ECS289: Scalable Machine Learning

Cho-Jui Hsieh
UC Davis

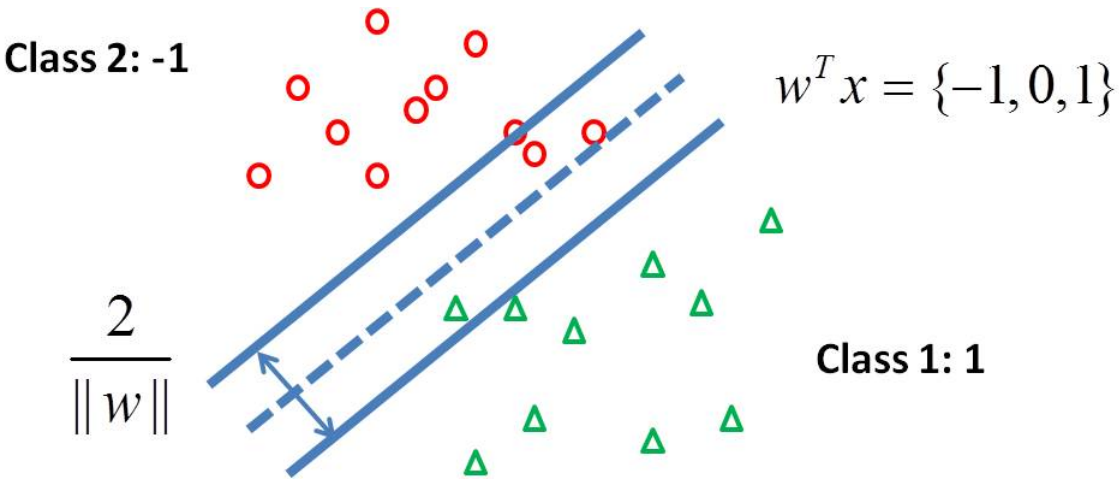
Oct 18, 2016

Outline

- Kernel Methods
- Solving the dual problem
- Kernel approximation

Support Vector Machines (SVM)

- SVM is a widely used classifier.
- Given:
  - Training data points $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n$.
  - Each $\boldsymbol{x}_i \in \mathbb{R}^d$ 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 $y_i = 1$, $\boldsymbol{w}^T \boldsymbol{x}_i \geq 1 - \xi_i$; $y_i = -1$, $\boldsymbol{w}^T \boldsymbol{x}_i \leq -1 + \xi_i$.



Support Vector Machines (SVM)

- Given training data $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n \in \mathbb{R}^d$ with labels $y_i \in \{+1, -1\}$.
- SVM primal problem (find optimal $\boldsymbol{w} \in \mathbb{R}^d$):

$$\min_{\boldsymbol{w},\xi} \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{n}\xi_i$$
$$\text{s.t. } y_i(\boldsymbol{w}^T\boldsymbol{x}_i) \geq 1 - \xi_i,\ \xi_i \geq 0, i = 1, \ldots, n,$$

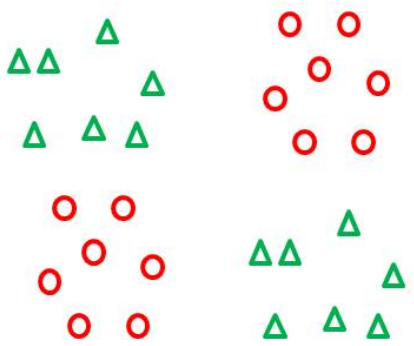
- SVM dual problem (find optimal $\boldsymbol{\alpha} \in \mathbb{R}^n$):

$$\min_{\boldsymbol{\alpha}} \frac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} - \boldsymbol{e}^T\boldsymbol{\alpha}$$
$$\text{s.t.} 0 \leq \alpha_i \leq C, \quad \forall i = 1, \ldots, n$$

  - Q: n-by-n kernel matrix, $Q_{ij} = y_i y_j \boldsymbol{x}_i^T\boldsymbol{x}_j$
  - Each α_i corresponds to one training data point.
  - Primal-dual relationship: $\boldsymbol{w} = \sum_i \alpha_i y_i \boldsymbol{x}_i$

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 $\boldsymbol{x}_i$ to higher dimensional(maybe infinite) feature space $\varphi(x_i)$, where they are linearly separable.

Support Vector Machines (SVM)

- SVM primal problem:

$$\begin{aligned} \min_{\boldsymbol{w},\xi} \ & \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{n}\xi_i \\ \text{s.t. } & y_i(\boldsymbol{w}^T\varphi(\boldsymbol{x}_i)) \geq 1-\xi_i, \quad \xi_i \geq 0, \ i = 1,\ldots,n, \end{aligned}$$

- The dual problem for SVM:

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \ & \frac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} - \boldsymbol{e}^T\boldsymbol{\alpha}, \\ \text{s.t. } & 0 \leq \alpha_i \leq C, \ \text{ for } \ i = 1,\ldots,n, \end{aligned}$$

  where $Q_{ij} = y_i y_j \varphi(\boldsymbol{x}_i)^T\varphi(\boldsymbol{x}_j)$ and $e = [1,\ldots,1]^T$.
- Kernel trick: define $K(\boldsymbol{x}_i,\boldsymbol{x}_j) = \varphi(\boldsymbol{x}_i)^T\varphi(\boldsymbol{x}_j)$.
- At optimum: $\boldsymbol{w} = \sum_{i=1}^{n}\alpha_i y_i \varphi(\boldsymbol{x}_i)$,

Various types of kernels

- Gaussian kernel: $K(\boldsymbol{x}_i, \mathbf{y}_j) = e^{-\gamma \|\boldsymbol{x}_i - \boldsymbol{x}_j\|_2^2}$;
- Polynomial kernel: $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\gamma \boldsymbol{x}_i^T \boldsymbol{x}_j + c)^d$.
- 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
    (Lodhi et al., "Text classification using string kernels". JMLR, 2002).

General Kernelized ERM

- L2-Regularized Empirical Risk Minimization:

$$\min_{\boldsymbol{w}} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n} \ell_i(\boldsymbol{w}^T\varphi(\boldsymbol{x}_i))$$

  $\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n$: training samples
  $\varphi(\boldsymbol{x}_i)$: nonlinear mapping to a higher dimensional space

- Dual problem:

$$\min_{\boldsymbol{\alpha}} \frac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} + \sum_{i=1}^{n} \ell_i^*(-\alpha_i)$$

  where $Q \in \mathbb{R}^{n\times n}$ and $Q_{ij} = \varphi(\boldsymbol{x}_i)^T\varphi(\boldsymbol{x}_j) = K(\boldsymbol{x}_i, \boldsymbol{x}_j)$.

Kernel Ridge Regression

- Given training samples $(\boldsymbol{x}_i, y_i), i = 1, \cdots, n$.

$$\min_{\boldsymbol{w}} \frac{1}{2}\|\boldsymbol{w}\|^2 + \frac{1}{2}\sum_{i=1}^{n}(\boldsymbol{w}^T\varphi(\boldsymbol{x}_i) - y_i)^2$$

- Dual problem:

$$\min_{\boldsymbol{\alpha}} \boldsymbol{\alpha}^T Q\boldsymbol{\alpha} + \|\boldsymbol{\alpha}\|^2 - 2\boldsymbol{\alpha}^T\mathbf{y}$$

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:

$$\min_{\boldsymbol{w},\xi} \ \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{n}\xi_i$$
$$\text{s.t. } y_i(\boldsymbol{w}^T\varphi(\boldsymbol{x}_i)) \geq 1-\xi_i, \quad \xi_i \geq 0, \ i = 1,\ldots,n,$$

  $\boldsymbol{w}$: an infinite dimensional vector, very hard to solve

- The dual problem for SVM:

$$\min_{\boldsymbol{\alpha}} \ \frac{1}{2}\boldsymbol{\alpha}^T Q\boldsymbol{\alpha} - \boldsymbol{e}^T\boldsymbol{\alpha},$$
$$\text{s.t. } 0 \leq \alpha_i \leq C, \ \text{ for } \ i = 1,\ldots,n,$$

  where $Q_{ij} = y_i y_j \varphi(\boldsymbol{x}_i)^T\varphi(\boldsymbol{x}_j)$ and $e = [1,\ldots,1]^T$.

- Kernel trick: define $K(\boldsymbol{x}_i,\boldsymbol{x}_j) = \varphi(\boldsymbol{x}_i)^T\varphi(\boldsymbol{x}_j)$.
- Example: Gaussian kernel $K(\boldsymbol{x}_i,\boldsymbol{x}_j) = e^{-\gamma\|\boldsymbol{x}_i-\boldsymbol{x}_j\|^2}$

Nonlinear SVM

- Can we solve the problem by dual coordinate descent?
- The vector $\boldsymbol{w} = \sum_i y_i \alpha_i \varphi(\boldsymbol{x}_i)$ may have infinite dimensionality:

  Cannot maintain $\boldsymbol{w}$
- Closed form solution needs $O(n)$ computational time:

$$\delta^* = \max(-\alpha_i \min(C - \alpha_i, \frac{1 - (Q\boldsymbol{\alpha})_i}{Q_{ii}}))$$

  (Assume Q is stored in memory)
- Can we improve coordinate descent using the same $O(n)$ time complexity?

Greedy Coordinate Descent

The Greedy Coordinate Descent (GCD) algorithm:

For $t = 1, 2, \ldots$

1. Compute $\delta_i^* := \operatorname{argmin}_\delta f(\boldsymbol{\alpha} + \delta \boldsymbol{e}_i)$ for all $i = 1, \ldots, n$
2. Find the best i^* according to the following criterion:

$$i^* = \operatorname*{argmax}_i |\delta_i^*|$$

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

Greedy Coordinate Descent

Other variable selection criterion:

- The coordinate with the maximum step size:

$$i^* = \underset{i}{\text{argmax}} \, |\delta_i^*|$$

- The coordinate with maximum objective function reduction:

$$i^* = \underset{i}{\text{argmax}} \left(f(\boldsymbol{\alpha}) - f(\boldsymbol{\alpha} + \delta_i^* \boldsymbol{e}_i) \right)$$

- The coordinate with the maximum projected gradient.
- ...

Greedy Coordinate Descent

- How to compute the optimal coordinate?
  - Closed form solution of best δ:

$$\delta_i^* = \max\left(-\alpha_i, \min\left(C - \alpha_i, \frac{1 - (Q\boldsymbol{\alpha})_i}{Q_{ii}}\right)\right)$$

  - Observations:
    1. Computing all δ_i^* needs $O(n^2)$ time
    2. If $Q\boldsymbol{\alpha}$ is stored in memory, computing all δ_i^* only needs $O(n)$ time
  - Maintaining $Q\boldsymbol{\alpha}$ also needs $O(n)$ time after each update

Greedy Coordinate Descent

Initial: $\boldsymbol{\alpha}$, $\boldsymbol{z} = Q\boldsymbol{\alpha}$

For $t = 1, 2, \ldots$

For all $i = 1, \ldots, n$, compute

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

Let $i^* = \text{argmax}_i |\delta_i^*|$

$\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha} + \delta_{i^*}^*$

$\boldsymbol{z} \leftarrow \boldsymbol{z} + \boldsymbol{q}_{i^*} \delta_{i^*}^*$ ($\boldsymbol{q}_{i^*}$ is the i^*-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 $\boldsymbol{q}_i$ is in memory, directly use it to update

  If $\boldsymbol{q}_i$ is not in memory

  1. Kick out the "Least Recent Used" column
  2. Recompute $\boldsymbol{q}_i$ and store it in memory
  3. Update α_i

LIBSVM

- 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 $G_{ij} = K(\boldsymbol{x}_i, \boldsymbol{x}_j)$

- Can we do SVD?

Kernel Matrix Approximation

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

- 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 $G_{ij} = K(\boldsymbol{x}_i, \boldsymbol{x}_j)$

- 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 $G_{ij} = K(\boldsymbol{x}_i, \boldsymbol{x}_j)$

- 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:

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

Nystrom Approximation

Algorithm:

1. Sample m "landmark points": $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_m$
2. Compute the kernel values between all training data to landmark points:

$$C = \begin{bmatrix} W \\ G_{21} \end{bmatrix} = \begin{bmatrix} K(\boldsymbol{x}_1, \boldsymbol{v}_1) & \cdots & K(\boldsymbol{x}_1, \boldsymbol{v}_m) \\ K(\boldsymbol{x}_2, \boldsymbol{v}_1) & \cdots & K(\boldsymbol{x}_2, \boldsymbol{v}_m) \\ \vdots & \vdots & \vdots \\ K(\boldsymbol{x}_n, \boldsymbol{v}_1) & \cdots & K(\boldsymbol{x}_n, \boldsymbol{v}_m) \end{bmatrix}$$

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

$$\min_{\boldsymbol{\alpha}} \boldsymbol{\alpha}^T \tilde{G} \boldsymbol{\alpha} + \lambda \|\boldsymbol{\alpha}\|^2 - \mathbf{y}^T \boldsymbol{\alpha}$$

  Solve a linear system $(\tilde{G} + \lambda I)\boldsymbol{\alpha} = \mathbf{y}$
- Use iterative methods (such as CG), with fast matrix-vector multiplication

$$(\tilde{G} + \lambda I)\boldsymbol{p} = C W^{\dagger} C^T \boldsymbol{p} + \lambda \boldsymbol{p}$$

  $O(nm)$ time complexity per iteration

Nystrom Approximation: Training

- 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 represent by linear kernel with feature matrix $C(W^{\dagger})^{1/2}$:
$$\tilde{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \tilde{G}_{ij} = \boldsymbol{u}_i^{T}\boldsymbol{u}_j,$$
where $\boldsymbol{u}_j = (W^{\dagger})^{1/2}\begin{bmatrix} K(\boldsymbol{x}_j, \boldsymbol{v}_1) \\ \vdots \\ K(\boldsymbol{x}_j, \boldsymbol{v}_m) \end{bmatrix}$
- The problem is equivalent to a linear models with features $\boldsymbol{u}_1, \cdots, \boldsymbol{u}_n \in \mathbb{R}^m$
  - 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 $\boldsymbol{\alpha}$.
- Prediction for testing sample $\boldsymbol{x}$:

$$\sum_{i=1}^{n} \alpha_i \tilde{K}(\boldsymbol{x}_i, \boldsymbol{x}) \quad \text{or} \quad \sum_{i=1}^{n} \alpha_i K(\boldsymbol{x}_i, \boldsymbol{x}) \ ?$$

Nystrom Approximation: Prediction

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

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

- **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 $\tilde{K}(\boldsymbol{x}, \boldsymbol{x}_i)$?

$$\tilde{G} = \begin{bmatrix} K(\boldsymbol{x}_1, \boldsymbol{v}_1) & \cdots & K(\boldsymbol{x}_1, \boldsymbol{v}_m) \\ \vdots & \vdots & \vdots \\ K(\boldsymbol{x}_n, \boldsymbol{v}_1) & \cdots & K(\boldsymbol{x}_1, \boldsymbol{v}_m) \\ K(\boldsymbol{x}, \boldsymbol{v}_1) & \cdots & K(\boldsymbol{x}_1, \boldsymbol{v}_m) \end{bmatrix} W^{\dagger} \begin{bmatrix} K(\boldsymbol{v}_1, \boldsymbol{x}_1) & \cdots & K(\boldsymbol{v}_1, \boldsymbol{x}) \\ \vdots & \vdots & \vdots \\ K(\boldsymbol{v}_m, \boldsymbol{x}_1) & \cdots & K(\boldsymbol{v}_m, \boldsymbol{x}) \end{bmatrix}$$

- Compute $\boldsymbol{u} = (W^{\dagger})^{1/2} \begin{bmatrix} K(\boldsymbol{v}_1, \boldsymbol{x}) \\ \vdots \\ K(\boldsymbol{v}_m, \boldsymbol{x}) \end{bmatrix}$ (need m kernel evaluations)
- Approximate kernel can be defined by

$$\tilde{K}(\boldsymbol{x}, \boldsymbol{x}_i) = \boldsymbol{u}^T \boldsymbol{u}_i$$

Nystrom Approximation: Prediction

- Prediction:

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i \boldsymbol{u}^T \boldsymbol{u}_i = \boldsymbol{u}^T (\sum_{i=1}^{n} \alpha_i \boldsymbol{u}_i)$$

  $(\sum_{i=1}^{n} \alpha_i \boldsymbol{u}_i)$ can be pre-computed
  $\Rightarrow$ prediction time is m kernel evaluations

- Original kernel method: need *n* kernel evaluations for prediction.
- Summary:

	Quality	Training time	Prediction time
Original kernel	exact	$n^{1.x}$ kernel + kernel training	n kernel
Nystrom (m)	rank m	nm kernel + linear training	m kernel

Nystrom Approximation

- How to select landmark points $(\boldsymbol{v}_1, \cdots, \boldsymbol{v}_m)$
  - Traditional approach: uniform random sampling from training data
  - Importance sampling (leverage score):
    - Drineas and Mahoney "On the Nystrom method for approximating a Gram matrix for improved kernel-based learning". JMLR 2015.
    - 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 $\boldsymbol{v}_1, \cdots, \boldsymbol{v}_m$ to be cluster centers
    - Zhang et al., "Improved Nystrom low rank approximation and error analysis". ICML 2008.
  - Subspace distance:
    - Lim et al., "Double Nystrom method: An efficient and accurate Nystrom scheme for large-scale data sets". ICML 2015.

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"

  $\Rightarrow$ 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 $K(\boldsymbol{x}, \mathbf{y}) = k(\boldsymbol{x} - \mathbf{y})$.
  - Gaussian kernel: $K(\boldsymbol{x}, \mathbf{y}) = e^{-\|\boldsymbol{x}-\mathbf{y}\|^2}$
  - Laplacian kernel: $K(\boldsymbol{x}, \mathbf{y}) = e^{-\|\boldsymbol{x}-\mathbf{y}\|_1}$
- Shift invariant kernel (if positive definite) can be written as:

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

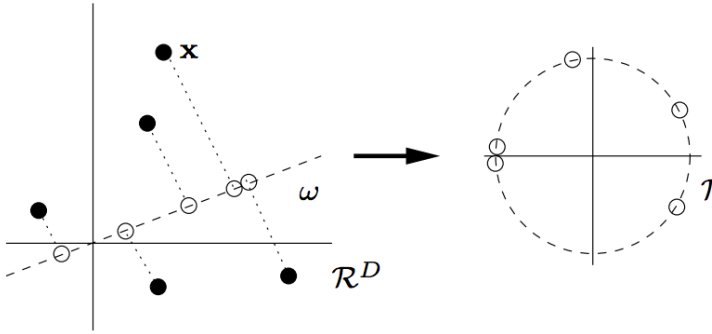
  for some probability distribution $P(\boldsymbol{w})$.

Random Fourier Features

- Taking the real part we have

$$
\begin{aligned}
k(\boldsymbol{x}-\mathbf{y}) &= \int_{\mathbb{R}^d} P(\boldsymbol{w}) \cos(\boldsymbol{w}^T\boldsymbol{x} - \boldsymbol{w}^T\mathbf{y}) \\
&= \int_{\mathbb{R}^d} P(\boldsymbol{w})(\cos(\boldsymbol{w}^T\boldsymbol{x})\cos(\boldsymbol{w}^T\mathbf{y}) + \sin(\boldsymbol{w}^T\boldsymbol{x})\sin(\boldsymbol{w}^T\mathbf{y})) \\
&= E_{\boldsymbol{w}\sim P(\boldsymbol{w})}[z_w(\boldsymbol{x})^T z_w(\mathbf{y})]
\end{aligned}
$$

  where $z_w(\boldsymbol{x})^T = [\cos(\boldsymbol{w}^T\boldsymbol{x}),\ \ \sin(\boldsymbol{w}^T\boldsymbol{x})]$.
- $z_w(\boldsymbol{x})^T z_w(\mathbf{y})$ is an unbiased estimator of $K(\boldsymbol{x}, \mathbf{y})$ if $\boldsymbol{w}$ is sampled from $P(\cdot)$



Kernel Name	$k(\Delta)$	$p(\omega)$
Gaussian	$e^{-\frac{\|\Delta\|_2^2}{2}}$	$(2\pi)^{-\frac{D}{2}} e^{-\frac{\|\omega\|_2^2}{2}}$
Laplacian	$e^{-\|\Delta\|_1}$	$\prod_d \frac{1}{\pi(1+\omega_d^2)}$
Cauchy	$\prod_d \frac{2}{1+\Delta_d^2}$	$e^{-\|\Delta\|_1}$

Random Fourier Features

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

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

- $K(\boldsymbol{x}_i, \boldsymbol{x}_j) \approx \boldsymbol{u}_i^T \boldsymbol{u}_j$

  (larger m leads to better approximation)

Random Features

- $G \approx UU^T$, where $U = \begin{bmatrix} \boldsymbol{u}_1^T \\ \vdots \\ \boldsymbol{u}_n^T \end{bmatrix}$
- Rank $2m$ approximation.
- The problem can be reduced to linear classification/regression for $\boldsymbol{u}_1, \cdots, \boldsymbol{u}_n$.
- 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 $\boldsymbol{w}_i^T \boldsymbol{x}$ for all $i = 1, \cdots, m$

- 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 $\boldsymbol{w}_i^T \boldsymbol{x}$ for all $i = 1, \cdots, m$

- 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 ($H_d \in \mathbb{R}^{d \times d}$)

$$H_d \boldsymbol{x} \text{ requires } O(d \log d) \text{ time}$$

- Hadamard matrix:

$$H_1 = [1]$$

$$H_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$H_{2^k} = \begin{bmatrix} H_{2^{k-1}} & H_{2^{k-1}} \\ H_{2^{k-1}} & -H_{2^{k-1}} \end{bmatrix}$$

- $H_d \boldsymbol{x}$ can be computed efficiently by Fast Hadamard Transform (dynamic programming)

Fastfood

- Sample $m = d$ random features by

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

  So $W\boldsymbol{x}$ 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:
  (Dai et al., "Scalable Kernel Methods via Doubly Stochastic Gradients". NIPS 2015.)
- Generalization bound (?)

Coming up

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

Questions?