Laplace operators and regularization
Fast algorithms for translation invariant and dot product kernels

Alexander Smola
alex@smola.org

Scalable Machine Learning Laboratory, CMU
Strategic Technologies, Google
Kernels and Regularization
Translation Group
• Laplace operator and RBF kernels
• Random Kitchen Sinks
• Fastfood
Rotation Group SO(n)
• Laplace-Beltrami and dot product kernels
• Fastfood
Graphs
• Graph Laplacian
• Approximate expansions
Kernels and Regularization
Risk minimization

• Data source
  \[(x, y) \sim p(x, y) \text{ drawn iid}\]

• Loss function (how well we do)
  \[l(x, y, f) \text{ such as } \frac{1}{2} (y - f(x))^2\]

• Training data drawn iid from \(p\)

• Goal: perform well in expectation
  \[R[f] := \mathbb{E}_{(x, y) \sim p} [l(x, y, f)]\]

• Realistic goal: perform well on training set
  \[R_{\text{emp}}[f] := \frac{1}{m} \sum_{i=1}^{m} l(x_i, y_i, f)\]
Regularization

• Problem
  If function class is too powerful, we overfit
  \[
  \min_{f \in \mathcal{F}} R_{\text{emp}}[f]
  \]

• Solutions
  • Restrict function class to grow with sample
    \[
    \min_{f} R_{\text{emp}}[f] \text{ subject to } \|f\| \leq F
    \]
  • Regularized risk functional (add penalty)
    \[
    \min_{f} R_{\text{emp}}[f] + \lambda \|f\|^p
    \]
  • Guarantees from learning theory / MAP Bayes
Regularization

• Problem
  If function class is too powerful, we overfit

• Solutions
  • Restrict function class to grow with sample
    \[
    \minimize_{f} R_{\text{emp}}[f] \text{ subject to } \|f\| \leq F
    \]
  • Regularized risk functional (add penalty)
    \[
    \minimize_{f} R_{\text{emp}}[f] + \lambda \|f\|^p
    \]
  • Guarantees from learning theory / MAP Bayes

more convex regularizers
more Banach Spaces
Regularization Operators

• Want to measure smoothness / simplicity
• $L_2$ norm of function $\|f\|^2_2 = \langle f, f \rangle$
• Laplace operator

$$
\|\partial_x f\|^2 = \sum_{i=1}^{d} \|\partial_i f\|^2_2 = \left\langle f, \sum_{i=1}^{d} \partial_i^2 f \right\rangle =: \langle f, \Delta f \rangle
$$

• PSD Self-adjoint operator $\langle f, Pf \rangle$
• Function expansion (regularization nets)

$$
f = \sum_i \alpha_i f_i \text{ hence } \langle f, Pf \rangle = \sum_{i,j} \alpha_i \alpha_j \langle f_i, Pf_j \rangle
$$
• Want to measure smoothness / simplicity
  • $L_2$ norm of function $\|f\|_2^2 = \langle f, f \rangle$
  • Laplace operator

$$\| \partial_x f \|_2^2 = \sum_{i=1}^d \| \partial_i f \|_2^2 = \left\langle f, \sum_{i=1}^d \partial_i^2 f \right\rangle =: \langle f, \Delta f \rangle$$

• PSD Self-adjoint operator

• Function expansion (regularization nets)

$$f = \sum_i \alpha_i f_i \text{ hence } \langle f, Pf \rangle = \sum_{i,j} \alpha_i \alpha_j \langle f_i, Pf_j \rangle$$

looks like a kernel expansion
Kernels

• Problems
  • Computing $\langle f_i, Pf_j \rangle$ for many $f_i$ is costly.
  • How many functions should we pick?
  • Properties of the function space

• Solution
  Pick $f$ and $P$ with reproducing property
  \[ k_x(x') = \langle k_x, P k_{x'} \rangle \]
  Since $P$ is self-adjoint we have
  \[ k_x(x') = k(x, x') = k(x', x) \]
Kernels

- Problems
  - Computing $\langle f_i, Pf_j \rangle$ for many $f_i$ is costly.
  - How many functions should we pick?
  - Properties of the function space
- Solution
  Pick $f$ and $P$ with reproducing property
  $$k_x(x') = \langle k_x, P k_{x'} \rangle$$
  Since $P$ is self-adjoint we have
  $$k_x(x') = k(x, x') = k(x', x)$$
Kernels

- Problems
  - Computing $\langle f_i, Pf_j \rangle$ for many $f_i$ is costly.
  - How many functions should we pick?
  - Properties of the function space

Solution
Pick $f$ and $P$ with reproducing property

Since $P$ is self-adjoint we have

$$k_x(x') = \langle k_x, Pk_x' \rangle$$

Find matching pairs of $k$ and $P$

Analyze the properties of $P$
Shift Invariant Kernels
Laplace Operator

- Operator
  \[ \Delta f = \sum_{i=1}^{d} \partial_i^2 f \]
- Self-adjoint and positive semidefinite
- Translation invariant (commutes with shift)
  \[ \Delta(T_z f) = T_z(\Delta f) \]
- In physics used for diffusion \( \Delta f = 0 \)
- Fourier transform diagonalizes operator
  \[ F[\Delta f](\omega) = ||\omega||^2 F[f](\omega) \]
Designing a kernel

• Want small functions and smoothness, e.g.
  \[ \langle f, Pf \rangle := \langle f, f \rangle + \langle f, \Delta f \rangle \]

• Need to find \( k \) with
  \[ k(x, x') = \langle k(x, \cdot), Pk(x', \cdot) \rangle \]

• Idea - Translation invariance of \( P \) - use ansatz
  \[ k(x, x') = k(x - x') \]

• Idea - Solve in Fourier domain
  \[
  k(x - x') = F^{-1} \left[ e^{-i\langle \omega, x \rangle} \overline{F[k]}(\omega) \left( 1 + \|\omega\|^2 \right) e^{i\langle \omega, x' \rangle} F[k](\omega) \right] \\
  = F^{-1} \left[ e^{-i\langle \omega, x-x' \rangle} \left( 1 + \|\omega\|^2 \right) |F[k](\omega)|^2 \right]
  \]
Designing a kernel

- Solve in Fourier domain

\[ k(x - x') = F^{-1} \left[ e^{-i\langle \omega, x-x' \rangle} \left( 1 + \|\omega\|^2 \right) |F[k](\omega)|^2 \right] \]

via

\[ F[k](\omega) = \left( 1 + \|\omega\|^2 \right)^{-1} \]

- In \( \mathbb{R}^1 \) this yields \( k(x - x') = \exp(-|x - x'|) \)
General Strategy

- Translation invariant P
  
  \[ k(x - x') = F^{-1} \left[ e^{-i\langle \omega, x-x' \rangle} \rho(\|\omega\|^{2}) |F[k](\omega)|^{2} \right] \]

- Series expansion via Laplace operator
  
  \[ P = \sum_{i} c_{i} \Delta^{i} \text{ and hence } \rho(\|\omega\|^{2}) = \sum_{i} c_{i} \|\omega\|^{2i} \]

- Kernel via
  
  \[ F[k](\omega) = 1/\rho(\|\omega\|^{2}) \]

- Example - Gaussian kernel
  
  \[ k(x - x') = e^{-\frac{1}{2} \sigma^{-2} \|x-x'\|^{2}} \]

  \[ \rho(\sigma) = e^{\frac{1}{2} \sigma^{2} \|\omega\|^{2}} \]

  \[ P = \sum_{n} 2^{-n} \frac{\sigma^{2n}}{2^{n}n!} \Delta^{n} \]
The trouble with kernels
The trouble with kernels

- Representer theorem \( f(x) = \sum_{i=1}^{m} \alpha_i k(x_i, x) \)
- Number of basis functions increases linearly with sample size

**naive approach**
The trouble with kernels

- Representer theorem: $f(x) = \sum_{i=1}^{m} \alpha_i k(x_i, x)$
- Number of basis functions increases linearly with sample size

$K_{mm}$

$K_{nm'}$

Reduced set

The trouble with kernels

- Representer theorem: \( f(x) = \sum_{i=1}^{m} \alpha_i k(x_i, x) \)
- Number of basis functions increases linearly with sample size

\[
\begin{align*}
K_{mn} & \quad K_{nm}^{-1} & \quad K_{nm} \\
\text{training} & \quad \text{testing} & \\
\text{reduced rank}
\end{align*}
\]

Williams & Seeger, 2001, Smola & Schölkopf, 2001
Scheinberg & Fine 2004, Kempe & Das 2010
The trouble with kernels

- Back to primal space
- Number of basis functions (should) increase with sample size
- Explicit formulation not always possible (and kernels in infinite dimensions)

\[ f(x) = \sum_{i=1}^{n} w_i \phi_i(x) \]

\[ X_{mn} \quad \text{training} \]
\[ X_{m'n} \quad \text{testing} \]

random kitchen sinks

The trouble with kernels

- Kernel expansion
- Function expansion
- Slow approximations (d dimensions, n features, m samples)

<table>
<thead>
<tr>
<th></th>
<th>CPU Training</th>
<th>CPU Test</th>
<th>RAM Training</th>
<th>RAM Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive</td>
<td>$O(m^2d)$</td>
<td>$O(md)$</td>
<td>$O(md)$</td>
<td>$O(md)$</td>
</tr>
<tr>
<td>Reduced set</td>
<td>$O(m^2d)$</td>
<td>$O(nd)$</td>
<td>$O(md)$</td>
<td>$O(nd)$</td>
</tr>
<tr>
<td>Low rank</td>
<td>$O(mnd)$</td>
<td>$O(nd)$</td>
<td>$O(nd)$</td>
<td>$O(nd)$</td>
</tr>
<tr>
<td>Random Kitchen Sinks</td>
<td>$O(mnd)$</td>
<td>$O(nd)$</td>
<td>$O(nd)$</td>
<td>$O(nd)$</td>
</tr>
</tbody>
</table>
# The trouble with kernels

- Kernel expansion
- Function expansion
- Slow approximations (d dimensions, n features, m samples)

<table>
<thead>
<tr>
<th></th>
<th>CPU Training</th>
<th>CPU Test</th>
<th>RAM Training</th>
<th>RAM Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive</td>
<td>$O(m^2d)$</td>
<td>$O(md)$</td>
<td>$O(md)$</td>
<td>$O(md)$</td>
</tr>
<tr>
<td>Reduced set</td>
<td>$O(m^2d)$</td>
<td>$O(nd)$</td>
<td>$O(md)$</td>
<td>$O(nd)$</td>
</tr>
<tr>
<td>Low rank</td>
<td>$O(mnd)$</td>
<td>$O(nd)$</td>
<td>$O(nd)$</td>
<td>$O(nd)$</td>
</tr>
<tr>
<td>Random Kitchen Sinks</td>
<td>$O(mnd)$</td>
<td>$O(nd)$</td>
<td>$O(nd)$</td>
<td>$O(nd)$</td>
</tr>
<tr>
<td>Fastfood</td>
<td>$O(mn \log d)$</td>
<td>$O(n \log d)$</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
</tbody>
</table>
Random Kitchen Sinks & Fastfood
Random Kitchen Sinks
(Rahimi & Recht 2008, 2009)

- Translation invariant kernel representation
  \[ k(x, x') = k(\|x - x'\|) = \int d\omega \kappa(\omega) e^{i\langle \omega, x \rangle} e^{-i\langle \omega, x' \rangle} \]

- Draw frequencies from \( k(w) \)
  \[ k(x, x') = \frac{1}{n} \sum_{j=1}^{n} e^{i\langle \omega_j, x \rangle} e^{-i\langle \omega_j, x' \rangle} \text{ where } \omega_j \sim \kappa(\omega) \]

- For Gaussian RBFs draw \( w_j \) from \( \mathcal{N}(0, 1) \)
- For general \( k \) draw \( w \) from spherically symmetric distribution (and rescale)
Random Kitchen Sinks
(Rahimi & Recht 2008, 2009)

• Sample and store frequencies \( w_i \)

• Multiply frequency matrix \( W \) by data \( x \)

• Compute (sin, cos) of \( Wx \)

• Use primal space solver

• \( Wx \) costs \( O(nd) \) CPU and MEMORY

\[
k(x, x') = \frac{1}{n} \sum_{j=1}^{n} e^{i\omega_j x} e^{-i\omega_j x'} \text{ where } \omega_j \sim \kappa(\omega)
\]

• Accelerate this. Smaller memory footprint
Why Gauss is bad for you

• Concentration of measure for draws from Gaussian
• Mode in d dimensions at $\sqrt{d - 1}$ for $r^{d-1}e^{-\frac{1}{2}r^2}$
• Explains why Gaussian RBF not great for large d
Fastfood

- Gaussian matrix $\Omega$ costs $O(nd)$ per multiplication
- $O(nd)$ for storage on a random matrix
- $O(nd)$ computation for multiplying by a random matrix
- Fake Gaussian (for the moment assume square matrices)

$$\tilde{M} = S H G \Pi H B$$

- Fast & cheap: multiplication is $O(d \log d)$, storage is $O(d)$. 

---

spectrum isotropy decorrelation

<table>
<thead>
<tr>
<th>O(d)</th>
<th>O(d log d)</th>
<th>O(d)</th>
<th>O(d)</th>
<th>O(d log d)</th>
<th>O(d)</th>
</tr>
</thead>
</table>

- Spectrum
- Isotropy
- Decorrelation
Fastfood

• Gaussian matrix $\Omega$ costs $O(nd)$ per multiplication
  • $O(nd)$ for storage on a random matrix
  • $O(nd)$ computation for multiplying by a random matrix
• Fake Gaussian (for the moment assume square matrices)

\[
\tilde{M} = SHG\Pi HB
\]

• $S$ is random diagonal scaling matrix (deals with spectrum)
• $H$ is Hadamard matrix admitting $O(d \log d)$ multiply

\[
H_{2n} = \begin{bmatrix} H_n & H_n \\ H_n & -H_n \end{bmatrix}
\mathrm{and} \ H_1 = 1
\]

• $G$ is random diagonal Gaussian matrix
• $\Pi$ is random permutation matrix
• $B$ is random binary $\{-1, 1\}$ diagonal matrix
• Fast & cheap: multiplication is $O(d \log d)$, storage is $O(d)$. 
Fastfood - is it safe?

• It’s fast but is it good? \( \tilde{M} = SHG\Pi HB \)
• Theorem - In expectation the kernel is correct.
• Proof - Each row of M (on its own) is Gaussian.
• Corollary - Other kernels by different rescaling S
• Example - Matern kernel

\[
k(x, x') = \|x - x'\|^{-\frac{nd}{2}} J_{\frac{n}{2}}(\|x - x'\|)
\]

Fourier transform is n-fold convolution of unit ball.
(to sample S draw n independent points and add)
Fastfood - is it really safe?

• Bounds on the variance \((v = x - x')\)

\[
\text{Var} [\psi_j(v)] = \frac{1}{2} \left( 1 - e^{-\|v\|^2} \right)^2
\]

\[
\text{Var} \left[ \sum_{j=1}^{d} \psi_j(v) \right] \leq \frac{d}{2} \left( 1 - e^{-\|v\|^2} \right)^2 + dC(\|v\|)
\]

where \(C(\alpha) = 6\alpha^4 \left[ e^{-\alpha^2} + \frac{\alpha^2}{3} \right] \).

• Things are very well behaved as long as features are properly rescaled

• Proof by brute force integration (similar to Dasgupta et al., 2012)
Spherical Symmetry
Laplace-Beltrami Operator

- Laplace Operator
  \[ \Delta f = \sum_{i=1}^{d} \partial_i^2 f \]

- Restriction to manifold is Laplace-Beltrami
  \[ \Delta f = \text{div grad } f \]

- On sphere Laplace operator decomposes
  \[ \Delta f = r^{1-n} \frac{\partial}{\partial r} \left( r^{n-1} \frac{\partial f}{\partial r} \right) + r^{-2} \Delta_{S^{n-1}} f. \]
Properties

- Homogeneous polynomials, e.g. Legendre polynomials \( L_{n,d}(\langle z, x \rangle) \) are eigenfunctions.
- Eigenvalues are \( d(d+n-1) \).
- Multiplicity is \( \binom{n+d}{n} \).
- Laplace Beltrami commutes with \( SU(n) \).
- Express smoothness in terms of L-B operator:

\[
k(x, x') = \sum_d c_d L_{n,d}(\langle x, x' \rangle)
\]

\[
\langle f, g(\Delta) f \rangle \text{ with } g(k(k+n-1)) = \binom{n+d}{n} \Omega_{n-1}^{-1} c_n^{-1}
\]
Dot product kernels on the sphere

- Symmetry group are rotations
- Spherical Harmonics diagonalize SO(n)

\[ k(x, x') = \sum_d c_d \int_{S_d} L_{n,d}(\langle x, v \rangle)L_{n,d}(\langle x', v \rangle) dv \]

- Random Kitchen Sinks for dot-product kernels
  - Sample degree using weights \( c_d \)
  - Draw uniformly from \( v \)
- Fastfood: replace uniform \( v \) by \( \tilde{M} = SHG\Pi HB \)
General Theory

- Kernel invariant under group action
  
  \[ k(x, x') = k(\rho \circ x, \rho \circ x') \text{ for all } \rho \in G \]

- Expansion in irreducible representations of symmetry group
  
  \[ k(x, x') = \int \lambda_z \phi_z(x) \phi_z(x') \text{ where } \phi_z(\rho \circ x) = U_{\rho \rho_r(z)}(x) \]

- Radial basis function kernels in Fourier expansion (Bochner 1932)
  
  \[ k(x, x') = k(\|x - x'\|) = \int d\omega \kappa(\omega) e^{i \langle \omega, x \rangle} e^{-i \langle \omega, x' \rangle} \]

- Inner product kernels in spherical harmonics (Schoenberg 1934)
  
  \[ k(x, x') = k(\langle x, x' \rangle) = \sum_{jlm} c_{jlm} Y_{j\ell}(\frac{x}{\|x\|}) \bar{Y}_{j\ell}(\frac{x'}{\|x'\|}) (\|x\| \|x'\|)^m \]

- Symmetric group analogous
The proof is in the pudding
Matrix approximation error

![Matrix approximation error graph](image)

- **Random Kitchen Sinks**
- **Fastfood (Fourier features)**
- **Fastfood (Hadamard features)**

**Number of basis functions**
- **FFT**
Generalization Performance
(UCI CPU dataset)

![Graph showing Generalization Performance](image)

- **Random Kitchen Sinks**
- **Fastfood (Fourier features)**
- **Fastfood (Hadamard features)**

The graph compares the performance of different kernel approximation methods as a function of the number of basis functions. It illustrates how these methods achieve lower mean squared error (MSE) with increasing numbers of basis functions, highlighting the advantages of Fastfood methods.

**Figure 1.** Uses a variant of the above construction. Instead of the random matrix given by Recht et al. ([2009](#)), we use a Hilbert matrix (i.e., with the random matrix given by \( \frac{1}{\sqrt{2}} \)). This demonstrates that it is necessary to have a large subset of columns from a (unitary) Fourier matrix (i.e., with the random matrix given by \( \frac{1}{\sqrt{2}} \)).

**Figure 2.** Comparison of different kernel approximation methods in terms of speed. Fastfood and Random Kitchen Sinks are faster than other variants and improve as the number of basis functions increases. The key point is that the Nystrom method should have a significant accuracy advantage over Random Kitchen Sinks. The results of the comparison are given in Table 1.

The intuition is that by picking a random subset of columns from a (unitary) Fourier matrix, we end up with vectors that are almost sparse. This is on par with exact kernel computation, the Nystrom method is as accurate. Hence, the concentration of measure effect is present regardless of dataset size. Interestingly, although Fourier features do not seem to approximate the kernel RBF function it performs surprisingly well. This indicates that the concentration of measure effect be taken into account.

In the previous experiments, we observe that Fastfood is much faster than other methods. In Figure 2, we also show regression performances as a function of number of basis functions \( n \) increases. As can be seen, there is virtually no difference between the exact kernel, the Nystrom approximation, and Gaussian scaling, we use a permutation matrix, and Gaussian scaling, we use a permutation matrix. The results of the comparison are given in Table 1.
In the previous experiments, we observe that Fastfood, our proposed method, outperforms Random Kitchen Sinks. The key point, Fastfood — Computing Hilbert Space Expansions in loglinear time. Random Kitchen Sinks achieve 62.4% with an expansion of a vector with dimension $d=3072$. In our experiments, linear SVMs achieve 42.3% accuracy on the test set. Non-linear expansions of random localized basis functions, since the convolution of two Gaussians is again a Gaussian. Note also that fast multiplication with a matrix of $n$-dimensional vectors is $O(dn^2)$, whereas Fastfood can compute the label for one input vector in $O(nd)$. Hence, instead of an expensive dot product in the input space, Fastfood can use the method, and Random Kitchen Sinks, and our method using Spiral for permutation invariant classifiers. These are also best known classification accuracies in order to learn highly nonlinear functions.

**Table: Speed & accuracy**

<table>
<thead>
<tr>
<th>Dataset</th>
<th>$m$</th>
<th>$d$</th>
<th>Exact</th>
<th>Nystrom</th>
<th>Random Kitchen Sinks</th>
<th>Fastfood FFT</th>
<th>Fastfood</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insurance Company (COIL2000)</td>
<td>5,822</td>
<td>85</td>
<td>0.231</td>
<td>0.232</td>
<td>0.266</td>
<td>0.266</td>
<td>0.264</td>
</tr>
<tr>
<td>Wine Quality</td>
<td>4,080</td>
<td>11</td>
<td>0.819</td>
<td>0.797</td>
<td>0.740</td>
<td>0.721</td>
<td>0.740</td>
</tr>
<tr>
<td>Parkinson Telemonitor</td>
<td>4,700</td>
<td>21</td>
<td>0.059</td>
<td>0.058</td>
<td>0.054</td>
<td>0.052</td>
<td>0.054</td>
</tr>
<tr>
<td>CPU</td>
<td>6,554</td>
<td>21</td>
<td>7.271</td>
<td>6.758</td>
<td>7.103</td>
<td>4.544</td>
<td>7.366</td>
</tr>
<tr>
<td>Relative location of CT slices (axial)</td>
<td>42,800</td>
<td>384</td>
<td>n.a.</td>
<td>60.683</td>
<td>49.491</td>
<td>58.425</td>
<td>43.858</td>
</tr>
<tr>
<td>KEGG Metabolic Reaction Network</td>
<td>51,686</td>
<td>27</td>
<td>n.a.</td>
<td>17.872</td>
<td>17.837</td>
<td>17.826</td>
<td>17.818</td>
</tr>
<tr>
<td>Year Prediction MSD</td>
<td>463,715</td>
<td>90</td>
<td>n.a.</td>
<td>0.113</td>
<td>0.123</td>
<td>0.106</td>
<td>0.115</td>
</tr>
<tr>
<td>Forest</td>
<td>522,910</td>
<td>54</td>
<td>n.a.</td>
<td>0.837</td>
<td>0.840</td>
<td>0.838</td>
<td>0.840</td>
</tr>
</tbody>
</table>

**Table: Speed & accuracy**

<table>
<thead>
<tr>
<th>$d$</th>
<th>$n$</th>
<th>Fastfood</th>
<th>RKS</th>
<th>Speedup</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1,024</td>
<td>16,384</td>
<td>0.00058s</td>
<td>0.0139s</td>
<td>24x</td>
<td>256x</td>
</tr>
<tr>
<td>4,096</td>
<td>32,768</td>
<td>0.00136s</td>
<td>0.1224s</td>
<td>90x</td>
<td>1024x</td>
</tr>
<tr>
<td>8,192</td>
<td>65,536</td>
<td>0.00268s</td>
<td>0.5360s</td>
<td>200x</td>
<td>2048x</td>
</tr>
</tbody>
</table>

State of the art on CIFAR-10 (63%) for permutation invariant classifiers.
<table>
<thead>
<tr>
<th>Dataset</th>
<th>Kitchen Sinks</th>
<th>Random Sinks</th>
<th>Fastfood FFT</th>
<th>Fastfood</th>
<th>Fastfood Matern</th>
</tr>
</thead>
<tbody>
<tr>
<td>Year Prediction MS</td>
<td>0.986</td>
<td>0.984</td>
<td>0.978</td>
<td>0.975</td>
<td>0.976</td>
</tr>
<tr>
<td>Parkinson Telemonitor</td>
<td>0.052</td>
<td>0.054</td>
<td>0.054</td>
<td>0.052</td>
<td>0.052</td>
</tr>
<tr>
<td>Wine Quality</td>
<td>0.052</td>
<td>0.054</td>
<td>0.054</td>
<td>0.052</td>
<td>0.052</td>
</tr>
<tr>
<td>Insurance Company COIL2000</td>
<td>0.052</td>
<td>0.054</td>
<td>0.054</td>
<td>0.052</td>
<td>0.052</td>
</tr>
</tbody>
</table>

Note: The table above compares the performance of different kernel computation methods. We can see Fastfood methods perform comparably.
• Extensible to other kernels
  • Easy to sample spectral distribution
  • Easy to learn ‘multiple kernels’ since dimensions are so cheap
• Extensible to localized basis functions
  • Instantiate Neal’s 1994 paper (Gaussian Process - Neural Network equivalence)
• Matrix valued functions
• Never store explicit feature map (requires memory)
ZOMBIES HATE FAST FOOD
References

- Regularization theory and neural networks architectures
  F Girosi, M Jones, T Poggio - Neural computation, 1995
- The connection between regularization operators and support vector kernels
- Special functions of mathematical physics
  H Hochstadt, 1961
- Weighted sums of random kitchen sinks: Replacing minimization with randomization in learning
  A Rahimi, B Recht, NIPS 2008
- Fastfood-computing hilbert space expansions in loglinear time
  Q Le, T Sarlos, A Smola, ICML 2013
- Regularization with dot-product kernels
  AJ Smola, ZL Ovari, RC Williamson, NIPS 2001