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

\[ y = \text{sgn}(\mathbf{w}^T \phi_d(\mathbf{x})) \]

Linear model
[LLM, Perceptron, etc.]

\[ y = \begin{cases} +1 & \text{(dependent)} \\ -1 & \text{(independent)} \end{cases} \]

with

\[ \mathbf{x} = \langle f_1, f_2, f_3, f_4 \rangle \]

active (primitive) features

\[ \phi_2(\mathbf{x}) = \langle f_1, f_2, f_3, f_4, f_1 \land 2, f_1 \land 3, \ldots, f_3 \land 4 \rangle \]

conjunctive features

\[ f_{i \land j} \neq 0 \quad \text{iff} \quad f_i \neq 0 \lor f_j \neq 0 \]
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.]

\[ y = \text{sgn}(\mathbf{w}^T \phi_d(\mathbf{x})) \]

- High-dimensional weight vector

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

\[ y = \text{sgn}\left( \sum_{\mathbf{s}_i \in S} \alpha_i k_d(\mathbf{s}_i, \mathbf{x}) \right) \]

- Polynomial kernel

\[ k_d(\mathbf{s}, \mathbf{x}) = \phi_d(\mathbf{s})^T \phi_d(\mathbf{x}) = (\mathbf{s}^T \mathbf{x} + 1)^d \]
Training with conjunctive features involves space-time tradeoff in the way conjunctive features are handled.

### Linear training (perceptron)

- \( w \leftarrow 0 \)
- For \( t = 1 \) to \( T \) do
  - \( m_t \leftarrow w^T \phi_d(x_t) \)
  - If \( y_t m_t < 0 \)
    - \( w \leftarrow w + y_t \phi_d(x_t) \)
  - Endif
- End

- \( w \) needs huge memory

### Kernel-based training (kernel perceptron)

- Initialize \( \alpha \leftarrow 0 \)
- For \( i = 1 \) to \( T \) do
  - Compute margin \( m_t \leftarrow \sum_{s_i \in S_{t-1}} \alpha_i k_d(s_i, x_t) \)
  - If \( y_t m_t < 0 \)
    - Update \( \alpha_t \leftarrow y_t, S_t = S_{t-1} \cup \{x_t\} \)
  - Endif
- End

- 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 $S$
- Explicitly consider conjunctions among common features
- Implicitly consider remaining conjunctions by kernel

$$\sum_{s_i \in S} \alpha_i k_d(s_i, x) = \sum_{s_i \in S} \alpha_i k_d(s_i, x_C) + \sum_{s_i \in S} \alpha_i \{k_d(s_i, x) - k_d(s_i, x_C)\}$$

$$= w_C^T \phi_d(x_C) + \sum_{s_i \in S_R} \alpha_i \{k_d(s_i, x) - k_d(s_i, x_C)\}$$
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

$$
\sum_{s_i \in \mathcal{S}} \alpha_i k_d(s_i, x) = \sum_{s_i \in \mathcal{S}} \alpha_i k_d(s_i, x_C) + \sum_{s_i \in \mathcal{S}} \alpha_i \{k_d(s_i, x) - k_d(s_i, x_C)\}
$$

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

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

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

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 $S$
- Explicitly consider conjunctions among common features
- Implicitly consider remaining conjunctions by kernel

$$
\sum_{s_i \in S} \alpha_i k_d(s_i, x) = \sum_{s_i \in S} \alpha_i k_d(s_i, x_C) + \sum_{s_i \in S} \alpha_i [k_d(s_i, x) - k_d(s_i, x_C)] \\
= \omega_C^T \phi_d(x_C) + \sum_{s_i \in S_R} \alpha_i [k_d(s_i, x) - k_d(s_i, x_C)]
$$

Consider a few support vectors $|S_R| \ll |S|$ that have rare feature $f_R \in x_R = x \setminus x_C$

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

\[
S_0 \leftarrow \emptyset, \alpha \leftarrow 0,
\]

for \( t = 1 \) to \( T \) do

\[
m_t \leftarrow \sum_{s_i \in S_{t-1}} \alpha_i k_d(s_i, x_t)
\]

if \( y_t m_t \leq 0 \)

\[
\alpha_t \leftarrow y_t, S_t = S_{t-1} \cup \{x_t\}
\]

endif

end
Online learning with kernel splitting

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

Kernel perceptron

\[
S_0 \leftarrow \emptyset, \alpha \leftarrow 0,
\]

for \( t = 1 \) to \( T \) do

\[
x_C \leftarrow x_t \cap \mathcal{F}_C
\]

\[
m_t \leftarrow w_C \phi_d(x_C) + \sum_{s_i \in S_r} \alpha_i [k_d(s_i, x_t) - k_d(s_i, x_C)]
\]

if \( y_t m_t \leq 0 \)

\[
\alpha_t \leftarrow y_t, S_t = S_{t-1} \cup \{x_t\}
\]

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 $\mathcal{F}_C$

A.2 Online-update $w_C$ to correspond with $\langle S_t, \alpha_t \rangle$

$S_0 \leftarrow \emptyset$, $\alpha \leftarrow 0$, 
$\mathcal{F}_C \leftarrow \{f \mid \text{RANK}(f) \leq N\}$, $w_C \leftarrow 0$

for $t = 1$ to $T$ do

$\mathbf{x}_C \leftarrow \mathbf{x}_t \cap \mathcal{F}_C$

$\mathbf{m}_t \leftarrow w_C \phi_d(\mathbf{x}_C)$

$+ \sum_{s_i \in \mathcal{S}_R} \alpha_i [k_d(s_i, \mathbf{x}_t) - k_d(s_i, \mathbf{x}_C)]$

if $y_t \mathbf{m}_t \leq 0$

$\alpha_t \leftarrow y_t$, $S_t = S_{t-1} \cup \{\mathbf{x}_t\}$

$w_C \leftarrow w_C + y_t \phi_d(\mathbf{x}_C)$

endif

end
Online learning with kernel splitting

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

Kernel perceptron with kernel splitting

\[
S_0 \leftarrow \emptyset, \alpha \leftarrow 0, \\
F_C \leftarrow \{f | \text{RANK}(f) \leq N\}, w_C \leftarrow 0 \\
\text{for } t = 1 \text{ to } T \text{ do} \\
x_C \leftarrow x_t \cap F_C \\
m_t \leftarrow w_C \phi_d(x_C) \\
\quad + \sum_{s_i \in S_R} \alpha_i [k_d(s_i, x_t) - k_d(s_i, x_C)] \\
\text{if } y_t m_t \leq 0 \\
\quad \alpha_t \leftarrow y_t, S_t = S_{t-1} \cup \{x_t\} \\
\quad w_C \leftarrow w_C + y_t \phi_d(x_C) \\
\text{endif} \\
\text{end}
\]

Assumption:
additive updates
\[\forall t' > t \quad \langle \alpha_t, S_t \rangle \subseteq \langle \alpha_{t'}, S_{t'} \rangle\]
Intricacy in setting Parameter $N$

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

\[
x_C \leftarrow x_t \cap F_C \\
m_t \leftarrow w_C \phi_d(x_C) + \sum_{s_i \in S_R} \alpha_i[k_d(s_i, x_t) - k_d(s_i, x_C)]
\]

Time complexity: $O(|x_C|^d + |S_R||x_C|)$

- $N (= |F_C|)$ should be smaller for higher-order conjunctive features (to keep $|x_C|^d$ and $|F_C|^d$ small)

- $N (= |F_C|)$ should be larger when we handle a larger number training examples (to keep $|S_R|$ 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

\[
S_0 \leftarrow \emptyset, \alpha \leftarrow \mathbf{0} \\
\text{for } i = 1 \text{ to } T \text{ do} \\
\quad m_t \leftarrow \sum_{s_i \in S_{t-1}} \alpha_i k_d(s_i, x_t) \\
\quad \text{if } y_t m_t < 0 \\
\quad \quad \alpha_t \leftarrow y_t, S_t = S_{t-1} \cup \{x_t\} \\
\quad \text{endif} \\
\text{end}
\]

reusing partial margins reduces support vectors to be considered
Kernel slicing | feature-wise splitting

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

  \[ m_t = m_t^0 + \sum_{j=1}^{\vert x_t \vert} m_t^j \]

  margin change when we add $j$-th frequent feature

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

  \[ m_t^j = \sum_{s_i \in S_t} \alpha_i (k_d(s, x_t^j) - k_d(s, x_t^{j-1})) \]

  temporal partial margin

  \[ m_t^j = \sum_{s_i \in S_t} \alpha_i (k_d(s, x_t^j) - k_d(s, x_t^{j-1})) \]

  partial margin computed for $x_t^j$ at past round $t' (< t)$

- when common feature $f_j \in F_C$ is added and the retrieved margin was too old, use $w_C$ to compute the partial margin

  \[ \vert \phi_d(x_j) - \phi_d(x_j-1) \vert < \vert S_j \vert \vert x_j-1 \vert \]

  \[ m_t^j = w_C^T (\phi_d(x_t^j) - \phi(x_t^{j-1})) \]

  \[ S_j = \{ s \in S_t \setminus S_t', \mid f_j \in s \} \]

  newly added support vectors since we finally see $x_t^j$
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 $\ell_1$-regularized 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

<table>
<thead>
<tr>
<th>DATA SET</th>
<th>dependency parsing</th>
<th>hyponymy extraction</th>
</tr>
</thead>
<tbody>
<tr>
<td>T (# examples)</td>
<td>296,776</td>
<td>201,664</td>
</tr>
<tr>
<td>Ave. of $</td>
<td>x</td>
<td>$</td>
</tr>
<tr>
<td>Ave. of $</td>
<td>\phi_3(x)</td>
<td>$</td>
</tr>
<tr>
<td>$</td>
<td>\mathcal{F}</td>
<td>$ (# features)</td>
</tr>
<tr>
<td>$</td>
<td>\mathcal{F}^3</td>
<td>$(# conj. features)</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>METHOD</th>
<th>ACC.</th>
<th>TIME</th>
<th>MEMORY</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVM (batch)</td>
<td>90.93%</td>
<td>25912s</td>
<td>243MB</td>
</tr>
<tr>
<td>PA-I</td>
<td>kernel</td>
<td>90.90%</td>
<td>8704s</td>
</tr>
<tr>
<td>PA-I</td>
<td>splitting</td>
<td>90.90%</td>
<td>351s</td>
</tr>
<tr>
<td>PA-I</td>
<td>slicing</td>
<td>90.89%</td>
<td>262s</td>
</tr>
<tr>
<td>PA-I</td>
<td>linear</td>
<td>90.90%</td>
<td>465s</td>
</tr>
<tr>
<td>$\ell_1$-LLM (SGD)</td>
<td>90.76%</td>
<td>4057s</td>
<td>21499MB</td>
</tr>
</tbody>
</table>
## 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

<table>
<thead>
<tr>
<th>METHOD</th>
<th>ACC.</th>
<th>TIME</th>
<th>MEMORY</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVM (batch)</td>
<td>93.09%</td>
<td>17354s</td>
<td>140MB</td>
</tr>
<tr>
<td>PA-I</td>
<td>kernel</td>
<td>93.14%</td>
<td>1074s</td>
</tr>
<tr>
<td>PA-I</td>
<td>splitting</td>
<td>93.10%</td>
<td>68s</td>
</tr>
<tr>
<td>PA-I</td>
<td>slicing</td>
<td>93.05%</td>
<td>17s</td>
</tr>
<tr>
<td>PA-I</td>
<td>linear</td>
<td>93.11%</td>
<td>103s</td>
</tr>
<tr>
<td>$\ell_1$-LLM (SGD)</td>
<td>92.86%</td>
<td>779s</td>
<td>14089MB</td>
</tr>
</tbody>
</table>

**Kernel-based training**

**Linear training**
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 $\rightarrow$ faster, more space-efficient, less accurate
    - $x17$ but $94.19\% \rightarrow 93.71\%$ (named entity recognition [Wu+ '07]),
    - $x37$ but $89.52\% \rightarrow 89.03\%$ (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