Snacks: A Fast Large-Scale Kernel SVM Solver

Sofiane Tanji [1]   Andrea Della Vecchia [2]   François Glineur [1]   Silvia Villa [2]

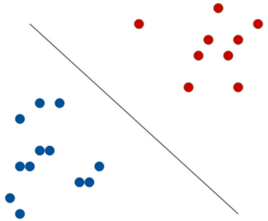
[1]UCLouvain, Belgium   [2]Università degli studi di Genova, Italy

European Control Conference, Bucarest, June 2023

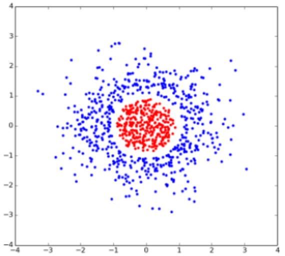
TRAINING DATA-DRIVEN EXPERTS IN OPTIMIZATION MSCA-ITN 2019

Context of binary classification

Goal: learn a prediction function $f : \mathcal{X} \to \mathcal{Y}$ given a labeled training dataset $(x_i, y_i)_{i=1}^n$ where $x_i \in \mathcal{X}, y_i \in \mathcal{Y} = \{-1, +1\}$ and such that $f(x_i) = y_i$ as often as possible for unknown (x_i, y_i)



f linear



f ?

From learning to optimization

To achieve this, we aim to solve the following optimization problem:

$$\min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^{n} L(y_i, f(x_i)) + \frac{\lambda}{2} \|f\|_2^2$$

where:

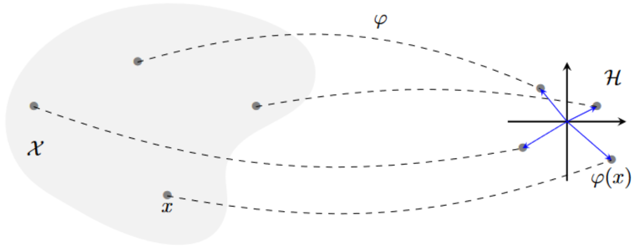
- L is a loss function (penalizes wrong predictions)
- a quadratic term is added to regularize the problem

Talk Outline

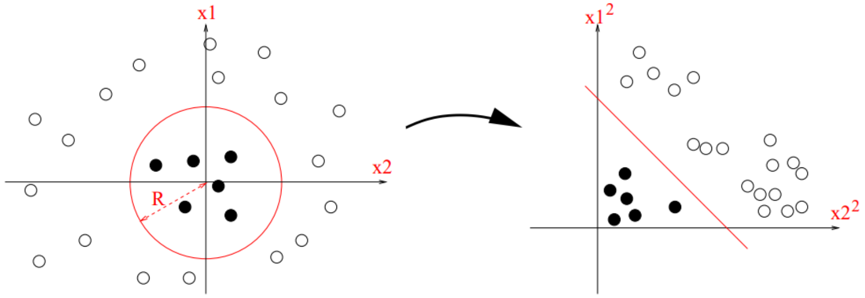
Kernel methods to the rescue

Data not linearly separable in input space ?
$\rightarrow$ Send data to "feature" space of higher dimension

- Map data x to high-dimensional Hilbert space with map $\varphi : \mathcal{X} \rightarrow \mathcal{H}$
- Functions $f \in \mathcal{H}$ are linear in features $f(x) = \langle f, \varphi(x) \rangle_{\mathcal{H}}$



A simple example



credit: Julien Mairal, Jean-Philippe Vert

Here, $K(x_1, x_2) = (x_1^\top x_2 + 1)^2$

Kernel methods for learning

- Functions $f \in \mathcal{H}$ are linear in features i.e. $f(x) = \langle f, \varphi(x) \rangle_{\mathcal{H}}$ with φ potentially infinite-dimensional
- To compare two points in $\mathcal{X}$, compute $K(x, y) = \langle \varphi(x), \varphi(y) \rangle_{\mathcal{H}}$
- Complete representation of comparisons between data points: kernel matrix $K = [K(x_i, x_j)]_{i,j}$
- Size of the kernel matrix is n^2 where n is the number of data points

From infinite dimensional to finite-dimensional

By the representer theorem, there exists a vector $\alpha \in \mathbb{R}^n$ such that:

$$f(x) = \sum_{i=1}^{n} \alpha_i K(x_i, x)$$

where f is solution of:

$$\min_{f \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} L(y_i, f(x_i)) + \lambda \|f\|_{\mathcal{H}}^2$$

What this talk is about

This talk is about solving

$$\min_{\alpha \in \mathbb{R}^n} \frac{1}{n} \sum_{i=1}^{n} L(y_i, [K\alpha]_i) + \lambda \alpha^T K \alpha \tag{1}$$

where $L(y_i, [K\alpha]_i) = \max(0, 1 - y_i[K\alpha]_i)$ is the hinge loss.
And tackling the following issues when solving the above problem:

- the hinge loss is not smooth

- the kernel matrix is of size $n \times n$: not scalable ⚠

Take-home messages so far

1. Learning problem: nonsmooth convex optimization problem
2. Representer theorem: makes the optimization problem finite dimensional
3. We want to accelerate optimization and tackle memory constraints

Talk Outline

Learning with kernels

Tackling the optimization problem

Tackling memory constraints

Numerical experiments

Conclusion

Nonsmooth optimization methods

The sub-gradient method

$$\begin{cases} x_1 \in \mathbb{R} \\ x^{k+1} = x^k - t_k g^k \quad \text{where} \quad g^k \in \partial f(x^k) \end{cases}$$

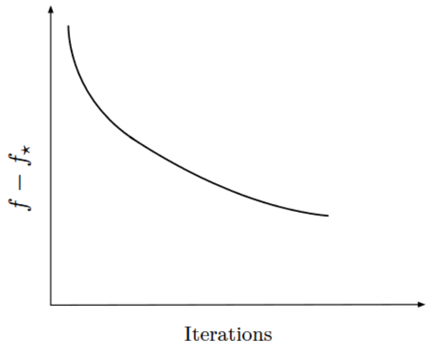
The stochastic sub-gradient method

$$\begin{cases} x_1 \in \mathbb{R} \\ x^{(k+1)} = x^{(k)} - t_k \tilde{g}^{(k)} \\ \text{where} \quad \mathbf{E}(\tilde{g}^{(k)} | x^{(k)}) = g^{(k)} \in \partial f(x^{(k)}) \end{cases}$$

We obtain an ε-optimal solution after $\mathcal{O}(\varepsilon^{-2})$ iterations

Restarts to accelerate optimizers

Optimization methods are faster at the beginning:



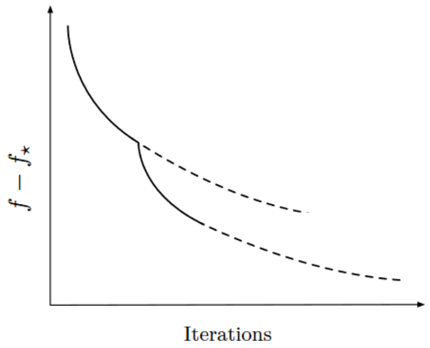


Figure taken from A. d'Aspremont et al., Acceleration Methods, ArXiv:2101.09545

How to restart, which schedule to choose ?

- It depends on: the loss function, the regularizer and the data.

- In our case, (1) is nonsmooth and sharp with parameter (μ, r)
  - Nonsmoothness: For all u, v, it holds: $\|\partial f(u) - \partial f(v)\| \leq M$
  - Sharpness: For every x, $\mu\|x - x^*\|^r \leq f(x) - f(x^*)$ where $x^* \in \arg\min f$ and $r = 2$

- Optimal and adaptative restart strategies can be found in [V. Roulet and A. d'Aspremont, Sharpness, Restart, Acceleration (NIPS 2017)]

ASSG: Accelerated Stochastic SubGradient

ASSG $\rightarrow$ **stochastic subgradient method** with **projections on a domain that shrinks** at each stage of an outer loop (= restarts)
The shrinking parameter is controlled by parameter r of the sharpness assumption

Two methods to adapt to unknown quantity μ:

- Log-scale grid search on possible schedules [Roulet et al. (2017)]
- *Projections on a domain that shrinks at each stage (ASSG)* [Xu et al. (2017)]

Convergence rate for ASSG

For Problem (1), the iteration complexity of ASSG for achieving an ε-optimal solution with high probability $1-\delta$ is:

$$\mathcal{O}\left(\frac{\log \delta^{-1}}{\varepsilon}\right)$$

Proof in [Xu et al. 2017].

Take-home messages so far

1. Kernel SVM amounts to a nonsmooth optimization problem
2. Large scale setting: we use stochastic subgradient + acceleration
3. To accelerate subgradient methods: restarts with domain shrinking

Talk Outline

Problem now scales !

The optimization method has a cheap cost per iteration and a better convergence rate.
But we still need to store the kernel matrix ! ($n \times n$, may be huge !) $\rightarrow$ **not possible** in large-scale settings !
Solution: **Nyström subsampling**

- Select m (with $m \leq n$) anchor points among the training data points.

- We search a solution in the basis formed by the anchor points, say $\tilde{f}(x) = \sum_{i=1}^{m} K(x, \tilde{x}_i)\tilde{c}_i \in span\{K_{\tilde{x}_1}, \dots, K\tilde{x}_m\}$

An embedding of the data

Define the embedding

$$x_i \mapsto \mathbb{x}_i = ((\tilde{K})^{1/2})^{\dagger}(K(\tilde{x}_1, x_i), \dots, K(\tilde{x}_m, x_i))^{\top}$$

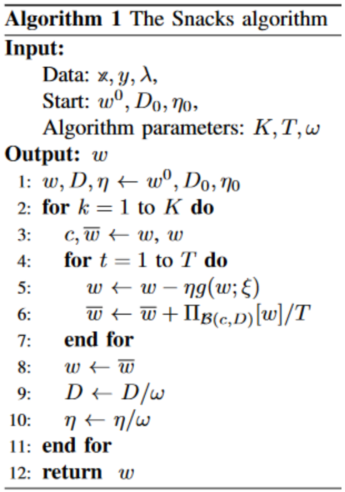
with $\tilde{K}_{i,j} = K(\tilde{x}_i, \tilde{x}_j)$.

This operation can be computed in $\mathcal{O}(m^3 + nm^2 C_K)$, with C_K the cost of evaluating one inner product.

SVM formulation with kernel embedding

$$\min_{c \in \mathbb{R}^m} \frac{1}{n} \sum_{i=1}^{n} \max(0, 1 - y_i \langle c, \mathbb{x}_i \rangle) + \frac{\lambda}{2} \|c\|_2^2 \qquad (2)$$

Snacks: ASSG applied to embedded dataset



Algorithm 1 The Snacks algorithm

```
Input:
    Data: $\mathbb{x}, y, \lambda$,
    Start: $w^0, D_0, \eta_0$,
    Algorithm parameters: $K, T, \omega$
Output: $w$
 1: $w, D, \eta \leftarrow w^0, D_0, \eta_0$
 2: for $k = 1$ to $K$ do
 3:     $c, \overline{w} \leftarrow w, w$
 4:     for $t = 1$ to $T$ do
 5:         $w \leftarrow w - \eta g(w; \xi)$
 6:         $\overline{w} \leftarrow \overline{w} + \Pi_{\mathcal{B}(c,D)}[w]/T$
 7:     end for
 8:     $w \leftarrow \overline{w}$
 9:     $D \leftarrow D/\omega$
10:     $\eta \leftarrow \eta/\omega$
11: end for
12: return $w$
```

K: number of restarts,
T: number of iterations per stage,
ω: shrinking coefficient,
D_0: initial ball diameter,
η_0: initial stepsize,
Picking i uniformly at random:

$$g(w; \xi^{k,t}) = \begin{cases} \lambda w - y_i \mathbb{x}_i & \text{if } y_i \langle w, \mathbb{x}_i \rangle < 1 \\ \lambda w & \text{else.} \end{cases}$$

How does it work step-by-step ?

1. Given a kernel function, compute the embedding of the data $\mathbb{x}$.
2. From an initial guess w^0, we generate a sequence of outer iterates w^k
3. To do so, T inner iterations are computed $w^{k,1}, \dots, w^{k,T}$.
4. To update $w^{k,t}$ to $w^{k,t+1}$, we perform the following step:

$$w^{k,t+1} \leftarrow \Pi_{\mathcal{B}(w^{k-1},D_k)}[w^{k,t} - \eta_k g(w^{k,t};\xi^{k,t})]$$

5. At the end of each stage, we shrink the stepsize η_k and the ball radius D_k by a factor ω.

Talk Outline

Datasets considered

Sizes of the original datasets and their corresponding kernel matrix

	ijcnn1	a9a	MNIST	rcv1	SUSY
# of points n	$5 \cdot 10^4$	$5 \cdot 10^4$	$6 \cdot 10^4$	$7 \cdot 10^5$	$5 \cdot 10^6$
# of features d	22	123	780	$4 \cdot 10^5$	18
Dataset (GiB)	0.01	0.05	0.37	263	0.72
Matrix K (GiB)	20	20	28.8	3900	$2 \cdot 10^5$

Other state-of-the-art SVM solvers

- LibLinear (on embedded dataset):
  Solves the dual formulation (a Quadratic Program) with a coordinate descent method

- Pegasos (on embedded dataset):
  Solves the primal formulation with a stochastic subgradient method

- ThunderSVM (on full dataset):
  Solves the dual formulation with a parallel coordinate descent method

Numerical experiments

a9a, $m = 800$. Kernel matrix precomputed in $2.2s$

a9a	Time (s)	C-err (optimal = 15.1 %)
LibLinear	39.0 s	15.8 %
ThunderSVM	2.97 s	15.6 %
Pegasos	52.0 s	20.0 %
Snacks	**1.01 s**	**15.2 %**

Numerical experiments

ijcnn1, $m = 5000$. Kernel matrix precomputed in $12.9s$

ijcnn1	Time (s)	C-err (optimal = 1.4 %)
LibLinear	67.1 s	1.8 %
ThunderSVM	31.2 s	**1.6 %**
Pegasos	1003.5 s	3.0 %
Snacks	**1.9 s**	**1.6 %**

Numerical experiments

mnist-bin, $m = 3000$. Kernel matrix precomputed in $31.6s$. Metric is F1-score.

mnist-bin	Time (s)	F1-score (optimal = 0.998)
LibLinear	19.9 s	**0.995**
ThunderSVM	23.6 s	**0.995**
Pegasos	91.8 s	0.982
Snacks	**14.6 s**	0.985

Numerical experiments

rcv1, $m = 1000$. Kernel matrix precomputed in $41.47s$.

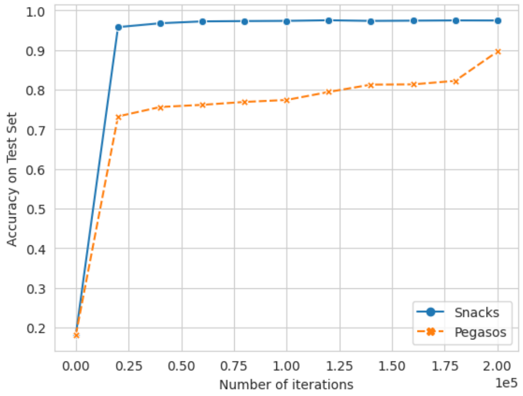
rcv1	Time (s)	C-err (optimal = 97.1 %)
LibLinear	1118 s	91.1 %
ThunderSVM	7779 s	96.9 %
Pegasos	61.3 s	93.7%
Snacks	**7.1 s**	95.6 %

Numerical experiments

SUSY, $m = 1000$. Kernel matrix precomputed in $74.1s$. ThunderSVM was stopped after 24 hours of training.

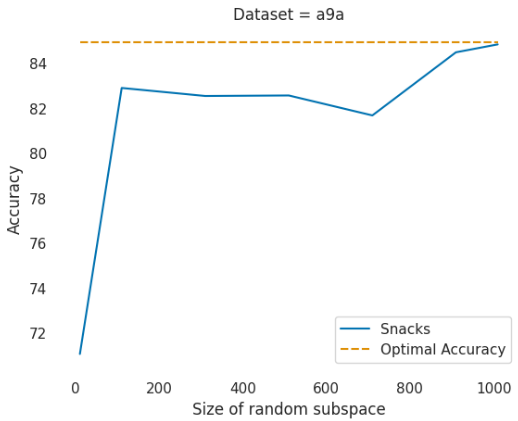
SUSY	Time (s)	C-err (optimal = 19.8 %)
LibLinear	9537 s	20.2 %
ThunderSVM	NaN	NaN
Pegasos	61.2 s	21.2 %
Snacks	**1.4 s**	**20.0 %**

Restarts accelerate convergence in practice !

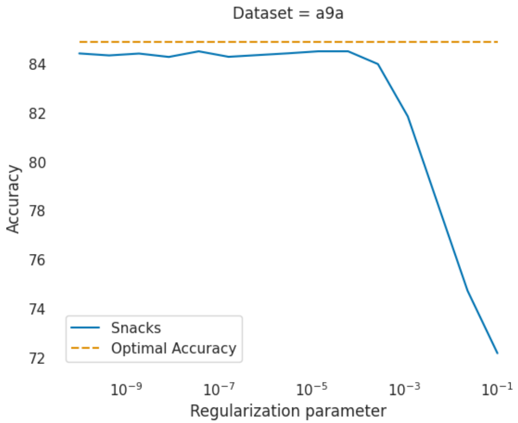


Comparison of accelerated primal and standard primal speed of convergence

Impact of SVM hyperparameters is limited



Subsampling parameter vs test accuracy



Regularization parameter vs test accuracy

Talk Outline

Summary and conclusion

Strategies used:

- Nyström subsampling
- Scheduled restarts to accelerate stochastic subgradient method

Benefits of Snacks:

- Simple implementation
- Handles huge datasets (with competitive runtime)
- Subsampling does not reduce accuracy

Snacks is available on Github

Python toolbox available on Github. It:

- follows scikit-learn's API
- handles large-scale dataset on a laptop with no GPU
- is benchmark-ready, you can easily add other solvers to compare

`https://github.com/sofianetanji/snacks`

Thank you for your attention!

Any questions ?

Related papers

- Tanji et al. (2023) Snacks: A Fast Large-Scale Kernel SVM Solver, ECC 2023.
- Della Vecchia et al. (2021) Regularized ERM on random subspaces, AISTATS 2021.
- Xu et al. (2017) Stochastic Convex Optimization: Faster Local Growth Implies Faster Global Convergence, ICML 2017.