SEMPRE Sentiment Classification with GATE

Version 1.00

Bernhard Jung bernhard.jung@ofai.at

Hannes Pirker hannes.pirker@ofai.at

Austrian Research Institute for Artificial Intelligence (OFAI)
Freyung 6/6
1010 Vienna, Austria

1 Introduction

In the SEMPRE project, sentiment classification is used to derive subjective scores of movie and music reviews for the purpose of supporting a recommender system. This specific report describes the work on setting up a process for performing all kinds of preprocessing of these reviews and for learning and evaluating classifiers.

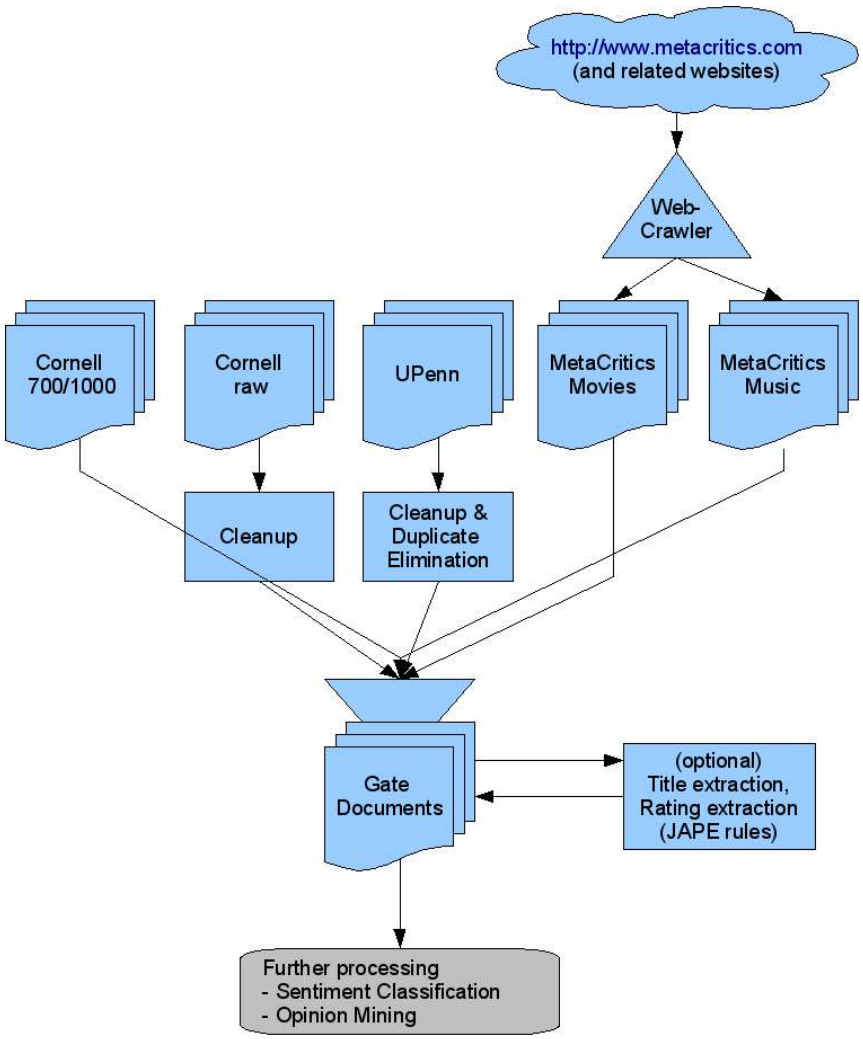
Our initial goal was to reproduce the results from Pang & Lee (2002, 2004) and then to extend and improve their approach by the use of linguistic knowledge as well as evaluate these approaches on other sentiment corpora.

The following describes the corpora and preprocessing steps we use, the handling of GATE (http://gate.ac.uk/) via jython (http://www.jython.org/) for scripting text processing pipelines, the configuration of the learning systems (built-in GATE learning API or external WEKA http://www.cs.waikato.ac.nz/ml/weka/).

2 Data Setup

Before classifiers can be learnt, we have to obtain corpora and apply specific preprocessing steps to them. The following diagram shows the 5 different types of corpora used, the individual steps applied to them (i.e. cleanup for cornell raw or duplicate elimination for upenn) and the central store as GATE documents which are then used for further processing and learning of classifiers within the GATE framework.

The following figure shows the corpora used and the processing/conversion steps that lead to a shared store of GATE documents that can be processed with the same tools.



2.1 Corpora

The used corpora are either freely available on the web or have been crawled within this project. The raw corpora used are:

- **Cornell datasets** from Pang & Lee available at http://www.cs.cornell.edu/People/pabo/movie-review-data/
    - Polarity dataset v1.0 (700 positive and 700 negative reviews)
      http://www.cs.cornell.edu/people/pabo/movie-review-data/mix20_rand700_tokens_cleaned.zip
      referred to as Cornell700
    - Polarity dataset v2.0 (1000 positive and 1000 negative reviews)
      http://www.cs.cornell.edu/People/pabo/movie-review-data/review_polarity.tar.gz
      referred to as Cornell1000
    - Pool of 27886 unprocessed html files
      http://www.cs.cornell.edu/People/pabo/movie-review-data/polarity_html.zip
      referred to as CornellRaw

  The Cornell corpora capture the movie domain. The have been generated from IMDB newsgroup reviews.
  Cornell700 and Cornell1000 are corpora that are already preprocessed, i.e. cleaned, lower-cased, and ratings mentioned in the text are transformed into a numeric score.
  Cornell Raw is a superset of Cornell700/1000 but includes also unprocessed data, i.e. with proper (original) case, punctation, etc. which is required for some processing steps in GATE. In

particular, named entity recognition requires (or is at least improved by) proper case.

- **UPenn Multi-Domain Sentiment Dataset**
  http://www.cs.jhu.edu/~mdredze/datasets/sentiment/index2.html
  http://www.cs.jhu.edu/~mdredze/datasets/sentiment/domain_sentiment_data.tar.gz
  The multi-domain dataset contains Amazon reviews of four different domains which are books, dvds, electronics and kitchen & houseware. We focussed on the DVD domain as it substantially overlaps (apart from some training or fitness DVDs) with the movie domain.
  Note: the new Multi-Domain Sentiment Dataset (version 2.0 - http://www.cs.jhu.edu/~mdredze/datasets/sentiment/) has been barely used for sentiment classification and results haven't been evaluated systematically. We therefore refer only to the original (version 1.0) dataset.

- **Metacritics.com (Film and Music) Reviews**
  This dataset was created in the context of SEMPRE by adapting a crawler to extract information directly from the metacritics.com website and referred subsites (such as Chicago Tribune, USA Today, Wall Street Journal, Rolling Stone, and others).
  The crawling was initially executed for the music domain only, where we retrieved a total of **55695** reviews, of which **17786** have an extended review (from one of the subsites) and a known author of this review, another **20718** also have this extended review but the author is unknown, and the rest of 17191 reviews have only the summary information and the short punchline text.
  For the movie domain, the crawler was not that perfectly adapted (due to time restrictions in the student's work). However, we retrieved a total of **88418** reviews from metacritics, of which **27131** have an extended review and the author is known, another 61262 should have an extended review, but this review could not be downloaded or parsed correctly. Another 25 reviews from metacritics.com could not be handled due to unknown errors which haven't been investigated further.

2.1.1 Format of metacritics review files

The information in metacritics review files is stored in key-value pairs. Each line contains such a pair. Newlines in the review text were stripped away respectively replace by
 tags by the crawler.
The most important keys in the review files for further processing are Title, CriticRating, and CriticText. All other keys refer to information about the reviewed entity directly available at metacritics.com or the site, from which the review is originally from. This 3rd-party site is specified in the key CriticURL.

Movies
ID: 00C17EE40D99D75F2694BDA02BF800DB
IMDB:
SourceURL: http://www.metacritics.com/video/titles/dayzero
Label: First Look Studios
Title: Day Zero
GeneralRating: 41
Released: DVD: February 26, 2008|Theatrical: January 18, 2008
Genre(s): Drama
WrittenBy: Robert Malkani
DirectedBy: Bryan Gunnar Cole
RunningTime: 92 minutes, Color
Origin: USA
Punchline: Its view of the near future may be vaguely plausible and its performances persuasive, but its formulaic construction, internal inconsistencies and fuzzy ending undermine its integrity. It

has nothing to say about the big issues -- manhood, war and friendship -- that hasn't been explored with more depth and honesty in a hundred other movies.
CriticPublication: The New York Times
CriticURL: http://movies.nytimes.com/2008/01/18/movies/18zero.html?ref=movies
CriticAuthor: Stephen Holden
CriticRating: 40
CriticText: <p>If the abysmal reception so far for films about the war in Iraq is any indication ...

Music
ID: 5339E20061539493E68C345806C1C14C
SourceURL: http://www.metacritics.com/music/artists/bluntjames/allthelostsouls
Artist: James Blunt
Title: All The Lost Souls
GeneralRating: 53
Label: Atlantic
Released: 18 September 2007
Discs: 1
Genre: Rock, Pop
Punchline: He shows the abandon and confidence of a long-term artist, not just a one-hit wonder.
CriticPublication: Billboard
CriticURL:
http://www.billboard.com/bbcom/content_display/reviews/albums/e3ib1b99da9d9b5d7e2bf15ff78f509f5de
CriticAuthor: Kerri Mason
CriticRating: 80
CriticText: James Blunt can do a lot in less than four minutes. ...

2.2 Directory Layout

All sempre-relevant data and programs are stored somewhere in a shared common directory defined in the environment variable SEMPRE_HOME, e.g.:

```
export SEMPRE_HOME=/home/sempre
```

Data is stored in a directory `data` and filled with all described corpora gives the following directory tree:

```
$SEMPRE_HOME
  data

    cornell-raw
      movie

    cornell1000
      txt_sentoken
        neg
        pos

    cornell700
      tokens
        neg
        pos

    upenn
      sorted_data_acl
        books
```

```
    dvd
    electronics
    kitchen_&_housewares

metacritics-movies
  known author
    no url
      ...
    undownloadable
      ...
    url
      ...
  unknown author
    no url
    undownloadable
      ...
    url
  weird
    no url
    undownloadable
      ...
    url

metacritics-music
  known author
    ...
  unknown author
    no url
      ...
    url
      ...
```

3 Analysis & Preprocessing

All tools and scripts are placed under $SEMPRE_HOME/tools unless otherwise noted.

3.1 metacritics/mc-stats.sh – Overall Statistics/Summary on MetaCritics Reviews

This small shell script produces statistics on the number of documents from metacritics sources grouped by their type (i.e whether the extended review from the subsite could be downloaded and parsed and whether the author is known or unknown).

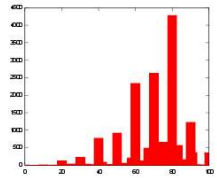
```
$ ./mc-stats.sh
metacritics-movies
. 88418
./unknown author 17
./unknown author/undownloadable 17
./weird 8
./weird/undownloadable 8
./known author 88393
./known author/undownloadable 61248
./known author/url 27131
./known author/no url 14

metacritics-music
. 55695
```

```
./unknown author 37909
./unknown author/url 20718
./unknown author/no url 17191
./known author 17786
```

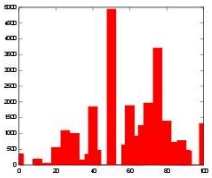
3.2 metacritics/score_hist.sh – Detailled Statistics/Score Histograms of MetaCritics Reviews

This shell scripts produces an overview of the rating distributions per metacritics directory. It requires gnuplot.
The scripts takes a parameter, which has to be either music or movies (matching the directory name of the data directory, i.e. $SEMPRE_HOME/data/metacritics-PARAMETER).
The result of the script is a temporary directory containing a HTML page including SVG drawings of the histograms.
The following diagram shows some diagrams for the music domain (left) and the movies domain (right).

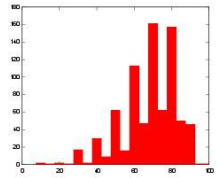
./known author (17787 reviews)



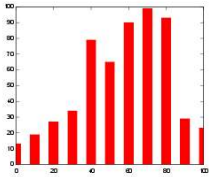
./known author/url (27132 reviews)



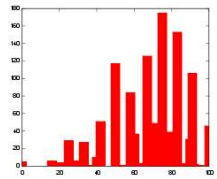
./known author/Prefix Magazine (781 reviews)



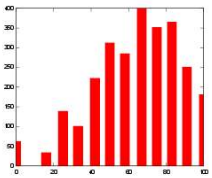
./known author/url/Film Threat (572 reviews)



./known author/Stylus Magazine (1143 reviews)



./known author/url/Entertainment Weekly (2706 reviews)



./known author/NOW Magazine (663 reviews)

./known author/url/The Onion (A.V. Club) (1112 reviews)

In addition to the html file showing histograms, detailled data is written to a log file. This log file contains statistics for each subdirectory in the form “score count”.

```
__known_author_undownloadable_Charlotte_Observer
0       3
12      9
25      48
38      174
50      192
63      190
75      279
88      163
100     36
```

3.3 metacritics/splitter.py – Partitioning and HTMLisation of MetaCritics

Reviews

For learning positive and negative reviews, they have to be split into these categories first. Choosing the thresholds is crucial for the classification performance.
The python script splitter.py allows to specify two tresholds as second and third parameter. The first (and lower one) specifies the upper bound for negative reviews. The second (and higher one) specifies the lower bound for positive reviews. Reviews within these thresholds are considered neutral.
The first parameter specifies wether movies or music is to be considered and the forth and last parameter specifies a subdirectory in the original metacritics directory to be processed.

```
$ python split.py movies 40 60 "known author/url"
splitting /home/bjung/sempre/data/metacritics-movies/known
author/url
output to /home/bjung/sempre/data/mc-split-movies/
neg: 3866
neutral: 9804
pos: 13461
```

The script places reviews according to their rating into three subdirectories of the output directory named pos, neg, and neutral.
The file written to these directories are (pseudo-)HTML files containing the review text only. All other attributes of the original metacritics reviews are stripped away.

3.4 upenn/split_upenn_dvd.py

This script converts upenn files into single documents and strips aways bad characters (e.g. < and > are replaced by the respective XML tags).

3.5 cornell/gen_ext_corpus.sh – Create extended Cornell700/1000 Corpora with full HTML

The Cornell700 and 1000 corpora are available only in preprocessed plain-text without proper case and punctuation. Fortunately, the documents contained in the cornell700/1000 sets are also available in the Cornell-Raw corpus and these corpora can be matched by a unique id.

This script generates a corpus cornell700-html or cornell1000-html containing the original HTML files used for the standard cornell700 and cornell1000 corpora. In addition, the script can produce movie titles only, which were used for checking coverage with IMDB and Wikipedia Movie pages.

```
gen_ext_corpus.sh [700|1000] [html|title]
```

```
$ ./gen_ext_corpus.sh 700 title
creating ... /home/bjung/sempre/data/cornell700-title/pos
creating ... /home/bjung/sempre/data/cornell700-title/neg
```

These new corpora allow to work with GATE NLP tools on the full text of the cornell700 and cornell1000 corpora with the same class labels (pos, neg) but the availability of punctuation, proper-case and HTML tags (i.e. paragraph structure).

3.6 Cornell Rating Extraction

Using the whole cornell-raw corpus with more than 27000 reviews requires definition of scores or class labels as the rating of the movie is not explicitely mentioned but comes in forms such as “three out of 5 stars”, “**** OUT OF ****”, and many others.
Pang & Lee used their own undocumented rules and manual intervention to classify their 700+700

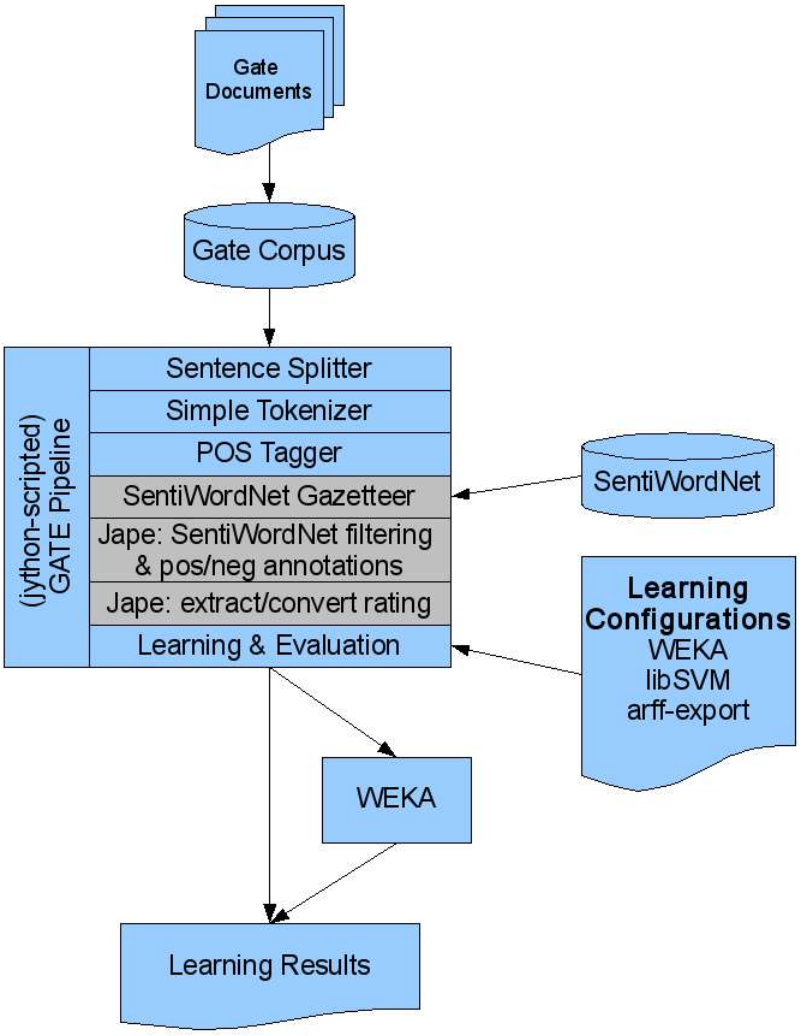
and 1000+1000 reviews.
We tried to reproduce their labelling by a set of hand-crafted JAPE rules (more than 20 pages), which are available on demand from Hannes Pirker. These rules achieve almost the same classification as in the Cornell700/1000 corpus. The few misclassifications can be attributed to different interpretation of the ratings respectively ambiguous scoring in the review text.

4 Jython Gate Pipelines

We use GATE as framework for processing the corpora. Instead of using the GUI of GATE or pure java code, we employ jython scripts for automating the whole process of pre-processing corpora (e.g. splitting of corpora see above), composing and running gate pipelines including the GATE learning plugin for producing feature files, converting to ARFF files, and running WEKA for learning and evaluating classifiers. Jython is a variant of the programming language python allowing for direct use of Java classes. It allows to directly script the use of GATE resources and operate directly on gate documents.

The jython scripts are kept simple and in declarative style to serve as documentation of processing steps executed to come to specific results.

The following diagram shows overall processing pipeline, starting with the import of GATE documents into a corpus, followed by the application of various GATE processing resources, and finally the use of the learning API to learn and evaluate a classifier or export data to be converted into ARFF files and processed by WEKA.



The Jython Gate base scripts can be found in the directory $SEMPRE_HOME/ofai-learning/basescripts.

Before running jython gate pipelines, the GATE environment has to be setup correctly. In particular, the GATE_HOME variable has to be set correctly and all gate libraries have to be added to the classpath. Additionally, the PYTHON_PATH should be adjusted to include the ofai-learning/base-scripts directory.

The shell script $SEMPRE_HOME/ofai-learning/env.sh can be used to perform these setup steps:

```
#!/bin/bash
[ -z $SEMPRE_HOME ] && echo SEMPRE_HOME not set && exit -1
[ -z $GATE_HOME ] && GATE_HOME=/home/bjung/sempre/gate

# jython jvm arguments (for debian)
export JAVA_OPTIONS=-Xmx3000M
# optional for debugging (i.e. starting java JDWP server)
[ "$1" == "DEBUG" ] && export JAVA_OPTIONS="-Xmx3000M -
agentlib:jdwp=transport=dt_socket,address=localhost:8000,server=y"

# jython jvm arguments (for gentoo)
export gjl_java_args="-Xmx3000M"
# optional for debugging (i.e. starting java JDWP server)
[ "$1" == "DEBUG" ] && export gjl_java_args="-Xmx3000M -
agentlib:jdwp=transport=dt_socket,address=localhost:8000,server=y"

# get classpath from gate build.xml
`ant -f $GATE_HOME/build.xml run-pre | grep CLASSPATH | sed -e
's/^.*\[echo\].*CLASSPATH=/export CLASSPATH=/'`

# setup python path
export PYTHON_PATH=$SEMPRE_HOME/ofai-learning/basescripts

# add WEKA to classpath
export CLASSPATH=$CLASSPATH:$SEMPRE_HOME/weka-3-6-1/weka.jar
export CLASSPATH=$CLASSPATH:$SEMPRE_HOME/libsvm-2.89/java/libsvm.jar

# spawn a new bash with correct environment
exec /bin/bash
```

4.1 Script Overview

The jython gate tools consist of a set of jython scripts in the $SEMPRE_HOME/ofai-learning/basescripts directory.

4.1.1 gatetools.py

A few helper functions that bridge some gaps between jython-based and pure java-based use of gate classes.

4.1.2 gate2arff.py

This script converts data in the “savedFiles” directory created by the GATE LearningAPI processing resource into the ARFF file format, used by the data mining software WEKA.

```
usage: gate2arff.py  [-bin|-tf|-tfidf|-tfidf2]
                     <savedFiles-dir> [outfile]
```

This script can produce different feature values:
-bin … binary feature value (feature present/absent)
-tf … term frequency
-tfidf … term frequency, inverse document frequency
-tfidf2 … variant of TFIDF with relative frequency instead of absolute frequency
The scripts writes the ARFF data either to the specified file or – if not specified – to stdout.

4.1.3 config.py

Defines an abstract class BaseConfig that configures a pipeline process, i.e. specifies the directories containing import document, the gate document store location, and the processing steps to execute. The exact use of these variables are described later for the concrete examples of processing cornell and metacritics corpora.

4.1.4 learning.py

This file contains constants used for creating the configuration for the GATE LearningAPI. These constants are used by the learn method defined in main.py.

learningProduceFeaturesCFG – the XML part describing the ProduceFeaturesOnly mode of the LearningAPI

learningSVM, learningNB, learningKNN – configuration of SVM, naïve Bayes and K-nearest neighbour classifiers.

datasetUnigramCFG – dataset definition for using unigrams of Token annotations as features

datasetBigramCFG – dataset definition for using bigrams of Token annotations as features

datasetUnigramMyTokCFG – dataset definition for using unigrams of MyTok annotations as features

datasetBigramMyTokCFG – dataset definition for using bigrams of MyTok annotations as features

4.1.5 process.py

This script contains functions that create regularly used processing resources in GATE pipelines such as deletePR() which creates a processing resource that removes annotations or jape(...) which create a processing resources that apply the JAPE rules specified in the given file.

4.1.6 main.py

This file defines the main class and initialises GATE. An instance of the main class takes a configuration object (see config.py) and exposes several functions to work with the corpus specified in the configuration and to apply processes.

getCorpus – sets up the corpus, either be loading an existing serial data store (SDS) or by importing documents from a directory (directories) in memory or an SDS.

process – executes the processing instructions specified in the steps variable of the configuration.

foreach – applies the given function to each document in the corpus. This can be used to perform scripted processing resources. It is used for instance to create class annotations in the documents based on their source url (/pos/,/neg/ subdirectories).

learn – executes the learning API processing resource on the corpus. This function takes two strings containing partial XML structures for constructing the configuration file for the learning API.

cleanup – closes and syncs data stores

4.1.7 eval.py - Running WEKA on ARFF files

WEKA classifiers can be executed by commands like the following:

```
java weka.classifiers.functions.LibSVM -S 1 -K 0 -D 3 -G 0.0 -R 0.0
-N 0.5 -M 40.0 -C 1.0 -E 0.0010 -P 0.1 -Z -x 3 -c 1 -t cornell1000-
unigram.arff
```

For easy configuration and systematic evaluation of various classifiers, the eval.py script can be used. It defines various WEKA classifiers together with parameters to be used. It applies these classifiers to a set of ARFF files and captures and stores the output for further analysis.

If the script is called without any parameters, the defined function genfiles is used to generate a list of filenames to apply classifiers to. The current version of the script contains a function to compose all filenames for the ARFF files generated by the cornell.py script for the cornell700 corpus (see below).

If the script is called with parameters, these are taken as filenames and the classifiers are applied to them.

```
$ jython eval.py /tmp/cornell700-bin-uni.arff
LibSVM applied to /tmp/cornell700-bin-uni.arff

  output to /tmp/cornell700-bin-uni.arff-LibSVM.res

  4 min 27 sec

IBk applied to /tmp/cornell700-bin-uni.arff

  output to /tmp/cornell700-bin-uni.arff-IBk.res

  2 min 33 sec
```

4.2 Working on cornell corpus

```
ofai-learning/scripts $ time jython -Dpython.path ../basescripts/ cornell.py 700

GATE home system property ("gate.home") not set.
Attempting to guess...
Using "/mnt/temp/home/bjung/sempre/gate" as GATE Home.
If this is not correct please set it manually using the -Dgate.home option in your start-
up script
Using /mnt/temp/home/bjung/sempre/gate as GATE home
Using /mnt/temp/home/bjung/sempre/gate/plugins as installed plug-ins directory.
Using /mnt/temp/home/bjung/sempre/gate/gate.xml as site configuration file.
Using /home/bjung/.gate.xml as user configuration file
Using /home/bjung/.gate.session as user session file
CREOLE plugin loaded: file:/mnt/temp/home/bjung/sempre/gate/plugins/ANNIE/
CREOLE plugin loaded: file:/mnt/temp/home/bjung/sempre/gate/plugins/learning/
corpus has 1400 documents
processing finished
Configuration File=file:/tmp/tmp4.ofai-learning-tmp


************************
A new session for NLP learning is starting.

bjung@stargate:~/sempre/ofai-learning/scripts$ emacs cornell.py
bjung@stargate:~/sempre/ofai-learning/scripts$ time jython -Dpython.path ../basescripts/
cornell.py 700
GATE home system property ("gate.home") not set.
Attempting to guess...
Using "/mnt/temp/home/bjung/sempre/gate" as GATE Home.
If this is not correct please set it manually using the -Dgate.home option in your start-
up script
Using /mnt/temp/home/bjung/sempre/gate as GATE home
```

```
Using /mnt/temp/home/bjung/sempre/gate/plugins as installed plug-ins directory.
Using /mnt/temp/home/bjung/sempre/gate/gate.xml as site configuration file.
Using /home/bjung/.gate.xml as user configuration file
Using /home/bjung/.gate.session as user session file
CREOLE plugin loaded: file:/mnt/temp/home/bjung/sempre/gate/plugins/ANNIE/
CREOLE plugin loaded: file:/mnt/temp/home/bjung/sempre/gate/plugins/learning/
corpus has 1400 documents
processing finished
Configuration File=file:/tmp/tmp0.ofai-learning-tmp


************************
A new session for NLP learning is starting.

real   5m14.198s
user   5m29.453s
sys    0m6.316s
```

This script takes the cornell700 corpus (or cornell1000 if specified) and produces 8 ARFF files in less than 6 minutes for learning in WEKA: 4 files with binary, term frequency, TFIDF, and TFIDF2 feature values for unigrams and another 4 for bigrams.

```
/tmp/cornell700-bin-bi.arff
/tmp/cornell700-bin-uni.arff
/tmp/cornell700-tf-bi.arff
/tmp/cornell700-tfidf2-bi.arff
/tmp/cornell700-tfidf2-uni.arff
/tmp/cornell700-tfidf-bi.arff
/tmp/cornell700-tfidf-uni.arff
/tmp/cornell700-tf-uni.arff
```

If the imp_sds variable in the configuration is set, the documents are imported into a serial data store, processing is applied, and the store is synced. This allows to save the results of long-running processing resources and continue executing the learning API on these cached results.

Note: once the corpus is populated (do_import=True in the call to getCorpus) and stored into the serial data store (imp_sds=True), further calls to getCorpus should have do_import=False to prevent importing duplicates.

The use of a serial data store for small corpora and fast processing resources is discouraged, because the document caching features of GATE are not efficient at all: Processing the whole cornell700 corpus in memory only takes less than 6 minutes; executing the learning resource only on a serial data store takes around 15 minutes.

In addition to producing ARFF files, the script can use the directly integrated classifiers in the GATE Learning API. The results logged to the logFileForNLPLearning.save are backed up to an unique log files for each configuration.

```
#!/usr/bin/env jython

from config import BaseConfig
from main import Main
import learning
import process
import gate
import os
#import gate2arff

##############################################################################
# processing functions
##############################################################################

def tagdoc(d):
    """Process a single GATE document. Set the Rating (i.e. class label)
    based on the filename (sourceurl). Mark the whole document as
    'review_text'.
```

```
    """
    if '/pos/' in d.getSourceUrl().toString(): classlabel='pos'
    else: classlabel='neg'

    fs=gate.util.SimpleFeatureMapImpl()
    fs.put('rating',classlabel)
    d.getAnnotations('Original markups').add(0,d.getContent().size(),'Rating',fs)
    fs=gate.util.SimpleFeatureMapImpl()
    d.getAnnotations('Original markups').add(0,d.getContent().size(),'review_text',fs)

##############################################################################
# Configurations
##############################################################################
class Cornell700(BaseConfig):
    """Configuration for the Cornell 700/700 Corpus
    """
    dataroot = BaseConfig.dataroot + '/data/cornell700-html'
    imp_dirs = ['neg','pos']

    # set this flag to TRUE if you want to generate a serialised GATE corpus
    imp_sds = False

    storename = 'cornell700'

    # list of processing steps to apply to the corpus
    steps = [process.sentenceSplitter(),
             process.tokenizer(),
             ]

class Cornell1000(BaseConfig):
    """Configuration for the Cornell 1000/1000 Corpus
    """
    dataroot = BaseConfig.dataroot + '/data/cornell1000-html'
    imp_dirs = ['neg','pos']

    # set this flag to TRUE if you want to generate a serialised GATE corpus
    imp_sds = False

    storename = 'cornell1000'
    # list of processing steps to apply to the corpus
    steps = [process.sentenceSplitter(),
             process.tokenizer(),
             ]

##############################################################################
# handle parameters, import corpus, process documents, and
# generate ARFF files or use the learning API to directly learn a classifier
##############################################################################

# instantiate a Cornell700 or Cornell1000 configuration object
import sys
if len(sys.argv) > 1 and sys.argv[1]=='700':
    cfg,prefix = Cornell700(),'cornell700'

elif len(sys.argv) > 1 and sys.argv[1]=='1000':
    cfg,prefix = Cornell1000(),'cornell1000'

else:
    print 'usage: cornell.py [700|1000]'
    sys.exit(-1)

# generate the main object initialized from the configuration
m=Main(cfg)

# retrieve the corpus and perform import of documents
# NOTE: set do_import to False if the imp_sds Flag of the configuration object
# is set to True and documents have already been imported
m.getCorpus(do_import=True)

# apply the tagdoc function to each document
m.foreach(tagdoc)
```

```
# execute the processing steps defined in the steps variable of the
# configuration object
m.process()

#
cfgs=(
    ('SVM',learning.learningSVM),
    ('KNN',learning.learningKNN),
#    ('NB', learning.learningNB),
)
for n,c in cfgs:
    logout='/tmp/%s-%s-uni.res' % (prefix,n)
    print n, 'Unigram', logout
    m.learn(c,learning.datasetUnigramCFG,learn=True,copylog=logout)
for n,c in cfgs:
    logout='/tmp/%s-%s-bi.res' % (prefix,n)
    print n, 'Bigram', logout
    m.learn(c,learning.datasetBigramCFG,learn=True,copylog=logout)

# execute the Learning Processing Resource to produce UNIGRAM features
# followed by the use of gate2arff.py to convert the GATE output to
# ARFF files (for different feature values: bin, tf, tfidf, tfidf2)
# Note: you can skip the producefeaturesCFG step if you did another
# learning step before, because the datafiles have then already been created.
featurevaluetypes=['-bin'] #,'-tf','-tfidf','-tfidf2']
m.learn(learning.learningProduceFeaturesCFG,learning.datasetUnigramCFG)
for t in featurevaluetypes:
    #gate2arff.convert("/tmp/savedFiles",
    #                  open("/tmp/%s-%s-uni.arff" %
(prefix,t),'w'),gate2arff.gentypes[t])
    # calling gate2arff via system call is MUCH MUCH FASTER
    os.system('python ../basescripts/gate2arff.py %s /tmp/savedFiles /tmp/%s%s-uni.arff'
%
              (t,prefix,t))

# the same for bigrams
m.learn(learning.learningProduceFeaturesCFG,learning.datasetBigramCFG)
for t in featurevaluetypes:
    #gate2arff.convert("/tmp/savedFiles",
    #                  open("/tmp/%s-%s-bi.arff" % (prefix,t),'w'),gate2arff.gentypes[t])
    # calling gate2arff via system call is MUCH MUCH FASTER
    os.system('python ../basescripts/gate2arff.py %s /tmp/savedFiles /tmp/%s%s-bi.arff' %
              (t,prefix,t))

# cleanup the gate objects, sync the corpus back to disk
m.cleanup()
```

4.3 Working on metacritics corpus

The script metacritics.py demonstrates further use of this scripting infrastructure. It shows the use of Gazatteers to tag positivity and negativity based on the Sentiword Net Dictionary (http://sentiwordnet.isti.cnr.it/) to filter out neutral words.

Executed on a subset of 1395 documents, the performance of a 3-fold CV with SVM classification with unigrams is around 73.8% F1, which is lower then the F1 score for unfiltered unigrams of 77% but at a reduction of the feature list from 42737 down to 1260 features.

4.4 Notes on reading the logFileForNLPLearning.save

When using the GATE integrated Learning API module, the learning and evaluation results are logged to the logFileForNLPLearning.save in the savedFiles directory. The following describes how to read this file and interpret the values.

```
...
...
*** Averaged results for each label over 3 runs as:


Results of single label:

0 LabelName=neg, number of instances=467

  (correct, partialCorrect, spurious, missing)= (196.33333, 0.0, 43.666668, 36.666668);
(precision, recall, F1)= (0.8182859, 0.84335047, 0.83022755);  Lenient: (0.8182859,
0.84335047, 0.83022755)

1 LabelName=pos, number of instances=467

  (correct, partialCorrect, spurious, missing)= (189.33333, 0.0, 36.666668, 43.666668);
(precision, recall, F1)= (0.83814186, 0.8127494, 0.8248221);  Lenient: (0.83814186,
0.8127494, 0.8248221)



Overall results as:

  (correct, partialCorrect, spurious, missing)= (385.66666, 0.0, 80.333336, 80.333336);
(precision, recall, F1)= (0.8276109, 0.8276109, 0.8276109);  Lenient: (0.8276109,
0.8276109, 0.8276109)



Jun 24, 2009 4:20:30 PM: This learning session finished!.
```

The following explains the meaning of the different numbers for correct, partialCorrect, spurious, and missing and documents how these values are calculated.

For each label X, all TEST instances are checked and evaluated via the AnnotationDiffer class.

In case of both original (ground truth) and assigned (classified) label being X, the result is 'correct'.

In case of original label being X and the assigned label something else, the result is 'missing'.

In case of assigned label being X and the original label something else, the result is 'spurious'.

Otherwise (neither original nor assigned label match X), the result is not counted.

In terms of 2-class problems:

correct = True Positive

missing = False Negative

spurious = False Positive

The difference between lenient and strict is not visible as we do not have partialCorrect results. Lenient values are where the partial matches are considered as correct (correct = correct + partialCorrect).

Precision = (GT=X and CLASS=X)/(CLASS=X)

Recall = (GT=X and CLASS=X)/(GT=X)

F1 = 2*(Precision*Recall)/(precision+recall)

4.4.1 Some Results on Cornell700/1000

(3-fold cross validation, averaged results over 3 runs):

	Precision	Recall	F1
Cornell (700+700), P1 (< 5 min)			
Test-neg	0.81636715	0.8132882	0.8138364
Test-pos	0.8139839	0.8180484	0.81500536
Test-total	0.81473535	0.81473535	0.81473535
Cornell (1000+1000), P1 (< 5 min)			
Test-neg	0.8687462	0.8567472	0.86257076
Test-pos	0.8591485	0.8710373	0.86491317
Test-total	0.8638639	0.8638639	0.86386395
Cornell (1000+1000), P2 (~ 100 min)			
Test-neg	0.32332334	0.6666667	0.43543375
Test-pos	0.15615615	0.33333334	0.21267892
Test-total	0.47947946	0.47947946	0.47947946
Cornell (1000+1000), P3 (~ 5 min)			
Test-neg	0.56541353	0.67991847	0.47296318
Test-pos	0.83822775	0.32945856	0.22918975
Test-total	0.511011	0.511011	0.511011
Cornell (1000+1000), P4 (~ 100 min)			
Test-neg	0.32882884	0.6666667	0.4404203
Test-pos	0.16216217	0.33333334	0.21818183
Test-total	0.490991	0.490991	0.490991

All runs were performed with Unigrams and TF-IDF as feature value.

P1 = splitter.SentenceSplitter, tokeniser.SimpleTokeniser,
SVMLibSvmJava -c 0.7 -t 0 -m 100 -tau 0.5

P2 = splitter.SentenceSplitter, tokeniser.SimpleTokeniser,
NaiveBayesWEKA

P3 = splitter.SentenceSplitter, tokeniser.SimpleTokeniser,
KNNWEKA (-k 1)

P4 = splitter.SentenceSplitter, tokeniser.SimpleTokeniser,
C4.5Weka

5 References

[Pang & Lee, 2002] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan, Thumbs up? Sentiment Classification using Machine Learning Techniques, Proceedings of EMNLP 2002.

[Pang & Lee, 2004] Bo Pang and Lillian Lee, A Sentimental Education: Sentiment Analysis Using Subjectivity Summarization Based on Minimum Cuts, Proceedings of ACL 2004.