ECML PKDD 2011

Fast Support Vector Machines for Structural Kernels

Aliaksei Severyn and Alessandro Moschitti

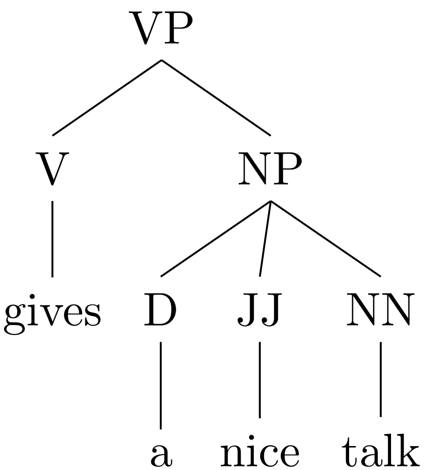
University of Trento, Italy

September 7, 2011

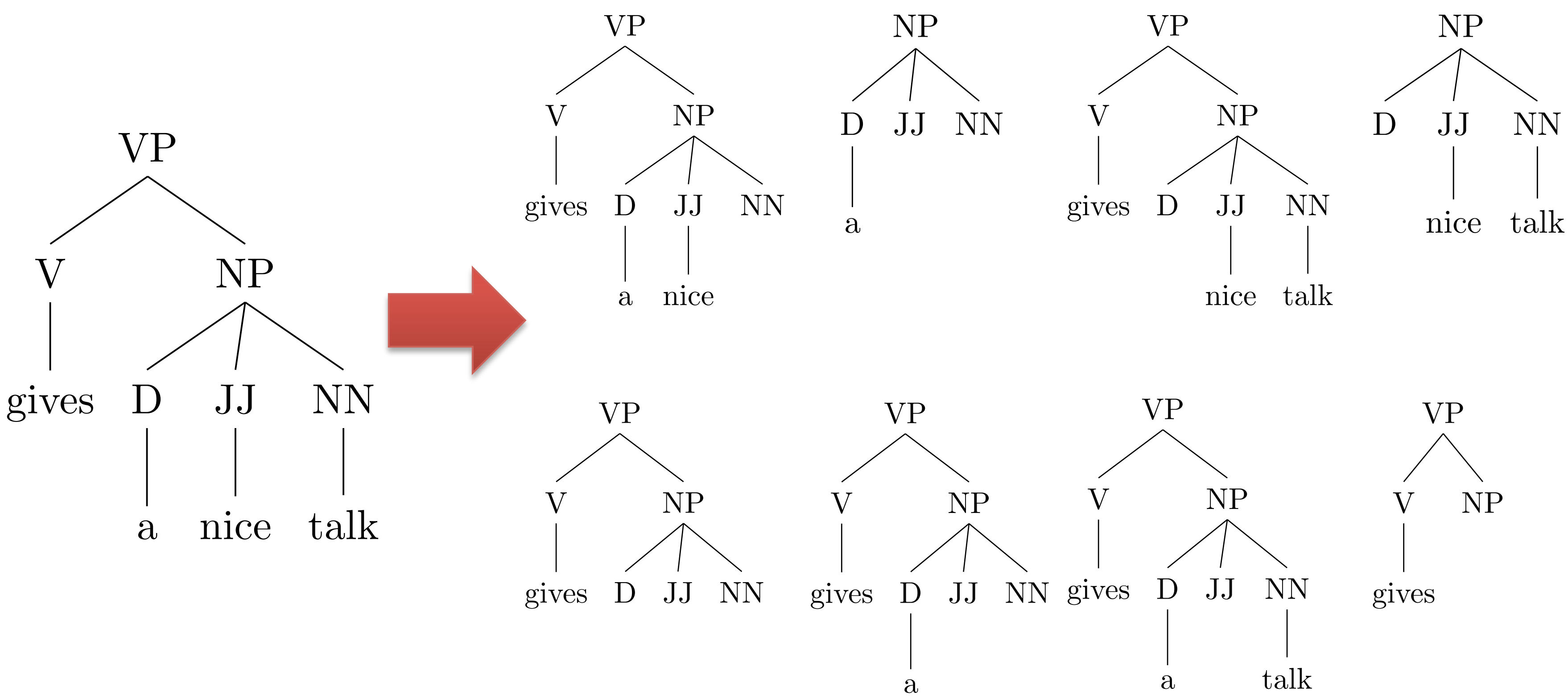
Structured Data

- Much of “real data” is structured:
  - Sequences
  - Trees
  - Graphs
- Embedding in real vector space requires **extensive** pre-processing and **feature engineering**

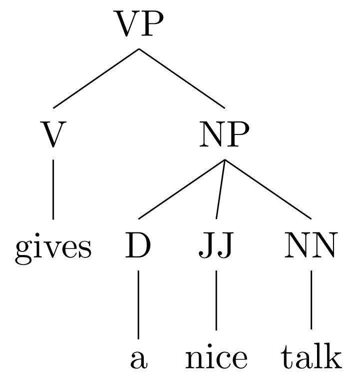
Ex: Predicate-Argument Identification



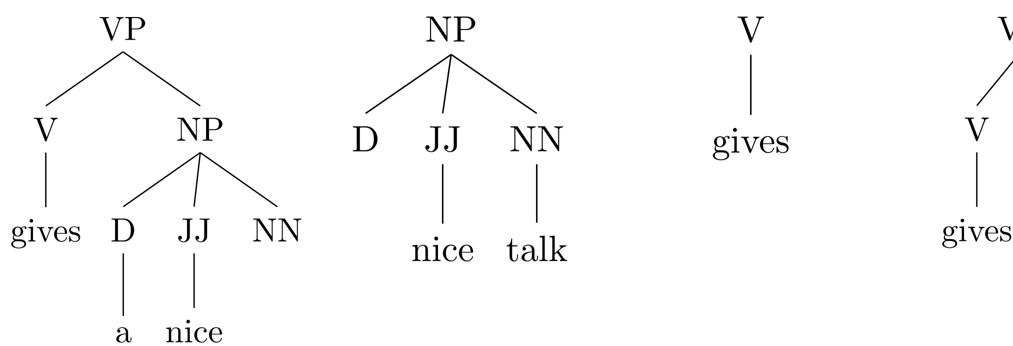
Ex: Predicate-Argument Identification



Explicit feature vector representation



$$\phi(T_x) = \vec{x} = (0, .., 1, .., 0, ..1, .., 0, ..1, ..0, ..1, ..0)$$



State of the art in many tasks

- Natural Language processing
  - Relation Extraction, Co-reference Resolution, Semantic Role Labeling, Textual Entailment Recognition, Question Classification…
- Information Retrieval and data mining:
  - Question Classification
- Bioinformatics
  - DNA classification, Protein-Protein Interaction

Training SVMs with Structural Kernels

- **Kernel methods** require learning in **dual spaces**
- Conventional methods (SVM-Light) or SMO **scale quadratically** in the number of examples
- This prohibits training of SVMs with **structural kernels** on **large data**

Key ideas in the paper

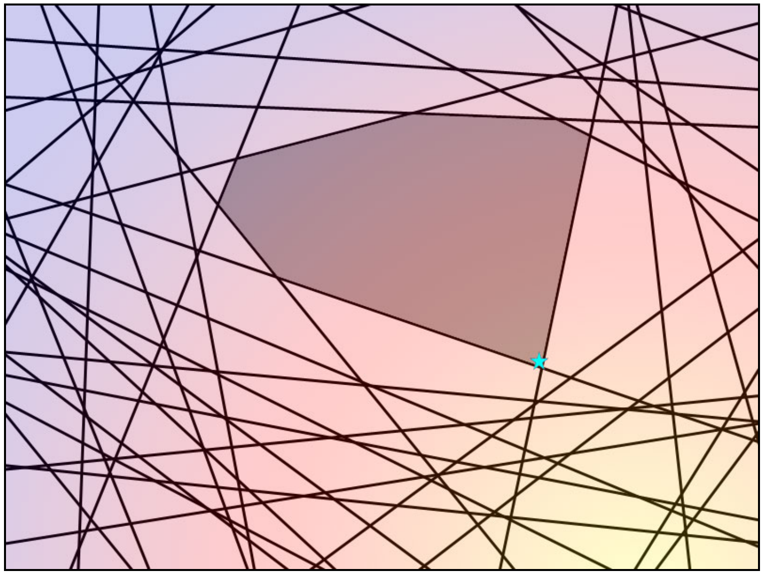
3 important enhancements of the approximate cutting plane algorithm (CPA) for SVMs with structural kernels:

1. **Compact** yet **exact** representation of cutting plane models using **directed acyclic graphs** to speed up training and classification
2. **Parallelization** to make the training **scale linearly** with the number of CPUs
3. **Alternative sampling** strategy for **class-imbalanced problem**

CPA in a nutshell

- Introduced in the context of **Structural SVMs** (Tsochantaridis et.al.,2004)
- Gives **linear training time** with **linear kernels**

CPA in a nutshell



Original SVM Problem

Exponential constraints

Most are dominated by a small set of "important" constraints

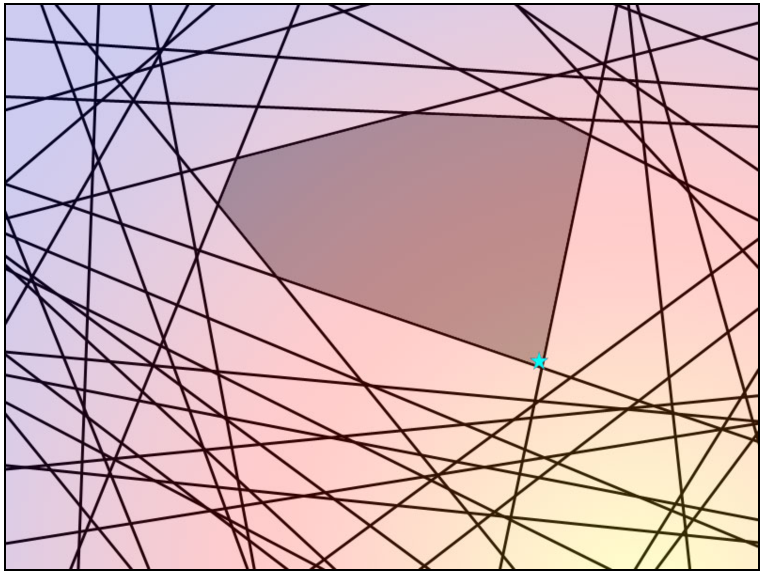
* courtesy of Thorsten Joachims

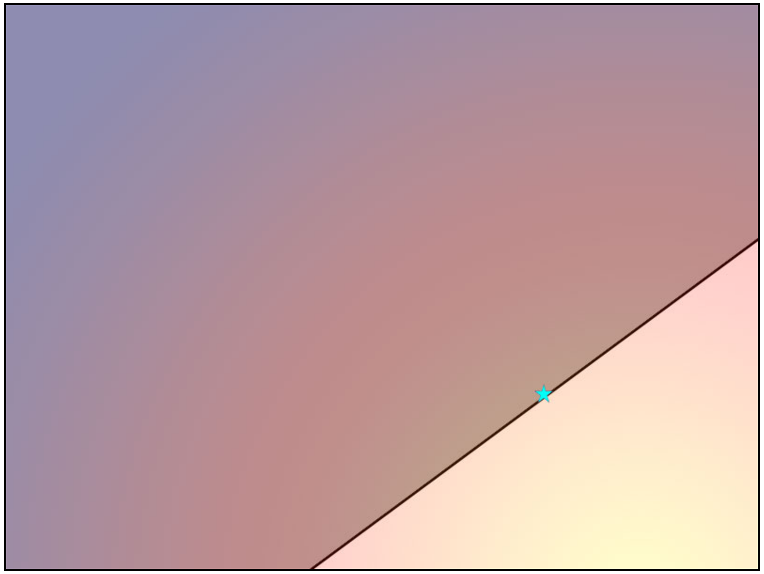
CPA SVM Approach

Repeatedly finds the next most violated constraint…

…until set of constraints is a good approximation.

CPA in a nutshell





Original SVM Problem

Exponential constraints

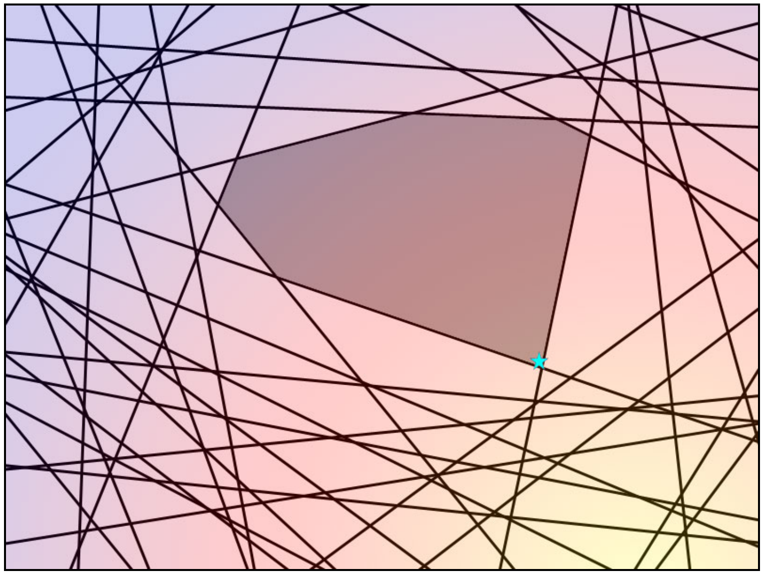
Most are dominated by a small set of "important" constraints

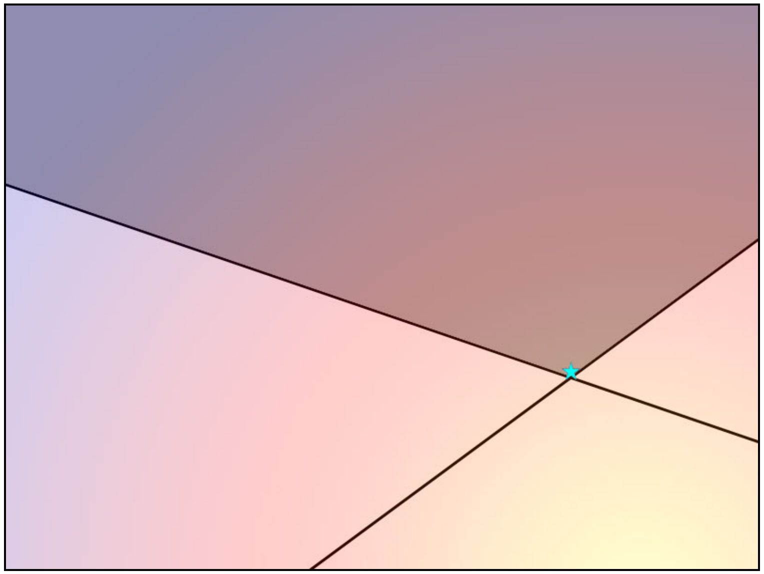
CPA SVM Approach

Repeatedly finds the next most violated constraint…

…until set of constraints is a good approximation.

CPA in a nutshell





Original SVM Problem

Exponential constraints

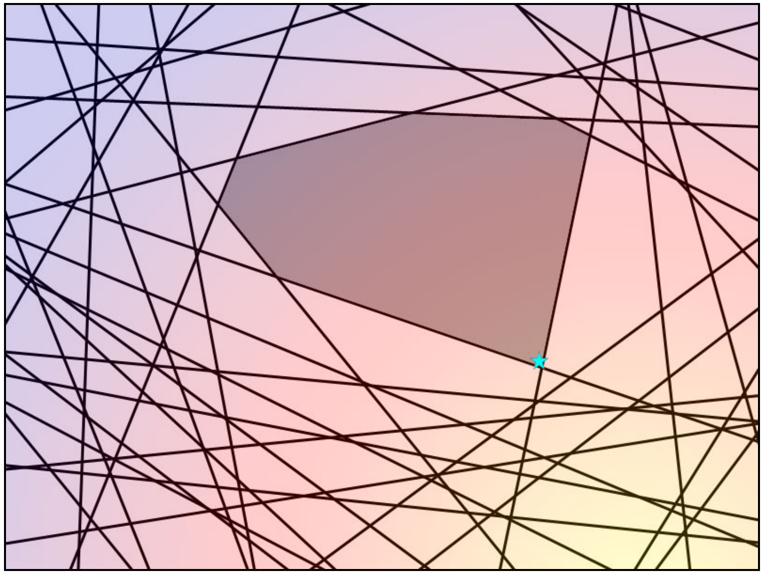
Most are dominated by a small set of "important" constraints

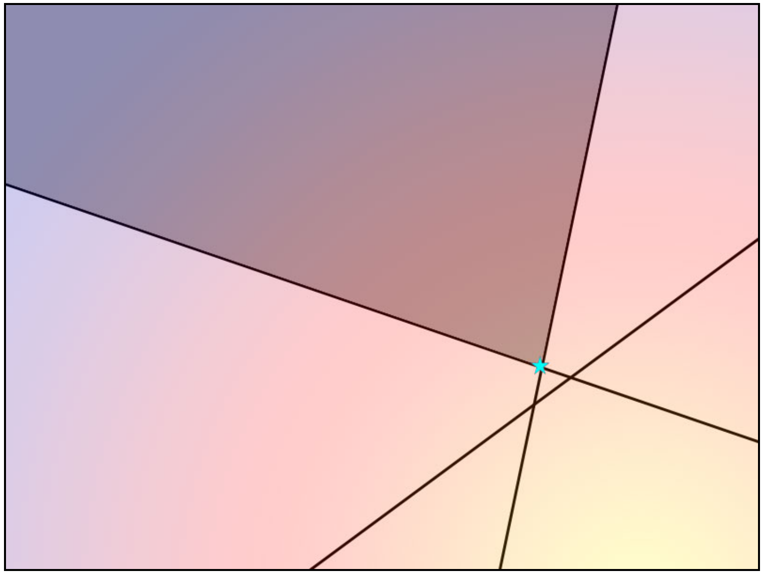
CPA SVM Approach

Repeatedly finds the next most violated constraint…

…until set of constraints is a good approximation.

CPA in a nutshell





Original SVM Problem

Exponential constraints

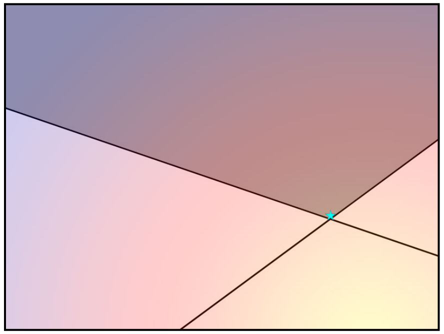
Most are dominated by a small set of "important" constraints

CPA SVM Approach

Repeatedly finds the next most violated constraint…

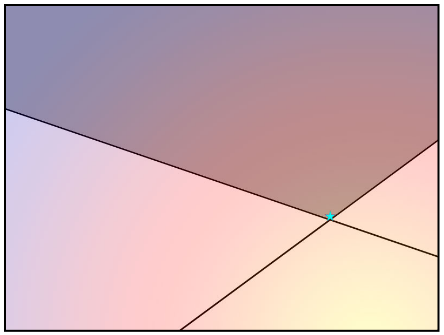
…until set of constraints is a good approximation.

Computing most violated constraint (MVC)



$$\vec{w} \cdot \phi(\vec{x}_i) = \sum_{j=1}^{t} \alpha_j \vec{g}^{(j)} \cdot \phi(\vec{x}_i)$$

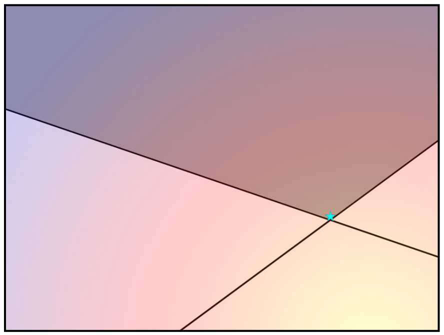
Computing most violated constraint (MVC)



$$\vec{w} \cdot \phi(\vec{x}_i) = \sum_{j=1}^{t} \alpha_j \vec{g}^{(j)} \cdot \phi(\vec{x}_i)$$

$$\vec{g}^{(j)} = \frac{1}{n} \sum_{k=1}^{n} c_k^{(j)} y_k \phi(\vec{x}_k)$$

Computing most violated constraint (MVC)



$$\vec{w} \cdot \phi(\vec{x}_i) = \sum_{j=1}^{t} \alpha_j \vec{g}^{(j)} \cdot \phi(\vec{x}_i)$$

$$\vec{g^{(j)}} = \frac{1}{n} \sum_{k=1}^{n} c_k^{(j)} y_k \phi(\vec{x}_k)$$

$$\vec{w} \cdot \phi(\vec{x}_i) = \sum_{j=1}^{t} \alpha_j \sum_{k=1}^{n} \left(\frac{1}{n} c_k^{(j)} y_k \right) K(\vec{x}_i, \vec{x}_k)$$

Computational bottleneck

- Main **bottleneck** to apply kernels comes from the inner product:

$$\vec{w} \cdot \phi(\vec{x}_i) = \sum_{j=1}^{t} \alpha_j \sum_{k=1}^{n} \left(\frac{1}{n} c_k^{(j)} y_k \right) K(\vec{x}_i, \vec{x}_k)$$

Computational bottleneck

- Main **bottleneck** to apply kernels comes from the inner product:

$$\vec{w} \cdot \phi(\vec{x}_i) = \sum_{j=1}^{t} \alpha_j \sum_{k=1}^{n} \left(\frac{1}{n} c_k^{(j)} y_k \right) K(\vec{x}_i, \vec{x}_k)$$

- Use **sampling** to **approximate** exact cutting plane models (Yu & Joachims, 2009)

$$\vec{w} \cdot \phi(\vec{x}_i) = \sum_{j=1}^{t} \alpha_j \sum_{k=1}^{r} \left(\frac{1}{r} c_k^{(j)} y_k \right) K(\vec{x}_i, \vec{x}_k)$$

Approximate CPA + Structural Kernels

(Severyn & Moschitti, ECML 2010)

applied **this idea** to SVM learning with **structural kernels**, e.g. tree kernels,

on **large data** (millions of examples)

achieving **speed up** factors up to 10

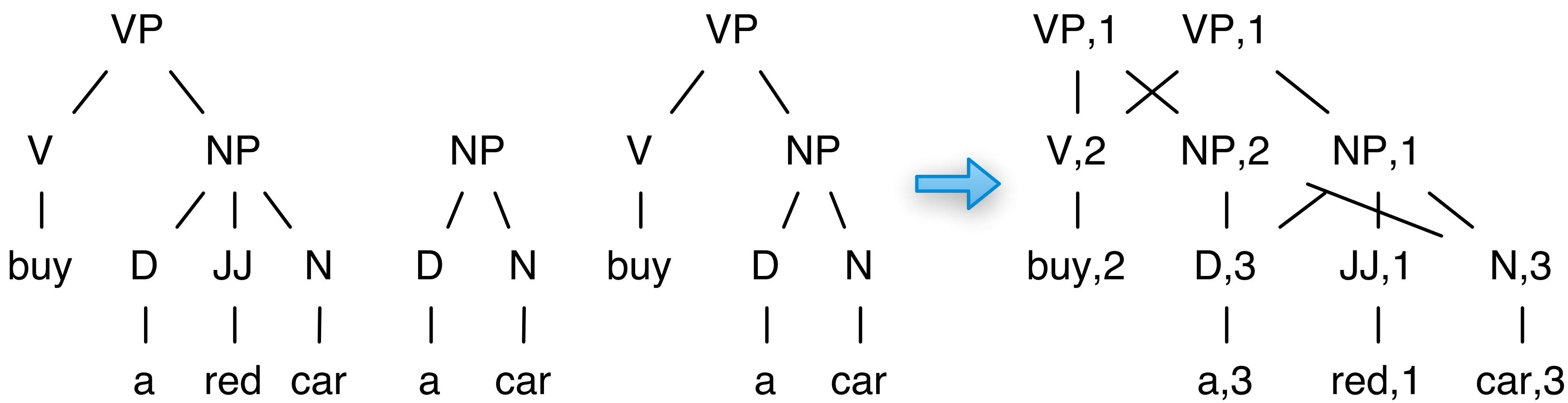
over conventional SVMs (SVM-light-TK)

7.5 days -> **0.5** days

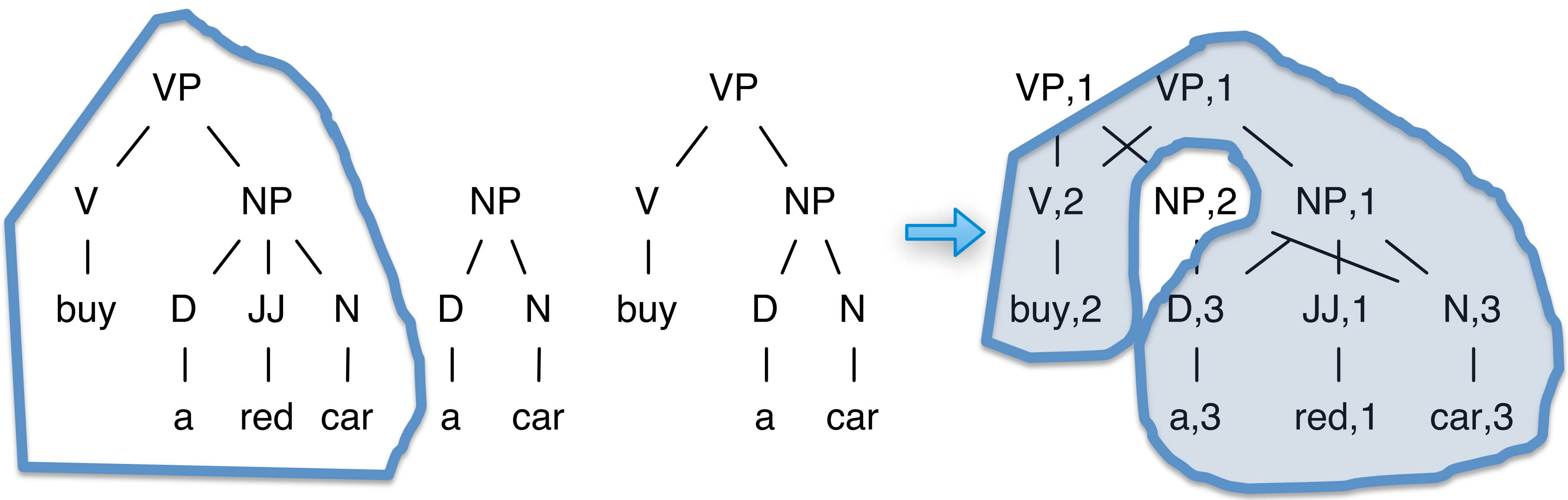
Compact model representation using DAGs

- For **structured data**, e.g. sequences, trees, graphs, many examples share **common sub-structures**
- **Key idea** to **reduce** the number of kernel evaluations - avoid computations over **repeating sub-structures**
- Use DAGs to **compact** a collection of trees
- Gives **exact** kernel evaluation (proof in the paper)

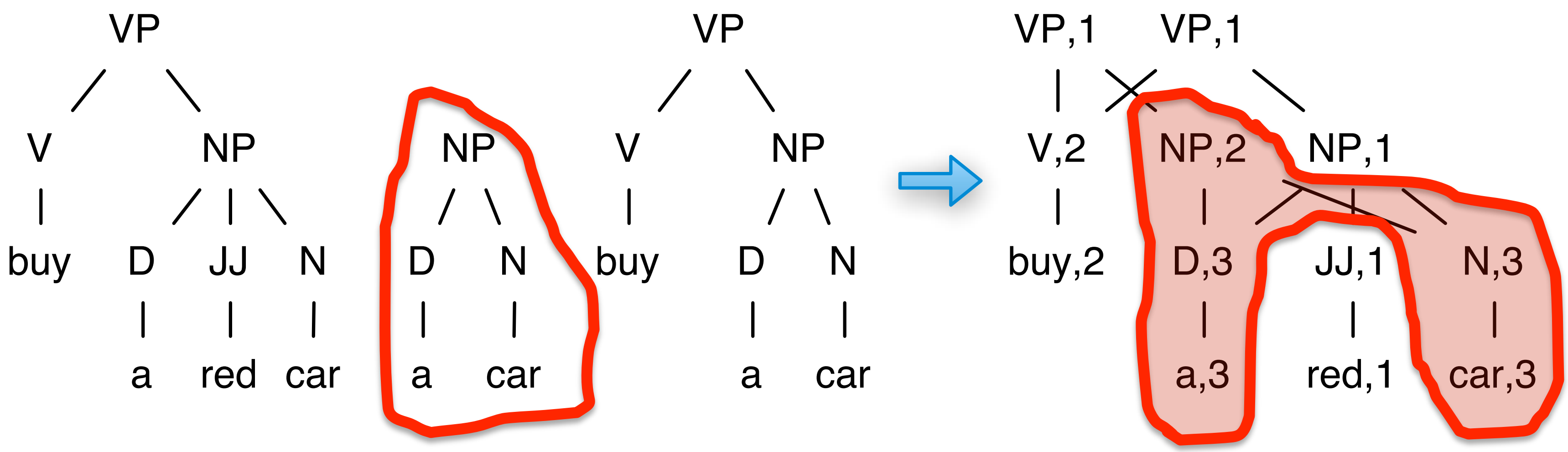
Three syntactic trees and the resulting DAG



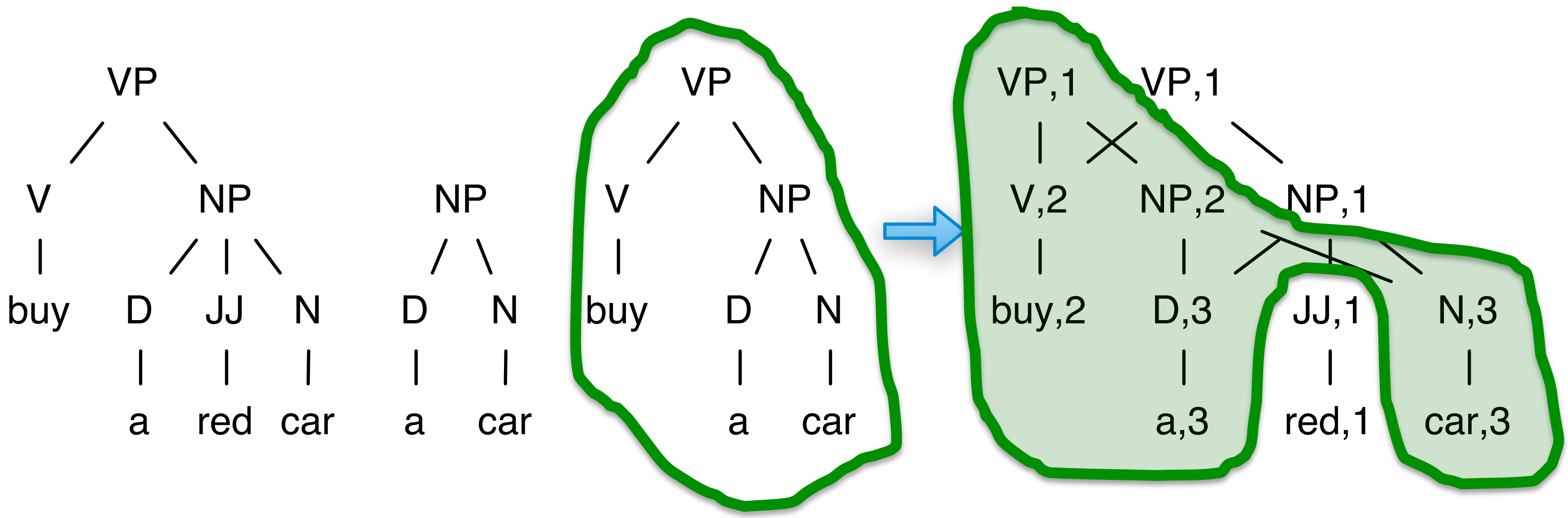
Three syntactic trees and the resulting DAG



Three syntactic trees and the resulting DAG



Three syntactic trees and the resulting DAG



Computational bottleneck

- Main **bottleneck** to apply kernels comes from the inner product:

$$\vec{w} \cdot \phi(\vec{x}_i) = \sum_{j=1}^{t} \alpha_j \sum_{k=1}^{n} \left(\frac{1}{n} c_k^{(j)} y_k \right) K(\vec{x}_i, \vec{x}_k)$$

SDAG

- Compacts **each** CPA model into a **single** DAG

$$\vec{w} \cdot \phi(\vec{x}_i) = \sum_{j=1}^{t} \alpha_j \boxed{\sum_{k=1}^{r} \left(\frac{1}{r} c_k^{(j)} y_k \right) K(\vec{x}_i, \vec{x}_k)}$$

SDAG

- Compacts **each** CPA model into a **single** DAG

$$\vec{w} \cdot \phi(\vec{x}_i) = \sum_{j=1}^{t} \alpha_j \boxed{\sum_{k=1}^{r} \left(\frac{1}{r} c_k^{(j)} y_k\right) K(\vec{x}_i, \vec{x}_k)}$$

$$\Downarrow$$

$$\vec{w} \cdot \phi(\vec{x}_i) = \sum_{j=1}^{t} \alpha_j \boxed{K_{dag}(\vec{dag}_{(j)}, \vec{x}_i)}$$

SDAG+

- Compacts **all** CPA models in the working set into a **single DAG**

$$\vec{w} \cdot \phi(\vec{x}_i) = \boxed{\sum_{j=1}^{t} \alpha_j \sum_{k=1}^{r} \left(\frac{1}{r} c_k^{(j)} y_k \right) K(\vec{x}_i, \vec{x}_k)}$$

SDAG+

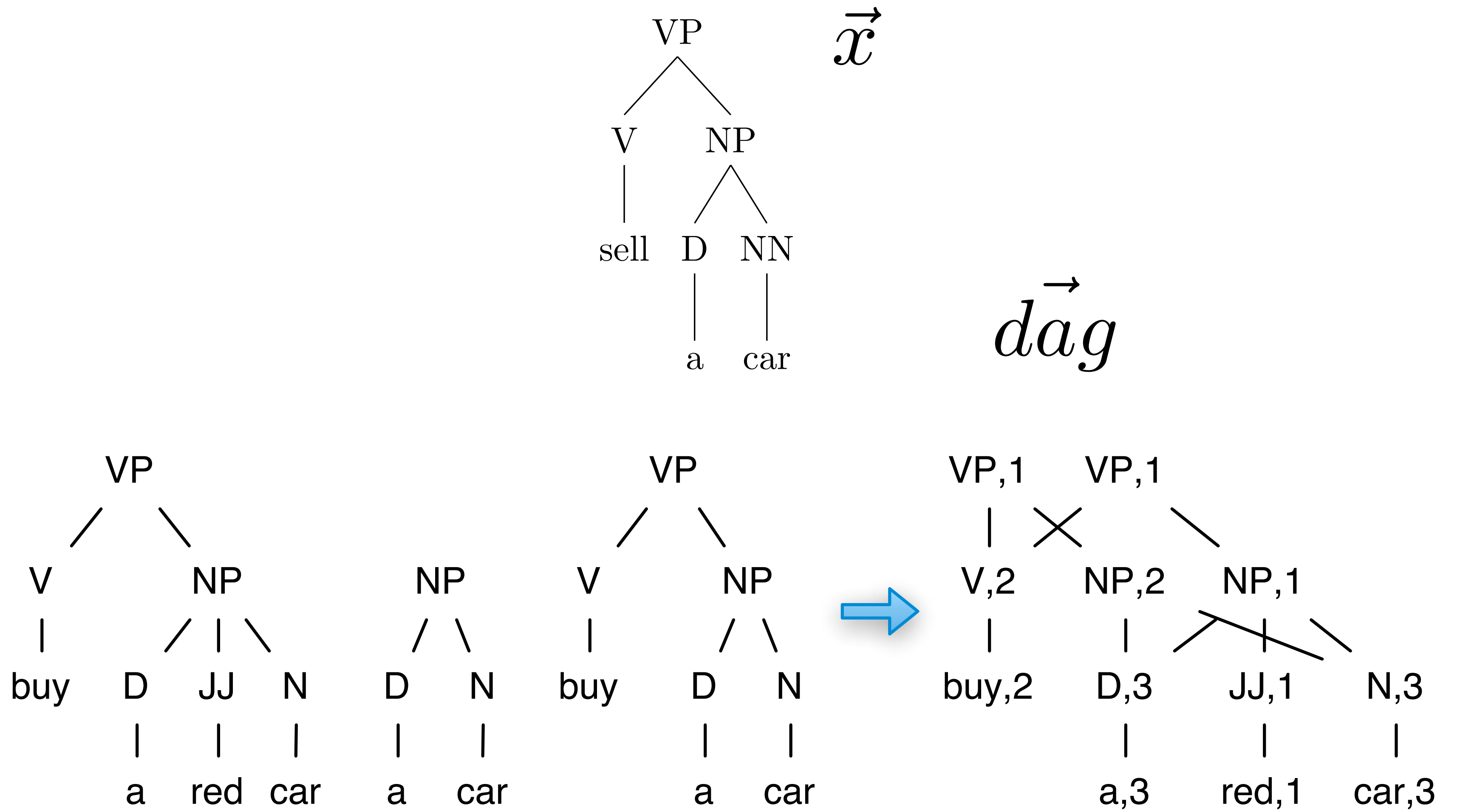
- Compacts **all** CPA models in the working set into a **single DAG**

$$\vec{w}\cdot\phi(\vec{x}_i) = \boxed{\sum_{j=1}^{t}\alpha_j\sum_{k=1}^{r}\left(\frac{1}{r}c_k^{(j)}y_k\right)K(\vec{x}_i,\vec{x}_k)}$$

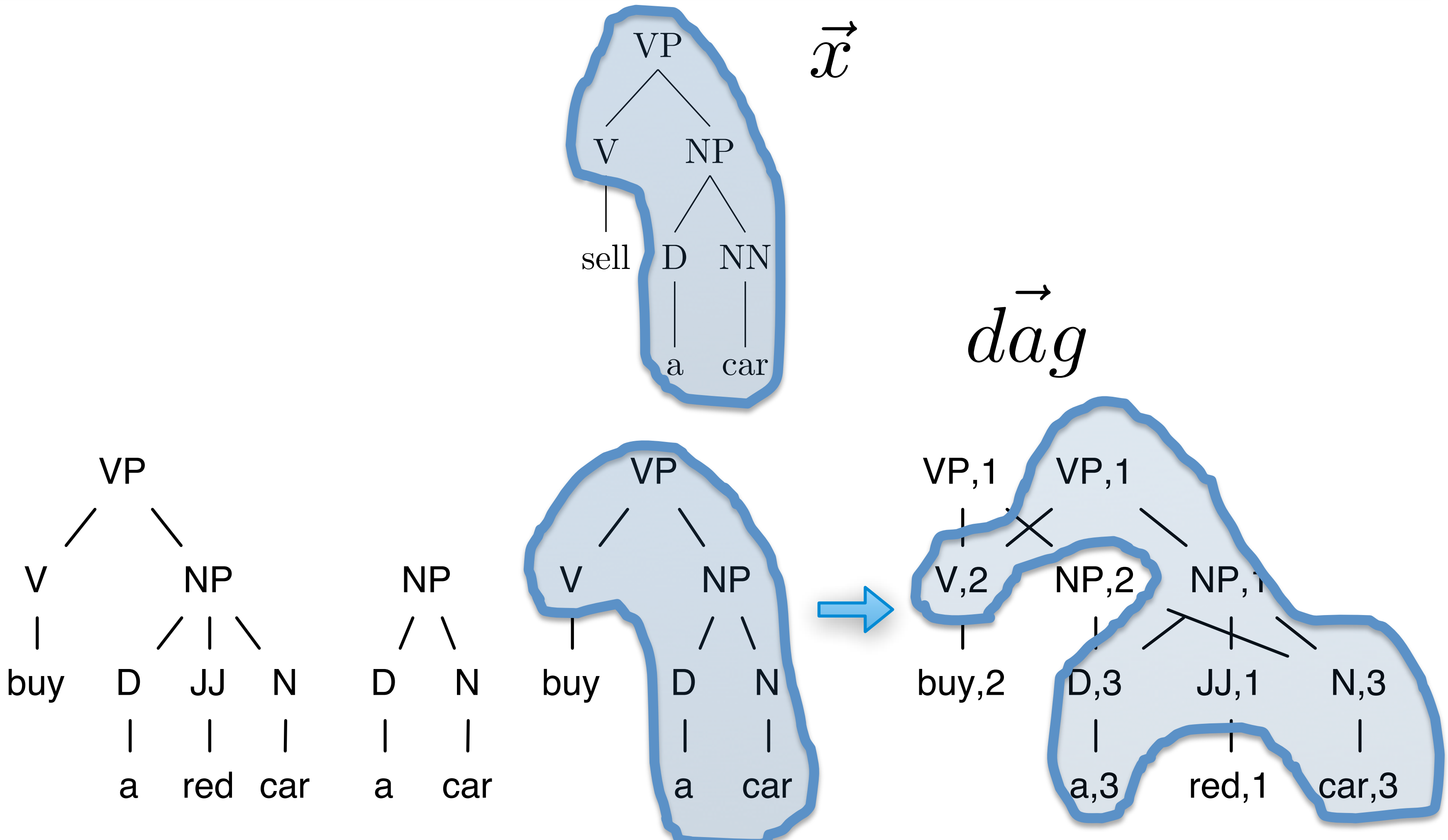
$$\Downarrow$$

$$\vec{w}\cdot\phi(\vec{x}_i) = \boxed{K_{dag}(\widehat{\vec{dag}_{(t)}},\vec{x}_i)}$$

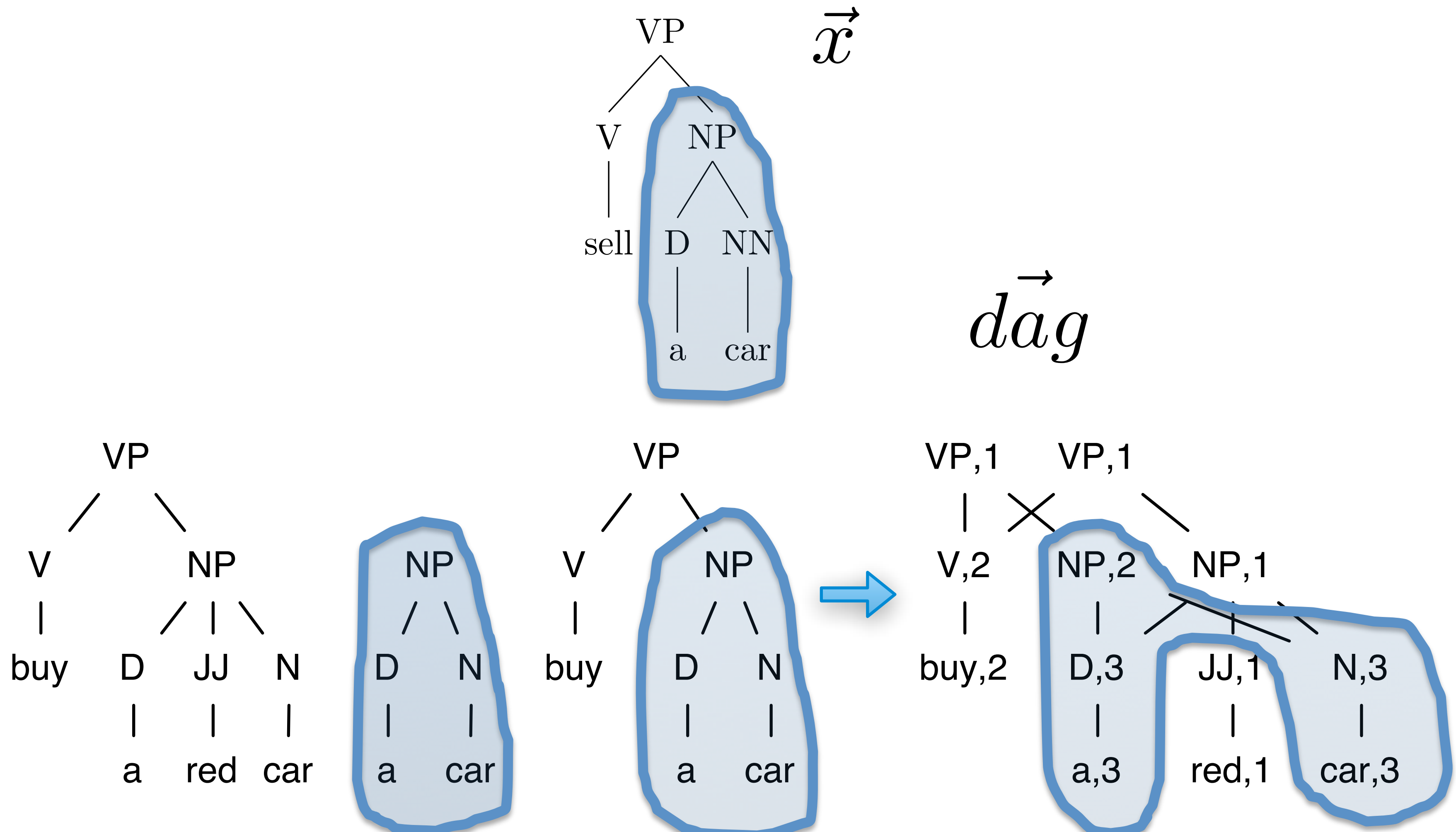
Example: computing $K_{dag}(\vec{dag}, \vec{x})$



Example: computing $K_{dag}(\vec{dag}, \vec{x})$



Example: computing $K_{dag}(\vec{dag}, \vec{x})$



Experimental setup

Datasets

1. Predicate-Argument Identification (Semantic Role Labeling)
2. Yahoo! Answers (Question/Answer Classification)
3. Question Classification from TREC-10

Semantic Role Labeling (SRL) dataset

- Task: identification of argument boundaries

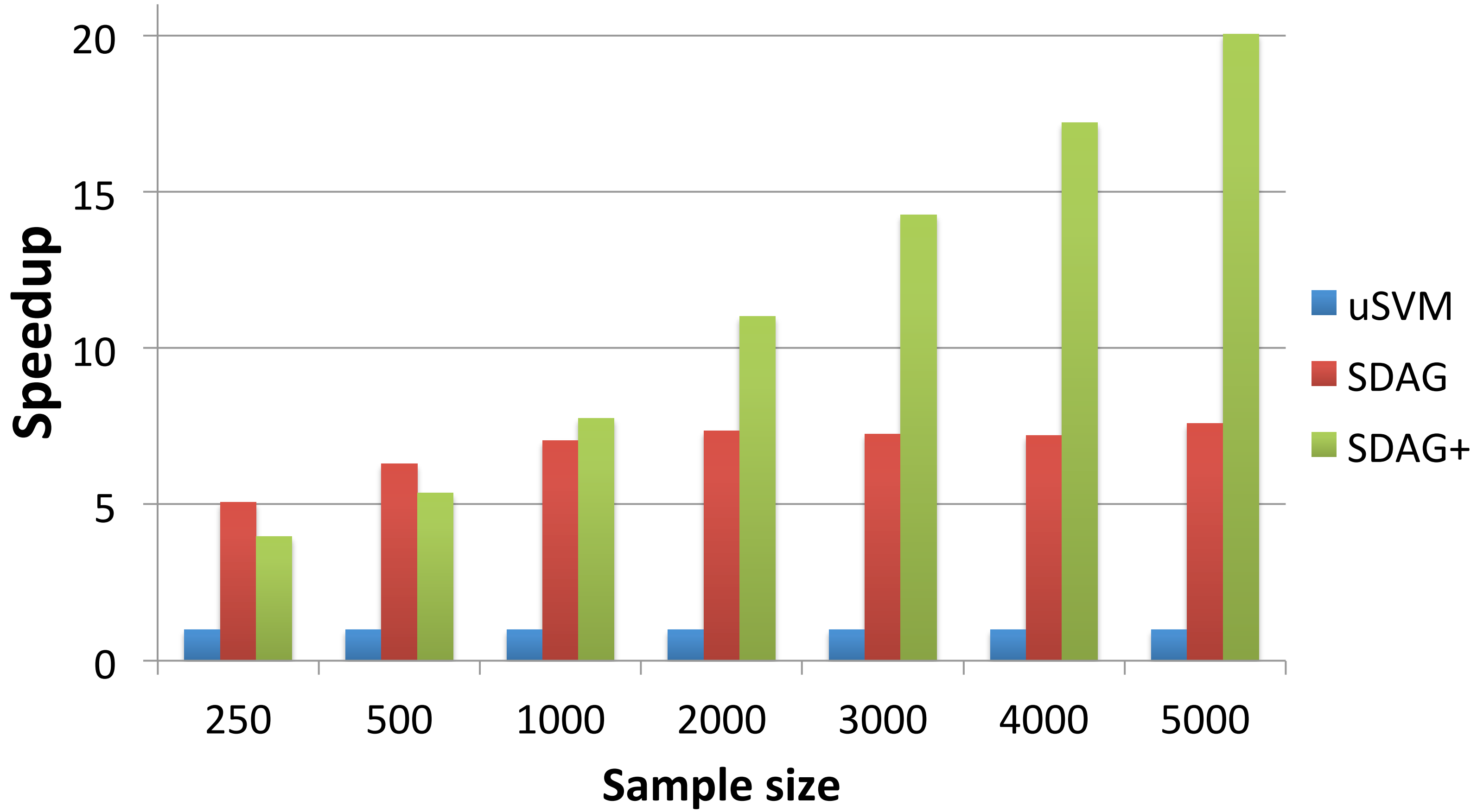
Example of SRL annotation:

Paul gives a talk in Rome

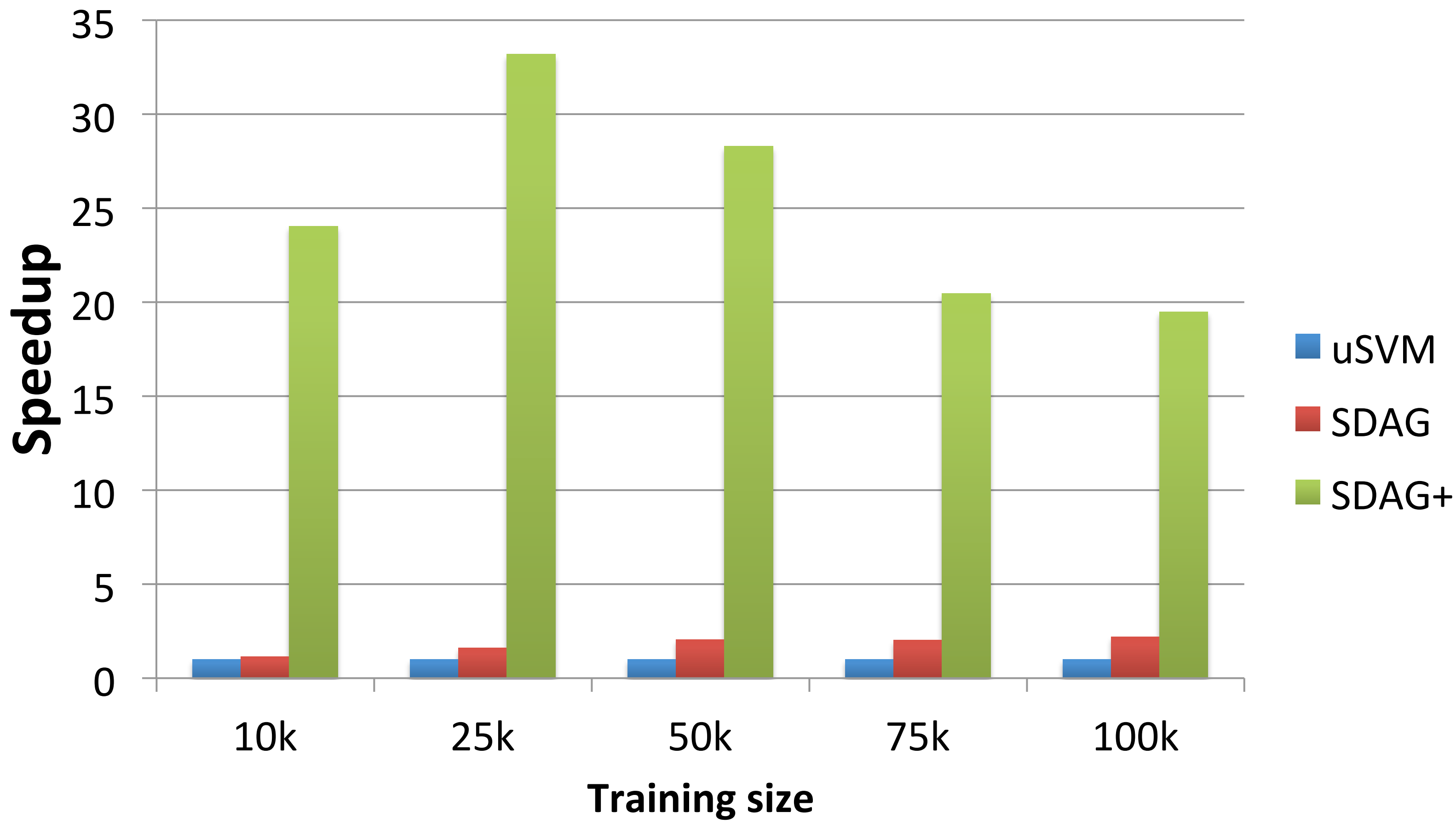
[$_{BD}$ Paul] [$_{target}$ gives] [$_{BD}$ a talk] [$_{BD}$ in Rome]

- Consists of PropBank, PennTree bank and Charniak parse trees as provided by CoNLL 2005
- Training set: 100,000
- Two Test sets:
  - Sections **23** and **24** (234,416 and 149,140 instances)

Speedups during training on SRL dataset (100k)



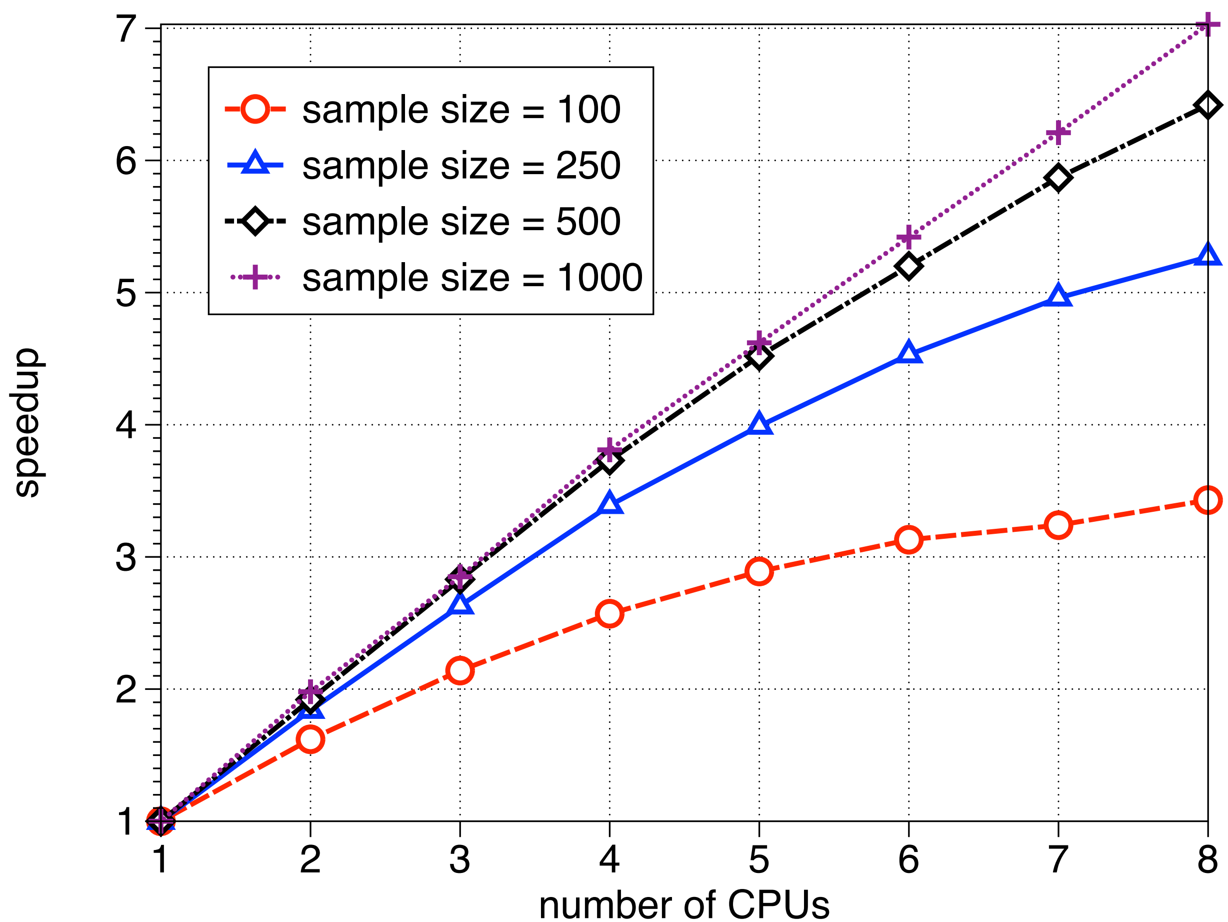
Classification speedups on SRL dataset (100k)



Parallelization

- **95% of time** is spent computing a cutting plane model at each iteration
- CPA allows for straight-forward **parallelization** to bring down complexity from $O(r^2)$ to $O(r^2/p)$

Speedups due to parallelization on 50k YA dataset



Handling class-imbalance problem

- 1-slack OP makes it **difficult** to include penalties for examples from different classes
- The idea of sampling to build approximate cutting plane at each iteration suggests a **straight- forward solution**
  - Use importance sampling
- **Preserves** theoretical convergence bounds

Results on QA classification: TREC and Yahoo! Answers

Our empirical findings reveal:

1. **Outperforms** approximate CPA when tuning is needed [Yu & Joachims, NIPS'08]
2. As **fast** as approximate CPAs [Severyn & Moschitti, ECML'10]
3. Gives a **more flexible control** over Precision/Recall than SVM-light

Conclusions

1. Two learning algorithms SDAG and SDAG+ that **compact** **CPA models** using **DAGs** to give much **faster** **training** and **classification** times
2. **Parallelization**
3. **Alternative sampling** to better handle **class-imbalanced data** at **large-scale**
4. **Solution for learning with structural kernels on large scale**

   Software will be available at
   http://projects.disi.unitn.it/iKernels/Severyn.html

Future work

- Extend the **DAG approach** to **more general kernels**, e.g. PT kernel [Moschitti, ECML 2006]
- Explore other **Structural Kernels**, e.g. graph kernels
- Other tasks, e.g. from NLP (relation extraction, co-reference resolution)
- Try alternative training algorithms, e.g. SGD, Pegasos

Thank you!

uSVM vs SDAG/SDAG+ on SRL dataset

TRAINING

SAMPLE	uSVM	SDAG		SDAG+	
1000	2196	312	(7.0)	283	(7.8)
2000	8282	1127	(7.3)	752	(11.0)
3000	18189	2509	(7.2)	1275	(14.3)
4000	31012	4306	(7.2)	1802	(17.2)
5000	50060	6591	(7.6)	2497	(20.0)

CLASSIFICATION

DATA	#SVs	uSVM	SDAG		SDAG+	
10K	1686	11	9	(1.1)	1	(24.0)
25K	3392	41	25	(1.6)	1	(33.2)
50K	5876	82	40	(2.1)	3	(28.3)
75K	7489	112	55	(2.0)	5	(20.5)
100K	8674	131	59	(2.2)	7	(19.5)

Handling class-imbalance on TREC-10 and YA datasets

TREC 10

DATA	RATIO	uSVM F-1	uSVM P/R	uSVM+J F-1	uSVM+J P/R	SVM F-1	SVM P/R
ABBR	1:60	87.5	100.0/77.8	84.2	80.0/88.9	84.2	80.0/88.9
DESC	1:4	96.1	95.0/97.1	96.1	95.0/97.1	94.8	97.7/92.0
ENTY	1:3	72.3	91.8/59.6	79.1	79.6/78.7	80.4	82.2/78.7
HUM	1:3	88.1	98.1/80.0	90.3	94.9/86.2	87.5	88.9/86.2
LOC	1:3	81.4	96.6/70.4	87.0	87.5/86.4	82.6	86.5/79.0
NUM	1:5	86.0	98.9/76.1	91.2	96.1/86.7	89.9	98.9/82.3

YAHOO ANSWERS

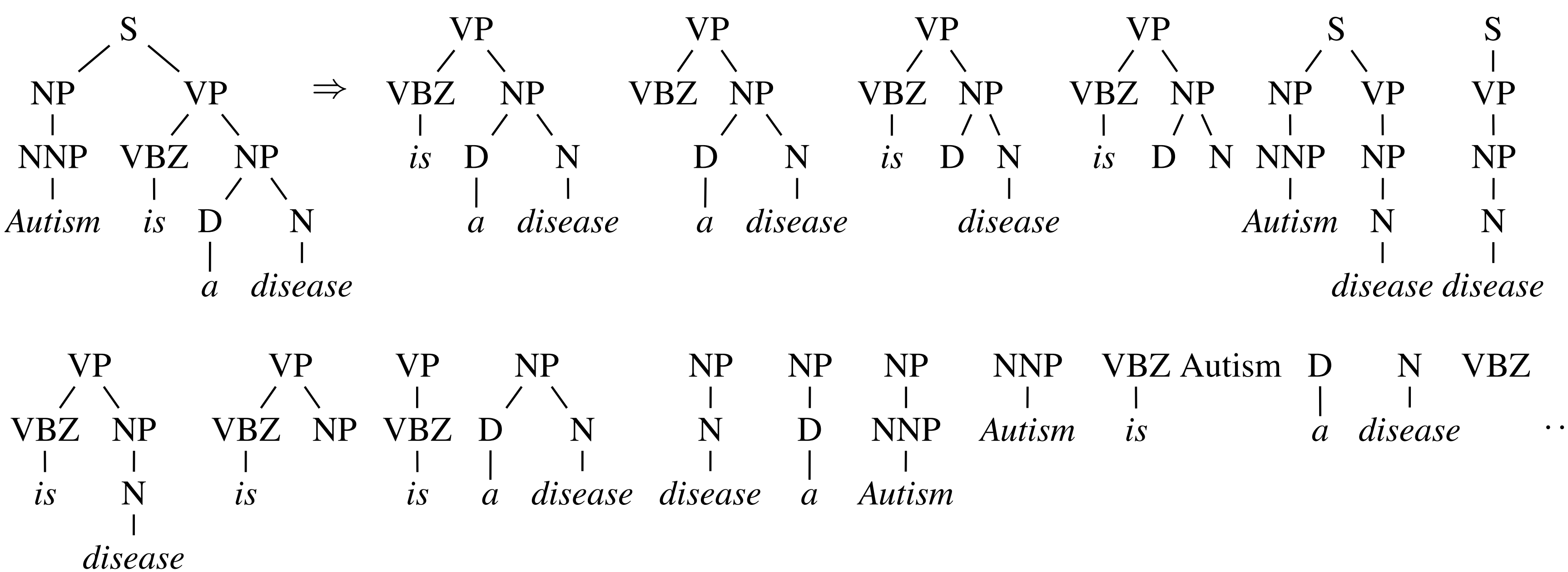
DATA	RATIO	uSVM F-1	uSVM P/R	uSVM+J F-1	uSVM+J P/R	SVM F-1	SVM P/R
10K	1:1.5	37.4	33.5/42.2	39.1	29.6/57.7	37.9	24.2/87.7
50K	1:2.0	36.5	36.0/36.9	40.6	30.0/62.5	39.6	25.7/86.9
100K	1:2.4	33.4	36.2/31.1	40.2	30.2/59.9	40.3	26.6/83.5
150K	1:2.8	33.5	36.9/30.7	41.0	30.2/64.0	-	-
300K	1:3.4	23.8	40.1/16.9	41.4	30.7/63.8	-	-
BOW	1:2.0	34.2	33.2/35.3	38.1	27.5/61.7	36.3	22.5/93.5

Background: Structural Kernels

- Given a parse tree of a sentence, they generate **exponentially many features**
- Help to avoid **manual feature engineering** for poorly understood linguistic phenomena
- Achieve **state-of-the-art results** in many NLP tasks, e.g. question answering, semantic role labeling, etc.
- Many levels of granularity to generate sub-trees, e.g. ST, SSK, PT

Some of the features generated by PT kernel

Autism is a disease



Questions…