Convolution Kernels for Natural Language

Paper by: Michael Collins, Nigel Duffy
Presented by: Ruinan (Renie) Lu

Outline

- Natural Language Processing (NLP) Tasks
- Introduction to Kernels
- A Tree Kernel
- Linear Models for Parsing and Tagging
- Experimental Results
- Conclusions

NLP tasks

- Assume: some training set of structures.
  - An "observed" string (a sentence)
  - Hidden structure (an underlying state sequence or tree)
- Task: learn mapping from an input string to its hidden structure.
  - Parsing – tasks involving trees
  - Tagging– tasks involving hidden state sequence

Three typical structures from NLP tasks

- Parse tree:
  Lou Gerstner is chairman of IBM →
  [S [NP Lou Gerstner] [VP is [NP chairman [PP of [NP IBM]]]]]
- Underlying state sequence:
  Lou Gerstner is chairman of IBM →
  Lou/SP Gerstner/CP is/N chairman/N of/N IBM/SC
- Part-of-speech tags:
  Lou/N Gerstner/N is/V chairman/N of/P IBM/N

NLP Key Issue

- Key issue: ambiguity
  - Although only one analysis is plausible, there may be many many possible analyses.

Dealing with Ambiguity…

- Stochastic grammar:
  -- PCFG (Probabilistic Context Free Grammar) for parsing
  -- HMM (Hidden Markov Model) for tagging
- Probabilities are attached to rules in the grammar.
- Rule probabilities are estimated using MLE (Maximum likelihood estimation).
- Probabilities are used to rank the competing analyses for the same sentence.

PCFGs as a parsing method

- Counts the relative # of occurrences of a given rule.
- Uses the count to represent its learned knowledge.
- Makes strong independence assumptions.
- Ignores substantial amounts of structural information (e.g. assume rules applied at level i in the parse tree are unrelated to those applied at level i+1).

Dealing with Ambiguities …

- Alternative suggested by the paper: Kernels
- Kernel approach in this paper:
  - sensitive to larger sub-structures of trees or state sequences;
  - discriminative parameter estimation method;
  - optimizes a criterion directly related to error rate.
- Other applications of Kernels:
  - PCA over discrete structures
  - Classification
  - regression problems

Outline

- Natural Language Processing (NLP) Tasks
- Introduction to Kernels
- A Tree Kernel
- Linear Models for Parsing and Tagging
- Experimental Results
- Conclusions

Introduction to Kernels

- Algorithms involving kernel methods:
  - Perceptron.
  - SVM (Support Vector Machine).
  - PCA (Principal Component Analysis).
- Key property of these algorithms:

  Dot product is the only operation required.
- Mercer kernels: $\Re^d \rightarrow \Re^n$

Applying kernel methods to NLP problems – this paper

- Problem in many NLP tasks: $\Re^d$

  input domain cannot be neatly represented as a subset of strings/trees/discrete structures

In this paper:
- Provides a mechanism to convert the objects into feature vectors
- Allow computationally feasible representations in high dimensional feature spaces (e.g. parse tree representation tracks all sub-trees)
- Applies tree kernel to parsing using perceptron algorithm

Outline

- Natural Language Processing (NLP) Tasks
- Introduction to Kernels
- A Tree Kernel
- Linear Models for Parsing and Tagging
- Experimental Results
- Conclusions

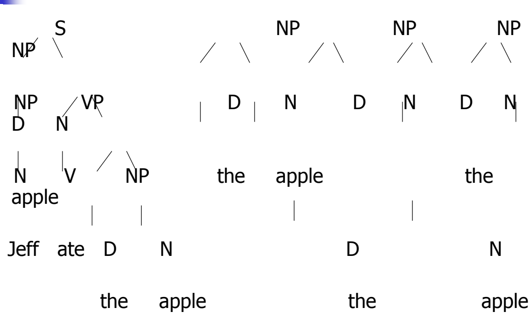
Recall: PCFGs as a parsing method

- Counts the relative # of occurrences of a given rule.
- Uses the count to represent its learned knowledge.
- Makes strong independence assumptions.
- Ignore substantial amounts of structural information (e.g. assume rules applied at level I in the parse tree are unrelated to those applied at level i+1).

More Structural Information!

- Why?
  - Capture higher order dependencies between grammar rules.
- How?
  - Consider all tree fragments.

An example tree and some sub-trees



Representations

- Enumerate (implicitly) all tree fragments: 1,…,n.
- Represent each tree by an n-d vector:
  - $h_i(T)$ = # of occurrences of the i'th tree fragment in tree T
  - Tree T is represented as: $h(T) = (h_1(T), h_2(T), ..., h_n(T))$.
  - Note: n will be huge.

Previous work on the representation

- Bod, R. (1998). Beyond Grammar: An Experience-Based Theory of Language. CSLI Publications/Cambridge University Press.
- Comments:
  - Exact implementation is infeasible.
  - Training and decoding algorithms depending on # of sub-trees.
  - Lack justification of the parameter estimation technique.
- Goodman, J.(1996). Efficient algorithms for parsing the DOP model. In Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP 96), pages 143-152 .
  - Approximately implement Bod's method efficiently.
  - Still lack justification of the parameter estimation techniques.

Key: definition of an appropriate kernel

- Define:
  - $h_i(T)$: # of occurrences of the i'th tree fragment in tree T
  - $K(T_1,T_2) = h(T_1)\cdot h(T_2) = \sum_i h_i(T_1) h_i(T_2)$ Inner product between two trees T_1 and T_2.
  - Problem: the sum is over an exponential number of sub-trees.

Definitions – cont.

- Cure :
  - Indicator function:

$$I_i(n)=\begin{cases}1, & \text{if subtree } i \text{ roots at node } n\\ 0, & \text{otherwise}\end{cases}$$

  - N_1 and N_2: set of nodes in T_1 and T_2.
  - Therefore,

$$h_i(T_1)=\sum_{n_1\in N_1} I_i(n_1)\ ,\ h_i(T_2)=\sum_{n_2\in N_2} I_i(n_2)$$

Definitions – cont.

- And

$$h(T_1)\cdot h(T_2)=\sum_i h_i(T_1)h_i(T_2)=\sum_{n_1\in N_1}\sum_{n_2\in N_2}\sum_i I_i(n_1)I_i(n_2)=\sum_{n_1\in N_1}\sum_{n_2\in N_2}C(n_1,n_2)$$

- Where

$$C(n_1,n_2)=\sum_i I_i(n_1)I_i(n_2)$$

- Define $C(n_1,n_2)$ recursively:
  - If the productions at n_1 and n_2 are different, $C(n_1,n_2)=0$.
  - If the productions at n_1 and n_2 are the same, and n_1 and n_2 are pre-terminals then $C(n_1,n_2)=1$.
  - Else if the productions at n_1 and n_2 are the same and n_1 and n_2 are not pre-terminals,

$$C(n_1,n_2)=\prod_{j=1}^{nc(n_1)}(1+C(ch(n_1,j),ch(n_2,j))).$$

  - This is the # of common sub-trees that are found rooted at both n_1 and n_2

Observations

- Observation from $h(T_1)\cdot h(T_2)=\sum_{n_1\in N_1}\sum_{n_2\in N_2}C(n_1,n_2)$

and the recursive definition of $C(n_1,n_2)\Rightarrow$ $h(T_1)\cdot h(T_2)$

can be calculated $O(|N_1||N_2|)$ time – pessimistic estimate.

Will run in time linear in the # of members $(n_1,n_2)\in N_1\times N_2$ s.t. the productions at n_1 and n_2 are the same.

Issues remain

- Value of $K(T_1,T_2)$ will depend greatly on the size of the trees
- Cure: normalize the kernel
- Peaked kernel (large kernel value when same trees)
- Cure:
  - Radialize the kernel (Haussler) – not actually helps
  - Down-weight the contribution of larger tree fragments to the kernel.
    - Restrict the depth of the tree fragment
    - Scale the importance according to size, pick $0<\lambda\le 1$, $C(n_1,n_2)=\lambda$ and $C(n_1,n_2)=\lambda\prod_{j=1}^{nc(n_1)}(1+C(ch(n_1,j),ch(n_2,j)))$.
- Modified kernel:

$$h(T_1)\cdot h(T_2)=\sum_i \lambda^{size_i}h_i(T_1)h_i(T_2)$$

Outline

- Natural Language Processing (NLP) Tasks
- Introduction to Kernels
- A Tree Kernel
- Linear Models for Parsing and Tagging
- Experimental Results
- Conclusions

Set up

- Training data in parsing $\{s_i,t_i\}$,
  - s_i: sentence, t_i: correct tree for s_i
- Enumerate a set of candidates for a particular sentence:
- x_{ij}: j'th candidate for the i'th sentence.
- $C(s_i)=\{x_{i1},x_{i2}\ldots\}$: set of candidates for s_i.
- Take x_{i1} to be the correct parse for s_i (i.e., $x_{i1}=t_i$)
- $h(x_{ij})$: feature vector of x_{ij} in the space $\Re^n$.
- $\bar{w}\in\Re^n$: parameters of the model.
- $\bar{w}\cdot h(x_{ij})$:

Goal and Approach

- Goal: $\arg\max_{x \in C(s)} \overline{w} \cdot h(x_{ij})$
- Observation: $\overline{w}(h(x_{i1}) - h(x_{ij})) > 0 \quad \forall i, \forall j \geq 2$
- Approach: modified Perceptron and SVM to search for "dual" parameters to determine $\overline{w}^*$: $\overline{w}^* = \sum_{(i,j)} a_{i,j}(h(x_{i1}) - h(x_{ij}))$

Modified Perceptron Algorithm

- Define: $F(x) = \sum_{(i,j)} a_{i,j}(h(x_{i1})h(x) - h(x_{ij})h(x))$
- Initialization: Set dual parameters $\alpha_{ij} = 0$
- For $i = 1 \ldots n \;,\; j = 2 \ldots n_i$
-     If $F(x_{i1}) > F(x_{ij})$ do nothing ;
-     else $\alpha_{ij} = \alpha_{ij} + 1$

Calculate Score of Parse

$$\overline{w}^* \cdot x = \sum_{(i,j)} a_{i,j}(h(x_{i1})h(x) - h(x_{i2})h(x)) > 0$$

Outline

- Natural Language Processing (NLP) Tasks
- Introduction to Kernels
- A Tree Kernel
- Linear Models for Parsing and Tagging
- Experimental Results
- Conclusions

Experiment Design

- Problem: parsing the Penn tree-bank ATIS corpus.
- Data preparation: split tree-bank randomly into
  - A training set of size 800
  - A development set of size 200
  - A test set of size 336
- Apply PCFG to training set –> 100 top candidate parse tree.
- Beam search to get a set of (20) parses.
- Apply Voted Perceptron using tree kernel to the test set.
  - Two parameters to consider:
    - Maximum depth of sub-tree examined
    - Scaling factor to down-weight deeper trees

Experiment Design cont.

- Report a parse score: $100\% * \frac{1}{\sum_i g_i} \sum_i g_i \times \frac{1}{2}(\frac{c_i}{p_i} + \frac{c_i}{g_i})$
- Where
  - c_i # of correctly placed constituents in the i'th test tree
  - p_i # of constituents proposed
  - g_i # of constituents in the true parse tree
  - Constituent: non-terminal label and its span.
- Use the development set to choose the best parameter settings
- Use the best parameter settings (on the development sets) for each split to train on both the training and development sets.
- Test on the test set.

maximum depth/ scaling factor of sub-tree

Depth	1	2	3	4	5	6
Score	73 ± 1	79 ± 1	80 ± 1	79 ± 1	79 ± 1	78 ± 0.01
Imp.	-1 ± 4	20 ± 6	23 ± 3	21 ± 4	19 ± 4	18 ± 3

Scale	0.1	0.2	0.3	0.4	0.5	0.6
Score	77 ± 1	78 ± 1	79 ± 1	79 ± 1	79 ± 1	79 ± 1
Imp.	11 ± 6	17 ± 5	20 ± 4	21 ± 3	21 ± 4	22 ± 4

Result Comparison

- PCFG: 74%
- Best choice of maximum depth 80% ± 1
- Best choice of scaling factor 80% ± 1

Outline

- Natural Language Processing (NLP) Tasks
- Introduction to Kernels
- A Tree Kernel
- Linear Models for Parsing and Tagging
- Experimental Results
- A Compressed Representation
- Conclusions

Conclusions

- Conclusions
  - Convolution kernels applied to NLP parsing
  - Tree structure
  - Example domain: parsing English sentences
- Future Work
  - Compare with other methods that perform better than a PCFG for NLP parsing.
  - Convolution kernels combined with other kernel based algorithms (kernel PCA and spectral clustering) to achieve computational attraction.

Thank you!

Convolution Kernels Overview

Presented by Alex Kosolapov

Convolution Math

- Continuous $h(x) = g(x) * f(x) = \int_{-\infty}^{\infty} g(x-u) f(x) dx$
- Descrete $h(x) = g(x) * f(x) = \sum g(x-u) f(x)$
- Properties:
  - Associative
  - Commutative
  - Distributive

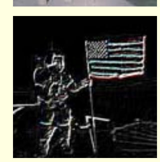
Applications

- Noise Filters
  - Filter out low frequency
  - Filter out high frequency
- Applied in signal processing, image processing

Convolution: Edge detection

- Edge detection (Laplacian kernel, Sobel kernel), smoothing
  - Kernel – a matrix applied to an image
- Sobel edge detection kernel

Vertical:	Horizontal:
-1 0 1	1 2 1
-2 0 2	0 0 0
-1 0 1	-1 -2 -1



Convolution Kernel

Haussler, D. (1999). Convolution kernels on discrete structures. *UCSC-CRL-99-10*

- For classifying discrete structures, e.g., strings, trees, graphs etc.
- Often not feasible to extract real-valued features of structures
- Convolution kernel: compute inner product of features without explicitly extracting features
- With a convolution kernel we can compute distance between structures x and y
- similarity metrics introduced based on radial basis, exponential, ANOVA kernels, hidden Markov random fields

Convolution Kernels

- Obtained from other kernels by sum over products
- Can do this iteratively

Convolution Kernels for Natural Languages

By,
Michael Collins
Nigel Duffy

Comments by,
Srikanth Varanasi

Representation

- Linear combination of parse trees
- Search for sub trees that occur more than once
- Construct a weighted acyclic graph
- The common sub tree appears only one in this graph
- Repeat the above process

Compressed representation

- The above process lead us to a compressed representation
- Perceptron may be evaluated on this new tree
- Advantage:

  Appears to save considerable amount of computation