mycoSORT `1.0`

User Manual

Hayda Almeida
Marie-Jean Meurs

Tsang Lab

May 2015

Contents

1 Introduction to mycoSORT

mycoSORT is an open source text classification software program written in Java. The software is based on data sampling and machine learning methods. **mycoSORT** was primarily developed to perform text classification of scientific literature related to fungal enzymes. However, it can also execute classification of literature related to different topics. This tool can be applied to support scientific researchers in the selection of relevant documents when performing literature review.

mycoSORT utilizes a labeled document collection to learn from, and generate a classification model through supervised learning, which will be used to predict a label for new scientific papers. In order to obtain a classification prediction for a given document, **mycoSORT** must first train or load a classification model.

To generate the classification models, **mycoSORT** makes use of the standard implementation of classification algorithms provided by the Weka workbench [1], which is also developed in Java. In addition, **mycoSORT** utilizes the following packages:

- Apache Commons Lang[1](version 3.2.1 or above), a Java utility package;
- jsoup[2](version 1.7.3 or above), a Java HTML/XML parser;
- LIBSVM[3](version 3.2 or above), a wrapper library for the SVM classifier;
- Apache Ant[4](version 1.9.3 or above), a Java library used to build Java applications.

In the following sections, you will find instructions on how to have access to the system source code, install, and run the software tool. **mycoSORT** toolkit is available at `https://github.com/TsangLab/mycosort`. This user manual describes **mycoSORT** version `1.0`.

2 Getting Started

To download **mycoSORT** toolkit, please access `https://github.com/TsangLab/mycosort`. This user manual is written for version `1.0`; and assumes that you have downloaded and extracted the `mycosort-pck-1.0` as a folder, and that this folder is currently in your working directory.

2.1 Package Content

mycoSORT toolkit contains several folders and files in its root folder. These items are used to provide inputs and keep outpus for the system subtasks. Their usage and content are further explained here.

[1] `https://commons.apache.org`
[2] `http://jsoup.org/`
[3] `http://www.csie.ntu.edu.tw/~cjlin/libsvm/`
[4] `http://ant.apache.org/`

Folders list *arff* → contains the .ARFF files representing the data as vector matrices. These files are used to generate the classification models.
corpus → keeps the training and test sets in .XML format, used to build and apply the classification models.
executables → holds the .JAR files that compose the system and are used to perform the system tasks.
features → contains the features extracted from the training sets, saved in .TXT format.
jars → contains the external .JAR packages which are bundled with the system.
src → holds the system .JAVA source files.

Files list *build.xml* → master file used by Apache Ant to build **mycoSORT** executables.
config-sample.cfg → a sample of the configuration file used to set specific parameters for the system different tasks.
entities.txt → a list of bioentities that are found annotated in the dataset.
stopList.txt → a list of stop-words to be considered for feature extraction.

2.2 Requirements

mycoSORT requires Java JDK (version 1.8.0 or above) and Apache Ant (version 1.9.3 or above) to be installed.
For further information on how to install Java JDK, please refer to:
`http://www.oracle.com/technetwork/java/javase/downloads/index.html`.
For information on how to install Apache Ant, please refer to:
`http://ant.apache.org/manual/install.html`.

3 Configuration Setup

The general working environment and configurations of **mycoSORT** are defined in a file named `config.cfg` . To generate a `config.cfg` file, create a copy of `config-sample.cfg` and rename it to `config.cfg` . Before compiling and running **mycoSORT** , it is required to edit the `config.cfg` file.

3.1 Directory Setting

To set up the main directory for using **mycoSORT** , firstly update the `HOME_DIR` directory for your own desired folder, as in the following example. The `HOME_DIR` should contain the path of the system main folder, where the **mycoSORT** toolkit was extracted.

```
HOME_DIR=/home/usr/mycosort-pck-version/
```

The following directories are set by default, and generally should not be changed, since they refer to folder paths inside of your `HOME_DIR`. The corpus directory contains the dataset .XML files, as well as the training and testing files.

```
CORPUS_DIR=corpus/
```

The positive and negative folders contain the positive and negative .XML instances, and are found inside of `CORPUS_DIR`.

```
POS_DIR=positives/
NEG_DIR=negatives/
```

The train and test folders will contain the train and test .XML instances, and are found inside of CORPUS_DIR.

```
TRAIN_DIR=train/
TEST_DIR=test/
```

The arff directory contains the .ARFF files, used to feed the classification algorithms.

```
OUTPUT_MODEL=arff/
```

The feature directory contains the .TXT files listing all feature types extracted from the training sets.

```
FEATURE_DIR=features/
```

The duplicates directory is a directory in which the user wants to look for duplicates. Its value should be edited to fit the folder name, which should be placed inside of CORPUS_DIR.

```
DUP_DIR=test/
```

3.2 Corpus Sampling Setting

Data sampling can be used to split the document collection into training and test collections, as well as to generate several training collections with different class distributions. To enable the training or test sampling, set the following variables to true:

```
SAMPLE_TRAIN=false
SAMPLE_TEST=false
```

The following variables will control the data sampling settings. To determine the size of the test set with regards to the entire document collection, use PERCT_TEST to set the percentage of the test collection.

```
PERCT_TEST=15
```

To generate a training collection, first define the percentage of positive instances to be sampled for this corpus. This variable is also used when generating .ARFF files.

```
PERCT_POS_TRAIN=50
```

To generate a test collection, first determine its the percentage of positive instances.

```
PERCT_POS_TEST=10
```

3.3 File Setting

We describe here the files used as input for **mycoSORT** . The TRAINING_FILE and TEST_FILE should contain the name of the XML files generated as training and test sets. The training file is used by the extractors to extract features, and to build the .ARFF files.

```
TRAINING_FILE=triage0.xml
TEST_FILE=triage1.xml
```

The .ARFF files are used to feed the classification models. When wanting to re-train a model, ARFF_TRAIN should contain the .ARFF file name used for training. When wanting to test new instances, ARFF_TEST should contain the .ARFF file name used for testing.

```
ARFF_TRAIN=triage0.arff
ARFF_TEST=triage1.arff
```

The stopwords list used by the extractors is defined here. We recommend to keep this variable as it is defined in the `config-sample.cfg` .

```
STOP_LIST=stopList.txt
```

When executing subtasks, **mycoSORT** produces the following files as output, which are later on used as input for new subtasks. These files contain the features extracted from a given training set.

```
ECNUM_FEATURES=ecnumbers.txt
JOURNAL_TITLE_FEATURES=journaltitles.txt
ANNOTATION_FEATURES=annotations.txt
TITLE_FEATURES=titleAnnotations.txt
NGRAM_FEATURES=ngrams_features.txt
TITLE_NGRAMS=titleGrams.txt
DOC_IDS=docIDs.txt
```

3.4 Feature Setting

The feature configuration is taken into account when generating .ARFF files. In order to choose a feature type to be used when creating an .ARFF file, simply set its value to "true", as the examples below.

General features More than one feature can be combined when generating .ARFF files. The following variables will load general features: the size of a paper abstract, the name of the publication journal, and the EC numbers found in a paper.

```
USE_TEXT_SIZE=true
USE_JOURNAL_TITLE_FEATURE=true
USE_ECNUM_FEATURE=true
```

The `USE_DOC_ID` variable extracts the paper PMID. This variable must be maintained with its value set to "true", since it is needed to output the classification predictions according to the document ID.

```
USE_DOC_ID=true
```

The following variables set specific conditions for feature frequency (number of times it was found in the training set) and feature lenght (number of characters in a feature) to be taking into account when extracting the feature list. The default parameters are defined below, but they can be adjusted according to the user needs.

```
FEATURE_MIN_FREQ=2
FEATURE_MIN_LENGTH=3
```

Annotation features The following variables will provide annotation features to generate .ARFF files. To load the bioentity annotations extracted from the training set, the value of `USE_ANNOTATION_FEATURE` must be set to true. When setting `USE_ANNOTATION_TYPE` to true, the bioentity types will be loaded to generate .ARFFs. Finally, when setting `USE_TITLE_FEATURE`

to true, the bioentities annotated in paper titles will be considered separately from the annotations found in abstracts.

```
USE_ANNOTATION_FEATURE=true
USE_ANNOTATION_TYPE=true
USE_TITLE_FEATURE=true
```

N-Gram features The following variables will provide n-gram features to generate .ARFF files. To load the n-grams extracted from the training set, the value of `USE_NGRAM_FEATURE` must be set to true. When setting `USE_TITLE_NGRAMS` to true, the n-grams found in paper titles will be considered separately from the n-grams found in abstracts. Use `NGRAM_STOP` to remove stopwords from the feature list.

```
USE_NGRAM_FEATURE=true
USE_TITLE_NGRAMS=false
NGRAM_STOP=true
```

The variable `NGRAM_SIZE` determines the number of words used to form n-grams. The default value is 1, however the system is also capable of generating bigrams (`NGRAM_SIZE=2`) and trigrams (`NGRAM_SIZE=3`).

```
NGRAM_SIZE=1
```

To apply a weight in a n-gram, set the following variable to true and determine the value of the weight. This configuration will simply multiply the current n-gram frequency by the value provided in `WEIGHT`.

```
USE_WEIGHTED_NGRAM=false
WEIGHT=3
```

3.5 Feature Selection Setting

The feature selection configuration is taken into account before feeding .ARFF files to the classification algorithms. To enable Odds Ratio (OR) or IDF filtering, just set one of the following variables to true:

```
USE_ODDS_RATIO=true
USE_IDF=false
```

It is recommended to apply Odds Ratio or IDF, but not both together. To determine the minimum threshold considered to keep a feature, adjust the following variables (default is set to 1):

```
OR_THRESHOLD=1
IDF_THRESHOLD=1
```

3.6 Experiment

The experiment type is used to generate .XML and .ARFF files. To generate training files, set `EXP_TYPE=0`, and to generate test files, set `EXP_TYPE=1`.

```
EXP_TYPE=0
```

4 Using mycoSORT

mycoSORT can be used from a command line interface. The system utilizes Apache Ant to build the five different modules (.JAR files), which are available in the `executables` folder. To execute **mycoSORT** modules, it is necessary to access the system home folder. In a command line interface (a terminal in Linux OS, or a prompt in Microsoft Windows), navigate until the `mycosort-pck-1.0` folder, such as:

```
user@machine $ cd /home/usr/mycosort-pck-version
```

On a Microsoft Windows system, the forward slashes should be replaced by back slashes (e.g. `home\usr\...`). From now on, the instructions will assume that a Linux OS is being used.

Compiling After accessing the system home folder, it is necessary to first compile **mycoSORT** modules. To do so, simply type `"ant"` in the command line, as the example below:

```
user@home/usr/mycosort-pck-version $ ant
```

The system should be re-compiled if any parameter is changed or edited in the `config.cfg` file. Following we describe the usage and configuration for each of the five **mycoSORT** modules:

- SampleCorpus
- CorpusHandler
- FeatureExtractor
- NgramExtractor
- BuildModel
- Trainer

4.1 SampleCorpus

The `SampleCorpus` module allows the user to generate training and test collections. It utilizes all .XML documents contained in the `corpus/positive` and `corpus/negative` folders.

Example 1 When generating the test collection, the .XML instances randomly selected will be moved from the `corpus/positive` and `corpus/negative` folders to the `corpus/test` folder. To execute the test sampling and generate a test collection that represents 15% of the entire document collection, and that contains 10% of positive instances, edit the following variables in the `config.cfg` :

```
SAMPLE_TEST=true
PERCT_TEST=15
PERCT_POS_TEST=10
```

Example 2 When generating the training collection, the .XML instances randomly selected will be copied from `corpus/positive` and `corpus/negative` folders to the `corpus/train_(PERCT_POS_TRAIN)` folder. To execute the training sampling and generate a trainng collection that contains 50% of negative instances and 50% of positive instances, edit the following variables in the `config.cfg` :

```
SAMPLE_TRAIN=true
PERCT_POS_TRAIN=50
```

To execute `SampleCorpus`, run the following instruction in the command line interface:

```
user@home/usr/mycosort-pck-version $ ant
user@home/usr/mycosort-pck-version $ ant sample-corpus
```

After running the instruction, the selected sampling (training or test) will be executed. The training collection will then be copied to a `corpus/train_50` folder. The training collection can be generated multiple times, with different class distributions.

4.2 CorpusHandler

The `CorpusHandler` module is used to create the training and test corpus, by generating a combined .XML file containing all .XML instances either in the `corpus/test` folder or in the `corpus/train_(PERCT_POS_TRAIN)` folder.

Example 3 Besides generating the training and test corpora, this module can also perform a check for duplicates. To check for duplicates between an existing training file and a given `DUP_DIR` folder containing several .XML files, edit the following variables in the `config.cfg` file:

```
TRAINING_FILE=triage0.xml
DUP_DIR=test/
```

To execute `CorpusHandler` and check for duplicates between training file and a given folder, run the following instruction in the command line interface:

```
user@home/usr/mycosort-pck-version $ ant
user@home/usr/mycosort-pck-version $ ant -Doptions=df corpus-handler
```

Example 4 To check for duplicates between the train or test folders containing all .XMLs and a given `DUP_DIR` folder containing other several .XML files, edit the following variables in the `config.cfg` file:

```
DUP_DIR=test/
EXP_TYPE=0
```

In this case, `EXP_TYPE=0` if the train .XMLs must be considered, or `EXP_TYPE=1` if the test .XMLs must be considered. To execute `CorpusHandler` and check for duplicates between training file and a given folder, run the following instruction in the command line interface:

```
user@home/usr/mycosort-pck-version $ ant
user@home/usr/mycosort-pck-version $ ant -Doptions=dc corpus-handler
```

When checking for duplicates, the duplicates found will by default renamed only in the `DUP_DIR` folder.

Example 5 To generate a training corpus, the following variables must be edited in the `config.cfg` file:

```
PERCT_POS_TRAIN=50
EXP_TYPE=0
```

To generate a testing corpus, edit the following variables:

```
PERCT_POS_TEST=10
EXP_TYPE=1
```

In order to generate the corpora, it is first required to clean (`-Doptions=cl`), and only then concatenate (`-Doptions=cc`) all .XMLs in a given folder. Thus, when creating the training or test corpora, run the following instruction in the command line interface:

```
user@home/usr/mycosort-pck-version $ ant
user@home/usr/mycosort-pck-version $ ant -Doptions=cl,cc corpus-handler
```

4.3 NgramExtractor

The `NgramExtractor` is a feature extraction module, used to extract n-grams (small units of text) from the paper article title and abstract. N-grams can be generated in three different sizes: unigrams (one word), bigrams (two words), and trigrams (three words). The default ngram size is one word, since this extraction already results in long list of features. It is also recommended to discard stopwords when extracting n-grams, and this can be set by keeping the value of `NGRAM_STOP` as `true`.

Example 6 To perform the ngram extraction, the following variables must be edited in the `config.cfg` file:

```
TRAINING_FILE=triage0.xml
NGRAM_STOP=true
NGRAM_SIZE=1
FEATURE_MIN_FREQ=2
FEATURE_MIN_LENGTH=3
```

To execute `NgramExtractor` run the following instruction in the command line interface:

```
user@home/usr/mycosort-pck-version $ ant
user@home/usr/mycosort-pck-version $ ant ngram-extractor
```

4.4 FeatureExtractor

The `FeatureExtractor` is a feature extraction module, used to extract domain annotations (specific XML tags) from the paper article title and abstract. To specify the list of annotations (tags) to be considered in the `FeatureExtractor` module, please refer to the `entities.txt` file in the root of `mycosort-pck-1.0` folder. In order to consider new annotation type (tag) beside the ones provided in the file, simply add a new line containing the annotation type name and its level (sentence or entity). Should a given type not be considered, simply add a # at the beggining of the specific line.

Example 7 To perform the feature extraction, the following variables must be edited in the `config.cfg` file:

```
TRAINING_FILE=triage0.xml
FEATURE_MIN_FREQ=2
FEATURE_MIN_LENGTH=3
```

To execute `FeatureExtractor` run the following instruction in the command line interface:

```
user@home/usr/mycosort-pck-version $ ant
user@home/usr/mycosort-pck-version $ ant feature-extractor
```

4.5 BuildModel

The `BuildModel` is the module used to represent the training and test sets as matrix of document vectors, that will be later fed to a classification algorithm. Models are saved in the .ARFF file format, and can be generated with several different configurations of features. All generated models are saved in the `arff` folder.

Example 8 To determine the feature configuration used in a given model, the chosen options, as described in 3.4, must be set to `true` in the `config.cfg` file. As an example, if the user wants to generate a model based only in unigram features, the setup of n-gram features must be set to true, as described in 3.4, while the annotation features setup must be set to false.

In addition, the following variables must also be edited, to indicate if the model should be generated based on the training set or the test set, as well as to indicate which percentage of positives was currently considered in the training set.

```
PERCT_POS_TRAIN=50
EXP_TYPE=1
```

To execute `BuildModel`, run the following instruction in the command line interface:

```
user@home/usr/mycosort-pck-version $ ant
user@home/usr/mycosort-pck-version $ ant build-model
```

4.6 Trainer

The `Trainer` module processes the training .ARFF files and utilizes a classification algorithm to generalize a function, and output predictions for instances in the test .ARFF files. The corresponding training and test .ARFF files must be indicated in the `config.cfg` file before executing the `Trainer` module. In order to specify the correct files, please refer to these two items:

```
ARFF_TRAIN=triage0.arff
ARFF_TEST=triage1.arff
```

While training and testing the models, feature selection methods can also be set up. To perform IDF or Odds Ratio filtering, please refer to the items described in 3.5. It is recommended to perform one of the filterings at once, either IDF or Odds Ratio, as opposed to both at the same execution.

Example 9 A model can be trained using three different classification algorithms: Naïve Bayes (`-Dclassifier=nb`), Support Vector Machine (`-Dclassifier=svm`), or Logistic Model Tree (`-Dclassifier=lmt`).

To execute `Trainer` using LMT, run the following instruction in the command line interface:

```
user@home/usr/mycosort-pck-version $ ant
user@home/usr/mycosort-pck-version $ ant -Dclassifier=lmt trainer
```

5 Contacts

Should you have any questions, comments or bug reports, the authors can be reached at the following addresses:
`hayda.almeida@concordia.ca`
`marie-jean.meurs@concordia.ca`

References

[1] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. The weka data mining software: an update. *ACM SIGKDD Explorations Newsletter 11*, 1 (2009), 10–18.