splitSVM: Fast, Space-Efficient, non-Heuristic, Polynomial Kernel Computation for NLP Applications

Yoav Goldberg  Michael Elhadad

Ben-Gurion University of the Negev

ACL 2008, Columbus, Ohio
Support Vector Machines

- SVMs are supervised binary classifiers
- Max-margin linear classification
- Can perform non-linear classification by use of a kernel function

SVMs in NLP

- SVM classifiers are used in many NLP applications
- Such applications usually involve a great number of binary valued features
- Using \( d \)-th order polynomial kernel amounts to effectively consider all \( d \)-tuples of features
- Low-degree (2-3) Polynomial Kernels constantly produce state-of-the-art results
The Problem

Kernel-SVMs are slow!
- Computation of kernel-based classifier decision is expensive!
- Can grow linearly with size of training data.
- Non-kernel classifiers are orders of magnitude faster.

We are not talking about learning, we are talking about the decision for a given model.

Enter splitSVM

We propose a method for speeding up the computation of low-degree polynomial kernel classifiers for NLP applications, while still computing the exact decision function, and with a modest memory overhead.
Kernel Decision Function Computation

\[ y(x) = \text{sgn} \left( \sum_{x_j \in SV} y_j \alpha_j K(x_j, x) + b \right) \]

A Set of *Support Vectors*. Each support vector is a weighted instance from the training set. There typically are many such vectors.

In every classification, the *kernel function* must be computed for each *Support Vector*.
$$y(x) = sgn \left( \sum_{x_j \in SV} y_j \alpha_j (\gamma x \cdot x_j + c)^d + b \right)$$

The polynomial kernel of degree $d$

Proportional to the number of $d$-tuples of features the classified item and the $sv$ have in common.
Polynomial Kernel Speedup 1

\[ y(x) = \text{sgn}\left(\sum_{x_j \in SV} y_j \alpha_j (\gamma x \cdot x_j + c)^d + b\right) \]

**Speedup method 1 – PKI (Kudo and Matsumoto 2003)**

- Feature vectors are sparse
- If the classified item and an sv don’t share any features, we can skip the kernel computation for this sv
  \[ \Rightarrow \] Keep an inverted index of *feature* → sv, and use it to find only the relevant sv's for each item

**Problem: the Zipfian distribution of language**

- Language data has a Zipfian distribution
  \[ \Rightarrow \] There is a small number of very frequent features
    - W: ’a’, POS:NN, POS:VB
  \[ \Rightarrow \] PKI pruning does not remove many sv's . . .
Polynomial Kernel Speedup 2

\[ y(x) = sgn(w \cdot x^d + b) \]

**Speedup method 2 – Kernel Expansion** *(Isozaki and Kazawa, 2002)*

⇒ transform the \( d \)-degree polynomial classifier into a linear one in the kernel space

- At classification time: transform the instance to be classified into the \( d \)-tuple space, and perform linear classification (each weight in \( w \) corresponds to a specific \( d \) – tuple)

**Problem: the Zipfian distribution of language**

- Language data has a Zipfian distribution

⇒ There is a huge number of very infrequent features

- \( W:\text{calculation}, W:\text{polynomial}, W:\text{ACL} \)

⇒ The number of \( d \)-tuples is Huge!

- Storing \( w \) is impractical
Our Solution: splitSVM

This work: splitSVM

- Features have Zipfian distribution

⇒ Split the features into rare and common features
  - Perform PKI inverted indexing on the rare features
  - Perform Kernel Expansion on the common features
  - Combine the result into a single decision

- For the math, see the paper

Yoav Goldberg, Michael Elhadad
splitSVM: Fast SVM Decoder
We provide a Java implementation: splitSVM
We provide the same interface as common SVM packages (libsvm, yamcha)
In order to use splitSVM in your application:
  - Train a libsvm/svmlight/tinySVM/yamcha model as you did before
  - Convert the model to our splitSVM format
  - Change 2 lines in your code
A Testcase - Speeding up MaltParser

MaltParser (Nivre et.al., 2006)
- A state of the art dependency parser
- Java implementation is freely available
- Uses 2nd degree polynomial kernel for classification
- Uses libsvm as classification engine
- ...is a bit slow...

Enter splitSVM
- We use the pre-trained English models
- We replaced the libsvm classifier with splitSVM
- (Rare features: those in less than 0.5% of the SVs)
A Testcase - Speeding up MaltParser

<table>
<thead>
<tr>
<th>Method</th>
<th>Mem.</th>
<th>Parsing Time</th>
<th>Sents/Sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Libsvm</td>
<td>240MB</td>
<td>2166 (sec)</td>
<td>1.73</td>
</tr>
<tr>
<td>ThisPaper</td>
<td>750MB</td>
<td>70 (sec)</td>
<td>53</td>
</tr>
</tbody>
</table>

Table: Parsing Time for WSJ Sections 23-24 (3762 sentences), on Pentium M, 1.73GHz

- Only 3 fold memory increase
- ~ 30 times faster
- A Java-based parser parsing > 50 sentences / sec!
To Conclude

- Simple idea.
- Works great.
- Simple to use.
- Use it.

http://www.cs.bgu.ac.il/~nlpproj/splitsvm