FREE eBook

LEARNING weka

Free unaffiliated eBook created from
Stack Overflow contributors.

#weka

Table of Contents

About 1

Chapter 1: Getting started with weka 2

Remarks 2

Examples 2

Installation or Setup 2

Downloading and installing Weka 2

Chapter 2: Getting Started With Jython in Weka 4

Introduction 4

Remarks 4

How to setup Jython in weka 4

Examples 4

Load and Filter Data 4

Build a classifier 5

Cross-validate Classifier 5

Make A Prediction 5

Cross-validate Classifier Error Bubble 6

Display Graph 7

Chapter 3: How to use CPython Scripting in Weka? 9

Remarks 9

How to install CPython in Weka? 9

Examples 9

Hello World Example for CPython of Weka 9

Chapter 4: How to use R in Weka 10

Remarks 10

Why use R in Weka? 10

How to setup R in Weka 10

How to receive data from Weka? 10

Playing R Codes 11

Examples 11

Plotting inside R Console ... 11
Chapter 5: Loading Instances ... 13
Examples ... 13
ARFF Files ... 13
Loading ARFF Files ... 14
Weka <3.5.5 ... 14
Weka >=3.5.5 ... 14
Loading from Database ... 14
Chapter 6: Mistakes easily made when using KnowledgeFlow ... 16
Introduction ... 16
Remarks ... 16
TrainingSetMaker and TestSetMaker ... 16
ArffSaver ... 16
How to use TimeSeriesForecasting in KnowledgeFlow? ... 16
Examples ... 16
How to open KnowledgeFlow file directly from terminal ... 16
Chapter 7: Simple Comparison of Weka Interfaces ... 18
Introduction ... 18
Remarks ... 18
Examples ... 19
simpleCLI and Jython examples ... 19
Chapter 8: Text Classification ... 20
Examples ... 20
Text classification with LibLinear ... 20
Credits ... 23

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: weka

It is an unofficial and free weka ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official weka.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with weka

Remarks

This section provides an overview of what weka is, and why a developer might want to use it.

It should also mention any large subjects within weka, and link out to the related topics. Since the Documentation for weka is new, you may need to create initial versions of those related topics.

Examples

Installation or Setup

Weka is a collection of machine learning algorithms for data mining tasks. The algorithms can either be applied directly to a dataset or called from your own Java code. Weka contains tools for data pre-processing, classification, regression, clustering, association rules, and visualization. It is also well-suited for developing new machine learning schemes.

Downloading and installing Weka

There are two versions of Weka: Weka 3.8 is the latest stable version, and Weka 3.9 is the development version. For the bleeding edge, it is also possible to download nightly snapshots.

Stable versions receive only bug fixes, while the development version receives new features. Weka 3.8 and 3.9 feature a package management system that makes it easy for the Weka community to add new functionality to Weka. The package management system requires an internet connection in order to download and install packages.

You can download the application for Windows/MacOS/Linux here.

Integrate WEKA library in your code:

pox.xml:

```
<dependency>
    <groupId>nz.ac.waikato.cms.weka</groupId>
    <artifactId>weka-dev</artifactId>
    <version>3.9.1</version>
</dependency>
```

gradle:

```
compile group: 'nz.ac.waikato.cms.weka', name: 'weka-dev', version: '3.9.1'
```

Read Getting started with weka online: https://riptutorial.com/weka/topic/3699/getting-started-with-

weka

Chapter 2: Getting Started With Jython in Weka

Introduction

Why would we use Jython inside Weka? 1. If you are unsatisfied with what Explorer, Experimenter, KnowledgeFlow, simpleCLI allow you to do, and looking for something to unleash the greater power of weka; 2. With Jython, we can access all functionalities provided by Weka API, right inside Weka; 3. Its syntax is Python-like, which is considered to be a beginner-friendly scripting language;

Remarks

How to setup Jython in weka

1. install `Jython` and `JFreeChart` library from Weka Package manager;
2. go to home directory's terminal, enter `nano .bash_profile`
3. inside `.bash_profile`, add a line of code as below

   ```
   export Weka_Data=User/Documents/Directory/Of/Your/Data
   ```

4. save and exit
5. inside terminal run `source .bash_profile`

Then, restart Weka, go to `tools` and click `Jython console`, and you can try those examples above

Examples

Load and Filter Data

```
# imports
import weka.core.converters.ConverterUtils.DataSource as DS
import weka.filters.Filter as Filter
import weka.filters.unsupervised.attribute.Remove as Remove
import os

# load data
data = DS.read(os.environ.get("MOOC_DATA") + os.sep + "iris.arff")

# remove class attribute
rem = Remove()
rem.setOptions(["-R", "last"])
rem.setInputFormat(data)
```

```
dataNew = Filter.useFilter(data, rem)

# output filtered dataset
print(dataNew)
```

Build a classifier

```
# imports
import weka.core.converters.ConverterUtils.DataSource as DS
import weka.classifiers.trees.J48 as J48
import os

# load data
data = DS.read(os.environ.get("MOOC_DATA") + os.sep + "anneal.arff")
data.setClassIndex(data.numAttributes() - 1)

# configure classifier
cls = J48()
cls.setOptions(["-C", "0.3"])

# build classifier
cls.buildClassifier(data)

# output model
print(cls)
```

Cross-validate Classifier

```
# imports
import weka.core.converters.ConverterUtils.DataSource as DS
import weka.classifiers.Evaluation as Evaluation
import weka.classifiers.trees.J48 as J48
import java.util.Random as Random
import os

# load data
data = DS.read(os.environ.get("MOOC_DATA") + os.sep + "anneal.arff")
data.setClassIndex(data.numAttributes() - 1)

# configure classifier
cls = J48()
cls.setOptions(["-C", "0.3"])

# cross-validate classifier
evl = Evaluation(data)
evl.crossValidateModel(cls, data, 10, Random(1))

# print statistics
print(evl.toSummaryString("=== J48 on anneal (stats) ===", False))
print(evl.toMatrixString("=== J48 on anneal (confusion matrix) ==="))
```

Make A Prediction

```
# imports
import weka.classifiers.trees.J48 as J48
import weka.core.converters.ConverterUtils.DataSource as DS
```

```
import os

# load training data
data = DS.read(os.environ.get("MOOC_DATA") + os.sep + "anneal_train.arff")
data.setClassIndex(data.numAttributes() - 1)

# configure classifier
cls = J48()
cls.setOptions(["-C", "0.3"])

# build classifier on training data
cls.buildClassifier(data)

# load unlabeled data
dataUnl = DS.read(os.environ.get("MOOC_DATA") + os.sep + "anneal_unlbl.arff")
dataUnl.setClassIndex(dataUnl.numAttributes() - 1)

# test compatibility of train/unlabeled datasets
msg = dataUnl.equalHeadersMsg(data)
if msg is not None:
    print("train and prediction data are not compatible:\n" + msg)

# make predictions
for inst in dataUnl:
    dist = cls.distributionForInstance(inst)
    labelIndex = cls.classifyInstance(inst)
    label = dataUnl.classAttribute().value(int(labelIndex))
    print(str(dist) + " - " + str(labelIndex) + " - " + label)
```

Cross-validate Classifier Error Bubble

```
# Note: install jfreechartOffscreenRenderer package as well for JFreeChart library

# imports
import weka.classifiers.Evaluation as Evaluation
import weka.classifiers.functions.LinearRegression as LinearRegression
import weka.core.converters.ConverterUtils.DataSource as DS
import java.util.Random as Random
import org.jfree.data.xy.DefaultXYZDataset as DefaultXYZDataset
import org.jfree.chart.ChartFactory as ChartFactory
import org.jfree.chart.plot.PlotOrientation as PlotOrientation
import org.jfree.chart.ChartPanel as ChartPanel
import org.jfree.chart.renderer.xy.XYBubbleRenderer as XYBubbleRenderer
import org.jfree.chart.ChartUtilities as ChartUtilities
import javax.swing.JFrame as JFrame
import java.io.File as File
import os

# load data
data = DS.read(os.environ.get("MOOC_DATA") + os.sep + "bodyfat.arff")
data.setClassIndex(data.numAttributes() - 1)

# configure classifier
cls = LinearRegression()
cls.setOptions(["-C", "-S", "1"])

# cross-validate classifier
evl = Evaluation(data)
evl.crossValidateModel(cls, data, 10, Random(1))
```

```
# collect predictions
act = []
prd = []
err = []
for i in range(evl.predictions().size()):
    prediction = evl.predictions().get(i)
    act.append(prediction.actual())
    prd.append(prediction.predicted())
    err.append(abs(prediction.actual() - prediction.predicted()))

# create plot
plotdata = DefaultXYZDataset()
plotdata.addSeries("LR on " + data.relationName(), [act, prd, err])
plot = ChartFactory.createScatterPlot(\
    "Classifier errors", "Actual", "Predicted", \
    plotdata, PlotOrientation.VERTICAL, True, True, True)
plot.getPlot().setRenderer(XYBubbleRenderer())

# display plot
frame = JFrame()
frame.setTitle("Weka")
frame.setSize(800, 800)
frame.setLocationRelativeTo(None)
frame.getContentPane().add(ChartPanel(plot))
frame.setVisible(True)
```

Display Graph

```
# imports
import weka.classifiers.bayes.BayesNet as BayesNet
import weka.core.converters.ConverterUtils.DataSource as DS
import weka.gui.graphvisualizer.GraphVisualizer as GraphVisualizer
import javax.swing.JFrame as JFrame
import os

# load data
data = DS.read(os.environ.get("MOOC_DATA") + os.sep + "iris.arff")
data.setClassIndex(data.numAttributes() - 1)

# configure classifier
cls = BayesNet()
cls.setOptions(["-Q", "weka.classifiers.bayes.net.search.local.K2", "--", "-P", "2"])

# build classifier
cls.buildClassifier(data)

# display tree
gv = GraphVisualizer()
gv.readBIF(cls.graph())
frame = JFrame("BayesNet - " + data.relationName())
frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE)
frame.setSize(800, 600)
frame.getContentPane().add(gv)
frame.setVisible(True)

# adjust tree layout
gv.layoutGraph()
```

Read Getting Started With Jython in Weka online: https://riptutorial.com/weka/topic/8046/getting-started-with-jython-in-weka

Chapter 3: How to use CPython Scripting in Weka?

Remarks

How to install CPython in Weka?

Install wekaPython

1. go to `tools`, open `package manager`
2. search `wekaPython`, select and click to install

Install Python libraries

1. install anaconda or conda
2. install four packages: numpy, pandas, matplotlib, scikit-learn
3. for full installation doc see conda

Examples

Hello World Example for CPython of Weka

Go to `Explorer`, Open `iris.arff` data, then go to `CPython Scripting`, Copy and Paste the following lines of codes into `Python Scripts`:

```
hi = "Hello, CPython of Weka!"
hello = hi.upper()
iris = py_data
info = iris.describe()
```

To see output, go to `Python Variables`, select `hi`, for example, and click `Get text`

Read How to use CPython Scripting in Weka? online: https://riptutorial.com/weka/topic/7921/how-to-use-cpython-scripting-in-weka-

Chapter 4: How to use R in Weka

Remarks

Why use R in Weka?

1. R is a powerful tool for preprocessing data
2. R has a huge number of libraries and keeps growing
3. R in Weka, can easily get data from, process it, and pass to Weka seamlessly

---

How to setup R in Weka

For Mac User

1. replace the old info.Plist with the new one given by Mark Hall
2. download and install R
3. install `rJava` inside R with

   install.packages('rJava')
4. install `Rplugin` with `Weka Package Manager`
5. go to `weka 3-8-0` folder (if it is the version you are using), and open its terminal, and
6. run the following 2 lines of codes (thanks to Michael Hall)

   export R_HOME=/Library/Frameworks/R.framework/Resources
   java -Xss10M -Xmx4096M -cp .:weka.jar weka.gui.GUIChooser
7. to make life easier, inside a directory where you want to work with weka, save the code above into a file named as `weka_r.sh`
8. make it executable, inside this directory's terminal, run the code below:

   chmod a+x weka_r.sh
9. paste `weka.jar` from weka 3-8-0 into the directory and run the code below:

   ./weka_r.sh

Now, you are ready to go. Next time, you just need to go to the directory's terminal and run `./weka_r.sh` to start R with Weka.

How to receive data from Weka?

open Weka from terminal:
go to directory of `Weka 3-8-0`, open its terminal, run the following code:

```
java -jar weka.jar
```

data through Weka Explorer:

1. `preprocess` panel, click `open file`, choose a data file from `weka data folder`;
2. go to `R console` panel, type R scripts inside `R console box`.

data through Weka KnowledgeFlow:

1. `Data mining processes` panel, click `DataSources` to choose `ArffLoader` for example, click it onto canvas;
2. double-click `ArffLoader` to load a data file
3. `Scripting` panel, click `RscriptExecutor` onto canvas
4. `option` + click `ArffLoader`, select `dataset`, then click `RScript Executor` to link them
5. double click `RScript Executor` to type R script, or
6. click `Settings` and select `R Scripting` to use R console with weka's data

---

Playing R Codes

1. load `iris.arff` with either Explorer or KnowledgeFlow;
2. try `Plotting inside R Console` example above

Examples

Plotting inside R Console

> The following Codes can be found from Weka course

Given `iris.arff` is loaded in weka, inside Weka Explorer's `R console` or Weka KnowledgeFlow's `R Scripting`, you can play with the following codes to make beautiful plots:

```
library(ggplot2)

ggplot(rdata, aes(x = petallength)) + geom_density()

ggplot(rdata, aes(x = petallength)) + geom_density() + xlim(0,8)

ggplot(rdata, aes(x = petallength)) + geom_density(adjust = 0.5) + xlim(0,8)

ggplot(rdata, aes(x = petallength, color = class)) + geom_density(adjust = 0.5) + xlim(0,8)
```

```
ggplot(rdata, aes(x = petallength, color = class, fill = class)) + geom_density(adjust = 0.5)
+ xlim(0,8)

ggplot(rdata, aes(x = petallength, color = class, fill = class)) + geom_density(adjust = 0.5,
alpha = 0.5) + xlim(0,8)




library(reshape2)
ndata = melt(rdata)
ndata

ggplot(ndata, aes(x = value, color = class, fill = class)) + geom_density(adjust = 0.5, alpha
= 0.5) + xlim(0,8) + facet_grid(variable ~ .)

ggplot(ndata, aes(x = value, color = class, fill = class)) + geom_density(adjust = 0.5, alpha
= 0.5) + xlim(0,8) + facet_grid(. ~ variable)

ggplot(ndata, aes(y = value, x = class, colour = class)) + geom_boxplot() + facet_grid(. ~
variable)
```

Read How to use R in Weka online: https://riptutorial.com/weka/topic/7916/how-to-use-r-in-weka

Chapter 5: Loading Instances

Examples

ARFF Files

ARFF files (Attribute-Relation File Format) are the most common format for data used in Weka. Each ARFF file must have a header describing what each data instance should be like. The attributes that can be used are as follows:

- Numeric

Real or integer numbers.

- Nominal

Nominal attributes must provide a set of possible values. For example:

```
@ATTRIBUTE class        {Iris-setosa,Iris-versicolor,Iris-virginica}
```

- String

Allows for arbitrary string values. Usually processed later using the `StringToWordVector` filter.

- Date

Allows for dates to be specified. As with Java's `SimpleDateFormat`, this date can also be formatted; it will default to ISO-8601 format.

An example header can be seen as follows:

```
@RELATION iris

@ATTRIBUTE sepallength  NUMERIC
@ATTRIBUTE sepalwidth   NUMERIC
@ATTRIBUTE petallength  NUMERIC
@ATTRIBUTE petalwidth   NUMERIC
@ATTRIBUTE class        {Iris-setosa,Iris-versicolor,Iris-virginica}
```

Following the header each instance must be listed with the correct number of instances; if an attributes value for an instance is not known a `?` can be used instead. The following shows an example of the set of instances in an ARFF file:

```
@DATA
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
4.6,3.1,1.5,0.2,Iris-setosa
5.0,3.6,1.4,0.2,Iris-setosa
```

Loading ARFF Files

Depending on the version of Weka being used different methods for loading ARFF files should be utilised.

Weka <3.5.5

The following sample code shows how to load an ARFF file:

```
import weka.core.Instances;
import java.io.BufferedReader;
import java.io.FileReader;
...
BufferedReader reader = new BufferedReader(new FileReader("data.arff"));
Instances data = new Instances(reader);
reader.close();
data.setClassIndex(data.numAttributes() - 1);
```

The class index shows what attribute should be used for classification. In most ARFF files this is the last attribute which is why it is set to `data.numAttributes() - 1`. If you are using a Weka function, such as `buildClassifier`, you must set the class index.

Weka >=3.5.5

In the latest version of Weka it is very easy to load an ARFF file. This method can also load CSV files and any other files Weka can understand.

```
import weka.core.converters.ConverterUtils.DataSource;
...
DataSource source = new DataSource("data.arff");
Instances data = source.getDataSet();
if (data.classIndex() == -1) {
  data.setClassIndex(data.numAttributes() - 1);
}
```

Loading from Database

Many databases can be used in Weka. Firstly, the DatabaseUtils.props file must be edited to match your database; specifically you must provide your database's name, location, port and correct driver.

```
jdbcDriver=org.gjt.mm.mysql.Driver
jdbcURL=jdbc:mysql://localhost:3306/my_database
```

Then the database can be loaded by using some simple code.

```
import weka.core.Instances;
```

```
import weka.experiment.InstanceQuery;
...
InstanceQuery query = new InstanceQuery();
query.setUsername("user");
query.setPassword("pass");
query.setQuery("select * from mytable");
Instances data = query.retrieveInstances();
```

Some notes about loading from a database:

- Make sure the correct JDBC driver is in your classpath.
- If you are using Microsoft Access then the JDBC-ODBC-driver which comes with the JDK can be used.
- The `InstanceQuery` method converts VARCHAR to nominal attributes and TEXT to string attributes. A filter, such as `NominalToString` or `StringToNormal`, can convert the attributes back to their correct type.

Read Loading Instances online: https://riptutorial.com/weka/topic/5928/loading-instances

Chapter 6: Mistakes easily made when using KnowledgeFlow

Introduction

Weka KnowledgeFlow(KF) is a great interface to use. However, Weka manual does not cover every little details of using KF. Here would be a place for collecting those little tricks or details I learnt from those errors I did or will make as time goes. Many thanks to people at Wekalist (especially Mark Hall, Eibe Frank) for building a wonderful learning environment for Weka!

Remarks

TrainingSetMaker and TestSetMaker

1. a `ClassAssigner` must be linked between `ArffLoader` and `TrainingSetMaker` or `TestSetMaker`.

ArffSaver

2. In order to save dataset into arff file successfully, it is safer to set `relationNameForFilename` to False inside configuration of `ArffSaver`.

How to use TimeSeriesForecasting in KnowledgeFlow?

1. Open knowledgeFlow, load dataset with ArffLoader
2. go to setting, check time series forecasting perspective, right-click ArffLoader to send to all perspective
3. go to time series forecasting perspective to set up a model
4. run the model and copy the model to clipboard
5. ctrl + v, and click to paste model to Data mining process canvas
6. save prediction along with original data with ArffSaver

Examples

How to open KnowledgeFlow file directly from terminal

1. add the following function into `.bash_profile`, save and exit

```
function wekaflstart() {
export R_HOME=/Library/Frameworks/R.framework/Resources
```

```
java -Xss10M -Xmx4096M -cp :weka.jar weka.gui.knowledgeflow.KnowledgeFlow "$1"
}
```

2. inside a directory with a `weka.jar` file, open its terminal, run `wekastart "path to a knowledgeflow file"`

Read Mistakes easily made when using KnowledgeFlow online:
https://riptutorial.com/weka/topic/8053/mistakes-easily-made-when-using-knowledgeflow

Chapter 7: Simple Comparison of Weka Interfaces

Introduction

Weka has many interfaces, Explorer, KnowledgeFlow, Experimenter, SimpleCLI, Workbench. All of them share mostly can do the same tasks, with different focus and flexibility. Here, we are going to explore their different focuses and flexibilities.

Remarks

Explorer

pro:

1. do all things quickly
2. give a quick and comprehensive view of data structure

cos: can't save the process;

Experimenter

pro:

1. compare several models at once, e.g., run 3 different classifiers against 5 datasets all together, and see the compared result at one place;
2. experiment can be saved

KnowledgeFlow

pro:

1. do almost all things that Explorer can do
2. can save the process

cos:

1. KF can't do Experimenter's job, as it doesn't support loops, but ADAMS can help;
2. KF can't access low-level functionalities inside Weka API;

simpleCLI

pro: run similar tasks of what Explorer does using command line

cos: it can't access all functionalities of Weka API, Jython or Groovy scripting is recommended for this task.

Workbench

pro: it gathers all other interfaces together into one place

Examples

simpleCLI and Jython examples

simpleCLI

go to simpleCLI, enter the following code

```
java weka.classifiers.rules.ZeroR -t path/to/a-file-of-dataset
```

Jython Example

codes from Advanced Weka MOOC course lesson 5.1

```
# imports
import weka.core.converters.ConverterUtils.DataSource as DS
import weka.filters.Filter as Filter
import weka.filters.unsupervised.attribute.Remove as Remove
import os

# load data
data = DS.read(os.environ.get("MOOC_DATA") + os.sep + "iris.arff")

# remove class attribute
rem = Remove()
rem.setOptions(["-R", "last"])
rem.setInputFormat(data)
dataNew = Filter.useFilter(data, rem)

# output filtered dataset
print(dataNew)
```

Read Simple Comparison of Weka Interfaces online:
https://riptutorial.com/weka/topic/8042/simple-comparison-of-weka-interfaces

Chapter 8: Text Classification

Examples

Text classification with LibLinear

- Create training instances from .arff file

```
private static Instances getDataFromFile(String path) throws Exception{

    DataSource source = new DataSource(path);
    Instances data = source.getDataSet();

    if (data.classIndex() == -1){
        data.setClassIndex(data.numAttributes()-1);
        //last attribute as class index
    }

    return data;
}
```

```
Instances trainingData = getDataFromFile(pathToArffFile);
```

- Use **StringToWordVector** to transform your string attributes to number representation:

  *Important features of this filter:

  1. tf-idf representation
  2. stemming
  3. lowercase wrods
  4. stopwords
  5. n-gram representation*

```
    StringToWordVector() filter = new StringToWordVector();
    filter.setWordsToKeep(1000000);
    if(useIdf){
        filter.setIDFTransform(true);
    }
    filter.setTFTransform(true);
    filter.setLowerCaseTokens(true);
    filter.setOutputWordCounts(true);
    filter.setMinTermFreq(minTermFreq);
    filter.setNormalizeDocLength(new
SelectedTag(StringToWordVector.FILTER_NORMALIZE_ALL,StringToWordVector.TAGS_FILTER));
    NGramTokenizer t = new NGramTokenizer();
    t.setNGramMaxSize(maxGrams);
    t.setNGramMinSize(minGrams);
    filter.setTokenizer(t);
    WordsFromFile stopwords = new WordsFromFile();
    stopwords.setStopwords(new File("data/stopwords/stopwords.txt"));
    filter.setStopwordsHandler(stopwords);
```

```
    if (useStemmer){
        Stemmer s = new /*Iterated*/LovinsStemmer();
        filter.setStemmer(s);
    }
    filter.setInputFormat(trainingData);
```

- Apply the filter to trainingData: `trainingData = Filter.useFilter(trainingData, filter);`
- Create the LibLinear Classifier

    1. SVMType 0 below corresponds to the L2-regularized logistic regression
    2. Set `setProbabilityEstimates(true)` to print the output probalities

```
        Classifier cls = null;
        LibLINEAR liblinear = new LibLINEAR();
        liblinear.setSVMType(new SelectedTag(0, LibLINEAR.TAGS_SVMTYPE));
        liblinear.setProbabilityEstimates(true);
        // liblinear.setBias(1); // default value
        cls = liblinear;
        cls.buildClassifier(trainingData);
```

- Save model

```
    System.out.println("Saving the model...");
    ObjectOutputStream oos;
    oos = new ObjectOutputStream(new FileOutputStream(path+"mymodel.model"));
    oos.writeObject(cls);
    oos.flush();
    oos.close();
```

- Create testing instances from `.arff` file

```
 Instances trainingData = getDataFromFile(pathToArffFile);
```

- Load classifier

```
Classifier myCls = (Classifier) weka.core.SerializationHelper.read(path+"mymodel.model");
```

- **Use the same StringToWordVector filter as above or create a new one for testingData, but remember to use the trainingData for this command:**
  `filter.setInputFormat(trainingData);` *This will make training and testing instances compatible. Alternatively you could use* *`InputMappedClassifier`*
- Apply the filter to testingData: `testingData = Filter.useFilter(testingData, filter);`
- Classify!

    1.Get the class value for every instance in the testing set

```
 for (int j = 0; j < testingData.numInstances(); j++) {
    double res = myCls.classifyInstance(testingData.get(j));
 }
```

`res` is a double value that corresponds to the nominal class that is defined in `.arff` file. To get the nominal class use : `testintData.classAttribute().value((int)res)`

2.Get the probability distribution for every instance

```
for (int j = 0; j < testingData.numInstances(); j++) {
   double[] dist = first.distributionForInstance(testInstances.get(j));
}
```

`dist` is a double array that contains the probabilities for every class defined in `.arff` file

Note. Classifier should support probability distributions and enable them with:
`myClassifier.setProbabilityEstimates(true);`

Read Text Classification online: https://riptutorial.com/weka/topic/7753/text-classification

Credits

S. No	Chapters	Contributors
1	Getting started with weka	Community, Gal Dreiman
2	Getting Started With Jython in Weka	Daniel
3	How to use CPython Scripting in Weka?	Daniel
4	How to use R in Weka	Daniel
5	Loading Instances	SJB
6	Mistakes easily made when using KnowledgeFlow	Daniel
7	Simple Comparison of Weka Interfaces	Daniel
8	Text Classification	xro7