Outline

- Kernel Methods
- Solving the dual problem
- Kernel approximation
Support Vector Machines (SVM)

- SVM is a widely used classifier.

Given:
- Training data points $x_1, \ldots, x_n$.
- Each $x_i \in \mathbb{R}^d$ is a feature vector:
- Consider a simple case with two classes: $y_i \in \{+1, -1\}$.

Goal: Find a hyperplane to separate these two classes of data:
if $y_i = 1$, $w^T x_i \geq 1 - \xi_i$; $y_i = -1$, $w^T x_i \leq -1 + \xi_i$. 

![Diagram showing SVM classification with hyperplane and support vectors](image)
Support Vector Machines (SVM)

- Given training data $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^d$ with labels $y_i \in \{+1, -1\}$.
- SVM primal problem (find optimal $\mathbf{w} \in \mathbb{R}^d$):
  \[
  \min_{\mathbf{w}, \xi} \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_{i=1}^{n} \xi_i \\
  \text{s.t. } y_i(\mathbf{w}^T \mathbf{x}_i) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad i = 1, \ldots, n,
  \]
- SVM dual problem (find optimal $\alpha \in \mathbb{R}^n$):
  \[
  \min_{\alpha} \frac{1}{2} \alpha^T Q \alpha - \mathbf{e}^T \alpha \\
  \text{s.t. } 0 \leq \alpha_i \leq C, \quad \forall i = 1, \ldots, n
  \]
- $Q$: $n$-by-$n$ kernel matrix, $Q_{ij} = y_i y_j \mathbf{x}_i^T \mathbf{x}_j$
- Each $\alpha_i$ corresponds to one training data point.
- Primal-dual relationship: $\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i$
Non-linearly separable problems

- What if the data is not linearly separable?

\[
x \rightarrow \varphi(x) = \begin{bmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{bmatrix}
\]

**Solution:** map data \( x_i \) to higher dimensional (maybe infinite) feature space \( \varphi(x_i) \), where they are linearly separable.
Support Vector Machines (SVM)

- **SVM primal problem:**

\[
\begin{align*}
\min \limits_{w, \xi} & \quad \frac{1}{2} w^T w + C \sum_{i=1}^{n} \xi_i \\
\text{s.t.} & \quad y_i(w^T \varphi(x_i)) \geq 1 - \xi_i, \quad \xi_i \geq 0, \quad i = 1, \ldots, n,
\end{align*}
\]

- **The dual problem for SVM:**

\[
\begin{align*}
\min \limits_{\alpha} & \quad \frac{1}{2} \alpha^T Q \alpha - e^T \alpha, \\
\text{s.t.} & \quad 0 \leq \alpha_i \leq C, \quad \text{for } i = 1, \ldots, n,
\end{align*}
\]

where \( Q_{ij} = y_i y_j \varphi(x_i)^T \varphi(x_j) \) and \( e = [1, \ldots, 1]^T \).

- **Kernel trick:** define \( K(x_i, x_j) = \varphi(x_i)^T \varphi(x_j) \).

- **At optimum:** \( w = \sum_{i=1}^{n} \alpha_i y_i \varphi(x_i) \),
Various types of kernels

- Gaussian kernel: \( K(x_i, y_j) = e^{-\gamma\|x_i - x_j\|^2_2} \);
- Polynomial kernel: \( K(x_i, x_j) = (\gamma x_i^T x_j + c)^d \).
- Other kernels for specific problems:
  - Graph kernels
    (Vishwanathan et al., “Graph Kernels”, JMLR, 2010)
  - Pyramid kernel for image matching
    (Grauman and Darrell, “The Pyramid Match Kernel: Discriminative Classification with Sets of Image Features”. In ICCV, 2005)
  - String kernel
General Kernelized ERM

- L2-Regularized Empirical Risk Minimization:

\[
\min_w \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{n} \ell_i(w^T \varphi(x_i))
\]

\(x_1, \ldots, x_n\): training samples

\(\varphi(x_i)\): nonlinear mapping to a higher dimensional space

- Dual problem:

\[
\min_\alpha \frac{1}{2} \alpha^T Q \alpha + \sum_{i=1}^{n} \ell^*_i (-\alpha_i)
\]

where \(Q \in \mathbb{R}^{n \times n}\) and \(Q_{ij} = \varphi(x_i)^T \varphi(x_j) = K(x_i, x_j)\).
Given training samples \((x_i, y_i), i = 1, \cdots, n\).

\[
\min_w \frac{1}{2}\|w\|^2 + \frac{1}{2} \sum_{i=1}^{n} (w^T \varphi(x_i) - y_i)^2
\]

Dual problem:

\[
\min_\alpha \alpha^T Q\alpha + \|\alpha\|^2 - 2\alpha^T y
\]
Scalability

Challenge for solving kernel SVMs (for dataset with $n$ samples):

- **Space:** $O(n^2)$ for storing the $n$-by-$n$ kernel matrix (can be reduced in some cases);
- **Time:** $O(n^3)$ for computing the exact solution.
Greedy Coordinate Descent for Kernel SVM
Nonlinear SVM

- SVM primal problem:
  \[
  \min_{w, \xi} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \xi_i \\
  \text{s.t. } y_i (w^T \varphi(x_i)) \geq 1 - \xi_i, \; \xi_i \geq 0, \; i = 1, \ldots, n,
  \]
  \(w\): an infinite dimensional vector, very hard to solve

- The dual problem for SVM:
  \[
  \min_{\alpha} \frac{1}{2} \alpha^T Q \alpha - e^T \alpha, \\
  \text{s.t. } 0 \leq \alpha_i \leq C, \; \text{for } i = 1, \ldots, n,
  \]
  where \(Q_{ij} = y_i y_j \varphi(x_i)^T \varphi(x_j)\) and \(e = [1, \ldots, 1]^T\).

- Kernel trick: define \(K(x_i, x_j) = \varphi(x_i)^T \varphi(x_j)\).

- Example: Gaussian kernel \(K(x_i, x_j) = e^{-\gamma \|x_i - x_j\|^2}\)
Nonlinear SVM

- Can we solve the problem by dual coordinate descent?
- The vector \( w = \sum_i y_i \alpha_i \varphi(x_i) \) may have infinite dimensionality: Cannot maintain \( w \)
- Closed form solution needs \( O(n) \) computational time:
  \[
  \delta^* = \max(-\alpha_i \min(C - \alpha_i, \frac{1 - (Q\alpha)_i}{Q_{ii}}))
  \]
  (Assume \( Q \) is stored in memory)
- Can we improve coordinate descent using the same \( O(n) \) time complexity?
The Greedy Coordinate Descent (GCD) algorithm:

For $t = 1, 2, \ldots$

1. Compute $\delta_i^* := \arg\min_\delta f(\alpha + \delta e_i)$ for all $i = 1, \ldots, n$

2. Find the best $i^*$ according to the following criterion:

   $$i^* = \arg\max_i |\delta_i^*|$$

3. $\alpha_{i^*} \leftarrow \alpha_{i^*} + \delta_{i^*}$
Greedy Coordinate Descent

Other variable selection criterion:

- The coordinate with the maximum step size:

\[ i^* = \arg\max_i |\delta_i^*| \]

- The coordinate with maximum objective function reduction:

\[ i^* = \arg\max_i (f(\alpha) - f(\alpha + \delta_i^* e_i)) \]

- The coordinate with the maximum projected gradient.

...
Greedy Coordinate Descent

- How to compute the optimal coordinate?
  - Closed form solution of best $\delta$:
    \[
    \delta_i^* = \max \left( -\alpha_i, \min \left( C - \alpha_i, \frac{1 - (Q\alpha)_i}{Q_{ii}} \right) \right)
    \]

- Observations:
  1. Computing all $\delta_i^*$ needs $O(n^2)$ time
  2. If $Q\alpha$ is stored in memory, computing all $\delta_i^*$ only needs $O(n)$ time
  3. Maintaining $Q\alpha$ also needs $O(n)$ time after each update
Greedy Coordinate Descent

Initial: \( \alpha, z = Q\alpha \)

For \( t = 1, 2, \ldots \)

For all \( i = 1, \ldots, n \), compute

\[
\delta_i^* = \max \left( -\alpha_i, \min \left( C - \alpha_i, \frac{1 - z_i}{Q_{ii}} \right) \right)
\]

Let \( i^* = \arg\max_i |\delta_i^*| \)

\[
\alpha \leftarrow \alpha + \delta_{i^*}
\]

\[
z \leftarrow z + q_{i^*} \delta_{i^*}^* \quad (q_{i^*} \text{ is the } i^*-\text{th column of } Q)
\]

(This is a simplified version of the Sequential Minimal Optimization (SMO) algorithm proposed in Platt el al., 1998)

(A similar version is implemented in LIBSVM)
How to solve problems with millions of samples?

- $Q \in \mathbb{R}^{n \times n}$ cannot be fully stored
- Have a fixed size of memory to “cache” the computed columns of $Q$
- For each coordinate update:
  - If $q_i$ is in memory, directly use it to update
  - If $q_i$ is not in memory
    1. Kick out the “Least Recent Used” column
    2. Recompute $q_i$ and store it in memory
    3. Update $\alpha_i$
Implemented in LIBSVM:

https://www.csie.ntu.edu.tw/~cjlin/libsvm/

Other functionalities:

- Multi-class classification
- Support vector regression
- Cross-validation
Kernel Approximation
Kernel Approximation

- Kernel methods are hard to scale up because
  Kernel matrix $G$ is an $n$-by-$n$ matrix
- Can we approximate the kernel using a low-rank representation?
- Two main algorithms:
  - Nystrom approximation: Approximate the kernel matrix
  - Random Features: Approximate the kernel function
Kernel Matrix Approximation

- We want to form a low rank approximation of $G \in \mathbb{R}^{n \times n}$, where $G_{ij} = K(x_i, x_j)$

- Can we do SVD?
Kernel Matrix Approximation

- We want to form a low rank approximation of $G \in \mathbb{R}^{n \times n}$, where $G_{ij} = K(x_i, x_j)$

- Can we do SVD?
  No, SVD needs to form the $n$-by-$n$ matrix $O(n^2)$ space and $O(n^3)$ time
Kernel Matrix Approximation

We want to form a low rank approximation of $G \in \mathbb{R}^{n \times n}$, where $G_{ij} = K(x_i, x_j)$

Can we do SVD?

No, SVD needs to form the $n$-by-$n$ matrix $O(n^2)$ space and $O(n^3)$ time

Can we do matrix completion or matrix sketching?
Kernel Matrix Approximation

- We want to form a low rank approximation of $G \in \mathbb{R}^{n \times n}$, where $G_{ij} = K(x_i, x_j)$

- Can we do SVD?
  - No, SVD needs to form the $n$-by-$n$ matrix
    $O(n^2)$ space and $O(n^3)$ time

- Can we do matrix completion or matrix sketching?
  - Main problem: need to generalize to new points
Nystrom Approximation

- Nystrom approximation:
  - Randomly sample $m$ columns of $G$:

  $$G = \begin{bmatrix} W & G_{12} \\ G_{21} & G_{22} \end{bmatrix}$$

  - $W$: $m$-by-$m$ square matrix (observed)
  - $G_{21}$: $(n - m)$-by-$m$ matrix (observed)
  - $G_{12} = G_{21}^T$ (observed)
  - $G_{22}$: $(n - m)$-by-$(n - m)$ matrix (not observed)

- Form a kernel approximation based on these $mn$ elements

  $$G \approx \tilde{G} = \begin{bmatrix} W \\ G_{21} \end{bmatrix} W^\dagger \begin{bmatrix} W & G_{12} \end{bmatrix}$$

  - $W^\dagger$: pseudo-inverse of $W$
Nystrom Approximation

Why $W^\dagger$?

Exact recover the top left $m$-by-$m$ matrix

The kernel approximation:

$$
\begin{bmatrix}
W & G_{12} \\
G_{21} & G_{22}
\end{bmatrix} \approx
\begin{bmatrix}
W \\
G_{21}
\end{bmatrix}
W^\dagger
\begin{bmatrix}
W & G_{12} \\
G_{21} & G_{22}
\end{bmatrix}
= \begin{bmatrix}
W & G_{12} \\
G_{21} & G_{21}W^\dagger G_{12}
\end{bmatrix}
$$
Nystrom Approximation

Algorithm:
1. Sample $m$ “landmark points”: $v_1, \cdots, v_m$
2. Compute the kernel values between all training data to landmark points:

$$C = \begin{bmatrix} W \\ G_{21} \end{bmatrix} = \begin{bmatrix} K(x_1, v_1) & \cdots & K(x_1, v_m) \\ K(x_2, v_1) & \cdots & K(x_2, v_m) \\ \vdots & \vdots & \vdots \\ K(x_n, v_1) & \cdots & K(x_n, v_m) \end{bmatrix}$$

3. Form the kernel approximation $\tilde{G} = CW^\dagger C^T$
   (No need to explicitly form the $n$-by-$n$ matrix)

- Time complexity: $mn$ kernel evaluations ($O(mnd)$ time if use classical kernels)
- How to choose $m$?
  Trade-off between accuracy v.s computational time and memory space
Nystrom Approximation: Training

- Solve the dual problem using low-rank representation.
- Example: Kernel ridge regression

\[
\min_{\alpha} \alpha^T \tilde{G} \alpha + \lambda \|\alpha\|^2 - y^T \alpha
\]

Solve a linear system \((\tilde{G} + \lambda I)\alpha = y\)

- Use iterative methods (such as CG), with fast matrix-vector multiplication

\[
(\tilde{G} + \lambda I)p = CW^\dagger C^T p + \lambda p
\]

\(O(nm)\) time complexity per iteration
Another approach: reduce to linear ERM problems!

Rewrite as

\[ \tilde{G} = CW^\dagger C^T = C(W^\dagger)^{1/2}(W^\dagger)^{1/2}C^T \]

So the approximate kernel can be represented by linear kernel with feature matrix \( C(W^\dagger)^{1/2} \):

\[ \tilde{K}(x_i, x_j) = \tilde{G}_{ij} = u_i^T u_j, \]

where \( u_j = (W^\dagger)^{1/2} \begin{bmatrix} K(x_j, v_1) \\ \vdots \\ K(x_j, v_m) \end{bmatrix} \)

The problem is equivalent to a linear models with features \( u_1, \ldots, u_n \in \mathbb{R}^m \)

- Kernel SVM \( \Rightarrow \) Linear SVM with \( m \) features
- Kernel Ridge Regression \( \Rightarrow \) Linear Ridge regression with \( m \) features
Nystrom Approximation: Prediction

- Given the dual solution $\alpha$.
- Prediction for testing sample $x$:

$$\sum_{i=1}^{n} \alpha_i \tilde{K}(x_i, x) \quad \text{or} \quad \sum_{i=1}^{n} \alpha_i K(x_i, x) ?$$
Nystrom Approximation: Prediction

- Given the dual solution $\alpha$.
- Prediction for testing sample $x$:

$$\sum_{i=1}^{n} \alpha_i \tilde{K}(x_i, x)$$

- Need to use approximate kernel instead of original kernel!
  - Approximate kernel gives much better accuracy!
  - Because training & testing should use the same kernel.

- Generalization bound:
  - (Alaoui and Mahoney, “Fast Randomized Kernel Methods With Statistical Guarantees”. 2014. )
  - (Rudi et al., “Less is More: Nystrom Computational Regularization”. NIPS 2015. )
  - (using small number of landmark points can be viewed as a regularization)
Nystrom Approximation: Prediction

- How to define $\tilde{K}(x, x_i)$?

$$\tilde{G} = \begin{bmatrix} K(x_1, v_1) & \cdots & K(x_1, v_m) \\ \vdots & \vdots & \vdots \\ K(x_n, v_1) & \cdots & K(x_1, v_m) \\ K(x, v_1) & \cdots & K(x_1, v_m) \end{bmatrix} W^\dagger \begin{bmatrix} K(v_1, x_1) & \cdots & K(v_1, x) \\ \vdots & \vdots & \vdots \\ K(v_m, x_1) & \cdots & K(v_m, x) \end{bmatrix}$$

- Compute $u = (W^\dagger)^{1/2}$ (need $m$ kernel evaluations)

- Approximate kernel can be defined by

$$\tilde{K}(x, x_i) = u^T u_i$$
Nystrom Approximation: Prediction

- Prediction:

\[ f(x) = \sum_{i=1}^{n} \alpha_i u^T u_i = u^T (\sum_{i=1}^{n} \alpha_i u_i) \]

\((\sum_{i=1}^{n} \alpha_i u_i)\) can be pre-computed

\(\Rightarrow\) prediction time is \(m\) kernel evaluations

- Original kernel method: need \(n\) kernel evaluations for prediction.

- Summary:

<table>
<thead>
<tr>
<th></th>
<th>Quality</th>
<th>Training time</th>
<th>Prediction time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original kernel</td>
<td>exact</td>
<td>(n^{1.5}) kernel + kernel training</td>
<td>(n) kernel</td>
</tr>
<tr>
<td>Nystrom ((m))</td>
<td>rank (m)</td>
<td>(nm) kernel + linear training</td>
<td>(m) kernel</td>
</tr>
</tbody>
</table>
Nystrom Approximation

- How to select landmark points \((v_1, \cdots, v_m)\)
  
  - Traditional approach: uniform random sampling from training data
  
  - Importance sampling (leverage score):  
    
    
    - Gittens and Mahoney “Revisiting the Nystrom method for improved large-scale machine learning”. ICML 2013

  - **Kmeans sampling (performs extremely well)**:
    Run kmeans clustering and set \(v_1, \cdots, v_m\) to be cluster centers
    

  - **Subspace distance**:
    
Nystrom Approximation (other related papers)

- Distributed setting: (Kumar et al., “Ensemble Nystrom method”. NIPS 2009).
- Block diagonal + low-rank approximation: (Si et al., “Memory efficient kernel approximation”. ICML 2014.)
- Nystrom method for fast prediction: (Hsieh et al., “Fast prediction for large-scale kernel machines.” NIPS, 2014.)
- Structured landmark points: (Si et al., “Computational Efficient Nystrom Approximation using Fast Transforms”. ICML 2016.)
Kernel Approximation (random features)
Random Features

- Directly approximation the kernel function
  (Data-independent)
- Generate nonlinear “features”
  \[ \Rightarrow \text{reduce the problem to linear model training} \]
- Several examples:
  - Random Fourier Features:
    (Rahimi and Recht, “Random features for large-scale kernel machines”. NIPS 2007)
  - Random features for polynomial kernel:
    (Kar and Karnick, “Random feature maps for dot product kernels”. AISTATS 2012.)
Random Fourier Features

- Random features for “Shift-invariant kernel”:
  A continuous kernel is shift-invariant if $K(x, y) = k(x - y)$.
  - Gaussian kernel: $K(x, y) = e^{-\|x-y\|^2}$
  - Laplacian kernel: $K(x, y) = e^{-\|x-y\|_1}$
- Shift invariant kernel (if positive definite) can be written as:

$$k(x - y) = \int_{\mathbb{R}^d} P(w) e^{jw^T(x-y)} dw$$

for some probability distribution $P(w)$. 
Random Fourier Features

- Taking the real part we have

\[ k(x - y) = \int_{\mathbb{R}^d} P(w) \cos(w^T x - w^T y) \]
\[ = \int_{\mathbb{R}^d} P(w)(\cos(w^T x) \cos(w^T y) + \sin(w^T x) \sin(w^T y)) \]
\[ = E_{w \sim P(w)}[z_w(x)^T z_w(y)] \]

where \( z_w(x)^T = [\cos(w^T x), \sin(w^T x)] \).

- \( z_w(x)^T z_w(y) \) is an unbiased estimator of \( K(x, y) \) if \( w \) is sampled from \( P(\cdot) \).
Random Fourier Features

- Sample $m$ vectors $\mathbf{w}_1, \cdots, \mathbf{w}_m$ from $P(\cdot)$ distribution
- Generate random features for each sample:

\[
\mathbf{u}_i = \begin{bmatrix}
\sin(\mathbf{w}_1^T \mathbf{x}_i) \\
\cos(\mathbf{w}_1^T \mathbf{x}_i) \\
\sin(\mathbf{w}_2^T \mathbf{x}_i) \\
\cos(\mathbf{w}_2^T \mathbf{x}_i) \\
\vdots \\
\sin(\mathbf{w}_m^T \mathbf{x}_i) \\
\cos(\mathbf{w}_m^T \mathbf{x}_i)
\end{bmatrix}
\]

- $K(\mathbf{x}_i, \mathbf{x}_j) \approx \mathbf{u}_i^T \mathbf{u}_j$
  (larger $m$ leads to better approximation)
Random Features

\[ G \approx UU^T, \text{ where } U = \begin{bmatrix} u_1^T \\ \vdots \\ u_n^T \end{bmatrix} \]

- Rank \(2m\) approximation.
- The problem can be reduced to linear classification/regression for \(u_1, \ldots, u_n\).
- Time complexity: \(O(nmd)\) time + linear training
  (Nystrom: \(O(nmd + m^3)\) time + linear training)
- Prediction time: \(O(md)\) per sample
  (Nystrom: \(O(md)\) per sample)
Fastfood

- Random Fourier features require $O(md)$ time to generate $m$ features:
  Main bottleneck: Computing $w_i^T x$ for all $i = 1, \ldots, m$
- Can we compute this in $O(m \log d)$ time?
Fastfood

- Random Fourier features require $O(md)$ time to generate $m$ features:
  - Main bottleneck: Computing $\mathbf{w}_i^T \mathbf{x}$ for all $i = 1, \ldots, m$
- Can we compute this in $O(m \log d)$ time?
- Fastfood: proposed in (Le et al., “Fastfood Approximating Kernel Expansions in Loglinear Time”. ICML 2013. )
- Reduce time by using “structured random features”
Fastfood

- Tool: fast matrix-vector multiplication for Hadamard matrix
  \((H_d \in \mathbb{R}^{d \times d})\)

  \(H_d x\) requires \(O(d \log d)\) time

- Hadamard matrix:

  \[
  H_1 = \begin{bmatrix} 1 \end{bmatrix}
  \]

  \[
  H_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}
  \]

  \[
  H_{2k} = \begin{bmatrix} H_{2k-1} & H_{2k-1} \\ H_{2k-1} & -H_{2k-1} \end{bmatrix}
  \]

- \(H_d x\) can be computed efficiently by Fast Hadamard Transform
  (dynamic programming)
Fastfood

- Sample $m = d$ random features by

$$W = [H_d] = \begin{bmatrix} v_1 & 0 & \cdots & 0 \\ 0 & v_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & v_d \end{bmatrix}$$

So $Wx$ only needs $O(d \log d)$ time.

- Instead of sampling $W$ ($d^2$ elements), we just sample $v_1, \cdots, v_d$ from $N(0, \sigma)$

- Each row of $W$ is still $N(0, \sigma)$ (but rows are not independent)

- Faster computation, but performance is slightly worse than original random features.
Extensions

- **Fastfood: structured random features to improve computational speed:**
  (Le et al., “Fastfood Approximating Kernel Expansions in Loglinear Time”. ICML 2013.)

- **Structured landmark points for Nystrom approximation:**
  (Si et al., “Computational Efficient Nystrom Approximation using Fast Transforms”. ICML 2016.)

- **Doubly stochastic gradient with random features:**

- **Generalization bound (?)**
Coming up

- Choose the paper for presentation (due this Sunday, Oct 23).

Questions?