Random Kitchen Sinks

Amit Rotem

Colorado School of Mines

November 15, 2019

Nonlinear Models

- Given data $\{x_i, y_i\}$ for $i = 1, \dots, m$. Assume model:

$$y_i = f(x_i) + \epsilon_i$$

  for some continuous $f : \mathcal{X} \to \mathbb{R}$ ($\mathcal{X} \subset \mathbb{R}^d$) and random noise ϵ_i.
- We find f by assuming the following basis expansion:

$$f(x) = \sum_{i=1}^{n} c_i \phi(x; \theta_i)$$

  for $c_i \in \mathbb{R}$ and $\theta_i \in \mathbb{R}^N$

The Kernel Trick

- Define some positive finite measure μ on the set of all possible θ then we can define a kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$:

$$k(x, \tilde{x}) = \langle \phi(x;\theta), \phi(\tilde{x};\theta) \rangle_\mu,$$

and represent f in a Reproducing Kernel Hilbert Space using our data:

$$f(x) = \sum_{j=1}^{m} w_j k(x, x_j)$$

for $w \in \mathbb{R}$.

The Kernel Trick

- Define some positive finite measure μ on the set of all possible θ then we can define a kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$:

$$k(x, \tilde{x}) = \langle \phi(x; \theta), \phi(\tilde{x}; \theta) \rangle_{\mu},$$

  and represent f in a Reproducing Kernel Hilbert Space using our data:

$$f(x) = \sum_{j=1}^{m} w_j k(x, x_j)$$

  for $w \in \mathbb{R}$.
- This representation is used in support vector machines, Gaussian process, kernel ridge regression, etc.

Kernel Methods are Expensive

- We find w by solving a large quadratic programming problem that scales with $\mathcal{O}(m^3)$.

Kernel Methods are Expensive

- We find w by solving a large quadratic programming problem that scales with $\mathcal{O}(m^3)$.
- We can approximate our solution by assuming the spectral decomposition of k only has $l \ll m$ relatively large eigenvalues (Nyström method).

Kernel Methods are Expensive

- We find w by solving a large quadratic programming problem that scales with $\mathcal{O}(m^3)$.
- We can approximate our solution by assuming the spectral decomposition of k only has $l \ll m$ relatively large eigenvalues (Nyström method).
  - requires computation of SVD *before* regression.
  - the eigenvalues may not go to zero quickly enough.
  - number of non-zero eigenvalues depends on data.

Kernel Methods are Expensive

- We find w by solving a large quadratic programming problem that scales with $\mathcal{O}(m^3)$.
- We can approximate our solution by assuming the spectral decomposition of k only has $l \ll m$ relatively large eigenvalues (Nyström method).
  - requires computation of SVD *before* regression.
  - the eigenvalues may not go to zero quickly enough.
  - number of non-zero eigenvalues depends on data.
- What can we do to make this problem scalable?

Random Fourier Features

recall $k(x, \tilde{x}) = \langle \phi(x;\theta)\phi(\tilde{x};\theta) \rangle_\mu$

- Rahimi and Recht 2008 showed that if we select μ to be a probability measure with density $p(\theta)$, and $\phi(x;\theta) = e^{i\theta^T x}$ then $k(x, \tilde{x}) = \hat{p}(||x - \tilde{x}||)$ where $\hat{p}$ is the Fourier transform of p.

Random Fourier Features

recall $k(x, \tilde{x}) = \langle \phi(x; \theta)\phi(\tilde{x}; \theta) \rangle_\mu$

- Rahimi and Recht 2008 showed that if we select μ to be a probability measure with density $p(\theta)$, and $\phi(x; \theta) = e^{i\theta^T x}$ then $k(x, \tilde{x}) = \hat{p}(||x - \tilde{x}||)$ where $\hat{p}$ is the Fourier transform of p.
- By sampling $D \ll m$ values of θ from p, we can approximate any function in the RKHS of k.

Random Fourier Features

recall $k(x, \tilde{x}) = \langle \phi(x; \theta)\phi(\tilde{x}; \theta) \rangle_\mu$

- Rahimi and Recht 2008 showed that if we select μ to be a probability measure with density $p(\theta)$, and $\phi(x; \theta) = e^{i\theta^T x}$ then $k(x, \tilde{x}) = \hat{p}(||x - \tilde{x}||)$ where $\hat{p}$ is the Fourier transform of p.
- By sampling $D \ll m$ values of θ from p, we can approximate any function in the RKHS of k.
- e.g. if we want $Pr(||\hat{f}_{\text{full}} - \hat{f}_{\text{RFF}}||_\mu < 10^{-1}) > 0.90$ then $D > 2500$.

Random Fourier Features

recall $k(x, \tilde{x}) = \langle \phi(x;\theta)\phi(\tilde{x};\theta)\rangle_\mu$

- Rahimi and Recht 2008 showed that if we select μ to be a probability measure with density $p(\theta)$, and $\phi(x;\theta) = e^{i\theta^T x}$ then $k(x, \tilde{x}) = \hat{p}(||x - \tilde{x}||)$ where $\hat{p}$ is the Fourier transform of p.
- By sampling $D \ll m$ values of θ from p, we can approximate any function in the RKHS of k.
- e.g. if we want $Pr(||\hat{f}_{\text{full}} - \hat{f}_{\text{RFF}}||_\mu < 10^{-1}) > 0.90$ then $D > 2500$.
- **D does *not* depend on the sample size m**

Random Fourier Features

recall $k(x, \tilde{x}) = \langle \phi(x; \theta)\phi(\tilde{x}; \theta) \rangle_\mu$

- Rahimi and Recht 2008 showed that if we select μ to be a probability measure with density $p(\theta)$, and $\phi(x; \theta) = e^{i\theta^T x}$ then $k(x, \tilde{x}) = \hat{p}(||x - \tilde{x}||)$ where $\hat{p}$ is the Fourier transform of p.
- By sampling $D \ll m$ values of θ from p, we can approximate any function in the RKHS of k.
- e.g. if we want $Pr(||\hat{f}_{\text{full}} - \hat{f}_{\text{RFF}}||_\mu < 10^{-1}) > 0.90$ then $D > 2500$.
- D **does *not* depend on the sample size** m
- only the linear coefficients need to be computed.

RFFs for the Gaussian Kernel

- The most broadly used kernel in ML is the Gaussian kernel

$$k(x, \tilde{x}) = \exp\left(-\gamma ||x - \tilde{x}||_2^2\right)$$

RFFs for the Gaussian Kernel

- The most broadly used kernel in ML is the Gaussian kernel

$$k(x, \tilde{x}) = \exp\left(-\gamma ||x - \tilde{x}||_2^2\right)$$

- We can approximate f in the RKHS of k by sampling D values for $\omega \sim \mathcal{N}(0, 2d\gamma I)$ and $b \sim \text{Unif}(-\pi, \pi)$ and computing D coefficients α_j so that:

$$\hat{f}(x) = \sum_{j=1}^{D} \alpha_j \cos\left(\omega_j^T x + b_j\right)$$

RFFs for the Gaussian Kernel...

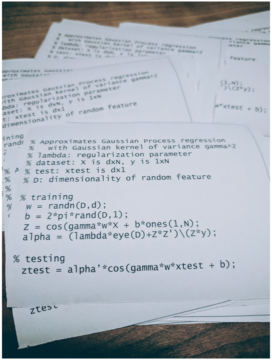
- we only need to compute D coefficients.

RFFs for the Gaussian Kernel...

- we only need to compute D coefficients.
- Problem scales linearly in m.

RFFs for the Gaussian Kernel...

- we only need to compute D coefficients.
- Problem scales linearly in m.



just few lines of MATLAB code!

Fitting Procedure

Algorithm 1: Random Kitchen Sinks Fitting Procedure

Data: a set of m points $\{x_i, y_i\}$, feature map $|\phi(x;\theta)| \leq 1$ with corresponding density $p(\theta)$ of the parameters of ϕ, an integer D, a loss function $c(y, \hat{y})$, a regularization parameter C.

Result: a set of D coefficients α_j and D parameters θ_j such that $\hat{f}(x_i) = \sum_{j=1}^{D} \alpha_j \phi(x_i; \theta_j)$.

1 sample $\theta_1, \cdots, \theta_D$ iid from p;

2 compute features $z_i \leftarrow [\phi(x_i;\theta_1), \cdots, \phi(x_i;\theta_D)]^T$;

3 with θ fixed, solve the loss minimization problem:

$$\min_{\alpha} \frac{1}{m} \sum_{i=1}^{m} c(y_i, \alpha^T z_i)$$

subject to constraints: $||\alpha||_\infty < C/D$;

Main Result

Define

$$\mathcal{F}_p = \left\{ f(x) = \int_{\Theta} \alpha(\theta)\phi(x;\theta)d\theta \;\middle|\; |\alpha(\theta)| < Cp(\theta) \right\}.$$

and $R[\hat{f}] = \mathbb{E}[c(y, \hat{f}(x))]$ for some L-Lipschitz convex loss function c (MSE, hinge loss, etc.).
Algorithm 1 produces an $\hat{f}$ that can estimate any $f \in \mathcal{F}_p$ with error:

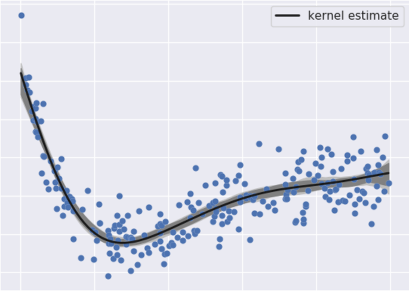
$$R[\hat{f}] - \min_{f \in \mathcal{F}_p} R[f] < \mathcal{O}\left(\left(\frac{1}{\sqrt{D}} + \frac{1}{\sqrt{m}} \right) LC \sqrt{\log \frac{1}{\delta}} \right)$$

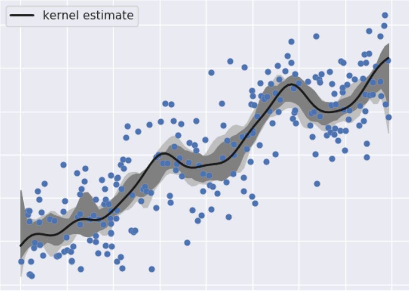
with probability $1 - 2\delta$.

Experimental Results

Dataset	Fourier+LS	Binning+LS	Exact SVM
`CPU` regression 6500 instances 21 dims	3.6% error 20 secs $D = 300$	5.3% 3 minutes $P = 350$	11% 31 secs ASVM
`Census` regression 18000 instances 119 dims	5% 36 secs $D = 500$	7.5% 19 mins $P = 30$	9% 13 mins SVMTorch
`Adult` classification 32000 instances 123 dims	14.9% 9 secs $D = 500$	15.3% 1.5 mins $P = 30$	15.1% 7 mins $\text{SVM}^{\text{light}}$
`Forest Cover` classification 522000 instances 54 dims	11.6% 71 mins $D = 5000$	2.2% 25 mins $P = 50$	2.2% 44 hrs libSVM

Table: experiments conducted by Rahimi and Recht 2008





variablity of RFF estimates ($D = 20$) for two sets of simulated non-linear data

When Random Kitchen Sinks are Slow

- for d-dimensional data most costly RKS computation is the storage and multiplication of $d \times D$ random matrix.
- very high dimensional data are common
  - e.g. ImageNet has average $d = 469 \times 387 \approx 189000$ features

When Random Kitchen Sinks are Slow

- for d-dimensional data most costly RKS computation is the storage and multiplication of $d \times D$ random matrix.
- very high dimensional data are common
  - e.g. ImageNet has average $d = 469 \times 387 \approx 189000$ features
- What if we replaced the purely random matrix by a “special” random matrix?

Approximate random matrices

Suppose we use RFFs for the Gaussian kernel:
$\hat{f}(x) = \alpha^T cos(Wx + b)$ for $W \sim \mathcal{N}(0, 2d\gamma I)$ and $b \sim \text{Unif}(-\pi, \pi)$
without loss of generalization, let $x \in \mathbb{R}^d$ for $d = 2^l$, $l \in \mathbb{Z}^+$

- let H be the $d \times d$ Hadamard matrix
- let B be a diagonal matrix such that $B_{ii} \sim \text{Unif}(\{-1, 1\})$
- let G be a diagonal matrix such that $G_{ii} \sim \mathcal{N}(0, 1)$
- let Π be a permutation matrix
- let S be a diagonal matrix such that $S_{ii} = ||G||_F^{-\frac{1}{2}} s_i$ and $s_i \sim \chi_d$

then $\tilde{W} = \frac{2\gamma}{\sqrt{d}} SHG\Pi HB$ has some very useful properties.

Fastfood approximation

- B, G, S can be stored as vectors.
- Π can be stored implicitly as a vector of indices.
- H need not be stored. Left multiplication of a vector by H can be computed via the fast Walsh Hadamard transform.
- let $\phi(x) = cos(\tilde{W}x + b)$ then

$$\mathbb{E}\langle\phi(x), \phi(y)\rangle_\mu = k_{Gauss}(x, y)$$

Fastfood

The Fastfood procedure is efficient:

- Reduces memory requirements from $\mathcal{O}(d \times D)$ to $\mathcal{O}(D)$
- Reduces time complexity from $\mathcal{O}(d \times D)$ to $\mathcal{O}(D \log d)$

Fastfood

The Fastfood procedure is efficient:

- Reduces memory requirements from $\mathcal{O}(d \times D)$ to $\mathcal{O}(D)$
- Reduces time complexity from $\mathcal{O}(d \times D)$ to $\mathcal{O}(D \log d)$

It is accurate:

- $\text{error}_{\text{fastfood}} < \mathcal{O}(\text{error}_{\text{RFF}} \log \frac{1}{\text{error}_{\text{RFF}}})$
- comparative experimental results to RFF.

Orthogonal Random Features

Felix et al. 2016 showed that if the rows of W are orthogonal, we reduce can significantly reduce error. They proposed two methods which reduce error from $\mathcal{O}(\frac{1}{\sqrt{D}})$ to $\mathcal{O}(\frac{1}{D})$.

Orthogonal Random Features

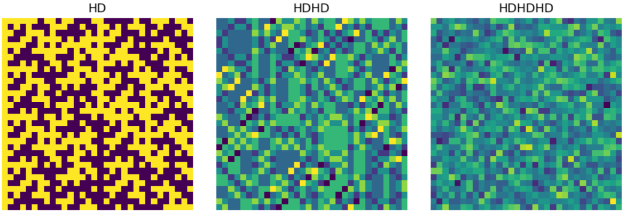
$$W_{ORF} = 2d\gamma SQ$$

- Q is produced from the economic QR decomp. of a matrix $G \sim N(0, 1)$
- S is diagonal such that $S_{ii} \sim \chi_d$

Structured Orthogonal Random Features

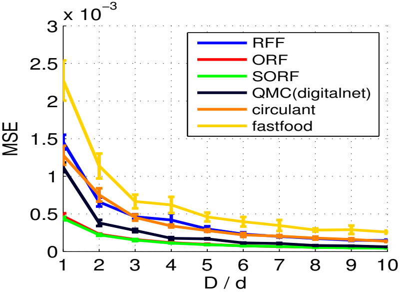
$$W_{SORF} = 2d\gamma\sqrt{d}HD_1HD_2HD_3$$

- $\sqrt{d} = \mathbb{E}S_{ii}$
- H encodes the Hadamard transform.
- $D_{j=1,2,3}$ are diagonal matrices where $D_{j,ii} \sim \text{Unif}(\{-1, 1\})$

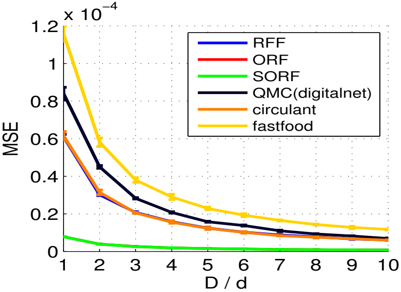


comparison of W_{SORF} for 1, 2, and 3 transformations

Comparison



USPS ($d = 256$)



GISSETTE ($d = 4096$)

experiments conducted by Felix et al. 2016

Who's using it?

- MATLAB function `fitrkernel` uses Fastfood
- scikit-learn (Python) class `RBFSampler` uses RFFs

AI Koan

In the days when Sussman was a novice, Minsky once came to him as he sat hacking at the PDP-6.
“What are you doing?” asked Minsky. “I am training a randomly wired neural net to play tic-tac-toe,” Sussman replied. “Why is the net wired randomly?” asked Minsky. Sussman replied, “I do not want it to have any preconceptions of how to play.” Minsky then shut his eyes. “Why do you close your eyes?” Sussman asked his teacher.
“So that the room will be empty,” replied Minsky. At that moment, Sussman was enlightened.

References

Felix, X Yu et al. (2016). "Orthogonal random features". In: *Advances in Neural Information Processing Systems*, pp. 1975–1983.

Quoc Le, Tamás Sarlós and Alex Smola (2013). "Fastfood-approximating kernel expansions in loglinear time". In: *Proceedings of the international conference on machine learning* vol. 85.

Rahimi, Ali and Benjamin Recht (2008). "Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning". In: *Advances in neural information processing systems*, pp. 1313–1320.