The description of the kernel method pipeline and training

Domonkos Tikk, Philippe Thomas

September 25, 2009

Abstract

This text gives a detailed technical "how-to" description on the kernel based PPI classification experiments. The main aim is to provide a reconstructible description for possible further work. Most of the work presented here has been first done by Peter Palaga as a part of his master thesis. Here we document each step of Palaga's work and introduce some modifications and extensions. These experiments are the basis of our to-be-written kernel method comparison paper.

1 Download & installation

1.1 Corpora

Five corpora are used in the experiments. These are LLL, AImed, IEPA, HPRD50, and BioInfer. For more details see Palaga's work [5, p. 10]. We used the derived version of coprora created by Pyysalo's group, which contains parses created with the Charniak-Lease parser [2] and transformed to collapsed Stanford format [3]. The derived versions of corpora were downloaded from `http://mars.cs.utu.fi/PPICorpora/GraphKernel.html`.

Additionally, the train–test splits of the corpora were specified by and received from Antti Airola [1]. These splits currently resides on `racer` under `/local/for_palaga/learning/corpora/splits` and under `/vol/home-vol3/wbi/tikk/Kernels/corpora/splits`.

1.2 Preprocessing

The preprocessing pipeline created by Peter Palaga is available with wbi account from the SVN repository: `svn+ssh://<user>@gruenau.informatik.hu-berlin.de/vol/drakan-vol4/wbi/wbi/shared/x_consensus_patterns/learning-format-api/trunk`

The suggested developing environment is Eclipse, with Subclipse sub-versioning plug-in. For Windows machines the latter works only if in Eclipse menu: window → preferences → team → svn, the default SVN interface is changed from JAVAHL(JNI) to JavaSVN(Pure JAVA) (source: `http://www.svnforum.org/2017/viewtopic.php?p=3220&sid=1ad744ec80dab84d2dc3a9930471724f`).

1.3 Parsers

We experienced with 2 versions of the Charniak–Lease–Johnson–McColsky parser and with 3 model files.

Parsers:

- As a courtesy of Charniak's group, we could perform experiments with the pre-release: `http://cs.brown.edu/~dmcc/post/reranking-parser-feb09-pre.tar.gz`
- The standard version: `ftp://ftp.cs.brown.edu/pub/nlparser/reranking-parserAug06.tar.gz`. We used this as default.

Model files:

- News model file can be downloaded from `http://cs.brown.edu/~dmcc/selftraining/selftrained.tar.gz` (retrieved on 18 May 2009). This version was trained on WSJ and NANC data.
- Bio model file can be downloaded from `http://bllip.cs.brown.edu/download/orig-selftrained-bio-model.tar.gz` (retrieved on 25 September 2009). This version was trained on PubMed abstracts.

- Enhanced bio model file can be downloaded from `http://bllip.cs.brown.edu/download/orig-selftrained-bio-model.tar.gz` (retrieved on 25 September 2009). This version was trained on PubMed abstracts and with GENIA reranker and improves the performance of the above bio model significantly. We used this as default.

Theoretically, the parsers can be easily installed by decompressing the above files in a directory and running the simply the `make`. However, with the current gcc compiler versions (we tried with 4.3.1 and 4.4.0), the official release (2006 Aug version) cannot be compiled, some minor errors occur that were rectified by Peter Palaga, that version is available on `racer` under `/local/for_palaga/bin/reranking-parser` and on `/vol/home-vol3/wbi/tikk/Kernels/parsers/Aug2006reranking-parser` (for gcc 4.3.1).

Additionally, we tried to run the parsers under Windows with cygwin. Here we had also compilation errors for both releases, which were also rectified (see Appendix A.2 for details). However, the parsers behaved abnormally with even medium size inputs, and stop to work abruptly after having parsed a few sentences. Probably a memory leakage is present, which we could not identify. Therefore the entire pipeline is currently not executable on Windows machines.

For optimized parsing speed the environment variable `GCCFLAGS` has to be set appropriately. The following site helps in specifying the proper value of the variable: `http://en.gentoo-wiki.com/wiki/Safe_Cflags/Intel`.

1.4 Classifiers

Peter Palaga's work [5] contains experiments executed with 5 kernel based classifiers. All available with WBI account from the SVN repositories listed below.

- k-band shortest path spectrum (kBSPS) kernels, developed by Palaga. `svn+ssh://<user>@gruenau.informatik.hu-berlin.de/vol/drakan-vol4/wbi/wbi/shared/x_consensus_patterns/svm_light/trunk`
- Spectrum tree (SpT) kernel, developed by Tetsui Kuboyama [4], reimplemented by Palaga. `svn+ssh://<user>@gruenau.informatik.hu-berlin.de/vol/drakan-vol4/wbi/wbi/shared/x_consensus_patterns/svm_light/branches/svm_light_spectrum_tree_kernel`
- Subtree (ST), subset tree (SST), and partial tree (PT) kernels, developed by Moschitti (Moschitti-1.5-prerelease, not available yet online). `svn+ssh://<user>@gruenau.informatik.hu-berlin.de/vol/drakan-vol4/wbi/wbi/shared/x_consensus_patterns/SVM-Light-1.5-to-be-released/trunk`

The classifiers can be installed simply by calling `make`, and they work both on Linux and with cygwin on Windows.

2 Preprocessing

Preprocessing consists of several steps.

Step 1 Creates the appropriate input format from the original xml files for the parse tree parser.

Step 2 Performs parsing.

Step 3 Injects the parsing results into the original xml files.

Step 4 Aligns the original sentence with the parsing results (specifies the character offsets in the raw text and the parse trees).

Step 5 Injects the results of the above alignment into the working xml files.

Step 6a Creates 10 fold-training datasets in the learning format of the two SVM variants.

Step 6b Prepares cross-corpus validation datasets in the learning format of the two SVM variants.

We wrote a script that execute the above steps, see in Appendix A.1. Please note, that some variables should be set according your own settings for correct operation. For that see also the following detailed description.

2.1 Step 1: preparing parsing input

The software is a part of Palaga's Learning format API Java library (see Section 1.2). The program extracts each sentence from the xml file and embeds them within `<s>...</s>` tags.

- Program: `PtbRawSentenceTransformer.java`

Table 1: Experiments with different parser versions and models

Parser version/ model file	2006 Aug	2009 Feb (pre-release)
bio-optimized	Parsing error on 4 sentences, parsing process hangs up after error and is not finished	Parsing error on 4 sentences, but parsing process is continued
normal (news)	Parsing error on 1 sentence, parser hang up after error, parsing is not finished	Parsing errors on 1 sentence, but parsing process is continued

- Input: derived XML files from Pyysalo's group (see Section 1.1).
- Parameter: original xml file.
- Run from command line:

  ```
  java -classpath $ECLIPSEWORKSPACE/learning-format-api/bin
    org.learningformat.transform.PtbRawSentenceTransformer example.xml
  ```

  where `$ECLIPSEWORKSPACE` is the the workspace directory of Eclipse. For Windows, this is environmental variable is called by `%ECLIPSEWORKSPACE%`, and can be set by

  ```
  set ECLIPSEWORKSPACE=C:\myworkingspace
  ```

- Output: `example.xml-ptb-s.txt`

2.2 Step 2: parsing

The second step is the parsing the corpora with Charniak-Lease-Johnson-McClosky parser (see Section 1.3). The input format is done in Step 1 (`example.xml-ptb-s.txt` file) the output is the parsed trees of the each sentence, e.g., the result of "*ykuD was transcribed by SigK RNA polymerase from T4 of sporulation.*" is

```
(S1 (S (NP (NNP ykuD)) (VP (AUX was) (VP (VBN transcribed)
(PP (IN by) (NP (NP (NNP SigK) (NNP RNA) (NN polymerase))
(PP (IN from) (NP (NNP T4))) (PP (IN of) (NP (NN sporulation))))))) (. .)))
```

- Software: Charniak–Lease–Johnson–McClosky parser executables: `parseIt` and `bestparses`, both called from the script file `parse.sh`.
- Input: `example.xml-ptb-s.txt` files in format `<s>sentence</s>`.
- Parameter: to-be-parsed file (`ptb-s.txt`), output and error file should be redirected
- Run from command line:

  ```
  ./parse.sh example.xml-ptb-s.txt > example.xml-ptb-s.txt-parsed.txt 2>example.err
  ```

  where `parse.sh` is the starting script of the parser. The command is then started in the root directory of the parser.

- Output: `example.xml-ptb-s.txt-parsed.txt`

We experimented with all the four combinations of the 2 parser versions and the 2 model files. Results are summarized in Table 1. Based on these experiments, we decided to use the version of 2009 February. TODO: model files: should be decided based on classification results. Most probably bio-optimized version is better.

2.3 Step 3: Injection of parsing results into xml files

The software is a part of Palaga's Learning format API Java library (see Section 1.2). This program injects into the original xml files the parsing results in `<bracketings> ... </bracketings>` tags.

- Program: `PtbTreeInjector.java`

- Input: `example.xml` and the corresponding `example.xml-ptb-s.txt-parsed.txt`, the latter should be placed in a subdirectory `charniak-johnson` of the directory of `example.xml`.
- Parameters:

  `-f|--file` input file name

  `-i|--inject` this is a switch which should be given for this step

  `-o|--out` output file name
- Running from command line:

```
java -classpath $ECLIPSEWORKSPACE/learning-format-api/bin;
$ECLIPSEWORKSPACE/learning-format-api/src/main/resources/jargs.jar
org.learningformat.transform.PtbTreeInjector
-f example.xml -i -o trees/example.xml
```

  By convention, the output xml file is placed into the directory `trees` under the root directory of `example.xml`.
- Output: specified by the `-o` option, an xml file that contains now the parsing results in `<bracketings>...</bracketings>` tags.

2.4 Step 4: alignments of the original sentence with the parsing results

The software is a part of Palaga's Learning format API Java library (see Section 1.2). This program aligns the original sentence with the parsing results, that is it specifies the character offsets of the (from-to) of the tokens of the original text in the parsed texts. For the example sentence "*ykuD was transcribed by SigK RNA polymerase from T4 of sporulation.*" it is:

```
LLL.d2.s1,0-3:16-19,5-7:32-34,9-19:46-56,21-22:67-68,24-27:84-87,29-31:95-97,
33-42:104-113,44-47:125-128,49-50:140-141,52-53:154-155,55-65:166-176,66-66:188-188
```

- Program: `BracketingTokenMapper.java`
- Input: output file of step 4, that is the xml files located in the folder `trees` (by convention).
- Parameter: `trees/example.xml`
- Running from command line:

```
java -classpath $ECLIPSEWORKSPACE/learning-format-api/bin
org.learningformat.transform.BracketingTokenMapper trees/example.xml
```

- Output: a text file containing the alignments. The output text files gets the `-bracketing-tokens.txt` suffix, i.e. for `example.xml` it is `example.xml-bracketing-tokens.txt` and placed in the directory `trees`.

2.5 Step 5: Injection of sentence–parsing alignments into xml files

The software is a part of Palaga's Learning format API Java library (see Section 1.2). This program injects into the original xml files the alignments created in step 4, the alignments are injected into the xml file within `<charOffsetMapEntry>` tags.

- Program: `PtbTreeInjector.java`
- Input: `example.xml` and the corresponding `example.xml-bracketing-tokens.txt`
- Parameters:

  `-f|--file` input file name

  `-i|--inject` this is a switch must not given for this step

  `-o|--out` output file name
- Running from command line:

```
java -classpath $ECLIPSEWORKSPACE/learning-format-api/bin;
$ECLIPSEWORKSPACE/learning-format-api/src/main/resources/jargs.jar
org.learningformat.transform.PtbTreeInjector
-f trees/example.xml -o mapped-trees/example.xml
```

  By convention, the output xml file is placed into the directory `mapped-trees` under the root directory of `example.xml`.
- Output: specified by the `-o` option, an xml file that contains now the parsing results in `<charOffsetMapEntry>` tags.

2.6 Step 6a: Creating folds for the training data set

This step is performed by two programs. They are part of Palaga's Learning format API Java library (see Section 1.2). These programs create the training format for SVM based classifier, therefore are the last step in the preprocessing pipeline. The programs assume the availability of test-train splits (see Section 1.1).

The first program creates the training data for the most of the SVM learners. The second program creates the training data for the k-band shortest path spectrum kernel based SVM learner.

- Program: SvmLightTreeKernelTransformer.java
- Input: the xml augmented with bracketing and alignment information, by convention located in the `mapped-trees` directory.
- Parameters:

  `-f|--file` input file name

  `-o|--out` output directory name

  `-s|--split` location of the split files (folder)

  `-m|--moschitti` flag for Moschitti style learning format (for subtree (ST), subset tree (SST), and partial tree (PT) kernels)

  `-c|--custom` flag for custom learning format (for spectrum tree (SpT) kernel)

  Exactly one of the `-m` and `-c` flags has to be given.
- Running from command line:

  ```
  java -classpath $ECLIPSEWORKSPACE/learning-format-api/bin;
  $ECLIPSEWORKSPACE/learning-format-api/src/main/resources/jargs.jar
  org.learningformat.transform.SvmLightTreeKernelTransformer
  -f example.xml -m -s splits -o trainingdir;
  ```

  where under the subdirectory `splits` resides the `example` directory, which includes the corresponding splits for the `example.xml`
- Output: under the directory specified by `-o` another directory named `MOSCHITTI` (for `-m` flag) and `CUSTOM_KERNEL` (for `-c` flag), under which a third directory `example.xml-folds` created, which contains 10 text files (named: 0.txt ... 9.txt). There is only a slight difference between the two format (Moschitti and custom), namely that the Moschitti based SVM learners require an embedding `|BT|..|ET|` tags around the training parsed tree instances, while the custom format follows the requirement of T. Joachims' svn-light.
- Program: SvmLightDependencyTreeKernelTransformer.java
- Input: the xml augmented with bracketing and alignment information, by convention located in the `mapped-trees` directory.
- Parameters:

  `-f|--file` input file name

  `-o|--out` output directory name

  `-s|--split` location of the split files (folder)
- Running from command line:

  ```
  java -classpath $ECLIPSEWORKSPACE/learning-format-api/bin;
  $ECLIPSEWORKSPACE/learning-format-api/src/main/resources/jargs.jar
  org.learningformat.transform.SvmLightDependencyTreeKernelTransformer
  -f example.xml -s splits -o trainingdir;
  ```

  where under the subdirectory `splits` resides the `example` directory, which includes the corresponding splits for the `example.xml`
- Output: under the directory specified by `-o` several other directories are created, under each of which another directory `example.xml-folds` created, which contains 10 text files (named: 0.txt ... 9.txt). The name of second level directories follow the following pattern: `CUSTOM_KERNEL-b-`x`to`y`-k`z`-UP_TO-OUTSIDE-NO_SELF_REF-STEM-none` where $x \leq y$ and $x \in \{1, 2\}$, $y \in \{1, 2, 3\}$ and $z \in \{0, 1\}$ and all possible combination is created. These numbers corresponds to the various parameters of the kBSPS kernels.

Remark: unlike in the case of the above program, here strings are converted into numbers in order to speed up the learning process of the SVM learner.

2.7 Step 6b: Creating folds for the training data set with cross-corpus validation

This step is performed by also two programs and they are very similar to the above Step 6a. They are part of Palaga's Learning format API Java library (see Section 1.2). These programs create the training format for SVM based classifier, therefore are the last step in the preprocessing pipeline in case cross-corpus cross validation is to be performed. The programs again assume the availability of test-train splits, which is in this case different from the above one (see Section 1.1).

The first program creates the training data for the most of the SVM learners. The second program creates the training data for the k-band shortest path spectrum kernel based SVM learner.

- Program: SvmLightTreeKernelTransformerCXval.java
- Input: the xml augmented with bracketing and alignment information, by convention located in the `tree/cross-corpus` directory. This xml file is the concatenation of all corpus files. We provide it in our package for the ease of use.
- Parameters:

  `-f|--file` input file name

  `-o|--out` output directory name

  `-s|--split` location of the split files (folder)

  `-t|--test` name of the corpus that is used for test, the other corpora are used for training.

  `-m|--moschitti` flag for Moschitti style learning format (for subtree (ST), subset tree (SST), and partial tree (PT) kernels)

  `-c|--custom` flag for custom learning format (for spectrum tree (SpT) kernel)

  Exactly one of the `-m` and `-c` flags has to be given.
- Running from command line:

```
java -classpath $ECLIPSEWORKSPACE/learning-format-api/bin;
$ECLIPSEWORKSPACE/learning-format-api/src/main/resources/jargs.jar
org.learningformat.transform.SvmLightTreeKernelTransformerCXval
-f All.xml -t example -m -s splits -o trainingdir;
```

  where under the subdirectory `splits` resides the `Test_on_example` directory, which includes the corresponding splits: `example.xml` and `All_but_example.xml`
- Output: under the directory specified by `-o` another directory named `MOSCHITTI` (for `-m` flag) and `CUSTOM_KERNEL` (for `-c` flag), under which a third directory `example-folds` created, which contains 2 text files: `0.txt` is test set and `1.txt` is the training set. There is only a slight difference between the two format (Moschitti and custom), namely that the Moschitti based SVM learners require an embedding `|BT|..|ET|` tags around the training parsed tree instances, while the custom format follows the requirement of T. Joachims' svn-light.
- Program: SvmLightDependencyTreeKernelTransformerCXval.java
- Input: the xml augmented with bracketing and alignment information, by convention located in the `tree/cross-corpus` directory. This xml file is the concatenation of all corpus files. We provide it in our package for the ease of use.
- Parameters:

  `-f|--file` input file name

  `-o|--out` output directory name

  `-s|--split` location of the split files (folder)

  `-t|--test` name of the corpus that is used for test, the other corpora are used for training.
- Running from command line:

```
java -classpath $ECLIPSEWORKSPACE/learning-format-api/bin;
$ECLIPSEWORKSPACE/learning-format-api/src/main/resources/jargs.jar
org.learningformat.transform.SvmLightDependencyTreeKernelTransformerCXval
-f All.xml -t example -s splits -o trainingdir;
```

  where under the subdirectory `splits` resides the `Test_on_example` directory, which includes the corresponding splits: `example.xml` and `All_but_example.xml`

- Output: under the directory specified by `-o` several other directories are created, under each of which another directory `example-folds` created, 2 text files: `0.txt` is test set and `1.txt` is the training set. The name of second level directories follow the following pattern: `CUSTOM_KERNEL-b-`x`to`y`-k`z`-UP_TO-OUTSIDE-NO_SELF_REF-STEM-none` where $x \leq y$ and $x \in \{1, 2\}$, $y \in \{1, 2, 3\}$ and $z \in \{0, 1\}$ and all possible combination is created. These numbers corresponds to the various parameters of the kBSPS kernels.

Remark: unlike in the case of the above program, here strings are converted into numbers in order to speed up the learning process of the SVM learner.

3 Running the SVM classifiers

The SVM learners are trained via scripts files:

kBSPS kernel the running script can be found on `gruenau2` at `/home/wbi/palaga/learning/experiments/090318-signed-LED-led/run`. When running in your on directory (at least) the followings should be set:

- `runHome` is the directory where the executable files of the learner reside.
- `corpusHome` is the the directory where the final output of the pipeline is located.

Caution: the script starts a lengthy learning session with an exhaustive parameter optimization, therefore it can take several hours or days to complete, depending on the machine you are running the experiments on.

SpT kernel the running script can be found on `gruenau2` at `/home/wbi/palaga/learning/spectrum/090328-spectrum-lll/run`. When running in your on directory (at least) the followings should be set:

- `runHome` is the directory where the executable files of the learner reside.
- `foldsDir` is the directory where the output of `SvmLightTreeKernelTransformer` resides (`CUSTOM_KERNEL` subdirectory).

ST, SST, PT the running script can be found on `gruenau2` at `/home/wbi/palaga/learning/moschitti/090327-ecml06-lll/run`. When running in your on directory (at least) the followings should be set:

- `runHome` is the directory where the executable files of the learner reside.
- `foldsDir` is the directory where the output of `SvmLightTreeKernelTransformer` resides (`MOSCHITTI` subdirectory).
- the set `kernels` specifies which kernel to apply: 0 for ST, 1 for SST, 3 for PT. For all let `kernels='0 1 3'`

The scripts produce the results in the `out/eval.txt` as database insert and update commands. Therefore in order to browse the results, one should upload them into a database. Before doing that a minor change should be done:

```
sed -ie 's/0::boolean/false/g' eval.txt
```

TODO: change this in the scripts already!

4 Browsing the results in the Database

The results of the scripts are PostgreSQL database command. The corresponding database can be found on `siegfried`. The following steps should be done to upload

- login into `siegfried` (from `paprika` or `racer`)
- login into the corresponding database, `ppi`, `ppi_test` or `ppi_production` by typing

  ```
  psql -U <username> <databasename>
  ```

- create the necessary table and views. See an example for the creation in Appendix A.3
- execute

  ```
  \i eval.txt
  ```

  here the pathname of `eval.txt` also should be given.

After that the results can be browsed in the database. For the attributes of the database see also Appendix A.3.

References

[1] A. Airola, S. Pyysalo, J. Björne, T. Pahikkala, F. Ginter, and T. Salakoski. All-paths graph kernel for protein-protein interaction extraction with evaluation of cross-corpus learning. *BMC bioinformatics*, 9(Suppl 11):S2, 2008.

[2] E. Charniak and M. Lease. Parsing biomedical literature. In *Proceedings of IJ CNLP'05*, 2005.

[3] M. C. de Marneffe, B. MacCartney, and C. D. Manning. Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC'06*, 2006.

[4] T. Kuboyama, K. Hirata, H. Kashima, K. Aoki-Kinoshita, and H. Yasuda. A spectrum tree kernel. *Information and Media Technologies*, 2(1):292–299, 2007.

[5] P. Palaga. Extracting relations from biomedical texts using syntactic information. Master's thesis, Humboldt-Universität zu Berlin, May 2009.

A Appendix

A.1 The preprocessing script

```
#0.) Download all 5 corpora into folder original
wget http://mars.cs.utu.fi/PPICorpora/AImed-learning-format.xml.gz
wget http://mars.cs.utu.fi/PPICorpora/BioInfer-learning-format.xml.gz
wget http://mars.cs.utu.fi/PPICorpora/HPRD50-learning-format.xml.gz
wget http://mars.cs.utu.fi/PPICorpora/IEPA-learning-format.xml.gz
wget http://mars.cs.utu.fi/PPICorpora/LLL-learning-format.xml.gz

gunzip *.gz

#For das File ein $PWD/$file
#1)
#Extracts sentences from the original XML
#Result is written in $file-ptb-s.txt
for file in *.xml
do
    java -classpath $HOME/workspace/learning-format-api/bin/ org.learningformat.transform.PtbRawSentenceTransformer $fi
done

#2.) Parsing
#Results in two files -parsed.txt and -parsed.err
for file in *-ptb-s.txt;
do
    ~/Desktop/svm/training/reranking-parser/parse.sh $file > $file-parsed.txt 2>$file-parsed.err;
done

#Also exectuted it on racer:
#To check if the results are similar between the two parses
#/vol/home-vol3/wbi/thomas/tmp/parsingTest/original

mkdir charniak-johnson
mv *-parsed.* charniak-johnson/
mkdir trees


#3.)Combine XML and PTB together
# Input is the original XML
# Requires the  corresponding file in directory "charniak-johnson" with file extension  "-ptb-s.txt-parsed.txt"
#Writes into trees
for file in $PWD/*.xml;
do
    java  -classpath $HOME/workspace/learning-format-api/bin/:$HOME/workspace/learning-format-api/lib/jargs.jar org.lea
done

#4.)Generates a mapping between POS and ??real text??
# Input is the XML-File in folder "trees"
# Writes output in the folder "trees" suffix "-bracketing-tokens.txt" in folder
for file in $PWD/trees/*.xml;
do
    java  -classpath /home/philippe/workspace/learning-format-api/bin/ org.learningformat.transform.BracketingTokenMapp
done
```

```
# 5.) Needs folder mapped trees,  because it stores results there
#Important InjectTreees=FALSE
#needs parametrization

mkdir trees/mapped-trees
for file in $PWD/trees/*.xml;
do
    java  -classpath /home/philippe/workspace/learning-format-api/bin/:$HOME/workspace/learning-format-api/lib/jargs.ja
done



#6.) Create final splits
#Needs final XML-Files in folder "trees/mapped-trees"
#Result is written into folder "$PWD/Corpora/" and then Moschitti or Custom
mkdir corpora


for file in $PWD/trees/mapped-trees/*.xml;
do
    java -classpath /home/philippe/workspace/learning-format-api/bin/:$HOME/workspace/learning-format-api/lib/jargs.jar
done
```

A.2 Parser fixes with gcc 4.4.0

The following steps need to be performed (tested on Windows under cygwin)
For the 2009 February pre-release version:

1. add into `first-stage/PARSE/utils.C`

   ```
   #include <cstdio>
   ```

2. add into `second-stage/programs/features/lexical_cast.h` and `second-stage/programs/prepare-data/lexical_cast.h`

   ```
   namespace std {
   typedef std::basic_string <wchar_t> wstring;
   };
   ```

3. add into `second-stage/programs/features/best-parses.cc`

   ```
   #include <unistd.h>
   ```

4. Additionally, the program flex is also needed for compilation (can be simply downloaded from cygwin).

For the 2006 August version:

1. add into `first-stage/PARSE/BchartSm.C`

   ```
   #include <stdlib.h>
   #include <stdio.h>
   #include <string.h>
   ```

2. add into `popen.h` (2 files)

   ```
   #include <string.h> //actually this is a change from <string>
   #include <stdio.h>
   ```

3. add into `second-stage/programs/features/lexical_cast.h` and `second-stage/programs/prepare-data/lexical_cast.h`

   ```
       namespace std {
   typedef std::basic_string <wchar_t> wstring;
   };
   ```

4. add into `second-stage/programs/features/best-parses.cc`

   ```
   #include <unistd.h>
   ```

5. add into `second-stage/programs/features/utility.h` and `second-stage/programs/prepare-data/utility.h`

   ```
   #include <memory>
   ```

   change lines 497, 509 from

   ```
   return is >> cp+1;
   ```

   to

   ```
   return is >> (cp+1);
   ```

6. add into `first-stage/PARSE/ECArgs.C`

   ```
   #include <algorithm>
   ```

7. add into `first-stage/PARSE/Feature.C`, `first-stage/PARSE/FeatureTree.C`, `first-stage/PARSE/InputTree.C`, `first-stage/PARSE/Params.C`, and `first-stage/PARSE/ParseIt.C`

   ```
   #include <stdlib.h>
   ```

8. add into `first-stage/PARSE/Params.C`

   ```
   #include <stdio.h>
   #include <string.h>
   ```

9. add into `first-stage/PARSE/utils.C`

   ```
   #include <cstdio>
   ```

10. change in `second-stage/programs/features/sstring.h` in lines 158, 160, 162

    ```
    basic_sstring
    ```

    to

    ```
    basic_sstring_
    ```

11. add into `second-stage/programs/prepare-data/lexical_cast.h`

    ```
    #include <limits>
    ```

A.3 Creating the table and views

This file can be found on `/vol/home-vol3/wbi/tikk/Kernels/experiments/init-db-exb`.

```
CREATE TABLE exb ( exbid serial NOT NULL, corpus text, kernel
integer, c double precision, j double precision, lmax integer,
lmin integer, k integer, match_ text, fold integer, normalized
boolean, input_format text, tp integer, fn integer, tn integer,
fp integer, total integer, auc double precision, precision_
double precision, recall double precision, f_measure double
precision, learn_sec double precision, classify_sec double
precision, kernel_script text, sv_num integer, led text,
CONSTRAINT exb_pkey PRIMARY KEY (exbid) );

CREATE OR REPLACE VIEW fold AS SELECT exb.corpus, exb.kernel,
exb.c, exb.j, exb.lmax, exb.lmin, exb.k, exb.match_, exb.normalized,
exb.input_format, exb.kernel_script, avg(exb.auc) AS auc,
avg(exb.precision_) AS precision_, avg(exb.recall) AS recall,
avg(exb.f_measure) AS f_measure, avg(exb.learn_sec) AS learn_sec,
avg(exb.classify_sec) AS classify_sec, avg(exb.sv_num) AS sv_num,
count(exb.exbid) AS cnt, exb.led FROM exb GROUP BY exb.corpus,
exb.kernel, exb.c, exb.j, exb.lmax, exb.lmin, exb.k, exb.match_,
exb.normalized, exb.input_format, exb.kernel_script, exb.led
ORDER BY exb.corpus, avg(exb.auc) DESC, exb.c, exb.j, exb.kernel_script;

CREATE OR REPLACE VIEW fold_top_6 AS ((( SELECT fold.corpus, fold.c,
fold.j, fold.kernel_script, fold.auc, fold.precision_, fold.recall,
fold.f_measure, fold.learn_sec, fold.classify_sec, fold.sv_num, fold.cnt,
fold.led FROM fold WHERE fold.corpus = 'AImed'::text LIMIT 6) UNION ALL
( SELECT fold.corpus, fold.c, fold.j, fold.kernel_script, fold.auc,
fold.precision_, fold.recall, fold.f_measure, fold.learn_sec,
fold.classify_sec, fold.sv_num, fold.cnt, fold.led FROM fold WHERE
```

```
fold.corpus = 'HPRD50'::text LIMIT 6)) UNION ALL ( SELECT fold.corpus,
fold.c, fold.j, fold.kernel_script, fold.auc, fold.precision_, fold.recall,
fold.f_measure, fold.learn_sec, fold.classify_sec, fold.sv_num, fold.cnt,
fold.led FROM fold WHERE fold.corpus = 'IEPA'::text LIMIT 6)) UNION ALL
( SELECT fold.corpus, fold.c, fold.j, fold.kernel_script, fold.auc,
fold.precision_, fold.recall, fold.f_measure, fold.learn_sec,
fold.classify_sec, fold.sv_num, fold.cnt, fold.led FROM fold WHERE
fold.corpus = 'LLL'::text LIMIT 6);
```