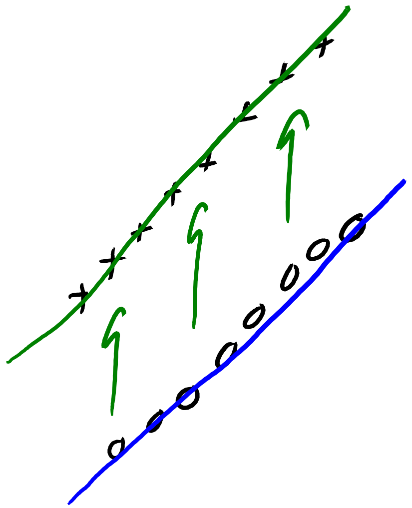
- What is the effect of a binary features on linear regression?

- Suppose we use a bag of words:
  - With 3 words "hello", "Vicodin", "340" our model would be:

$$\hat{y}_i = \underbrace{w_1 x_{i1}}_{\text{whether "hello" appears}} + w_2 x_{i2} + \underbrace{w_3 x_{i3}}_{\text{whether "340" appears}}$$

  - If e-mail only has "hello" and "340" our prediction is:

$$\hat{y}_i = \underbrace{w_1}_{\text{"hello" weight}} + \underbrace{w_3}_{\text{"340" weight}}$$



- So having the binary feature 'j' increases $\hat{y}_i$ by the fixed amount w_j.
  - Predictions are a bit like naïve Bayes where we combine features independently.
  - But now we're learning all w_j together so this tends to work better.

“Global” and “Local” Features

- Consider the following weird feature transformation:

“340”
1
1
1
0
1

“340” (any user)	“340” (user?)
1	User 1
1	User 1
1	User 2
0	<no “340”>
1	User 3

- First feature: did “340” appear in this e-mail?
- Second feature: if “340” appeared in this e-mail, who was it addressed to?

- First feature will increase/decrease importance of “340” for every user (including new users).
- Second (categorical feature) increases/decreases important of “340” for specific users.
  - Lets us learn more about users where we have a lot of data

“Global” and “Local” Features

- Recall we usually represent categorical features using “1 of k” binaries:

“340”
1
1
1
0
1

$\Rightarrow$

“340” (any user)	“340” (user = 1)	“340” (user = 2)
1	1	0
1	1	0
1	0	1
0	0	0
1	0	0

The Big Global/Local Feature Table for E-mails

- Each row is one e-mail (there are lots of rows):

$$X = \begin{bmatrix} \quad \end{bmatrix} \qquad y = \begin{bmatrix} \text{"important"} \\ \text{"not important"} \\ \vdots \end{bmatrix}$$

"global" features: shared by all users

"local" features for user "1": set to 0 for all other users.

"local" features for user "2"

We only need to store the user ID and list of non-zero features.

Predicting Importance of E-mail For New User

- Consider a new user:
  - We start out with no information about them.
  - So we use global features to predict what is important to a generic user.

$$\hat{y}_i = \text{sign}(\underbrace{w_g^T x_{ig}}_{\text{features/weights shared across users.}})$$

- With more data, update global features and user's local features:
  - Local features make prediction *personalized*.

$$\hat{y}_i = \text{sign}(w_g^T x_{ig} + \underbrace{w_u^T x_{iu}}_{\text{features/weights specific to user.}})$$

  - What is important to *this* user?
- G-mail system: classification with logistic regression.
  - Trained with a variant of stochastic gradient.

Summary

- Kernel trick allows us to use high-dimensional bases efficiently.
  - Write model to only depend on inner products between features vectors.

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

$t \times n$ matrix $\tilde{Z}Z^T$ containing inner products between test examples and training examples

$n \times n$ matrix ZZ^T containing inner products between all training examples

- 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-reg. or gradient descent).
- Global vs. local features allow "personalized" predictions.

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

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 $\sigma = 1$,

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

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

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

    then we obtain

$$\phi(x_i) = \exp(-x_i^2)\begin{bmatrix} 1 & \sqrt{\frac{2}{1!}} x_i & \sqrt{\frac{2^2}{2!}} x_i^2 & \sqrt{\frac{2^3}{3!}} x_i^3 & \cdots \end{bmatrix}.$$

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:

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

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" (bonus slide) 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 $\underbrace{\tilde{Z}v}_{X^T u} = \underbrace{\tilde{Z}Z^T}_{\tilde{K}} u = \tilde{K}u$