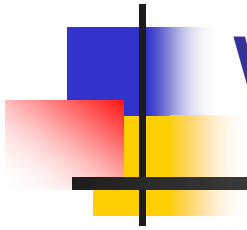
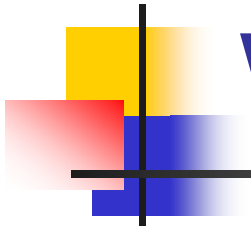


WEKA: Struttura ed Estensioni

Esercitazione Data Mining

01/03/2007

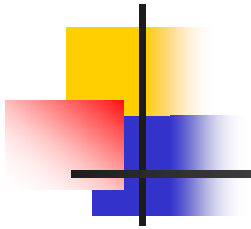
Howard Scordio



Weka Dataset

- Rappresentazione:
  - Su disco: formato ARFF (file .arff)
  - In memoria centrale: Instances
- Il formato arff ha due distinte sezioni
  - Intestazione (Metadati)
    - @relation (nome della relazione)
    - @attribute (lista degli attributi)
  - Corpo (Le istanze)
    - @data

ARFF: tipi di dati



Numeric, Real

@attribute <name> numeric

@attribute <name> real

Nominal Specification, lista di valori:

@attribute <name> {<nominal-name1>, <nominal-name2>,...}

String:

@attribute <name> string

Date:

@attribute <name> date [<date-format>]

Relational

@attribute <name> relational

<further attribute definitions>

@end <name>

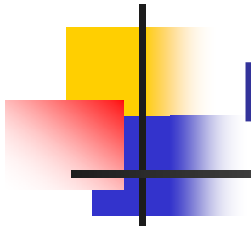
Esempio di file ARFF

```
% This is a toy example, the UCI weather dataset.
% Any relation to real weather is purely coincidental.
```

```
@relation golfWeatherMichigan_1988/02/10_14days
```

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

```
@data
sunny,FALSE,85,85,no
sunny,TRUE,80,90,no
overcast,FALSE,83,86,yes
rainy,FALSE,70,96,yes
rainy,FALSE,68,80,yes
```



Esempio di file ARFF (multi-relational)

```
@RELATION relational_example
   @ATTRIBUTE attr1 integer
   @ATTRIBUTE rel relational
       @ATTRIBUTE rel_attr1 integer
       @ATTRIBUTE rel_attr2 string
   @END rel

   @DATA

   1, "1,'a' \n 2,'b' \n 3,'a' \n 4,'c'"
   2, "1,'a' \n 2,'b' \n 3,'b' \n 4,'b'"
   3, "1,'c' \n 2,'a' \n 3,'c'"
   4, "1,'b' \n 2,'b'"
```

ARFF formato sparso

I dati con valore 0 non vengono rappresentati

L'intestazione non cambia, cambia il modo di definire la sezione @data

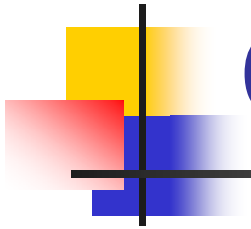
Standard

```
@data
0, X, 0, Y, "class A"
0, 0, W, 0, "class B"
```

Sparso

```
@data
{1 X, 3 Y, 4 "class A"}
{2 W, 4 "class B"}
```

Weka Architettura



Classe Instances

La classe *weka.core.Instances* è l'implementazione di un dataset in Weka.

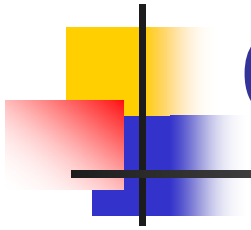
Un oggetto *Instances* è una collezione di esempi della classe *weka.core.Instance* (tuple) e di oggetti della classe *weka.core.Attribute* (metadati)

Creare un Instances a partire da un file arff

```
Instances dataset = new Instances
          (new FileReader("URL del file arff"));
```

Metodi principali di Instances:

- Instance instance(int index)
- void setClassIndex(int index)
- Attribute classAttribute()
- void add (Instance instance)
- Attribute attribute(int index)
- int numAttribute()
- int numInstances()



Classe Instance

Un oggetto Instance mappa una tupla

Una istanza in Weka è rappresentata con un array di **`double`**

Ogni valore **`double`** rappresenta:

- Il valore che l'attributo assume in quell'istanza se si tratta di un attributo numerico (**`int`** o **`double`**) o di una data
- L'indice relativo ad un array di **`Object`** (contenuto nella classe *weka.core.Attribute*) altrimenti
  - L'array contiene *String* se l' attributo è di tipo nominale o stringa, contiene oggetti di tipo *Instances* se l'attributo è relational

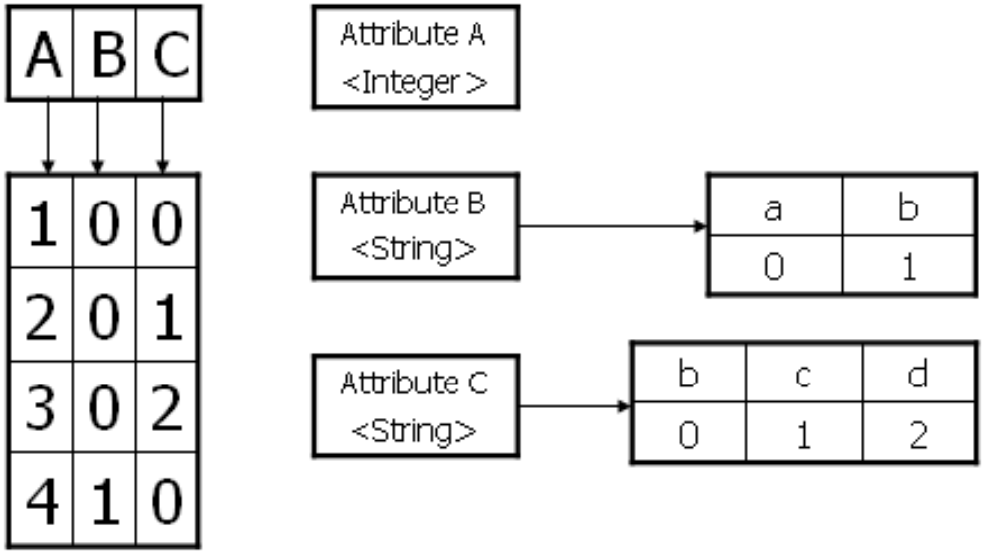
Metodi di Instance

- Instances dataset()
- setValue(int attIndex, double value)
- Instances relationalValue(int attIndex)
- double[] toDoubleArray()

Esercizio 1

@relational esempio
@attribute A integer
@attribute B string
@attribute C string

@data
1 , ‘a’, ‘b’
2 , ‘a’, ‘c’
3, ‘a’ , ‘d’
4, ‘b’, ‘b’

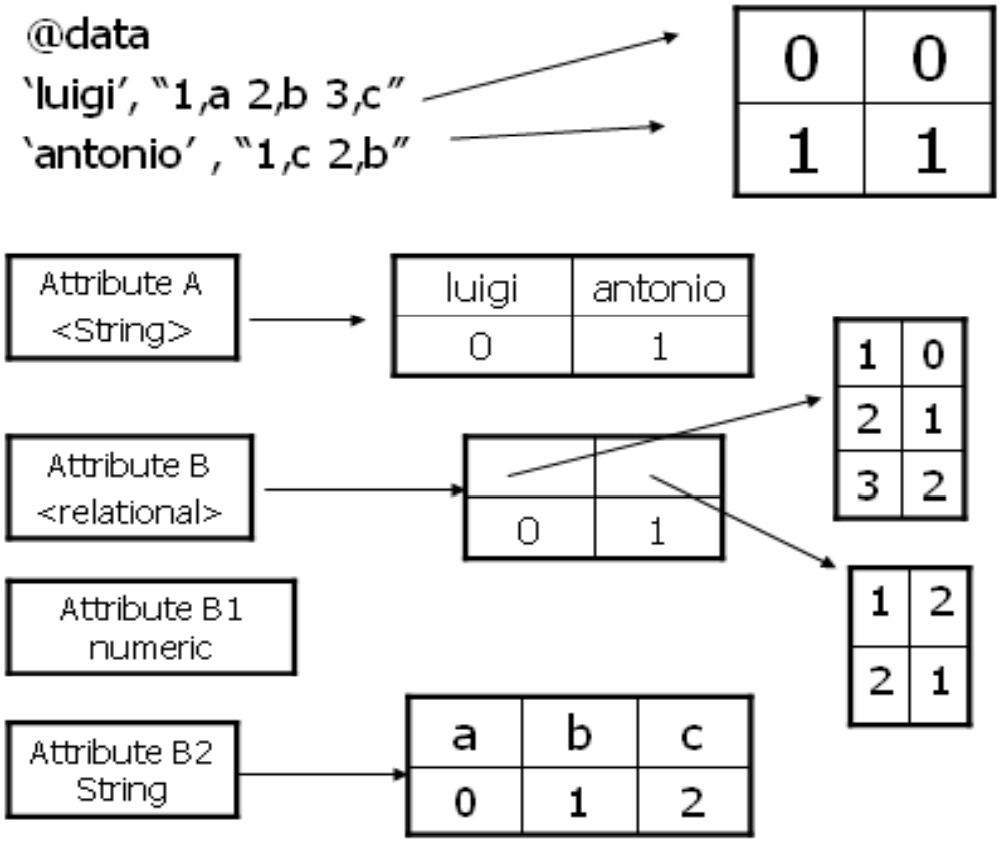


Come rappresenta WEKA in memoria questo dataset?

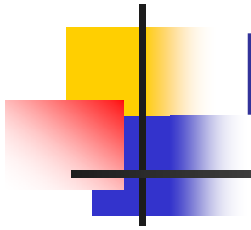
Esercizio 2

@relational esempio
@attribute A string
@attribute B relational
    @attribute B1 integer
    @attribute B2 String
@end B

@data
‘luigi’, “1,a 2,b 3,c”
‘antonio’ , “1,c 2,b”



Come rappresenta WEKA in memoria questo dataset?

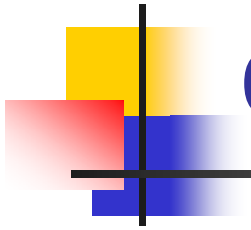


Estendere Weka: Classifier

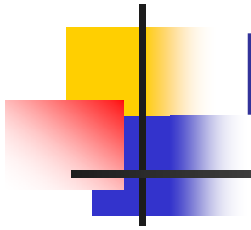
Creare un nuovo Classificatore

- Estendere la classe *weka.classifiers.Classifier*
- Implementare i metodi
  - void buildClassifier(Instances): genera il modello di classificazione
  - double classifyInstance(Instance): restituisce l'indice della classe assegnata (dal modello) all'istanza in ingresso
  - double[] distributionForInstance(Instance): genera una distribuzione di probabilità per ciascuna classe

Esempio di utilizzo di un classificatore generico



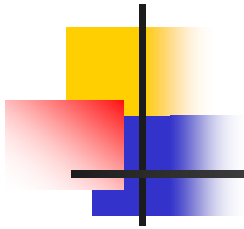
```
Instances trainingData=... //recupera le istanze di training
Classifier classifier= // crea un nuovo classificatore
classifier.buildClassifier(trainingData);
Instance toClassifyInstance=... recupera l'istanza da classificare
Attribute classAttribute= trainingData.classAttribute();
String classLabel =
classAttribute.value((int) classifier.classifyInstance(toClassifyInstance));
System.out.println("l'istanza "+ toClassifyInstance.toString()+
" è stata classificata come: "+ classLabel );
```



Estendere Weka: Clusterer

Creare un nuovo Clusterer

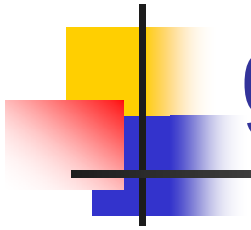
- Estendere la classe *weka.clusterer.Clusterer*
- Implementare i metodi
  - void buildClusterer(Instances): genera il modello di clustering
  - int clusterInstance(Instance): restituisce l'indice del cluster assegnato (dal modello) all'istanza in ingresso
  - double[] distributionForInstance(Instance): genera una distribuzione di probabilità per ciascun cluster



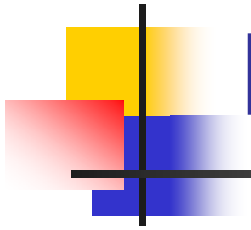
Estendere Weka: Filter

- Creare un nuovo Filtro
  - Estendere la classe *weka.filters.Filter*
  - Se si implementa un batch Filter
    - boolean input(instance): accetta l'istanza da processare e la bufferizza
    - boolean batchFinished(): avvisa il filtro che questo gruppo (batch) di input è finito, filtra le istanze e le inserisce nella coda.
  - Nel caso si implementi un "on line" Filter
    - boolean input(instance): accetta l'istanza da processare, la processa e la inserisce nella coda.
  - In entrambi i casi il metodo:
    - Instance output(): estrae dalla coda l'istanza convertita e la restituisce
  - Il metodo setInputFormat(Instances) serve per impostare il formato delle tuple in ingresso, mentre il metodo
  - Instances getOutputFormat() restituisce il formato delle istanze filtrate.

Esempio di utilizzo di un filtro generico



```
Filter filter = ..some type of filter.. *
Instances instances = ..some instances.. *
for (int i = 0; i < data.numInstances(); i++)
   filter.input(data.instance(i));
filter.batchFinished();
Instances newData = filter.getOutputFormat();
Instance processed;
while ((processed = filter.output()) != null)
   newData.add(processed);
..do something with newData..
```



Estendere Weka: Associator

Creare un nuovo Associator

Estendere la classe *weka.associations.Associator*

Implementare il metodo

void buildAssociations(Instances): genera l'associatore

Esercizio 1

1) Estendere Weka con la classe *weka.classifiers.trees.Id3* che implementi l'algoritmo Id3

2) Testare l'algoritmo sul dataset playtennis

Day	*Outlook*	*Temperature*	*Humidity*	*Wind*	*PlayTennis*
1	Sunny	Hot	High	Light	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Light	Yes
4	Rain	Mild	High	Light	Yes
5	Rain	Cool	Normal	Light	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Light	No
9	Sunny	Cool	Normal	Light	Yes
10	Rain	Mild	Normal	Light	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Light	Yes
14	Rain	Mild	High	Strong	No

Id3 Algorithm in Weka

```
public class Id3  extends Classifier
{
 /** The node's successors. */
 private Id3[] m_Successors;

/** Attribute used for splitting. */
 private Attribute m_Attribute;

 /** Class value if node is leaf. */
 private double m_ClassValue;

 /** Class distribution if node is leaf. */
 private double[] m_Distribution;
```

```
public void buildClassifier(Instances data)
    throws Exception {

// remove instances with missing class
   data = new Instances(data);
   data.deleteWithMissingClass();

   makeTree(data);
 }
```

Make Tree

```
private void makeTree(Instances data)
 throws Exception {
   // Check if no instances have
   // reached this node.
   if (data.numInstances() == 0) {
     m_Attribute = null;
     m_ClassValue = Instance.missingValue();
     m_Distribution = new
      double[data.numClasses()];
     return;
   }
   // Compute attribute with maximum
   // information gain.
   double[] infoGains = new
      double[data.numAttributes()];
   Enumeration attEnum =
      data.enumerateAttributes();
   while (attEnum.hasMoreElements()) {
     Attribute att = (Attribute) attEnum.nextElement();
     infoGains[att.index()] =
                computeInfoGain(data, att);
   }
   m_Attribute =
      data.attribute(Utils.maxIndex(infoGains));
   // Make leaf if information gain is zero.
   // Otherwise create successors.
   if (Utils.eq(infoGains[m_Attribute.index()], 0)) {
     m_Attribute = null;
     m_Distribution = new double[data.numClasses()];
     Enumeration instEnum =
               data.enumerateInstances();
     while (instEnum.hasMoreElements()) {
       Instance inst = (Instance)
               instEnum.nextElement();
       m_Distribution[(int) inst.classValue()]++;
     }
     Utils.normalize(m_Distribution);
     m_ClassValue = Utils.maxIndex(m_Distribution);
     } else {
     Instances[] splitData = splitData(data, m_Attribute);
     m_Successors = new
               Id3[m_Attribute.numValues()];
     for (int j = 0; j < m_Attribute.numValues(); j++) {
       m_Successors[j] = new Id3();
       m_Successors[j].makeTree(splitData[j]);
     }
   }
 }
```

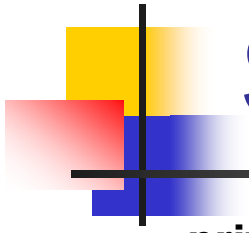
Info Gain

```
private double computeInfoGain(Instances data,
      Attribute att)
   throws Exception {

   double infoGain = computeEntropy(data);
   Instances[] splitData = splitData(data, att);
   for (int j = 0; j < att.numValues(); j++) {
    if (splitData[j].numInstances() > 0) {
      infoGain -= ((double)
      splitData[j].numInstances() /
                  (double) data.numInstances()) *
        computeEntropy(splitData[j]);
    }
     }
   return infoGain;
 }
```

$$Gain(D,A) \equiv -H(D) - \sum_{v \in values(A)} \left[\frac{|D_v|}{|D|} \bullet H(D_v) \right]$$

```
private double computeEntropy(Instances data)
      throws Exception {

   double [] classCounts = new
      double[data.numClasses()];
   Enumeration instEnum =
      data.enumerateInstances();
   while (instEnum.hasMoreElements()) {
     Instance inst = (Instance)
      instEnum.nextElement();
    classCounts[(int) inst.classValue()]++;
   }
   double entropy = 0;
   for (int j = 0; j < data.numClasses(); j++) {
     if (classCounts[j] > 0) {
       entropy -= classCounts[j] *
      Utils.log2(classCounts[j]);
     }
   }
   entropy /= (double) data.numInstances();
   return entropy +
      Utils.log2(data.numInstances());
 }
```

$$H(D) \equiv -p_+ \log_b(p_+) - p_- \log_b(p_-)$$



Split Data

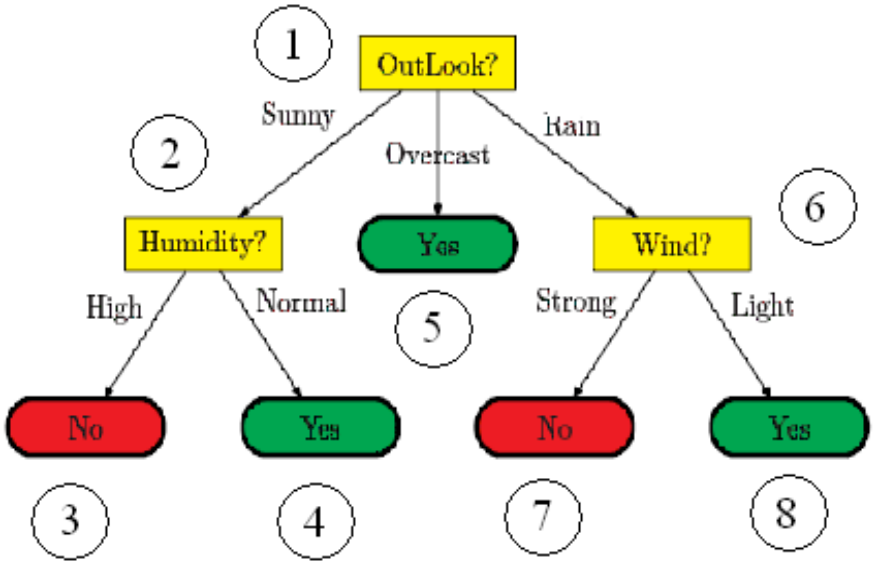
```
private Instances[] splitData(Instances data, Attribute att) {

  Instances[] splitData = new Instances[att.numValues()];
  for (int j = 0; j < att.numValues(); j++) {
    splitData[j] = new Instances(data, data.numInstances());
  }
  Enumeration instEnum = data.enumerateInstances();
  while (instEnum.hasMoreElements()) {
    Instance inst = (Instance) instEnum.nextElement();
    splitData[(int) inst.value(att)].add(inst);
  }
  for (int i = 0; i < splitData.length; i++) {
    splitData[i].compactify();
  }
  return splitData;
}
```

ClassifyInstace & DistributionForInstance

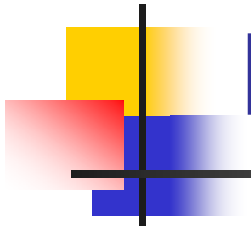
```
public double classifyInstance
(Instance instance) {
if (m_Attribute == null) {
      return m_ClassValue;
    } else {
      return m_Successors[(int)
    instance.value(m_Attribute)].
        classifyInstance(instance);
    }
  }
```

```
public double[] distributionForInstance
    (Instance instance) {
    if (m_Attribute == null) {
        return m_Distribution;
    } else {
      return m_Successors[(int)
    instance.value(m_Attribute)].
        distributionForInstance(instance);
    }
  }
```

Play Tennis: Risultato



Info Gains					
NODO	**Outlook**	**Temp.**	**Humidity**	**windy**	**play**
1	**0.2467**	**0.0292**	**0.1518**	**0.0481**	**0.0**
2	**0.0**	**0.5709**	**0.9709**	**0.0199**	**0.0**
3	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
4	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
5	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
6	**0.0**	**0.0199**	**0.9709**	**0.0**	**0.0**
7	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**
8	**0.0**	**0.0**	**0.0**	**0.0**	**0.0**



Esercizio 2

Estendere Weka con la classe *weka.filters.unsupervised.attribute.ReplaceMissingValues* che implementi un filtro che sostituisce tutti i valori mancanti (nominali o numerici) di un dataset con la relativa moda o media.

Replace Missing Values

```
public class ReplaceMissingValues
  extends PotentialClassIgnorer
  implements UnsupervisedFilter {

/** The modes and means */
private double[] m_ModesAndMeans =
      null;

public boolean setInputFormat(Instances
      instanceInfo)  throws Exception {
super.setInputFormat(instanceInfo);
    setOutputFormat(instanceInfo);
    m_ModesAndMeans = null;
    return true;
 }
```

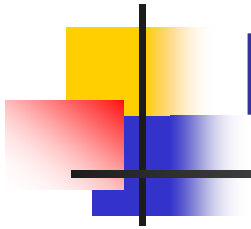
```
public boolean input(Instance instance){

    if (getInputFormat() == null) {
      throw new IllegalStateException("No input
      instance format defined");
    }
    if (m_NewBatch) {
      resetQueue();
      m_NewBatch = false;
    }
    if (m_ModesAndMeans == null) {
      bufferInput(instance);
      return false;
    } else {
      convertInstance(instance);
      return true;
    }
}
```

BatchFinished & ConvertInstance

```
public boolean batchFinished() {
  if (getInputFormat() == null) {
      throw new IllegalStateException("No input
      instance format defined");
   }
   if (m_ModesAndMeans == null) {
    // Compute modes and means
    m_ModesAndMeans = new double[getInputFormat()
      .numAttributes()];
    for (int i = 0; i < getInputFormat() .numAttributes();
      i++) {
   if (getInputFormat() .attribute(i).isNominal() ||
      getInputFormat().attribute(i).isNumeric()) {
          m_ModesAndMeans[i] =
      getInputFormat().meanOrMode(i);
     }
    }
    // Convert pending input instances
    for(int i = 0; i < getInputFormat() .numInstances();
      i++) {
    Instance current = getInputFormat() .instance(i);
    convertInstance(current);
     }
  }
  flushInput();// Free memory
  m_NewBatch = true;
  return (numPendingOutput() != 0);
}
```

```
private void convertInstance(Instance instance) throws
      Exception {
    Instance newInstance = new Instance(instance.weight(),
      instance.toDoubleArray());
   for(int j = 0; j < m_InputFormat.numAttributes(); j++){
       if (instance.isMissing(j) &&
        (m_InputFormat.attribute(j).isNominal() ||
          m_InputFormat.attribute(j).isNumeric())) {
            newInstance.setValue(j, m_ModesAndMeans[j]);
      }
    }
   newInstance.setDataset(instance.dataset());
    push(newInstance);
}
```

```
public static void main(String [] argv) {
    try {
      if (Utils.getFlag('b', argv)) {
 Filter.batchFilterFile(new ReplaceMissingValues(), argv);
      } else {
Filter.filterFile(new ReplaceMissingValues(), argv);
      }
    } catch (Exception ex) {
      System.out.println(ex.getMessage());
    }
  }
```

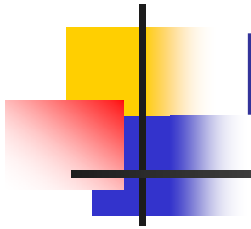


Parametri per gli algoritmi

Se un particolare algoritmo ha bisogno di parametri che devono essere specificati dall'utente (es. il *K* del *K-Means*) bisogna che la classe che identifica l'algoritmo:

- contenga una variabile di classe (una per ogni parametro) e i relativi metodi accessori e mutatori.
- Implementi l'interfaccia *OptionHandler* (due soli metodi *listOptions*, *setOptions*)

Se ci si attiene a queste specifiche sarà Weka stesso (mediante interfaccia grafica) a richiedere all'utente i valori relativi ai parametri.



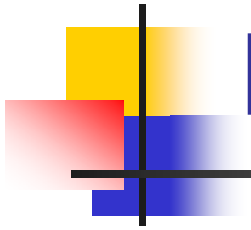
File di configurazione

GenericObjectEditor.props: file contenente la lista di tutti gli algoritmi implementati all'interno di Weka (quelli visualizzati dalla GUI).

GenericsPropertiesCreator.props: file contenente la liste dei package (e relativa struttura) presente in Weka.

Se si estende Weka aggiungendo un nuovo algoritmo (una sola classe) in un particolare package già presente in weka, bisognerà solamente aggiungere la il nome della classe nel file GenericObjectEditor.props

Se invece si aggiunge un algoritmo che si trova in un package definito dall'utente, bisognerà modificare anche il file GenericsPropertiesCreator.props aggiungendo il package nella relativa gerarchia se presente o crearne una nuova.



Esercizio 3

Si implementi un semplice riconoscitore di spam basato sulle seguenti parole chiave sospette: product, only, offer, great, amazing, phantastic, opportunity, buy, now.

Il programma, dato in input uno o più messaggi (memorizzati in file di testo) già classificati come Hit (da accettare) o Miss (spam), deve essere in grado di riconoscere se un nuovo messaggio va accettato o rigettato.

Esempio

Msg1.txt

product only offer great
amazing phantastic
opp y now
pro only offer great
amazing phantastic
opportunity buy now
product only offer great

Msg2.txt

product only offer great amazing
phantastic opportunity buy now
product only offer great amazing
phanta tunity buy now
product only offer great amazing
phantastic opportunity buy now
product only offer great

Msg3.txt

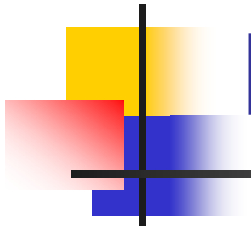
Questo pio di
mes che non
contiene parole chiave.

Msg4.txt

phantastic opportunity buy now
phantastic opportunity buy now
phantastic o nity buy now
amazing pha stic opportunity
amazing phantastic opportunity
amazing phantastic opportunity

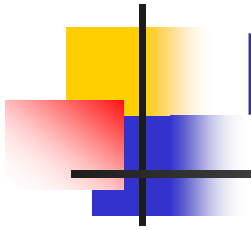
Msg5.txt

Questo file .txt non contiene
parole c ave



MessageClassifier

- L’utente inizialmente deve istruire il learner
  - Esegue tanti run (main) per quante sono le tuple (i messaggi) contenute nel training set
    - Ad ogni run specifica l’url del messaggio (parametro -m) e la rispettiva classe (con il parametro –c, hit o miss)
    - Con il parametro –t infine seleziona il percorso dove si trova il learner (classificatore) che bisogna andare ad istruire
- L’utente può decidere se continuare ad istruire il learner o utilizzarlo per la classificazione di un messaggio utilizzando opportunamente i parametri
  - Il parametro –c è opzionale
    - Se –c non è presente il classificatore viene utilizzato per classificare altrimenti viene istruito con una nuova istanza di training



MessageClassifier code

```
public class MessageClassifier
    implements Serializable {

/* The keywords. */
private final String[] m_Keywords =
    {"product", "only", "offer", "great",
    "amazing", "phantastic", "opportunity",
    "buy", "now"};
/* The training data. */
private Instances m_Data = null;
/* The filter. */
private Filter m_Filter = new Discretize();
/* The classifier. */
private Classifier m_Classifier = new IBk();
```

```
public MessageClassifier() throws Exception {
String nameOfDataset = "MessageClassificationProblem";
// Create numeric attributes.
FastVector attributes = new
      FastVector(m_Keywords.length + 1);
for (int i = 0 ; i < m_Keywords.length; i++) {
  attributes.addElement(new Attribute(m_Keywords[i]));
}
// Add class attribute.
FastVector classValues = new FastVector(2);
classValues.addElement("miss");
classValues.addElement("hit");
attributes.addElement(
      new Attribute("Class", classValues));
// Create dataset with initial capacity of 100,
// and set index of class.
m_Data = new Instances(
       nameOfDataset, attributes, 100);
m_Data.setClassIndex(m_Data.numAttributes() - 1);
}
```

UpdateModel & ClassifyInstance

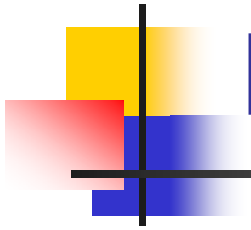
```
public void updateModel(String
    message, String classValue)
throws Exception {
// Convert message string into
    instance.
Instance instance =
    makeInstance(cleanupString(message
    ));
// Add class value to instance.
instance.setClassValue(classValue);
// Add instance to training data.
m_Data.add(instance);
// Use filter.
m_Filter.setInputFormat(m_Data);
Instances filteredData =
    Filter.useFilter(m_Data, m_Filter);
// Rebuild classifier.
m_Classifier.buildClassifier(filteredData);
}
```

```
public void classifyMessage(String message)
    throws Exception {
// Check if classifier has been built.
if (m_Data.numInstances() == 0) {
throw new Exception("No classifier available.");
}
// Convert message string into instance.
Instance instance =
    makeInstance(cleanupString(message));
// Filter instance.
m_Filter.input(instance);
Instance filteredInstance = m_Filter.output();
// Get index of predicted class value.
double predicted =
    m_Classifier.classifyInstance(filteredInstance);
// Classify instance.
System.err.println("Message classified as : " +
m_Data.classAttribute().value((int)predicted));
}
```

MakeInstance & CleanupString

```
private Instance makeInstance(String messageText) {
StringTokenizer tokenizer = new
      StringTokenizer(messageText);
Instance instance = new Instance
                (m_Keywords.length + 1);
String token;
// Initialize counts to zero.
for (int i = 0; i < m_Keywords.length; i++) {
    instance.setValue(i, 0);
}
// Compute attribute values.
while (tokenizer.hasMoreTokens()) {
     token = tokenizer.nextToken();
for (int i = 0; i < m_Keywords.length; i++) {
if (token.equals(m_Keywords[i])) {
   instance.setValue(i, instance.value(i) + 1.0);
break;
                }
      }
}
// Give instance access to attribute
// information from the dataset.
instance.setDataset(m_Data);
return instance;
}
```

```
private String cleanupString(String messageText) {
char[] result = new char[messageText.length()];
int position = 0;
for (int i = 0; i < messageText.length(); i++) {
if (Character.isLetter(messageText.charAt(i)) ||
Character.isWhitespace(messageText.charAt(i))) {
result[position++] =
      Character.toLowerCase(messageText.charAt(i));
}
}
return new String(result);
}
```



Main

- Interroga i parametri per
  - Stabilire se il classificatore deve valutare un istanza o aggiornare il modello (tramite il parametro -c)
  - Recuperare il messaggio da classificare o da trattare come istanza di training (tramite il parametro -m)
  - Recuperare il classificatore (percorso specificato dal parametro -t)
  - Salvare il nuovo modello se il messaggio ricevuto era un istanza di training