Mallet Guides for LING 572

Fei Xia

Jan 3, 2007

1 Introduction

This guide is based on my understanding of the package. Since I am a new Mallet user too, who started a few weeks before you, I would greatly appreciate it if you could let me know (1) any errors you notice in this guide, and/or (2) any tricks/bugs/tips in the Mallet package.

1.1 What is Mallet?

Mallet is "an integrated collection of Java code useful for statistical natural language processing, document classification, clustering, information extraction, and other machine learning applications to text".[1]

1.2 Why use Mallet?

We use Mallet in LING 572 for several reasons:

- Mallet includes most machine learning algorithms that we will cover in LING 572, including Naive Bayes, decision tree, MaxEnt, boosting, bagging, finite-state transducer (FST), HMM, and so on. There is no better way of understanding an algorithm than to look at its implementation and run it with real data.
- Mallet includes many basic data types (e.g., SparseMatrix, FeatureVector, ...). You can easily write new code with the existing infrastructure.
- It is an important skill for a computational linguist to be able to use code written by others. The assignments will force you to practice and improve your code-reading skill. Compared to many other NLP pakages, Mallet code is well written and well organized.
- Mallet has been used by CompLing researchers from all over the world. There is also a mailing list for Mallet users. It is good to know the package and become a member of this community.

That being said, it will take you some time to become familiar with the package, so be prepared! I hope that you will find this guide useful. Also, please go to Bill's Mallet Tutorial.

1.3 How to use Mallet?

There are several ways of using Mallet:

1. To use command-line: see http://mallet.cs.umass.edu/index.php/Command_line_tutorial. The commands are easy to use.
2. To write Java code to call Java classes directly: This is a good option for Java programmers.
3. To write code in other languages (e.g., Python) to call Java classes: this could be a good option for people who prefer other languages over Java. However, the interface between the other language and Java could be complicated.

[1] Unless specified otherwise, all the quoted messages come from Mallet's website at http://mallet.cs.umass.edu.

4. To use Jython/Mallethon: see mallet.cs.umass.edu/index.php/Jython:Main.

Among the four options, (1) is simplest, but it allows only simple operations. You will need to use other options for your assignments except for Hw1.

Among (2), (3), or (4), I would highly recommend (2). If you want to use (3) or (4), you should talk to me first.

1.4 Where to start?

Start with Hw1 and go to Bill's Mallet tutorial if possible.

2 Important resources

Here are the urls of some important resources:

Mallet :

- Mallet main page: http://mallet.cs.umass.edu/index.php/Main_Page
- Classes in Mallet: http://mallet.cs.umass.edu/mallet/javadoc/index.html
- Command-line tutorial: http://mallet.cs.umass.edu/index.php/Command_line_tutorial.
- Mallethon: http://mallet.cs.umass.edu/index.php/Jython:Main.

Java in general :

- Java API: http://java.sun.com/j2se/1.4.2/docs/api/
- Java tutorial: http://java.sun.com/docs/books/tutorial/
- Java development platforms (debuggers): Here are some debuggers you can download for free.[2]
  - Eclipse: http://www.eclipse.org/
  - Netbeans: http://www.netbeans.org/index.html
  - gdb: http://gnu.paradoxical.co.uk/software/gcc/java/gdb.html
  - drjava: http://www.drjava.org
- Jython: http://www.jython.org/Project/index.html

3 Main directories in the Mallet package

Mallet is installed under /opt/mallet-0.4 on Pongo. Let's call this directory $MalletDir.

3.1 Subdirectories of $MalletDir

It has several subdirectories:

- **src/**: source code
- class/: compiled classes. Its structure is parallel to that of src/.
- lib/: all the *.jar files

[2] If you prefer to use other debuggers, please let me know. I'd like to compile a list of good Java debuggers.

- doc/: three documents about the Mallet packages.
- html/: automatically generated java class documents.
- bin/: the command-line tools as described in mallet.cs.umass.edu/index.php/Command_line_tutorial. They call java code defined under src/edu/umass/cs/mallet/base/classifier/tui/.
- scripts/: Python code that tests CRF

Among all the subdirectories, src/ is the most relevant.

3.2 Source code

Almost all the relevant source code are stored under $MalletDir/src/edu/umass/cs/mallet/base. Let's call that directory $baseDir.

Here are the subdirectories under $baseDir:

1. **classify/**: Supervised classification algorithms, including Naive Bayes, MaxEnt, DecisionTree, Winnow, AdaBoost, Bagging, and so on.
2. **types/**: Important data-struture-type classes like Instance and FeatureVector
3. **pipe/**: objects that transform instances, mostly for feature extraction.
4. maximize/ and minimize/: various optimization algorithms: Conjugate Gradient, BFGS, etc.
5. extract/: Wrapper classes for performing extraction. These use classes like CRF to perform extraction from source text.
6. cluster/: clustering algorithms
7. fst/: transducer-based algorithms: FST, CRF, HMM, MEMM, etc.
8. util/: various tools. util/tests/ includes some test code.

Among them, (1) is most relevant. In order to understand how classifiers work, you need to look at some code under (2)-(3). You rarely need to look at code under (4)-(8) for this course.

4 Important classes

When you read the source code, there is no need to go through every line; instead, just focus on the major memebers and methods in those classes. In this section, I list some important Mallet classes.[3]

4.1 Under $baseDir/classify

All classification techniques in MALLET are implemented as two classes: a trainer and a classifier. For example, MaxEntTrainer.java is the trainer and MaxEnt.java is the classifier.

- The trainer takes the training data as input and creates a classifier as output. Each trainer is a subclass of ClassifierTrainer.
- A classifier that holds the parameters set learned during training. It applies those parameters to an Instance to produce a classification of the Instance. Each classifier is a subclass of Classifer.

Almost all the files under this directory are relevant.

[3]Much of the text in this section comes from comment lines in the Mallet Java source code.

4.1.1 Classifier

It is the parent class of all classifiers. Its abstract methods classify() should be defined in each of its subclasses. The print() function is also very useful, but it is defined in only a few classifiers.

```
classification classify(Instance instance); // classify an instance
void print();  // Outputs human-readable description of classifier
               //  (e.g., list of weights, decision tree) to System.out
```

4.1.2 ClassifierTrainer

It is the parent class of all trainer classes. Its abstract method train() should be defined in each of its subclasses.

```
Classifier train(InstanceList trainingSet, InstanceList validationSet,
                 InstanceList testSet, ClassifierEvaluating evaluator,
                 Classifier initialClassifier);
```

4.1.3 Classification

It stores the result of classifying a single instance. It contains the instance, the classifier used, and the labeling the classifier produced. It also includes methods for comparing the correct (true) label contained in the target field of the instance with the one produced by the classifier.

4.1.4 Trial

It is a convenience class for running an InstanceList through a Classifier and storing the Instancelist, the Classifier, and the resulting Classifications of each Instance. Also has methods for computing f1 and accuracy over the classifications. One commonly used constructor is
Trial(Classifier c, InstanceList ilist).

4.1.5 Command-line classes

They are all defined under $baseDir/classify/tui.

4.2 Under $baseDir/types

This directory defines the important data types used in classifiers, pipes, and other classes.

4.2.1 Instance

An Instance corresponds to one row in the attribute-value table. It has four major members (the first two are the most important members):

- data: it holds the data represented by the instance. It will eventually be represented as a feature vector.
- target: it is a label associated with the instance.
- name: is a short identifying name for the instance (such as a filename),
- source: is human-readable source information, (such as the original text).

4.2.2 InstanceList

A list of Instances. For instance, the training data is an InstanceList, so is the testing data or the validation data. The most common way of adding instances to an InstanceList is through the add(PipeInputIterator) method. One can use load() to load instanceList from a file (See Section 6.1.3).

4.2.3 FeatureVector

A FeatureVector is a SparseVector with a new member, "dictionary", which stores the names of all the features. A FeatureVector stores the values for the features in an Instance. The data field in an Instance will eventually be presented as a FeatureVector.

4.2.4 FeatureSelection

FeatureSelection includes an Alphabet (which corresponds to a set of features), and a BitSet that shows the subset of features that are selected.

```
Some Constructors:

FeatureSelection(Alphabet dictionary);
FeatureSelection(Alphabet dictionary, java.util.BitSet selectedFeatures);
```

4.2.5 Labeling

Labeling is an interface (not a class) that represents a distribution over possible labels for an instance. It stores a value for each label. Conceptually, a Labeling is a ranked list of labels with associated values. Labels is a class that implements Labeling.

4.3 Under $baseDir/pipe

A pipe modifies instances that are handed to it: A pipe reads from and writes to fields in the Instance when it is requested to process the instance. It is up to the pipe which fields in the Instance it reads from and writes to, but usually a pipe will read its input from and write its output to the "data" field of an instance.

A pipe includes two Alphabets: one for the symbols (feature names) in the data field, and the other for the symbols in the target field.

4.3.1 Pipe

A Pipe is an abstract class that is the parent class of all pipes. Its main method pipe() should be defined in each of its subclasses.

```
Instance pipe(Instance carrier);
      Process an Instance.

Instance pipe(java.lang.Object data, java.lang.Object target,
              java.lang.Object name, java.lang.Object source,
              Instance parent, PropertyList properties)
      Create and process an Instance.
```

4.3.2 Subclasses of Pipe

Pipe has many subclasses, and it is likely that you will write your own. Here are some subclasses that could be useful to you.

1. Target2Label: Convert object in the target field into a label in the target field
2. Filename2CharSequence: Given a filename contained in a string, read in contents of file into a CharSequence.
3. CharSequence2TokenSequence: character sequence ⇒ a TokenSequence.
4. TokenSequence2FeatureSequence: a TokenSequence ⇒ a FeatureSequence
5. FeatureSequence2AugmentableFeatureVector: a FeatureSequence ⇒ a FeatureVector
6. Token2FeatureVector: convert the property list on a token into a feature vector
7. Noop: A pipe that does nothing to the instance fields but which has side effects on the dictionary.
8. SerialPipes: Convert an instance through a sequence of pipes.
9. InstanceListTrimFeaturesByCount: it selects features by feature counts
10. PrintInput: Print the data field of each instance.
11. PrintInputAndTarget: Print the data and target fields of each instance.
12. PrintTokenSequenceFeatures: Print properties of the token sequence in the data field and the corresponding value of any token in a token sequence or feature in a featur sequence in the target field.

You can define a serial of pipes (see SerialPipe) to process an instance. For instance, the SerialPipe that consists of (2)-(6) will convert a file into a feature vector.

4.4 Other subdirectories under $baseDir

On rare occasions, you will need to look at some source code under other subdirectories.

5 Main steps of using a ML method to solve an NLP task

1. Convert the problem into a classification problem (if needed)
2. Get data: split it into training/test/dev
3. Define feature templates
4. Process data and create attribute-value table and select features before training (optional). (see base/pipe and base/extract)
5. Training: (see base/classifier/*Trainer.java, base/maximize/, base/minimize/, base/types/* to calculate gain, etc.)

6. Classifying: (see base/classifier/*)

7. Calculate classification results: (see base/classifier.java, base/trial.java, etc.)

Out of the seven stages, Mallet provides classes for (3)-(7). Your homework will mostly be focused on (a) writing code for Stages (3)-(4), and (b) reading code used in Stages (5)-(6).

6 Sample code

Here are some sample code that I wrote, based on some source code in Mallet. The comments after "%" are mine.

6.1 Stage 4: Processing data to create attribute-value table

6.1.1 Processing an instance

Normal steps: Class1 and Class2 can be the same or different.

The typical code for pipe() function in Pipe looks like the following:

```
public Instance pipe (Instance carrier){
 %%%% get the current data field.
 Class1 ts = (Class1) instance.getData();

 %%%% init the new data field
 Class2 ret = new Class2 ....

 %%% from ts, sett a new data ret;
 ......

 %%%% set the data field to the instance
 instance.setData(ret);

 %%%% return the instance
 return instance;
}
```

```
The following pipe() function comes from TokenSequenceLowercase.java.
It lowercases each token in the data field.

public Instance pipe (Instance carrier)
{
  TokenSequence ts = (TokenSequence) carrier.getData();

  %%%% process and set the data
  for (int i = 0; i < ts.size(); i++) {
     Token t = ts.getToken(i);
     t.setText(t.getText().toLowerCase());
  }
```

```
  %%%% return the instance
  return carrier;
}
```

6.1.2 Processing an InstanceList

The following code is modified from Csv2Vectors.java.

```
%%%%%%% step 1: get a Pipe with either 1(a) or 1(b)
%% 1(a) get a Pipe from an existing InstanceList
Pipe instancePipe = InstanceList.getPipe();

%% 1(b) create a new Pipe
Pipe instancePipe = new SerialPipes (new Pipe[] {
    new Target2Label (), new Noop (), ....});

%%%%%% step 2: init InstanceList
InstanceList ilist = new InstanceList (instancePipe);

%%%%%  step 3: process InstanceList
FileReader fileReader = new FileReader (inputFile.value);

%%% add instances from the input file. You would need to change the
%%%  PipeInputIterator if the input file is of a different format.
ilist.add (new CsvIterator (fileReader, Pattern.compile(lineRegex.value),
                 dataOption.value, labelOption.value, nameOption.value));


%%%% step 4: write out InstanceList
ObjectOutputStream oos =
  new ObjectOutputStream(new FileOutputStream(outputFile.value));

oos.writeObject(ilist);
oos.close();
```

6.1.3 Loading and printing an InstanceList after converting the data into a feature vector

The following code is modified from Vectors2Info.java

```
InstanceList ilist = InstanceList.load(File.value);
int numInstances = ilist.size();

for (int i = 0; i < ilist.size(); i++) {
   %%%% get the instance
   Instance inst = ilist.getInstance(i);
   ...
```

```
}

%%% get the number of features
int numFeatures = ilist.getDataAlphabet().size();

%% get the feature vector for an instance
FeatureVector fv = (FeatureVector) inst.getData ();

for (int fvi=0; fvi<numFeatures; fvi++){
  %%%% fvi is the index of a feature
  %%%% feat is the fvi-th feat
  object feat = dataAlphabet.lookupObject(fvi);

  %%%% get the value for the fvi-th feat in this instance
  double feat_value = fv.value(fvi);

  ....
}
```

6.2 Stages 5-7: Training, test, and evaluation

```
The following code is modified from Vectors2Classify.java.

%%%%% Step 1: Reading an InstanceList
InstanceList train_ilist =  InstanceList.load (new File(trainFile.value));
InstanceList test_ilist =  InstanceList.load (new File(testFile.value));

%%%%% Step 2: Initializing a trainer
ClassifierTrainer trainer = new NaiveBayesTrainer(...);

%%%%% Step 3: Training
Classifier classifier = trainer.train(train_ilist);

%%%%% Step 4: Testing on the training data and test data
Trial trainTrial = new Trial (classifier, train_ilist);

Trial testTrial = new Trial (classifier, test_ilist);

%%%%% Step 5: Outputing the classifier

%%% 5(a): output in binary format
ObjectOutputStream oos =
   new ObjectOutputStream(new FileOutputStream (filename));
oos.writeObject (classifier);  // binary format.
oos.close();

%%% 5(b): output the classifier in the human-readable format
classifier.print ();  // print() is implemented in some classifiers
```

```
%%%%%% Step 6: output the decoding result
printTrialClassification(testTrial);
```

7 Mallet FAQ

The following is a list of questions I had when I studied the code. The answers come from various sources.[4]

```
1. Q: Are there any good documents to explain how the code work?

   A: Not currently, but it's on the to-do list for the Mallet team at UMass.

2. Q: Can training and testing be done separately using current command-lines?

   A:  No. You can save a classifier with --output-classifier,
        but currently there is not a command-line to read a saved classifier.
        Nevertheless, it should be pretty easy to write code that does that.

3. Q: How to see the output file of training (e.g., weights)

   A:  When running vectors2classify,
       use "--output-classifier model_output_file" option
         to specify an output file.

      Then use "classifier2info --classifer model_output_file" to
       view model parameters (e.g., features and feature weights).

      Note: the option name ``--classifier'' is misspelled as
       ``--classifer'' in the
       Java code. Also, some classifiers do not define print() method
       so classifier2info will not work.

4. Q: How to view the output for test stages (system output)?

   A: Use the ``--report test:raw'' option. Ex:
       vectors2classify --input text.vectors --training-portion 0.7
         --report test:raw

5. Q: How to get the topN results from test stages?

   A:  ``--report test:raw'' will make the system
        to output the score for each prediction,
        but not an N-best list. However, since the predictions are sorted
        into a ranked list,
        it is trivial to write some code to get the topN predictions.
```

[4]Special thanks to Aron at UMass, who replied to many of my Mallet questions.

```
6. Q: How many learners are implemented in Mallet?

   A: Many: NaiveBayes, MaxEnt, DecisionTree, C4.5, Winnow,
            Bagging, AdaBoostM2, AdaBoost, etc.
            Just see files under base/classifiers/.

7. Q: How to specify certain non-model parameters for training
       (e.g., iteration number, etc)?

   A: Some can be specified in the command line.
       e.g., --trainer ``new AdaBoostM2Trainer(new DecisionTreeTrainer())''
       Others cannot. The easiest way to specify
       those parameters is to write Java code to call
       trainer constructors directly.

8. Q: Is feature selection done internally? If so, how is it done
      (e.g., value of threshold used for feature selection)?

   A: The default is to include all the features.
      You will be asked to write your own code to select features
        in Hw3. Some trainers (e.g., MaxEnt) have
        trainWithFeatureInduction() methods which do feature selection
        during the training.

9. Q: How to use the classifiers for sequence labeling (e.g., POS tagging)?

    A: It is not implemented yet. You will be asked to
       implement this in one of your assignments.
```

8 Final Notes

Some final notes:

- When you read the source code, do **not** go through every line; instead, you should focus on the major class members and methods. When you write Java code to call Mallet classes, it will save you a lot of time if you start with existing Mallet classes and modify them.
- Please feel free to discuss with your classmates any problems you run into. You can also email the TA or me, but please do not ask me to debug the code for you. For debugging, you should install a good Java debugger, and some of them are listed in Section 2.
- If you plan to email the Mallet mailing list for Mallet-related questions, please ask your classmates and us first. We might know the answers.
- Finally, if you find any bugs or learn any good tricks in Mallet, please let me know. I would like to pass that information to other students and the Mallet community.

Have fun with Mallet. It is a nice package.