Greedy Linearization of Tree Kernel Functions

Daniele Pighin
daniele.pighin@gmail.com

University of Trento, Italy

June 2010

NLP and Tree-like Data

Tree-like structures naturally occur in many NLP contexts, e.g.:
- Syntactic/dependency parses,
- Structured documents (e.g. XML),
- Ontologies ...

```html
<html>
<head></head>
<body>
<div>...</div>
</body>
</html>
```
Tree Kernel Functions

- **Natural** candidates to tackle problems involving structured data
- **Implicitly** measure tree similarity as the number of shared fragments

\[ \phi(t_1) = [2, 1, 1, 1, 1, 0, 0] \]
\[ \phi(t_2) = [0, 0, 0, 0, 1, 1, 1] \]
\[ TK(t_1, t_2) = \langle \phi(t_1), \phi(t_2) \rangle = 1 \]

Advantages of Tree Kernels:

- Evaluate pairwise tree similarity in a **very high** dimensional space (\(10^5 \sim 10^{40}\) dimensions, depending on the task)
- **Easy** to exploit complex structured features in rich syntactic spaces
- **SVM + TK** = accurate and flexible classifier for tree-like objects

Many applications in NLP, e.g.:

- Parse Reranking [Collins and Duffy, 2001, Shen et al., 2003]
- Relation Extraction [Zhang et al., 2006, Nguyễn et al., 2009]
- Question Classification [Zhang and Lee, 2003, Moschitti et al., 2007]
- Semantic Role Labeling [Diab et al., 2008, Moschitti et al., 2008]
- Recognizing Textual Entailment [Zanzotto and Moschitti, 2006]
- **State-of-the-art accuracy** in RE, QC, SRL-Arabic, RTE
Drawbacks of Tree Kernels

And, more generally, of kernel methods

Complexity issues:

- **Time complexity**: learning and classification take place in the dual space
  
  Problem when:
  
  - Training with very large amounts of data
  - Fast classification time required (i.e. on-line applications)

- **Space complexity**: implicit TK algorithms use dynamic programming
  
  Problem when:
  
  - Evaluating the kernel product for very large structures

Interpretability issues:

- The models we learn are implicit
- No insights on the studied problems

Proposed Solution

A feature selection framework for TK functions and SVMs

Innovative aspects (CoNLL’09, EMNLP’09, CoNLL’10)

- Insights about the inner working of TK functions
- **Theoretical framework** for feature selection in convolution kernel spaces
  
  based on statistical learning theory and support vector learning principles
- **Greedy gradient-based strategy** to mine the TK feature space
- **Explicit representation** of relevant fragments
- Efficient data structure and algorithms for fragment indexing
- **Accurate** classifiers with efficient training/testing cycles
Design Principles

Main elements of our feature selection strategy

- Learn an SVM model in the TK space
- The model describes the separating hyperplane’s gradient
- Use gradient components to assess fragment relevance
- Explicitly store relevant fragments in a dictionary
- Decode (linearize) input data by looking up fragments in the dictionary
  (trees $\rightarrow$ vectors)
- Use decoded data for learning/classification in the lower dimensional space

To explore the huge fragment space, a greedy strategy is necessary
Introduction

Mining the Fragment Space

Evaluation

Closing Remarks

Greedy Mining Algorithm

The approach:
- **Small to large** fragment generation
- Combines two **basic operations**:
  - \textsc{frag}(n) generates the smallest (base) fragment rooted in \textit{n} (prod. rule)
  - \textsc{expand}(f) generates expansions of a fragment \textit{f} by including the productions of some of its nodes

The Algorithm:
- Set the parameter \(\sigma\) as the **minimum** acceptable fragment weight
- Generate all the **base fragments** encoded in the model
- Recursively expand fragments with relevance greater than \(\sigma\)
- Discard fragments with frequency < \(k\) (generally, \(k = 3\))

Overview

Focus on the **Syntactic Tree Kernel** (STK) [Collins and Duffy, 2001]:
- Production rules cannot be broken
- The smallest fragments are production rules
- Very suitable for constituency parsed data

Evaluation objectives:
- Evaluate the **efficiency** of process and the algorithms
- Compare the **accuracy** after feature selection against a non-projected STK
Data sets

3 multiclass tasks (one-vs-all binary classifiers):

- **Question Classification**, QC (~ 5k training points)
  TREC 10 data [Voorhees, 2001], coarse grained, 6 classes
  Assign the most appropriate category to a question
  (e.g. Entity, Location, Description, ...)

- **Relation Extraction**, RE (~ 34k training points)
  ACE 2004 English Corpus data [Doddington et al., 2004], 7 classes
  Label relations between constituents in which entities are mentioned
  (e.g. Person-Affiliation, Agent-Artifact, ...)

- **Semantic Role Labeling**, SRL (~ 170k training points)
  CoNLL 2005 data [Carreras and Màrquez, 2005], core roles only, 6 classes
  Assign the proper role label to a constituent identified as argument of a
  predicate (e.g. A0 (subject), A1 (object), ...)

Different Classes, Different Fragments

**QC/Description**

- SBARQ: What is color.
- SBARQ: What is kind.
- SBARQ: What is film.
- SBARQ: WHNP is beer.
- SBARQ: program.

**QC/Entity**

- WHNP: VHS.
- WHNP: NP.
Fragments for Feature Discovery

- Most relevant fragments for English-SRL encode features manually designed by computational linguists in the last decade, e.g.:
  - Path / Partial Path
  - Governing Category
  - Subcategorization Frame
  - Phrase Type / Predicate Phrase Type

- In this context, TK’s feature discovery capabilities are not fully exploited
- If applied to new problems (e.g. Arabic-SRL), TK linearization can give us important clues about relevant linguistic features

Mining and Decoding Performance

- Mining time sub-linear with mined fragments
- Decoding time sub-linear with mined fragments; linear with size of tree
- Exploits properties of the dictionary; would be exponential otherwise
Architectural Components

Combined into **different architectures** to achieve different results

**KSL** Kernel Space Learning
A model is learned in the TK space

**KSM** Kernel Space Mining
The most relevant fragments are mined from the SVs and indexed in a dictionary

**LSG** Linear Space Generation
Input data are decoded (trees $\rightarrow$ vectors)

**LSL** Linear Space Learning
A model is learned in the explicit space

**LSC** Linear Space Classification
The linear model is used to classify linear data

---

**Linear Space Optimization (LOpt)**

- **Rationale:** Decode training data and learn a classifier in the linear space
- **Useful for:** Accuracy, fast classification
Multiclass Accuracy: LOpt vs. STK

### Classification accuracy

<table>
<thead>
<tr>
<th>QC</th>
<th>ABBR</th>
<th>DESC</th>
<th>ENTRY</th>
<th>HUM</th>
<th>LOC</th>
<th>NUM</th>
<th>MULTI</th>
</tr>
</thead>
<tbody>
<tr>
<td>STK</td>
<td>80.00</td>
<td>86.26</td>
<td>76.86</td>
<td>84.92</td>
<td>81.69</td>
<td>92.31</td>
<td>83.71</td>
</tr>
<tr>
<td>LOpt</td>
<td>89.66</td>
<td>87.50</td>
<td>75.56</td>
<td>84.53</td>
<td>92.08</td>
<td>94.60</td>
<td>83.92</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RE</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>MULTI</th>
</tr>
</thead>
<tbody>
<tr>
<td>STK</td>
<td>51.61</td>
<td>81.66</td>
<td>79.75</td>
<td>76.19</td>
<td>16.00</td>
<td>57.69</td>
<td>68.69</td>
<td>68.01</td>
</tr>
<tr>
<td>LOpt</td>
<td>63.96</td>
<td>67.33</td>
<td>74.14</td>
<td>63.33</td>
<td>58.82</td>
<td>62.83</td>
<td>61.70</td>
<td>67.08</td>
</tr>
</tbody>
</table>

For SRL, features before: $\sim 10^{25}$  Features after: $\sim 10^4$!

LOpt Classification Efficiency

Process efficiency

- Test time for LOpt: LSG + LSC
- LSC is negligible, the bottleneck is decoding time (LSG)
- The larger the dataset, the better the improvement
- For optimal values of $\sigma$, LOpt binary classifiers are approximately 3 (QC, RE) and 4 (SRL) times faster than STK
- For a multiclass problem with $C$ classes, each example is only decoded once: $\Rightarrow$ test efficiency is boosted by a factor $C$
- Native implementation could significantly speed up LSG (currently: Python3)
Analysis of LOpt training time

- **Training time:**
  \[ \text{KSL} + \text{KSM} + \text{LSG} \text{ (train)} + \text{LSL} \]
- KSL and LSL are the main players
- We want global linear optimization
  \[ \Rightarrow \text{cannot reduce LSL} \]
  (Though we could use faster optimizers, e.g. LinearSVM)
- We can reduce KSL by splitting training data (Cascades of SVMs, [Graf et al., 2004])

### Split KSL - LOpt (Split)

- **Rationale:** Split training data to make KSL faster
- **Useful for:** Faster learning cycles

\[ \langle y_i, t_i \rangle_{\text{TR}} \xrightarrow{S_5, (y_i, t_i)} \text{KSL} \xrightarrow{M_5, (y_i, t_i)} \text{KSM} \]
\[ \langle y_i, x_i \rangle_{\text{TR}} \xrightarrow{S_3, (y_i, x_i)} \text{KSL} \xrightarrow{M_3, (y_i, x_i)} \text{KSM} \]
\[ \langle y_i, t_i \rangle_{\text{TE}} \]

\[ \langle y_i, x_i \rangle_{\text{TE}} \]

Dictionary → LSG

\[ \langle y_i, x_i \rangle_{\text{TE}} \]

Predictions
For more than $10^4$ fragments, Split accuracy is at least 97% of STK.

- Split5 is as accurate as STK for more than $10^4$ fragments.
- Split5 training is twice as fast as STK.
- With 5 CPUs, Split5 is up to 3 times as fast.
- The bottleneck for parallelization is LSL.

[Pighin and Moschitti, 2009a]: 10x speed up on binary classifier trained on 1M instances with Split5 and 5 CPUs.
- Study relevant fragments for new/unexplored tasks (e.g. SRL-Arabic)
- Use of fast linearized classifiers and relevant fragments as building blocks for larger software architectures, e.g. SMT or automatic pattern discovery
- An analytic solution to find the optimal $\sigma$ for KSM
- Experiments with other kernel families (PTK already supported)
- Experiments in other fields (e.g. bio-informatics, networking)
- From fragments to fragment classes: mining prototypes in the rich TK space
References


References (3)