

CPSC 540: Machine Learning

Kernel Methods, Fenchel Duality

Mark Schmidt

University of British Columbia

Winter 2017

Admin

- **Assignment 2:**
 - Due February 6 (1 week).
 - Start early, use Piazza.
- **Office hours this week:**
 - I can't make it this Friday.
 - Move to Thursday or have it with TAs?

Last Few Lectures: Large-Scale Machine Learning Algorithms

- Consider optimization problem:

$$\operatorname{argmin}_{x \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^n f_i(x).$$

- Coordinate optimization: update one x_j based on all examples:
 - Fast convergence rate, but iterations must be d times cheaper than gradient method.
 - Functions f_i must be smooth (possibly with non-smooth but separable regularizer).

Last Few Lectures: Large-Scale Machine Learning Algorithms

- Consider optimization problem:

$$\operatorname{argmin}_{x \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^n f_i(x).$$

- Coordinate optimization:** update **one x_j** based on all examples:
 - Fast convergence rate, but **iterations must be d times cheaper** than gradient method.
 - Functions **f_i must be smooth** (possibly with non-smooth but separable regularizer).
- Stochastic gradient:** update **all x_j** based on one example:
 - Slow convergence rate**, and **iterations are n times cheaper** than gradient method.
 - Functions **f_i can be non-smooth** and **n can be infinite**.

Last Few Lectures: Large-Scale Machine Learning Algorithms

- Consider optimization problem:

$$\operatorname{argmin}_{x \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^n f_i(x).$$

- **Coordinate optimization**: update **one x_j** based on all examples:
 - Fast convergence rate, but **iterations must be d times cheaper** than gradient method.
 - Functions **f_i must be smooth** (possibly with non-smooth but separable regularizer).
- **Stochastic gradient**: update **all x_j** based on one example:
 - **Slow convergence rate**, and iterations are **n times cheaper** than gradient method.
 - Functions **f_i can be non-smooth** and **n can be infinite**.
- **SAG**: update **all x_j** based on one example (and old versions of others):
 - Fast convergence rate, and iterations are **n times cheaper** than gradient method.
 - Functions **f_i must be smooth** (possibly with “simple” non-smooth regularizer).

Last Time: Kernel Trick

- Alternative approach to L2-regularized least squares with features Z :

- 1 Derive non-linear features Z from X .
- 2 Compute $K = ZZ^T$ containing all **inner products** $z_i^T z_j$.
- 3 Fit model,

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

- 4 Use the model to make predictions,

$$\hat{y} = \underbrace{\hat{Z}Z^T}_{\hat{K}} v.$$

- This **assumes we can compute Z** .

Last Time: Kernel Trick

- Kernel trick for L2-regularized least squares with features Z :

- 1 Derive non-linear features Z from X .
- 2 Compute $K = ZZ^T$ containing all inner products $z_i^T z_j$.
- 3 Fit model,

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

- 4 Use the model to make predictions,

$$\hat{y} = \underbrace{\hat{Z}Z^T}_{\hat{K}} v.$$

- This assumes we can compute Z .

Last Time: Kernel Trick

- Kernel trick for L2-regularized least squares with features Z :

- ① (No need for explicit features Z)
- ② Compute $K = ZZ^T$ containing all inner products $z_i^T z_j$.
- ③ Fit model,

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

- ④ Use the model to make predictions,

$$\hat{y} = \underbrace{\hat{Z}Z^T}_{\hat{K}} v.$$

- This assumes we can compute Z .

Last Time: Kernel Trick

- Kernel trick for L2-regularized least squares with features Z :
 - ① (No need for explicit features Z)
 - ② Compute $K = ZZ^T$ containing all inner products $z_i^T z_j = k(x^i, x^j)$.
 - ③ Fit model,

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

- ④ Use the model to make predictions,

$$\hat{y} = \underbrace{\hat{Z}Z^T}_{\hat{K}} v.$$

- This assumes we can compute Z .

Last Time: Kernel Trick

- Kernel trick for L2-regularized least squares with features Z :
 - 1 (No need for explicit features Z)
 - 2 Compute $K = ZZ^T$ containing all inner products $z_i^T z_j = k(x^i, x^j)$.
 - 3 Fit model,

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

- 4 Use the model to make predictions,

$$\hat{y} = \underbrace{\hat{Z}Z^T}_{\hat{K}} v.$$

- This assumes we can compute Z .

Last Time: Kernel Trick

- Kernel trick for L2-regularized least squares with features Z :
 - ① (No need for explicit features Z)
 - ② Compute $K = ZZ^T$ containing all inner products $z_i^T z_j = k(x^i, x^j)$.
 - ③ Fit model,

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

- ④ Use the model to make predictions,

$$\hat{y} = \hat{K}v.$$

- This assumes we can compute Z .

Last Time: Kernel Trick

- Kernel trick for L2-regularized least squares with features Z :
 - 1 (No need for explicit features Z)
 - 2 Compute $K = ZZ^T$ containing all inner products $z_i^T z_j = k(x^i, x^j)$.
 - 3 Fit model,

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

- 4 Use the model to make predictions,

$$\hat{y} = \hat{K}v.$$

- This does not assume we can compute Z .
 - Allows exponential- or infinite-sized features.
 - Instead of features, work with “similarity” $k(x^i, x^j)$.
 - We'll define valid kernels today.

Kernels Trick for Distance-Based Methods

- Besides ridge regression, when can we apply the kernel trick?

Kernels Trick for Distance-Based Methods

- Besides ridge regression, when can we apply the kernel trick?
 - Distance-based methods from CPSC 340:

$$\begin{aligned}\|z_i - z_j\|^2 &= z_i^T z_i - 2z_i^T z_j + z_j^T z_j \\ &= k(x^i, x^i) - 2k(x^i, x^j) + k(x^j, x^j).\end{aligned}$$

Kernels Trick for Distance-Based Methods

- Besides ridge regression, when can we apply the kernel trick?
 - Distance-based methods from CPSC 340:

$$\begin{aligned}\|z_i - z_j\|^2 &= z_i^T z_i - 2z_i^T z_j + z_j^T z_j \\ &= k(x^i, x^i) - 2k(x^i, x^j) + k(x^j, x^j).\end{aligned}$$

- k -nearest neighbours.
- Clustering algorithms (k -means, density-based clustering, hierarchical clustering).
- Amazon item-to-item product recommendation.
- Non-parametric regression.
- Outlier ratio.
- Multi-dimensional scaling.
- Graph-based semi-supervised learning.

Kernels Trick for Distance-Based Methods

- Besides ridge regression, when can we apply the kernel trick?
 - Distance-based methods from CPSC 340:

$$\begin{aligned}\|z_i - z_j\|^2 &= z_i^T z_i - 2z_i^T z_j + z_j^T z_j \\ &= k(x^i, x^i) - 2k(x^i, x^j) + k(x^j, x^j).\end{aligned}$$

- k -nearest neighbours.
 - Clustering algorithms (k -means, density-based clustering, hierarchical clustering).
 - Amazon item-to-item product recommendation.
 - Non-parametric regression.
 - Outlier ratio.
 - Multi-dimensional scaling.
 - Graph-based semi-supervised learning.
- L2-regularized linear models (today).
- Eigenvalue methods:
 - Principle component analysis (need trick for centering in high-dimensional space).
 - Canonical correlation analysis.
 - Spectral clustering.

Outline

- 1 Valid Kernels and Representer Theorem
- 2 Fenchel Duality
- 3 Large-Scale Kernel Methods

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 ϕ such that $k(x^i, x^j) = \langle \phi(x^i), \phi(x^j) \rangle$.

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 ϕ 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 $K \succeq 0$.

- Bonus slide proves for finite domains, general case is called Mercer's Theorem.

Valid Kernels

- Mercer's Theorem is nice in theory, what do we do in practice?
 - Show explicitly that $k(x^i, x^j)$ is an inner product.
 - Show that K is positive semi-definite by construction.

Valid Kernels

- Mercer's Theorem is nice in theory, what do we do in practice?
 - Show explicitly that $k(x^i, x^j)$ is an inner product.
 - Show that K is positive semi-definite by construction.
 - Or show it can be 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 $\alpha \geq 0$.

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**: $k_1(x^i, x^j) + k_2(x^i, x^j)$.

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: $\phi(x^i)k_1(x^i, x^j)\phi(x^j)$.

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: $\phi(x^i)k_1(x^i, x^j)\phi(x^j)$.
 - **Exponentiation**: $\exp(k_1(x^i, x^j))$.
 - **Recursion**: $k_1(\phi(x^i), \phi(x^j))$.

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: $\phi(x^i)k_1(x^i, x^j)\phi(x^j)$.
 - **Exponentiation**: $\exp(k_1(x^i, x^j))$.
 - **Recursion**: $k_1(\phi(x^i), \phi(x^j))$.
- Example: Gaussian-RBF kernel:

$$\begin{aligned}
 k(x^i, x^j) &= \exp\left(-\frac{\|x^i - x^j\|^2}{2\sigma^2}\right) \\
 &= \underbrace{\exp\left(-\frac{\|x^i\|^2}{2\sigma^2}\right)}_{\phi(x^i)} \underbrace{\exp\left(\underbrace{\frac{1}{\sigma^2}}_{\alpha \geq 0} \underbrace{(x^i)^T x^j}_{\text{valid}}\right)}_{\exp(\text{valid})} \underbrace{\exp\left(-\frac{\|x^j\|^2}{2\sigma^2}\right)}_{\phi(x^j)}.
 \end{aligned}$$

Applicability of Kernel Tricks...

- Kernel trick **does not apply** to many problems.
 - L1-regularized least squares.
- But it works for **L2-regularized linear models**...

Representer Theorem

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

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

Representer Theorem

- Consider linear model with losses **differentiable** 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

$$0 = \sum_{i=1}^n \nabla f'_i(w^T x^i) x^i + \lambda w.$$

Representer Theorem

- Consider linear model with losses **differentiable** 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

$$0 = \sum_{i=1}^n \nabla f'_i(w^T x^i) x^i + \lambda w.$$

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

$$w^* = -\frac{1}{\lambda} \sum_{i=1}^n \nabla f_i((w^*)^T x^i) x^i = \sum_{i=1}^n v_i x^i$$

Representer Theorem

- Consider linear model with losses **differentiable** 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

$$0 = \sum_{i=1}^n \nabla f'_i(w^T x^i) x^i + \lambda w.$$

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

$$w^* = -\frac{1}{\lambda} \sum_{i=1}^n \nabla f_i((w^*)^T x^i) x^i = \sum_{i=1}^n v_i x^i = X^T v.$$

Representer Theorem

- Using representer theorem we can use $w = X^T v$ in original problem,

$$\begin{aligned} & \operatorname{argmin}_{w \in \mathbb{R}^d} \sum_{i=1}^n f_i(w^T x^i) + \frac{\lambda}{2} \|w\|^2 \\ &= \operatorname{argmin}_{v \in \mathbb{R}^n} \sum_{i=1}^n f_i(\underbrace{v^T X x^i}_{(x^i)^T X^T v}) + \frac{\lambda}{2} \|X^T v\|^2 \end{aligned}$$

Representer Theorem

- Using representer theorem we can use $w = X^T v$ in original problem,

$$\begin{aligned} & \operatorname{argmin}_{w \in \mathbb{R}^d} \sum_{i=1}^n f_i(w^T x^i) + \frac{\lambda}{2} \|w\|^2 \\ &= \operatorname{argmin}_{v \in \mathbb{R}^n} \sum_{i=1}^n f_i(\underbrace{v^T X x^i}_{(x^i)^T X^T v}) + \frac{\lambda}{2} \|X^T v\|^2 \end{aligned}$$

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

$$\equiv \operatorname{argmin}_{v \in \mathbb{R}^n} f(X X^T v) + \frac{\lambda}{2} v^T X X^T v$$

Representer Theorem

- Using representer theorem we can use $w = X^T v$ in original problem,

$$\begin{aligned} & \operatorname{argmin}_{w \in \mathbb{R}^d} \sum_{i=1}^n f_i(w^T x^i) + \frac{\lambda}{2} \|w\|^2 \\ &= \operatorname{argmin}_{v \in \mathbb{R}^n} \sum_{i=1}^n f_i(\underbrace{v^T X x^i}_{(x^i)^T X^T v}) + \frac{\lambda}{2} \|X^T v\|^2 \end{aligned}$$

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

$$\begin{aligned} & \equiv \operatorname{argmin}_{v \in \mathbb{R}^n} f(X X^T v) + \frac{\lambda}{2} v^T X X^T v \\ & \equiv \operatorname{argmin}_{v \in \mathbb{R}^n} f(K v) + \frac{\lambda}{2} v^T K v. \end{aligned}$$

Representer Theorem

- Using representer theorem we can use $w = X^T v$ in original problem,

$$\begin{aligned} & \operatorname{argmin}_{w \in \mathbb{R}^d} \sum_{i=1}^n f_i(w^T x^i) + \frac{\lambda}{2} \|w\|^2 \\ &= \operatorname{argmin}_{v \in \mathbb{R}^n} \sum_{i=1}^n f_i(\underbrace{v^T X x^i}_{(x^i)^T X^T v}) + \frac{\lambda}{2} \|X^T v\|^2 \end{aligned}$$

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

$$\begin{aligned} & \equiv \operatorname{argmin}_{v \in \mathbb{R}^n} f(X X^T v) + \frac{\lambda}{2} v^T X X^T v \\ & \equiv \operatorname{argmin}_{v \in \mathbb{R}^n} f(K v) + \frac{\lambda}{2} v^T K v. \end{aligned}$$

- Which is a kernelized version of the problem.

Representer Theorem

- Using the representation $w^* = X^T v$ for some v , our predictions are given by

$$\hat{y} = \hat{X} w^*$$

Representer Theorem

- Using the representation $w^* = X^T v$ for some v , our predictions are given by

$$\begin{aligned}\hat{y} &= \hat{X} w^* \\ &= \hat{X} X^T v\end{aligned}$$

Representer Theorem

- Using the representation $w^* = X^T v$ for some v , our predictions are given by

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

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

Representer Theorem

- Using the representation $w^* = X^T v$ for some v , our predictions are given by

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

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

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

Outline

- 1 Valid Kernels and Representer Theorem
- 2 Fenchel Duality
- 3 Large-Scale Kernel Methods

Motivation: Getting Rid of the Step-Size

- **SVMs** are a widely-used model but objective is **non-differentiable**.
 - We can't apply coordinate optimization or proximal-gradient or SAG.
 - The non-differentiable part is the loss, which isn't nice.

Motivation: Getting Rid of the Step-Size

- **SVMs** are a widely-used model but objective is **non-differentiable**.
 - We can't apply coordinate optimization or proximal-gradient or SAG.
 - The non-differentiable part is the loss, which isn't nice.
- Stochastic subgradient methods achieve $O(1/\epsilon)$ without dependence on n .
 - But **choosing the step-size is painful**.

Motivation: Getting Rid of the Step-Size

- SVMs are a widely-used model but objective is non-differentiable.
 - We can't apply coordinate optimization or proximal-gradient or SAG.
 - The non-differentiable part is the loss, which isn't nice.
- 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**,

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

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

$$L(x, v) = f(x) + z^T(Ax - b),$$

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

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

$$L(x, v) = f(x) + z^T(Ax - b),$$

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

- At a solution of the problem we must have

$$\nabla_x L(x, z) = \nabla f(x) + A^T z = 0 \quad (\text{gradient is orthogonal to constraints})$$

$$\nabla_z L(x, z) = Ax - b = 0 \quad (\text{constraints are satisfied})$$

- 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)

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**.

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**.

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:

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

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

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

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,

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

- 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,

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

(restricted to z where the max is finite)

Dual function

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

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

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

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

Dual function

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

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

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

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

- Why?????

Dual function

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

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

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

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

- Why?????

- If f is strongly-convex, dual is smooth (not obvious).

Dual function

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

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

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

(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 function

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

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

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

(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).

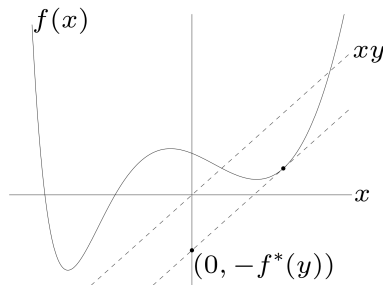
(see Michael Friedlander's class for details/conditions.)

Convex Conjugate

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

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

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

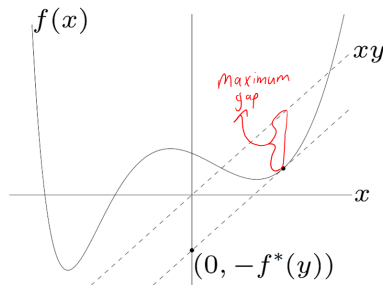


Convex Conjugate

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

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

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



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

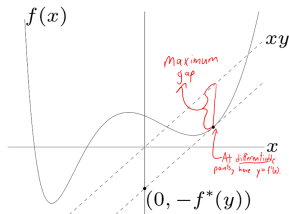
- It's the **maximum** that the linear function $y^T x$ can get above $f(x)$.

Convex Conjugate

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

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

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



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

- If f is differentiable, 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
 - $f^*(y) = \sup_x \{y^T x - \frac{1}{2}\|x\|^2\}$

Convex Conjugate Examples

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

$$0 = y - x,$$

and pluggin in $x = y$ we get

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

Convex Conjugate Examples

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

$$0 = y - x,$$

and pluggin in $x = y$ we get

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

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

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

- For other examples, see Boyd & Vandenberghe.

Fenchel Dual

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

$$\operatorname{argmin}_{w \in \mathbb{R}^d} f(Xw) + r(w).$$

- If we **introduce equality constraints**,

$$\operatorname{argmin}_{v=Xw} f(v) + r(w).$$

Fenchel Dual

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

$$\operatorname{argmin}_{w \in \mathbb{R}^d} f(Xw) + r(w).$$

- If we **introduce equality constraints**,

$$\operatorname{argmin}_{v=Xw} f(v) + r(w).$$

then dual has a special form called the **Fenchel dual**,

$$\operatorname{argmax}_{z \in \mathbb{R}^n} D(z) = -f^*(-z) - r^*(X^T z),$$

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,

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

- The **Fenchel dual** is given by

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

with $w^* = \frac{1}{\lambda} X^T Y z^*$ and constraints coming from $f^* < \infty$.

Fenchel Dual of SVMs

- Consider support vector machines,

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

- The **Fenchel dual** is given by

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

with $w^* = \frac{1}{\lambda} X^T Y z^*$ and constraints coming from $f^* < \infty$.

- A couple magical things have happened:
 - We can apply **kernel trick**.

Fenchel Dual of SVMs

- Consider support vector machines,

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

- The **Fenchel dual** is given by

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

with $w^* = \frac{1}{\lambda} X^T Y z^*$ 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.

Fenchel Dual of SVMs

- Consider support vector machines,

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

- The **Fenchel dual** is given by

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

with $w^* = \frac{1}{\lambda} X^T Y z^*$ 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,

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

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

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

Stochastic Dual Coordinate Ascent

- If we have an L2-regularized linear model,

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

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

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

- It's known as **stochastic dual coordinate ascent (SDCA)**:
 - Only needs to look at one training example on each iteration.

Stochastic Dual Coordinate Ascent

- If we have an L2-regularized linear model,

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

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

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

- It's known as **stochastic dual coordinate ascent (SDCA)**:
 - Only needs to look 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 $\mu \gg \lambda$.

Stochastic Dual Coordinate Ascent

- If we have an L2-regularized linear model,

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

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

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

- It's known as **stochastic dual coordinate ascent (SDCA)**:
 - Only needs to look 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 $\mu \gg \lambda$.
 - Obtains $O(1/\epsilon)$ rate for non-smooth f :
 - Same rate/cost as stochastic subgradient, but we can **use exact/adaptive step-size**.

Outline

- 1 Valid Kernels and Representer Theorem
- 2 Fenchel Duality
- 3 Large-Scale Kernel Methods**

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.

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

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

Kernels with Special Structure

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

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

- 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 $O(n)$.

Kernels with Special Structure

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

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

- 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).

Kernels with Special Structure

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

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

- 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, $D + UV^T$.
 - 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.

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 .

Subsampling Methods

- Consider partitioning our matrices as

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

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

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

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

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

- 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 $Z^T Z \approx K$.
 - But where Z has a small number of columns of m .
- We then use our non-kernelized approach with features Z ,

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

Explicit Feature Construction

- In **explicit feature** methods, we form Z such that $Z^T Z \approx K$.
 - But where Z has a small number of columns of m .

- We then use our non-kernelized approach with features Z ,

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

- **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

- Valid kernels are typically constructed from other valid kernels.

Summary

- Valid kernels are typically constructed from other valid kernels.
- Representer theorem allows kernel trick for L2-regularized linear models.

Summary

- 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.

Summary

- 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.
- Next time: we start unsupervised learning.

Bonus Slide: Constructing Feature Space (Finite Domain)

- Why is positive semi-definiteness important?
 - With finite domain we can define K over all points.
 - The condition $K \succeq 0$ means it has a spectral decomposition

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

where the eigenvalues $\lambda_i \geq 0$ and so we have a real $\Lambda^{\frac{1}{2}}$.

- Thus we have $K = U^T \Lambda^{\frac{1}{2}} \Lambda^{\frac{1}{2}} U = \|\Lambda^{\frac{1}{2}} U\|^2$ and we could use

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

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

Bonus Slide: Fenchel Dual

- Lagrangian for constrained problem is

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

so the dual function is

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

- For the inf wrt v we have

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

- For the inf wrt w we have

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

- This gives

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

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