A GPU Optimized Hierarchical Decomposition Algorithm for Support Vector Machine Training

Ing. Josef Michálek

2018

Osnova

Cíle práce

- V posledních letech se ve strojovém učení používají zejména neuronové sítě
- SVM rychleji natrénují model pro menší a středně velké datasety, mají méně hyperparametrů, pouze jedno řešení a regularizace je ovlivněná jediným parametrem
- Stávající algoritmy trénování SVM jsou několik let staré a neumí využít rychle se vyvíjející hardware
- Cíl práce je navrhnout a implementovat algoritmus trénování SVM tak, aby byl vhodný pro dnešní grafické karty
- Navrhnout efektivní verzi algoritmu pro hustá i řídká data

Co je SVM

- SVM je metoda strojového učení s učitelem
- Používá se ke klasifikaci i k regresi
- Trénovací množina obsahuje prvky náležející do první nebo druhé třídy. Trénovací algoritmus SVM pak vytvoří model, který predikuje, do které třídy patří nové prvky.

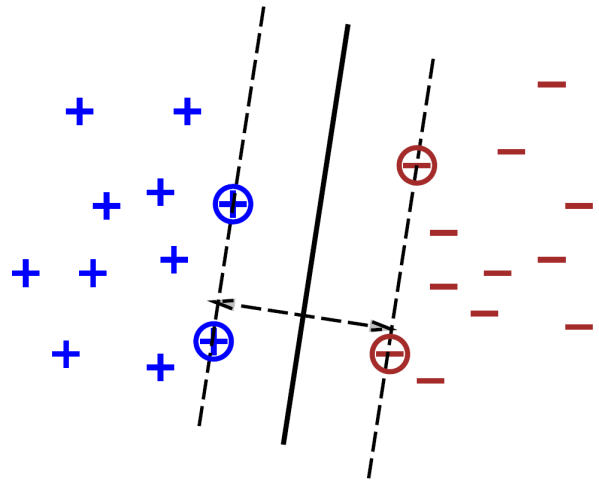
Lineární SVM

- Trénovací data označíme

  $\{x_i, y_i\},\ i = 1, \ldots, N,\ y_i \in \{-1, 1\},\ x_i \in \mathbb{R}^d$

- Předpokládáme existenci nadroviny oddělující pozitivní prvky od negativních ve tvaru

  $$w \cdot x + b = 0,$$



- Nejkratší vzdálenost od prvků na každé straně roviny označíme d_+ a d_-.
- Součet $d_+ + d_-$ se nazývá šířka hraničního pásma (*margin*).

Cíl trénovacího algoritmu SVM

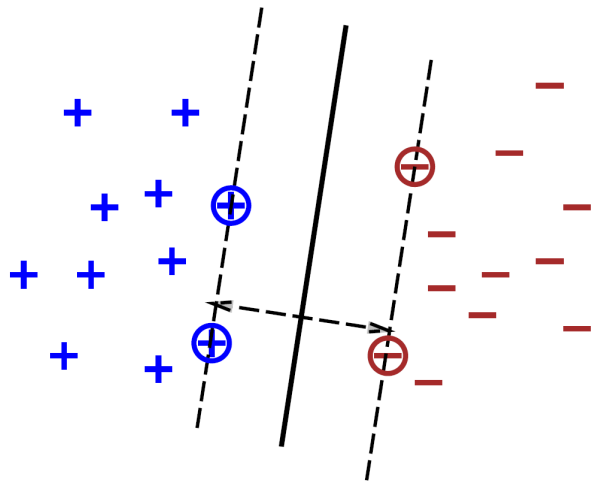
Nalézt takovou rozdělující nadrovinu, která má největší šířku hraničního pásma.

Formulace problému

- Předpokládejme, že trénovací množina splňuje

$$x_i \cdot w + b \geq +1 \quad \text{pro } y_i = +1 \qquad (1)$$

$$x_i \cdot w + b \leq -1 \quad \text{pro } y_i = -1 \qquad (2)$$



- Podmínky lze sloučit do

$$y_i(x_i \cdot w + b) - 1 \geq 0 \quad \forall i \qquad (3)$$

- Prvky splňující rovnost z (1) leží na nadrovině $H_1 : x_i \cdot w + b = +1$
- Prvky splňující rovnost z (2) leží na nadrovině $H_2 : x_i \cdot w + b = -1$
- Šířka hraničního pásma (vzdálenost H_1 od H_2) je $2/||w||$.

Formulace problému

Minimalizace $||w||^2/2$ vzhledem k (3).

- Podpůrné vektory (*support vectors*) – prvky pro které platí rovnost z (3).

Lagrangeovská formulace

- Zavení ceny za porušení omezení označené $\xi_i, \quad i = 1, \ldots, N$. Nová omezení pak mají tvar

$$y_i(\boldsymbol{x}_i \cdot \boldsymbol{w} + b) - 1 + \xi_i \geq 0, \quad \xi_i \geq 0 \quad \forall i \tag{4}$$

- Ztrátová funkce se změní na $||\boldsymbol{w}||^2/2 + C\sum_i \xi_i$, kde C je parametr volený uživatelem.

Primární Lagrangeovská formulace problému

Minimalizace $$L_P = \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_i \xi_i - \sum_{i=1}^{N} \alpha_i(y_i(\boldsymbol{x}_i \cdot \boldsymbol{w} + b) - 1 + \xi_i) - \sum_{i=1}^{N} \mu_i \xi_i \tag{5}$$

vzhledem k w a b, derivace L_P vzhledem ke všem α_i musejí být nulové a $\alpha_i \geq 0 \quad \forall i$

Duální formulace

- Minimalizace L_P je problém kvadratického programování, lze tedy řešit duální problém hledání maxima L_P za podmínek, že gradient L_P vzhledem k w a b je nulový a $\alpha_i \geq 0 \quad \forall i$.
- Aby byl gradient L_P vzhledem k w a b nulový, musí platit

$$w = \sum_i \alpha_i y_i x_i \tag{6}$$

$$\sum_i \alpha_i y_i = 0 \tag{7}$$

Po dosazení těchto omezení do 5 získáme:

Duální Lagrangeovská formulace problému

$$\text{Maximalizace } L_D = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \boldsymbol{x}_i \cdot \boldsymbol{x}_j \tag{8}$$

vzhledem k α_i při omezení $0 \leq \alpha_i \leq C \ \forall i$ a $\sum_i \alpha_i y_i = 0$

Nelineární SVM

- Definujeme jádrovou funkci (*kernel function*) K jako

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \Phi(\boldsymbol{x}_i) \cdot \Phi(\boldsymbol{x}_j) \tag{9}$$

  kde Φ je transformační funkce z původního prostoru do jiného Euklidovského prostoru H.

$$\Phi : \mathbb{R}^d \to H \tag{10}$$

- **Kernel trick** – V trénovacím algoritmu můžeme použít pouze hodnoty jádrové funkce K, aniž bychom znali vztah pro výpočet Φ.

Duální Lagrangeovská formulace problému

$$\text{Maximalizace } L_D = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) \tag{11}$$

vzhledem k α_i při omezení $0 \leq \alpha_i \leq C \ \forall i$ a $\sum_i \alpha_i y_i = 0$

- Jedná se o kvadratický problém – **existuje pouze jedno optimum**

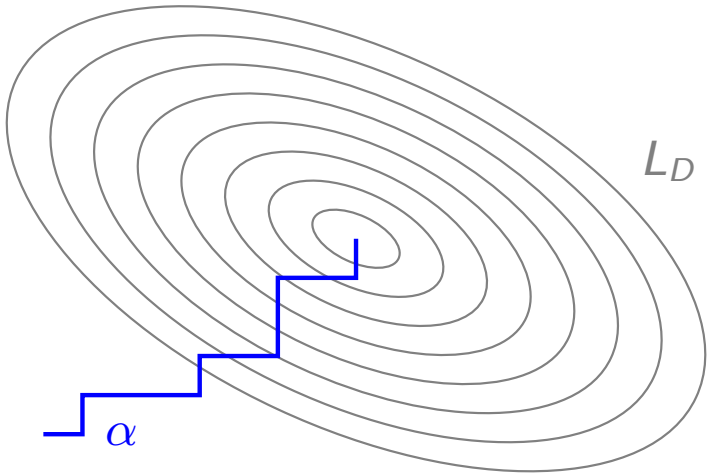
Jádrové funkce

Příklady jádrových funkcí

Lineární	$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i \cdot \boldsymbol{x}_j$
Polynomiální	$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = (a\boldsymbol{x}_i \cdot \boldsymbol{x}_j + b)^d$
RBF (Radial basis function)	$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp(-\gamma \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2)$
Sigmoid	$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \tanh(a\boldsymbol{x}_i \cdot \boldsymbol{x}_j + r)$

Dekompozice problému

- Kvadratický problém – lze řešit jako několik menších podproblémů
- V každém podproblému optimalizujeme pouze některé dimenze vektoru řešení α
- Po konečném počtu kroků dosáhneme stejného řešení jako u původního problému



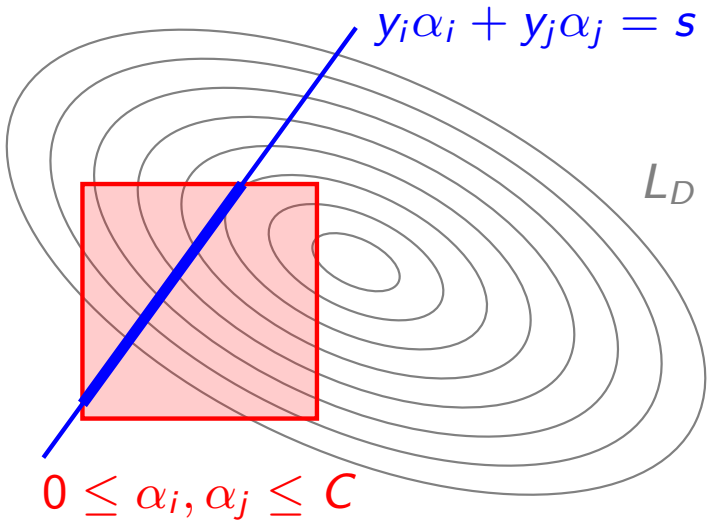
Sequential Minimal Optimization (SMO)

- Extrémní verze dekompoziční metody používající nejmenší možný podproblém.
- Z omezení $\sum_k \alpha_k y_k = 0$ plyne, že nejmenší možný podproblém zahrnuje dva Lagrangeovy multiplikátory. Omezení se pak pro jakékoliv dva multiplikátory α_i, α_j změní na

$$0 \leq \alpha_i, \alpha_j \leq C \tag{12}$$

$$y_i \alpha_i + y_j \alpha_j = s \tag{13}$$

  kde s je záporný součet všech ostatních $y_k \alpha_k$.
- Lze řešit analyticky, je třeba najít minimum jednorozměrné kvadratické funkce.



Výběr pracovního páru prvků pro SMO

- Jedna iterace SMO posune řešení α ve směru u^{ij}, tj. vektor s nenulovými hodnotami na pozicích i, j odpovídajících prvkům pracovního páru
- Nejefektivnější směr je takový, který maximalizuje růst duální funkce

$$u^* = \underset{u^{ij}}{\operatorname{argmax}} \max_{0 \leq \lambda} L_D(\alpha + \lambda u^{ij}) - L_D(\alpha) \tag{14}$$

při omezení $0 \leq y_i \alpha_i + \lambda \leq C$ a $0 \leq y_j \alpha_j - \lambda \leq C$.

- Vztah pro u^* nahradíme aproximací 1. nebo 2. řádu a dostaneme vztahy pro volbu pracovního páru

Heuristika 1. řádu

$$i = \underset{k}{\operatorname{argmax}}\, y_k g_k \tag{15}$$

$$j = \underset{k}{\operatorname{argmin}}\, y_k g_k \tag{16}$$

Heuristika 2. řádu

$$i = \underset{k}{\operatorname{argmax}}\, y_k g_k \tag{17}$$

$$j = \underset{k}{\operatorname{argmax}} \frac{(y_i g_i - y_k g_k)^2}{2(K_{ii} + K_{kk} - 2K_{ik})}, \quad y_i g_i > y_k g_k \tag{18}$$

kde $g_k = \frac{\partial L_D(\alpha)}{\partial \alpha_k} = 1 - y_k \sum_n y_n \alpha_n K_{kn}$

Optimized Hierarchical Decomposition SVM

Cíl práce

- Navrhnout a implementovat algoritmus trénování SVM tak, aby byl vhodný pro dnešní grafické karty
- Navrhnout efektivní verzi algoritmu pro hustá i řídká data

Předpoklady pro vhodný algoritmus

- Zpracovat najednou co nejvíce dat
- Převést co nejvíce kroků algoritmu na maticové nebo vektorové operace
- Vhodně pracovat s kernelovou maticí, protože se celá nevejde do paměti
- Všechny části algoritmu na GPU, maximálně omezit komunikaci mezi CPU a GPU

Popis algoritmu OHD-SVM

Rozdělení algoritmu do dvou úrovní:

1. **Lokální úroveň** – algoritmus založený na SMO pracující s omezenou pracovní množinou. Celý běží bez přerušení na jádře GPU.
2. **Globální úroveň** – vybírá z trénovací množiny pracovní množinu o předem dané velikosti *WS*. Tu pak optimalizuje lokální úroveň algoritmu. Výběr pracovní množiny vychází od Serafiniho[1], ale zbytek jeho algoritmu je na GPU nevhodný.

Všechny části mého algoritmu běží na GPU, včetně správy cache kernelové matice.

[1] Serafini, T., & Zanni, L. (2005). On the working set selection in gradient projection-based decomposition techniques for support vector machines. Optimization Methods and Software, 20(4-5), 583-596.

Lokální úroveň algoritmu OHD-SVM

Pseudokód lokální úrovně

1: Zkopírování hodnot y_k, g_k, α_k, ws_k do registrů $\forall k$
2: $local\_iter \leftarrow 0$
3: **loop**
4: Volba páru prvků i, j pomocí heuristiky 1. řádu
5: **if** $y_i * g_i - y_j * g_j < \epsilon$ **then** ▷ Test konvergence
6: $\rho \leftarrow -(y_i * g_i + y_j * g_j)/2$
7: **return** $local\_iter, \rho, \alpha$
8: **end if**
9: Volba prvku j pomocí heuristiky 2. řádu
10: Optimalizace α_i, α_j
11: Aktualizace gradientu $g_k \forall k$
12: $local\_iter \leftarrow local\_iter + 1$
13: **end loop**

Globální úroveň algoritmu OHD-SVM

Pseudokód globální úrovně

1: Spočtení diagonály kernelové matice K
2: $global\_iter \leftarrow 0$
3: **loop**
4: **if** $global\_iter = 0$ **then**
5: Volba náhodné pracovní množiny
6: **else**
7: Volba nové pracovní množiny
8: **end if**
9: Spočtení kernelové matice pro pracovní množinu
10: $\alpha' \leftarrow \alpha$
11: $local\_iter, \rho, \alpha \leftarrow$ Lokální úroveň algoritmu
12: **if** $local\_iter = 0$ **then**
13: **return** ρ, α
14: **end if**
15: $\Delta\alpha \leftarrow \alpha - \alpha'$
16: Spočtení všech řádků i kernelové matice K a jejich uložení do cache $\forall \Delta\alpha_i \neq 0$
17: Aktualizace g s využitím $\Delta\alpha$
18: $global\_iter \leftarrow global\_iter + 1$
19: **end loop**

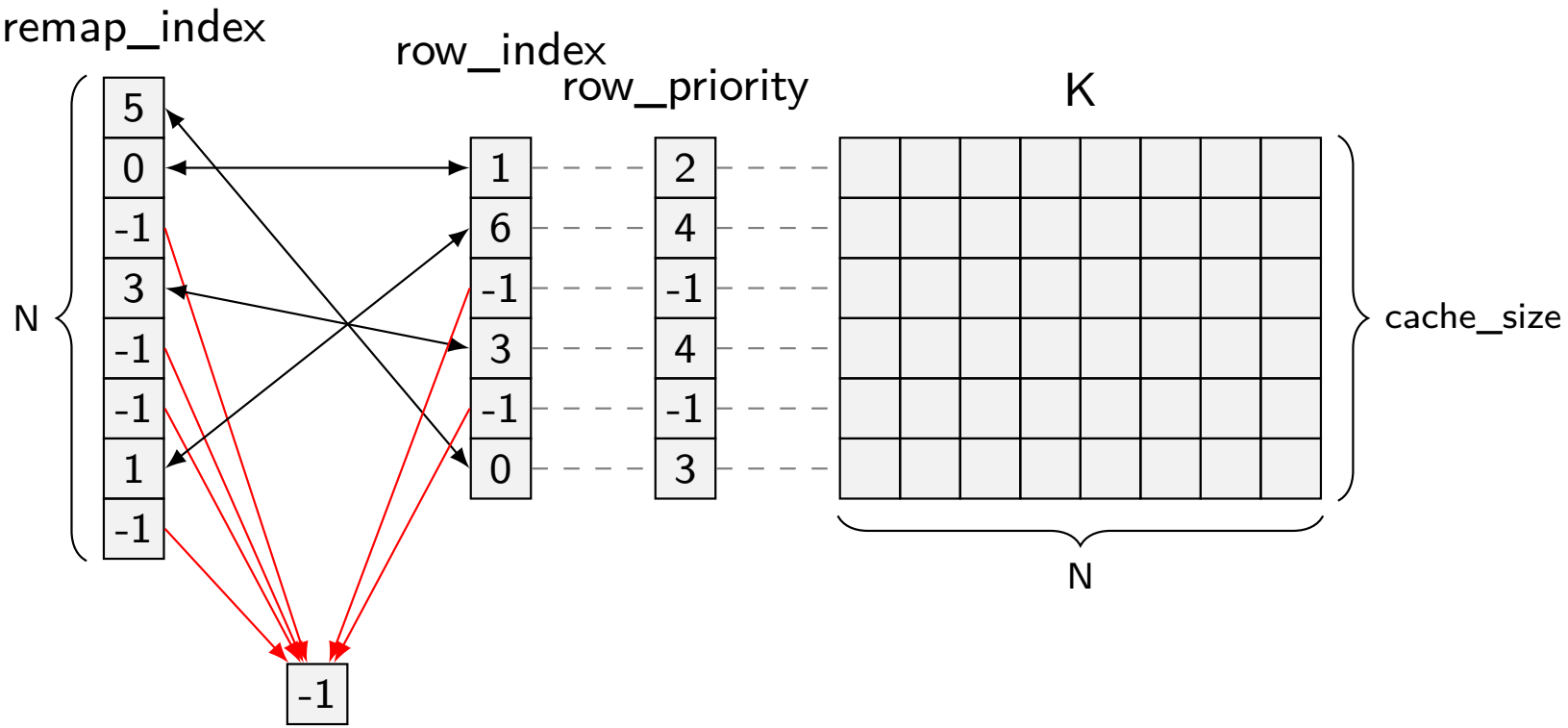
Metoda výběru pracovní množiny

Metoda výběru pracovní množiny

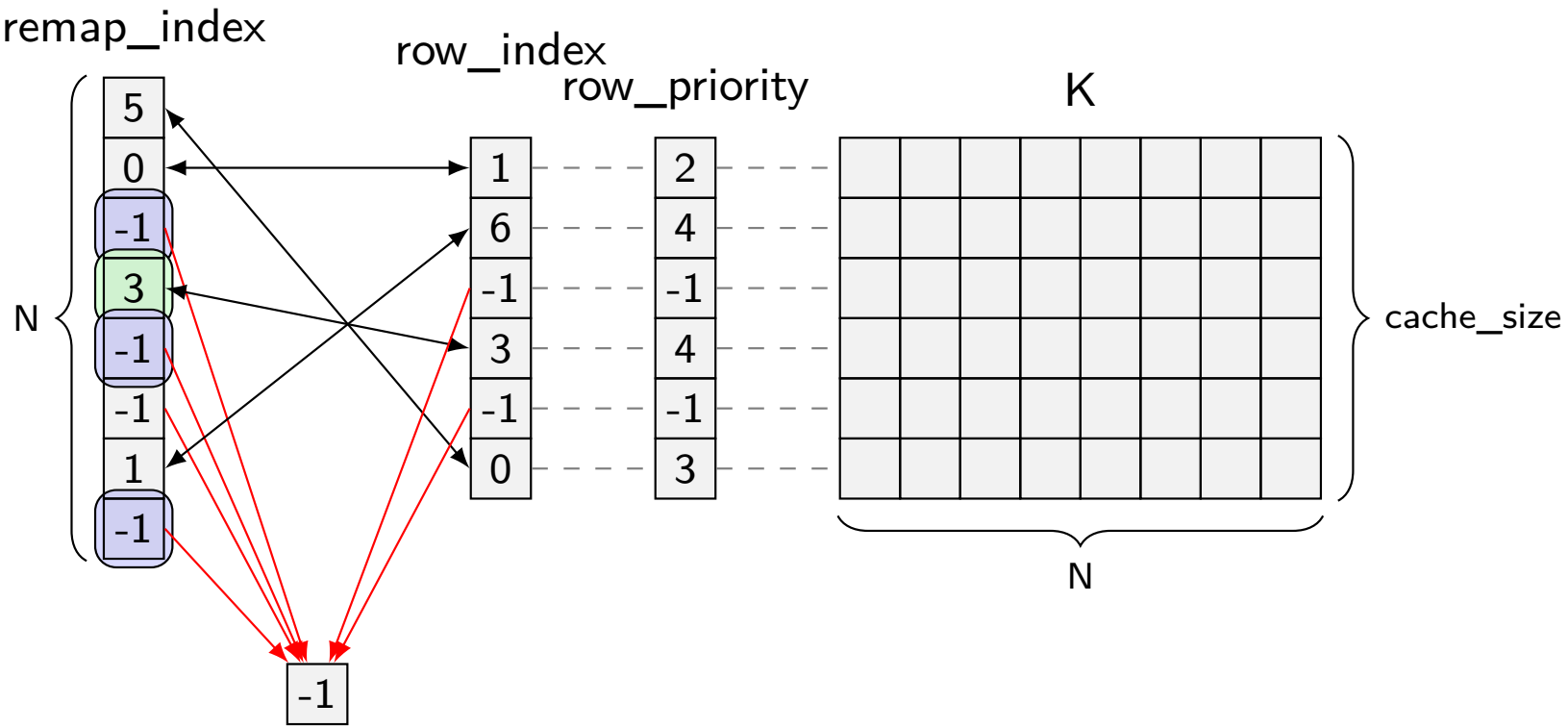
1: Výběr poloviny nové pracovní množiny pomocí heuristiky 1. řádu
2: Seřazení prvků ve staré pracovní množině vzestupně podle počtu globálních iterací, ve kterých byly zvolené v pracovní množině
3: Postupné zaplnění nové pracovní množiny prvky ze staré pracovní množiny. Nejdříve se volí volné vektory, poté omezené zdola a nakonec omezené shora.

- V každé globální iteraci se zvolí až polovina nových prvků, ostatní se použijí ze staré pracovní množiny.
- Předpokládá se, že optimalizovaná pracovní množina po změně některých prvků už optimalizovaní není, ale je blízko řešení.

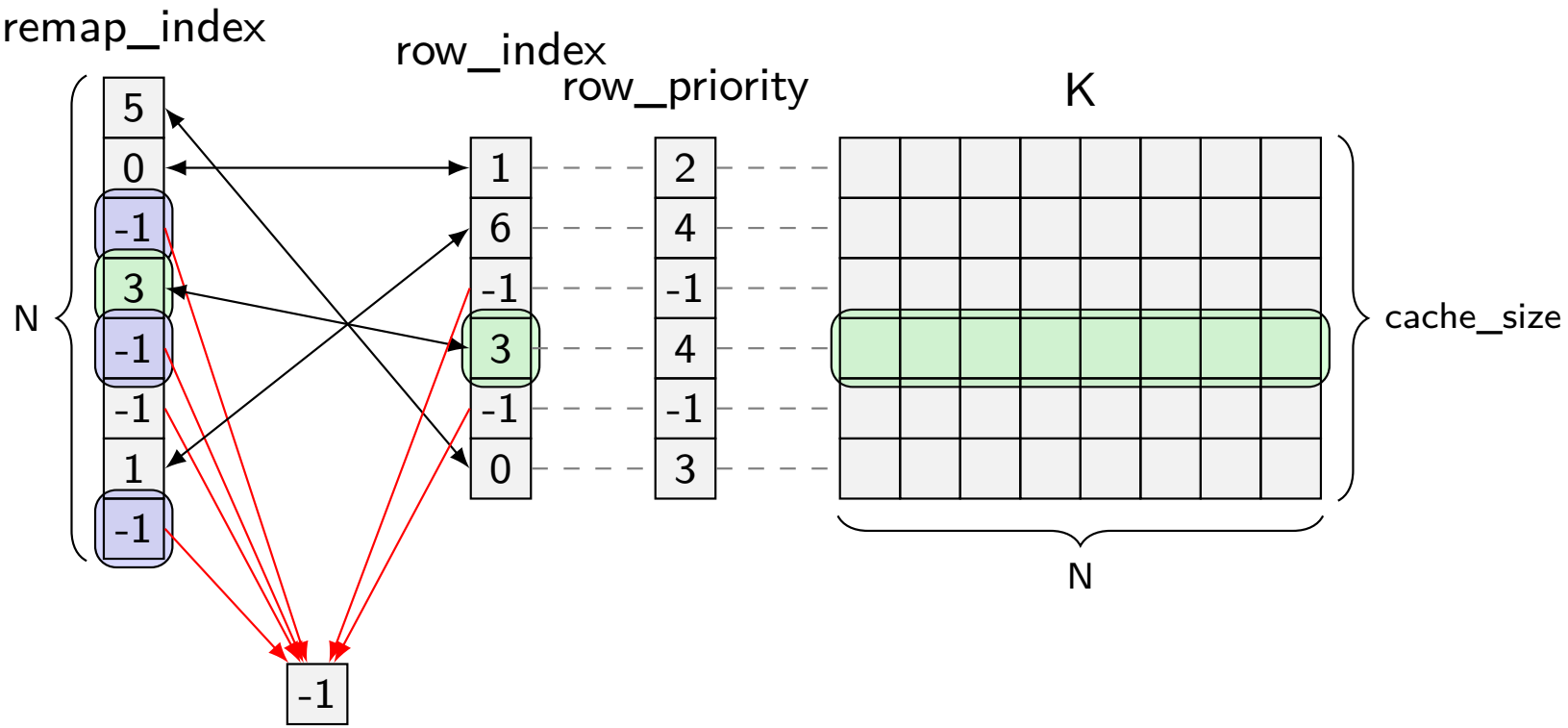
Cache kernelové matice



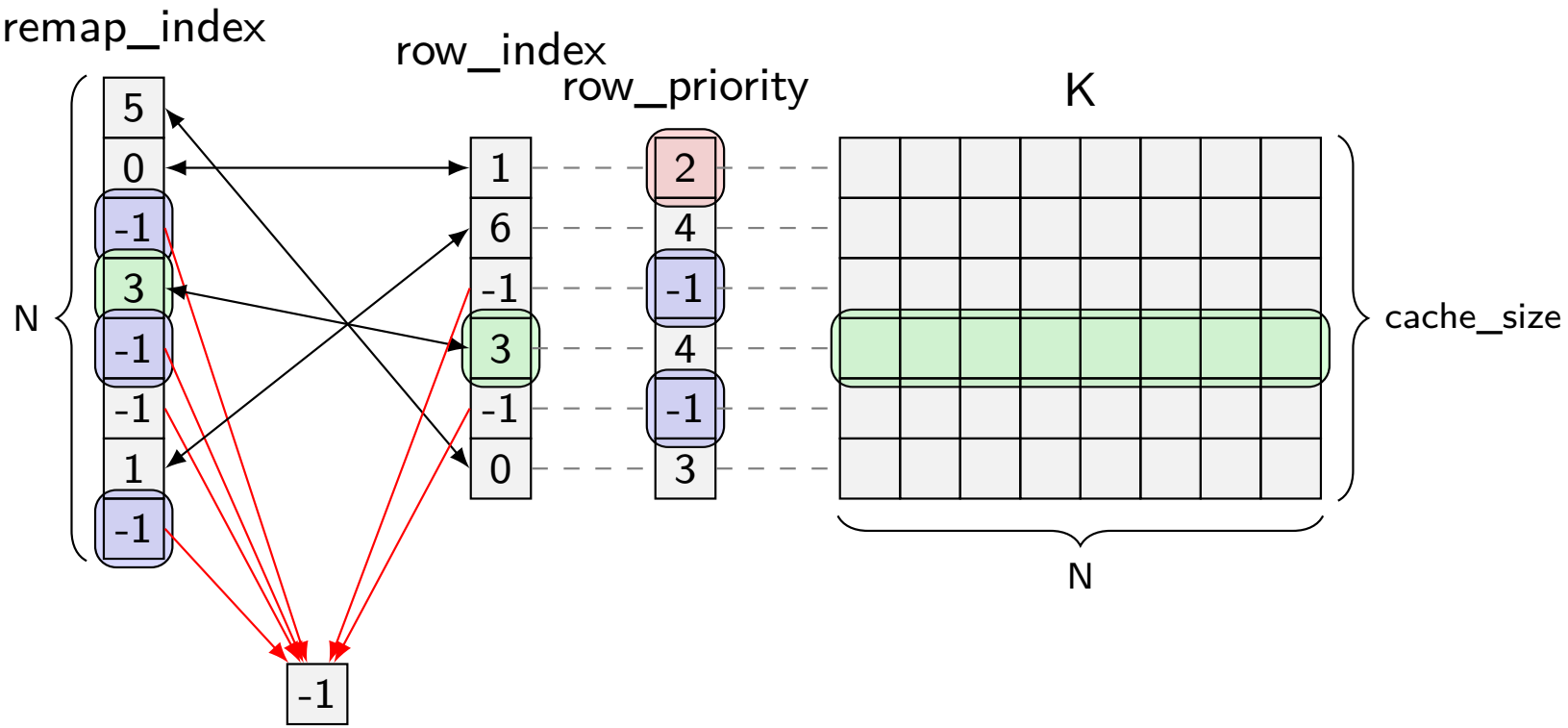
Cache kernelové matice



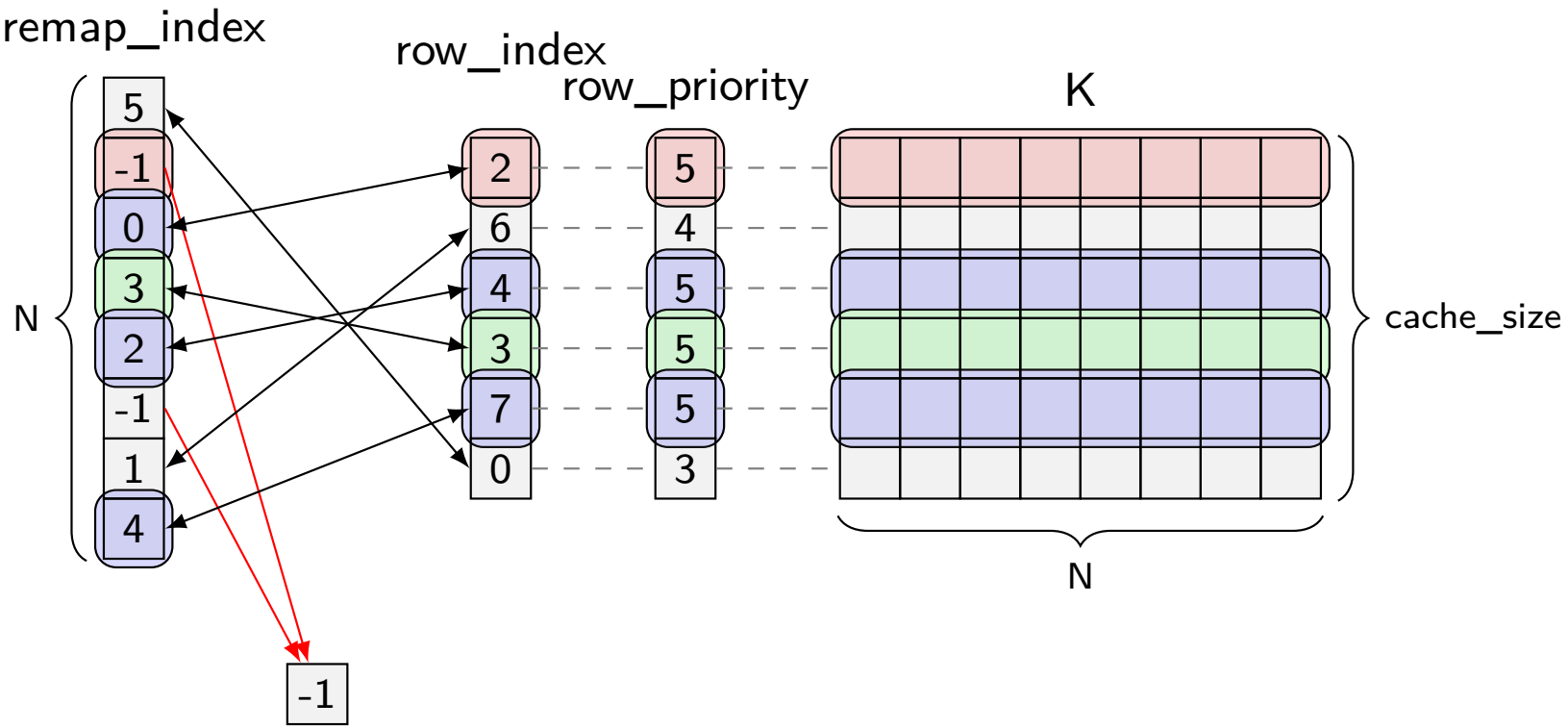
Cache kernelové matice



Cache kernelové matice



Cache kernelové matice



Výsledky

Metodika získání výsledků

- Našel jsem a zprovoznil všechny dosud publikované open-source implementace trénování SVM na GPU
- Vytvořil jsem program umožňující natrénování SVM modelu ze stejných dat pomocí jakékoliv použité implementace včetně mé
- Naměřil jsem čas trénování na stejných datasetech, které byly použity autory ostatních implementací v jejich článcích
- Byl použit PC s Intel Core i7-4790K @ 4 GHz, 32 GB RAM @ 1600 MHz, NVIDIA GTX 1080

Publikované výsledky

- J. Vaněk, J. Michálek and J. Psutka, "A GPU-Architecture Optimized Hierarchical Decomposition Algorithm for Support Vector Machine Training," IEEE Transactions on Parallel and Distributed Systems, vol. 28, no. 12, pp. 3330-3343, Dec. 1 2017.
- Ocenění Best Poster Award za prezentaci průběžných výsledků na letní škole paralelního programování v Barceloně PUMPS 2016, součástí ocenění byla grafická karta NVIDIA Tesla K20

Porovnání času trénování s různými implementacemi

Tab.: Čas trénování pro husté datasety [s]

Implementace	Epsilon	Alpha	Adult	Web	COV1	MNIST	TIMIT
cuSVM	39.25	101.38	17.45	22.02	265.27	11.92	17.04
gpuSVM	28.04	26.26	3.36	7.58	301.04	5.89	2.86
multiSVM	51.34	90.04	18.69	22.52	–	12.22	17.73
wuSVM	122.08	61.96	131.56	13.61	596.86	79.21	203.86
gtSVM LC	47.80	31.45	3.19	4.02	–	3.72	2.43
gtSVM SC	77.22	36.38	3.37	4.43	–	4.55	3.17
OHD-SVM	**2.31**	**5.77**	**1.69**	**2.43**	**29.53**	**1.35**	**1.80**

Tab.: Čas trénování pro řídké datasety [s]

Implementace	Adult	Web	COV1	MNIST	News20	RCV1	Real-Sim
gtSVM LC	3.30	4.03	–	**3.69**	605.53	15.83	53.66
gtSVM SC	3.44	4.27	–	4.60	486.83	10.11	20.81
KMLib	18.65	19.30	874.65	21.55	145.01	10.02	55.80
OHD-SVM JDS	2.15	**3.08**	119.01	5.37	38.57	2.08	6.83
OHD-SVM EllR-T	2.04	3.53	**78.50**	4.22	25.33	1.85	6.18
OHD-SVM EllR-T*	**2.00**	3.35	**78.50**	4.15	**25.27**	**1.78**	**5.85**

Statistiky využití GPU

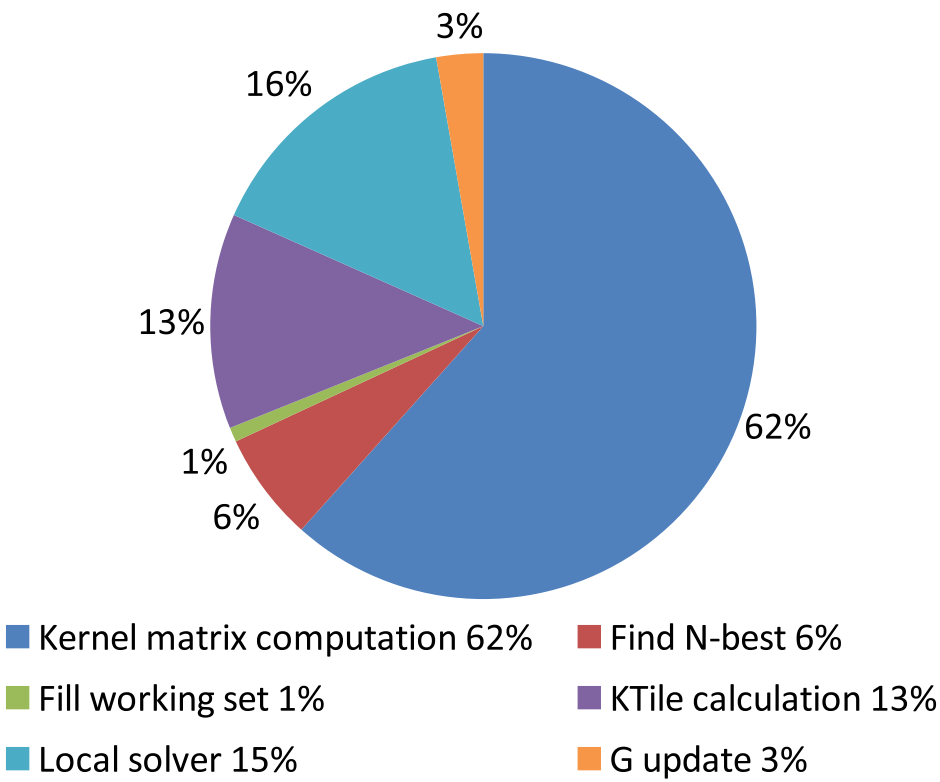
Tab.: Hustý dataset Epsilon

Implementace	Trénovací čas [s]	# TFLOP	Celková načtená data	L2 Hit rate	% Sdílená paměť	PCI-E Transakce Počet	PCI-E Transakce Celková velikost	PCI-E Transakce Celkový čas [ms]	Využití GPU
gpuSVM	28.04	4.52	13.97 TB	50.01 %	9.95 %	60,393	613.00 MB	1,184.77	86.33 %
OHD-SVM	2.31	3.8	0.77 TB	50.70 %	41.04 %	113	307.93 MB	68.94	61.65 %

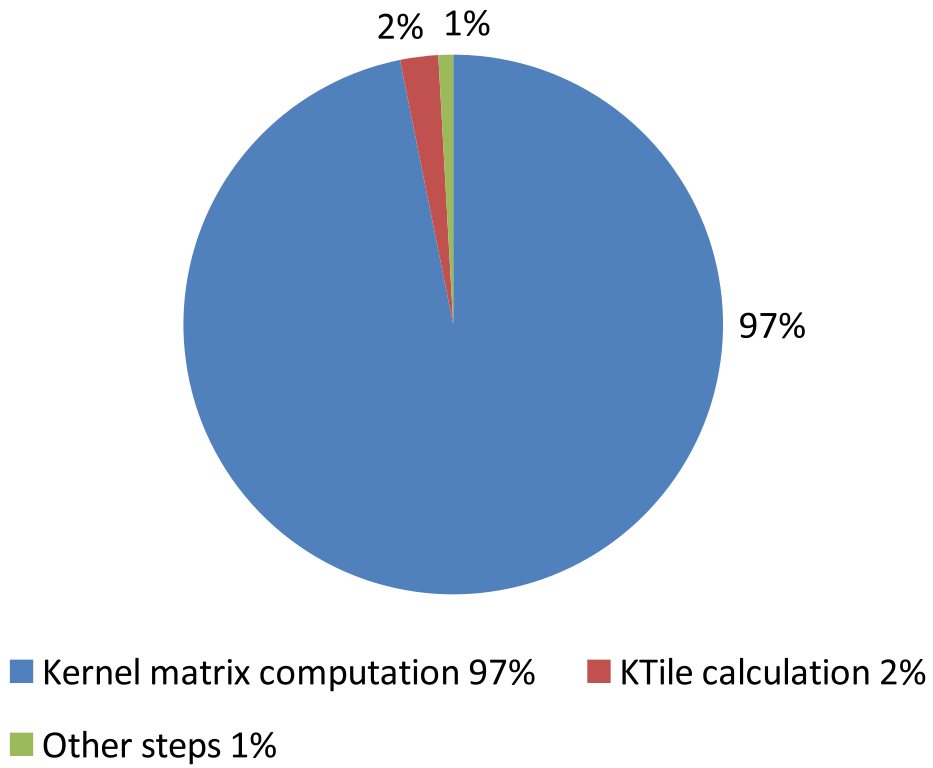
Tab.: Řídký dataset News20

Implementace	Trénovací čas [s]	# TFLOP	Celková načtená data	L2 Hit rate	% Sdílená paměť	PCI-E Transakce Počet	PCI-E Transakce Celková velikost	PCI-E Transakce Celkový čas [ms]	Využití GPU
gtSVM LC	605.53	170.93	141.21 TB	50.13 %	55.93 %	32,282	293.15 GB	24,279.40	23.91 %
gtSVM SC	486.83	14.19	24.55 TB	49.95 %	59.29 %	34,772	289.09 GB	23,924.11	9.02 %
OHD-SVM JDS	38.57	0.22	9.02 TB	74.06 %	0.06 %	23	139.27 MB	107.22	96.98 %
OHD-SVM EllR-T	25.33	0.24	6.73 TB	71.77 %	1.15 %	22	632.32 MB	107.38	95.67 %

Relativní čas jednotlivých částí trénovacího algoritmu



Obr.: Hustý dataset Epsilon



Obr.: Řídký dataset News20

Pokračování práce

- Akustické modely pomocí hlubokých neuronových sítí v kombinaci se SVM.
  - SVM lépe zobecňují neviděná data, lze to ovlivnit volbou C.
  - Neuronové sítě i SVM se obojí trénují na GPU.
- Upravit navržený algoritmus trénování SVM i na jiné grafické karty