Random Features for Large Scale Kernel Machines

Andrea Vedaldi

Explicit feature maps

- Fast algorithm for linear kernels
  - one-slack SVM (SVM-perf)
  - primal SVM (liblinear)
  - stochastic SVM (PEGASOS)
  - PCA

- Slow for non linear kernel K

- However any kernel is linear in an appropriate feature space
  \[ K(x, y) = \langle \Psi(x), \Psi(y) \rangle \]

- Can we get a finite dimensional feature map approximating the kernel?
  \[ K(x, y) \approx \langle \hat{\Psi}(x), \hat{\Psi}(y) \rangle, \quad \Psi(x) \in \mathbb{R}^D \]
Random Fourier features 1/2

- Translation invariant kernel: $K(x, y) = K(y - x)$

- Bochner
  If $K$ is PD and translation invariant, then
  $$K(\lambda) = \int e^{-i\omega^\top \lambda} \kappa(\lambda) \, d\lambda, \quad \kappa(\lambda) = \kappa(-\lambda) \in \mathbb{R}_0^+$$
  for some real, symmetric, non neg. measure $\kappa(\lambda) \, d\lambda$.

- Bochner as expected value
  If $K(0) = 1$ (normalized), then
  $$1 = K(0) = \int \kappa(\lambda) \, d\lambda,$$
  $$K(\lambda) = E \left[ e^{-i\omega^\top \lambda} \right], \quad \lambda \sim \kappa(\lambda)$$
Random Fourier features

Obtain approximate feature map from random sampling

\[ K(\lambda) = E \left[ e^{-i\omega^T \lambda} \right] \approx \frac{1}{D} \sum_{i=1}^{D} e^{-i\omega_i^T \lambda}, \quad \omega_i \sim \kappa(\omega) \]

\[ e^{-i\omega_i^T \lambda} = (e^{-i\omega_i^T x})^* e^{-i\omega_i^T y} \]

\[ K(\lambda) \approx \langle \widehat{\Psi}(x), \widehat{\Psi}(y) \rangle, \quad \widehat{\Psi}(x) = \frac{1}{\sqrt{D}} \left[ e^{-i\omega_1^T x} \ldots e^{-i\omega_D^T x} \right]^T \]

Gaussian random Fourier features

\[ K(\lambda) = \exp(-\|\lambda\|^2/(2\sigma^2)) \quad \kappa(\omega) \propto \exp(-\|\omega\|^2\sigma^2/2) \]

MATLAB pseudocode

```matlab
function psix = gaussianRandomFeatures(x, D)
d = size(x, 1);
omega = randn(D, d);
psix = exp(-i*omega*x);
```
Random Fourier features: Errors

**Claim 1** (Uniform convergence of Fourier features). Let $\mathcal{M}$ be a compact subset of $\mathbb{R}^d$ with diameter $\text{diam}(\mathcal{M})$. Then, for the mapping $z$ defined in Algorithm 1, we have

$$
\Pr \left[ \sup_{x, y \in \mathcal{M}} |z(x)'z(y) - k(x, y)| \geq \epsilon \right] \leq 2^8 \left( \frac{\sigma_p \text{diam}(\mathcal{M})}{\epsilon} \right)^2 \exp \left( - \frac{D\epsilon^2}{4(d + 2)} \right),
$$

where $\sigma_p^2 = E_p[\omega'\omega]$ is the second moment of the Fourier transform of $k$. Further, $\sup_{x, y \in \mathcal{M}} |z(x)'z(y) - k(y, x)| \leq \epsilon$ with any constant probability when $D = \Omega \left( \frac{d}{\epsilon^2} \log \frac{\sigma_p \text{diam}(\mathcal{M})}{\epsilon} \right)$.

- **Uniform error bound in a ball**

$$
\forall x, y \in B(R) \subset \mathbb{R}^d : \quad |\langle \hat{\Psi}(x), \hat{\Psi}(y) \rangle - K(x, y)| < \epsilon
$$

(with any fixed probability) requires

$$
D = \Omega \left( \frac{d}{\epsilon^2} \log \frac{2Rd}{\sigma^2 \epsilon} \right)
$$

random projections.
Random Binning features 1/2

\[ P[\exists i : x \in B_i \land y \in B_i] = \]

\[ k(x, y) = \max \left( 0, 1 - \frac{|y - x|}{\delta} \right) \]

- \( \hat{\Psi}(x; u) \) indicator vector for the bin occupied by \( x \) under shift \( u \)

\[ k(x; y) = E[\langle \hat{\Psi}(x; u), \hat{\Psi}(y; u) \rangle] \approx \frac{1}{D} \sum_{j=1}^{D} \langle \hat{\Psi}(x; u_j), \hat{\Psi}(y; u_j) \rangle \]

\[ = \langle \text{stack} \sum_{j=1}^{D} \frac{\hat{\Psi}(x; u_j)}{\sqrt{D}} \rangle, \text{stack} \sum_{j=1}^{D} \frac{\hat{\Psi}(y; u_j)}{\sqrt{D}} \rangle \]
Random binning features 2/2

- Decompose other kernels as averages over hat kernels
  \[
  k(\lambda) = \int_{0}^{+\infty} k_{\text{hat}}(\lambda; \delta) p(\delta) \delta
  \]
  - it’s a simple deconvolution problem solved by \( p(\delta) = \delta \hat{k}(\delta) \)

- Assuming that \( \hat{k}(\delta) \geq 0 \)
  - interpret \( p(\delta) \) as a probability density
  - expand kernel as
  \[
  k(\lambda) = E[k_{\text{hat}}(\lambda; \delta)] \approx \left\langle \text{stack} \sum_{j=1}^{D} \frac{\hat{\Psi}(x; \delta_j, u_j)}{\sqrt{D}}, \text{stack} \sum_{j=1}^{D} \frac{\hat{\Psi}(y; \delta_j, u_j)}{\sqrt{D}} \right\rangle
  \]
  - \( u_j \sim \) uniformly at random in \([0, \delta]\)
  - \( \delta_j \sim p(\delta) \)
Random binning features 3/3

- **Separable kernel**

\[
K(x, y) = K(y - x) = K(\lambda) = \prod_{m=1}^{d} k(\lambda_m)
\]

- represent as the probability of being binned together simultaneously in the \(d\) dimensions

\[
K(x - y) \approx \left\langle \text{stack} \prod_{j=1}^{D} \prod_{m=1}^{d} \frac{\widehat{\Psi}(x_m; \delta_{m,j}, u_{m,j})}{\sqrt{D}}, \text{stack} \prod_{j=1}^{D} \prod_{m=1}^{d} \frac{\widehat{\Psi}(y_m; \delta_{m,j}, u_{m,j})}{\sqrt{D}} \right\rangle
\]

- **Remarks**

- The feature map has very high dimension, but it is very sparse
- Must be stored in a hash structure
- Convergence rates similar to the random Fourier features
To evaluate the resulting machine on a datapoint computing random features and then applying the associated linear technique can be applied to accelerate other kernel methods, including semi-supervised and unsupervised learning.

Catenating these features can improve accuracy, training time, and evaluation time.

We have presented randomized features whose inner products uniformly approximate many popular kernels. Performance can be obtained even from a modest number of features.

Figure 1 illustrates the benefit of training classifiers on larger datasets, where accuracy can be improved.

SVM requires many support vectors because they explicitly preserve locality in the input space. Hand, random binning features perform better on memorization tasks with those for which the standard approaches are less effective.

Random Fourier features perform better on the tasks that largely rely on interpolation.

The table compares the performance of ridge regression with random features, Core Vector Machine (CVM), and Exact SVM on various datasets. Despite its simplicity, ridge regression with random features is faster than and provides competitive accuracy with alternative methods. It also produces very compact functions.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Fourier+LS</th>
<th>Binning+LS</th>
<th>CVM</th>
<th>Exact SVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU regression 6500 instances 21 dims</td>
<td>3.6% 20 secs $D = 300$</td>
<td>5.3% 3 mins $P = 350$</td>
<td>5.5% 51 secs</td>
<td>11% 31 secs ASVM</td>
</tr>
<tr>
<td>Census regression 18,000 instances 119 dims</td>
<td>5% 36 secs $D = 500$</td>
<td>7.5% 19 mins $P = 30$</td>
<td>8.8% 7.5 mins</td>
<td>9% 13 mins SVM^light</td>
</tr>
<tr>
<td>Adult classification 32,000 instances 123 dims</td>
<td>14.9% 9 secs $D = 500$</td>
<td>15.3% 1.5 mins $P = 30$</td>
<td>14.8% 73 mins</td>
<td>15.1% 7 mins SVM^light</td>
</tr>
<tr>
<td>Forest Cover classification 522,000 instances 54 dims</td>
<td>11.6% 71 mins $D = 5000$</td>
<td>2.2% 25 mins $P = 50$</td>
<td>2.3% 7.5 hrs</td>
<td>2.2% 44 hrs libSVM</td>
</tr>
<tr>
<td>KDDCUP99 (see footnote) classification 4,900,000 instances 127 dims</td>
<td>7.3% 1.5 min $D = 50$</td>
<td>7.3% 35 mins $P = 10$</td>
<td>6.2% (18%) 1.4 secs (20 secs)</td>
<td>8.3% &lt; 1 s SVM+sampling</td>
</tr>
</tbody>
</table>

Despite its simplicity, ridge regression with random features is faster than and provides competitive accuracy with alternative methods. It also produces very compact functions. The table compares the performance of ridge regression with random features, Core Vector Machine (CVM), and Exact SVM on various datasets. Despite its simplicity, ridge regression with random features is faster than and provides competitive accuracy with alternative methods. It also produces very compact functions.

The figure on the left illustrates the benefit of training classifiers on larger datasets, where accuracy can be improved. The figure in the middle shows the error rate as a function of $P$. The figure on the right shows the training and testing time as a function of $P$. The figure demonstrates that the error rate decreases as $P$ increases, and the training and testing time increases as $P$ increases.