A Computational Framework for Linear SVMs

(COFFIN — large scale (non)-linear learning)

Sören Sonnenburg
TU Berlin

joint work with Vojtech Franc

Outline

Motivation

Many Applications have huge sample sizes

- Bioinformatics (Splice Sites, Gene Boundaries,. . .)
- IT-Security (Network traffic)
- Text-Classification (Spam vs. Non-Spam)
- Image Recognition

AIM: Development of a large scale learning framework for SVMs

- Training on full sample necessary to achieve state-of-the-art results
- Apply the learner to massive data sets

Motivation

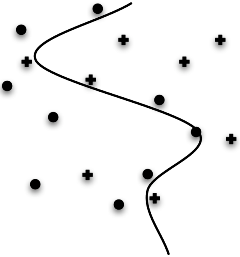
Many Applications have huge sample sizes

- Bioinformatics (Splice Sites, Gene Boundaries,. . .)
- IT-Security (Network traffic)
- Text-Classification (Spam vs. Non-Spam)
- Image Recognition

AIM: Development of a large scale learning framework for SVMs

- Training on full sample necessary to achieve state-of-the-art results
- Apply the learner to massive data sets

Support Vector Machines (SVMs)



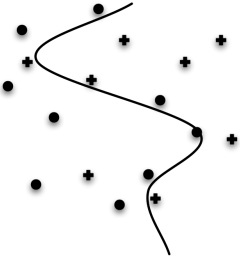
- SVMs learn weights $\boldsymbol{\alpha} \in \mathbb{R}^m$ over training examples in kernel feature space $\Phi : \mathbf{x} \mapsto \mathbb{R}^n$

- Decision function $f(\mathbf{x}) = \operatorname{sign}\left(\sum_{i=1}^{m} y_i \alpha_i k(\mathbf{x}, \mathbf{x}_i) + b\right)$,
  with kernel $k(\mathbf{x}, \mathbf{x}') = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}')$

SVMs rock!

- Kernels - flexible!
- In many applications SVMs define the state-of-the-art!
- But not large scale!

Support Vector Machines (SVMs)



- SVMs learn weights $\boldsymbol{\alpha} \in \mathbb{R}^m$ over training examples in kernel feature space $\Phi : \mathbf{x} \mapsto \mathbb{R}^n$

- Decision function $f(\mathbf{x}) = \mathrm{sign}\left(\sum_{i=1}^{m} y_i \alpha_i \mathrm{k}(\mathbf{x}, \mathbf{x}_i) + b\right)$,
  with kernel $k(\mathbf{x}, \mathbf{x}') = \Phi(\mathbf{x}) \cdot \Phi(\mathbf{x}')$

SVMs rock!

- Kernels - flexible!
- In many applications SVMs define the state-of-the-art!
- But not large scale!

Support Vector Machines are a Dead End

The Curse of Support Vectors

To compute output on all m examples $\mathbf{x}_1, \ldots, \mathbf{x}_m$:

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

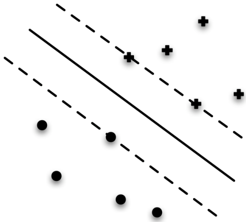
Computational effort:

- All $\mathcal{O}(m_s m T)$, (T time to compute the kernel)
- Effort Scales linearly with $m_s = \mathcal{O}(m) := \#\mathrm{SVs}$

$\Rightarrow$ **SVM's in bigO are not faster than standard k-NN.**

$\Rightarrow$ **Kernel Machines are just not large-scale!**

What about Linear SVMs ?



- **Linear** Support Vector Machines learn weights $\mathbf{w} \in \mathbb{R}^n$
- Decision function $\mathbf{f}(\mathbf{x}) = \mathbf{w} \cdot x + b$

Recent Progress in Linear SVM solvers

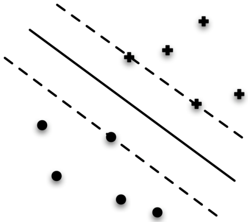
- SGD (Bottou 2007), SGD-QN (Bordes et al., 2009)
- SVM^{perf} (Joachims 2006)
- BMRM (Teo et.al. 2007, OCAS (Franc, Sonnenburg 2009)

$\Rightarrow$ **Linear training Effort** $\mathcal{O}(m)$

$\Rightarrow$ **Computing Outputs Linear Effort** $\mathcal{O}(nm)$

... but already linear time and just linear

What about Linear SVMs ?



- **Linear** Support Vector Machines learn weights $\mathbf{w} \in \mathbb{R}^n$
- Decision function $\mathbf{f}(\mathbf{x}) = \mathbf{w} \cdot x + b$

Recent Progress in Linear SVM solvers

- SGD (Bottou 2007), SGD-QN (Bordes et al., 2009)
- SVM^{perf} (Joachims 2006)
- BMRM (Teo et.al. 2007, OCAS (Franc, Sonnenburg 2009)

$\Rightarrow$ **Linear training Effort** $\mathcal{O}(m)$

$\Rightarrow$ **Computing Outputs Linear Effort** $\mathcal{O}(nm)$

. . . but already linear time and just linear

Motivation

AIM: Development of a large scale learning framework for SVMs

> *"Algorithm [linear SVM solver] improvements do not improve the order of test error convergence. They can simply improve constant factors and therefore* **compete evenly with the implementation improvements**. *Time spent refining the implementation is time well spent."*

from: Bordes, Bottou, Gallinari: SQD-QN: Careful Quasi-Newton Stochastic Gradient Descent. JMLR 2009.

Towards a computational framework for linear SVMs

Linear SVM solvers like liblinear, SGD, BMRM, Ocas all only require two operations:

(i) dot product between feature vector and the vector $\mathbf{w}$:
$r \leftarrow \langle \mathbf{x}, \mathbf{w} \rangle$ **DOT**

(ii) multiplication with a scalar $\alpha \in \Re$ and addition to the vector $\mathbf{v} \in \Re^n$: $\mathbf{v} \leftarrow \alpha \mathbf{x} + \mathbf{v}$ **ADD**

COFFIN

COFFIN really is just two simple ideas:

On demand compute. . .

1. Features $\Phi(\mathbf{x})$ (only non-zero dims)
   - Non-Linearity Possible
   - Examples: Low Degree Polynomial Kernel, Spectrum Kernel, Weighted Degree Kernel
   - On-the-fly (de)compression
2. Virtual Examples
   - Incorporating Invariances possible
   - Examples: Image translation, rotation, etc

Needs efficient data structure for w!

- . . ., dense, sorted array, trees, hashes
- fast only when $|\Phi_{\neq 0}(\mathbf{z})| \sim dim(\mathbf{z})$

COFFIN

COFFIN really is just two simple ideas:

On demand compute...

1. Features $\Phi(\mathbf{x})$ (only non-zero dims)
   - Non-Linearity Possible
   - Examples: Low Degree Polynomial Kernel, Spectrum Kernel, Weighted Degree Kernel
   - On-the-fly (de)compression
2. Virtual Examples
   - Incorporating Invariances possible
   - Examples: Image translation, rotation, etc

Needs efficient data structure for w!

- ..., dense, sorted array, trees, hashes
- fast only when $|\Phi_{\neq 0}(\mathbf{z})| \sim \mathit{dim}(\mathbf{z})$

Data Structures

Effort of **ADD** and **DOT** for $\mathbf{z}$ and memory requirement of $\mathbf{w}$.

	Dense	Sorted Array	Tree
Add	$\mathcal{O}(\lvert\Phi_{\neq 0}(\mathbf{z})\rvert)$	$\mathcal{O}(\lvert\mathbf{w}\rvert_{\neq 0}) + \lvert\Phi_{\neq 0}(\mathbf{z})\rvert)$	$\mathcal{O}(\Phi_{\neq 0}(\mathbf{z})\rvert)$ to $\mathcal{O}(K\Phi_{\neq 0}(\mathbf{z})\rvert)$
Dot	$\mathcal{O}(\lvert\Phi_{\neq 0}(\mathbf{z})\rvert)$	$\mathcal{O}(\lvert\mathbf{w}\rvert_{\neq 0}) + \lvert\Phi_{\neq 0}(\mathbf{z})\rvert)$	$\mathcal{O}(\Phi_{\neq 0}(\mathbf{z})\rvert)$ to $\mathcal{O}(K\Phi_{\neq 0}(\mathbf{z})\rvert)$
Mem	$\mathcal{O}(n)$	$\mathcal{O}(\sum_{i=1}^{m} \lvert\Phi_{\neq 0}(\mathbf{z}_i)\rvert)$	$\mathcal{O}(\sum_{i=1}^{m} \lvert\Phi_{\neq 0}(\mathbf{z}_i)\rvert)$

Sparse data structures have huge overhead!

Hashing to the Rescue (Shi et al (2009))

- Always use *dense* $\mathbf{w}$ with "compressed index"
- Hash function $h(J) \mapsto 1, \ldots, 2^{\gamma}$,
- $(\widehat{\Phi}(\mathbf{z}))_j = \sum_{i \in J; h(i)=j} (\Phi(\mathbf{z}))_i$

Data Structures

Effort of **ADD** and **DOT** for $\mathbf{z}$ and memory requirement of $\mathbf{w}$.

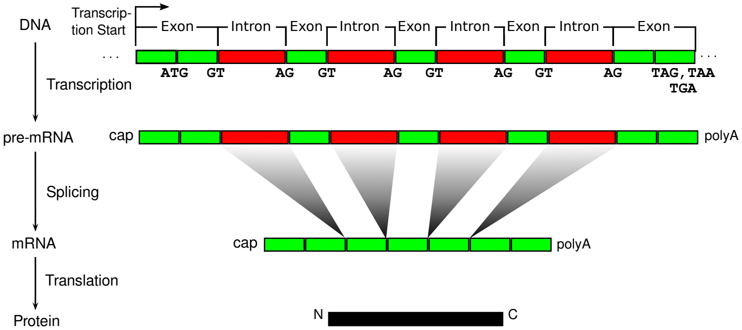
	Dense	Sorted Array	Tree
Add	$\mathcal{O}(\lvert\Phi_{\neq 0}(\mathbf{z})\rvert)$	$\mathcal{O}(\lvert\mathbf{w}\rvert_{\neq 0}) + \lvert\Phi_{\neq 0}(\mathbf{z})\rvert)$	$\mathcal{O}(\Phi_{\neq 0}(\mathbf{z})\rvert)$ to $\mathcal{O}(K\Phi_{\neq 0}(\mathbf{z})\rvert)$
Dot	$\mathcal{O}(\lvert\Phi_{\neq 0}(\mathbf{z})\rvert)$	$\mathcal{O}(\lvert\mathbf{w}\rvert_{\neq 0}) + \lvert\Phi_{\neq 0}(\mathbf{z})\rvert)$	$\mathcal{O}(\Phi_{\neq 0}(\mathbf{z})\rvert)$ to $\mathcal{O}(K\Phi_{\neq 0}(\mathbf{z})\rvert)$
Mem	$\mathcal{O}(n)$	$\mathcal{O}(\sum_{i=1}^{m} \lvert\Phi_{\neq 0}(\mathbf{z}_i)\rvert)$	$\mathcal{O}(\sum_{i=1}^{m} \lvert\Phi_{\neq 0}(\mathbf{z}_i)\rvert)$

Sparse data structures have huge overhead!

Hashing to the Rescue (Shi et al (2009))

- Always use *dense* $\mathbf{w}$ with "compressed index"
- Hash function $h(J) \mapsto 1, \ldots, 2^\gamma$,
- $(\widehat{\Phi}(\mathbf{z}))_j = \sum_{i \in J; h(i)=j} (\Phi(\mathbf{z}))_i$

Splice Site Predictions



Application to Human Acceptor Splice Site Prediction

Splice Site Prediction

Discriminate true signal positions against all other positions

≈ 150 nucleotides window around dimer

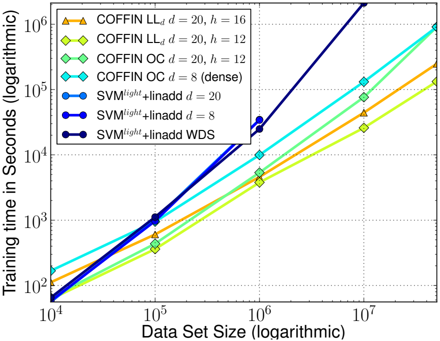
CT...GTCGTA...GAAGCTAGGAGCGC...ACGCGT...GA

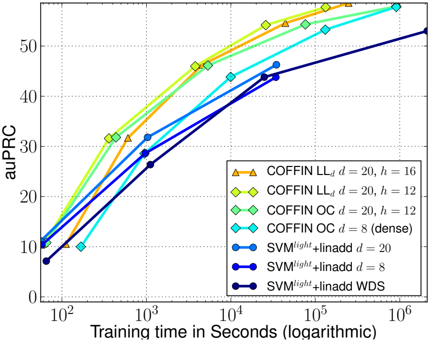
- True sites: fixed window around a true site
- Decoy sites: all other consensus sites

AAACAAATAAGTAACTAATCTTTTAGGAAGAACGTTTCAACCATTTTGAG
AAGATTAAAAAAAAACAAATTTTTAGCATTACAGATATAATAATCTAATT
CACTCCCCAAATCAACGATATTTTAGTTCACTAACACATCCGTCTGTGCC
TTAATTTCACTTCCACATACTTCCAGATCATCAATCTCCAAAACCAACAC
TTGTTTTAATATTCAATTTTTTACAGTAAGTTGCCAATTCAATGTTCCAC
TACCTAATTATGAAATTAAAATTCAGTGTGCTGATGGAAACGGAGAAGTC

- 50 million training examples
- COFFIN with kernels: weighted spectrum and weighted degree (explicit and hashed representation) $\approx 200,000,000$ dims

Results



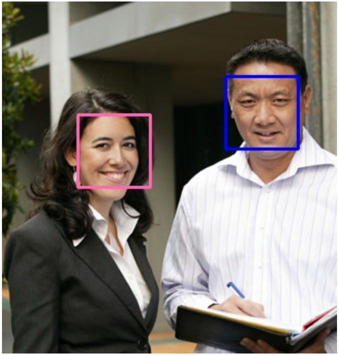


It's fast and works!

- Factor 47 faster on $10 \cdot 10^6$ examples than linadd
- New state-of-the-art results auPRC 58.57% vs. 53.01%

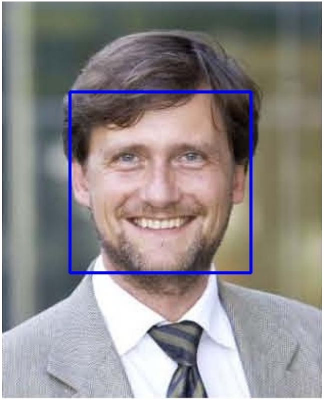
Gender Classification

Distinguish Females from Males solely based on Faces



- learn COFFIN on labelled faces
- virtual examples: translation, rotation, scale
- train $\approx$ 5 million sample (that would require 50GB) on Vojtechs notebook

Results I



It's fast and works - again!

- auROC 95.44%
- (vs. auROC 89.57% without VE)

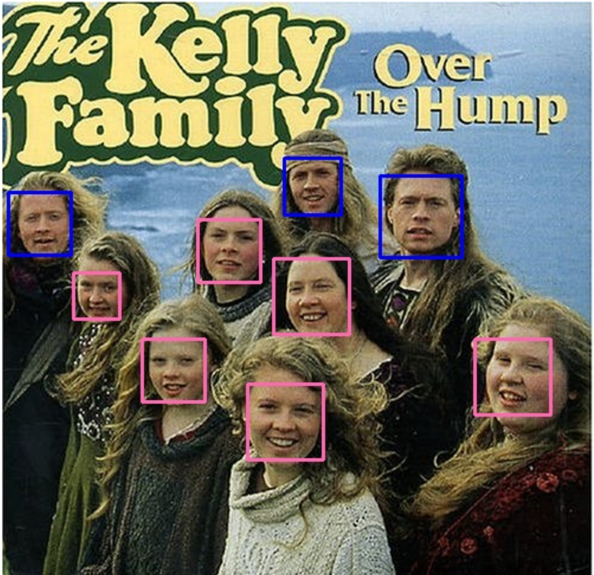
Klaus-Robert Müller is a male!

Results II


UEFA
EURO2008
Austria-Switzerland
FOOTBALL-WALLPAPERS.com

Results III

Results IV



Conclusions

COFFIN: Computational Framework for Linear SVMs

- Allows non-linearity
- Applicable to huge datasets
- General and often state-of-the art detectors

Datasets, Scripts, Efficient implementation

- Data and Scripts `http://sonnenburgs.de/soeren/coffin`
- Implementation `http://www.shogun-toolbox.org`
- More machine learning software `http://mloss.org`

Discussion

- Training on $\approx 2 \cdot 10^8$ dimensional $5 \cdot 10^7$ sample feasible
- Drastically reduced memory requirements; depending on features speed gain or speed penalty