Professionalization of UMass Project “MALLET” (PALLET)

Prepared by
Sharath Jagannath
&
Pralabh Kumar

Under the guidance of
Dr. Bhavani Thuraisingham

Table of Contents

Introduction:

This work which we call A Professionalization of Umass project “MALLET” (PALLET) aims at providing a well structured documentation and code snippets that describes MALLET's features. It also contains an extension to MALLET that abstracts the Machine Learning algorithms that suffice the requirements of many applications. Further, we provide a framework which can be used to write Semantic Web applications using MALLET algorithms and we consider this is as the important work which we have achieved.

MALLET:

MALLET is a Java-based package for statistical natural language processing, document classification, clustering, topic modeling, information extraction, and other machine learning applications to text.

MALLET includes sophisticated tools for document classification: efficient routines for converting text to "features", a wide variety of algorithms (including Naïve Bayes, Maximum Entropy, and Decision Trees), and code for evaluating classifier performance using several commonly used metrics.

In addition to classification, MALLET includes tools for sequence tagging for applications such as named-entity extraction from text. Algorithms include Hidden Markov Models, Maximum Entropy Markov Models, and Conditional Random Fields. These methods are implemented in an extensible system for finite state transducers

Topic models are useful for analyzing large collections of unlabeled text. The MALLET topic modeling toolkit contains efficient, sampling-based implementations of Latent Dirichlet Allocation, Pachinko Allocation, and Hierarchical LDA.

In addition to sophisticated Machine Learning applications, MALLET includes routines for transforming text documents into numerical representations that can then be processed efficiently. This process is implemented through a flexible system of "pipes", which handle distinct tasks such as tokenizing strings, removing stopwords, and converting sequences into count vectors.

Note : Above description on MALLET is obtained from its homepage.

MALLET Installation:

Step 1: Download MALLET from its homepage.
Step 2: MALLET is an ANT project, download ant.
Step 3: Building Mallet

```
cd ${mallet directory}
ant
```

MALLET Directory Structure

Mallet follows the standard directory structure which is being followed by all the professionalized software.

Mallet directory structure contains following folders

1. Bin: It contains all the Batch files.
2. Src: It contains all the source files.

3. Class : It contains all the java compiled class files
4. Dist: It contains jar files.
5. Lib: It contains all the jar files.
6. Sample-data: It contains all the sample-data on which MALLET can be tested.
7. Doc: It is an empty in mallet-2.0.5.But is should contain all the important documentation.

Data Pre-processing:

MALLET uses Instances to represent data, and pre-processing is the process of transforming the data to instance list.

MALLET instance is composed of the following 4 fields:

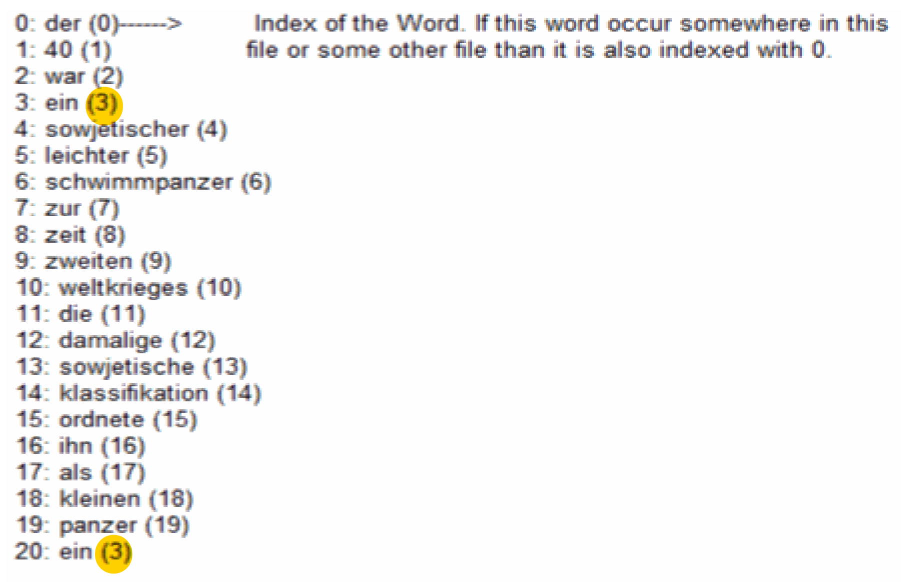
- **Name:** This field acts as the Name of the instance and use for the identification of instances.
- **Label:** Label is primarily used to classify the instances. Labels represent the classes in the classification module.
- **Data:** The data is generally a Feature Vector or Feature Sequence (for example sequence of words).Classifier does the classification of the instances on the basis of the data of instances and provide the Label to it(if it is unclassified data)
- **Source:** It tells the information about the source of the Mallet Instance.

MALLET's Important Data Pre-processing Components

Feature Sequence:

Mallet provides different types of sequences which are basically used to store Objects .All the sequences in MALLET implements Sequence interface.

FeatureSequence is used to store the Objects of the same class. It is mutable and can expand as new objects are added.For eg if one stores all the words of the files in a FeatureSequence ,then FeatureSequence would be



```
0: der (0)------>        Index of the Word. If this word occur somewhere in this
1: 40 (1)               file or some other file than it is also indexed with 0.
2: war (2)
3: ein (3)
4: sowjetischer (4)
5: leichter (5)
6: schwimmpanzer (6)
7: zur (7)
8: zeit (8)
9: zweiten (9)
10: weltkrieges (10)
11: die (11)
12: damalige (12)
13: sowjetische (13)
14: klassifikation (14)
15: ordnete (15)
16: ihn (16)
17: als (17)
18: kleinen (18)
19: panzer (19)
20: ein (3)
```

FeatureVector:

It is a subset of class “Alphabet”. An “Alphabet” class represents the mapping between integers and objects. Integers are assigned consecutively ,starting at zero, as objects are added to the Alphabet. one cannot delete objects from the alphabet and thus the integers are never used. When classifying documents using MALLET, all the unique words in a document would be unique entry in the alphabet with a unique integer is associated with it. FeatureVectors use this integer part inorder to represent the subset of Alphabet. FeatureVector is represented as a SparseVector.A location in a FeatureVector represents the index in Alphabet.For eg

swahili(4)=1.0
ngoma(5)=1.0
means(6)=1.0
dance(7)=1.0------------->Number of times index(7) or Word(dance) occurs in the data.

InstanceList:

It contains Mallet instances which are typically used for training and testing of machine learning algorithm. All the instances in the Mallet list must have passed through the same sequence of pipes and hence share the same data and target alphabets.

Pipe:

Mallet uses different types of pipes in order to preprocess the data. For eg Mallet provides TokenSequenceLowerCase which converts the in coming tokens to lower case.Pipe is an abstract superclass of all these pipes. Some of the important pipes which Mallet provides.

Input2CharSequence.
CharSequence2Token-Sequence.
TokenSequenceLowercase.
TokenSequenceRemoveStopwords.
TokenSequence2FeatureSequence.
Target2Label.
FeatureSequence2FeatureVector.
PrintInputAndTarget.

Data importing (pre-processing) is achieved by using a series of pipes. Pipes take as input instance List and modify the data field based on the series of pipes specified. The pipes which are used in the project

Input2CharSequence:

It is a Pipe that can read data from various kinds of sources and convert the given input into character sequence.

CharSequence2Token-Sequence:

It is pipe that tokenizes a character sequence. It expects a character sequence and converts into tokens. For e.g. following data

“Der T-40 war ein sowjetischer leichter Schwimmpanzer “ is converted to:

Token#0:Der span[0..3]

```
Token#1:T  span[4..5]
Token#2:40  span[6..8]
Token#3:war  span[9..12]
Token#4:ein  span[13..16]
Token#5:sowjetischer  span[17..29]
Token#6:leichter  span[30..38]
Token#7:Schwimmpanzer  span[39..52]
```

TokenSequenceLowercase:

It is pipe which converts all the tokens received through previous pipes to lower case. For e.g.

```
Token#0:Der  Span[0..3]          Token#0:der  span[0..3]
Token#1:T  span[4..5]             Token#1:t  span[4..5]
```

TokenSequenceRemoveStopwords:

It is the pipe which is used to remove the stop words from the tokens. MALLET contains a list of stopword(which are basically useless). If tokens contains these stopwords than that tokens are removed from the

TokenList. For e.g. the some of the stopwords english language:

```
“a",
"able",
"about",
"above",
"according",
"accordingly",
"across",
"actually",
"after",
"afterwards",
"again",
"against",
"all"
```

TokenSequence2FeatureSequence:

It converts the token sequence to feature sequence.It indexes each token.

```
0: der (0)------> Index of the Word. If this word occur somewhere
1: 40 (1)        in this file or some other file than it is also
2: war (2)        indexed with 0.
3: ein (3)
4: sowjetischer (4)
5: leichter (5)
6: schwimmpanzer (6)
7: zur (7)
8: zeit (8)
```

9: zweiten (9)

Target2Label:

It Convert Object in the target field into the label. These Labels are used as the classes at the time of classification.

FeatureSequence2FeatureVector:

It Converts a data field from Feature Sequence to Feature vector. This class does not insist on getting its own Alphabet because it rely on getting from the FeatureSequence input.For eg:

Feature vector counts the number of times index 0 occurred in a FeatureSequence
[0]=11.0
Note : The index 0 is related to the word “der”. So it means that
“der” occurred 11 times in a feature sequence.

Data Classification:

MALLET supports rich set of Document Classification Algorithms:

- Naïve Bayes.
- Maximum Entropy.
- C45 Decision Tree.
- Boosting Algorithms and many more.

Every Classifier is implemented as a Trainer and Classifier, and performs classification only on the instances processed with the pipe (import data pipe) associated with this classifier. Instances are generally Feature Vectors. Trainer is responsible for training data and the Classifier performs the classification based on the trained data.

Every Trainer consists train () method, which is used to train the data. A call to this train() method instantiates the Classifier which can classify the data based on its training. Further, all Trainers and Classifiers can be found in cc.mallet.classify. Also, each trainer and classifier is implemented as an object in the MALLET. All the trainer mentioned above extends *ClassifierTrainer* class and the trainers mentioned extends Classifier respectively, both of which can be found at **cc.mallet.classify.**

Complete Documentation of data-import and classification can be found here.

Professionalization:

MALLET itself is a well structured code, but lacks proper documentation, an useful and mature library like MALLET can be acknowledged only if it has a good document which acts as a references to its interface. Our work was a small step towards making MALLET useful in the Open Source Community. Documentation included writing description to the MALLET interfaces and code snippets showing how to use the interfaces.

Most important extension to the MALLET which we undertook was to build the framework to extend MALLET as an algorithm to Semantic Web Framework. As a by-product of this work, we provided an abstract factory that encapsulates the creation of MALLET's object and also provided a facade to simplify the interface making it easier to understand and use. This extension supports most of the common tasks for data import and classification. We named our project PALLET (A

Professionalization of the UMass project "Mallet"). We built our Pallet as a maven project and thus adhering to modern software practices and handling the dependencies automatically. Currently, Pallet supports Data import and Classification of RDF/XML data.

Pre-processing includes, conversion of RDF/XML data to MALLET Instances. We use Jena to retrieve RDF data, further we consider, rdf resources as the data field of the Instance Object, Predicate values as the class labels and rdf objects as object field. This Instance List is processed through the pipes converting them to feature vector.

Feature vector generated is used to construct the training model, and we use the factory we created to instantiate the training model, and the trained model is persisted as a RDF data for incremental learning.

Query data should be processed in the same way and the classified query data is written back as a RDF data with the reification statement added to the query data. Reified statement provides the confidence value for the data being classified.

Our framework was robust enough to implement the mallet algorithms as Blackbook algorithms.

MALLET Documentation:

Mallet is huge toolkit. It contains lots of files ,packages and thousands of lines of code. But the problem with MALLET is that it is not properly documented. One has to spend a huge amount of time to understand the working of classes and packages in MALLET. Therefore it is very tough to use MALLET for the purpose of Classification ,Sequence Tagging etc.

That's why documentation of MALLET is the cardinal part of,our final goal which is professionalization of MALLET . Some of the MALLET packages ,which is used in classification, is properly documented so that even a naïve user can understand them.The whole goal to document MALLET is to provide the convienece to the user, to understand some of the complexe classes of MALLET.

1)cc.mallet.pipe :Package which is used in the processing of input data is properly documented

• Input2CharSequence • CharSequence2TokenSequence • TokenSequenceLowercase • TokenSequenceRemoveStopwords • TokenSequence2FeatureSequence • Target2Label • FeatureSequence2FeatureVector • PrintInputAndTarget • CharSequenceReplace • CharSequenceRemoveHTML	• StringAddNewLineDelimiter • CharSequenceRemoveUUEncodedBlocks • StringList2FeatureSequence • FeatureSequenceConvolution • FeatureVectorConjunction • TokenSequenceRemoveNonAlpha • TokenSequenceNGrams • SGML2TokenSequence • Target2FeatureSequence • Target2LabelSequence	• PrintInput • PrintTokenSequenceFeatures • SaveDataInSource • SelectiveSGML2TokenSequence • SimpleTaggerSentence2StringTokenization • Token2FeatureVector • TokenSequence2TokenInstances • AugmentableFeatureVectorAddConjunction • BranchingPipe • Directory2FileIterator

• MakeAmpersandXMLFriendly • FeatureSequence2AugmentableFeatureVector • AugmentableFeatureVectorLogScale • CharSequence2charngrams • CharSequenceLowercase • CharSubsequence • LineGroupString2TokenSequence • Filename2CharSequence • CharSequenceArray2TokenSequence • Csv2Array	• TokenSequenceParseFeatureString • TokenSequenceMatchDataAndTarget • TokenSequence2FeatureSequenceWithBigrams • TargetRememberLastLabel • SourceLocation2TokenSequence • FeatureCountPipe • FeatureValueString2FeatureVector • Pipeutils	• Serial Pipes • TokenSequence2FeatureVectorSequence • FeatureVectorSequence2FeatureVectors • FilterEmptyFeatureVector • InstanceListTrimFeaturesByCount • Noop • Pipe • Array2FeatureVector • Csv2FeatureVector • TargetStringToFeatures • SimpleTaggerSentence2TokenSequence • SimpleTokenize

PALLET:

PALLET Overview

Currently, PALLET is a maven project which consists of four sub-projects.

1. Pallet-Data: provides pre-processing functionality for RDF data.
2. Pallet-Classify: provides classification functionality.
3. Pallet-Test: Creates an application to demonstrate Pallet-Data and Pallet Classification.
4. Pallet-Blackbook: Independent of all the mentioned sub-projects, converts MALLET algorithms as Blackbook algorithms.

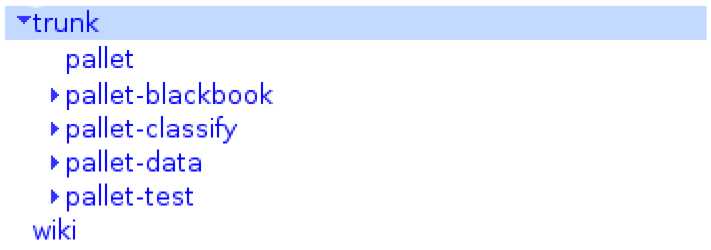
PALLET Installation

Requirements:

1. Subversion: In most cases svn should be already installed on UNIX/Linux box, if not download it from here.
2. Maven: If not ubuntu install maven using sudo apt-get install maven2, if not download and install a copy from here.

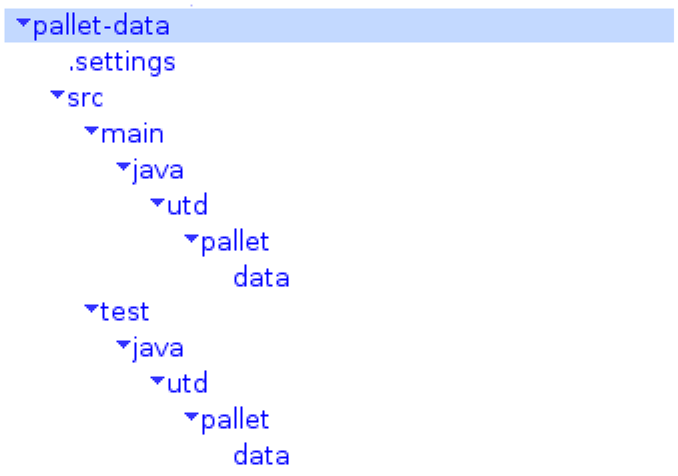
Pallet Directory Structure:

Pallet project adheres to maven directory structure. Root of the directory structure contains pom file which defines the four above mentioned modules: Pallet-Data, Pallet-Classify, Pallet-Test and Pallet-Blackbook.



```
trunk
    pallet
    pallet-blackbook
    pallet-classify
    pallet-data
    pallet-test
  wiki
```

Each module of the pallet workspace uses the following packaging structure:



```
pallet-data
    .settings
  src
    main
      java
        utd
          pallet
            data
    test
      java
        utd
          pallet
            data
```

Building PALLET :

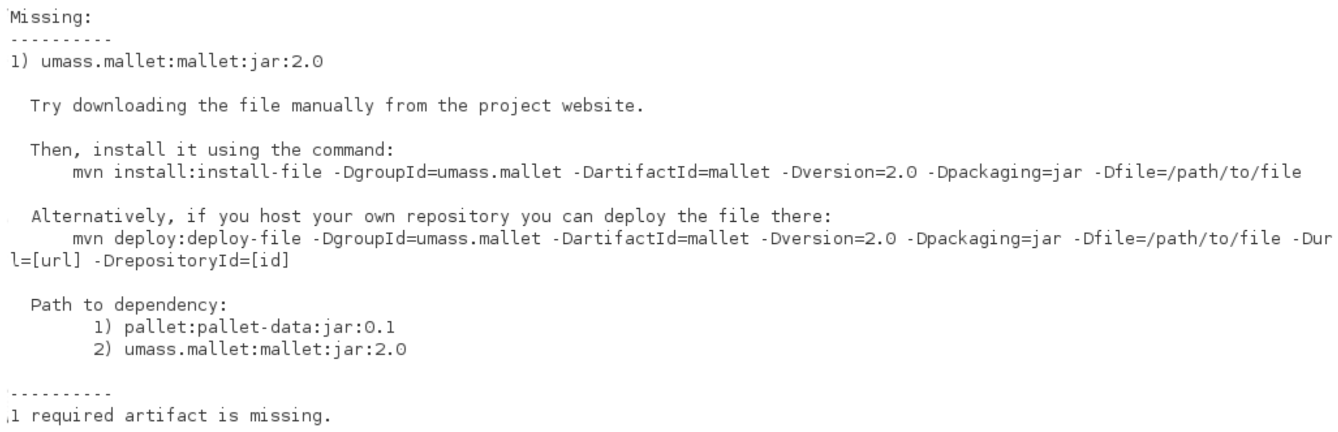
Step 1: Checkout a copy of the source code using the following command:

svn checkout *http*://pallet.googlecode.com/svn/trunk/ pallet-read-only

Step 2: Pallet is dependent on non-maven mallet, download mallet library from here

Step 3: cd to pallet directory(root directory of the project) and build the project using mvn clean install.

Step 4: If you are building pallet for the first time then it is likely to encounter the following build error:



```
Missing:
----------
1) umass.mallet:mallet:jar:2.0

  Try downloading the file manually from the project website.

  Then, install it using the command:
      mvn install:install-file -DgroupId=umass.mallet -DartifactId=mallet -Dversion=2.0 -Dpackaging=jar -Dfile=/path/to/file

  Alternatively, if you host your own repository you can deploy the file there:
      mvn deploy:deploy-file -DgroupId=umass.mallet -DartifactId=mallet -Dversion=2.0 -Dpackaging=jar -Dfile=/path/to/file -Dur
l=[url] -DrepositoryId=[id]

  Path to dependency:
        1) pallet:pallet-data:jar:0.1
        2) umass.mallet:mallet:jar:2.0

----------
1 required artifact is missing.
```

Step 5: Install the missing artifact using:

mvn install:install-file -DgroupId=umass.mallet -DartifactId=mallet -Dversion=2.0 -Dpackaging=jar -Dfile=*/path/to/mallet/jar/of/step 2*

Step 6: Use mvn clean install to complete the installation.

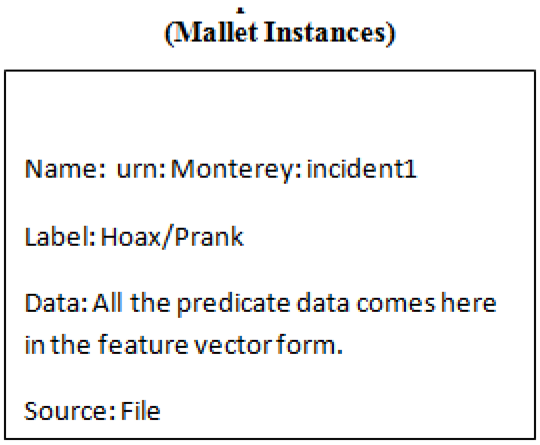
Pallet Data:

Purpose of Pallet Data

It Provides interfaces to MALLET Data Import module

Interfaces to handle RDF/XML making it possible to handle semantic data.

Output Data Format



(Mallet Instances)

Name: urn: Monterey: incident1

Label: Hoax/Prank

Data: All the predicate data comes here in the feature vector form.

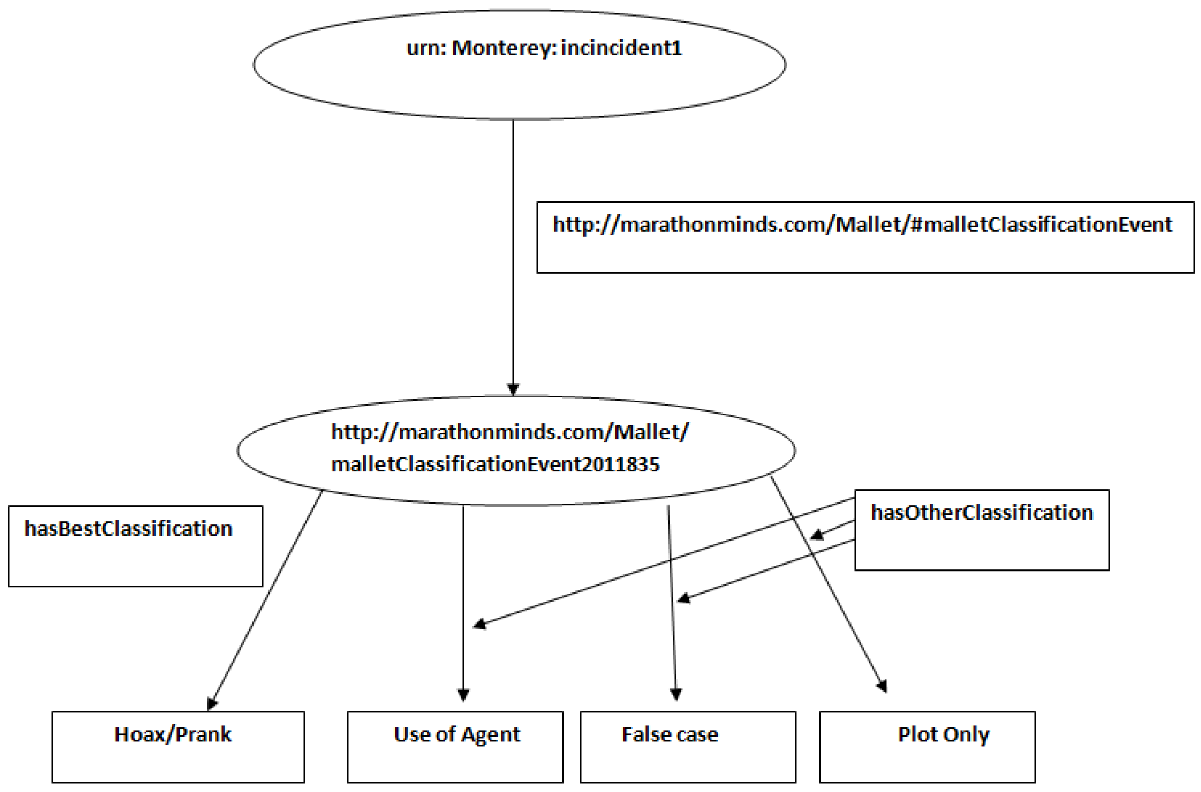
Source: File

Once data is preprocessed it goes to classification state.

The final outcome of the data is also in RDF format. The unlabeled data is labeled with all the classes which are provided by the classifier. But ,it also provides confidence prediction value for each class. For eg we have event : urn: Monterey: incident1 with its description but is currently untagged. The classifier can divide the data into following categories (which was provided to the classifier at the time of training)

1. Hoax/Prank.
2. Use of agent.
3. False case.
4. Plot only.

So the final outcome will be the resource ”urn: Monterey: incident1” which is tagged with all these classes as its objects and with corresponding confidence value.


urn: Monterey: incincident1
http://marathonminds.com/Mallet/#malletClassificationEvent
http://marathonminds.com/Mallet/
malletClassificationEvent2011835
hasBestClassification
hasOtherClassification
Hoax/Prank
Use of Agent
False case
Plot Only

MalletDataImport

The purpose of the creation of MalletDataImport is to let the user to understand the functionality of the some of the most important preprocessing pipes of the MALLET.It demonstrates the functionality of the pipes used by **BuildPipeDiffPipe.**After knowing the functionality of the different preprocessing units of MALLET ,a user can use them in its own application.

BuildPipe

Mallet uses different types of pipes in order to preprocess the data.This module further demonstraes the functionality of the some of the different pipes of Mallet.The whole purpose to demonstrate the functionalities of different type of pipes of Mallet to the user is to make user more familiar with them .So that user can easily use them in its own application.BuilPipe demonstrates the functionalties of the following files

Input2CharSequence
CharSequence2TokenSequence
TokenSequenceRemoveStopwords
TokenSequence2FeatureSequence
Target2Label
FeatureSequence2FeatureVector
Printinputandlabel

BuildPipeDiffPipe

BuildPipeDiffPipe is similar to BuildPipe in functionality and purpose.But it is showing the functionalites of the different pipes of MALLET.

CharSequenceReplace
CharSequenceRemoveHTML
MakeAmpersandXMLFriendly
FeatureSequence2AugmentableFeatureVector
AugmentableFeatureVectorLogScale
Printinputandlabel

RDF2MalletInstances and RDFUtils

These two modules are the heart of Pallet-Data. They provide the whole functionaltiy to convert RDF/XML data into Mallet instances.
The Algorithm which is followed

1. Iterate through each resource of the RDF
2. If Resource is not uri resource or Blank node then find the Original Resource of the Blank node.
3. List all the properties of the Resource.
4. If the object value of the property is literal then add it to the String.
5. Store the data into HashMap having the key as the Resource.
6. If while iterating through the resources we get the same resource (because resource of blank node) which is already in the HashMap then append the new data.
7. Finally,we add each instance to the List , process the instance through the Pipe and add all the instances to the instanceList.

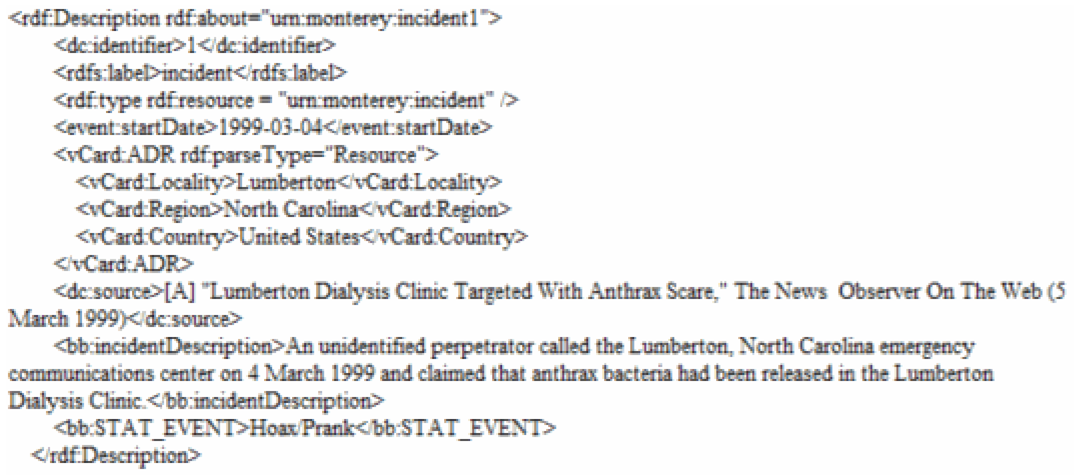
Moreover there are certain utlity features provided by RDFUtils module

1. Rdf2JenaModel: This feature or method convert the data in the given filename into jena model into RDF/XML format.
2. DeserializeJenaModel: This method convert the RDF/XML given in string format to Jena model.
3. SerializeJenaModel: This method serealizes jena model into String.

All these features of RDFUtils can be used by user for its own purpose.

Note: Currently Pallet-Data is only supporting Montery.rdf file format.

We have the input data in the RDF form, which we have to convert:



```
<rdf:Description rdf:about="urn:monterey:incident1">
    <dc:identifier>1</dc:identifier>
    <rdfs:label>incident</rdfs:label>
    <rdf:type rdf:resource = "urn:monterey:incident" />
    <event:startDate>1999-03-04</event:startDate>
    <vCard:ADR rdf:parseType="Resource">
      <vCard:Locality>Lumberton</vCard:Locality>
      <vCard:Region>North Carolina</vCard:Region>
      <vCard:Country>United States</vCard:Country>
    </vCard:ADR>
    <dc:source>[A] "Lumberton Dialysis Clinic Targeted With Anthrax Scare," The News  Observer On The Web (5
March 1999)</dc:source>
    <bb:incidentDescription>An unidentified perpetrator called the Lumberton, North Carolina emergency
communications center on 4 March 1999 and claimed that anthrax bacteria had been released in the Lumberton
Dialysis Clinic.</bb:incidentDescription>
    <bb:STAT_EVENT>Hoax/Prank</bb:STAT_EVENT>
  </rdf:Description>
```

This instance is converted to mallet instance having the following properties:

Name: urn:monterey:incident1.
Label: Hoax/Prank.
Data: All the predicate data comes here in the feature vector form.
Source: File.

Pallet Classify:

Pallet-Classify is a wrapper for the MALLET's Classification algorithms extended to provide API to support Semantic Web application. MALLET provides the interface “Classifier” which is common to all of its classifiers, we reuse the same interface in our module, further we separate the Classifiers to one which supports incremental training and one which does not. MALLET supports incremental training for it naïve bayes implementation and thus our incremental trainer supports naïve bayes. We provide interfaces to serialize the classifier as a RDF model and also interfaces to convert it back as MALLET instances. Though our implementation does not expose all the features that the MALLET algorithms provide, it still provides sufficient API which should suffice the requirements of most of the applications. We would suggests to use MALLETs interface if you are planning to make changes to the algorithm, out implementation aims at providing an user-friendly interfaces to the MALLET's algorithms.

Pallet exposes it features using “MalletTextDataTrainer” class. This class provides methods to create Trainer and incremental trainer, You can create Instance List for these methods either using Pallet-Data or using MALLET interfaces. We would suggest to use MALLET since our Pallet-Data does not support all the data processing pipes which forms the integral part of MALLET training algorithms.

PalletClassify project is dependent on PalletData. Further, all the files are defined under utd.pallet.classification package.

Trainer Object:

As explained earlier, MALLET Classifier extends two abstract classes ClassifierTrainer and Classifier, but they are instantiated at different points in time. Classifier is instantiated only after the trainer is trained with instances, but we assume that trainer can be serialized even before it is associated with instances. Thus, we separate these objects from the MALLET Classifier and create

"TrainerObject".

TrainerObject is defined as a nested class within MalletTextDataTrainer (Explained below) and implements Serializable interface. It has getters and setters for the ClassifierTrainer and Classifier.

Pallet Utility:

As explained through-out this document, extension we did to the MALLET was to support the Semantic Web applications. Thus, the TrainerObject needs to be serialized in a format which the semantic web applications can make use of. We used Jena API to serialize the TrainerObject as RDF model. Pallet provides MalletUtils class to convert TrainerObject to RDF model and vice-versa.

Creating Training model:

Trainer can be created using createTrainer () method of MalletTextDataTrainer class. It takes algorithm type as its input parameter and return Trainer Object. We currently support creating of naïve bayes, maximum entropy and decision tree based trainer. This method provides common interface to create all the types of trainer. Under hood, createTrainer () method uses "MalletTrainerFactory" to create these instances. MalletTrainerFactory provides factory interface for naïve bayes, max-entropy, decision trees, balanced winnowing and many more but is tested only for naïve bayes, max-entropy and decision trees.

Trainer can be created as shown:

```
InstanceList iList = //From somewhere
MalletTextDataTrainer trainer = new MalletTextDataTrainer ();
int trainingAlgorithm = MalletTextDataTrainer.NAIVE_BAYES;
TrainerObject trainerObject = trainer.trainIncremental (iList, trainingAlgorithm);
```

Incremental training model:

At the time of writing this document, MALLET supported incremental training only with the naïve bayes algorithm, Code snippet for incrementally training the model using naïve bayes:

```
InstanceList iList1 = //From somewhere
ClassifierTrainer <NaiveBayes> prevTrainer = trainerObject.getTrainer ();
MalletTextDataTrainer trainer = new MalletTextDataTrainer ();
trainerObject = trainer.trainIncremental (prevTrainer, iList1);
```

Saving the training model (as RDF data):

Code snippet below demonstrates how to convert the trainer created in the previous step as a RDF data.

```
Model model = ModelFactory.createDefaultModel();
String uri = // Some valid URI
Statement stmt =                    MalletUtils.convertTrainertoRDFStatement(model,trainerObj,uri);
model.add(stmt);
File file = new File(filename);
FileWriter writer = new FileWriter(file);
BufferedWriter bWriter = new BufferedWriter(writer);
model.write(writer, "RDF/XML");
writer.close();
```

Restoring the training model(from RDF data):

Code snippet below demonstrates how to retrieve the saved trainer:

```
FileReader reader = new FileReader (file);
BufferedReader bReader = new BufferedReader(reader);
String line = "";
String rdfData = "";
while (null != (line = bReader.readLine())) {
        rdfData += line;
}
TrainerObject trnObj =
        MalletUtils.convertRDFToTrainerObj(rdfData, "RDF/XML");
```

Classifying the test instances:

Classification of test instances can be achieved in 2 ways:

1. Using MALLET Interface:

   As explained above TrainerObject also consists of MALLET's Classifier interface within it, we can directly use this interface to classify the test instances.

   Code snippet below demonstrates how to classify the test data instances:

   ```
   ArrayList <Classification> classificationList = trnObj.getClassifier().classify(testInstanceList);
   ```

2. Using Pallet's MalletTextClassify class:

   Pallet also provides interfaces of it's own to classify the test instances.

   Code snippet:

   ```
   MalletTextClassify classifier = new MalletTextClassify();
         classificationList =
                   classifier.classify(trnObj.getClassifier(), testInstanceList);
   ```

Decision of using either of the method depends on what you want. Pallet uses the same MALLET interface and exposes simpler methods to fetch class labels and accuracy of classification. If you want more than what pallet provides, you can then switch to MALLET's interface but we would strongly suggest to use our interface before jumping into MALLET.

Getting accuracy values for the classified data:

Accuracy of classification is represented as an object of “MalletAccuracyVector” class. It provides methods to fetch the following:

1. Best class label for the test data instance.
2. Accuracy of the class labels in terms of percentage.
3. Name and source of test data.
4. Accuracy Vector.

MalletAccuracyVector class is defined in PalletData project.

Pallet Test:

Pallet Test workspace behaves both as a command-line interface to PalletData and PalletClassify project and also as a simulator demonstrating the Semantic web application development using Pallet. In this document we discuss Pallet Test with respect to command-line interface.

BlackBookSimulator class provides the command-line interface.

Command Line Interface:

Training the classifier:

If user want to train the classifier and want to store it in some output file then the command line arguments would be

TRAIN <training data set> <training algorithm> <output file name> <classification property >
for eg
TRAIN C:\Users\pralabh\workspace2\Mallet1\Montery2RDF.rdf NaiveBayes C:\Users\pralabh \workspace2\Mallet1\MonteryTrainer2 http://blackbook.com/terms#STAT_EVENT

- C:\Users\pralabh\workspace2\Mallet1\Montery2RDF.rdf : The training data
- NaiveBayes: Algorithm which is used to train the classifier.
- C:\Users\pralabh\workspace2\Mallet1\MonteryTrainer2: The location where trained classifier will be stored.
- http://blackbook.com/terms#STAT_EVENT : The classification Property on the basis of which classification will be done.

Incremental Train:

INC_TRAIN <training data set> <Filename where the classifier stored> <Output file name> <classification Property>
for eg
Inc_TRAIN C:\Users\pralabh\workspace2\Mallet1\Montery2RDF.rdf C:\User\old_classifier c:\User\new_classifier http://blackbook.com/terms#STAT_EVENT

- C:\Users\pralabh\workspace2\Mallet1\Montery2RDF.rdf : New Training Data.
- C:\User\old_classifier: The location where old classifier is stored.
- c:\User\new_classifier: The location where new classifier will be stored.
- http://blackbook.com/terms#STAT_EVENT: Classification Property.

Classification:

1. **Classification using the saved trainer.**

   CLASSIFY_SD <data which is to be classified> <trainer object file name> <validation source> <Classification Property>
   for eg
   CLASSIFY_SD C:\Users\pralabh\workspace2\Mallet\Montery2RDF.rdf
   C:\Users\pralabh\workspace2\Mallet1\MonteryTrainer2
   C:\Users\pralabh\workspace2\Mallet1\Montery2RDF.rdf

http://blackbook.com/terms#STAT_EVENT

- C:\Users\pralabh\workspace2\Mallet\Montery2RDF.rdf : Data which is to be classified
- C:\Users\pralabh\workspace2\Mallet1\MonteryTrainer2 : Place where the already trained classifier is stored.
- C:\Users\pralabh\workspace2\Mallet1\Montery2RDF.rdf : The Validation model which contains then classification of the unclassified data.(This is to compare the results of Mallet classification with actual classification)
  http://blackbook.com/terms#STAT_EVENT: The classification Property on the basis of which classification occurred.

2. **Classification by building trainer.**

   CLASSIFY <Training data Set> <test data set> <Training algorithm> <Trainer Destination File> <Validation data src> <Classification Property>

   Note: In the option ,training the classifier,storing it into the file ,classifying the test data and validating the final classified data all these things happening simultaneously.

3. **Classification using incrementally trained model.**

   CLASSIFY_INC <test data set> <trained data Set> <trainer source file> <output file> <validating data source> <classification Property>

   Note: In this option the already trained classifier is incrementally trained ,stored in a new position, classifying the test data and validating it with validating data source.

Important Points

1. If your test data set and trained data set contains multiple files or folders then you can use them by seperating them with ,. for eg
   1. If your test data set contains two files from different location then specified them as path_of_filename1,path_of_filename2(C:/user/abc.rdf,d:/user/cba.edf)
2. User can use above stated option simultaneously. For e.g. if user want to use train and classify stand alone option simultaneously then

   TRAIN C:\Users\pralabh\workspace2\Mallet1\Montery2RDF.rdf NaiveBayes
   C:\Users\pralabh\workspace2\Mallet1\MonteryTrainer2
   http://blackbook.com/terms#STAT_EVENT@CLASSIFY_SD
   C:\Users\pralabh\workspace2\Mallet\Montery2RDF.rdf
   C:\Users\pralabh\workspace2\Mallet1\MonteryTrainer2
   C:\Users\pralabh\workspace2\Mallet1\Montery2RDF.rdf
   http://blackbook.com/terms#STAT_EVENT

Note : The two options are seperated by @ symbol.

Entity Extraction:

MALLET supports sequence tagging module which under cc.mallet.fst package. It provides HMM, MEMM and Linear chain CRF algorithms. We again intend to develop a wrapper with which we can extend the MALLET to support semantic web applications. The plan is to make it an independent project which supports named-entity extraction of free text data and persists it as an RDF/XML data which can be used as data feed for the semantic web application. At the time of writing this document, this module is still in design phase.

Whats left to do:

1. test with other data sets.
2. k-fold cross validation.
3. use algorithm implementations other than Naive Bayes.
4. go beyond classification, attempting entity extraction, etc.
5. extend Pallet Data module to work against all RDF/XML data.
6. test against other machine learning toolkits.

Summary:

This document intends to provide an overview of MALLET's classification and sequence tagging modules and further explain our work of professionalizing the MALLET package. As a part of professionalization we have documented various interfaces that MALLET provides and further we provide easier interfaces which suffices the requirements of application development. This framework which we created for the pre-processing and classification supports interfaces to build both stand-alone and Semantic Web applications.

References:

1. http://mallet.cs.umass.edu/
2. http://en.wikipedia.org/wiki/Naive_Bayes_classifier
3. http://en.wikipedia.org/wiki/Decision_tree_learning
4. http://en.wikipedia.org/wiki/Maximum_entropy_classifier
5. http://www.authorstream.com/Presentation/aSGuest4218-114230-nips-ie-tutorial-ppt-education-powerpoint/
6. http://jena.sourceforge.net/
7. http://www.w3.org/TR/REC-rdf-syntax/

Appendix:

INDEX

Trainer Object -15
Trainer Model - 16

J	U
K	V