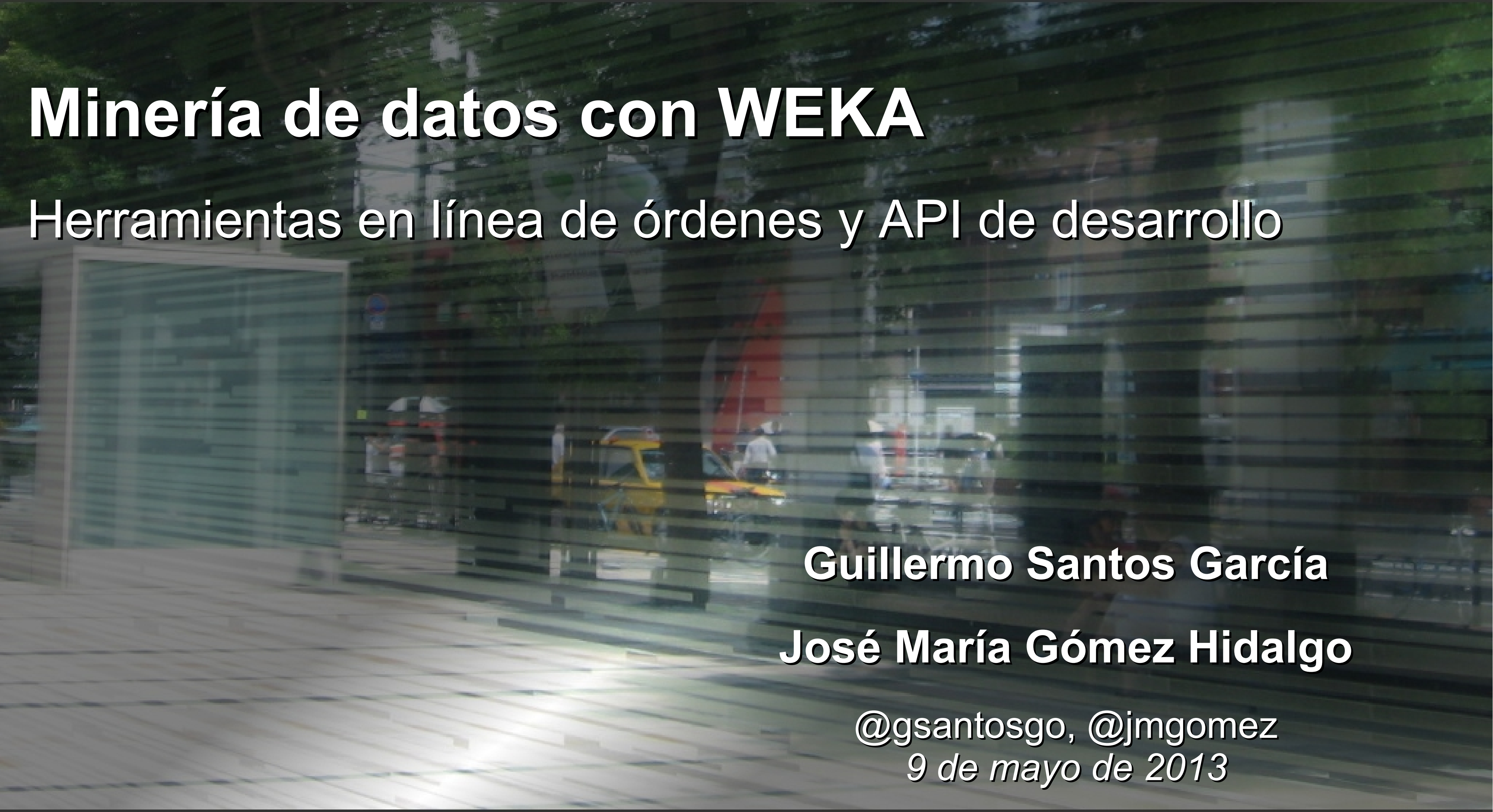


# Minería de datos con WEKA

## Herramientas en línea de órdenes y API de desarrollo

**Guillermo Santos García**

**José María Gómez Hidalgo**

@gsantosgo, @jmgomez

*9 de mayo de 2013*

## WEKA en la línea de órdenes

## WEKA en la línea de órdenes

- WEKA proporciona clases para los principales procesos de minería de datos
  - Gestión y selección de atributos
  - Entrenamiento y evaluación de clasificadores

## WEKA en la línea de órdenes

- Las clases admiten llamadas desde la línea de órdenes
  - Múltiples opciones heredadas de las superclases abstractas
  - Y opciones propias de las clases específicas

## WEKA en la línea de órdenes

- Opciones de un clasificador
  - Evaluación
    - Con colección de prueba
    - Por validación cruzada
    - Por partición
  - Entrenamiento y almacenamiento del modelo
  - Predicción sobre ejemplares de prueba

# WEKA en la línea de órdenes

- Clasificador – evaluación con archivo de test

```
$> java weka.classifiers.bayes.NaiveBayes
-t <name of training file>
        Sets training file.
-T <name of test file>
        Sets test file. If missing, a cross-validation will be
        performed on the training data.
-c <class index>
        Sets index of class attribute (default: last).
-v
        Outputs no statistics for training data.
-o
        Outputs statistics only, not the classifier.
-i
        Outputs detailed information-retrieval
        statistics for each class.
-k
        Outputs information-theoretic statistics.
```

- Clasificador – evaluación con archivo de test

```
$> java weka.classifiers.bayes.NaiveBayes -t zip.train.arff -T zip.test.arff
   -c first -v -o -i -k

=== Error on test data ===
Correctly Classified Instances          1499               74.6886 %
Incorrectly Classified Instances         508               25.3114 %
Kappa statistic                            0.7166
K&B Relative Info Score               145249.1244 %
K&B Information Score                   4756.1354 bits      2.3698 bits/instance
Class complexity | order 0              6559.7021 bits      3.2684 bits/instance
Class complexity | scheme             137029.0351 bits     68.2756 bits/instance
Complexity improvement     (Sf)      -130469.333  bits    -65.0071 bits/instance
Mean absolute error                        0.0507
Root mean squared error                    0.224
Relative absolute error                   28.4178 %
Root relative squared error               75.0231 %
Coverage of cases (0.95 level)            75.1868 %
Mean rel. region size (0.95 level)        10.1445 %
Total Number of Instances               2007
../..
```

## WEKA en la línea de órdenes

- Clasificador – evaluación por validación cruzada o partición

```
$> java weka.classifiers.bayes.NaiveBayes
-t <name of training file>
        Sets training file.
-x <number of folds>
        Sets number of folds for cross-validation (default: 10).
-no-cv
        Do not perform any cross validation.
-split-percentage <percentage>
        Sets the percentage for the train/test set split, e.g.,
        66.
-preserve-order
        Preserves the order in the percentage split.
-s <random number seed>
        Sets random number seed for cross-validation or percentage
        Split (default: 1).
```

## WEKA en la línea de órdenes

- Clasificador – evaluación por validación cruzada

```
$> java weka.classifiers.bayes.NaiveBayes -t spambase.arff -x 10 -v -o

=== Stratified cross-validation ===
Correctly Classified Instances        3648               79.2871 %
Incorrectly Classified Instances       953               20.7129 %
Kappa statistic                          0.5965
Mean absolute error                      0.2066
Root mean squared error                  0.4527
Relative absolute error                 43.2668 %
Root relative squared error             92.6423 %
Coverage of cases (0.95 level)          79.787  %
Mean rel. region size (0.95 level)      50.4347 %
Total Number of Instances             4601

=== Confusion Matrix ===
    a    b   <-- classified as
 1923  865 |    a = email
   88 1725 |    b = spam
```

## WEKA en la línea de órdenes

- Clasificador – evaluación por partición

```
$> java weka.classifiers.bayes.NaiveBayes -t spambase.arff
   -split-percentage 80 -v -o

=== Error on test split ===
Correctly Classified Instances         723               78.587  %
Incorrectly Classified Instances       197               21.413  %
Kappa statistic                          0.5844
Mean absolute error                      0.2142
Root mean squared error                  0.4615
Relative absolute error                 44.6542 %
Root relative squared error             93.9037 %
Coverage of cases (0.95 level)          78.913  %
Mean rel. region size (0.95 level)      50.3261 %
Total Number of Instances              920

=== Confusion Matrix ===
   a   b   <-- classified as
 367 179 |   a = email
  18 356 |   b = spam
```

## WEKA en la línea de órdenes

- Clasificador – almacenamiento del modelo

```
$> java weka.classifiers.bayes.NaiveBayes
-t <name of training file>
        Sets training file.
-l <name of input file>
        Sets model input file. In case the filename ends with
        '.xml', a PMML file is loaded or, if that fails, options
        are loaded from the XML file.
-d <name of output file>
        Sets model output file. In case the filename ends with
        '.xml', only the options are saved to the XML file,
        not the model.
```

## WEKA en la línea de órdenes

- Clasificador – almacenamiento del modelo

```
$> java weka.classifiers.bayes.NaiveBayes -t spambase.arff
   -d spambase.NB.data -no-cv -o
Time taken to build model: 0.11 seconds
Time taken to test model on training data: 0.31 seconds
=== Error on training data ===
Correctly Classified Instances        3659               79.5262 %
Incorrectly Classified Instances       942               20.4738 %
Kappa statistic                          0.6014
Mean absolute error                      0.2041
Root mean squared error                  0.45
Relative absolute error                 42.7361 %
Root relative squared error             92.0846 %
Coverage of cases (0.95 level)          80.0696 %
Mean rel. region size (0.95 level)      50.4456 %
Total Number of Instances             4601
$> more spambase.NB.data
¼Ý
♣sr
!weka.classifiers.bayes.NaiveBayesS3W♥ÉãUw☻
```

- Clasificador – predicción

```
$> java weka.classifiers.bayes.NaiveBayes
-classifications "weka.classifiers.evaluation.output.prediction.AbstractOutput
        + options"
        Uses the specified class for generating the classification output.
        E.g.: weka.classifiers.evaluation.output.prediction.PlainText
-p range
        Outputs predictions for test instances (or the train instances if
        no test instances provided and -no-cv is used), along with the
        attributes in the specified range (and nothing else).
        Use '-p 0' if no attributes are desired.
        Deprecated: use "-classifications ..." instead.
-distribution
        Outputs the distribution instead of only the prediction
        in conjunction with the '-p' option (only nominal classes).
        Deprecated: use "-classifications ..." instead.
```

## WEKA en la línea de órdenes

- Clasificador – predicción

```
$> java weka.classifiers.bayes.NaiveBayes -l spambase.NB.data -T spambase.arff
   -p 0
=== Predictions on test data ===
 inst#     actual  predicted error prediction
     1     2:spam     2:spam       1
     2     2:spam     2:spam       1
     3     2:spam     2:spam       1
../..
  4599    1:email    1:email       0.996
  4600    1:email     2:spam   +   1
  4601    1:email     2:spam   +   0.987

$> java weka.classifiers.bayes.NaiveBayes -l spambase.NB.data -T spambase.arff
 -classifications weka.classifiers.evaluation.output.prediction.PlainText
=== Predictions on test data ===
 inst#     actual  predicted error prediction
     1     2:spam     2:spam       1
../..
```

## WEKA en la línea de órdenes

- Selección de atributos
  - Atributos seleccionados respecto a métrica de calidad
    - Opciones de búsqueda
    - Funciones de medición de calidad

- Selección de atributos

```
$> java weka.filters.supervised.attribute.AttributeSelection -h
Help requested.
Filter options:
-S <"Name of search class [search options]">
        Sets search method for subset evaluators.
        eg. -S "weka.attributeSelection.BestFirst -S 8"
-E <"Name of attribute/subset evaluation class [evaluator options]">
        Sets attribute/subset evaluator.
        eg. -E "weka.attributeSelection.CfsSubsetEval -L"
General options:
-i <file>
        The name of the file containing input instances.
        If not supplied then instances will be read from stdin.
-o <file>
        The name of the file output instances will be written to.
        If not supplied then instances will be written to stdout.
```

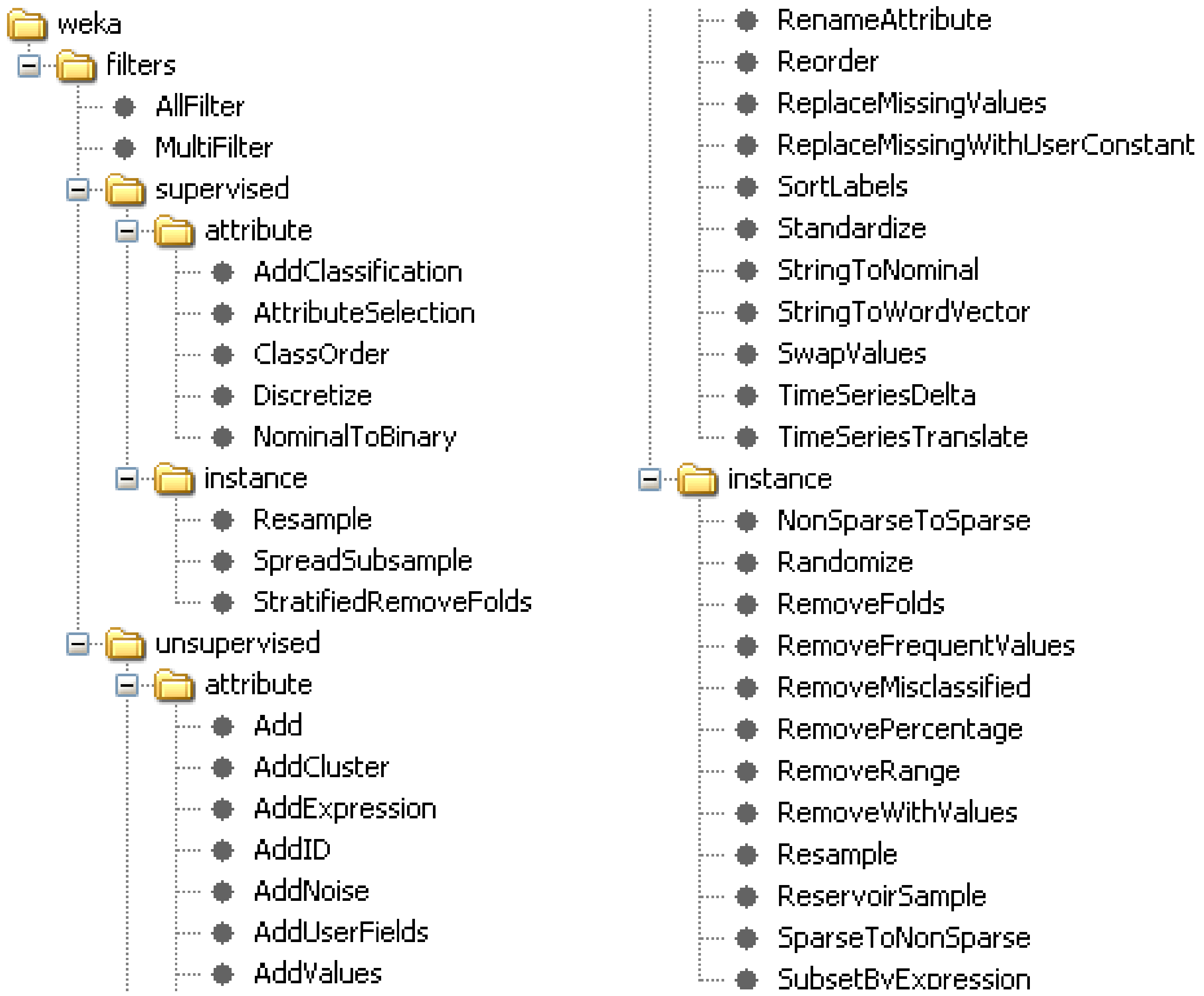
## WEKA en la línea de órdenes

- Selección de atributos

```
$> java weka.filters.supervised.attribute.AttributeSelection
   -i spambase.arff -o spambase.IG0.arff
   -E weka.attributeSelection.InfoGainAttributeEval
   -S "weka.attributeSelection.Ranker -T 0.2"

$> more spambase.IG0.arff
@relation '...'
@attribute char_freq_ch! numeric
@attribute char_freq_ch$ numeric
@attribute capital_run_length_longest numeric
@attribute word_freq_remove numeric
@attribute word_freq_your numeric
@attribute capital_run_length_average numeric
@attribute spam {email,spam}
@data
0.778,0,61,0,0.96,3.756,spam
0.372,0.18,101,0.21,1.59,5.114,spam
0.276,0.184,485,0.19,0.51,9.821,spam
```

## WEKA en la línea de órdenes

- Otros filtros



```
weka
  filters
    AllFilter
    MultiFilter
    supervised
      attribute
        AddClassification
        AttributeSelection
        ClassOrder
        Discretize
        NominalToBinary
      instance
        Resample
        SpreadSubsample
        StratifiedRemoveFolds
    unsupervised
      attribute
        Add
        AddCluster
        AddExpression
        AddID
        AddNoise
        AddUserFields
        AddValues
        RenameAttribute
        Reorder
        ReplaceMissingValues
        ReplaceMissingWithUserConstant
        SortLabels
        Standardize
        StringToNominal
        StringToWordVector
        SwapValues
        TimeSeriesDelta
        TimeSeriesTranslate
      instance
        NonSparseToSparse
        Randomize
        RemoveFolds
        RemoveFrequentValues
        RemoveMisclassified
        RemovePercentage
        RemoveRange
        RemoveWithValues
        Resample
        ReservoirSample
        SparseToNonSparse
        SubsetByExpression
```

## WEKA en la línea de órdenes

- Otros filtros

```
$> more spambase.arff
../..
@data
0,0.64,0.64,0,0.32,0,0,0,0,0,0,0.64,0,0,0,0.32,0,1.29,1.93,0,0.96,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0.778,0,0,
3.756,61,278,'spam'
../..

$> java weka.filters.unsupervised.instance.NonSparseToSparse
-i spambase.arff -o spambase.sparse.arff

$> more spambase.sparse.arff
../..
@data
{1 0.64,2 0.64,4 0.32,11 0.64,15 0.32,17 1.29,18 1.93,20 0.96,
51 0.778,54 3.756,55 61,56 278,57 spam}
../..
```

## WEKA en la línea de órdenes

- Minería de texto – de textos a términos
  - Es necesario transformar textos en vectores de pesos de términos
  - Filtro StringToWordVector
    - Múltiples opciones: tokenización, pesos, raices, lista de parada, etc.

## WEKA en la línea de órdenes

- Filtro StringToWordVector

```
$> java weka.filters.unsupervised.attribute.StringToWordVector -h
Filter options:
-C
        Output word counts rather than boolean word presence.
-T
        Transform the word frequencies into log(1+fij)
        where fij is the frequency of word i in jth document(instance).
-I
        Transform each word frequency into:
        fij*log(num of Documents/num of documents containing word i)
          where fij if frequency of word i in jth document(instance).
-M <int>
        The minimum term frequency (default = 1).
-W <number of words to keep>
        Specify approximate number of word fields to create.
        Surplus words will be discarded. (default: 1000)
-N
        Whether to 0=not normalize/1=normalize all data/2=normalize test
        data only to average length of training documents (default
        0=don't normalize).
```

## WEKA en la línea de órdenes

- Filtro StringToWordVector

```
$> java weka.filters.unsupervised.attribute.StringToWordVector -h
Filter options:
-L
        Convert all tokens to lowercase before adding to the dictionary.
-S
        Ignore words that are in the stoplist.
-stemmer <spec>
        The stemmering algorihtm (classname plus parameters) to use.
-stopwords <file>
        A file containing stopwords to override the default ones.
        Using this option automatically sets the flag ('-S') to use the
        stoplist if the file exists.
        Format: one stopword per line, lines starting with '#'
        are interpreted as comments and ignored.
-tokenizer <spec>
        The tokenizing algorihtm (classname plus parameters) to use.
        (default: weka.core.tokenizers.WordTokenizer)
```

## WEKA en la línea de órdenes

- Filtro StringToWordVector

```
$> more smsspam.small.arff
@relation sms_test
@attribute spamclass {spam,ham}
@attribute text String
@data
ham,'U dun say so early hor... U c already then say...'
../..
$> java weka.filters.unsupervised.attribute.StringToWordVector
   -i smsspam.small.arff -o smsspam.small.vector.arff

$> more smsspam.small.vector.arff
@relation ../..
@attribute spamclass {spam,ham}
@attribute 1 numeric
@attribute Account numeric
../..
@data
{0 ham,259 1,312 1,876 1,1016 1,1274 1,1327 1}
../..
```

- Filtro StringToWordVector

```
$> java weka.classifiers.bayes.NaiveBayes -t smsspam.small.arff -c first
weka.core.UnsupportedAttributeTypeException: weka.classifiers.bayes
NaiveBayes: Cannot handle string attributes!
        at weka.core.Capabilities.test(Capabilities.java:979)
        at weka.core.Capabilities.test(Capabilities.java:868)

$> java weka.classifiers.bayes.NaiveBayes -t smsspam.small.vector.arff
   -c first
Naive Bayes Classifier
../..
=== Stratified cross-validation ===
Correctly Classified Instances         186               93      %
Incorrectly Classified Instances        14                7      %
../..
=== Confusion Matrix ===
   a   b   <-- classified as
  22  11 |   a = spam
   3 164 |   b = ham
```

- Filtro StringToWordVector

```
$> more smsspam.small.arff
relation sms_test
@attribute spamclass {spam,ham}
@attribute text String
@data
ham,'U dun say so early hor... U c already then say...'
../..
$> java weka.filters.unsupervised.attribute.StringToWordVector
   -i smsspam.small.arff -o smsspam.small.vector.arff

$> more smsspam.small.vector.arff
@relation ../..
@attribute spamclass {spam,ham}
@attribute 1 numeric
@attribute Account numeric
../..
@data
{0 ham,259 1,312 1,876 1,1016 1,1274 1,1327 1}
../..
```

## API de desarrollo de WEKA

## API de desarrollo de WEKA

- Las clases anteriores son accesibles programaticamente
- Necesitamos saber, al menos
  - Cargar un dataset
  - Evaluar un modelo o clasificador
  - Entrenar, cargar y guardar un modelo o clasificador
  - Construir un ejemplar
  - Clasificar un ejemplar

## API de desarrollo de WEKA

- Cargar un dataset

```
import weka.core.Instances;
import weka.core.converters.ArffLoader.ArffReader;
import java.io.*;
// ../..

    BufferedReader reader = new
            BufferedReader(new FileReader(fileName));
    ArffReader arff = new ArffReader(reader);
    Instances trainData = arff.getData();
    reader.close();
```

- Evaluar un clasificador

```
import weka.core.Instances;
import weka.classifiers.Evaluation;
import java.util.Random;
import weka.classifiers.bayes.NaiveBayes;
// ../..

        // trainData contains previous instances
        trainData.setClassIndex(trainData.numAttributes()-1);
        classifier = new NaiveBayes();
        Evaluation eval = new Evaluation(trainData);
        eval.crossValidateModel(classifier, trainData, 4, new Random(1));
        System.out.println(eval.toSummaryString());
        System.out.println(eval.toClassDetailsString());
```

## API de desarrollo de WEKA

- Entrenar, cargar y guardar un clasificador

```
import weka.core.Instances;
import weka.classifiers.bayes.NaiveBayes;
import java.io.*;
// ../..

       ObjectInputStream in = new
           ObjectInputStream(new FileInputStream(fileName));
       Object tmp = in.readObject();
       NaiveBayes classifier = (NaiveBayes) tmp;
       in.close();

       // trainData contains previous instances
       trainData.setClassIndex(trainData.numAttributes()-1);
       classifier = new NaiveBayes();
       classifier.buildClassifier(trainData);

       ObjectOutputStream out = new
           ObjectOutputStream(new FileOutputStream(fileName));
       out.writeObject(classifier);
       out.close();
```

## API de desarrollo de WEKA

- Para construir un ejemplar
  - Crear el dataset de referencia
  - Crear el ejemplar
  - *Enlazar el ejemplar al dataset*
  - Dar valores a los atributos

## API de desarrollo de WEKA

- Construir un ejemplar – dataset de referencia

```
@relation weather

@attribute outlook {sunny, overcast, rainy}
@attribute temperature real
@attribute humidity real
@attribute windy {TRUE, FALSE}
@attribute play {yes, no}

@data
sunny,85,85,FALSE,no
sunny,80,90,TRUE,no
../..
```

# API de desarrollo de WEKA

- Construir un ejemplar – Construir el dataset
  - Atributos nominales

```
import weka.core.*;
import java.util.List;
import java.util.ArrayList;
// ../..

        // Create the header
        List attributeList = new ArrayList(5);

        // Atribute "outlook"
        List values = new ArrayList(3);
        values.add("sunny");
        values.add("overcast");
        values.add("rainy");
        Attribute attribute = new Attribute("outlook", values);
        attributeList.add(attribute);
```

## API de desarrollo de WEKA

- Construir un ejemplar – Construir el dataset
  - Atributos numéricos

```
import weka.core.*;
import java.util.List;
import java.util.ArrayList;
// ../..

      // Create the header
      List attributeList = new ArrayList(5);

      // Atribute "temperature" - default numeric
      attribute = new Attribute("temperature");
      attributeList.add(attribute);
```

## API de desarrollo de WEKA

- Construir un ejemplar – Construir el dataset
  - Cabecera del dataset

```
import weka.core.*;
import java.util.ArrayList;
// ../..

        // Build instance set with just one instance
        Instances instances = new Instances("Test relation",
            (java.util.ArrayList<Attribute>) attributeList, 1);
        // Set class index
        instances.setClassIndex(instances.numAttributes()-1);
```

## API de desarrollo de WEKA

- Construir un ejemplar – Construir el dataset
  - Alternativa más simple
    - Almacenar el encabezado en un archivo
    - Leer el encabezado como un dataset

## API de desarrollo de WEKA

- Construir y enlazar un ejemplar

```
import weka.core.*;
// ../..

      // Create and add the instance
      DenseInstance instance = new DenseInstance(5);
      instance.setDataset(instances);
```

## API de desarrollo de WEKA

- Dar valores al ejemplar

```
import weka.core.*;
// ../..

        // Assumed the instance is in CSV:
        // "sunny,85,85,FALSE", class (last) undefined
        String[] stringValues = csvInstance.split(",");
        instance.setValue(0, stringValues[0]);
        instance.setValue(1, Integer.parseInt(stringValues[1]));
        instance.setValue(2, Integer.parseInt(stringValues[2]));
        instance.setValue(3, stringValues[3]);
        instances.add(instance);
```

- Clasificar un ejemplar

```
import weka.core.*;
// ../..

      double pred =
         classifier.classifyInstance(instances.instance(0));
      System.out.println("Class predicted: " +
         instances.classAttribute().value((int) pred));
```

## Ejemplos de programas

## Ejemplos de programas

- Entrenamiento y almacenamiento del modelo

```
$> java MyLearner weather.numeric.arff weather.NB.data
===== Loaded dataset: weather.numeric.arff =====
Correctly Classified Instances          7               50      %
Incorrectly Classified Instances        7               50      %
../..
=== Detailed Accuracy By Class ===
  TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
  0,556    0,600    0,625      0,556    0,588      -0,043   0,333     0,578     yes
../..
===== Evaluating on filtered (training) dataset done =====
Naive Bayes Classifier
                 Class
Attribute          yes      no
                (0.63)  (0.38)
===============================
outlook
  sunny            3.0     4.0
../..
  [total]         12.0     8.0
../..
===== Training on filtered (training) dataset done =====
===== Saved model: weather.NB.data =====
```

## Ejemplos de programas

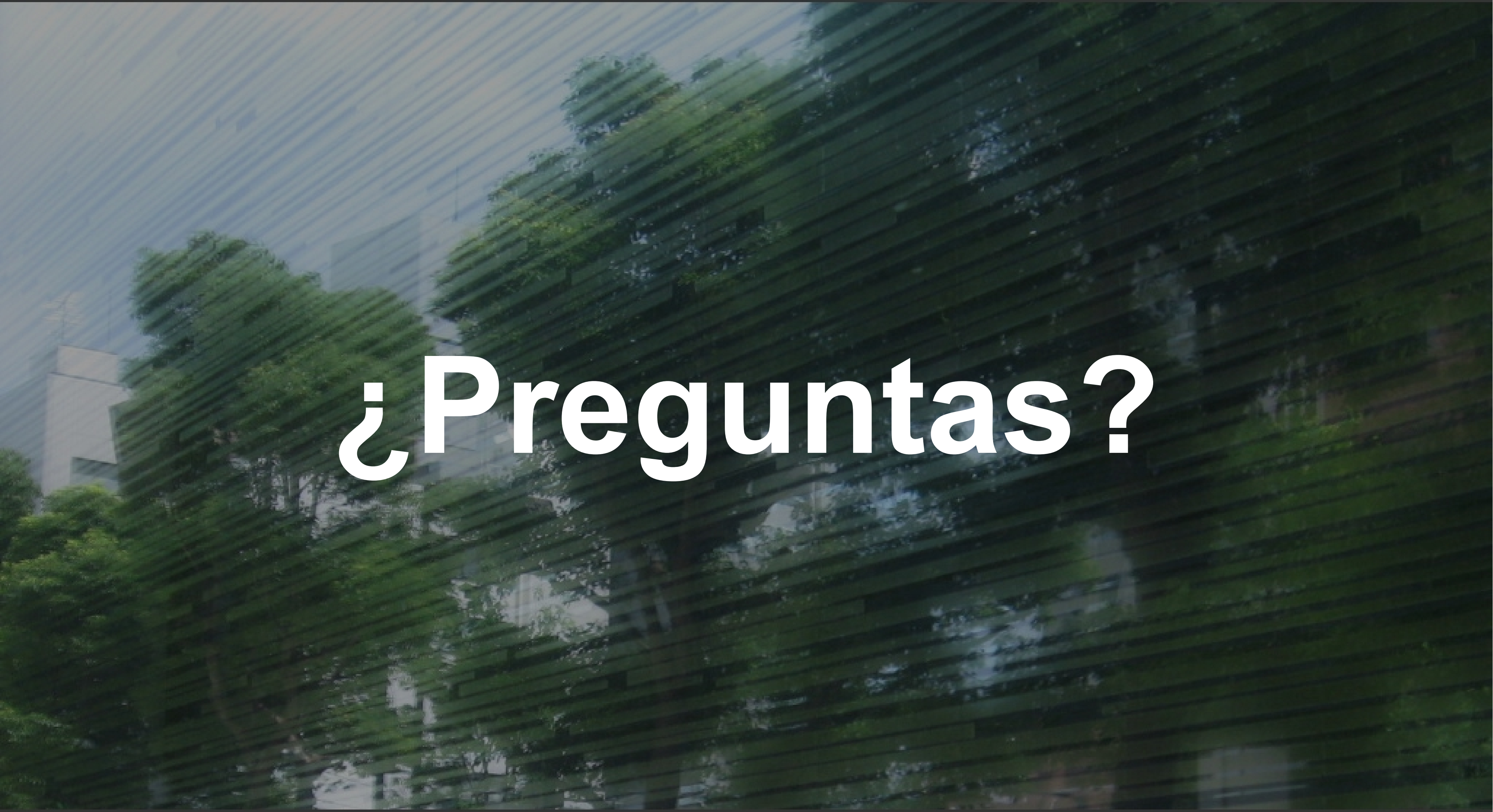
- Carga de modelo y clasificación de un ejemplar

```
$> java MyClassifier "sunny,85,85,FALSE" weather.NB.data weather.numeric.header.arff
===== Loaded model: weather.NB.data =====
===== Instance created with reference dataset =====
@relation 'Test relation'

@attribute outlook {sunny,overcast,rainy}
@attribute temperature numeric
@attribute humidity numeric
@attribute windy {TRUE,FALSE}
@attribute play {yes,no}

@data
sunny,85,85,FALSE,?
===== Classified instance =====
Class predicted: no
```

## Referencias

## Referencias

- Presentación, ejemplos de órdenes, datasets y código
  - https://github.com/jmgomezh/tmweka/tree/master/MadridJUG
- Mi página de minería de texto con WEKA
  - http://www.esp.uem.es/jmgomez/tmweka
- Referencias de la wiki de WEKA
  - http://weka.wikispaces.com/Use+WEKA+in+your+Java+code
  - http://weka.wikispaces.com/Programmatic+Use



# ¿Preguntas?