Large Scale Learning and Optimization

(using Support Vector Machines)

S. Sonnenburg[†], K. -R. Müller[†,+]

[+]Technical University Berlin,
[†]Fraunhofer FIRST.IDA, Berlin

Vorlesung SS 2008, 3. Juni

FIRST
Fraunhofer Institut Rechnerarchitektur und Softwaretechnik

Outline

Outline

Large Scale Problems

What makes a Problem Large Scale?

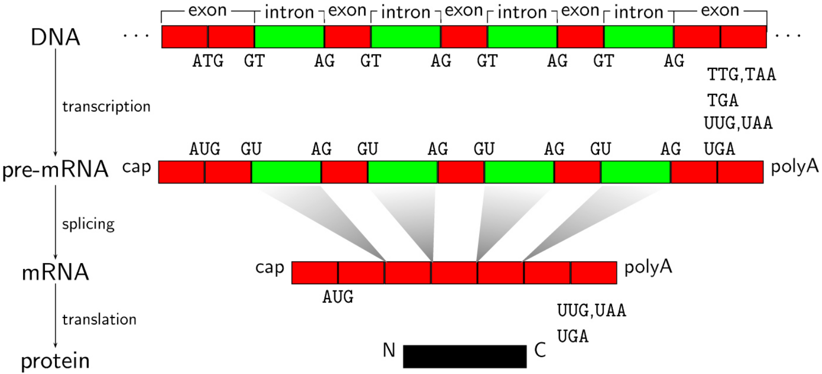
- Large number of data points
- Extremely high dimensionality
- High effort algorithms $\mathcal{O}(N^3)$
- Large memory requirements

$\Rightarrow$ **Anything that reaches current computers limits: computational, memory, transfer costs**

One may define a large scale problem to be a problem which to solve reaches current computers limits be it computational, memory or transfer costs wise. For machine learning this translates to high effort algorithms (e.g.$\mathcal{O}(N^3)$), large number of data points or high dimensionality.

Applications I

Bioinformatics (Splice Sites, Gene Boundaries, . . .)



Learning and predicting on the human genome

- Learn on $50 \cdot 10^6$ examples
- Predict on $\approx 2 \cdot 3 \cdot 10^9$ locations

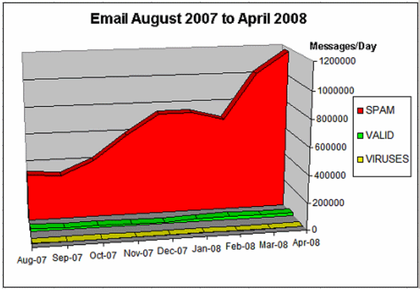
Applications II

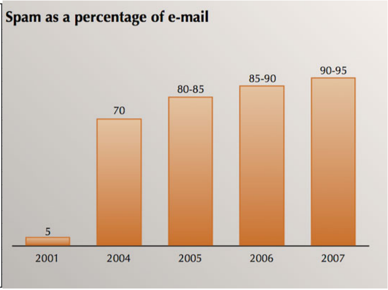
IT-Security (Network traffic)

- Google 15,000,000 http queries (5.7GB of traffic) per day
- Learning and Prediction has to be done in real-time.

Applications III

Text-Classification (Spam vs. Non-Spam)

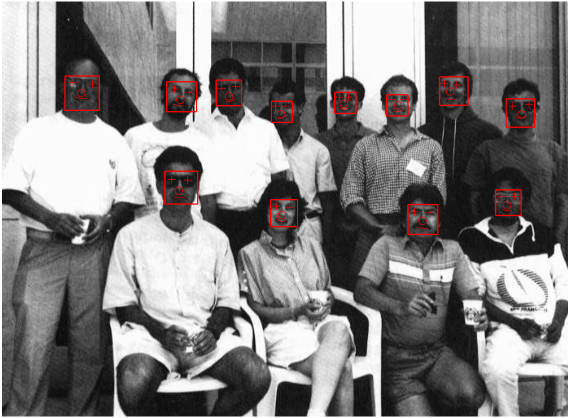




- Email Spam increases drastically

Applications IV

Image Recognition



- Face recognition: Examples are generated by sliding rectangles over the image (different scale, rotation)
- Training expensive, and real-time requirements on predictor.

Approaching LSL Problems

LSL might create challenges just to load/process data!
Therefore avoid it if not necessary!

- Obtain a reference solution via sub-sampling!
- More data = better performance?
- Approximative vs. exact algorithms.
- Simplest effective methods first.

If LSL is needed focus changes drastically.

Paradigm Shift

Suddenly, low-level details matter a lot!

Design decision become critical:

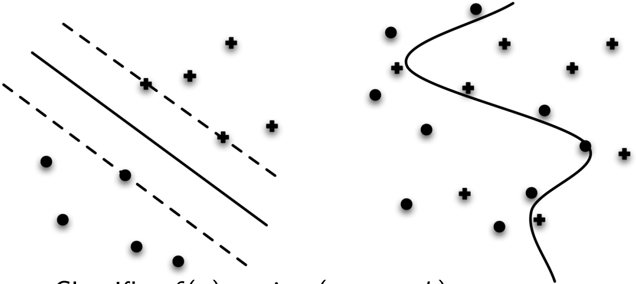
- Data representation (float/byte matrices, sparse matrices, hierarchical (trees), bit-representations)
- Programming language (ruby, python, java, c++, assembly)
- Problem formulation (how is the problem cast, are there equivalent but faster re-formulations?)
- Choice of algorithm (complexity,...)

In this lecture: How to speed up SVMs.

Outline

Recall Support Vector Machines

Given training examples $(\mathbf{x}_i, y_i)_{i=1}^N \in (\mathcal{X}, \{-1, +1\})^N$



- Linear Classifier $f(\mathbf{x}) = \operatorname{sign}(\mathbf{w} \cdot \mathbf{x} + b)$
- SVMs learn $\boldsymbol{\alpha} \in \mathbb{R}^N$ on training examples in kernel feature space $\Phi(\mathbf{x})$

$$f(\mathbf{x}) = \operatorname{sign}\left(\sum_{i=1}^{N} y_i \alpha_i \mathrm{k}(\mathbf{x}, \mathbf{x}_i) + b\right),$$

where Kernel $k(\mathbf{x}, \mathbf{x}') = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}')$

SVM Primal

$$P(\mathbf{w}, b, \xi) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\xi_i$$

$$\begin{array}{cc}
\min\limits_{\mathbf{w},b,\xi} & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\xi_i \\
\text{wrt :} & \mathbf{w} \in \mathbb{R}^D, b \in \mathbb{R}, \xi \in R^N \\
\text{s.t. :} & -\xi_i \leq 0, \forall i = 1 \ldots N \\
& 1 - y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i + b) - \xi_i \leq 0, \forall i = 1 \ldots N
\end{array}$$

Lagrangian

$$
\begin{aligned}
\min_{\mathbf{w},b,\xi} \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\xi_i \\
\text{wrt :} \quad & \mathbf{w} \in \mathbb{R}^D, b \in \mathbb{R}, \xi \in R^N \\
\text{s.t. :} \quad & -\xi_i \leq 0, \forall i = 1 \dots N \\
& 1 - y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i + b) - \xi_i \leq 0, \forall i = 1 \dots N
\end{aligned}
$$

$$
\begin{aligned}
L(\mathbf{w}, b, \xi, \boldsymbol{\alpha}, \boldsymbol{\lambda}) &= \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\alpha_i y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i + b) \\
&+ \sum_{i=1}^{N}\alpha_i - \sum_{i=1}^{N}(\lambda_i + \alpha_i)\xi_i
\end{aligned}
$$

where $\boldsymbol{\alpha} \geq \mathbf{0}, \ \boldsymbol{\lambda} \geq \mathbf{0}$

Derivatives of the Lagrangian

$$\begin{aligned} L(\mathbf{w}, b, \xi, \boldsymbol{\alpha}, \boldsymbol{\lambda}) &= \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\alpha_i y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i + b) \\ &\quad + \sum_{i=1}^{N}\alpha_i - \sum_{i=1}^{N}(\lambda_i + \alpha_i)\xi_i \end{aligned}$$

where $\boldsymbol{\alpha} \geq \mathbf{0},\ \boldsymbol{\lambda} \geq \mathbf{0}$

$$\begin{aligned} \partial_{\mathbf{w}} L &= w - \sum_{i=1}^{N}\alpha_i y_i \mathbf{x}_i \stackrel{!}{=} 0 \ \Rightarrow \mathbf{w} = \sum_{i=1}^{N}\alpha_i y_i \mathbf{x}_i \\ \partial_b L &= -\sum_{i=1}^{N}\alpha_i y_i \stackrel{!}{=} 0 \ \Rightarrow \sum_{i=1}^{N}\alpha_i y_i = 0. \\ \partial_\xi L &= C\mathbf{1} - \boldsymbol{\alpha} - \lambda \stackrel{!}{=} 0 \end{aligned}$$

Derivatives of the Lagrangian

$$
\begin{aligned}
\partial_{\mathbf{w}} L &= w - \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i \stackrel{!}{=} 0 \Rightarrow \mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \mathbf{x}_i \\
\partial_b L &= -\sum_{i=1}^{N} \alpha_i y_i \stackrel{!}{=} 0 \Rightarrow \sum_{i=1}^{N} \alpha_i y_i = 0. \\
\partial_\xi L &= C\mathbf{1} - \boldsymbol{\alpha} - \lambda \stackrel{!}{=} 0
\end{aligned}
$$

Due to $\boldsymbol{\alpha} \geq \mathbf{0},\ \boldsymbol{\lambda} \geq \mathbf{0}$:

$$0 \leq \boldsymbol{\alpha} \leq C\mathbf{1}$$

$$0 \leq \boldsymbol{\lambda} \leq C\mathbf{1}$$

$$\lambda = C\mathbf{1} - \boldsymbol{\alpha}$$

Re-substitute into Lagrangian

$$
\begin{aligned}
D(\boldsymbol{\alpha}) &= \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i y_i \mathbf{x}_i^{\mathsf{T}}\alpha_j y_j \mathbf{x}_j + C\sum_{i=1}^{N}\xi_i \\
&\quad - \sum_{i=1}^{N}\alpha_i y_i\left(\sum_{j=1}^{N}\alpha_j y_j \mathbf{x}_i^{\mathsf{T}}\mathbf{x}_j + b\right) \\
&\quad + \sum_{i=1}^{N}\alpha_i - \sum_{i=1}^{N}(C - \alpha_i + \alpha_i)\xi_i \\
&= \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j \mathbf{x}_i^{\mathsf{T}}\mathbf{x}_j - \sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j \mathbf{x}_i^{\mathsf{T}}\mathbf{x}_j + \sum_{i=1}^{N}\alpha_i \\
&= \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j \mathbf{x}_i^{\mathsf{T}}\mathbf{x}_j
\end{aligned}
$$

SVM Dual

$$
\begin{aligned}
\max_{\boldsymbol{\alpha}} \quad & \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^{\mathsf{T}} \mathbf{x}_j \\
\text{wrt :} \quad & \boldsymbol{\alpha} \in \mathbb{R}^N \\
\text{s.t. :} \quad & 0 \le \alpha_i \le C, \forall i = 1 \ldots N \\
& \sum_{i=1}^{N} \alpha_i y_i = 0
\end{aligned}
$$

Solve using off-the-shelf Optimizers

- Use some general purpose solver to solve the problem (mosek, cplex, quadprog,...)
- Memory requirements
  - Requires to store whole kernel matrix $\mathcal{O}(N^2) \Rightarrow 30000$ examples $\approx$ 7GB
  - Computation time at least $\mathcal{O}(N^2 \cdot D)$ just for kernel elements (linear kernel).
- Computational Complexity
  - Lower bound $\mathcal{O}(N^2)$, to check optimality needs output for all examples $f(\mathbf{x}_i) = \sum_{j=1}^{N_s} \alpha_j \mathbf{y}_j K(\mathbf{x}_j, x_i)$, $\forall i = 1 \dots N$
  - Worst case $\mathcal{O}(N^3)$

Chunking and Sequential Minimal Optimization

General idea: Split large problem into smaller sub-problems.
(Called active set methods in optimization theory.)

- Chunking select q variables $\alpha_{i_1}, \ldots, \alpha_{i_q}$
- SMO is special case with $q = 2$, sub-problem can be solved analytically, but clever and efficient subset selection strategy needed.

> **Training algorithm (chunking):**
>
> **while** optimality conditions are violated **do**
>   select q variables for the working set.
>   solve reduced problem on the working set.
> **end while**

Chunking is implemented in SVMlight, SMO in Libsvm

Identifying inefficiencies A

At each iteration, the vector $\mathbf{f}$, $f_j = \sum_{i=1}^{N} \alpha_i y_i \, \mathrm{k}(x_i, x_j)$, $j = 1 \ldots N$ is needed for checking termination criteria and selecting new working set $\Rightarrow$ Effort $\mathcal{O}(D \cdot N^2)$

Speedup A:

1. Avoid to recompute $\mathbf{f}$ from scratch
2. Start with $\mathbf{f} = 0$
3. Compute "linear updates" on $\mathbf{f}$ on the working set W

$$f_j \leftarrow f_j^{old} + \sum_{i \in W} (\alpha_i - \alpha_i^{old}) y_i \, \mathrm{k}(x_i, x_j)$$

Effort $\mathcal{O}(D \cdot N \cdot q)$

Identifying inefficiencies B

Speedup B: Update rule: $f_j \leftarrow f_j^{old} + \sum_{i \in W} (\alpha_i - \alpha_i^{old}) y_i \, \mathrm{k}(x_i, x_j)$

1. Exploit $\mathrm{k}(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$ and obtain

$$f_j \leftarrow f_j^{old} + \sum_{i \in W} (\alpha_i - \alpha_i^{old}) y_i \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$$

2. Use $\mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \Phi(\mathbf{x}_i)$ to get

$$f_j \leftarrow f_j^{old} + \mathbf{w}^W \cdot \Phi(\mathbf{x}_j)$$

($\mathbf{w}^W$ normal on working set)

Observations

- $q := |W|$ is very small in practice $\Rightarrow$ precomputing $\mathbf{w}$ is cheap
- computing dot products still dominates computing time
- Overall effort $\mathcal{O}(N \cdot D + q \cdot D)$

Dual Formulation for Linear SVMs is *inefficient*

Number of variables $dim(\alpha) = N$ **depends on** N

Recall the Primal Formulation

$$\begin{array}{ll} \min\limits_{\mathbf{w},b,\xi} & \frac{1}{2}\left\|w\right\|^2 + C\sum_{i=1}^{N}\xi_i \\ \text{wrt :} & \mathbf{w} \in \mathbb{R}^D, b \in \mathbb{R}, \xi \in R^N \\ \text{s.t. :} & -\xi_i \leq 0, \forall i = 1 \dots N \\ & 1 - y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i + b) - \xi_i \leq 0, \forall i = 1 \dots N \end{array}$$

Number of variables in Primal is $dim(\mathbf{w}) = D + N + 1$
$\Rightarrow$ **Primal even worse?**

Working in the Primal

- Standard SVM Primal
- Convert into (equivalent) unconstrained Primal

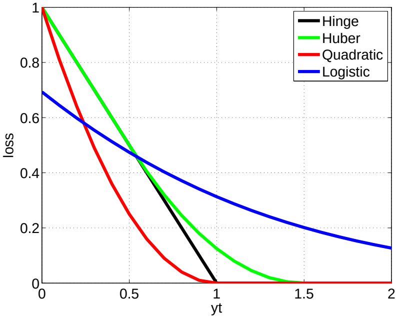
$$\frac{1}{2}\|\mathbf{w}\|^2 \quad + \quad C\sum_{i=1}^{N}(\max\{0, 1 - y_i(\mathbf{w}\cdot\mathbf{x}_i + b)\})$$

$$\frac{1}{2}\|\mathbf{w}\|^2 \quad + \quad C\sum_{i=1}^{N}L(y_i, f(\mathbf{x}_i))$$

- Hinge Loss $L(y_i, f(\mathbf{x}_i)) = \max\{0, 1 - y_i(f(\mathbf{x}_i)\}$

Number of variables is now $D+1$ Can be solved using e.g. gradient descent and newton for differentiable losses

Differentiable approximations to the Hinge Loss



- Hinge Loss
  $L(y, t) = \max\{0, 1 - yt\}$
- Squared Loss
  $L(y, t) = \max\{0, 1 - yt\}^2$
- Logistic Loss
  $L(y, t) = \log(1 + e^{-yt})$

$$\text{Hubers Loss } L(y, t) = \begin{cases} 0 & \text{if } yt > 1 + h \\ \frac{(1+h-yt)^2}{4h} & \text{if } |1 - yt| \leq h \\ 1 - yt & \text{if } yt < 1 - h \end{cases}$$

Descent Method for Unconstrained Problems

Let us consider an unconstrained convex problem: $\min f(\mathbf{x})$

Initialization: set $\mathbf{x} \in \mathbf{dom} f$.

repeat

1. Determine a descent direction $\boldsymbol{\delta}$.
2. Line-search: find a step size $t = \operatorname{argmin}_{t'>0} f(\mathbf{x} + t'\boldsymbol{\delta})$.
3. Update $\mathbf{x} := \mathbf{x} + t\boldsymbol{\delta}$.

until stopping condition is satisfied.

- It generates $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots$ such that $f(\mathbf{x}^{(k)}) > f(\mathbf{x}^{(k+1)})$.
- For f differentiable, a vector $\boldsymbol{\delta}$ is a descent direction if

$$\nabla f(\mathbf{x})^T \boldsymbol{\delta} < 0$$

e.g., gradient descent methods use $\boldsymbol{\delta} = -\nabla f(\mathbf{x})$.

Stochastic Gradient Descent

- SGD is a Online Method - works well for huge N
- Objective function as sum over training examples

$$\frac{1}{N}\sum_{i=1}^{N}\underbrace{\|w\|^2 + N \cdot C \cdot L(y, f(x))}_{\ell_i(\mathbf{w})}$$

- Update Rule: At each iteration t choose a random i

$$\mathbf{w} \leftarrow \mathbf{w} - \frac{\eta}{t}\nabla\ell_i(\mathbf{w})$$

- η learning rate, critical parameter, $\eta = C \cdot N$ works OK in practice, for tuning cf. Bottou 2007
- Iteration cost $\mathcal{O}(D)$

$\Rightarrow$ **Good approximations after a few passes through the data.**

Newton Methods for Unconstrained Problems

Let us consider equality constrained convex problem $\min f(\mathbf{x})$

- Using the KKT optimality conditions, $\mathbf{x} \in \mathbf{dom} f$ is optimal iff there exist $\boldsymbol{\nu}$ such that

$$\nabla f(\mathbf{x}) = 0 \,.$$

- For a **convex quadratic function** $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T\mathbf{H}\mathbf{x} + \mathbf{c}^T\mathbf{x}$ the **KKT conditions** lead to an efficiently solvable **set of linear equations**:

$$\mathbf{H}\mathbf{x} + \mathbf{c} = 0 \,.$$

- **Newton method** is applicable for a general twice differentiable function $f(\mathbf{x})$: it iteratively approximates $f(\mathbf{x})$ by a quadratic function

$$\hat{f}(\mathbf{x}) = \frac{1}{2}(\mathbf{x} - \mathbf{x}')\nabla^2 f(\mathbf{x}')(\mathbf{x} - \mathbf{x}') + \nabla f(\mathbf{x}')^T(\mathbf{x} - \mathbf{x}') + f(\mathbf{x}')$$

and solves the KKT conditions for the approximation $\hat{f}(\mathbf{x})$.

Cutting Plane

Unconstrained convex minimization problem

$$\mathbf{w}^* = \underset{\mathbf{w}\in\Re^n}{\operatorname{argmin}}\, F(\mathbf{w}) := \left(\frac{1}{2}\|\mathbf{w}\|^2 + C \cdot R(\mathbf{w})\right)$$

Difficulty stems from the risk term $R(\mathbf{w})$.

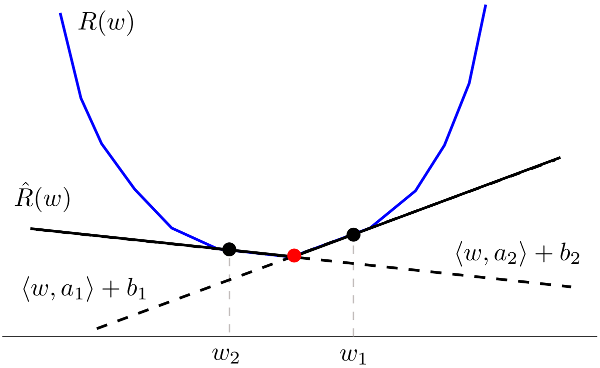
Idea

Approximate $R(\mathbf{w})$ by a simpler term $\hat{R}(\mathbf{w})$ constructed as point-wise maximum of linear function.

Cutting plane approximation

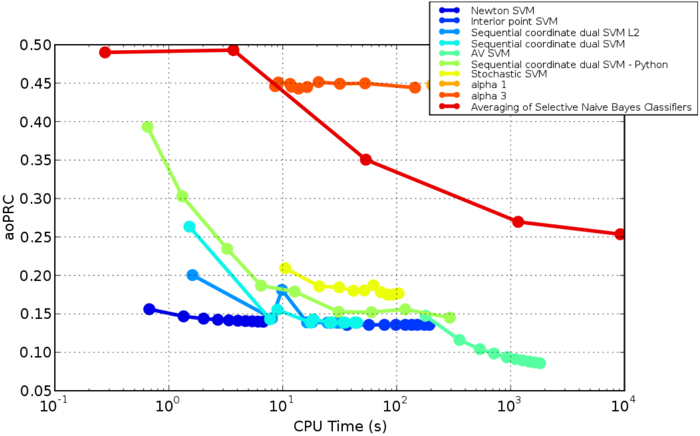
$$R(\mathbf{w}) \geq \hat{R}(\mathbf{w}) \quad \text{where} \quad \hat{R}(\mathbf{w}) = \max_{i=1,\ldots,t} \left(\langle \mathbf{w}, \mathbf{a}_i \rangle + b_i \right)$$

$\{(\mathbf{a}_1, b_1), \ldots, (\mathbf{a}_m, b_m)\}$ are cutting planes at points $\{\mathbf{w}_1, \ldots, \mathbf{w}_t\}$.



Running times (from Large Scale Learning Challenge)

`http://largescale.first.fraunhofer.de`



- Newton quite fast for low dimensional data.
- Stochastic Gradient Descent for high-dimensional data.

Outline

LSL and Kernel SVMs

- Most common: dual based chunking methods (svm-light, libsvm)
- Other approach in dual e.g. Low rank decomposition, aim find $\hat{K}$ that is close to K, e.g.

$$\left\| \hat{K} - K \right\|^2 \leq \epsilon$$

- What about non-vectorial based string kernel SVMs?

Large Scale Learning with Strings

- Text Classification (Spam, Web-Spam, Categorization)
  - Task: Given N documents, with class label ± 1, predict text type.
- Security (Network Traffic, Viruses, Trojans)
  - Task: Given N executables, with class label ± 1, predict whether executable is a virus.
- Biology (Promoter, Splice Site Prediction)
  - Task: Given N sequences around Promoter/Splice Site (label $+1$) and fake examples (label -1), predict whether there is a Promoter/Splice Site in the middle

$\Rightarrow$ **Approach: String kernel + Support Vector Machine**
$\Rightarrow$ **Large N is needed to achieve high accuracy (i.e. $N = 10^7$)**

Formally

- Given:
  - N training examples $(\mathbf{x}_i, y_i) \in (\mathcal{X}, \pm 1)$, $i = 1 \ldots N$
  - string kernel $K(\mathbf{x}, \mathbf{x}') = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}')$
- Examples:
  - words-in-a-bag-kernel
  - k-mer based kernels (Spectrum, Weighted Degree)
- Task:
  - Train Kernelmachine on Large Scale Datasets, e.g. $N = 10^7$
  - Apply Kernelmachine on Large Scale Datasets, e.g. $N = 10^9$

String Kernels

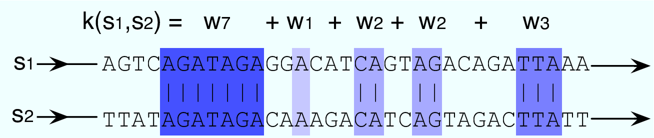
- Spectrum Kernel (with mismatches, gaps)

$$K(\mathbf{x}, \mathbf{x}') = \Phi_{sp}(\mathbf{x}) \cdot \Phi_{sp}(\mathbf{x}')$$

x AAACAAATAAGTAACTAATCTTTTAGGAAGAACGTTTCAACCATTTTGAG

x' TACCTAATTATGAAATTAAATTTCAGTGTGCTGATGGAAACGGAGAAGTC

- Weighted Degree Kernel (with shift)



For string kernels $\mathcal{X}$ discrete space and $\Phi(x)$ sparse

Kernel Machine

Kernel Machine Classifier:

$$f(\mathbf{x}) = \mathrm{sign}\left(\sum_{i=1}^{N} \alpha_i y_i \,\mathrm{k}(\mathbf{x}_i, \mathbf{x}) + b\right)$$

To compute output on all M examples:

$$\forall j = 1, \ldots, M : \quad \sum_{i=1}^{N} \alpha_i y_i \,\mathrm{k}(\mathbf{x}_i, \mathbf{x}_j) + b$$

Computational effort:

- Single $\mathcal{O}(NT)$ (T time to compute the kernel)
- All $\mathcal{O}(NMT)$

$\Rightarrow$ **Costly!**

$\Rightarrow$ **Used in training and testing - worth tuning.**

$\Rightarrow$ **How to further speed up if $T = dim(\mathcal{X})$ already linear?**

Linadd Speedup Idea

Key Idea: Store $\mathbf{w}$ and compute $\mathbf{w} \cdot \Phi(\mathbf{x})$ *efficiently*

$$\sum_{i=1}^{N} \alpha_i y_i \, \mathrm{k}(\mathbf{x}_i, \mathbf{x}_j) = \underbrace{\sum_{i=1}^{N} \alpha_i y_i \Phi(\mathbf{x}_i)}_{\mathbf{w}} \cdot \Phi(\mathbf{x}_j) = \mathbf{w} \cdot \Phi(\mathbf{x}_j)$$

When is that possible ?

1. $\mathbf{w}$ has low dimensionality and sparse (e.g. 4^8 for Feature map of Spectrum Kernel of order 8 DNA)
2. $\mathbf{w}$ is extremely sparse although high dimensional (e.g. 10^{14} for Weighted Degree Kernel of order 20 on DNA sequences of length 100)

Effort: $\mathcal{O}(MT')$ $\Rightarrow$ **Potential speedup of factor** N

Technical Remark

Treating w

- **w** must be accessible by some index u (i.e. $u = 1 \ldots 4^8$ for 8-mers of Spectrum Kernel on DNA or word index for word-in-a-bag kernel)
- Needed Operations
  - `Clear`: $\mathbf{w} = \mathbf{0}$
  - `Add`: $w_u \leftarrow w_u + v$ (only needed $|W|$ times per iteration)
  - `Lookup`: obtain w_u (must be highly efficient)
- Storage
  - **Explicit Map** (store dense **w**); `Lookup` in $\mathcal{O}(1)$
  - **Sorted Array** (word-in-bag-kernel: all words sorted with value attached); `Lookup` in $\mathcal{O}(\log(\sum_u I(w_u \neq 0)))$
  - **Suffix Tries, Trees**; `Lookup` in $\mathcal{O}(K)$

Datastructures - Summary of Computational Costs

Comparison of worst-case run-times for operations

- `clear` of $\mathbf{w}$
- `add` of all k-mers $\mathbf{u}$ from string $\mathbf{x}$ to $\mathbf{w}$
- `lookup` of all k-mers $\mathbf{u}$ from $\mathbf{x}'$ in $\mathbf{w}$

	Explicit map	Sorted arrays	Tries	Suffix trees
`clear`	$\mathcal{O}(\lvert\Sigma\rvert^d)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
`add`	$\mathcal{O}(l_\mathbf{x})$	$\mathcal{O}(l_\mathbf{x} \log l_\mathbf{x})$	$\mathcal{O}(l_\mathbf{x} d)$	$\mathcal{O}(l_\mathbf{x})$
`lookup`	$\mathcal{O}(l_{\mathbf{x}'})$	$\mathcal{O}(l_\mathbf{x} + l_{\mathbf{x}'})$	$\mathcal{O}(l_{\mathbf{x}'} d)$	$\mathcal{O}(l_{\mathbf{x}'})$

Conclusions

- Explicit map ideal for small $\lvert\Sigma\rvert$
- Sorted Arrays for larger alphabets
- Suffix Arrays for large alphabets and order (overhead!)

Support Vector Machine

`Linadd` **directly applicable when applying the classifier.**

$$f(\mathbf{x}) = \operatorname{sign}\left(\sum_{i=1}^{N} \alpha_i y_i \, \mathrm{k}(\mathbf{x}_i, \mathbf{x}) + b\right)$$

Problems

- $\mathbf{w}$ may still be huge $\Rightarrow$ fix by not constructing whole $\mathbf{w}$ but only blocks and computing batches

What about training?

- general purpose QP-solvers, Chunking, SMO
- optimize kernel (i.e. find $O(L)$ formulation, where $L = dim(\mathcal{X})$)
- Kernel Caching infeasable
  (for $N = 10^6$ only 125 kernel rows fit in 1GiB memory)

$\Rightarrow$ **Use `linadd` again: Faster + needs no kernel caching**

Derivation I

Analyzing Chunking SVMs (GPDT, SVM^{light}:)

Training algorithm (chunking):

while optimality conditions are violated **do**
  select q variables for the working set.
  solve reduced problem on the working set.
end while

- At each iteration, the vector $\mathbf{f}$, $f_j = \sum_{i=1}^{N} \alpha_i y_i \, \mathrm{k}(x_i, x_j)$, $j = 1 \dots N$ is needed for checking termination criteria and selecting new working set (based on $\boldsymbol{\alpha}$ and gradient w.r.t. $\boldsymbol{\alpha}$).
- Avoiding to recompute $\mathbf{f}$, most time is spend computing "linear updates" on $\mathbf{f}$ on the working set W

$$f_j \leftarrow f_j^{old} + \sum_{i \in W} (\alpha_i - \alpha_i^{old}) y_i \, \mathrm{k}(x_i, x_j)$$

Derivation II

Use `linadd` to compute updates.

Update rule: $f_j \leftarrow f_j^{old} + \sum_{i \in W}(\alpha_i - \alpha_i^{old}) y_i \, \mathrm{k}(x_i, x_j)$
Exploiting $\mathrm{k}(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$ and $\mathbf{w} = \sum_{i=1}^{N} \alpha_i y_i \Phi(\mathbf{x}_i)$:

$$f_j \leftarrow f_j^{old} + \sum_{i \in W}(\alpha_i - \alpha_i^{old}) y_i \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j) = f_j^{old} + \mathbf{w}^W \cdot \Phi(\mathbf{x}_j)$$

($\mathbf{w}^W$ normal on working set)

Observations

- $q := |W|$ is very small in practice $\Rightarrow$ can effort more complex $\mathbf{w}$ and `clear`,`add` operation
- `lookups` dominate computing time

Algorithm

Recall we need to compute updates on $\mathbf{f}$ (effort $c_1|W|LN$):

$$f_j \leftarrow f_j^{old} + \sum_{i \in W} (\alpha_i - \alpha_i^{old}) y_i \, \mathrm{k}(x_i, x_j) \text{ for all } j = 1 \dots N$$

Modified SVM^{light} using "LinAdd" algorithm (effort $c_2 \ell LN$, ℓ Lookup cost)

$f_j = 0$, $\alpha_j = 0$ for $j = 1, \dots, N$
for $t = 1, 2, \dots$ **do**
Check optimality conditions and stop if optimal, select working set W based on $\mathbf{f}$ and $\boldsymbol{\alpha}$, store $\boldsymbol{\alpha}^{old} = \boldsymbol{\alpha}$
solve reduced problem W and update $\boldsymbol{\alpha}$
clear $\mathbf{w}$
$\mathbf{w} \leftarrow \mathbf{w} + (\alpha_i - \alpha_i^{old}) y_i \Phi(\mathbf{x}_i)$ for all $i \in W$
update $f_j = f_j + \mathbf{w} \cdot \Phi(\mathbf{x}_j)$ for all $j = 1, \dots, N$
end for

Speedup of factor $\frac{c_1}{c_2 \ell}|W|$

Datasets

- Web Spam
  - Negative data: Use Webb Spam corpus `http://spamarchive.org/gt/` (350,000 pages)
  - Positive data: Download 250,000 pages randomly from the web (e.g. `cnn.com`, `microsoft.com`, `slashdot.org` and `heise.de`)
  - Use spectrum kernel $k = 4$ using **sorted arrays** on 100,000 examples train and test (average string length 30Kb, 4 GB in total, 64bit variables $\Rightarrow$ 30GB)

Web Spam results

Classification Accuracy and Training Time

N	100	500	5,000	10,000	20,000	50,000	70,000	100,000
Spec	2	97	1977	6039	19063	94012	193327	-
LinSpec	3	255	4030	9128	11948	44706	83802	107661
Accuracy	89.59	92.12	96.36	97.03	97.46	97.83	97.98	98.18
auROC	94.37	97.82	99.11	99.32	99.43	99.59	99.61	99.64

Speed and classification accuracy comparison of the spectrum kernel without (*Spec*) and with `linadd` (*LinSpec*)

Datasets

- Splice Site Recognition
  - Negative Data: 14,868,555 DNA sequences of fixed length 141 base pairs
  - Positive Data: 159,771 Acceptor Splice Site Sequences
  - Use WD kernel $k = 20$ (using **Tries**) and spectrum kernel $k = 8$ (using **explicit maps**) on $10,000,000$ train and 5,028,326 examples

Linadd for WD kernel

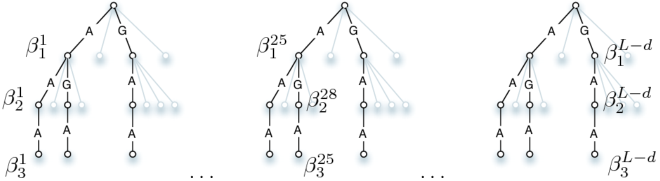
For linear combination of kernels:

$\sum_{j \in W}(\alpha_j - \alpha_j^{old}) y_j \, \mathrm{k}(x_i, x_j) \; (\mathcal{O}(Ld|W|N))$

AAACTAATTATGAAATTAAATTTCAGAGTGCTGATGGAAACGGAGAAGAA

- use one tree of depth d per position in sequence
- for `Lookup` use traverse one tree of depth d per position in sequence

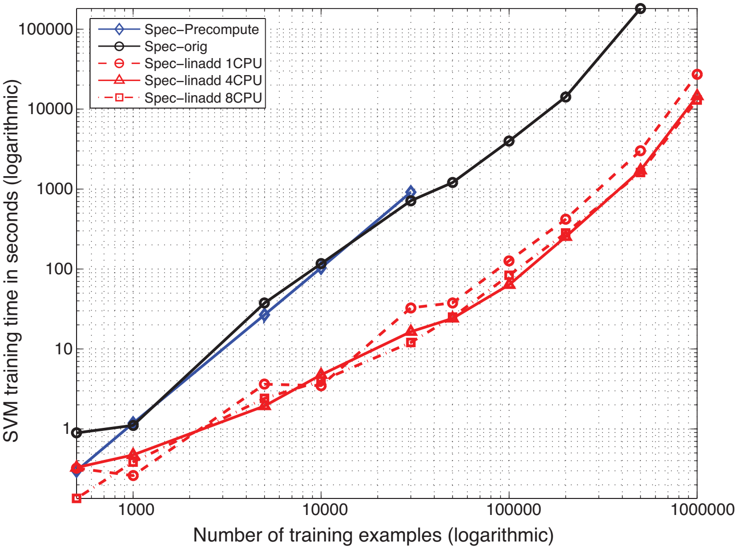
Example $d = 3$:



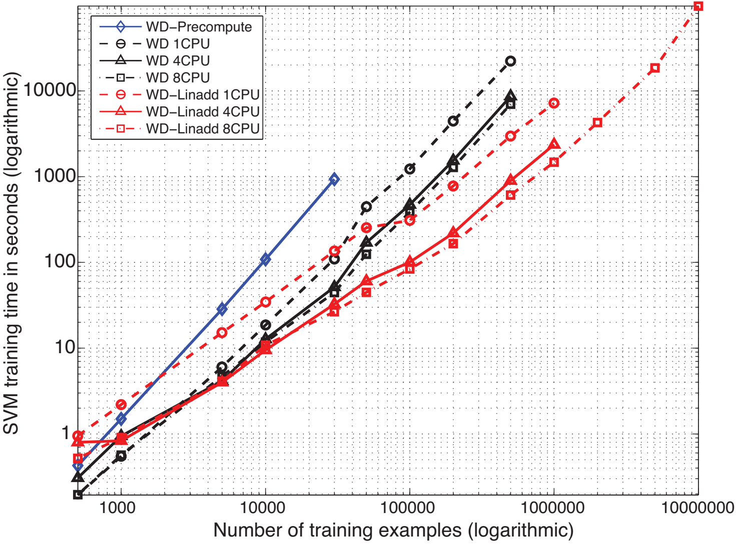
output for N sequences of length L in $\mathcal{O}(Ld \cdot N)$

(d depth of tree $\triangleq$ degree of WD kernel)

Spectrum Kernel on Splice Data



Weighted Degree Kernel on Splice Data



More data helps

N	*auROC*	*auPRC*
500	75.55	3.94
1,000	79.86	6.22
5,000	90.49	15.07
10,000	92.83	25.25
30,000	94.77	34.76
50,000	95.52	41.06
100,000	96.14	47.61

N	*auROC*	*auPRC*
200,000	96.57	53.04
500,000	96.93	59.09
1,000,000	97.19	63.51
2,000,000	97.36	67.04
5,000,000	97.54	70.47
10,000,000	97.67	72.46
10,000,000	96.03*	44.64*

Outline

Summary Large Scale Learning I

`linadd` for (string) kernel SVMs

- General speedup trick (`clear`, `add`, `lookup` operations) for string kernels
- Shared memory parallelization, able to train on **10 million** human splice sites
- Gives reasonable speedups and can be further parallelized
- State-of-the-art accuracy

Implementations

- `linadd`
- stochastic gradient descent, cutting plane based SVMs
- current fastest SVM solver (OCAS)

Implemented in SHOGUN `http://www.shogun-toolbox.org`

Summary Large Scale Learning II

Choose your weapons.

- LSL is machine learning at its (practical) limits.
- Design decisions are critical and should be made with care.

Further reading and sources to prepare this lecture.

- Chapelle, Training a support vector machine in the primal
- Sonnenburg et.al., Large scale learning with string kernels
- Bottou, Stochastic Gradient Learning in Neural Networks
- Joachims, Making Large-Scale SVM Learning Practical
- Joachims, Training Linear SVMs in Linear Time
- Teo et.al., Bundle Methods for regularized Risk Minimization (BMRM)
- Lin et.al., Trust Region Newton Method for Large-Scale Logistic Regression
- Chang et.al., LIBSVM a library for support vector machines