CPSC 340:
Machine Learning and Data Mining

Kernel Trick
Fall 2017
Admin

• Assignment 3:
  – Due Friday.

• Midterm:
  – Can view your exam during instructor office hours or after class this week.
Digression: the “other” Normal Equations

• Recall the L2-regularized least squares objective:
  \[ f(w) = \frac{1}{2} \| Xw - y \|^2 + \frac{\lambda}{2} \| w \|^2 \]

• We showed that the minimum is given by
  \[ w = (X^T X + \lambda I)^{-1} X^T y \]
  (in practice you still solve the linear system, since inverse can be numerically unstable – see CPSC 302)

• With some work (bonus), this can equivalently be written as:
  \[ w = X^T (X X^T + \lambda I)^{-1} y \]

• This is faster if \( n << d \):
  – Cost is \( O(n^2d + n^3) \) instead of \( O(nd^2 + d^3) \).
Gram Matrix

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

$$K = XX^T = \begin{bmatrix} x_1^T & x_2^T & \cdots & x_n^T \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

- $K$ contains the inner products between all training examples.
  - Similar to ‘Z’ in RBFs, but using dot product as “similarity” instead of distance.
Support Vector Machines for Non-Separable

• What about data that is not even close to separable?
Support Vector Machines for Non-Separable

• What about data that is not even close to separable?
  – It may be separable under change of basis (or closer to separable).

Support Vector Machines for Non-Separable

• What about data that is not even close to separable?
  – It may be separable under change of basis (or closer to separable).

\[ Z_i = w_1 x_{i1}^2 + w_2 \sqrt{2} x_{i2} x_{i1} + w_3 x_{i2}^2 \]

Multi-Dimensional Polynomial Basis

• Recall fitting polynomials when we only have 1 feature:

\[ y_i = w_0 + w_1 x_i + w_2 x_i^2 \]

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

\[
\begin{pmatrix}
0.2 \\
-0.5 \\
1 \\
4
\end{pmatrix}
\begin{pmatrix}
0.2 & (0.2)^2 \\
-0.5 & (-0.5)^2 \\
1 & (1)^2 \\
4 & (4)^2
\end{pmatrix}
\]

• How can we do this when we have a lot of features?
Multi-Dimensional Polynomial Basis

• Polynomial basis for \(d=2\) and \(p=2\):

\[
X = \begin{bmatrix}
0.2 & 0.3 \\
1 & 0.5 \\
-0.5 & -0.1
\end{bmatrix}
\quad Z = \begin{bmatrix}
1 & 0.2 & 0.3 & (0.2)^2 & (0.3)^2 & (0.1)(0.3) \\
1 & 1 & 0.5 & (0.1)^2 & (0.5)^2 & (1)(0.5) \\
1 & 1 & 0.5 & -0.1 & (0.5)^2 & (-0.5)(-0.1) \\
\hline
h_{ia_5} & x_{i1} & x_{i2} & (x_{i1})^2 & (x_{i2})^2 & (x_{i1})(x_{i2})
\end{bmatrix}
\]

• 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: \(x_{i1}x_{i2}\), \(x_{i1}x_{i3}\), \(x_{i1}x_{i4}\), \(x_{i2}x_{i3}\), \(x_{i2}x_{i4}\), \(x_{i3}x_{i4}\), \(x_{i1}x_{i3}\).
  - Cubic interactions: \(x_{i1}x_{i2}x_{i3}\), \(x_{i2}x_{i3}x_{i4}\), \(x_{i1}x_{i3}x_{i4}\), \(x_{i1}^2x_{i2}\), \(x_{i1}^2x_{i3}\), \(x_{i1}^2x_{i4}\), \(x_{i1}x_{i2}^2\), \(x_{i1}x_{i3}^2\), \(x_{i1}x_{i4}^2\), \(x_{i2}x_{i3}^2\), \(x_{i2}x_{i4}^2\), \(x_{i3}x_{i4}^2\).
Kernel Trick

• If we go to degree \( p=5 \), we’ll have \( O(d^5) \) quintic terms:

\[
X_{i1}^5 X_{i2}^4 X_{i3}^4 \ldots \ldots X_{id}^4 X_{i1}^3 X_{i2}^2 X_{i3}^2 \ldots \ldots X_{id}^2 X_{i1} X_{i2} X_{i3} \ldots \ldots X_{id}^5
\]

  – For large ‘d’ and ‘p’, we can’t even store ‘Z’ or ‘w’.

• But, we can use this basis efficiently with the kernel trick (medium ‘n’).

• Basic idea:
  – We can sometimes efficiently compute dot product \( z_i^T z_j \) directly from \( x_i \) and \( x_j \).
  – Use this to make the Gram matrix \( ZZ^T \) and make predictions.
Kernel Trick

• Given test data $\hat{X}$, predict $\hat{y}$ by forming and $\hat{Z}$ using:

$$\hat{y} = \hat{Z}w$$

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

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

• Key observation behind kernel trick:
  – Predictions $\hat{y}$ only depend on features through $K$ and $\hat{K}$.
  – If we have a function that computes $K$ and $\hat{K}$, we don’t need the features.
Kernel Trick

• ‘K’ contains the inner products between all training examples.
  – Intuition: inner product can be viewed as a measure of similarity, so this matrix gives a similarity between each pair of examples.
• ‘\( \hat{K} \)’ contains the inner products between training and test examples.

• Kernel trick:
  – I want to use a basis \( z_i \) that is too huge to store (very large ‘d’).
  – But I only need \( z_i \) to compute Gram matrix \( K = ZZ^T \) and \( \hat{K} = \hat{Z}\hat{Z}^T \).
    • The sizes of these matrices are independent of \( d \).
    • Everything we need to know about \( z_i \) is summarized by the \( z_i^Tz_j \).
  – I can use this basis if I have a kernel function that computes \( k(x_i, x_j) = z_i^Tz_j \).
    • I don’t need to compute the basis \( z_i \) explicitly.
Example: Degree-2 Kernel

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

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

• And consider a particular degree-2 basis:

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

• We can compute inner product $z_i^T z_j$ without forming $z_i$ and $z_j$:

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

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

$$= (x_{i1} x_{j1} + x_{i2} x_{j2})^2 \quad \text{"completing the square"}$$

$$= (x_i^T x_j)^2 \quad \text{No need for } z_i \text{ to compute } z_i^T z_j$$
Polynomial Kernel with Higher Degrees

- Let’s add a bias and linear terms to our degree-2 basis:
  \[ z_i = \begin{bmatrix} 1 & \sqrt{2}x_{i1} & \sqrt{2}x_{i2} & x_{i1}^2 & \sqrt{2}x_{i1}x_{i2} & x_{i2}^2 \end{bmatrix}^T \]

- I can compute inner products using:

  \[
  (1 + x_i^T x_j)^2 = 1 + 2x_i^T x_j + (x_i^T x_j)^2
  = 1 + 2x_{i1}x_{j1} + 2x_{i2}x_{j2} + x_{i1}^2x_{j1}^2 + 2x_{i1}x_{i2}x_{j1}x_{j2} + x_{i2}^2x_{j2}^2
  \]

\[
= \begin{bmatrix} 1 & \sqrt{2}x_{i1} & \sqrt{2}x_{i2} & x_{i1}^2 & \sqrt{2}x_{i1}x_{i2} & x_{i2}^2 \end{bmatrix} \begin{bmatrix} 1 \\ \sqrt{2}x_{j1} \\ \sqrt{2}x_{j2} \\ x_{j1}^2 \\ x_{j2}^2 \\ \sqrt{2}x_{j1}x_{j2} \end{bmatrix}^T
= z_i^T z_j
\]
Polynomial Kernel with Higher Degrees

• To get all degree-4 “monomials” I can use:

\[ z_i^\top z_j = (x_i^\top x_j)^4 \]

Equivalent to using a \( z_i \) with weighted versions of \( x_i^4, x_i^3 x_j, x_i^2 x_j^2, x_i x_j^3, \ldots \).

• To also get lower-order terms use \( z_i^\top z_j = (1 + x_i^\top x_j)^4 \).

• The general degree-\( p \) polynomial kernel function:

\[ k(x_i, x_j) = (1 + x_i^\top x_j)^p \]

– Works for any number of features ‘\( d \)’.
– But cost of computing one \( z_i^\top z_j \) is \( O(d) \) instead of \( O(d^p) \).
– 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:
    
    $K_{ij} = (1 + x_i^T x_j)^p$  \quad $\tilde{K}_{ij} = (1 + \tilde{x}_i^T \tilde{x}_j)^p$

  – Make predictions using:
    
    $\hat{y} = \tilde{K} (K + \lambda I)^{-1} y$

• Training cost is only $O(n^2d + n^3)$, despite using $O(d^p)$ 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 $d \tilde{K}$. 

Linear Regression vs. Kernel Regression

**Linear Regression**

**Training**
1. Form basis $Z$ from $X$.
2. Compute $w = (Z^T Z + \lambda I)^{-1} Z^T y$

**Testing**
1. Form basis $\hat{Z}$ from $\hat{X}$
2. Compute $\hat{y} = \hat{Z} w$

**Kernel Regression**

**Training**
1. Form inner products $K$ from $X$.
2. Compute $v = (K + \lambda I)^{-1} y$

**Testing**
1. Form inner products $K$ from $X$ and $\hat{X}$
2. Compute $\hat{y} = K v$

Non-parametric
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).

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 \( \sigma=1 \), it corresponds to using this basis (bonus slide):

\[ z_j = \exp(-x_i^2) \left[ 1 \quad \sqrt{\frac{3}{2}} x_i \quad \sqrt{\frac{3}{2^3}} x_i^2 \quad \sqrt{\frac{3}{2^5}} x_i^3 \quad \sqrt{\frac{3}{2^9}} x_i^4 \quad \ldots \right] \]
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:

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

• 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 $x_i$ 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 eigenvalue equation.

• 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:
    \[ ||z_i - z_j||^2 = z_i^T z_i - 2 z_i^T z_j + z_j^T z_j = k(x_i, x_i) - 2 k(x_i, x_j) + k(x_j, x_j) \]

  – 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**:
    • 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 \(O(ndl)\) to \(O(md)\).]

  \(n\) Number of support vectors.
Logistic Regression with Kernels
Summary

- **High-dimensional bases** allows us to separate non-separable data.
- **Kernel trick** allows us to use high-dimensional bases efficiently.
  - Write model to only depend on inner products between features vectors.
    \[ \hat{\gamma} = \hat{k}(K + \lambda I)^{-1}y \]
  - Kernels let us use similarity between objects, rather than features.
    - Allows some exponential- or infinite-sized feature sets.
    - Applies to L2-regularized linear models and distance-based models.
- 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:

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

- 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.
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 \( E^{-1} = (\frac{1}{\lambda}) I \)), \( F = X^T \), \( H = -I \) (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 \(-\lambda\),

\[
-(\frac{1}{\lambda}) I X^T (-I - X \left(\frac{1}{\lambda}\right) X^T)^{-1} = X^T (\lambda I + X X^T)^{-1} = X^T (X X^T + \lambda I)^{-1}.
\]
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 \( \sigma = 1 \),
    \[ k(x_i, x_j) = \exp(-x_i^2 + 2x_i x_j - x_j^2) \]
    \[ = \exp(-x_i^2) \exp(2x_i x_j) \exp(-x_j^2), \]
    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
    \[ \phi(x_i) = \exp(-x_i^2) \left[ 1 \quad \sqrt{\frac{2}{1!}} x_i \quad \sqrt{\frac{2^2}{2!}} x_i^2 \quad \sqrt{\frac{2^3}{3!}} x_i^3 \quad \cdots \right]. \]
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:

\[
\begin{align*}
  k(x_i, x_j) &= \exp \left( -\frac{\|x_i - x_j\|^2}{\sigma^2} \right) \\
  &= \exp \left( -\frac{\|x_i\|^2}{\sigma^2} \right) \exp \left( \frac{2}{\sigma^2} x_i^T x_j \right) \exp \left( -\frac{\|x_j\|^2}{\sigma^2} \right) \\
  &= \exp(\text{valid}) \cdot \phi(x_i) \cdot \phi(x_j)
\end{align*}
\]
Representer Theorem

- Consider linear model differentiable with losses $f_i$ and L2-regularization,
  
  $$\arg\min_{w \in \mathbb{R}^d} \sum_{i=1}^{n} f_i(w^T x_i) + \frac{\lambda}{2} \|w\|^2.$$

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

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

- Using representer theorem we can use \( w = X^T z \) in original problem,

\[
\arg\min_{w \in \mathbb{R}^d} \sum_{i=1}^{n} f_i(w^T x_i) + \frac{\lambda}{2} \|w\|^2
\]

\[
= \arg\min_{z \in \mathbb{R}^n} \sum_{i=1}^{n} f_i(z^T x_i) + \frac{\lambda}{2} \|X^T z\|^2
\]

- Now defining \( f(z) = \sum_{i=1}^{n} f_i(z_i) \) for a vector \( z \) we have

\[
= \arg\min_{z \in \mathbb{R}^n} f(X X^T z) + \frac{\lambda}{2} z^T X X^T z
\]

\[
= \arg\min_{z \in \mathbb{R}^n} f(K z) + \frac{\lambda}{2} z^T K z.
\]

- Similarly, at test time we can use the \( n \) variables \( z \),

\[
\hat{X} w = \hat{X} X^T z = \hat{K} z.
\]