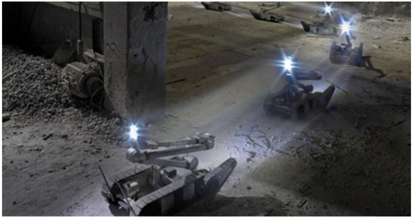
Decentralized Efficient Nonparametric Stochastic Optimization

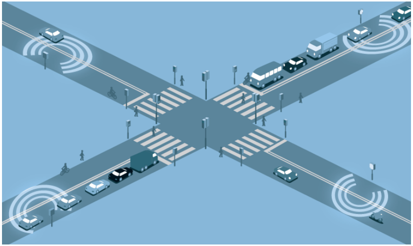
Alec Koppel*, Santiago Paternain †, Cédric Richard§, Alejandro Ribeiro†
*U.S. Army Research Laboratory, Adelphi, MD
†University of Pennsylvania
§ Laboratoire Lagrange at the University of Nice Sophia-Antipolis.

GlobalSIP, November 16, 2017, Montreal, Canada.

Distributed Learning

- Network of agents $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ aims to make inferences from data
- Sensor Networks, multi-robot teams, internet of things
- For instance, distributed training of a classifier for some data set





A Centralized Solution

- $(\mathbf{x}, \mathbf{y}) \in \mathcal{X} \times \mathcal{Y}$ is random pair $\Rightarrow$ training examples
- $\ell : \mathcal{W} \to \mathbb{R}$ convex loss ($\mathcal{W} \subset \mathbb{R}^p$), merit of statistical model
- Find parameters $\mathbf{w}^* \in \mathbb{R}^p$ that minimize expected risk $L(\mathbf{w})$

$$\mathbf{w}^* := \operatorname*{argmin}_{\mathbf{w}} L(\mathbf{w}) := \operatorname*{argmin}_{\mathbf{w}} \mathbb{E}_{\mathbf{x},\mathbf{y}}[\ell(\mathbf{w}^\top \mathbf{x}, \mathbf{y})]$$

- Convex Optimization Problem for *linear statistical models*

  $\Rightarrow$ e.g., $y = \mathbf{w}^T \mathbf{x} \in \mathbb{R}$ or $y = \text{sgn}(\mathbf{w}^T \mathbf{x}) \in \{-1, 1\}$
- Solve with favorite descent method $\Rightarrow$ Good Performance

Easy to Implement over Networks

- Each agent i has a local copy of the classifier $\mathbf{w}_i$ with $i = 1 \ldots |\mathcal{V}|$
  $\Rightarrow$ Observes some training examples $\Rightarrow (\mathbf{x}, \mathbf{y}) \in \mathcal{X}_i \times \mathcal{Y}_i$

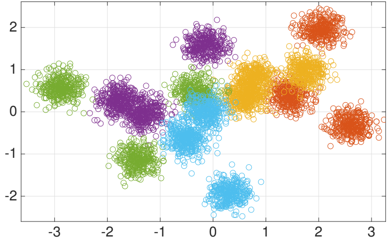
$$\mathbf{w}^* := \underset{\mathbf{w} \in \mathbb{R}^{p|\mathcal{V}|}}{\text{argmin}} \sum_{i=1}^{|\mathcal{V}|} \mathbb{E}_{\mathbf{x}_i, y_i} \left[\ell(\mathbf{w}_i^\top \mathbf{x}_i, \mathbf{y}_i) \right]$$

$$s.t. \quad \mathbf{w}_i = \mathbf{w}_j \quad \text{for all} \quad j \in \mathcal{N}_i$$

- Convex Optimization Problem for *linear statistical models*
- Solve with saddle point algorithms or penalty methods
  $\Rightarrow$ Can be implemented in a distributed fashion

Data is not (always) linear

- The statistical model of complex data sets is nonlinear



- Neural Newtworks or Kernel Methods in centralized solution
- In this talk we focus on Distributed Kernel Methods

Large-Scale Function Estimation

- Learning nonlinear statistical models is equivalent to finding
  $\Rightarrow f : \mathcal{X} \to \mathcal{Y}$, such that $y = f(\mathbf{x})$
- Want to find $f^* \in \mathcal{H}$ to minimize regularized expected risk

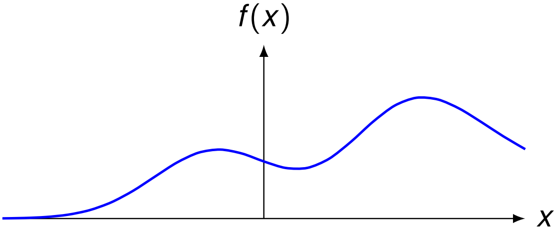
$$f^* = \operatorname*{argmin}_{f \in \mathcal{H}} \mathbb{E}_{\mathbf{x},\mathbf{y}}[\ell(f(\mathbf{x}), y)] + \frac{\lambda}{2}\|f\|^2_{\mathcal{H}}$$

  $\Rightarrow$ Loss $\ell : \mathcal{H} \times \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ penalize deviations between $f(\mathbf{x})$, $\mathbf{y}$
- In general, function estimation is intractable
  $\Rightarrow$ infinite dimensional data-dependent optimization
- Reproducing kernels $\Rightarrow$ framework to make this task possible!

Large-Scale Function Estimation

- Equip $\mathcal{H}$ with a unique *kernel function*, $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, such that:

$$(i)\ \langle f, \kappa(\mathbf{x}, \cdot)\rangle_{\mathcal{H}} = f(\mathbf{x}) \quad \text{for all } \mathbf{x} \in \mathcal{X}\ ,$$
$$(ii)\ \mathcal{H} = \overline{\text{span}\{\kappa(\mathbf{x}, \cdot)\}} \quad \text{for all } \mathbf{x} \in \mathcal{X}\ .$$



- Property (i) $\Rightarrow$ Will allow us to compute derivatives
- Kernel examples:

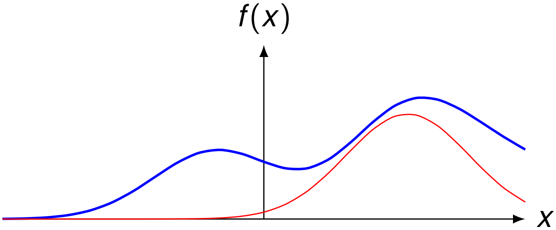
$\Rightarrow$ Gaussian/RBF $\kappa(\mathbf{x}, \mathbf{x}') = \exp\left\{-\frac{\|\mathbf{x}-\mathbf{x}'\|_2^2}{2c^2}\right\}$

$\Rightarrow$ polynomial $\kappa(\mathbf{x}, \mathbf{x}') = \left(\mathbf{x}^T\mathbf{x}' + b\right)^c$

Large-Scale Function Estimation

- Equip $\mathcal{H}$ with a unique *kernel function*, $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, such that:

$$(i)\ \langle f, \kappa(\mathbf{x}, \cdot)\rangle_{\mathcal{H}} = f(\mathbf{x}) \quad \text{for all } \mathbf{x} \in \mathcal{X}\ ,$$
$$(ii)\ \mathcal{H} = \overline{\text{span}\{\kappa(\mathbf{x}, \cdot)\}} \quad \text{for all } \mathbf{x} \in \mathcal{X}\ .$$



- Property (i) $\Rightarrow$ Will allow us to compute derivatives
- Kernel examples:

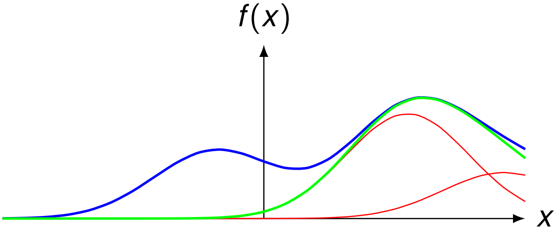
$\Rightarrow$ Gaussian/RBF $\kappa(\mathbf{x}, \mathbf{x}') = \exp\left\{-\frac{\|\mathbf{x}-\mathbf{x}'\|_2^2}{2c^2}\right\}$

$\Rightarrow$ polynomial $\kappa(\mathbf{x}, \mathbf{x}') = \left(\mathbf{x}^T\mathbf{x}' + b\right)^c$

Large-Scale Function Estimation

- Equip $\mathcal{H}$ with a unique *kernel function*, $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, such that:

$$(i)\ \langle f, \kappa(\mathbf{x}, \cdot)\rangle_{\mathcal{H}} = f(\mathbf{x}) \quad \text{for all } \mathbf{x} \in \mathcal{X}\ ,$$
$$(ii)\ \mathcal{H} = \overline{\text{span}\{\kappa(\mathbf{x}, \cdot)\}} \quad \text{for all } \mathbf{x} \in \mathcal{X}\ .$$

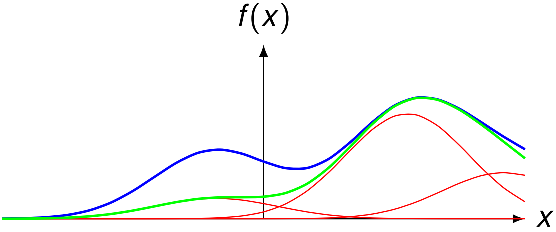


- Property (i) $\Rightarrow$ Will allow us to compute derivatives
- Kernel examples:
  - $\Rightarrow$ Gaussian/RBF $\kappa(\mathbf{x}, \mathbf{x}') = \exp\left\{-\frac{\|\mathbf{x}-\mathbf{x}'\|_2^2}{2c^2}\right\}$
  - $\Rightarrow$ polynomial $\kappa(\mathbf{x}, \mathbf{x}') = \left(\mathbf{x}^T\mathbf{x}' + b\right)^c$

Large-Scale Function Estimation

- Equip $\mathcal{H}$ with a unique *kernel function*, $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, such that:

$$(i)\ \langle f, \kappa(\mathbf{x}, \cdot)\rangle_{\mathcal{H}} = f(\mathbf{x}) \quad \text{for all } \mathbf{x} \in \mathcal{X}\ ,$$
$$(ii)\ \mathcal{H} = \overline{\text{span}\{\kappa(\mathbf{x}, \cdot)\}} \quad \text{for all } \mathbf{x} \in \mathcal{X}\ .$$



- Property (i) $\Rightarrow$ Will allow us to compute derivatives
- Kernel examples:

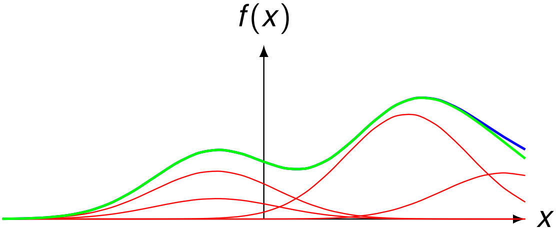
$\Rightarrow$ Gaussian/RBF $\kappa(\mathbf{x}, \mathbf{x}') = \exp\left\{-\frac{\|\mathbf{x}-\mathbf{x}'\|_2^2}{2c^2}\right\}$

$\Rightarrow$ polynomial $\kappa(\mathbf{x}, \mathbf{x}') = \left(\mathbf{x}^T\mathbf{x}' + b\right)^c$

Large-Scale Function Estimation

- Equip $\mathcal{H}$ with a unique *kernel function*, $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, such that:

$$(i)\ \langle f, \kappa(\mathbf{x}, \cdot)\rangle_{\mathcal{H}} = f(\mathbf{x}) \quad \text{for all } \mathbf{x} \in \mathcal{X}\ ,$$
$$(ii)\ \mathcal{H} = \overline{\text{span}\{\kappa(\mathbf{x}, \cdot)\}} \quad \text{for all } \mathbf{x} \in \mathcal{X}\ .$$



- Property (i) $\Rightarrow$ Will allow us to compute derivatives
- Kernel examples:

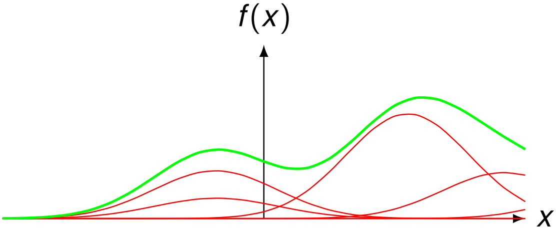
$\Rightarrow$ Gaussian/RBF $\kappa(\mathbf{x}, \mathbf{x}') = \exp\left\{-\frac{\|\mathbf{x}-\mathbf{x}'\|_2^2}{2c^2}\right\}$

$\Rightarrow$ polynomial $\kappa(\mathbf{x}, \mathbf{x}') = \left(\mathbf{x}^T\mathbf{x}' + b\right)^c$

Large-Scale Function Estimation

- Equip $\mathcal{H}$ with a unique *kernel function*, $\kappa : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$, such that:

$$(i)\ \langle f, \kappa(\mathbf{x}, \cdot)\rangle_{\mathcal{H}} = f(\mathbf{x}) \quad \text{for all } \mathbf{x} \in \mathcal{X}\ ,$$
$$(ii)\ \mathcal{H} = \overline{\text{span}\{\kappa(\mathbf{x}, \cdot)\}} \quad \text{for all } \mathbf{x} \in \mathcal{X}\ .$$



- Property (i) $\Rightarrow$ Will allow us to compute derivatives
- Kernel examples:

$\Rightarrow$ Gaussian/RBF $\kappa(\mathbf{x}, \mathbf{x}') = \exp\left\{-\frac{\|\mathbf{x}-\mathbf{x}'\|_2^2}{2c^2}\right\}$

$\Rightarrow$ polynomial $\kappa(\mathbf{x}, \mathbf{x}') = \left(\mathbf{x}^T\mathbf{x}' + b\right)^c$

Function Representation

- Consider empirical risk minimization case: sample size $N < \infty$
- Representer Theorem:

$$f^* = \underset{f}{\text{argmin}} \frac{1}{N} \sum_{n=1}^{N} \ell(f(\mathbf{x}_n), y_n) + \frac{\lambda}{2} \|f\|_{\mathcal{H}}^2 = \sum_{m=1}^{N} w_m^* \, \kappa(\mathbf{x}_m, \mathbf{x}) \, .$$

- Representer Thm. into ERM $\Rightarrow$ opt. over $\mathcal{H}$ reduces to $\mathbf{w} \in \mathbb{R}^N$

$$f^* = \underset{\mathbf{w} \in \mathbb{R}^N}{\text{argmin}} \frac{1}{N} \sum_{n=1}^{N} \ell\Big(\sum_{m=1}^{N} w_m \kappa(\mathbf{x}_m, \mathbf{x}_n), y_n\Big) + \frac{\lambda}{2} \sum_{n,m=1}^{N} w_n w_m \kappa(\mathbf{x}_m, \mathbf{x}_n)$$

- Reduces to solve a convex optimization problem of dimension N.
- As $N \to \infty$ storage and computation issues are present
  $\Rightarrow$ This is known as the Curse of Kernelization

Distributed Function Estimation

- Each agents has a local copy $f_i \in \mathcal{H}$ with $i = 1 \ldots |\mathcal{V}|$
- Define the stacked function $f = [f_1, f_2, \ldots f_{|\mathcal{V}|}]^\top \in \mathcal{H}^{|\mathcal{V}|}$ and solve

$$p^* := \min_{f \in \mathcal{H}^{|\mathcal{V}|}} \sum_{i=1}^{|\mathcal{V}|} \mathbb{E}_{\mathbf{x}_i, y_i}\left[\ell(f_i(\mathbf{x}_i), \mathbf{y}_i)\right] + \frac{\lambda}{2}\|f\|_{\mathcal{H}}^2$$

$$s.t. \quad f_i = f_j \quad \text{for all} \quad i \in \mathcal{V} \quad \text{and} \quad j \in \mathcal{N}_i$$

- We solve it approximately using a penalty method

$$f_c^* = \underset{f \in \mathcal{H}^{|\mathcal{V}|}}{\text{argmin}}\, \psi_c(f) = \underset{f \in \mathcal{H}^{|\mathcal{V}|}}{\text{argmin}} \sum_{i \in \mathcal{V}} \mathbb{E}_{\mathbf{x}_i, \mathbf{y}_i}\left[\ell_i(f_i(\mathbf{x}_i), y_i)\right] + \frac{\lambda}{2}\|f\|_{\mathcal{H}}^2$$
$$+ \frac{c}{2} \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{N}_i} \mathbb{E}_{\mathbf{x}_i}\left[\left(f_i(\mathbf{x}_i) - f_j(\mathbf{x}_i)\right)^2\right]$$

Distributed Function Estimation

- How far from consensus is the approximate solution?

Proposition

Let $f_c^* = \text{argmin}_{f \in \mathcal{H}^{|\mathcal{V}|}} \psi_c(f)$ *and let* p^* *be the optimal cost of the distributed learning problem. Then for all penalties* $c > 0$ *we have that*

$$\frac{1}{2} \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{N}_i} \mathbb{E}_{\mathbf{x}_i} \left\{ \left[f_{c,i}^*(\mathbf{x}_i) - f_{c,j}^*(\mathbf{x}_i) \right]^2 \right\} \leq \frac{p^*}{c}$$

- Expected disagreement arbitrarily small by increasing c

Functional Derivative

- Let $L(f)$ be the loss functional

$$L(f) = \sum_{i \in \mathcal{V}} \mathbb{E}_{\mathbf{x}_i, y_i}[\ell(f_i(\mathbf{x}_i), y_i)]$$

- Compute stochastic functional gradient of Ł(f)

$$\nabla_{f_i} \ell(f_i(\mathbf{x}_{i,t}), y_{i,t})(\cdot) = \frac{\partial \ell(f_i(\mathbf{x}_{i,t}), y_{i,t})}{\partial f_i(\mathbf{x}_{i,t})} \frac{\partial f_i(\mathbf{x}_{i,t})}{\partial f_i}(\cdot)$$

- Use reproducing property of kernel (i), differentiate both sides:

$$\frac{\partial f_i(\mathbf{x}_{i,t})}{\partial f_i}(\cdot) = \frac{\partial \langle f_i, \kappa(\mathbf{x}_{i,t}, \cdot) \rangle_{\mathcal{H}}}{\partial f_i} = \kappa(\mathbf{x}_{i,t}, \cdot)$$

Functional Distributed SGD

- FDSGD applied to $\psi_c(f)$, given independent example $(\mathbf{x}_{i,t}, \mathbf{y}_{i,t})$:

$$f_{i,t+1} = f_{i,t} - \eta_t \hat{\nabla}_{f_i} \psi_c(f_{i,t}(\mathbf{x}_{i,t}), y_{i,t}) = (1 - \eta_t \lambda) f_{i,t} - \eta_t \omega_{i,t+1} \kappa(\mathbf{x}_{i,t}, \cdot)$$

$$\omega_{i,t+1} = \left(\ell'(f_i(\mathbf{x}_{i,t}), y_{i,t}) + c \sum_{j \in \mathcal{N}_i} (f_{i,t}(\mathbf{x}_{i,t}) - f_{j,t}(\mathbf{x}_{i,t})) \right)$$

- Use the kernel expansion of $f_{i,t}$ to write

$$f_{i,t+1}(\mathbf{x}) = (1 - \eta_t \lambda) \sum_{n=1}^{t-1} w_{i,n} \kappa(\mathbf{x}_{i,n}, \mathbf{x}) - \eta_t \omega_{i,t+1} \kappa(\mathbf{x}_{i,t}, .)$$

- FDSGD: parametric updates on weights and dictionary

$$\mathbf{X}_{i,t+1} = [\mathbf{X}_{i,t}, \;\; \mathbf{x}_{i,t}], \quad \mathbf{w}_{i,t+1} = [(1 - \eta_t \lambda) \mathbf{w}_{i,t}, \;\; -\eta_t \omega_{i,t+1}] \,,$$

- Note that model order $M_t = t - 1$ grows by one at each step

Convergence Result

Theorem

Let $f_c^* := \operatorname{argmin}_{f \in \mathcal{H}} \psi_c(f)$*, under diminishing step-size rules* $\sum_{t=1}^{\infty} \eta_t = \infty$, $\sum_{t=1}^{\infty} \eta_t^2 < \infty$ *, with* $\eta_0 < 1/\lambda$*,*

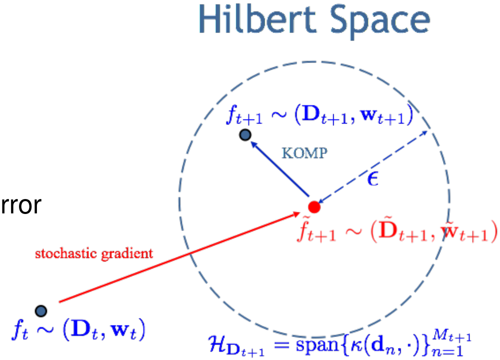
$$\lim_{t \to \infty} \|f_t - f_c^*\|_{\mathcal{H}}^2 = 0 \qquad a.s.$$

Controlling Model Order

- Each agent learns $f^*_{c,i}$ in such a way that $M_{i,t} << \infty$ for each $f_{i,t}$
- Accomplished by fixing a error nbhd. around FDSGD iterates
  $\Rightarrow$ Remove maximal no. kernel dict. elements while inside nbhd.
- We propose using KOMP $\Rightarrow$ kernel orthogonal matching pursuit
  $\Rightarrow$ a greedy compressive technique (Vincent & Bengio, 2002)

Kernel Matching Pursuit

- Fix approximation error ϵ
- $\tilde{f}_{t+1} = f_t - \eta \hat{\nabla}_f \psi_c(f_t)$
- Remove kernel element smallest error
- Project $\tilde{f}_{t+1}$ onto resulting RKHS
- Repeat until error is larger than ε



Convergence Results

Theorem

Let $f_c^* := \text{argmin}_{f \in \mathcal{H}} \psi_c(f)$. Given regularizer $\lambda > 0$, constant algorithm step-size η chosen such that $\eta < 1/\lambda$ and compression error $\epsilon = K\eta^{3/2} = \mathcal{O}(\eta^{3/2})$, where K is a positive scalar,

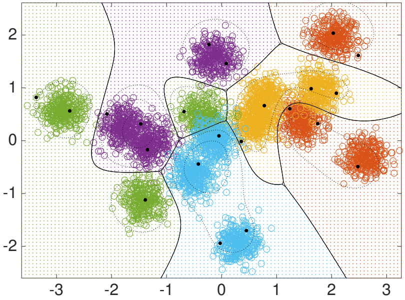
$$\liminf_{t \to \infty} \|f_t - f_c^*\|_{\mathcal{H}} \leq \frac{\sqrt{\eta}}{\lambda}\left(K|\mathcal{V}| + \sqrt{K^2|\mathcal{V}|^2 + \lambda\sigma^2}\right) = \mathcal{O}(\sqrt{\eta}) \qquad a.s.$$

The model order of the function, M_t is finite for all t

- Bias induced by sparsification asymptotically doesn't hurt too bad
- Constant step-size, approx. budget $\Rightarrow$ model order always finite

Online Multi-Class Kernel SVM

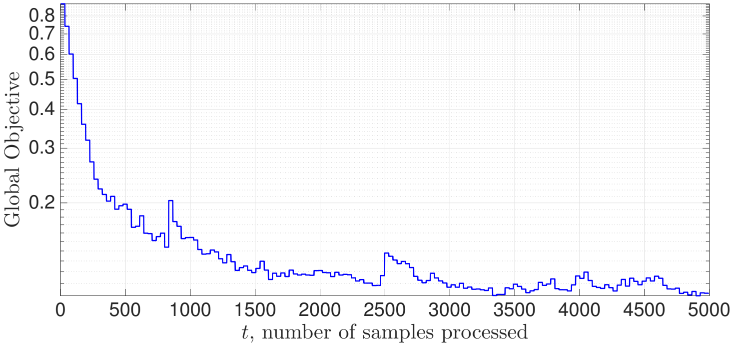
- 3 Gaussians per mixture, $C = 5$ classes total for this experiment
  $\Rightarrow$ 15 total Gaussians generate data
- $\ell(\mathbf{f}(\mathbf{x}), y) = \max(0, 1 + f_r(\mathbf{x}) - f_y(\mathbf{x})),\ r = \text{argmax}_{c' \neq y} f_{c'}(\mathbf{x})$



- Grid colors $\Rightarrow$ decision
- Black dots $\Rightarrow$ kernels
- $\sim$ 95.7% accuracy

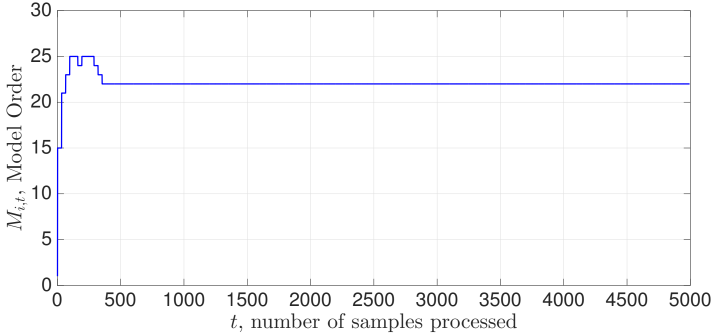
Online Multi-Class Kenrel SVM

- Convergence to optimal solution



Online Multi-Class Kenrel SVM

- Bounded model order

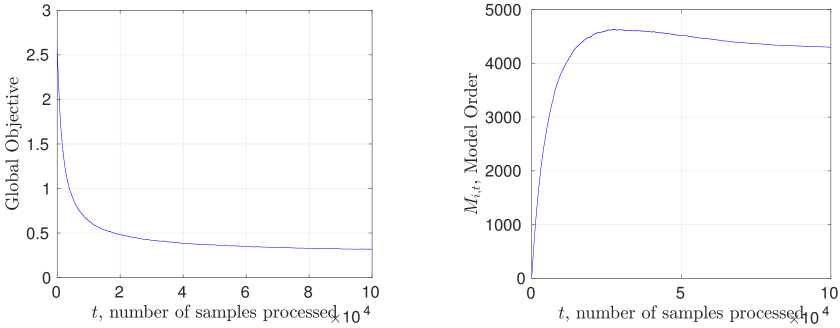


Texture Classification

- Texture classification on Broadatz dataset via SVM

Texture Classification

- We observe convergence and finite Model Order



- Accuracy of 93.5% comparable to centralized case (95.6%)

Conclusion

- We need to go beyond linear statistical models to do Learning
- Kernels and Neural Networks are the common tools to do so
  - ⇒ Kernel methods yield convex optimization problems
- We presented a distributed Learning algorithm (FDSGD)
  - ⇒ Converges to a neighborhood of the optimal function
  - ⇒ while ensuring a bound on the model order for all times