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: \( \rightarrow \mathbb{R} \)

Applying kernel methods to NLP problems – this paper

- Problem in many NLP tasks: 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
- Allows 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 1 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:
  - # of occurrences of the i'th tree fragment in tree T
  - Tree T is represented as:

Previous work on the representation
- Comments:
  - Exact implementation is infeasible.
  - Training and decoding algorithms depending on # of sub-trees.
  - Lack justification of the parameter estimation technique.
  - Approximately implement Bod's method efficiently.
  - Still lack justification of the parameter estimation techniques.

Key: definition of an appropriate kernel
- Define:
  - : # of occurrences of the i'th tree fragment in tree T
  - Inner product between two trees
- Problem: the sum is over an exponential number of sub-trees.
Definitions – cont.

- **Cure**: 
  - Indicator function: 
    \[ I(n) = \begin{cases} 1 & \text{if subtree roots at node } n \\ 0 & \text{otherwise} \end{cases} \]
  - \( N \) and \( N_i \): set of nodes in.
  - Therefore, 
    \[ h(T_i) = \sum_{n \in N_i} I(n), \quad h(T_i) = \sum_{n \in N} I(n) \]

- And 
  \[ h(T_i) = \sum_{n \in N_i} C(n, n_i) \]

- Observations
  - Observation from 
    \[ h(T_i) = \sum_{n \in N_i} \sum C(n_i, n) \]
  - and the recursive definition of 
    \[ C(n_i, n_j) \Rightarrow h(T_i) = h(T_j) \]
  - can be calculated \( \log |N_i| \) \( \times \) \# of members \( N \) \( \times \) \# of members \( N_i \)
  - Will run in time \( \log |N_i| \) \# of members 
  - s.t. the productions at 
  - are the same.

- Issues remain
  - Value of \( k(T_i, T_j) \) 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 (Hausler) – 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
    \[ \lambda \leq 10 \]
    \[ C(n, n_i) \Rightarrow h(T_i) = \sum \lambda h(T_i) \]
    \[ \lambda \leq 10 \]

- Set up
  - Training data in parsing: \( x_i \), \( w \)
    \[ x_i = \text{sentence}, \quad w_i = \text{correct tree for} \]
  - Enumerate a set of candidates for a particular sentence:
    \[ C(x_i) \] \( \times \) \# of candidates for the \( i \)th sentence.
  - set of candidates for
    \[ \{ x_i \} \text{ to be the correct parse for} \]
    \[ \lambda \leq 10 \]
    \[ \text{take } x_i \text{ to be the correct parse for} \]
    \[ \lambda \leq 10 \]
    \[ \text{feature vector of } w_i \text{ in the space} \]
    \[ w_i \] parameters of the model.
  - Natural Language Processing (NLP)
  - Tasks
    - Introduction to Kernels
    - A Tree Kernel
    - Linear Models for Parsing and Tagging
    - Experimental Results
    - Conclusions
Goal and Approach

**Goal:**
\[ \text{max}_{w \in \mathbb{R}^n} \langle w, x \rangle \]

**Observation:**
\[ h(x_j) - h(x_j) > 0 \quad \forall i, j \geq 2 \]

**Approach:** modified Perceptron and SVM to search for “dual” parameters to determine:
\[ \sum_{i,j} a_{i,j} (h(x_i) - h(x_j)) \]

Modified Perceptron Algorithm

**Define:**
\[ f(x) = \sum_{j \geq 1} a_{j} h(x_j) h(x) - h(x_j) h(x) \]

**Initialization:** Set dual parameters
**For** \( i = 1 \ldots n \), \( j = 2 \ldots n \)
- If \( f(x_i) > f(x_j) \) do nothing;
- else \( a_{i,j} = a_{i,j} + 1 \)

Calculate Score of Parse

\[ \bar{w} \cdot x = \sum_{i,j} a_{i,j} (h(x_i) h(x) - h(x_j) h(x)) > 0 \]

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:**
\[ \frac{1}{g_t} \sum_{i \geq 1} \frac{1}{p_t} \sum_{j \geq 1} \left( \frac{c_t}{g_t} + \frac{c_t}{g_t} \right) \]

Where
- \( c_t \) # of correctly placed constituents in the i’th test tree
- \( p_t \) # of constituents proposed
- \( g_t \) # 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.
**Result Comparison**

- PCFG: 74%
- Best choice of maximum depth: 1
- Best choice of scaling factor: 0.1

**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
  \[ h(x) = g(x) * f(x) = \int g(x-u)f(u)du \]

- Discrete
  \[ h(x) = \sum g(x-u)f(u) \]

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

Convolution Kernel


- 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,
Srikarth 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