CPSC 540: Machine Learning

Kernel Methods and Fenchel Duality

Mark Schmidt

University of British Columbia

Winter 2019

Motivation: Multi-Dimensional Polynomial Basis

- Consider quadratic polynomial basis with only have two features ($x^i \in \mathbb{R}^2$):

$$\hat{y}^i = w_0 + w_1 x_1^i + w_2 x_2^i + w_2 (x_1^i)^2 + w_3 (x_2^i)^2 + w_4 x_1^i x_2^i.$$

- In 340 we saw that we can fit this model using a change of basis:

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

- If you have $d = 100$ and $p = 5$, there are $O(100^5)$ possible degree-5 terms:

$$(x_1^i)^5, (x_1^i)^4 x_2^i, (x_1)^4 x_3^i, \ldots, (x_1^i)^3 (x_2^i)^2, (x_1^i)^3 (x_2^i)^2, \ldots, (x_1^i)^3 x_2^i x_3^i, \ldots$$

- How can we do this when number of features k in basis is huge?

The "Other" Normal Equations

- Recall the L2-regularized least squares model with basis Z,

$$\underset{v \in \mathbb{R}^d}{\text{argmin}} \frac{1}{2}\|Zv - y\|^2 + \frac{\lambda}{2}\|v\|^2.$$

- By solving for $\nabla f(v) = 0$ we get that

$$v = (\underbrace{Z^\top Z}_{k \text{ by } k} + \lambda I_d)^{-1} Z^\top y,$$

  where I_d is the k by k identity matrix.
- An equivalent way to write the solution is:

$$v = Z^\top (\underbrace{ZZ^\top}_{n \text{ by } n} + \lambda I_n)^{-1} y,$$

  by using a variant of the matrix inversion lemma (bonus slide).
- Computing v with this formula is faster if $n << k$:
  - $ZZ^\top$ is n by n while $Z^\top Z$ is k by k.

Predictions using Equivalent Form

- Given test data $\tilde{X}$, we predict $\hat{y}$ using:

$$\begin{aligned}\hat{y} &= \tilde{Z}v \\ &= \tilde{Z}\underbrace{Z^\top(ZZ^\top + \lambda I_n)^{-1}y}_{\text{"other" normal equations}}\end{aligned}$$

- If we define $K = ZZ^\top$ (Gram matrix) and $\tilde{K} = \tilde{Z}Z^\top$, then we have

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

  where K is $n \times n$ and $\tilde{K}$ is $t \times n$.

- Key observation behind kernel trick:
  - If we can directly compute K and $\tilde{K}$, we don't need to form Z or $\tilde{Z}$.

Gram Matrix

- The Gram matrix K is defined by:

$$K = ZZ^\top = \begin{bmatrix} — & (z^1)^\top & — \\ — & (z^2)^\top & — \\ & \vdots & \\ — & (z^n)^\top & — \end{bmatrix} \begin{bmatrix} | & | & | & \cdots & | \\ z^1 & z^2 & z^3 & \cdots & z^n \\ | & | & | & \cdots & | \end{bmatrix}$$

$$= \begin{bmatrix} \langle z^1, z^1 \rangle & \langle z^1, z^2 \rangle & \cdots & \langle z^1, z^n \rangle \\ \langle z^2, z^1 \rangle & \langle z^2, z^2 \rangle & \cdots & \langle z^2, z^n \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle z^n, z^1 \rangle & \langle z^n, z^2 \rangle & \cdots & \langle z^n, z^n \rangle \end{bmatrix} = \begin{bmatrix} k(x^1, x^1) & k(x^1, x^2) & \cdots & k(x^1, x^n) \\ k(x^2, x^1) & k(x^2, x^2) & \cdots & k(x^2, x^n) \\ \vdots & \vdots & \ddots & \vdots \\ k(x^n, x^1) & k(x^n, x^2) & \cdots & k(x^n, x^n) \end{bmatrix}$$

- K contains the inner products between all training examples in basis z
- $\tilde{K}$ contains the inner products between training and test examples.
  - Kernel trick: if we can compute $k(x^i, x^j) = \langle z^i, z^j \rangle$, we don't need z^i and z^j.

Polynomial Kernel

- In 340 we saw the polynomial kernel of degree p,

$$k(x^i, x^j) = (1 + \langle x^i, x^j \rangle)^p,$$

  which corresponds to a general degree-p polynomial z^i.

- You can make predictions with these z^i using

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

- Total cost is only $O(n^2 d + n^3)$ even though number of features is $O(d^p)$.

- Kernel trick:
  - We have kernel function $k(x^i, x^j)$ that gives $\langle z^i, z^j \rangle$.
  - Skip forming Z and directly form K and $\tilde{K}$.
  - Size of K is n by n even if Z has exponential or infinite columns.

Guasian-RBF Kernels

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

$$k(x^i, x^j) = \exp\left(-\frac{\|x^i - x^j\|^2}{2\sigma^2}\right).$$

- What features z_i would lead to this as the inner-product?
  - To simplify, assume $d = 1$ and $\sigma = 1$,

$$k(x^i, x^j) = \exp\left(-\frac{1}{2}(x^i)^2 + x^i x^j - \frac{1}{2}(x^j)^2\right) = \exp\left(-\frac{1}{2}(x^i)^2\right)\exp(x^i x^j)\exp\left(-\frac{1}{2}(x^j)^2\right),$$

    so we need $z_i = \exp(-\frac{1}{2}(x^i)^2)u_i$ where $u_i u_j = \exp(x^i x^j)$.
    - For this to work for *all* x^i and x^j, z_i must be infinite-dimensional.
  - If we use that

$$\exp(x^i x^j) = \sum_{k=0}^{\infty} \frac{(x^i)^k (x^j)^k}{k!},$$

    then we obtain

$$z_i = \exp\left(-\frac{1}{2}(x^i)^2\right)\begin{bmatrix} 1 & \frac{1}{\sqrt{1!}}x^i & \frac{1}{\sqrt{2!}}(x^i)^2 & \frac{1}{\sqrt{3!}}(x^i)^3 & \cdots \end{bmatrix}.$$

Kernel Trick for Structured Data

- Kernel trick can be useful for structured data:
  - Consider data 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:

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

  - It might be easier to define a "similarity" between sentences than to define features.

Kernel Trick for Structured Data

- A classic "string kernel":
  - We want to compute k("cat", "cart").
  - Find common subsequences: 'c', 'a', 't', 'ca', 'at', 'ct', 'cat'.
  - Weight them by total length in original strings:
    - 'c' is has lengths (1,1), 'ca' has lengths (2,2), 'ct' has lengths (3,4), and son.
  - 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 linear time by dynamic programming.

- Many variations exist. And there are "image kernels", "graph kernels", and so on.
  - You can turn probabilities over examples (second half of course) into kernels.
  - A survey on the topic is here.

Valid Kernels

- Can we use any function k for our kernel/similarity function $k(x^i, x^j)$?

- We need to have kernel k be an inner product in some space:
  - There exists transformation $z^i = \phi(x^i)$ such that $k(x^i, x^j) = \langle \phi(x^i), \phi(x^j) \rangle$.

We can decompose a (continuous or finite-domain) function k into

$$k(x^i, x^j) = \langle \phi(x^i), \phi(x^j) \rangle,$$

iff it is symmetric and for any finite $\{x^1, x^2, \dots, x^n\}$ we have $K \succeq 0$.

- For finite domains you can show existence of ϕ using spectral theorem (bonus).
  - The general case is called Mercer's Theorem.

Valid Kernels

- Mercer's Theorem is nice in theory, what do we do in practice?
  - You could show explicitly that $k(x^i, x^j) = \langle \langle \phi(x^i), \phi(x^j) \rangle$ for some function ϕ.
  - You could that K is positive semi-definite by construction.
  - Or you can show k is constructed from other valid kernels.

(If we use invalid kernel, lose feature-space interpretation but may work fine.)

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:
  - Non-negative scaling: $\alpha k_1(x^i, x^j)$ for $\alpha \geq 0$.
  - Sum: $k_1(x^i, x^j) + k_2(x^i, x^j)$.
  - Product: $k_1(x^i, x^j)k_2(x^i, x^j)$.
    - Special case: $\phi(x^i)k_1(x^i, x^j)\phi(x^j)$ for any function ϕ.
  - Exponentiation: $\exp(k_1(x^i, x^j))$.
  - Recursion: $k_1(\phi(x^i), \phi(x^j))$ for any function ϕ.

- Example: Gaussian-RBF kernel:

$$k(x^i, x^j) = \exp\left(-\frac{\|x^i - x^j\|^2}{2\sigma^2}\right) = \exp\left(-\frac{\|x^i\|^2}{2\sigma^2} + \frac{1}{\sigma^2}\langle x^i, x_j\rangle - \frac{1}{2\sigma^2}\|x^j\|^2\right)$$

$$= \underbrace{\exp\left(-\frac{\|x^i\|^2}{2\sigma^2}\right)}_{\phi(x^i)} \underbrace{\exp\left(\underbrace{\frac{1}{\sigma^2}}_{\alpha>0} \underbrace{\langle x^i, x^j\rangle}_{\text{valid}}\right)}_{\exp(\text{valid})} \underbrace{\exp\left(-\frac{\|x^j\|^2}{2\sigma^2}\right)}_{\phi(x^j)}.$$

Models allowing Kernel Trick

- Besides L2-regularized least squares, when can we apply the kernel trick?
  - Distance-based methods from CPSC 340:

$$\begin{aligned}\|z^i - z^j\|^2 &= \langle z^i, z^i\rangle - 2\langle z^i, z^j\rangle + \langle z^j, z^j\rangle \\ &= k(x^i, x^i) - 2k(x^i, x^j) + k(x^j, x^j).\end{aligned}$$

    - k-nearest neighbours.
    - Clustering algorithms (k-means, density-based clustering, hierarchical clustering).
    - Distance-based outlier detection (KNN-based, outlier ratio)
    - "Amazon product recommendation".
    - Multi-dimensional scaling (ISOMAP, t-SNE).
    - Label propagation.
  - L2-regularized linear models (today).
  - Eigenvalue methods:
    - Principle component analysis (need trick for centering in high-dimensional space).
    - Canonical correlation analysis.
    - Spectral clustering.

Representer Theorem

- Consider L2-regularized loss only depending on Xw,

$$\operatorname*{argmin}_{w \in \mathbb{R}^d} f(Xw) + \frac{\lambda}{2}\|w\|^2.$$

- Setting the gradient equal to zero we get

$$0 = X^\top r + \lambda w,$$

where $r = \nabla f(Aw)$.

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

$$w^* = -\frac{1}{\lambda} X^\top r = X^\top v,$$

where $v = \frac{1}{\lambda} r$ (this means we can restrict to w satisfying $w = X^\top v$).

Representer Theorem

- Since we know $w^* = X^\top v$ for some v, let's optimize over v instead of w:

$$\begin{aligned}
&\operatorname*{argmin}_{w \in \mathbb{R}^d} f(Xw) + \frac{\lambda}{2}\|w\|^2 \\
=&\operatorname*{argmin}_{v \in \mathbb{R}^n} f(XX^T v) + \frac{\lambda}{2}\|X^\top v\|^2 \\
=&\operatorname*{argmin}_{v \in \mathbb{R}^n} f(XX^T v) + \frac{\lambda}{2} v^T XX^T v \\
\equiv&\operatorname*{argmin}_{v \in \mathbb{R}^n} f(Kv) + \frac{\lambda}{2} v^\top Kv.
\end{aligned}$$

- Which is a kernelized version of the problem.

Representer Theorem

- Using $w = X^\top v$, at test time we use

$$\begin{aligned}\hat{y} &= \tilde{X}w \\ &= \tilde{X}X^\top v \\ &= \tilde{K}v,\end{aligned}$$

  or that each $\hat{y}^i = \sum_{j=1}^n v_j k(\tilde{x}^i, x^j)$.

- That prediction is a linear combination of kernels is called representer theorem.
  - It holds under more general conditions, including non-smooth f_i like SVMs.

Multiple Kernel Learning

- We can kernelize L2-regularized linear models,

$$\underset{w\in\mathbb{R}^d}{\operatorname{argmin}} f(Xw,y)+\frac{\lambda}{2}\|w\|^2 \Leftrightarrow \underset{v\in\mathbb{R}^n}{\operatorname{argmin}} f(Kv,y)+\frac{\lambda}{2}\|v\|_K^2,$$

  under fairly general conditions.
- What if we have multiple potential kernels and don't know which to use?
  - Obvious approach: cross-validation to choose the best one.
- What if we have multiple potentially-relevant kernels?
  - Multiple kernel learning:

$$\underset{v_1\in\mathbb{R}^n,v_2\in\mathbb{R}^n,\ldots,v_k\in\mathbb{R}^n}{\operatorname{argmin}} f\left(\sum_{c=1}^{k}K_c v_c,y\right)+\frac{1}{2}\sum_{c=1}^{k}\lambda_c\|v\|_{K_c}.$$

  - Defines a valid kernel and is convex if f is convex (affine function).
  - Group L1-regularization of parameters associated with each kernel.
    - Selects a sparse set of kernels.
  - Hiearchical kernel learning:
    - Use structured sparsity to search through exponential number of kernels.

Outline

Motvation: Getting Rid of the Step-Size

- SVMs are a widely-used model but objective is non-differentiable.
  - The non-differentiable part is the loss, which isn't nice like L1-regularization.
  - We can't apply coordinate optimization or proximal-gradient or SAG.
- Stochastic subgradient methods achieve $O(1/\epsilon)$ without dependence on n.
  - But choosing the step-size is painful.
- Can we develop a method where choosing the step-size is easy?
  - To do this, we first need the concept of the Lagrangian...

Lagrangian Function for Equality Constraints

- Consider minimizing a differentiable f with linear equality constraints,

$$\operatorname*{argmin}_{Ax=b} f(x).$$

- The Lagrangian of this problem is defined by

$$L(x, z) = f(x) + z^\top (Ax - b),$$

for a vector $z \in \mathbb{R}^n$ (with A being n by d).

- At a solution of the problem we must have

$$\nabla_x L(x, z) = \nabla f(x) + A^\top z = 0 \qquad \text{(gradient is orthogonal to constraints)}$$
$$\nabla_z L(x, z) = Ax - b = 0 \qquad \text{(constraints are satisfied)}$$

- So solution is stationary point of Lagrangian.

Dual Function

- But we can't just minimize with respect to x and z.
- The solution for convex f is actually a saddle point,

$$\max_z \min_x L(x, z).$$

(in cases where the $\max$ and $\min$ have solutions)

- One way to solve this is to eliminate x,

$$\max_z D(z),$$

where $D(z) = \min_x L(x, z)$ is called the dual function.

- Another method is eliminate constraints (see Michael Friedlander's course).

(find a feasible x, find basis for null-space of A, optimize f over null-space.)

Digression: Supremum and Infimum

- To handle case where $\min_x f(x)$ is not achieved for any x, we can use infimum.
- Generalization of $\min$ that includes limits:

$$\min_{x \in \mathbb{R}} x^2 = 0, \quad \inf_{x \in \mathbb{R}} x^2 = 0,$$

but

$$\min_{x \in \mathbb{R}} e^x = \text{DNE}, \quad \inf_{x \in \mathbb{R}} e^x = 0.$$

- The infimum of a function f is its largest lower-bound,

$$\inf f(x) = \max_{y \mid y \leq f(x)} y.$$

- The analogy for $\max$ is called the supremum ($\sup$).

Dual function

- Even for non-smooth convex f solution is a saddle point of the Lagrangian,

$$\max_z \inf_x \underbrace{f(x) + z^\top (Ax - b)}_{L(x,z)}.$$

(restricted to z where the $\max$ is finite)

- We're going to eliminate x by working with the dual function,

$$\max_z D(z),$$

with $D(z) = \inf_x \{f(x) + z^\top (Ax - b)\}$.

(D is concave for any f, so $-D$ is convex)

- Why?????
  - If f is strongly-convex, dual is smooth (not obvious).
  - Dual sometimes has sparse kernel representation.
  - Dual has fewer variables if $n < d$.
  - Dual gives lower bound, $D(z) \leq f(x)$ (weak duality).
  - We can solve dual instead of primal, $D(z^*) = f(x^*)$ (strong duality).

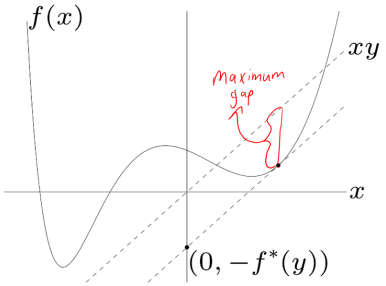
(see Michael Friedlander's class for details/conditions.)

Convex Conjugate

- The convex conjugate f^* of a function f is given by

$$f^*(y) = \sup_{x \in \mathcal{X}} \{y^\top x - f(x)\},$$

where $\mathcal{X}$ is values where $\sup$ is finite.



http://www.seas.ucla.edu/~vandenbe/236C/lectures/conj.pdf

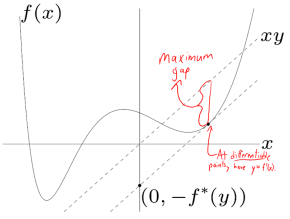
- It's the maximum that the linear function $y^\top x$ can get above $f(x)$.

Convex Conjugate

- The convex conjugate f^* of a function f is given by

$$f^*(y) = \sup_{x \in \mathcal{X}} \{y^\top x - f(x)\},$$

where $\mathcal{X}$ is values where $\sup$ is finite.



http://www.seas.ucla.edu/~vandenbe/236C/lectures/conj.pdf

- If f is differentable, then $\sup$ occurs at x where $y = \nabla f(x)$.
- Note that f^* is convex even if f is not (but we may lose strong duality).
- If f is convex then $f^{**} = f$ ("closed" f).

Convex Conjugate Examples

- If $f(x) = \frac{1}{2}\|x\|^2$ we have
  - $f^*(y) = \sup_x\{y^\top x - \frac{1}{2}\|x\|^2\}$ or equivalently (by taking derivative and setting to 0):

$$0 = y - x,$$

and pluggin in $x = y$ we get

$$f^*(y) = y^\top y - \frac{1}{2}\|y\|^2 = \frac{1}{2}\|y\|^2.$$

- If $f(x) = a^\top x$ we have

$$f^*(y) = \sup_x\{y^\top x - a^\top x\} = \sup_x\{(y-a)^\top x\} = \begin{cases} 0 & y = a \\ \infty & \text{otherwise.} \end{cases}$$

- For other examples, see Boyd & Vandenberghe.

Fenchel Dual

- In machine learning our primal problem is usually (for convex f and r)

$$\underset{w \in \mathbb{R}^d}{\text{argmin}}\, f(Xw) + r(w).$$

- If we introduce equality constraints,

$$\underset{v = Xw}{\text{argmin}}\, f(v) + r(w).$$

then dual has a special form called the Fenchel dual,

$$\underset{z \in \mathbb{R}^n}{\text{argmax}}\, D(z) = -f^*(-z) - r^*(X^\top z),$$

where we're maximizing the (negative) convex conjugates f^* and r^*.

(bonus slide)

- If r is strongly-convex, dual will be smooth...

Fenchel Dual of SVMs

- Consider support vector machines,

$$\underset{w\in\mathbb{R}^d}{\text{argmin}} \sum_{i=1}^{n} \max\{0, 1 - y_i w^\top x_i\} + \frac{\lambda}{2}\|w\|^2.$$

- The Fenchel dual is given by

$$\underset{0\leq z\leq 1}{\text{argmax}} \sum_{i=1}^{n} z_i - \frac{1}{2\lambda} \underbrace{\|X^\top Y z\|^2}_{z^\top Y X X^\top Y z},$$

  with $w^* = \frac{1}{\lambda} X^\top Y z^*$ and constraints coming from $f^* < \infty$.
- A couple magical things have happened:
  - We can apply kernel trick.
  - Non-negativity makes dual variables z sparse (non-zeroes are "support vectors"):
    - Can give faster training and testing.
  - Dual is differentiable (though not strongly-convex).
    - And for this function coordinate optimization is efficient.

Stochastic Dual Coordinate Ascent

- If we have an L2-regularized linear model,

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

then Fenchel dual is a problem where we can apply coordinate optimization,

$$\underset{z\in\mathbb{R}^n}{\text{argmax}} -\underbrace{\sum_{i=1}^{n} f_i^*(z_i)}_{\text{separable}} - \frac{1}{2\lambda}\underbrace{\|X^\top z\|^2}_{z^\top XX^\top z}.$$

- It's known as stochastic dual coordinate ascent (SDCA):
  - Only needs to looks at one training example on each iteration.
  - Obtains $O(\log(1/\epsilon))$ rate if ∇f_i are L-Lipschitz.
    - Performance similar to SAG for many problems, worse if $\mu >> \lambda$.
  - Obtains $O(1/\epsilon)$ rate for non-smooth f:
    - Same rate/cost as stochastic subgradient, but we can use exact/adaptive step-size.

Outline

Large-Scale Kernel Methods

- Let's go back to the basic L2-regularized least squares setting,

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

- Obvious drawback of kernel methods: we can't compute/store K.
  - It has $O(n^2)$ elements.

- Standard general approaches:
  1. Kernels with special structure.
  2. Subsampling methods.
  3. Explicit feature construction.

Kernels with Special Structure

- The bottleneck in fitting the model is $O(n^3)$ cost of solving the linear system

$$(K + \lambda I)v = y.$$

- Consider using the "identity" kernel,

$$k(x^i, x^j) = \mathbb{I}[x^i = x^j].$$

- In this case K is diagonal so we can solve linear system in $O(n)$.

- More interesting special K structures that support fast linear algebra:
  - Band-diagonal matrices.
  - Sparse matrices (via conjugate gradient).
  - Diagonal plus low-rank, $D + UV^\top$.
  - Toeplitz matrices.
  - Kronecker product matrices.
  - Fast Gauss transform.

Subsampling Methods

- In subsampling methods we only use a subset of the kernels.
- For example, some loss functions have support vectors.
  - But this mainly helps at testing time, and some problems have $O(n)$ support vectors.
- Nystrom approximation chooses a random and fixed subset of training examples.
  - Many variations exist such as greedily choosing kernels.
- A common variation is the subset of regressors approach....

Subsampling Methods

- Consider partitioning our matrices as

$$K = \begin{bmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{bmatrix} = \begin{bmatrix} K_1 & K_2 \end{bmatrix}, \quad \hat{K} = \begin{bmatrix} \hat{K}_1 & \hat{K}_2 \end{bmatrix},$$

  where K_{11} corresponds to a set of m training examples
  - K is m by m, K_1 is n by m.

- In subset of regressors we use the approximation

$$K \approx K_1 K_{11}^{-1} K_1^\top, \quad \hat{K} \approx \hat{K}_1 K_{11}^{-1} K_1^\top.$$

- Which for L2-regularized least squares can be shown to give

$$\hat{y} = \hat{K}_1 \underbrace{(K_1^\top K_1 + \lambda K_{11})^{-1} K_1^\top y}_{v}.$$

- Given K_1 and K_{11}, computing v costs $O(m^2 n + m^3)$ which is cheap for small m.

Explicit Feature Construction

- In explicit feature methods, we form Z such that $Z^\top Z \approx K$.
  - But where Z has a small number of columns of m.
- We then use our non-kernelized approach with features Z,

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

- Random kitchen sinks approach does this for translation-invariant kernels,

$$k(x^i, x^j) = k(x^i - x^j, 0),$$

  by sampling elements of inverse Fourier transform (not obvious).
- In the special case of the Gaussian RBF kernel this gives $Z = \exp(iXR)$.
  - R is a d by m matrix with elements sampled from the Gaussian (same variance).
  - i is $\sqrt{-1}$ and $\exp$ is taken element-wise.

Summary

- Kernel trick: allows working with "similarity" instead of features.
  - Also allows exponential- or infinite-sized feature spaces.
- Valid kernels are typically constructed from other valid kernels.
- Representer theorem allows kernel trick for L2-regularized linear models.
- Fenchel dual re-writes sum of convex functions with convex conjugates:
  - Dual may have nice structure: differentiable, sparse, coordinate optimization.
- Large-scale kernel methods is an active research area.
  - Special K structures, subsampling methods, explicit feature construction.

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

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

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 $-\lambda$,

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

Constructing Feature Space (Finite Domain)

- Why is positive semi-definiteness important?
  - With finite domain we can define K over all points.
  - By symmetry of K it has a spectral decomposition

$$K = U^\top \Lambda U,$$

    and $K \succeq 0$ means $\lambda_i \geq 0$ and so we have a real diagonal $\Lambda^{\frac{1}{2}}$.
  - Thus we hav $K = U^\top \Lambda^{\frac{1}{2}} \Lambda^{\frac{1}{2}} U = (\Lambda^{\frac{1}{2}} U)^\top (\Lambda^{\frac{1}{2}} U)$ and we could use

$$Z = \Lambda^{\frac{1}{2}} U, \text{ which means } z_i = \Lambda^{\frac{1}{2}} U_{:,i}.$$

- The above reasoning isn't quite right for continuous domains.
- The more careful generalization is known as "Mercer's theorem".

Fenchel Dual

- Lagrangian for constrained problem is

$$L(v, w, z) = f(v) + r(w) + z^\top (Xw - v),$$

so the dual function is

$$D(z) = \inf_{v,w} \{f(v) + r(w) + z^\top (Xw - v)\}$$

- For the $\inf$ wrt v we have

$$\inf_v \{f(v) - z^\top v\} = -\sup_v \{v^\top z - f(v)\} = -f^*(z).$$

- For the $\inf$ wrt w we have

$$\inf_w \{r(w) + z^\top Xw\} = -r^*(-X^\top z).$$

- This gives

$$D(z) = -f^*(z) - r^*(-X^\top z),$$

but we could alternately get this in terms of $-z$ by replacing $(Xw - v)$ with $(v - Xw)$ in the Lagrangian.