EBook Gratuito

APPRENDIMENTO weka

Free unaffiliated eBook created from
Stack Overflow contributors.

#weka

Sommario

Di

Capitolo 1: Iniziare con weka

Osservazioni

Questa sezione fornisce una panoramica su cosa sia weka e perché uno sviluppatore potrebbe volerlo utilizzare.

Dovrebbe anche menzionare qualsiasi argomento di grandi dimensioni all'interno di weka e collegarsi agli argomenti correlati. Poiché la documentazione di weka è nuova, potrebbe essere necessario creare versioni iniziali di tali argomenti correlati.

Examples

Installazione o configurazione

Weka è una raccolta di algoritmi di apprendimento automatico per attività di data mining. Gli algoritmi possono essere applicati direttamente a un set di dati o richiamati dal proprio codice Java. Weka contiene strumenti per la pre-elaborazione dei dati, la classificazione, la regressione, il clustering, le regole di associazione e la visualizzazione. È anche adatto per lo sviluppo di nuovi schemi di apprendimento automatico.

Download e installazione di Weka

Esistono due versioni di Weka: Weka 3.8 è l'ultima versione stabile e Weka 3.9 è la versione di sviluppo. Per il sanguinamento, è anche possibile scaricare istantanee notturne.

Le versioni stabili ricevono solo correzioni di bug, mentre la versione di sviluppo riceve nuove funzionalità. Weka 3.8 e 3.9 dispongono di un sistema di gestione dei pacchetti che consente alla comunità di Weka di aggiungere nuove funzionalità a Weka. Il sistema di gestione dei pacchetti richiede una connessione Internet per scaricare e installare i pacchetti.

Puoi scaricare l'applicazione per Windows / MacOS / Linux qui .

Integra la libreria WEKA nel tuo codice:

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'
```

Leggi Iniziare con weka online: https://riptutorial.com/it/weka/topic/3699/iniziare-con-weka

Capitolo 2: Classificazione del testo

Examples

Classificazione del testo con LibLinear

- Crea istanze di allenamento dal file .arff

```
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);
```

- Usa **StringToWordVector** per trasformare i tuoi attributi di stringa in rappresentazione numerica:

  * Caratteristiche importanti di questo filtro:

    1. rappresentazione tf-idf
    2. derivanti
    3. wrods minuscoli
    4. stopwords
    5. rappresentazione in n-grammi *

```
    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);
```

- Applicare il filtro a trainingData: `trainingData = Filter.useFilter(trainingData, filter);`
- Creare il classificatore LibLinear

    1. SVMType 0 sotto corrisponde alla regressione logistica L2-regolarizzata
    2. Imposta `setProbabilityEstimates(true)` per stampare le probabilità di output

```
        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);
```

- Salva modello

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

- Creare istanze di test dal file `.arff`

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

- Carica classificatore

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

- **Utilizzare lo stesso filtro StringToWordVector come sopra o crearne uno nuovo per testData, ma ricordarsi di utilizzare trainingData per questo comando:** **`filter.setInputFormat(trainingData);`** *Ciò renderà compatibili le istanze di addestramento e test. In alternativa è possibile utilizzare* *`InputMappedClassifier`*
- Applicare il filtro a testingData: `testingData = Filter.useFilter(testingData, filter);`
- Classificare!

    1. Ottenere il valore della classe per ogni istanza nel set di test

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

```
}
```

`res` è un doppio valore che corrisponde alla classe nominale definita nel file `.arff`. Per ottenere la classe nominale utilizzare: `testintData.classAttribute().value((int)res)`

---

2. Ottenere la distribuzione di probabilità per ogni istanza

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

`dist` è un array doppio che contiene le probabilità per ogni classe definita nel file `.arff`

Nota. Il classificatore dovrebbe supportare le distribuzioni di probabilità e abilitarle con: `myClassifier.setProbabilityEstimates(true);`

Leggi Classificazione del testo online: https://riptutorial.com/it/weka/topic/7753/classificazione-del-testo

Capitolo 3: Come usare CPython Scripting in Weka?

Osservazioni

Come installare CPython in Weka?

Installa wekaPython

1. vai agli `tools` , apri il `package manager`
2. cerca `wekaPython` , seleziona e fai clic per installare

Installa le librerie Python

1. installa anaconda o conda
2. installa quattro pacchetti: numpy, pandas, matplotlib, scikit-learn
3. per la documentazione di installazione completa vedi conda

Examples

Ciao World Example per CPython di Weka

> Vai a `Explorer` , `iris.arff` dati `iris.arff` , quindi vai a `CPython Scripting` , copia e incolla le seguenti righe di codice in `Python Scripts` :

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

> Per vedere l'output, vai su `Python Variables` , seleziona `hi` , ad esempio, e fai clic su `Get text`

Leggi Come usare CPython Scripting in Weka? online:
https://riptutorial.com/it/weka/topic/7921/come-usare-cpython-scripting-in-weka-

Capitolo 4: Come usare R in Weka

Osservazioni

Perché usare R in Weka?

1. R è un potente strumento per la pre-elaborazione dei dati
2. R ha un numero enorme di librerie e continua a crescere
3. R in Weka, può facilmente ottenere dati, elaborarli e passare a Weka senza problemi

---

Come impostare R in Weka

Per utente Mac

1. sostituire la vecchia info.Plist con quella nuova fornita da Mark Hall
2. scarica e installa R
3. installa `rJava` all'interno di R con

   install.packages ('Rjava')
4. installa `Rplugin` con `Weka Package Manager`
5. vai alla cartella `weka 3-8-0` (se è la versione che stai usando), e apri il suo terminale, e
6. eseguire le seguenti 2 righe di codici (grazie a Michael Hall)

   export R_HOME = / Library / Frameworks / R.framework / Resources
   java -Xss10M -Xmx4096M -cp.: weka.jar weka.gui.GUIChooser
7. per semplificarti la vita, all'interno di una directory in cui vuoi lavorare con weka, salva il codice sopra in un file chiamato `weka_r.sh`
8. rendilo eseguibile, all'interno del terminale di questa directory, esegui il codice qui sotto:

   chmod a + x weka_r.sh
9. incollare `weka.jar` da weka 3-8-0 nella directory ed eseguire il seguente codice:

   ./weka_r.sh

Ora sei pronto per partire. La prossima volta, devi solo andare al terminale della directory ed eseguire `./weka_r.sh` per avviare R con Weka.

Come ricevere dati da Weka?

aprire Weka dal terminal :
vai alla directory di `Weka 3-8-0` , apri il suo terminale, esegui il seguente codice:

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

dati tramite Weka Explorer :

1. pannello di `preprocess` , fare clic su `open file` , scegliere un file di dati dalla `weka data folder` ;
2. andare al pannello della `R console` , digitare gli script `R console box` all'interno della `R console box`

dati tramite Weka KnowledgeFlow :

1. Pannello dei `Data mining processes` , fare clic su `ArffLoader DataSources` per scegliere `ArffLoader` ad esempio, fare clic su di esso su tela;
2. fare doppio clic su `ArffLoader` per caricare un file di dati
3. Pannello di `Scripting` , fare clic su `RscriptExecutor` su tela
4. `option` + fare clic su `ArffLoader` , selezionare `dataset` , quindi fare clic su `RScript Executor` per collegarli
5. fare doppio clic su `RScript Executor` per digitare lo script R o
6. fai clic su `Settings` e seleziona `R Scripting` per utilizzare la console R con i dati di weka

---

Riproduzione di codici R

1. caricare `iris.arff` con Explorer o KnowledgeFlow;
2. prova `Plotting inside R Console` dell'esempio di `Plotting inside R Console` sopra

Examples

Tracciare all'interno di R Console

I seguenti codici possono essere trovati dal corso Weka

Dato che `iris.arff` è caricato in weka, nella `R console` Weka Explorer o in `R Scripting` Weka KnowledgeFlow, puoi giocare con i seguenti codici per realizzare splendidi grafici:

```
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)
```

Leggi Come usare R in Weka online: https://riptutorial.com/it/weka/topic/7916/come-usare-r-in-weka

Capitolo 5: Errori facilmente commessi quando si utilizza KnowledgeFlow

introduzione

Weka KnowledgeFlow (KF) è una grande interfaccia da usare. Tuttavia, il manuale di Weka non copre ogni piccolo dettaglio dell'uso di KF. Qui sarebbe un posto per raccogliere quei piccoli trucchi o dettagli che ho imparato da quegli errori che ho fatto o farò col passare del tempo. Mille grazie alle persone di Wekalist (in particolare Mark Hall, Eibe Frank) per aver creato un ambiente di apprendimento meraviglioso per Weka!

Osservazioni

TrainingSetMaker e TestSetMaker

1. un `ClassAssigner` deve essere collegato tra `ArffLoader` e `TrainingSetMaker` o `TestSetMaker` .

ArffSaver

2. Per salvare correttamente il set di dati nel file arff, è più sicuro impostare `relationNameForFilename` su False all'interno della configurazione di `ArffSaver` .

Come utilizzare TimeSeriesForecasting in KnowledgeFlow?

1. Apri knowledgeFlow, carica il set di dati con ArffLoader
2. vai alle impostazioni, controlla la prospettiva di previsione delle serie temporali, fai clic con il pulsante destro del mouse su ArffLoader per inviare a tutte le prospettive
3. vai alla prospettiva di previsione delle serie temporali per impostare un modello
4. eseguire il modello e copiare il modello negli appunti
5. ctrl + v, quindi fare clic per incollare il modello sul canvas del processo di data mining
6. salva la previsione insieme ai dati originali con ArffSaver

---

Examples

Come aprire il file di KnowledgeFlow direttamente dal terminale

1. aggiungi la seguente funzione in `.bash_profile` , salva e esci

function wekaflstart () {

export R_HOME = / Library / Frameworks / R.framework / Resources
java -Xss10M -Xmx4096M -cp: weka.jar weka.gui.knowledgeflow.KnowledgeFlow "$ 1"
}

2. all'interno di una directory con un file `weka.jar` , apri il suo terminale, esegui `wekastart "path to a knowledgeflow file"`

Leggi Errori facilmente commessi quando si utilizza KnowledgeFlow online: https://riptutorial.com/it/weka/topic/8053/errori-facilmente-commessi-quando-si-utilizza-knowledgeflow

Capitolo 6: Iniziare con Jython in Weka

introduzione

Perché dovremmo usare Jython dentro Weka? 1. Se sei insoddisfatto di ciò che Explorer, Experimenter, KnowledgeFlow, simpleCLI ti permettono di fare, e stai cercando qualcosa per liberare il maggior potere di weka; 2. Con Jython, possiamo accedere a tutte le funzionalità fornite da Weka API, proprio all'interno di Weka; 3. La sua sintassi è simile a Python, che è considerato un linguaggio di scripting per principianti;

Osservazioni

Come configurare Jython in weka

1. installa la libreria `Jython` e `JFreeChart` dal gestore pacchetti di Weka;
2. vai al terminale della directory home, inserisci `nano .bash_profile`
3. all'interno di `.bash_profile` , aggiungi una riga di codice come sotto

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

4. salva ed esci
5. all'interno del terminale eseguire `source .bash_profile`

Quindi, riavvia Weka, vai agli `tools` e fai clic su `Jython console` e puoi provare questi esempi sopra

Examples

Carica e filtra i dati

```
# 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)
```

Costruisci un classificatore

```
# 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)
```

Classificatore di convalida incrociata

```
# 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) ==="))
```

Fare una previsione

```
# 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)
```

Bolla di errore del classificatore di convalida incrociata

```
# 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)
```

Visualizza grafico

```
# 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()
```

Leggi Iniziare con Jython in Weka online: https://riptutorial.com/it/weka/topic/8046/iniziare-con-jython-in-weka

Capitolo 7: Istanze di caricamento

Examples

File ARFF

I file ARFF (Attribute-Relation File Format) sono il formato più comune per i dati utilizzati in Weka. Ogni file ARFF deve avere un'intestazione che descrive come dovrebbe essere ogni istanza di dati. Gli attributi che possono essere utilizzati sono i seguenti:

- Numerico

Numeri reali o interi.

- Nominale

Gli attributi nominali devono fornire un insieme di valori possibili. Per esempio:

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

- Stringa

Consente valori di stringa arbitrari. Solitamente elaborato in seguito utilizzando il filtro `StringToWordVector` .

- Data

Consente di specificare le date. Come con `SimpleDateFormat` di Java, anche questa data può essere formattata; per impostazione predefinita il formato ISO-8601.

Un esempio di intestazione può essere visto come segue:

```
@RELATION iris

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

Seguendo l'intestazione ogni istanza deve essere elencata con il numero corretto di istanze; se un valore di attributi per un'istanza non è noto a `?` può essere usato invece. Quanto segue mostra un esempio dell'insieme di istanze in un file ARFF:

```
@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
```

Caricamento dei file ARFF

A seconda della versione di Weka utilizzata, è necessario utilizzare diversi metodi per caricare i file ARFF.

Weka <3.5.5

Il seguente codice di esempio mostra come caricare un file ARFF:

```
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);
```

L'indice di classe mostra quale attributo deve essere usato per la classificazione. Nella maggior parte dei file ARFF questo è l'ultimo attributo, motivo per cui è impostato su `data.numAttributes() - 1` . Se stai usando una funzione Weka, come `buildClassifier` , devi impostare l'indice della classe.

Weka> = 3.5.5

Nell'ultima versione di Weka è molto facile caricare un file ARFF. Questo metodo può anche caricare file CSV e qualsiasi altro file che Weka possa capire.

```
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);
}
```

Caricamento dal database

Molti database possono essere utilizzati in Weka. In primo luogo, il file DatabaseUtils.props deve essere modificato per corrispondere al tuo database; in particolare è necessario fornire nome, posizione, porta e driver corretto del database.

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

Quindi il database può essere caricato utilizzando un semplice codice.

```
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();
```

Alcune note sul caricamento da un database:

- Assicurarsi che il driver JDBC corretto si trovi nel classpath.
- Se si utilizza Microsoft Access, è possibile utilizzare il driver JDBC-ODBC fornito con JDK.
- Il metodo `InstanceQuery` converte VARCHAR in attributi nominali e TEXT in attributi di stringa. Un filtro, come `NominalToString` o `StringToNormal` , può convertire gli attributi nel loro tipo corretto.

Leggi Istanze di caricamento online: https://riptutorial.com/it/weka/topic/5928/istanze-di-caricamento

Capitolo 8: Semplice confronto delle interfacce Weka

introduzione

Weka ha molte interfacce, Explorer, KnowledgeFlow, Experimenter, SimpleCLI, Workbench. Tutti condividono per lo più possono svolgere gli stessi compiti, con attenzione e flessibilità diverse. Qui, esploreremo i loro diversi focus e flessibilità.

Osservazioni

Esploratore

pro:

1. fai tutto velocemente
2. fornire una visione rapida e completa della struttura dei dati

cos: non può salvare il processo;

Sperimentatore

pro:

1. confrontare più modelli contemporaneamente, ad esempio, eseguire 3 classificatori diversi su 5 set di dati tutti insieme e vedere il risultato comparato in un unico punto;
2. l'esperimento può essere salvato

KnowledgeFlow

pro:

1. fare quasi tutte le cose che Explorer può fare
2. può salvare il processo

cos:

1. KF non può fare il lavoro di Experimenter, in quanto non supporta i loop, ma ADAMS può aiutarti;
2. KF non può accedere a funzionalità di basso livello all'interno dell'API di Weka;

simpleCLI

pro: esegui attività simili a quelle di Explorer utilizzando la riga di comando

cos: non può accedere a tutte le funzionalità dell'API di Weka, lo scripting Jython o Groovy è raccomandato per questa attività.

banco di lavoro

pro: riunisce tutte le altre interfacce in un unico punto

Examples

esempi SimpleCLI e Jython

simpleCLI

vai a simpleCLI, inserisci il seguente codice

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

Esempio di Jython

codici della lezione del corso Advanced Weka MOOC 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)
```

Leggi Semplice confronto delle interfacce Weka online:
https://riptutorial.com/it/weka/topic/8042/semplice-confronto-delle-interfacce-weka

Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare con weka	Community, Gal Dreiman
2	Classificazione del testo	xro7
3	Come usare CPython Scripting in Weka?	Daniel
4	Come usare R in Weka	Daniel
5	Errori facilmente commessi quando si utilizza KnowledgeFlow	Daniel
6	Iniziare con Jython in Weka	Daniel
7	Istanze di caricamento	SJB
8	Semplice confronto delle interfacce Weka	Daniel