Efficient Learning of Label Ranking by Soft Projections onto Polyhedra

Thomas Achenbach

Technische Universität Darmstadt

18. Januar 2008

Gegeben:

- Menge von Instanzen (z.B. Newsfeeds) aus *instance space* χ
- Endliche Menge von *Labeln*: γ mit $\gamma = \{1, 2, \ldots, k\} = [k]$
- Feeds können ein oder mehrere Label zugeordnet werden: *Mapping* von Feed auf Label
- Jedes Label hat für jeden Feed bestimmte Relevanz: *Preference* (Feed handelt z.B. hauptsächlich von Politik, ein wenig von Wirtschaft usw.)

Problem:
Wie *möglichst schnell* automatisch den Feeds die richtigen Labels mit den richtigen Präferenzen zuordnen???

Also Ziel:

- Exaktes Mapping von *instance space* (Newsfeeds) zum *target space* (labels).
- dabei richtige Voraussage der *label preferences*
- möglichst schnell

Hier: Lernen aus Trainingsbeispielen (batch learning)

Betonung auf *möglichst schnell*
Geht hier nur um Performance - Optimierung; dass Ziel erreichbar schon bekannt.
Also zuerst: ansehen, was genau optimiert werden soll
Dazu: Vorarbeiten mit Notation... :

Ausgangspunkt:

- Instanz aus instance space: $\mathbf{x} \in \chi, \chi \subseteq \mathbb{R}^n$
- Vordefinierte endliche Menge von Labeln γ mit $\gamma = \{1, 2, \ldots, k\} = [k]$

Lernziel: *Label Ranking Function*:

- $\mathbf{f} : \chi \rightarrow \mathbb{R}^k$ mit Rückgabewert:
  - $\vec{\gamma} \in \mathbb{R}^k$ target vector / label ranking
  - $\vec{\gamma}_i \in \mathbb{R}$
  - $f_r(\mathbf{x})$ rtes Element von $\vec{\gamma}$
  - $\vec{\gamma}_y > \vec{\gamma}_{y'} \Rightarrow \gamma$ relevanter für $\mathbf{x}$ als γ'
  - $\vec{\gamma}_y = \vec{\gamma}_{y'}$ möglich
  - Darstellung von $\vec{\gamma}$ als gerichteter gewichteter Graph möglich z.B. Edge (3,1): $\vec{\gamma}_3 - \vec{\gamma}_1 = 3$ bei $\vec{\gamma} = (-1, 0, 2, 0, -1)$

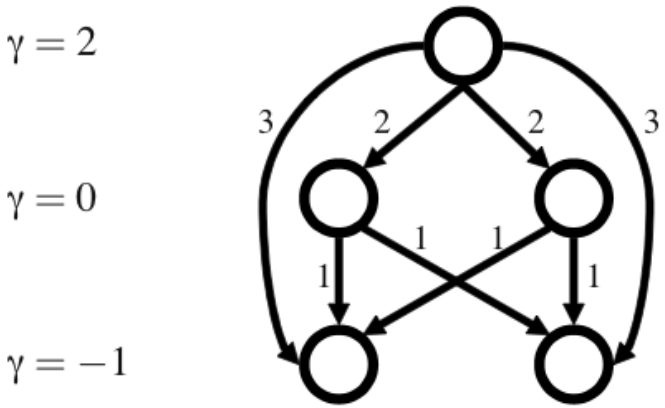


Figure 1: The graph induced by the feedback $\gamma = (-1, 0, 2, 0, -1)$.

- $f_\gamma(\mathbf{x}) > f_{\gamma'}(\mathbf{x}) \Rightarrow \gamma$ relevanter für $\mathbf{x}$ als γ'
- $f_r(\mathbf{x}) = \mathbf{w}_r \cdot \mathbf{x}$; $\mathbf{w}_r \in \mathbb{R}^n, \chi \subseteq \mathbb{R}^n \Rightarrow$ lineare Funktion (Das aber keine echte Einschränkung (SVM Kernel trick...)

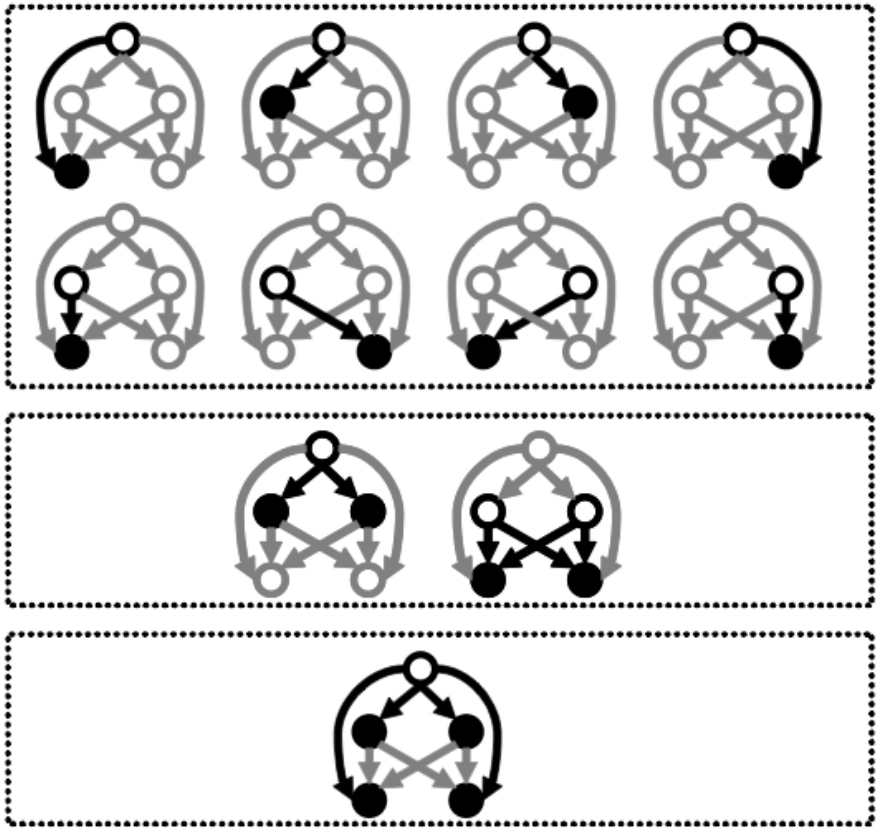
- Trainingsbeispiel: $S = \{(\mathbf{x}^i, \vec{\gamma}^i)\}_{i=1}^m$; $\mathbf{x} \in \chi, \vec{\gamma}^i \in \mathbb{R}^k$

Performance via *Loss Function* evaluiert
(Notation: $(a)_+ = max\{0, a\}$):

- $\ell : \mathbb{R}^k \times \mathbb{R}^k \to \mathbb{R}$
  - $\ell_{r,s}(\mathbf{f}(\mathbf{x}), \vec{\gamma}) = (((\vec{\gamma}_r - \vec{\gamma}_s) - (f_r(\mathbf{x}) - f_s(\mathbf{x})))_+$
  - $\ell_{r,s}$ zeigt wieweit constraint $f_r(\mathbf{x}) - f_s(\mathbf{x}) \geq \vec{\gamma}_r - \vec{\gamma}_s$ nicht berücksichtigt wird

Bis hierher nur paarweiser Vergleich der Label (r, s)
Jetzt: Alle paarbasierten losses in einen loss zusammenfassen!

Paare von Labeln in d unabhängige Mengen aufteilen. Jede Menge isomorph zu vollst. biparititem Graphen



- loss eines Subgraphen (V_j, E_j): maximum über den losses der Paare im Subgraphen
- zusätzl. Gewichtung der einzelnen Subgraphen mit (nichtnegativem) σ_j

$\Longrightarrow$ Loss:

$$\ell(\mathbf{f}(\mathbf{x}), \vec{\gamma}) = \sum_{j=1}^{d} \sigma_j max_{(r,s)\in E_j} \ell_{r,s}(\mathbf{f}(\mathbf{x}), \vec{\gamma})$$

Mit loss-function jetzt label-ranking-function bilden:

- constrained optimization problem definieren (mit SVM Paradigma)
- optimale Lösung davon: label-ranking-function mit 2 Termen:
  1. empirischer loss der label-ranking-function bezgl. Trainingsset
  2. penalty für Komplexität der Funktion (Regularisierungsterm), i.e. Summe der Quadrate der Normen von $\{\mathbf{w}_1, \ldots, \mathbf{w}_k\}$
  3. Tradeoff zwischen beiden Termen durch Parameter C kontrolliert

$$min_{\mathbf{w}_1,\ldots,\mathbf{w}_k} \frac{1}{2} \sum_{j=1}^{k} \|\mathbf{w}_j\|^2 + C \sum_{i=1}^{m} \ell(\mathbf{f}(\mathbf{x}^i), \vec{\gamma}^i) \quad mit : f_\gamma(\mathbf{x}^i) = \mathbf{w}_\gamma \cdot \mathbf{x}^i$$

Und andere Notation der loss-function:

$$\ell(\mathbf{f}(\mathbf{x}), \vec{\gamma}) = min_{\xi \in \mathbb{R}^d_+} \sum_{j=1}^{d} \sigma_j \xi_j$$

mit: $\forall j \in [d], \forall (r,s) \in E_j, \mathbf{f}_r(\mathbf{x}) - \mathbf{f}_s(\mathbf{x}) \geq \vec{\gamma}_r - \vec{\gamma}_s - \xi_j$

Ergibt das quadratische Optimierungsproblem:

$$min_{\mathbf{w}_1,\ldots,\mathbf{w}_k,\xi} \frac{1}{2} \sum_{j=1}^{k} \|\mathbf{w}_j\|^2 + C \sum_{i=1}^{m} \sum_{j=1}^{|\mathbf{E}(\vec{\gamma}^i)|} \sigma_j \xi_j^i$$

mit: $\forall i \in [m], \forall E_j \in \mathbf{E}(\vec{\gamma}^i), \forall (r,s) \in \mathbf{E}_j, \mathbf{w}_r \cdot \mathbf{x}^i - \mathbf{w}_s \cdot \mathbf{x}^i \geq \vec{\gamma}_r^i - \vec{\gamma}_s^i - \xi_j^i \quad \forall i,j,\xi_j^i \geq 0$

Ziele Optimierung:

- schnell
- soll mit allen Dekompositionen in Subgraphen zurechtkommen.

Herleitung des Algorithmus komplex!

Grober Ablauf:

1. Iteration über die Trainingsbeispiele
2. Iteration über die Subgraphen
3. dort die ranking-function für neues Beispiel verbessern

Kern des Algorithmus:
3. Punkt: ranking-function für neues Beispiel verbessern:
Dabei Ausgangsposition:

- schon label-ranking-function vorhanden ($\{\mathbf{u}_1, \ldots, \mathbf{u}_k\}$)
- $\mathbf{E}(\vec{\gamma})$ besteht aus nur einem vollständigen bipartiten Graphen
- Ziel: Funktion verbessern wenn neues Beispiel verarbeitet wird

Ergibt constrained optimization problem:

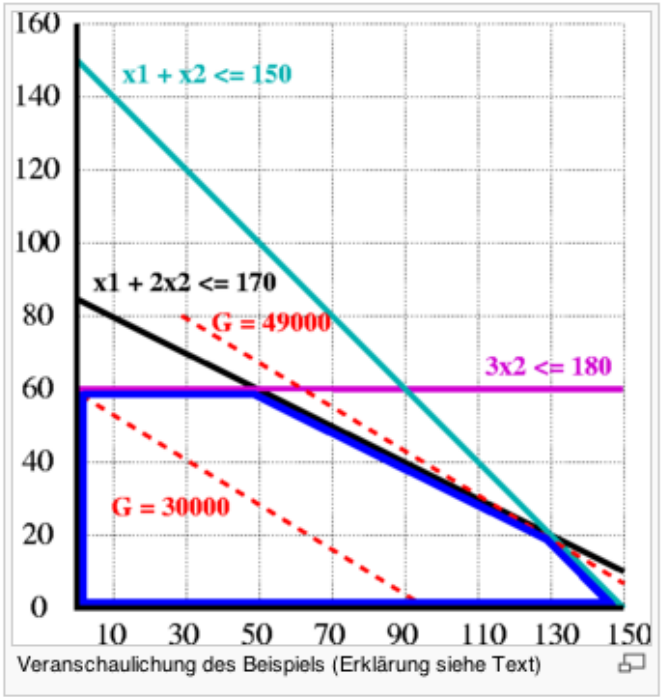
$$min_{\mathbf{w}_1,\ldots,\mathbf{w}_k,\xi} \frac{1}{2} \sum_{y=1}^{k} \|\mathbf{w}_y - \mathbf{u}_y\|^2 + C\xi$$

mit: $\forall (r,s) \in E, \mathbf{w}_r \cdot \mathbf{x} - \mathbf{w}_s \cdot \mathbf{x} \geq \vec{\gamma}_r - \vec{\gamma}_s - \xi, \xi \geq 0$

- label-ranking function schon vorhanden, repräsentiert durch $\{\mathbf{u}_1, \ldots, \mathbf{u}_k\}$
- Erinnerung: label-ranking-function $f_r(\mathbf{x}) = \mathbf{u}_r \cdot \mathbf{x}$; $\mathbf{u}_r \in \mathbb{R}^n, \chi \subseteq \mathbb{R}^n$

Problemlösung kann aufgefasst werden als: Projektion von $\{\mathbf{u}_1, \ldots, \mathbf{u}_k\}$ auf das Polyeder, dass mit den Beschränkungen Definiert wird.
Daher Name des Algorithmus: SOft-Projection Onto POlyhedra SOPOPO

Erläuterung zur Projektion (aus Wikipedia):



Veranschaulichung des Beispiels (Erklärung siehe Text)

türkis, schwarz, violett sind die Beschränkungen; erlaubte punkte im blauen Polyeder; gestrichelte Rote sollen optimiert werden (soweit wie möglich nach rechts schieben)

Lösung des Problems in mehreren Schritten:

- Duales Problem finden
- Anzahl m der Variablen im dualen Problem von $k^2/4 \geq m$ auf k reduzieren
- Aufteilung des Problems in zwei einfachere

Ergebnis: Komplexität von $O(k \log(k))$

Duales Problem:

- primäre Zielfunktion ist konvex
- primäre constraints sind linear
- Es gibt Lösung für primäres Problem (setze $\mathbf{w}_y = 0$ und $\xi = max_{(r,s)\in E}(\vec{\gamma}_r - \vec{\gamma}_s)$

$\Longrightarrow$ strong duality

Finden des Problems:
Lagrange des Primärproblems bilden:

$$\begin{aligned}
\mathcal{L} &= \frac{1}{2}\sum_{y=1}^{k}\|\mathbf{w}_y-\mathbf{u}_y\|^2 + C\xi + \sum_{(r,s)\in E}\tau_{r,s}\left(\gamma_r-\gamma_s-\xi+\mathbf{w}_s\cdot\mathbf{x}-\mathbf{w}_r\cdot\mathbf{x}\right)-\zeta\xi \\
&= \frac{1}{2}\sum_{y=1}^{k}\|\mathbf{w}_y-\mathbf{u}_y\|^2 + \xi\left(C-\sum_{(r,s)\in E}\tau_{r,s}-\zeta\right)+\sum_{(r,s)\in E}\tau_{r,s}\left(\gamma_r-\mathbf{w}_r\cdot\mathbf{x}-\gamma_s+\mathbf{w}_s\cdot\mathbf{x}\right) \quad ,
\end{aligned}$$

mit: $\forall(r,s)\in E : \tau_{r,s}\geq 0, \zeta\geq 0$

- teilterme fallen weg
- Primaervariablen können beseitigt werden
- Umformung führt zu Ergebnis
- Dann Anzahl der Variablen im dualen Problem auf k reduzieren.
- Problem in zwei einfache Teilprobleme aufteilen

Zur Ansicht:

INPUT: instance $\mathbf{x} \in X$; target ranking γ ; sets A, B

current prototypes $\mathbf{u}^1, \ldots, \mathbf{u}^k$; regularization parameter C

MARGINS:

$$\mu = \text{sort}\left\{(\gamma_a - \mathbf{u}^a \cdot \mathbf{x})/\|\mathbf{x}\|^2 \mid a \in A\right\}$$
$$\nu = \text{sort}\left\{(\mathbf{u}^b \cdot \mathbf{x} - \gamma_b)/\|\mathbf{x}\|^2 \mid b \in B\right\}$$

KNOTS:

$$\forall i \in [p] : z_i = \textstyle\sum_{r=1}^{i} \mu_r - i\mu_i \quad \forall j \in [q] : \tilde{z}_j = \sum_{s=1}^{j} \nu_s - j\nu_j$$
$$Q = \{z_i : z_i < C\} \cup \{\tilde{z}_j : \tilde{z}_j < C\} \cup \{C\}$$

INTERVALS:

$$\forall z \in Q : \; R(z) = |\{z_i : z_i \le z\}| \; ; \; S(z) = |\{\tilde{z}_j : \tilde{z}_j \le z\}|$$
$$\forall z \in Q : \; N(z) = \min\{z' \in Q : z' > z\} \cup \{C\}$$

LOCAL MIN:

$$O(z) = \left(S(z) \sum_{r=1}^{R(z)} \mu_r + R(z) \sum_{r=1}^{S(z)} \nu_r\right) / (R(z) + S(z))$$

GLOBAL MIN:

If $(\exists z \in Q$ s.t. $O(z) \in [z, N(z)])$ **Then**

$z^* = O(z)$; $i^* = R(z)$; $j^* = S(z)$

Else If $(\mu_1 + \nu_1 \le 0)$

$z^* = 0$; $i^* = 1$; $j^* = 1$

Else

$z^* = C$; $i^* = R(C)$; $j^* = S(C)$

DUAL'S AUXILIARIES:

$$\theta_\alpha = \frac{1}{i^*}\left(\sum_{r=1}^{i^*} \mu_r - z^*\right) \; ; \; \theta_\beta = \frac{1}{j^*}\left(\sum_{r=1}^{j^*} \nu_r - z^*\right)$$

OUTPUT:

$$\forall a \in A : \; \alpha_a = \left(\tfrac{\gamma_a - \mathbf{u}_a \cdot \mathbf{x}}{\|\mathbf{x}\|^2} - \theta_\alpha\right)_+ \text{ and } \mathbf{w}_a = \mathbf{u}_a + \alpha_a \mathbf{x}$$
$$\forall b \in B : \; \beta_b = \left(\tfrac{\mathbf{u}_b \cdot \mathbf{x} - \gamma_b}{\|\mathbf{x}\|^2} - \theta_\beta\right)_+ \text{ and } \mathbf{w}_b = \mathbf{u}_b - \beta_b \mathbf{x}$$

Weiter im Algorithmus:

- Jetzt nur eine Projektion für einen einzigen vollständigen bipartiten Graphen
- Algorithmus soll aber auf jeder beliebigen Aufteilung funktionieren
- Gesucht also Algorithmus der Urspruengliche Aufgabe löst, indem er immer wieder SOPOPO verwendet
  - ursprüngliches Problem vereinfachen (syntax) $\Rightarrow$

$$min_{\mathbf{w'},\xi} \frac{1}{2}\|\mathbf{w'}\|^2 + \sum_{i=1}^{m} C_i \xi_i$$

Mit Beschränkungen

Für die Vereinfachung:

- Duales Problem suchen (Lagrange...)
- Dann Algorithmus der in Runden arbeitet:
  - In jeder Runde wird eine Menge von dualen Variablen upgedatet
  - alle anderen variablen sind fix

Ansicht des Algorithmus in Pseudocode:

INPUT: training set $\{(\mathbf{x}^i, \gamma^i)\}_{i=1}^m$; decomposition function $\mathbf{E}(\gamma)$;
regularization parameter C

INITIALIZE:

$\forall i \in [m], A_j \times B_j \in \mathbf{E}(\gamma^i), (a,b) \in A_j \times B_j,$ set $\alpha_a^{i,j} = 0, \beta_b^{i,j} = 0$

$\forall r \in [k]$, set $\mathbf{w}_r = \mathbf{0}$

LOOP:

Choose a sub-graph $i \in [m], A_j \times B_j \in \mathbf{E}(\gamma^i)$

UPDATE:

$$\forall a \in A_j : \mathbf{u}_a = \mathbf{w}_a - \alpha_a^{i,j}\mathbf{x}_i \quad \forall b \in B_j : \mathbf{u}_b = \mathbf{w}_b + \beta_b^{i,j}\mathbf{x}_i$$

SOLVE:

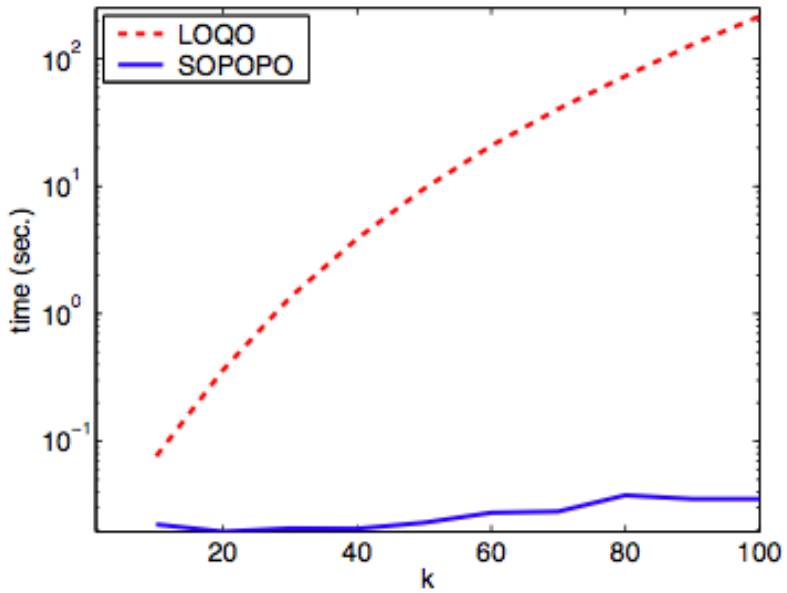
$$(\alpha^{i,j}, \beta^{i,j}, \{\mathbf{w}_r\}) = \text{SOPOPO}(\{\mathbf{u}_r\}, \mathbf{x}^i, \gamma^i, A_j, B_j, C\sigma_j^i)$$

OUTPUT: The final vectors $\{\mathbf{w}_r\}_{r=1}^k$

LOQO: Kommerzielles Paket zur linearen Optimierung, basiert auf interior point Methode
Angesteuert mit Matlab-Interface, Implementierung selbst: C++
SOPOPO: Implementiert in Matlab
Testdaten immer zufällig gewählt (Normalverteilt)

Erster Test: Performance Soft-projection auf ein Polyeder / originäres Problem

$$\min_{\mathbf{w}_1,\ldots,\mathbf{w}_k,\xi} \quad \frac{1}{2}\sum_{y=1}^{k} \|\mathbf{w}_y - \mathbf{u}_y\|^2 + C\xi$$
$$\text{s.t. } \forall (r,s) \in E, \quad \mathbf{w}_r \cdot \mathbf{x} - \mathbf{w}_s \cdot \mathbf{x} \geq \gamma_r - \gamma_s - \xi$$
$$\xi \geq 0 \ .$$

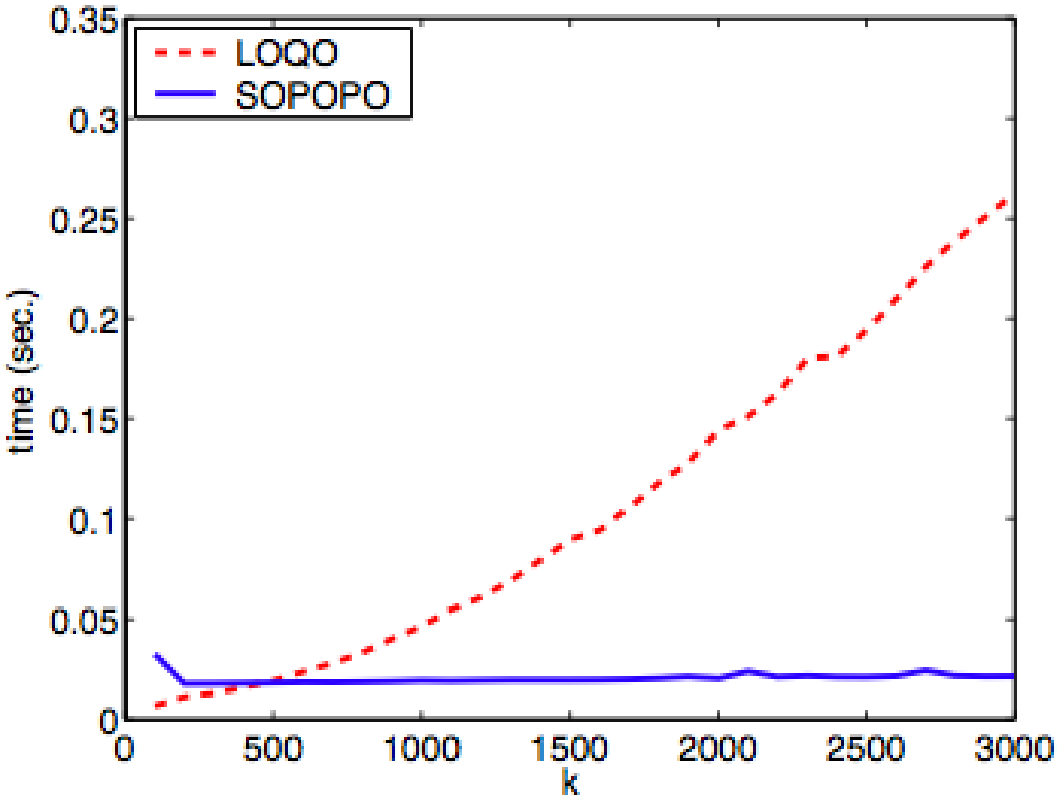


Aber: Test 'unfair', da:

- LOQO ist general purpose, nimmt daher immer interial point Algorithmus
- SOPOPO optimiert immer auf:

$$\min_{\alpha \in \mathbb{R}_+^p, \beta \in \mathbb{R}_+^q} \quad \frac{1}{2}\|\alpha - \mu\|^2 + \frac{1}{2}\|\beta - \nu\|^2$$
$$\text{s.t.} \quad \sum_{i=1}^{p} \alpha_i = \sum_{j=1}^{q} \beta_j \leq C \ .$$

Daher LOQO direkt auch auf das optimierte Problem angesetzt:



Gesamttest batchlearning:
Jeweils $10x$ mehr Beispiele als Labels (k) generiert