Machine Learning

Lecture BigData Analytics

Julian M. Kunkel

julian.kunkel@googlemail.com

University of Hamburg / German Climate Computing Center (DKRZ)

04-12-2015

Outline

1 Introduction
- Data Mining
- CRIP-DM
- Terminology

2 Training

3 Classification & Regression

4 Clustering

5 Association Rule Mining

Data Mining (Knowledge Discovery) [35]

- **Data mining**: process of discovering patterns in large data sets
  - (Semi-)Automatic analysis of large data to identify interesting patterns
  - Using artificial intelligence, machine learning, statistics and databases

Tasks / Problems

- **Anomaly detection**: identify unusual data (relevant or error)
- **Association rule learning**: identify relationships between variables
- **Classification**: generalize known structures and apply them to new data
- **Clustering**: discover and classify similar data into structures and groups
- **Regression**: find a function to model data with the least error
- **Summarization**: find a compact representation of the data

Cross Industry Standard Process for Data Mining [39]

CRIP-DM is a commonly used methodology from data mining experts

Phases

- **Business understanding**: business objectives, requirements, constraints; converting the problem to a data mining problem
- **Data understanding**: collecting initial data, exploration, assessing data quality, identify interesting subsets
- **Data preparation**: creation of derived data from the raw data (data munging)
- **Modeling**: modeling techniques are selected and applied to create models, assess model quality/validation
- **Evaluation** (wrt business): check business requirements, review construction of the model(s), decide use
- **Deployment**: applying the model for knowledge extraction; creating a report, implementing repeatable data mining process

Figure: Source: Kenneth Jensen [38]

Terminology [40]

- **Feature**: measurable property of a phenomenon (explanatory variable)
- **Label**: Outcome/property of interest that should be analyzed/predicted
  - Dependent variable
  - Discrete in classification, continuous in regression
- **Online learning**: update the model constantly while it is applied
- **Offline (batch) learning**: learn from data (training phase), then apply
- **Supervised learning**: feature and label are provided in the training
- **Unsupervised learning**: no labels are provided, relevant structures must be identified by the algorithms
- **Reinforcement learning**: algorithm tries to perform a goal while interacting with the environment
  - Humans use reinforcement, (semi)-supervised and unsupervised learning

Strategy for Learning [40]

- Goal: Learn properties of the population from a sample
- Data quality is usually suboptimal
  - Errornous samples (random noise, ambivalent data)
  - **Overfitting**: a model describes noise in the sample instead of population properties
  - **Robust** algorithms reduce the chance of fitting noise
- How accurate is a specific model on the **population**?
  - Should we train a model on our data and check its accuracy on the same?
- Good practice: split data into training and validation set
  - Training set: Build/train model from this data sample
  - Validation set: Check model accuracy on this set
- Validate model accuracy via. k-fold cross validation[1]

[1] Leave-one-out cross validation builds model with all elements except one

Picking Training and Validation Sets

- k-fold cross validation
  - Prevents cases in which we partition data suboptimally
  1. Split data into k sets
  2. For all permutations: train from k-1 sets, validate with remaining set
  3. Compute average error metrics

Example with the iris data set

```
library(cvTools)
set.seed(123) # initialize random seed generator

data(iris)
# create 10 folds
f = cvFolds(nrow(iris), K=10, R=1, type="random")

# retrieve all sets
for (set in 1:10){
  validation = iris[ f$subsets[f$which == set] ,] # 135 elements
  training = iris[ f$subsets[f$which != set], ] # 15 elements

  # TODO Now build your model with training data and validate it
  # TODO Build error metrics for this repeat
}

# Output aggregated error metrics for all repeats

# Some packages perform the k-cross validation for you
```

Creating only one training set

```
# create two classes, train and validation set
mask = sample(2, nrow(iris), repl=T, prob=c(0.9,0.1))
validation = iris[mask==1, ]
training = iris[mask==2, ]
```

Classification: Supervised Learning

- Goal: Identify/predict the class of previously unknown instances

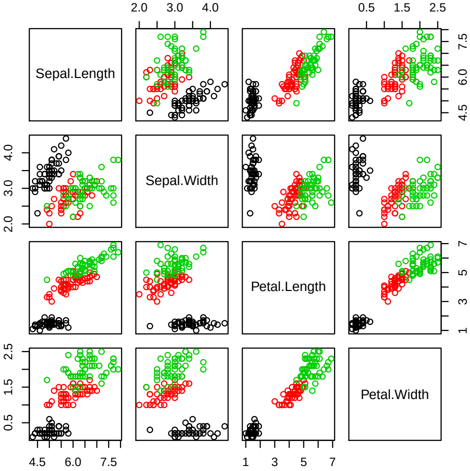


Figure: Each class (flower type) is visualized in its own color

Classification

- k-nearest neighbor a simple supervised learning algorithm
- No training algorithm needed
- Prediction: compute distance of new sample to k nearest samples
  - Majority of neighbors vote for new class
- Confusion matrix: visualizes the performance of the classification
  - Shows observation (row) and prediction class (column)

```
library(kknn)
m = kknn(Species ~ Sepal.Width + Petal.Length + Petal.Width + Sepal.Length, train=training, test=validation, k=3)

# Create a confusion matrix
table(validation$Species, m$fit)
#             setosa versicolor virginica
#  setosa          3          0         0
#  versicolor      0          7         0
#  virginica       0          1         4
```

Decision Trees

- Tree data structures, a node indicates an attribute and threshhold
  - Follow left edge if value is below threshold
  - Follow right edge if value is above
  - Leafs are decisions
  - Can separate data horizontally and vertically
- Classification trees (for classes) and regression trees for continuous vars
- Various algorithms to construct a tree
  - CART: Pick the attribute to maximize information gain of the split
- Knowledge (decision rules) can be extracted from the tree
- Tree pruning: Recursively remove unlikely leafs (reduces overfitting)

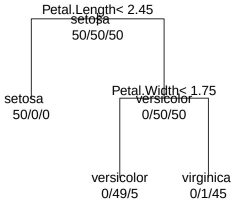


Figure: Decision tree for the iris data set with observations and labels

Decision Trees with R

- Rpart package supports regression (method="anova")
- and Classification (with 2 classes method="poisson" else "class")
- Control object defines requirements for splitting
  (e.g. observations per leaf, cost complexity (cp) factor)

```r
library(rpart)
data(iris)

# Create a classification tree based on all inputs
m = rpart(Species ~ Sepal.Width + Petal.Length + Petal.Width + Sepal.Length, data=iris, method="class",
  control = rpart.control(minsplit=5, cp = 0.05)) # require a minimum number of 5 observations

summary(m) # print details of the tree

plot(m, compress=T, uniform=T, margin=0.7) # plot the tree
text(m, use.n=T, all=T) # add text to the tree, plot all nodes not only leafs
m = prune(m, cp=0.05) # prune the tree, won't change anything here

p = predict(m, iris[150,], type="class") # predict class of data in the data frame, here one value
# virginica
p = predict(m, iris[150,], type="prob") # predict probabilities
#    setosa versicolor virginica
# 150      0 0.02173913 0.9782609

# confusion matrix
table(iris$Species, predict(m, iris, type="class"))
#             setosa versicolor virginica
#  setosa         50          0         0
#  versicolor      0         49         1
#  virginica       0          5        45
```

Regression Trees

- Regression trees predict numeric values
- They usually optimize mean-squared error
- Party package uses statistical stopping rules (no pruning needed)

```r
# Create a regression tree for Sepal.Width which optimizes mean-squared error
m = rpart( Sepal.Width ~ Species + Petal.Length + Petal.Width + Sepal.Length, data=iris, method="anova")
plot(m, compress=T, uniform=T, margin=0.7) # plot the tree
text(m, use.n=T) # add text to the tree

library(party) # package for recursive partitioning using nonparametric regression
m = ctree( Sepal.Width ~ Species + Petal.Length + Petal.Width + Sepal.Length, data=iris)
```

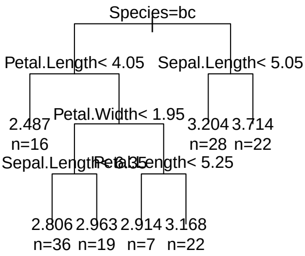


Figure: Regression tree for Sepal.Width

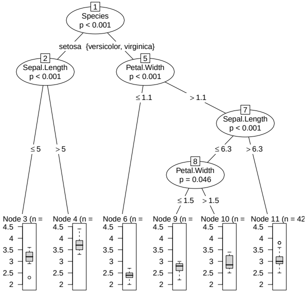


Figure: Regression tree with party

Clustering

- Partition data into “similar” observations
- Allows prediction of class for new observations
- Unsupervised learning strategy
- Clustering based on distance metrics to a center (usually euclidean)
  - Can identify regular (convex) shapes
  - k-means: k-clusters, start with a random center, iterative refinement
- Hierarchical clustering: distance based methods
  - Usually based on N^2 distance matrix
  - Agglomerative (start with individual points) or divisive
- Density based clustering uses proximity to cluster members
  - Can identify any shape
  - DBSCAN: requires the density parameter (eps)
  - OPTICS: nonparametric
- Model-based: automatic selection of the model and clusters
- Normalization of variable ranges is usually vital
  - One dimension with values in 0 to 1 is always dominated by one of 10 to 100

Density-based Clustering

```
library(fpc) # for dbscan
# For illustration purpose, we cluster the 4D feature set only using two variables

# 2D plot coloring the species
plot(iris$Sepal.Width, iris$Sepal.Length, col=iris$Species)

# Create a 2D matrix as input for dbscan
d = cbind(iris$Sepal.Width, iris$Sepal.Length)

# try to identify classes, showplot illustrates the process
p = dbscan(d, eps=0.35, showplot=1)
```

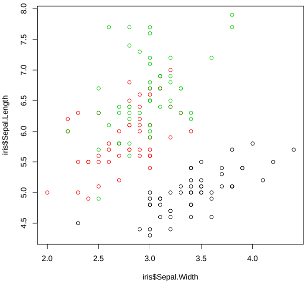


Figure: Real species (classes)

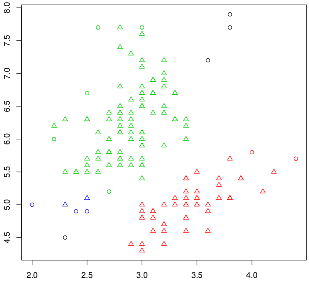


Figure: Output of dbscan

K-means Clustering

```
p = kmeans(iris[,1:4], centers=3) # cluster in 4 dimensions, exclude species
plot(iris, col=p$cluster)
```

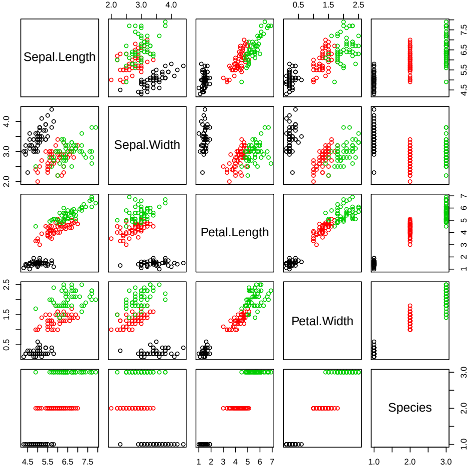


Figure: Real species

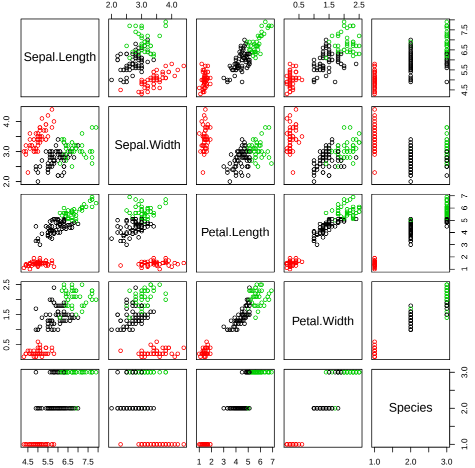


Figure: Kmeans in 4D

Model Based Clustering

- Automatic selection of model and cluster number
- Uses bayesian information criterion (BIC) and expectation-maximization

```
library(mclust)
m = Mclust(iris[,1:4]) # chooses a ellipsoidal model
```

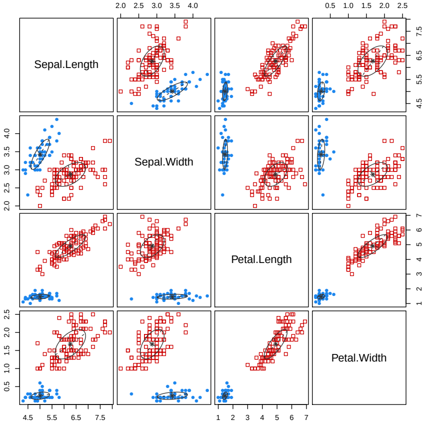


Figure: Model-based classification

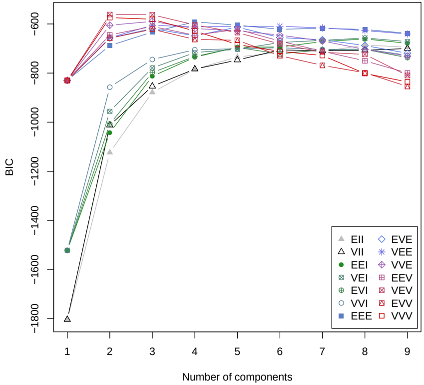
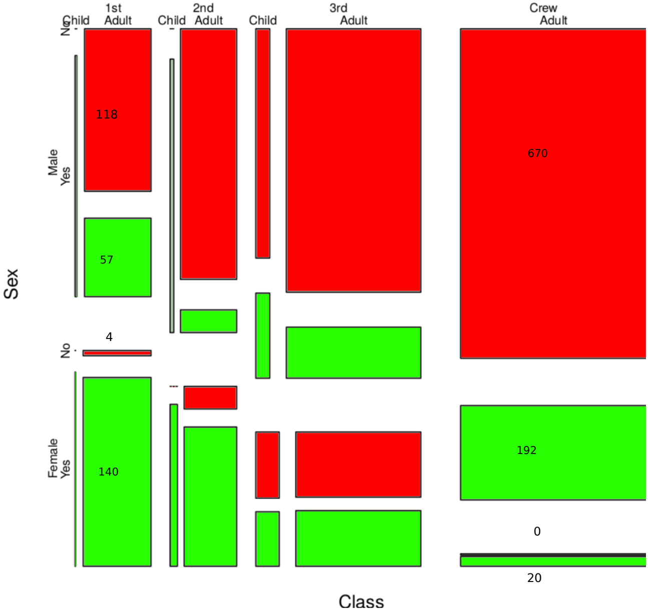


Figure: BIC

Association Rule Mining [44]

- Discover interesting relations in correlated facts and extract rules
- Identify frequent item sets "likes HR, likes BigData"
- Example association rule: "likes HR, likes BigData $\Rightarrow$ likes NTHR"
- Data are individual transactions, e.g. purchases, with items
  - Items $I = i_1, ..., i_n$
  - Transactions $T = t_1, ..., t_n$
  - Each t_i is a subset of I, e.g. items bought together in a market basket
- Several algorithms exist e.g. APRIORI, RELIM
- Relevance of rules is defined by support and confidence
  - Assume $X \Rightarrow Y$ be an association rule, X, Y are item-sets
  - supp(X): number of transactions which contains item-set X
  - conf($X \Rightarrow Y$) = $supp(X \cup Y)/supp(X)$: fraction of transactions which contain X and Y. Indicates if the rule is good

Titanic Dataset Shows if People Survived



Association Analysis with Python Using Pyming[2]

```python
from pymining import itemmining, assocrules
import csv
with open(’titanic2.csv’, ’r’) as csvfile:
  reader = csv.reader(csvfile)
  data = [ r for r in reader ]

# apply relim algorithm
r = itemmining.get_relim_input(data)
# find frequent items (more than 1000 instances)
itemsets = itemmining.relim(r, min_support=1000)
# {frozenset([’No’]): 1490, frozenset([’Male’, ’Adult’, ’No’]): 1329, frozenset([’Adult’, ’No’]): 1438, frozenset([’Adult’]): 2092, frozenset([’Male’, ’Adult’]): 1667, frozenset([’Male’, ’No’]): 1364, frozenset([’Male’]): 1731}

# mine the association rules
r = itemmining.get_relim_input(data)
itemsets = itemmining.relim(r, min_support=1)
rules = assocrules.mine_assoc_rules(itemsets, min_support=2, min_confidence=0.7)
# [(([’Adult’, ’No’]), ([’Male’]), 1329, 0.9242002781641169), (([’No’]), ([’Male’, ’Adult’]), 1329, 0.8919463087248322), ...
# identify only survival-relevant rules with two or one items/attributes
relevant = [ (p, "Yes" in c,supp,conf) for p, c, supp, conf in rules if (c == frozenset([’No’]) or c == frozenset([’Yes’])) and len(p) <= 2]
relevant.sort(key=lambda x : x[1]) # sort based on the survival
for p, c, supp, conf in relevant:
  print(("%d,%.2f: %s <= %s" % (supp, conf, c, p)).replace("frozenset",""))
#1329,0.80: False <= ([’Male’, ’Adult’])
#476,0.76: False <= ([’Adult’, ’3rd’])
#154,0.86: False <= ([’Male’, ’2nd’])
#422,0.83: False <= ([’Male’, ’3rd’])
#344,0.73: True <= ([’Female’])
#316,0.74: True <= ([’Adult’, ’Female’])
#6,1.00: True <= ([’1st’, ’Child’])
#24,1.00: True <= ([’2nd’, ’Child’])
```

[2] https://github.com/bartdag/pymining

Machine Learning with Python

- Recommended package: `scikit-learn` [3]
- Provides classification, regression, clustering, dimensionality reduction
- Supports via model selection and preprocessing

Example: Decision tree

```python
from sklearn.datasets import load_iris
from sklearn import tree
iris = load_iris()
m = tree.DecisionTreeClassifier()
m = m.fit(iris.data, iris.target)

# export the tree for graphviz
with open("iris.dot", 'w') as f:
  tree.export_graphviz(m, out_file=f)

# To plot run: dot -Tpdf iris.dot
```

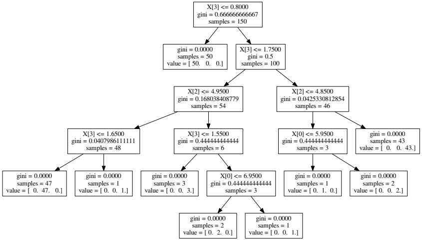


Figure: Sklearn decision tree

[3] http://scikit-learn.org/stable/

Bibliography

35 https://en.wikipedia.org/wiki/Data_mining

38 https://en.wikipedia.org/wiki/Cross_Industry_Standard_Process_for_Data_Mining

39 ftp://ftp.software.ibm.com/software/analytics/spss/support/Modeler/Documentation/14/UserManual/CRISP-DM.pdf

40 https://en.wikipedia.org/wiki/Machine_learning

41 http://www.rdatamining.com/docs/introduction-to-data-mining-with-r

42 CRAN Task View: Machine Learning & Statistical Learning https://cran.r-project.org/web/views/MachineLearning.html

43 http://www.rdatamining.com/examples/text-mining

44 https://en.wikipedia.org/wiki/Association_rule_learning