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?
Consider optimization problem:

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

- **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 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,
   $$v = (ZZ^T + \lambda I)^{-1} y,$$
4. Use the model to make predictions,
   $$\hat{y} = \hat{Z}Z^T v.$$

This assumes 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.
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?

Distance-based methods from CPSC 340:

\[ \| z_i - z_j \|^2 = z_i^T z_i - 2z_i^T z_j + z_j^T z_j = k(x^i, x^i) - 2k(x^i, x^j) + k(x^j, x^j). \]

- \( k \)-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 $\phi$ 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 $\{x^1, x^2, \ldots, x^n\}$ 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.
  - Or show it can be constructed from other valid kernels.

(If we use invalid kernel, lose feature-space interpretation but may work fine.)
Valid Kernels and Representer Theorem Fenchel Duality Large-Scale Kernel Methods

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:

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

$$= \exp\left(-\frac{\|x^i\|^2}{2\sigma^2}\right) \exp\left(\frac{1}{\sigma^2} (x^i)^T x^j\right) \exp\left(-\frac{\|x^j\|^2}{2\sigma^2}\right).$$
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,

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

- Setting the gradient equal to zero we get

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

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

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

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

\[
\arg\min_{w \in \mathbb{R}^d} \sum_{i=1}^{n} f_i(w^T x^i) + \frac{\lambda}{2} \| w \|^2 \\
= \arg\min_{v \in \mathbb{R}^n} \sum_{i=1}^{n} f_i \left( \underbrace{v^T X x^i}_{(x^i)^T X^T v} \right) + \frac{\lambda}{2} \| X^T v \|^2
\]

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

\[
\equiv \arg\min_{v \in \mathbb{R}^n} f(X X^T v) + \frac{\lambda}{2} v^T X X^T v \\
\equiv \arg\min_{v \in \mathbb{R}^n} f(Kv) + \frac{\lambda}{2} v^T K v.
\]

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

$$\hat{y} = \hat{X} w^* = \hat{X} X^T v = \hat{K} v,$$

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.

- 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...
Consider minimizing a differentiable \( f \) with linear equality constraints, 

\[
\arg\min_{Ax=b} f(x).
\]

The Lagrangian of this problem is defined by

\[
L(x, z) = f(x) + z^T(Ax - b),
\]

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)

- 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,

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

- The analogy for max is called the supremum (sup).
Even for non-smooth convex $f$ solution is a saddle point of the Lagrangian,

$$\max_z \inf_x \left( f(x) + z^T (Ax - b) \right).$$

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

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

  where $X$ is values where $\sup$ is finite.

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

  [Diagram showing the convex conjugate relationship with a graph with points $(0, -f^*(y))$ and $f(x)$ on the axes.]
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.

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

    and plugging 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 $r$)
  \[
  \text{argmin}_{w \in \mathbb{R}^d} f(Xw) + r(w).
  \]

- If we introduce equality constraints,
  \[
  \text{argmin}_{v=Xw} f(v) + r(w).
  \]
  
  then dual has a special form called the **Fenchel dual**, 
  \[
  \text{argmax}_{z \in \mathbb{R}^n} D(z) = -f^*(-z) - r^*(X^Tz),
  \]
  
  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,

\[
\arg\min_{w \in \mathbb{R}^d} \sum_{i=1}^n \max\{0, 1 - y_i w^T x_i\} + \frac{\lambda}{2} \|w\|^2.
\]

- The Fenchel dual is given by

\[
\arg\max_{0 \leq z \leq 1} \sum_{i=1}^n z_i - \frac{1}{2\lambda} \left\| X^T Y z \right\|^2,
\]

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,
  \[
  \arg\min_{w \in \mathbb{R}^d} \sum_{i=1}^{n} f_i(w^T x_i) + \frac{\lambda}{2} \|w\|^2,
  \]
  then Fenchel dual is a problem where we can apply coordinate optimization,
  \[
  \arg\max_{z \in \mathbb{R}^n} - \sum_{i=1}^{n} f_i^*(z_i) - \frac{1}{2\lambda} \left\| X^T z \right\|^2.
  \]
  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 $\nabla f_i$ 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
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, $D + UV^T$.
  - 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.

- **Nyström 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

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

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

\[
\hat{y} = \hat{K}_1 \left( K_1^T K_1 + \lambda K_{11} \right)^{-1} K_1^T y.
\]

Given \(K_1\) and \(K_{11}\), computing \(v\) costs \(O(m^2n + m^3)\) 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 $Z$,
  $$w = (Z^T Z + \lambda I)^{-1}(Z^T y).$$

- **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.
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.
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”.

The above reasoning isn’t quite right for continuous domains.
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.