Data Mining Tutorial

Session 2: Tools, Loading and Visualizing

Erich Schubert, Eirini Ntoutsi

Ludwig-Maximilians-Universität München

2012-05-10 — KDD class tutorial

The Iris data set

We will use a simple data set, available from `http://aima.cs.berkeley.edu/data/iris.csv`

Four measurements:
sepal length, sepal width, petal length, petal width

Three species:
Iris Setosa, Iris Versicolour and Iris Virginica.

This is a classic example data set for *classification*, as it is linearly separable.

Weka

WEKA
The University
of Waikato

Open-source Java application, available at:
`http://www.cs.waikato.ac.nz/ml/weka/`
Popular for *classification* and *prediction*.

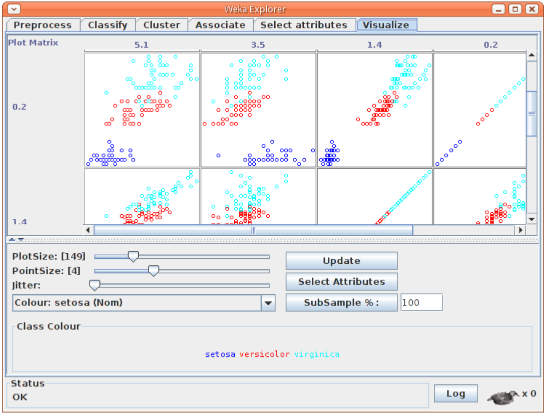
Debian and Ubuntu: package `weka`.
Installed in the CIP pool.

Run with `weka` or manually: `java -jar weka.jar`.
You may need to increase the memory, e.g. `weka -m 2g`

Uses a lot of memory, and sometime is rather slow.

Loading into Weka

Open the “Explorer”, “Open file”. Check “Invoke options dialog” to set “noHeaderRowPresent”. Go to “visualize” to see something like this:



PlotSize: [149]

PointSize: [4]

Jitter:

Colour: setosa (Nom)

SubSample % : 100

Class Colour

setosa versicolor virginica

Status

OK

Normalizing data in Weka

Weka has a wide choice of filters, including normalization.

“unsupervised.attribute.Normalize” normalizes to $[0 \ldots 1]$

“unsupervised.attribute.Standardize” standardizes to have mean $\mu = 0$ and variance $\sigma^2 = 1$.

“Attribute” filters work on single attributes.
“Instance” filters work on instances (e.g. shuffle)

ELKI

Environment for
Deve**L**oping
KDD-Applications
Supported by **I**ndex-Structures

Our own open source Java framework:
`http://elki.dbs.ifi.lmu.de/`
Focus on *clustering* and *outlier detection* along with *index structures* for acceleration.

Debian and Ubuntu: package `elki`.
Installed in the CIP pool.

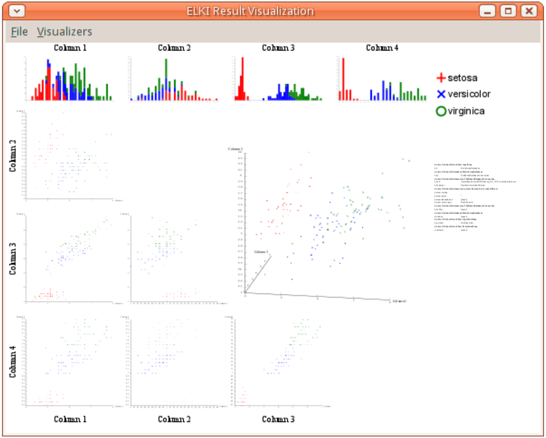
Run with `elki` or manually: `java -jar elki.jar`.

MiniGUI is mostly a command line builder (work in progress).

Algorithms are fast, but visualization (currently) rather slow because of SVG library (which is nice for print output).

Loading into ELKI

Set the parameter “-dbc.in” to the input file and as “-algorithm” choose “NullAlgorithm”. Run it to get this:



Normalizing data in ELKI

ELKI filters can be set via `-dbc.filter`.

`normalization.AttributeWiseMinMaxNormalization` normalizes each attribute to $[0 \ldots 1]$

`normalization.AttributeWiseVarianceNormalization` standardizes each attribute to mean $\mu = 0$, variance $\sigma^2 = 1$.

Many filters will perform conversions on the data that you might need, for example turn label columns into class columns.

SciPy, NumPy and MatPlotLib

Stack of Python libraries consisting of NumPy (lowlevel math), MatPlotLib (visualization) and SciPy (highlevel math, statistics)
`http://scipy.org/`

Not just data mining, any kind of “science”.
Quite fast for matrix operations due to use of BLAS, otherwise slow because interpreted.

Easy to write python code, various extensions.

Debian and Ubuntu: packages `python-scipy` and `python-matplotlib`. Installed in the CIP pool.

Run with `python` or `ipython` interactively (not a GUI).

Loading into SciPy

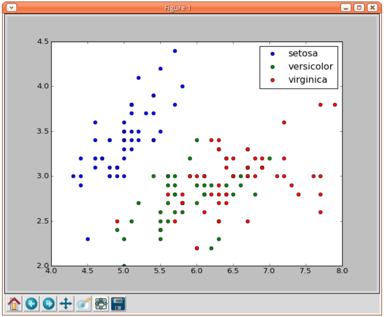
A quick Python script:

```
import numpy as np, pylab as p

# Load CSV with mixed data types
iris = np.genfromtxt("data/iris.csv",
          delimiter=",", dtype=None)
# Get fields f0, f1 and f4:
x, y = iris["f0"], iris["f1"]
species = iris["f4"]

# Plot each species (for colors)
for s in np.unique(species):
  cond = (species == s) # Filter
  p.plot(x[cond], y[cond], label=s,
    linestyle="none", marker="o")

p.legend(numpoints=1)
p.show()
```



Yes, that is the complete program. Try it interactively!

Normalizing data in SciPy

The NumPy way of doing things:
Normalization to $[0 \dots 1]$:

```
y = (y - y.min()) / (y.max() - y.min())
```

Standardize: `ddof=1`: use sample standard deviation.

```
x = (x - x.mean()) / x.std(ddof=1)
```

SciPy:
Standardize (also known as z-score):

```
y = scipy.stats.zscore(y)
```

Fast when you can write them as matrix operations.

GNU R

Open-source mathematics and statistics software, with hundrets of extension packages.
`http://r-project.org/`

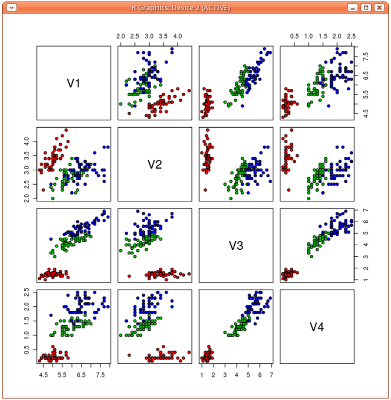
Launch: `R`, then type `library(Rcmdr)` for a GUI.
There should be a menu entry at the CIP pool!

Very fast on math operations such as matrix multiplication due to the use of BLAS libraries. Essentially, it is an programming language on its own. Many modules written however are written in C for performance.

Huge collection of libraries, including a lot of data mining.

Loading into GNU R

```
iris <- read.csv("data/iris.csv", header=FALSE)
cols <- c("red","green3","blue")[unclass(iris$V5)]
plot(iris[1:4], bg=cols, pch=21)
```



Normalizing data in GNU R

The GNU R way of doing things:
Normalization to $[0 \ldots 1]$:

```
y = (y - y.min()) / (y.max() - y.min())
```

Standardization:

```
iris$V1 <-(iris$V1-mean(iris$V1)) / sd(iris$V1)
```

Explicit, but a one-liner.
Benefit of a full scripting language: can express these things inline, instead of reyling on a specialized class (Weka, ELKI) to do the same.

Which tool to choose?

Many factors play a role:

- Has it the functions you need
  Weka: classification, ELKI: clustering and outliers, NumPy/R: fast math
- Do you know the language
  Weka/ELKI: Java, SciPy: Python, R: R
- Prototyping or for polished code
  Python/R: prototyping, Weka/ELKI: refined code
- Personal preference
  I sketch in Python, implement thoroughly in ELKI