Scalable Online Training with Conjunctive Features

Naoki Yoshinaga and Masaru Kitsuregawa

IIS, The University of Tokyo

Proposal

- Kernel slicing for online training with conjunctive features
  - explicitly consider conjunctions among frequent features, while implicitly considering the others by polynomial kernel
  - reuse temporal margins of partial feature vectors
- Performance evaluation on two NLP tasks
  (dependency parsing and hyponymy relation extraction)
  - orders of magnitudes faster than kernel-based online training, while retaining its space efficiency
  - model accuracy: comparable to batch SVM

Overview

- Research Backgrounds
  - Space-time trade-off in training with conjunctive features
  - Kernel splitting [Goldberg+ '08] for testing
- Methods
  - Online learning with kernel splitting
  - Online learning with kernel slicing
- Experiments
- Conclusion

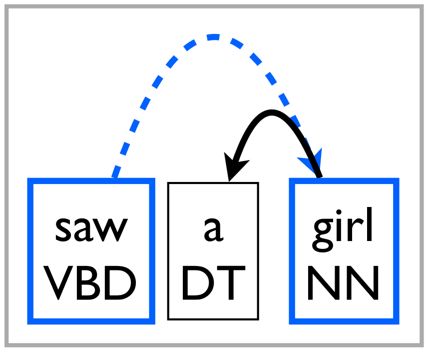
Conjunctive features in NLP

- Conjunctive features play a key role to obtain a high degree of accuracy in NLP classification problems
  - dependency parsing [Koo+ '08], pronoun resolution [Nguyen+, 08], semantic role labeling [Liu+, '07], relation extraction [Sumida+ '08]

ex. | dependency parsing

$$y = \begin{cases} +1 & (\mathrm{dependent}) \\ -1 & (\mathrm{independent}) \end{cases}$$

I RRP



with IN ...

Linear model
[LLM, Perceptron, etc.]

$$y = \mathrm{sgn}(\underline{\boldsymbol{w}}^\top \phi_d(\boldsymbol{x}))$$

high-dimensional weight vector

$\boldsymbol{x} = \underline{\langle f_1, f_2, f_3, f_4 \rangle}$ active (primitive) features

$$\phi_2(\boldsymbol{x}) = \langle f_1, f_2, f_3, f_4, \underbrace{f_{1\wedge 2}, f_{1\wedge 3}, \ldots, f_{3\wedge 4}}_{\text{conjunctive features}} \rangle$$

$$f_{i\wedge j} \neq 0 \quad \mathrm{iff} \quad f_i \neq 0 \cup f_j \neq 0$$

Conjunctive features in NLP

- Conjunctive features play a key role to obtain a high degree of accuracy in NLP classification problems
  - dependency parsing [Koo+ '08], pronoun resolution [Nguyen+, 08], semantic role labeling [Liu+, '07], relation extraction [Sumida+ '08]

Linear model
[LLM, Perceptron, etc.]

$$\boxed{y = \mathrm{sgn}(\underline{\boldsymbol{w}}^\top \phi_d(\boldsymbol{x}))}$$

high-dimensional
weight vector

Kernel-based model
[SVM, Kernel perceptron, etc.]

$$\boxed{y = \mathrm{sgn}\Big(\sum_{\boldsymbol{s}_i \in \underline{\mathcal{S}}} \alpha_i k_d(\boldsymbol{s}_i, \boldsymbol{x})\Big)}$$

support set
(subset of training examples)

polynomial kernel

$$\begin{aligned} & k_d(\boldsymbol{s}, \boldsymbol{x}) \\ & = \phi_d(\boldsymbol{s})^\top \phi_d(\boldsymbol{x}) \\ & = (\boldsymbol{s}^\top \boldsymbol{x} + 1)^d \end{aligned}$$

Space-time tradeoff in training with conjunctive features

- Training with conjunctive features involves space-time tradeoff in the way conjunctive features are handled

linear training (perceptron)		kernel-based training (kernel perceptron)
$\boldsymbol{w} \leftarrow \mathbf{0}$	initialize	$\mathcal{S}_0 \leftarrow \emptyset, \boldsymbol{\alpha} \leftarrow \mathbf{0}$
for $t = 1$ to T do		for $i = 1$ to T do
$\quad m_t \leftarrow \boldsymbol{w}^{\mathrm{T}} \phi_d(\boldsymbol{x}_t)$	compute margin	$\quad m_t \leftarrow \sum_{\boldsymbol{s}_i \in \mathcal{S}_{t-1}} \alpha_i k_d(\boldsymbol{s}_i, \boldsymbol{x}_t)$
$\quad$ if $y_t m_t < 0$		$\quad$ if $y_t m_t < 0$
$\quad\quad \boldsymbol{w} \leftarrow \boldsymbol{w} + y_t \phi_d(\boldsymbol{x}_t)$	update	$\quad\quad \alpha_t \leftarrow y_t, \mathcal{S}_t = \mathcal{S}_{t-1} \cup \{\boldsymbol{x}_t\}$
$\quad$ endif		$\quad$ endif
end		end

needs huge memory

linearly increase as training proceeds

Linear training: polynomial space in the number of primitive features

Kernel-based training: quadratic time in the number of examples

Kernel splitting [Goldberg+ 2008] (for testing)

- Split features into common ones $\mathcal{F}_C$ and rare ones $\mathcal{F} \setminus \mathcal{F}_C$ and divide margin computation: *according to frequency in* $\mathcal{S}$
  - explicitly consider conjunctions among common features
  - implicitly consider remaining conjunctions by kernel

$$
\begin{aligned}
\sum_{\boldsymbol{s}_i \in \mathcal{S}} \alpha_i k_d(\boldsymbol{s}_i, \boldsymbol{x}) &= \sum_{\boldsymbol{s}_i \in \mathcal{S}} \alpha_i k_d(\boldsymbol{s}_i, \underbrace{\boldsymbol{x}_C}_{\boldsymbol{x} \cap \mathcal{F}_C}) + \sum_{\boldsymbol{s}_i \in \mathcal{S}} \alpha_i \{k_d(\boldsymbol{s}_i, \boldsymbol{x}) - k_d(\boldsymbol{s}_i, \boldsymbol{x}_C)\} \\
&= \boldsymbol{w}_C^T \phi_d(\boldsymbol{x}_C) + \sum_{\boldsymbol{s}_i \in \mathcal{S}_R} \alpha_i \{k_d(\boldsymbol{s}_i, \boldsymbol{x}) - k_d(\boldsymbol{s}_i, \boldsymbol{x}_C)\}
\end{aligned}
$$

Kernel splitting [Goldberg+ 2008] (for testing)

- Split features into common ones $\mathcal{F}_C$ and rare ones $\mathcal{F} \setminus \mathcal{F}_C$ and divide margin computation: *according to frequency in $\mathcal{S}$*
  - explicitly consider conjunctions among common features
  - implicitly consider remaining conjunctions by kernel

$$\sum_{s_i \in \mathcal{S}} \alpha_i k_d(s_i, x) = \sum_{s_i \in \mathcal{S}} \alpha_i k_d(s_i, \underbrace{x_C}_{x \cap \mathcal{F}_C}) + \sum_{s_i \in \mathcal{S}} \alpha_i \{k_d(s_i, x) - k_d(s_i, x_C)\}$$

$$= w_C^\top \phi_d(x_C) + \sum_{s_i \in \mathcal{S}_R} \alpha_i \{k_d(s_i, x) - k_d(s_i, x_C)\}$$

explicit weights for common conjunctive features $|w_C| \ll |w|$

$$w_C = \sum_{s_j \in \mathcal{S}} \alpha_i \phi_d(s_i \cap \mathcal{F}_C)$$

space-efficient linear classification

Kernel splitting [Goldberg+ 2008] (for testing)

- Split features into common ones $\mathcal{F}_C$ and rare ones $\mathcal{F} \setminus \mathcal{F}_C$ and divide margin computation: *according to frequency in* $\mathcal{S}$
  - explicitly consider conjunctions among common features
  - implicitly consider remaining conjunctions by kernel

$$\sum_{\boldsymbol{s}_i \in \mathcal{S}} \alpha_i k_d(\boldsymbol{s}_i, \boldsymbol{x}) = \sum_{\boldsymbol{s}_i \in \mathcal{S}} \alpha_i k_d(\boldsymbol{s}_i, \underbrace{\boldsymbol{x}_C}_{\boldsymbol{x} \cap \mathcal{F}_C}) + \sum_{\boldsymbol{s}_i \in \mathcal{S}} \alpha_i \{\underbrace{k_d(\boldsymbol{s}_i, \boldsymbol{x}) - k_d(\boldsymbol{s}_i, \boldsymbol{x}_C)}_{= 0 \text{ when } \boldsymbol{s}_j^{\mathrm{T}} \boldsymbol{x} = \boldsymbol{s}_j^{\mathrm{T}} \boldsymbol{x}_C}\}$$

$$= \boldsymbol{w}_C^{\mathrm{T}} \phi_d(\boldsymbol{x}_C) + \boxed{\sum_{\boldsymbol{s}_i \in \mathcal{S}_R} \alpha_i \{k_d(\boldsymbol{s}_i, \boldsymbol{x}) - k_d(\boldsymbol{s}_i, \boldsymbol{x}_C)\}}$$

consider *a few* support vectors $|\mathcal{S}_R| \ll |\mathcal{S}|$ that have rare feature $f_R \in \boldsymbol{x}_R = \boldsymbol{x} \setminus \boldsymbol{x}_C$

space-efficient linear classification + efficient kernel-based testing

Online learning with kernel splitting

- Replace margin computation part in kernel-based online learning with kernel splitting

Kernel perceptron

$\mathcal{S}_0 \leftarrow \emptyset, \boldsymbol{\alpha} \leftarrow \mathbf{0},$

for $t = 1$ to T do

$$\boxed{m_t \leftarrow \sum_{\boldsymbol{s}_i \in \mathcal{S}_{t-1}} \alpha_i k_d(\boldsymbol{s}_i, \boldsymbol{x}_t)}$$

if $y_t m_t \leq 0$

$\alpha_t \leftarrow y_t, \mathcal{S}_t = \mathcal{S}_{t-1} \cup \{\boldsymbol{x}_t\}$

endif

end

replace this with kernel splitting

Online learning with kernel splitting

- Replace margin computation part in kernel-based online learning with kernel splitting

Kernel perceptron

kernel splitting

$\mathcal{S}_0 \leftarrow \emptyset, \boldsymbol{\alpha} \leftarrow \mathbf{0},$

for $t = 1$ to T do

$\quad \boldsymbol{x}_C \leftarrow \boldsymbol{x}_t \cap \mathcal{F}_C$ Q. 1: how to determine $\mathcal{F}_C$?

$\quad m_t \leftarrow \boldsymbol{w}_C \phi_d(\boldsymbol{x}_C)$ Q. 2: how to maintain $\boldsymbol{w}_C$?

$\qquad\qquad + \sum_{\boldsymbol{s}_i \in \mathcal{S}_R} \alpha_i \{k_d(\boldsymbol{s}_i, \boldsymbol{x}_t) - k_d(\boldsymbol{s}_i, \boldsymbol{x}_C)\}$

$\quad$ if $y_t m_t \leq 0$

$\qquad \alpha_t \leftarrow y_t, \mathcal{S}_t = \mathcal{S}_{t-1} \cup \{\boldsymbol{x}_t\}$

$\quad$ endif

end

Online learning with kernel splitting

- Replace margin computation part in kernel-based online learning with kernel splitting

Kernel perceptron with kernel splitting

A. 1: Choose top-N frequent features in the training examples as $\mathcal{F}_C$

kernel splitting

A.2 Online-update $\boldsymbol{w}_C$ to correspond with $\langle \mathcal{S}_t, \boldsymbol{\alpha}_t \rangle$

$\mathcal{S}_0 \leftarrow \emptyset, \boldsymbol{\alpha} \leftarrow \mathbf{0},$

$\mathcal{F}_C \leftarrow \{f \mid \text{RANK}(f) \leq N\}, \boldsymbol{w}_C \leftarrow \mathbf{0}$

for $t = 1$ to T do

$\quad \boldsymbol{x}_C \leftarrow \boldsymbol{x}_t \cap \mathcal{F}_C$ Q. 1: how to determine $\mathcal{F}_C$?

$\quad m_t \leftarrow \boldsymbol{w}_C \phi_d(\boldsymbol{x}_C)$ Q. 2: how to maintain $\boldsymbol{w}_C$?

$\qquad + \sum_{\boldsymbol{s}_i \in \mathcal{S}_R} \alpha_i \{k_d(\boldsymbol{s}_i, \boldsymbol{x}_t) - k_d(\boldsymbol{s}_i, \boldsymbol{x}_C)\}$

$\quad$ if $y_t m_t \leq 0$

$\qquad \alpha_t \leftarrow y_t, \mathcal{S}_t = \mathcal{S}_{t-1} \cup \{\boldsymbol{x}_t\}$

$\qquad \boldsymbol{w}_C \leftarrow \boldsymbol{w}_C + y_t \phi_d(\boldsymbol{x}_C)$

$\quad$ endif

end

Online learning with kernel splitting

- Replace margin computation part in kernel-based online learning with kernel splitting

Kernel perceptron with kernel splitting

$$
\begin{array}{l}
\mathcal{S}_0 \leftarrow \emptyset, \boldsymbol{\alpha} \leftarrow \mathbf{0}, \\
\mathcal{F}_C \leftarrow \{f \mid \text{RANK}(f) \leq N\}, \boldsymbol{w}_C \leftarrow \mathbf{0} \\
\text{for } t = 1 \text{ to } T \text{ do} \\
\quad \boldsymbol{x}_C \leftarrow \boldsymbol{x}_t \cap \mathcal{F}_C \\
\quad m_t \leftarrow \boldsymbol{w}_C \phi_d(\boldsymbol{x}_C) \\
\qquad\qquad + \sum_{\boldsymbol{s}_i \in \mathcal{S}_R} \alpha_i \{k_d(\boldsymbol{s}_i, \boldsymbol{x}_t) - k_d(\boldsymbol{s}_i, \boldsymbol{x}_C)\} \\
\quad \text{if } y_t m_t \leq 0 \\
\qquad \alpha_t \leftarrow y_t, \mathcal{S}_t = \mathcal{S}_{t-1} \cup \{\boldsymbol{x}_t\} \\
\qquad \boldsymbol{w}_C \leftarrow \boldsymbol{w}_C + y_t \phi_d(\boldsymbol{x}_C) \\
\quad \text{endif} \\
\text{end}
\end{array}
$$

Assumption:
additive updates

$\forall t' > t$

$\langle \boldsymbol{\alpha}_t, \mathcal{S}_t \rangle \subseteq \langle \boldsymbol{\alpha}_{t'}, \mathcal{S}_{t'} \rangle$

Intricacy in setting Parameter N

- Kernel splitting can *control* space-time trade-off in training with conjunctive features, but it *does not resolve* it

$$\boldsymbol{x}_{\mathrm{C}} \leftarrow \boldsymbol{x}_{\mathrm{t}} \cap \mathcal{F}_{\mathrm{C}}$$
$$m_{\mathrm{t}} \leftarrow \boldsymbol{w}_{\mathrm{C}} \phi_{\mathrm{d}}(\boldsymbol{x}_{\mathrm{C}}) + \sum_{\boldsymbol{s}_i \in \mathcal{S}_{\mathrm{R}}} \alpha_i \{ k_{\mathrm{d}}(\boldsymbol{s}_i, \boldsymbol{x}_{\mathrm{t}}) - k_{\mathrm{d}}(\boldsymbol{s}_i, \boldsymbol{x}_{\mathrm{C}}) \}$$

Time complexity : $\mathcal{O}(|\boldsymbol{x}_{\mathrm{C}}|^{\mathrm{d}} + |\mathcal{S}_{\mathrm{R}}||\boldsymbol{x}_{\mathrm{C}}|)$

  - $\mathrm{N}\ (= |\mathcal{F}_{\mathrm{C}}|)$ should be smaller for higher-order conjunctive features (to keep $|\boldsymbol{x}_{\mathrm{C}}|^{\mathrm{d}}$ and $|\mathcal{F}_{\mathrm{C}}|^{\mathrm{d}}$ small)
  - $\mathrm{N}\ (= |\mathcal{F}_{\mathrm{C}}|)$ should be larger when we handle a larger number training examples (to keep $|\mathcal{S}_{\mathrm{R}}|$ small)

Kernel slicing | basic idea

- Examples in real-world data are redundant [Yoshinaga+ '09]
    - Online learner will repeatedly compute margins of common partial feature vectors

kernel perceptron

$$
\begin{array}{l}
\mathcal{S}_0 \leftarrow \emptyset, \boldsymbol{\alpha} \leftarrow \mathbf{0} \\
\text{for } i = 1 \text{ to } T \text{ do} \\
\quad m_t \leftarrow \sum_{\boldsymbol{s}_i \in \mathcal{S}_{t-1}} \alpha_i k_d(\boldsymbol{s}_i, \boldsymbol{x}_t) \\
\quad \text{if } y_t m_t < 0 \\
\qquad \alpha_t \leftarrow y_t, \mathcal{S}_t = \mathcal{S}_{t-1} \cup \{\boldsymbol{x}_t\} \\
\quad \text{endif} \\
\text{end}
\end{array}
$$

i	*sequence of examples*	*parameters*
	$\cdots$ features sorted	
$t-2$	$\boldsymbol{x}_{t-2} = \langle f_1, f_3, f_7, f_{19} \rangle$	$\langle \mathcal{S}_{t-2}, \boldsymbol{\alpha}_{t-2} \rangle$
$t-1$	$\boldsymbol{x}_{t-1} = \langle f_1, f_3, f_9, f_{21} \rangle$	$\langle \mathcal{S}_{t-1}, \boldsymbol{\alpha}_{t-1} \rangle$
t	$\boldsymbol{x}_t = \langle \boxed{f_1, f_3}, \boxed{f_7}, \boxed{f_{25}} \rangle$	$\langle \mathcal{S}_t, \boldsymbol{\alpha}_t \rangle$
	$\cdots$	

$$
m_t = \underbrace{\square}_{\{\boldsymbol{x}_{t-1}\}} + \underbrace{\boxed{\square - \square}}_{\{\boldsymbol{x}_{t-2}\}} + \underbrace{\boxed{m_t - \square}}_{\{\boldsymbol{s} \in \mathcal{S}_t | f_{25} \in \boldsymbol{s}\}}
$$

reusing partial margins reduces support vectors to be considered

Kernel slicing | feature-wise splitting

- Kernel slicing: incrementally compute a partial margin of $\boldsymbol{x}_t$ when adding features to $\boldsymbol{x}_t^0\ (=\emptyset)$ from frequent to rare

$$m_t = m_t^0 + \sum_{j=1}^{|\boldsymbol{x}_t|} \underline{m_t^j}$$

margin change when we add j-th frequent feature

temporal partial margin $m_t^j = \sum_{\boldsymbol{s}_i \in \mathcal{S}_t} \alpha_i (k_d(\boldsymbol{s}, \boldsymbol{x}_t^j) - k_d(\boldsymbol{s}, \boldsymbol{x}_t^{j-1}))$

- retrieve / update partial margins (with time index t) in a trie

$$m_t^j = \underline{m_{t'}^j} + \underline{\sum_{\boldsymbol{s}_i \in \mathcal{S}_j}} \alpha_i \{k_d(\boldsymbol{s}_i, \boldsymbol{x}_j) - k_d(\boldsymbol{s}_i, \boldsymbol{x}_{j-1})\}$$

partial margin computed for $\boldsymbol{x}_t^j$ at past round $t'(< t)$

$\mathcal{S}_j = \{\boldsymbol{s} \in \mathcal{S}_t \setminus \mathcal{S}_{t'}\ |f_j \in \boldsymbol{s}\}$

newly added support vectors since we finally see $\boldsymbol{x}_t^j$

- when common feature $f_j \in \mathcal{F}_C$ is added and the retrieved margin was <u>too old</u>, use $\boldsymbol{w}_C$ to compute the partial margin

$|\phi_d(\boldsymbol{x}_j) - \phi_d(\boldsymbol{x}_{j-1})| < |\mathcal{S}_j||\boldsymbol{x}_{j-1}|$ $\qquad m_t^j = \boldsymbol{w}_C^T\{\phi_d(\boldsymbol{x}_t^j) - \phi(\boldsymbol{x}_t^{j-1})\}$

Experiments

- Implement online passive aggressive I (PA-I) [Crammer+ '06] with kernel slicing
- Compare our learner with
  - Support vector machine (SVM) [TinySVM by T. Kudo]
  - kernel-based PA-I with inverted indices [Okanohara+ '07]
  - SGD-training of ℓ_1- reguralized log-linear model [Tsuruoka, '09]
- Evaluate on two NLP tasks:
  Japanese dependency parsing and hyponymy relation extraction

Task settings

- Japanese dependency parsing
    - Classifier judges whether a given head/dependent candidate has a dependency relation (in shift-reduce parser [Sassano, '04])
    - Features: POS(-subcategory), inflection form of head / dependent, and surrounding contexts (distance etc.)
- Hyponymy relation extraction
    - Classifier judges whether a given pair of entities extracted from Wikipedia articles forms a hyponymy relation [Sumida+ '08]
    - Features: POS, surface string, morpheme, listing type of each entity, and surrounding contexts (distance etc.)

We considered third-order conjunctive features in training

Example / Feature Statistics

- Feature conjunctions dramatically increase
    - the average number of active features
    - the feature space

DATA SET		dependency parsing	hyponymy extraction	
T (# examples)		296,776	201,664	
Ave. of $\lvert\boldsymbol{x}\rvert$	x130	27.6	15.4	x50
Ave. of $\lvert\phi_3(\boldsymbol{x})\rvert$		3558.3	798.7	
$\lvert\mathcal{F}\rvert$ (# features)	x900	64,493	306,036	x210
$\lvert\mathcal{F}^3\rvert$ (# conj. features)		58,361,669	64,249,234	

Labeled examples are available from: http://www.tkl.iis.u-tokyo.ac.jp/~ynaga/pecco/
http://nlpwww.nict.go.jp/hyponymy/

Results | dependency parsing

- PA-I with kernel slicing was the fastest, while retaining space-efficiency of kernel-based training
  - hyper-parameters are tuned to maximize model accuracy on development set

	METHOD	ACC.	TIME	MEMORY
kernel-based training	SVM (batch)	90.93%	25912s	243MB
kernel-based training	PA-I | kernel	90.90%	8704s	83MB
	PA-I | splitting	90.90%	351s	149MB
	PA-I | slicing	90.89%	262s	175MB
linear training	PA-I | linear	90.90%	465s	993MB
linear training	ℓ_1-LLM (SGD)	90.76%	4057s	21499MB

x30

1/6

Results | hyponymy extraction

- PA-I with kernel slicing was the fastest, while retaining space-efficiency of kernel-based training
  - hyper-parameters are tuned to maximize model accuracy on development set

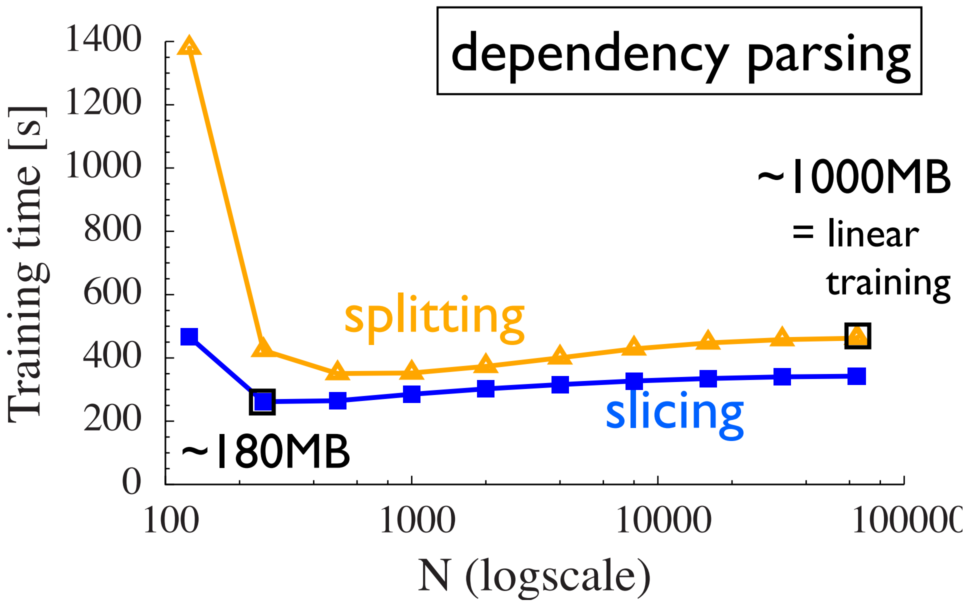
	METHOD	ACC.	TIME	MEMORY
kernel-based training	SVM (batch)	93.09%	17354s	140MB
kernel-based training	PA-I | kernel	93.14%	1074s	49MB
	PA-I | splitting	93.10%	68s	108MB
	PA-I | slicing	93.05%	17s	131MB
linear training	PA-I | linear	93.11%	103s	751MB
linear training	ℓ_1-LLM (SGD)	92.86%	779s	14089MB

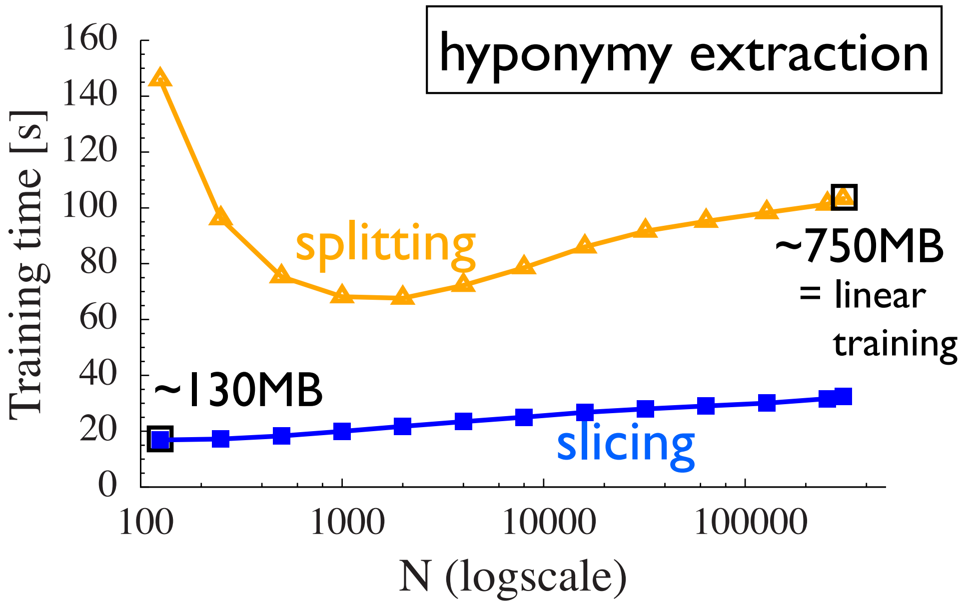
x70

1/6

Splitting vs. Slicing | Parameter N

- Training time as a function of parameter N [mem. usage]
  - kernel splitting: parameter-sensitive
  - kernel slicing: parameter-insensitive

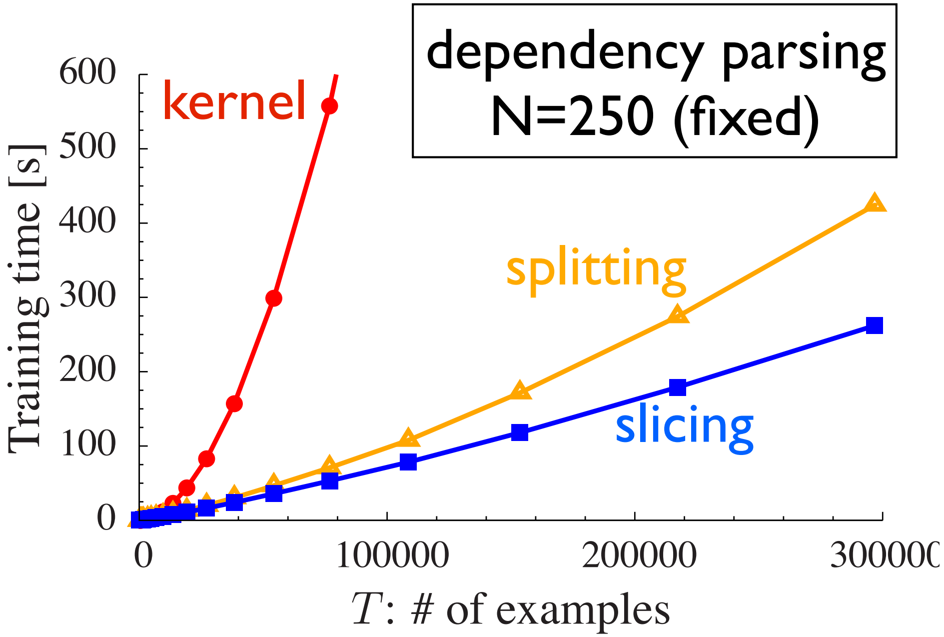


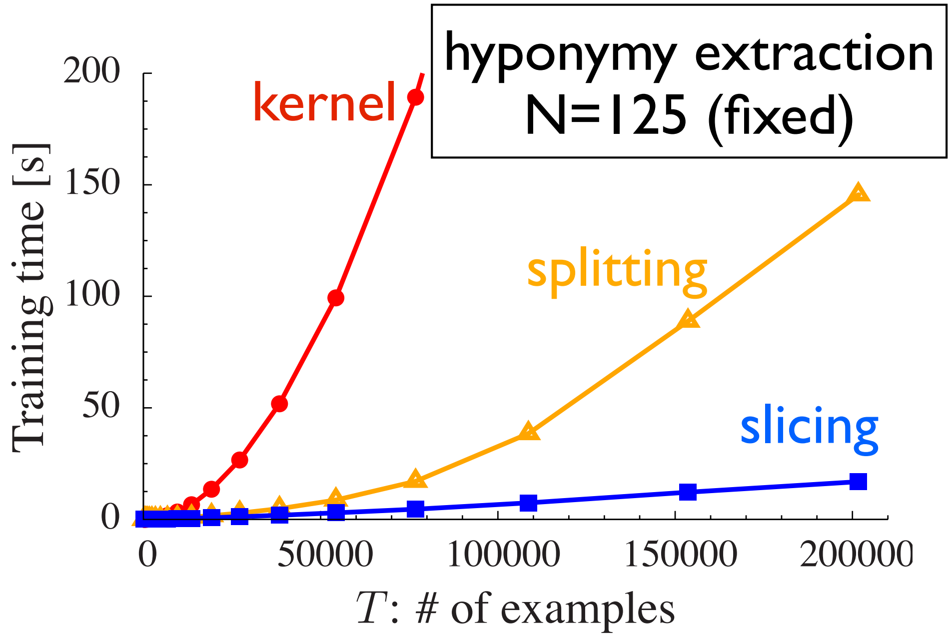


You don't need to *tune* N (it doesn't change the model, anyway)

Splitting vs. Slicing | # examples, T

- Training time as a function of the number of examples
  - kernel splitting: in between linear and quadratic
  - kernel slicing: almost linear





Reusing temporal margins → more scalable training

Related Work

- Feature selection in linear training [Wu+ '07, Okanohara+, '09] (limit the number of conjunctive features)
  - Simpler model → faster, more space-efficient, *less accurate*
    x17 but 94.19%→93.71% (named entity recognition [Wu+ '07]),
    x37 but 89.52%→89.03% (dependency parsing [Okanohara+ '09])
- Bounded Kernel-based training [Dekel+ '06; Cavallanti+ '07] (limit the number of support vectors)
  - These lightweight algorithms could not bound the number of support vectors, while retaining model accuracy [Orabona+ '09]

Our method exploits the data redundancy in evaluating the kernel to train the same model as the base learner

Conclusion

- Scalable online training method with kernel slicing
  - Kernel slicing generalizes kernel splitting [Goldberg+ '08], to reuse temporal partial margins for common partial feature vectors
  - orders of magnitude faster than kernel-based online training, while retaining its space efficiency
- Things I didn't mention in this talk (see our paper):
  - Efficient management of feature weights and partial margins (packed training examples) with a double-array trie [Yata+ '09]
  - Termination of margin computations that will never contribute to parameter updates (safely skipping rare features)

Future work

- ~~Release C++ implementation and dataset:~~ *done.*
  http://www.tkl.iis.u-tokyo.ac.jp/~ynaga/opal/
  - Fast testing? - you may want to try pecco [Yoshinaga+, EMNLP '09]
    http://www.tkl.iis.u-tokyo.ac.jp/~ynaga/pecco/
- Implement kernel slicing for other online algorithms
- Generalize kernel slicing to accommodate other kernels

Thank you