Greedy Linearization of Tree Kernel Functions

Daniele Pighin
daniele.pighin@gmail.com

University of Trento, Italy

June 2010
Tree-like structures naturally occur in many NLP contexts, e.g.:
- Syntactic/dependency parses,
- Structured documents (e.g. XML),
- Ontologies ...

<html>
<head></head>
<body>
<div>...</div>
<div>...</div>
<div>...</div>
<table>...</table>
</body>
</html>
• **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, Nguyen 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
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
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
Talk Outline

- Mining the Fragment Space
  - Design Principles
  - Greedy Mining Algorithm

- Evaluation
  - Overview
  - Relevant Fragments
  - Algorithmic Efficiency
  - Accuracy of Linearized Classifiers
  - Process Efficiency

- Closing Remarks
  - Conclusions and Research Directions
Outline

- **Mining the Fragment Space**
  - Design Principles
    - Greedy Mining Algorithm
  
- **Evaluation**
  - Overview
  - Relevant Fragments
  - Algorithmic Efficiency
  - Accuracy of Linearized Classifiers
  - Process Efficiency

- **Closing Remarks**
  - Conclusions and Research Directions
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 → vectors)
- Use decoded data for learning/classification in the lower dimensional space

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

- **Mining the Fragment Space**
  - Design Principles
  - Greedy Mining Algorithm

- **Evaluation**
  - Overview
  - Relevant Fragments
  - Algorithmic Efficiency
  - Accuracy of Linearized Classifiers
  - Process Efficiency

- **Closing Remarks**
  - Conclusions and Research Directions
Greedy Mining Algorithm

The approach:

- **Small to large** fragment generation
- Combines two **basic operations**:
  
  \[
  \text{FRAG}(n) \quad \text{generates the smallest (base) fragment rooted in } n \text{ (prod. rule)}
  \]
  
  \[
  \text{EXPAND}(f) \quad \text{generates expansions of a fragment } f \text{ by including the productions of some of its nodes}
  \]
Greedy Mining Algorithm
Greedy Mining Algorithm
Greedy Mining Algorithm
Greedy Mining Algorithm
The approach:

- **Small to large** fragment generation
- Combines two **basic operations**:
  
  - $\text{FRAG}(n)$ generates the smallest (base) fragment rooted in $n$ (prod. rule)
  
  - $\text{EXPAND}(f)$ generates expansions of a fragment $f$ 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, $k = 3$)
Greedy Mining Algorithm

relevance < σ
Outline

- Mining the Fragment Space
  - Design Principles
  - Greedy Mining Algorithm

- Evaluation
  - Overview
  - Relevant Fragments
  - Algorithmic Efficiency
  - Accuracy of Linearized Classifiers
  - Process Efficiency

- Closing Remarks
  - Conclusions and Research Directions
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
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), …)
Outline

- Mining the Fragment Space
  - Design Principles
  - Greedy Mining Algorithm

- Evaluation
  - Overview
  - Relevant Fragments
    - Algorithmic Efficiency
    - Accuracy of Linearized Classifiers
    - Process Efficiency

- Closing Remarks
  - Conclusions and Research Directions
Different Classes, Different Fragments

QC/Description

QBARQ
WHNP
SQ
.
WP
VBZ
NP
What
is

QBARQ
WHADVP
SQ
.
WP
VBZ
NP
Why

QBARQ
WHNP
SQ
.
WP
VBZ
NP
ADJP
What
is
How

QBARQ
WHADVP
SQ
.
WP
VBZ
NP
ADJP

QC/Entity

NN
NN
NN
NNP
NN
NN
NP
NNP
NNP
NNP
NNP
NNP

color
kind
film
VHS
beer
program

Daniele Pighin — Greedy Linearization of Tree Kernel Functions
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
Outline

- Mining the Fragment Space
  - Design Principles
  - Greedy Mining Algorithm

- Evaluation
  - Overview
  - Relevant Fragments
  - Algorithmic Efficiency
    - Accuracy of Linearized Classifiers
    - Process Efficiency

- Closing Remarks
  - Conclusions and Research Directions
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
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 → 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
Outline

- Mining the Fragment Space
  - Design Principles
  - Greedy Mining Algorithm

- Evaluation
  - Overview
  - Relevant Fragments
  - Algorithmic Efficiency
  - Accuracy of Linearized Classifiers
  - Process Efficiency

- Closing Remarks
  - Conclusions and Research Directions
Linear Space Optimization (LOpt)

- **Rationale:** Decode training data and learn a classifier in the linear space
- **Useful for:** Accuracy, fast classification
Linear Space Optimization (LOpt)

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

\[
\langle y_i, t_i \rangle_{TR} \xrightarrow{KSL} M(\alpha_i y_i, t_i)
\]
Linear Space Optimization (LOpt)

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

\[
\langle y_i, t_i \rangle_{TR} \xrightarrow{\text{KSL}} M(\alpha_i y_i, t_i) \xrightarrow{\text{KSM}} \text{Dictionary}
\]
**Linear Space Optimization (LOpt)**

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

\[
\langle y_i, t_i \rangle_{TR} \rightarrow \text{KSL} \rightarrow M(\alpha_i, y_i, t_i) \rightarrow \text{KSM} \\
\langle y_i, x_i \rangle_{TR} \\
\langle y_i, x_i \rangle_{TR} \rightarrow \text{LSG} \rightarrow \text{Dictionary} \\
\]
**Rationale:** Decode training data and learn a classifier in the linear space

**Useful for:** Accuracy, fast classification

\[
\langle y_i, t_i \rangle_{TR} \rightarrow \text{KSL} \rightarrow M_{\langle \alpha_i y_i, t_i \rangle} \rightarrow \text{KSM}
\]

\[
\langle y_i, x_i \rangle_{TR} \rightarrow \text{LSG} \rightarrow \text{Dictionary} \rightarrow \text{LSL} \rightarrow M_{\langle \alpha_i y_i, x_i \rangle}
\]
Linear Space Optimization (LOpt)

- **Rationale**: Decode training data and learn a classifier in the linear space
- **Useful for**: Accuracy, fast classification
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>ENTY</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><strong>89.66</strong></td>
<td><strong>87.50</strong></td>
<td><strong>75.56</strong></td>
<td><strong>84.53</strong></td>
<td><strong>92.08</strong></td>
<td><strong>94.60</strong></td>
<td><strong>83.92</strong></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><strong>81.66</strong></td>
<td>79.75</td>
<td><strong>76.19</strong></td>
<td>16.00</td>
<td>57.69</td>
<td>68.69</td>
<td>68.01</td>
</tr>
<tr>
<td>LOpt</td>
<td><strong>63.96</strong></td>
<td>67.33</td>
<td>74.14</td>
<td>63.33</td>
<td><strong>58.82</strong></td>
<td>62.83</td>
<td>61.70</td>
<td>67.08</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SRL</th>
<th>A0</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
<th>A5</th>
<th>MULTI</th>
</tr>
</thead>
<tbody>
<tr>
<td>STK</td>
<td>91.55</td>
<td>88.50</td>
<td>72.33</td>
<td>57.34</td>
<td>65.96</td>
<td>66.67</td>
<td>87.69</td>
</tr>
<tr>
<td>LOpt</td>
<td><strong>91.77</strong></td>
<td><strong>90.09</strong></td>
<td><strong>78.63</strong></td>
<td><strong>62.67</strong></td>
<td><strong>72.16</strong></td>
<td><strong>66.67</strong></td>
<td><strong>88.90</strong></td>
</tr>
</tbody>
</table>
### Multiclass Accuracy: LOpt vs. STK

**Classification accuracy**

<table>
<thead>
<tr>
<th>QC</th>
<th>ABBR</th>
<th>DESC</th>
<th>ENTY</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><strong>89.66</strong></td>
<td><strong>87.50</strong></td>
<td><strong>75.56</strong></td>
<td><strong>84.53</strong></td>
<td><strong>92.08</strong></td>
<td><strong>94.60</strong></td>
<td><strong>83.92</strong></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><strong>81.66</strong></td>
<td>79.75</td>
<td><strong>76.19</strong></td>
<td>16.00</td>
<td>57.69</td>
<td>68.69</td>
<td><strong>68.01</strong></td>
</tr>
<tr>
<td>LOpt</td>
<td><strong>63.96</strong></td>
<td>67.33</td>
<td>74.14</td>
<td>63.33</td>
<td><strong>58.82</strong></td>
<td>62.83</td>
<td>61.70</td>
<td>67.08</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SRL</th>
<th>A0</th>
<th>A1</th>
<th>A2</th>
<th>A3</th>
<th>A4</th>
<th>A5</th>
<th>MULTI</th>
</tr>
</thead>
<tbody>
<tr>
<td>STK</td>
<td>91.55</td>
<td>88.50</td>
<td>72.33</td>
<td>57.34</td>
<td>65.96</td>
<td><strong>66.67</strong></td>
<td>87.69</td>
</tr>
<tr>
<td>LOpt</td>
<td><strong>91.77</strong></td>
<td><strong>90.09</strong></td>
<td><strong>78.63</strong></td>
<td><strong>62.67</strong></td>
<td><strong>72.16</strong></td>
<td><strong>66.67</strong></td>
<td><strong>88.90</strong></td>
</tr>
</tbody>
</table>

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

- **Mining the Fragment Space**
  - Design Principles
  - Greedy Mining Algorithm

- **Evaluation**
  - Overview
  - Relevant Fragments
  - Algorithmic Efficiency
  - Accuracy of Linearized Classifiers
  - Process Efficiency

- **Closing Remarks**
  - Conclusions and Research Directions
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:**
  \[ KSL + KSM + LSG \text{ (train)} + LSL \]
- KSL and LSL are the main players
- We want global linear optimization \( \Rightarrow \) 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])

**Diagram:**
- Number of mined fragments vs. A1-LOpt, % of STK training time
- STK \( \equiv \) KSL
- LOpt
- KSM
- LSG
- LSL

**Graph:**
- X-axis: Number of mined fragments
- Y-axis: A1-LOpt, % of STK training time
- Data points for different components (STK, LOpt, KSM, LSG, LSL)
Rationale: Split training data to make KSL faster

Useful for: Faster learning cycles
Split Learning Efficiency and Accuracy

- For more than $10^4$ fragments, Split accuracy is at least 97% of STK.
- $\text{Split}_5$ is as accurate as STK for more than $10^4$ fragments.
- $\text{Split}_5$ training is twice as fast as STK.
- With 5 CPUs, $\text{Split}_5$ is up to 3 times as fast.
- The bottleneck for parallelization is LSL.
- [Pighin and Moschitti, 2009a]: 10x speedup on binary classifier trained on 1M instances with $\text{Split}_5$ and 5 CPUs.
Outline

- **Mining the Fragment Space**
  - Design Principles
  - Greedy Mining Algorithm

- **Evaluation**
  - Overview
  - Relevant Fragments
  - Algorithmic Efficiency
  - Accuracy of Linearized Classifiers
  - Process Efficiency

- **Closing Remarks**
  - Conclusions and Research Directions
Conclusions

- A **novel method** for feature selection in tree kernel spaces
- **Modeling simplicity** of convolution kernels with the **speed and clarity** of linear models
- Very **aggressive**, greedy feature selection in the huge TK space
- Significant dimensionality reduction (for SRL, \(\sim 20 - 30\) OOM!)
- Relevant features discovered in the rich TK space are **observable**
- Decoding time **linear** with size of input tree
- Linearized classifiers sport very **good accuracy** and **faster classification time**
- **Training time** on larger datasets can also be improved (Split)
Directions for Future Work

- 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
Thanks!

Q&A Time!
References


References (2)

  Speeding up training with tree kernels for node relation labeling.
  In *Proceedings of HLT-EMNLP'05*.

  Fast methods for kernel-based text analysis.
  In *Proceedings of ACL'03*.

  Fast methods for kernel-based text analysis.
  In *IPSJ SIG Notes, 2004(29):163–168*.

  Tree Kernels for Semantic Role Labeling.

  Exploiting Syntactic and Shallow Semantic Kernels for Question/Answer Classification.
  In *Proceedings of ACL'07*.

  Convolution kernels on constituent, dependency and sequential structures for relation extraction.
  In *Proceedings of EMNLP'09*.

- Pei, J., Han, J., Asl, M. B., Pinto, H., Chen, Q., Deyal, U., and Hsu, M. C. (2001).
  PrefixSpan Mining Sequential Patterns Efficiently by Prefix Projected Pattern Growth.
  In *Proceedings of ICDE’01*.

  Efficient Linearization of Tree Kernel Functions.
  In *Proceedings of CoNLL’09*.

  Reverse Engineering of Tree Kernel Feature Spaces.

Daniele Pighin — Greedy Linearization of Tree Kernel Functions
References (3)

  Approximate tree kernels.

  Using ltag based features in parse reranking.
  Association for Computational Linguistics.

  Sequence and Tree Kernels with Statistical Feature Mining.
  In Proceedings of NIPS’05.

  Overview of the TREC 2001 Question Answering Track.
  In In Proceedings of the Tenth Text REtrieval Conference (TREC), pages 42–51.

  Efficiently mining frequent trees in a forest.
  In Proceedings of KDD’02.

  Question classification using support vector machines.
  In Proceedings of SIGIR’03, pages 26–32.

  Exploring Syntactic Features for Relation Extraction using a Convolution Tree Kernel.
  In Proceedings of NAACL.

  Automatic Learning of Textual Entailments with Cross-Pair Similarities.
  In Proceedings of ACL’06.