

NlpLearner 0.2.1

NlpLearner is a program that (1) learns how to classify text documents into categories and (2) evaluates how well it learned. This document describes NlpLearner. Specifically, it summarizes what comes with the distribution, some of the ways that a user can use the program, and some of the major changes since the previous version. There are also some notes at the end of the document.

1 The Distribution

The distribution consists of the following:

1. config/ : A few sample configuration files.
2. scripts/ : A few supporting scripts.
3. mark/ : Java source.
4. docs/ : Java docs.
5. nlpLearner.jar : Java classes.
6. readme.pdf : Documentation.

2 Using the Program

Following is a UML Use Case diagram describing the primary ways users interact with NlpLearner. Detail for each use case follows the diagram. Throughout the descriptions, this section assumes that the distribution is in ~/dist.

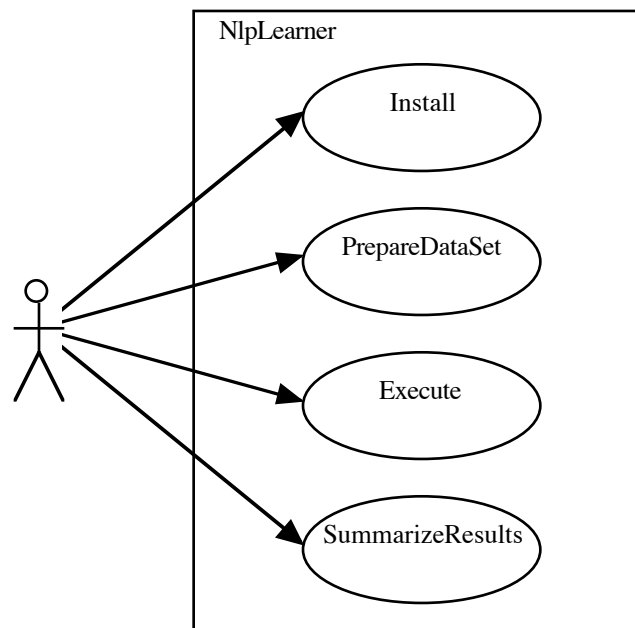


Diagram 1: NlpLearner Use Cases

The Install Use Case

The user initiates the use case in order to install NlpLearner.

Preconditions

Java 1.3 (or later) and weka 3.2.1 (or later) must be installed.

Main Flow

The user first creates a directory for NlpLearner, places the distribution file nlpLearner0.2.1.jar into the directory, and then extracts the distribution.

```
mkdir ~/dist
mv nlpLearner0.2.1.jar ~/dist
cd ~/dist
jar xf nlpLearner0.2.1.jar
```

Next, the user must add a path to the CLASSPATH environment variable.

```
setenv CLASSPATH ${CLASSPATH}:${HOME}/dist/nlpLearner.jar
```

The user finally restarts the shell session. The use case then terminates.

Subflows

None described.

Errors

None described.

The PrepareDataSet Use Case

The user initiates this use case in order to prepare a data set for use by NlpLearner.

Preconditions

The user must have executed the Install use case.

Main Flow

The user first obtains a data set from some source. The user then packages the data set so that the following conditions hold:

1. The text of each instance exists in a single file.
2. All instance files are underneath some directory.
3. No other files are underneath this directory, except that the instance files may be organized into a directory hierarchy.
4. For each category, there is a string which exists in the full path of an instance file if and only if that instance is in that category.
5. For each desired test set, there is a string which exists in the full path of an instance file if and only if that instance is in the test set.

The user may satisfy the last two conditions by executing (S-1) or (S-2) or in some other way. The use case then terminates.

Subflows

S-1: Define Three Level Directory Structure

The user defines a directory underneath the main directory for each test set and one for files (assuming there are any) that are not in any test set. Underneath each of these directories, the user defines a directory for each category. Underneath each category directory, the user places the instance files that belong to the corresponding category and corresponding test set.

S-2: Use the Stratify Script

The user defines a directory underneath the main directory for each category. Underneath each category directory, the user then places the instance files that belong to the corresponding category. The user then executes the `~/dist/scripts/nlpStratify` script on each category directory to rename each instance file so that it is in one of ten test sets.

Errors

None described.

The Execute Use Case

The user initiates the use case in order to apply NlpLearner to a data set.

Preconditions

The user must have executed the PrepareDataSet use case. If the user wishes to execute the OutputAdditionalInformation subflow, the user must have removed all files whose names begin with "out-" from the current directory.

Main Flow

The user first decides which data set to use. The user then configures two input text files that specify various parameters. The "settings" file specifies parameters, which, for a given data set, seldom change. These parameters include:

1. Name: The name of the data set. NlpLearner embeds this parameter within the output and within the names of any output files.
2. DataSet : the path of the data set's directory. This directory is the main directory created in the PrepareDataSet use case.
3. CatSetMatch : A set of strings, one for each category. The full path of each instance file contains exactly one of these strings. The string an instance file contains determines the category to which the instance belongs.
4. TestSets : A set of strings, one for each test set. If the full path of an instance file contains a given string, then it is in the corresponding test set. NlpLearner will perform the same operations for each training/test split specified.

The "parameters" file specifies one or more sets of parameters that often change. The user configures a parameters file according to the directions in ~/dist/config/parameters.txt. The user then executes the ~/dist/scripts/nlpLearn script, passing in the two parameter files (E-1) and whether NlpLearner should generate "extra files". Usage is:

```
perl ~/dist/scripts/nlpLearn <settingsFile> <parametersFile> <generateFiles>
```

where

<settingsFile>	- is the settings parameter file.
<parametersFile>	- is the parameters parameter file.
<generateFiles>	- is a boolean ("0" or "1") telling whether to generate the extra files.

For each (training/test split) X (parameter set) pair, NlpLearner performs a run. A run consists of:

1. Tokenizing the data set.
2. Selecting a set of features using only the training set.
3. Vectorizing the training and test sets.
4. Training the classifier on the training vectors.
5. Classifying the training vectors and test vectors.
6. Reporting results to standard out. Results include all of the parameters, plus percentage correct classification for both training and test sets.

If the user specified that NlpLearner generate files, NlpLearner will also execute (S-1) for each run. Each time NlpLearner performs a run, it uses as much of the work from the previous run as it can. Prior to performing the runs, NlpLearner orders the runs so that it will save the most amount of work overall. As a result, the order in

which NlpLearner performs the runs may be different from the order in which the user specified the runs in the input files. This ordering scheme provides extreme time efficiency gains. The use case terminates when NlpLearner has processed all runs.

Subflows

S-1: OutputAdditionalInformation

For each run, NlpLearner outputs three files, named according to the parameters in the two parameter files. The first file lists all of the features selected and the vectors for the training set. The second file lists all of the features selected and the vectors for the test set. The final file lists more detailed information about how the classifier performed on the training and test sets, including confusion matrices.

Errors

E-1: If any of the parameters are invalid, the program will at some point, although not necessarily immediately, exit with an exception.

The SummarizeResults Use Case

The user initiates the use case in order to summarize results from the files generated in the (S-1) subflow of the Execute use case.

Preconditions

The use must have executed the (S-1) subflow of the Execute use case. The files produced by this subflow must all be in the same directory, and no other files beginning with "out-" can be in this directory.

Main Flow

The user first changes to the directory containing the files produced by the (S-1) subflow of the Execute use case. The user then executes the following command:

```
perl ~/dist/scripts/mysummarize | perl ~/dist/scripts/myformat
```

NlpLearner then obtains information from the files produced by the (S-1) subflow of the Execute use case (E-1). NlpLearner then sends to standard out the same information the original Execute use case sent to standard out (although possibly formatted a bit differently). The use case terminates after all output has been sent to standard out.

Subflows

None.

Errors

E-1: If the files don't exist, if other files beginning with "out-" exist in the directory, or if the files have been corrupted, then the results are undefined.

3 Changes

0.2.0:

1. Replaced JLex with JavaCC. JavaCC is a slightly better scanner generator and, unlike JLex, also generates parsers.
2. I have added the ability to pass a parameter to classifiers from the "parameters" file.
3. Before there were five separate java programs and three separate perl scripts. Now there is just one java program and one perl script that executes it.
4. Before, data sets were tokenized twice. Now they are tokenized once.
5. I have added many javadocs. Although far from complete, comments continue to improve.
6. I have cleaned up the code. Although far from complete, code continues to improve (the hierarchical design still needs to improve some, but it is functional).
7. The mysummarize script now works even when there are zero or one out-*-results.txt file. Previously there had to be more than one.
8. I have improved the documentation. Although far from complete, documentation continues to improve (the hierarchical functions still need better documentation).
9. I have partitioned off the classes that depend on weka. All classes that depend on weka are now in the weka package.
10. The hierarchical components are now in there.
11. The parameter file is now much more flexible. In particular, you can now specify "/" style comments and specify lists of values.
12. Via the parameter file, one can now specify any number of levels for hierarchical classification.
13. NlpLearner now orders runs for maximum efficiency.

0.2.1

1. Fixed a bug that made outputting extra files for the hierarchical classifiers take so long that the feature was useless.
2. Fixed a bug that made the output of the results files for the hierarchical classifiers not work with the mysummarize script.
3. Improved the underlying design of the hierarchical classifiers. You will now find these classifiers in the mark.nlp.classifiers package instead of the mark.nlp.hier package.
4. Changed the sample settings scripts to not use an environment variable.
5. Improved the comments of the hierarchical classifiers. It should now be easier to use them based on these comments.

6. Updated documentation in readme.pdf file.

4 Additional Notes

1. I used JavaCC to generate the scanners. If you wish to define your own scanner, I recommend that you use JavaCC 2.0. You should also use one of my scanners (`mark/core/text/*Scanner.jj`) as a guide.
2. The `nlpLearn` script provides a simple interface to the Learner program. Executing the `mark.nlp.apps.Learner` program directly provides a great deal more flexibility. I plan to document this program in the future. For most circumstances, the script suffices. However, if you are curious, you can run the `NlpLearner` program with no parameters to get a usage display. If you are very ambitious, you can also look at the source.
3. Most feature orderers ignore the feature parameter in the parameter file. The only feature orderers that currently use the parameter are subclasses of `TwoLevelFeatureOrderer` and are currently experimental. These orderers attempt to enhance feature selection by also considering how well features distinguish documents within categories.
4. Each classifier documents in its comments what parameter it takes. There are only three types of classifiers at this point. See comments for `mark.nlp.weka.WekaClassifier`, `mark.nlp.hier.ErrorCorrecting`, and `mark.nlp.hier.HierarchyGenerating` for details on parameters.