Data Mining Tutorial

Software und Visualisierungen

Erich Schubert, Dr. Arthur Zimek

Ludwig-Maximilians-Universität München

2013-0X-XX — KDD Übung

Iris datensatz

Ein recht einfacher Datensatz, online unter:
`http://aima.cs.berkeley.edu/data/iris.csv`

Vier Messwerte (Dimensionen):
sepal length, sepal width, petal length, petal width

Drei Spezies Schwertlilien (Klassen):
Iris Setosa, Iris Versicolour, Iris Virginica.

Dies ist ein klassischer Beispieldatensatz aus dem Bereich *Klassifikation*, insbesondere für lineare Separierbarkeit.

Weka

Open-Source Java-Anwendung, online unter:
`http://www.cs.waikato.ac.nz/ml/weka/`
Beliebt insbesondere für *Klassifikation*.

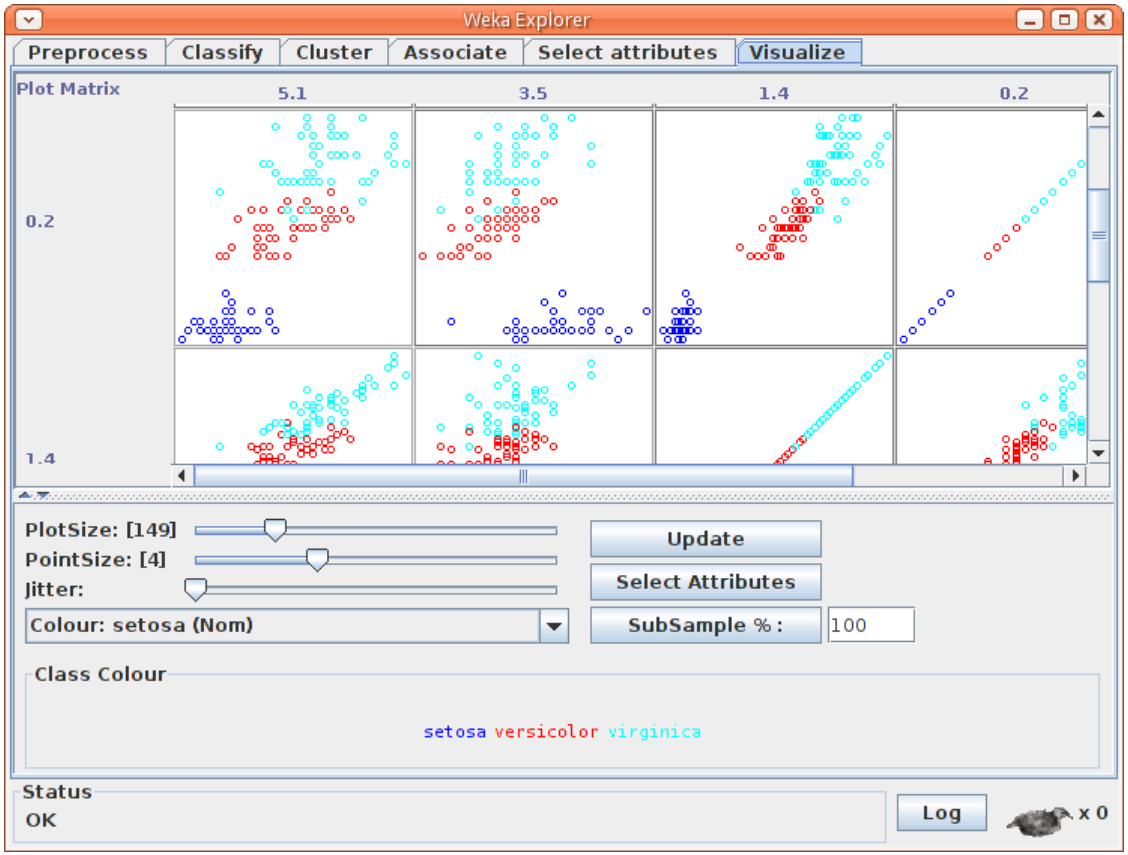
Debian und Ubuntu: Paket `weka`.
Am CIP Pool installiert.

Aufruf per `weka` oder manuell: `java -jar weka.jar`.
Ggf. Weka mehr Speicher zuweisen: `weka -m 2g`

Braucht recht viel Speicher und ist manchmal ziemlich langsam.

Einlesen von Daten in Weka

Im “Explorer”, “Open file”. Mit Hilfe von “Invoke options dialog” kann man den Parameter “noHeaderRowPresent” setzen. Dann per “visualize” visualisieren:



Normalisieren von Daten in Weka

Weka hat zahlreiche Filter, auch zur Normalisierung.

"unsupervised.attribute.Normalize" normalisiert zu $[0 \dots 1]$

"unsupervised.attribute.Standardize" standardisiert so dass der Mittelwert $\mu = 0$ und die Varianz $\sigma^2 = 1$ werden.

"Attribute" Filter operieren auf einzelnen Attributen.
"Instance" Filter arbeiten auf Instanzen (z.B. shuffle)

ELKI

Environment for DeveLoping KDD-Applications Supported by Index-Structures

LMU/DBS Open-Source Java Projekt:
`http://elki.dbs.ifi.lmu.de/`
Schwerpunkt auf *Clustering* und *Outlier Detection* mit Unterstützung für *Index-Strukturen* zur Beschleunigung.

Debian und Ubuntu: Paket `elki`.
Im CIP-Pool installiert.

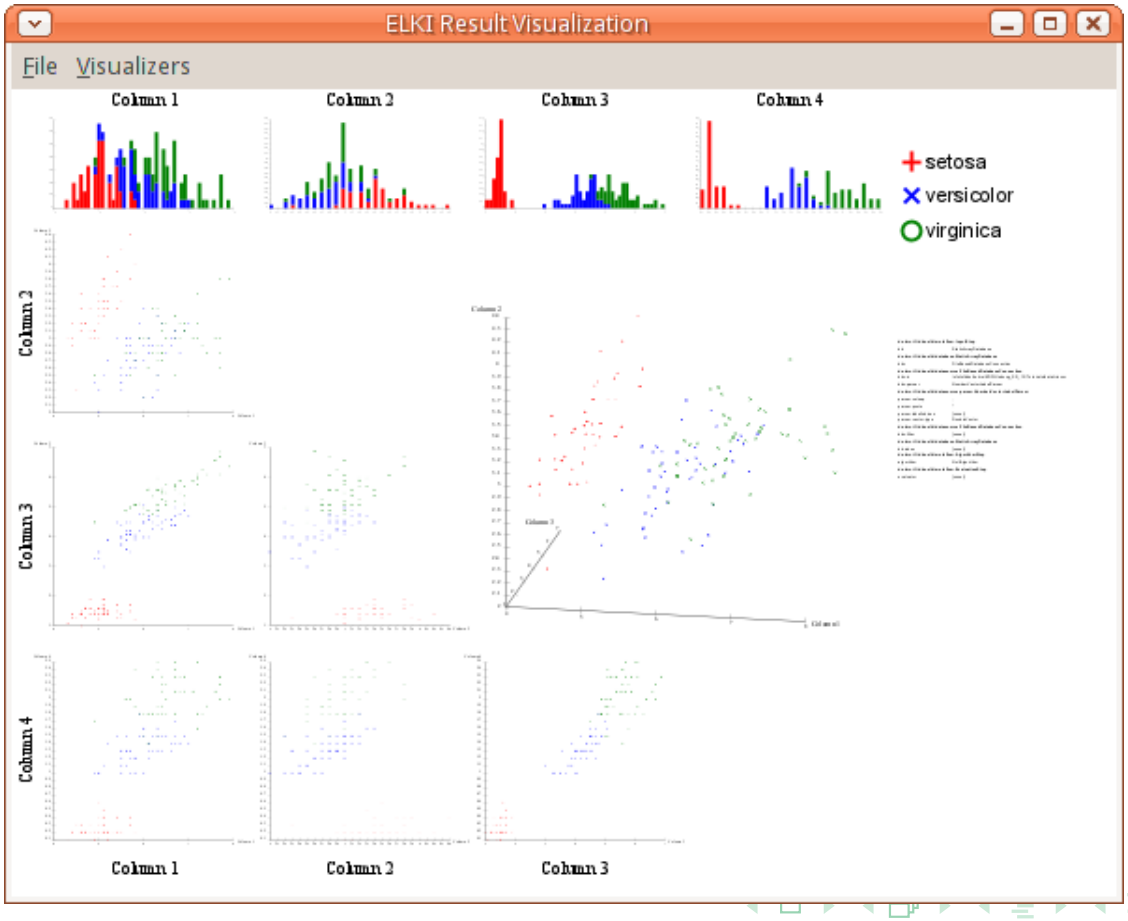
Aufruf als `elki` oder manuell: `java -jar elki.jar`.

MiniGUI ist eigentlich ein Tool, um eine Kommandozeile zu konstruieren!

Die Algorithmen sind recht schnell, aber die Visualisierung ist etwas langsam durch die verwendete SVG-Bibliothek (die aber hervorragenden export erlaubt).

Laden von Daten in ELKI

Der Parameter `-dbc.in` ist die Eingabedatei, und mit `-algorithm NullAlgorithm` bekommt man (nur) eine Visualisierung:



Normalisieren von Daten in ELKI

Filter in ELKI setzt man über den Parameter `-dbc.filter`.

`normalization.AttributeWiseMinMaxNormalization` normalisiert jedes Attribut zu $[0 \dots 1]$

`normalization.AttributeWiseVarianceNormalization` standardisiert jedes Attribut zum Mittelwert $\mu = 0$ und Varianz $\sigma^2 = 1$.

Viele Filter konvertieren Datentypen – beispielsweise eine Text-Spalte in ein Klassenlabel.

SciPy, NumPy und MatPlotLib

Sammlung von aufeinander abgestimmten Python-Bibliotheken, insbesondere *NumPy* (schnelle Mathematik auf Arrays), *MatPlotLib* (Visualisierung) und *SciPy* (Komplexere Mathematik und Statistik) `http://scipy.org/`

Nicht primär für Data Mining, aber allgemein "Wissenschaft". Ziemlich schnelle Matrixoperationen durch Verwendung der BLAS Bibliotheken intern, aber in Python geschriebener Code kann auch langsam sein...

Python Code ist sehr leicht zu schreiben, und es gibt zahlreiche Module.

Debian und Ubuntu: Pakete `python-scipy` und `python-matplotlib`. Im CIP-Pool installiert.

Aufruf per `python` oder `ipython`.

Laden von Daten nach SciPy

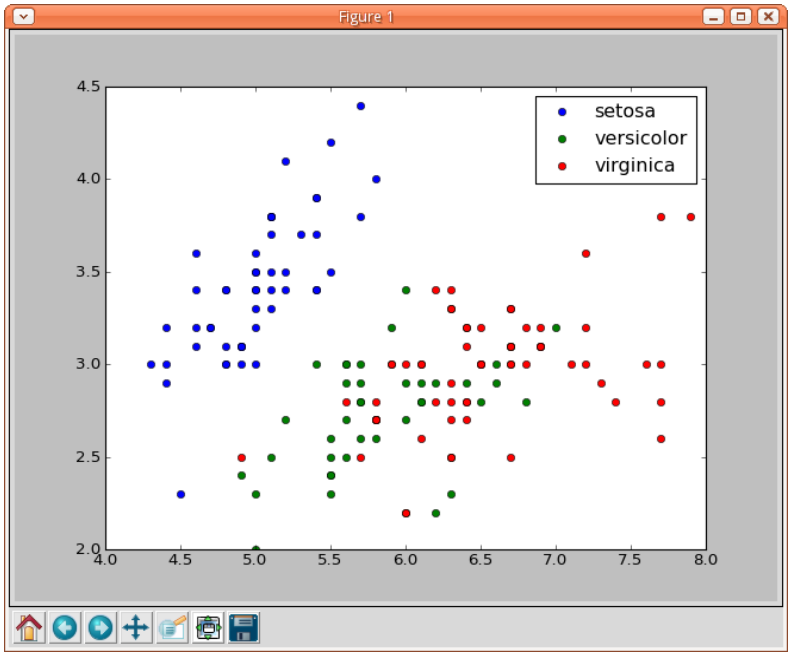
Ein einfaches Python-Skript:

```
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()
```



Ja, das ist das komplette Programm.
Einfach mal ausprobieren!

Normalisieren von Daten in SciPy

Normalisierung in NumPy nach $[0 \ldots 1]$:

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

Standardisierung: (`ddof=1` ist die Stichprobenvarianz)

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

SciPy: Standardisierung (auch bekannt als z-score):

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

Operationen sind dann schnell, wenn man sie als Matrixoperationen formuliert.

GNU R

Open-source Mathematik und Statistik-Software, mit hunderten von Erweiterungspaketen.
`http://r-project.org/`

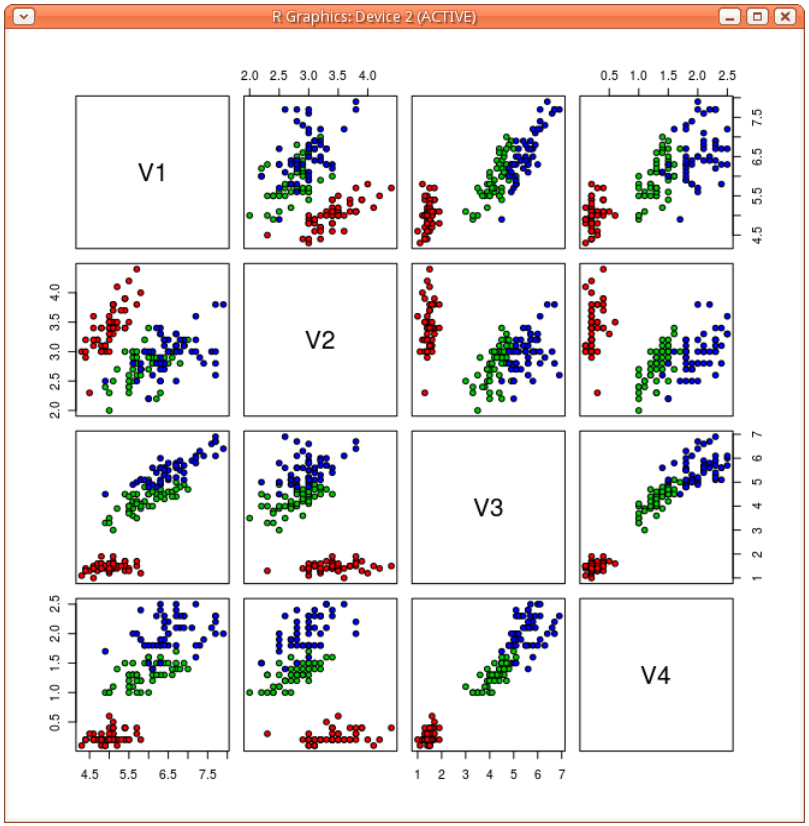
Aufruf: `R`, dann z.B. per `library(Rcmdr)` GUI starten.
Im CIP-Pool sollte es einen Menüeintrag geben!

Ziemlich schnell bei Matrix-Operationen wie Multiplikation durch verwendung der BLAS Bibliotheken. Im Prinzip ist R aber eine eigene Programmiersprache. Viele Module sind dann aber doch in C geschrieben aus Geschwindigkeitsgründen.

Enorme Sammlung von Erweiterungen, auch vieles für Data Mining, aber oft nicht ganz zusammenpassend.

Laden von Daten in 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)
```



Normalisieren von Daten in GNU R

Normalisieren nach $[0 \ldots 1]$:

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

Standardisieren:

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

Explizit, aber immerhin nur 1 Zeile.
Der Vorteil einer Skriptsprache: man kann recht viel "inline" ausdrücken, während in Weka und ELKI man eine ganze Java-Klasse implementieren muss.

Welches Programm soll man verwenden?

Zahlreiche Faktoren spielen eine Rolle:

- Hat es die Funktionen, die man braucht
  Weka: Klassifikation, ELKI: Clustering und Ausreißer,
  NumPy/R: schnelle Mathematik
- Wie vertraut ist die Programmiersprache
  Weka/ELKI: Java, SciPy: Python, R: R
- Prototyping oder Maximale Leistung
  Python/R: Prototyping, Weka/ELKI: wartbarer code
- Persönliche Vorlieben
  Ich skizziere in Python, langfristig dann in ELKI