Predictive models:
building and validation

Paolo Marcatili
CBS, DTU
paolo.marcatili@gmail.com
Schedule

9.00 to 9.20: Recap
9.20 to 9.50: Clustering
9.50 to 10.00: Pause
10.00 to 11.00: LM and Classifiers
11.00 to 11.10: Pause
11.10 to 12.00: Validation
12.00 to 13.00: Lunch Break

13.00 to 16.00: Exercise
Lesson goals

- See what’s out there
- Be able to understand the main pitfalls
- Train your first predictor
Unsupervised Learning

If we have no idea of actual data classification, we can try to guess.
Clustering

Put together similar objects to define classes
Clustering

Put together similar objects to define classes
Put together similar objects to define classes

How?

K-means
Hierarchical top-down
Hierarchical down-up
Fuzzy
Clustering

Put together similar objects to define classes

How? Which metric?

Euclidean
Correlation
Spearman Rank
Manhattan
Clustering

Put together similar objects to define classes

How? Which metric? Which “shape”? Compact Concave Outliers Inner radius cluster separation
Hierarchical Clustering

- We start with every data point in a separate cluster
- We keep merging the most similar pairs of data points/clusters until we have one big cluster left
- This is called a bottom-up or agglomerative method
Hierarchical Clustering

- We start with every data point in a separate cluster
- We keep merging the most similar pairs of data points/clusters until we have one big cluster left
- This is called a bottom-up or agglomerative method
Hierarchical Clustering

- We start with every data point in a separate cluster
- We keep merging the most similar pairs of data points/clusters until we have one big cluster left
- This is called a bottom-up or agglomerative method
Hierarchical Clustering

• We start with every data point in a separate cluster
• We keep merging the most similar pairs of data points/clusters until we have one big cluster left
• This is called a bottom-up or agglomerative method
Hierarchical Clustering

- We start with every data point in a separate cluster
- We keep merging the most similar pairs of data points/clusters until we have one big cluster left
- This is called a bottom-up or agglomerative method
Hierarchical Clustering

• We start with every data point in a separate cluster
• We keep merging the most similar pairs of data points/clusters until we have one big cluster left
• This is called a bottom-up or agglomerative method
Hierarchical Clustering

- We start with every data point in a separate cluster
- We keep merging the most similar pairs of data points/clusters until we have one big cluster left
- This is called a bottom-up or agglomerative method
Hierarchical Clustering

• We start with every data point in a separate cluster
• We keep merging the most similar pairs of data points/clusters until we have one big cluster left
• This is called a bottom-up or agglomerative method
Hierarchical Clustering in R

```r
clust1 = hclust(dist(data), method = "average")
cut1 = cutree(clust1, k = 3)
plot(clust1)
```

dist(x) calculates the Euclidean distances between each pair of elements in x

cutree(x, k = 3) cuts the hierarchical tree at the level of k = 3 (in this case) clusters
K-means

- Start with K random centers
- Assign each sample to the closest center
- Recompute centers (samples average)
- Repeat until converged
K-means

- Start with K random centers
- Assign each sample to the closest center
- Recompute centers (samples average)
- Repeat until converged
K-means

• Start with K random centers

• Assign each sample to the closest center

• Recompute centers (samples average)

• Repeat until converged
K-means

- Start with K random centers
- Assign each sample to the closest center
- Recompute centers (samples average)
- Repeat until converged
K-means

- Start with K random centers

- Assign each sample to the closest center

- Recompute centers (samples average)

- Repeat until converged
K-means

- Start with K random centers
- Assign each sample to the closest center
- Recompute centers (samples average)
- Repeat until converged
K-means

- Start with K random centers
- Assign each sample to the closest center
- Recompute centers (samples average)
- Repeat until converged
K-means

- Start with K random centers

- Assign each sample to the closest center

- Recompute centers (samples average)

- Repeat until converged
K-means

- Start with K random centers
- Assign each sample to the closest center
- Recompute centers (samples average)
- Repeat until converged
K-means

- Start with K random centers
- Assign each sample to the closest center
- Recompute centers (samples average)
- Repeat until converged
K-means

• Start with K random centers

• Assign each sample to the closest center

• Recompute centers (samples average)

• Repeat until converged
K-means

- Start with $K$ random centers
- Assign each sample to the closest center
- Recompute centers (samples average)
- Repeat until converged
K-means

- Start with K random centers
- Assign each sample to the closest center
- Recompute centers (samples average)
- Repeat until converged
K-means

• Start with K random centers

• Assign each sample to the closest center

• Recompute centers (samples average)

• Repeat until converged
K-means

• Start with K random centers

• Assign each sample to the closest center

• Recompute centers (samples average)

• Repeat until converged
K-means

- Start with K random centers
- Assign each sample to the closest center
- Recompute centers (samples average)
- Repeat until converged
K-means

• Start with K random centers

• Assign each sample to the closest center

• Recompute centers (samples average)

• Repeat until converged
K-means

- Start with K random centers
- Assign each sample to the closest center
- Recompute centers (samples average)
- Repeat until converged
K-means

• Start with K random centers

• Assign each sample to the closest center

• Recompute centers (samples average)

• Repeat until converged
K-means

- Start with K random centers
- Assign each sample to the closest center
- Recompute centers (samples average)
- Repeat until converged
K-means

• Start with K random centers

• Assign each sample to the closest center

• Recompute centers (samples average)

• Repeat until converged
K-means

- Start with K random centers
- Assign each sample to the closest center
- Recompute centers (samples average)
- Repeat until converged
K-means

- Start with K random centers
- Assign each sample to the closest center
- Recompute centers (samples average)
- Repeat until converged
K-means

- Start with K random centers
- Assign each sample to the closest center
- Recompute centers (samples average)
- Repeat until converged
Kmeans in R

clust1=kmeans(x, centers=3)

Compare (by visual inspection) the results of hclust using the ward and average methods and cutting with k=3,4 and 5 to the results of kmeans with centers=3, 4 and 5. Are the clusterings similar?

You can compare 2 clustering by using the cluster_similarity function:

install.packages("clusteval")
library("clusteval")
cluster_similarity(clust1, clust2,similarity = c("rand"))

Check the similarity between the clusterings. Which are more and less similar?
Clustering is useful to find patterns and groups in the data.

Clustering always return a result, but it might be not meaningful!

Try to use different methods and to compare the results.
Predictions

part I: the training

Input Variables
(e.g. peptide sequence)

Data to be “learned”
(e.g. aggregation propensity)
Predictions

part I: the training

Input Variables (e.g. peptide sequence)

Data to be “learned” (e.g. aggregation propensity)

Prediction (aggregation propensity)
Predictions

part I: the training

Input Variables (e.g. peptide sequence)

Data to be “learned”
(e.g. aggregation propensity)

if prediction ≠ training data
change something
(method, variables, project)

prediction
(aggregation propensity)
Predictions

part I: the training

Input Variables  
(e.g. peptide sequence)

Data to be “learned”  
(e.g. aggregation propensity)

prediction  
(aggregation propensity)

if prediction ≠ training data
change something  
(method, variables, project)

This is called overfitting
Predictions

part II: the validation

NEW Input Variables (e.g. peptide sequence)

Measure the accuracy by comparing prediction with validation data
Supervised Learning

Basic Idea: use data+classification of known samples
find “fingerprints” of classes in the data
Supervised Learning

Basic Idea: use data + classification of known samples
find “fingerprints” of classes in the data

Example:
use microarray data,
different condition

classes:
genres related/unrelated
to different cancer types
Least squares line fitting

- Data: \((x_1, y_1), \ldots, (x_n, y_n)\)
- Line equation: \(y_i = mx_i + b\)
- Find \((m, b)\) to minimize

\[
E = \sum_{i=1}^{n} (y_i - mx_i - b)^2
\]
Robustness to noise

- Least squares fit to the red points:
Robustness to noise

- Least squares fit with an outlier:

Problem: squared error heavily penalizes outliers
linear models in R

model=lm(Weight ~ Height + Age, data=nba)

Which variable has more weight?
What is the sum of the residuals?
Which player has the largest residual?
model2 = lm(Weight ~ Height, data=nba)
plot(nba$Height, nba$Weight)
abline(model2)

Which variable has more weight?
What is the sum of the residuals?
Which player has the largest residual?
Decision trees

Mimics the behavior of an expert
Decision trees

The idea is to use the variables to divide the sets into subsets that are more homogenous.
Decision trees

The idea is to use the variables to divide the sets into subsets that are more homogenous.

Initial set: 100 play, 80 non play

If humid: 60 play, 80 non play
If non humid: 40 play, 20 non play
Decision trees

The idea is to use the variables to divide the sets into subsets that are more homogenous.

Initial set: 100 play, 80 non play
If sunny: 30 play, 10 non play
If overcast: 30 play, 5 non play
If rainy: 40 play, 65 non play
Pros:
Easy to interpreter
Statistical analysis
Informative results

Cons:
A single variable
Not optimal
Not robust

Majority rules!
install.packages("rpart")
library(rpart)
model2=rpart(Role ~ ., data=nba[,2:5])
plot(model2,margin = .1)
text(model2)

Is the model good?
Is it meaningful?
Random Forests

Split the data in several subsets, construct a DT for each set

Each DT expresses a vote, the majority wins

Much more accurate and robust (bootstrap)
Random Forest

1) Generate bootstrap samples
Random Forest

Take out $n$ samples and replace with duplicates. Usually $n \approx 1/3$ of your samples.

1) Generate bootstrap samples
Random Forest

1) Generate bootstrap samples

2) Select the best variable among m randomly chosen to start the tree (m is user submitted, can be tuned)
Random Forest

1) Generate bootstrap samples
2) Random variable selection
3) Fit unpruned decision trees
4) Apply to testing data & combine predictions
Random Forest

1) Generate bootstrap samples

2) Random variable selection

3) Fit unpruned decision trees

4) Apply to testing data & combine predictions
install.packages("randomForest")
library(randomForest)
model3=randomForest(Role ~ ., data=nba[,2:5])
model3
varImpPlot(model3)

What is the accuracy of the model?
What are the differences with the DT?
Validation Methods
Validation Methods

Split Data in 2
Not reliable nor efficient
Validation Methods

Training set

- Validate on 1 sample

Repeat on each sample

Leave-one-out

Not efficient
Validation Methods

Train on (K-1) subsets
Validate on 1 subset
Repeat K time
Cross-fold validation
The most common

N/K samples
Caveats

Cross-fold is necessary
but *not* sufficient

Overfitting can be hidden:

- Redundancy
- Negative dataset
- Overparametrization
Assessment

**True Positive Rate:** \( \frac{TP}{TP + FN} \)
Given the disease is present, the likelihood of testing positive.

**False Positive Rate:** \( \frac{FP}{TN + FP} \)
Given the disease is not present, the likelihood of testing positive.

**Positive Predictive Value:** \( \frac{TP}{TP + FP} \)
Given a test is positive, the likelihood disease is present.
Assessment

receiver operating characteristic (ROC) is a graphical plot of the sensitivity vs. (1 - specificity) for a binary classifier system as its discrimination threshold is varied.

**True Positive Rate:** TP / (TP + FN)
Given the disease is present, the likelihood of testing positive.

**False Positive Rate:** FP / (TN + FP)
Given the disease is not present, the likelihood of testing positive.

**Positive Predictive Value:** TP / (TP + FP)
Given a test is positive, the likelihood disease is present.
ROC curve

Score to separate Positives from Negatives
High threshold -> less FP, more FN
Low threshold -> more FP, less FN
Validation in R

install.packages("ROCR")
library(ROCR)
training=sample(1:504,252)
is_forward= as.factor(nba[,2] =="F")
nba2=cbind(nba[,3:5],is_forward)
model4=randomForest(is_forward ~ ., data=nba2[training,] )
rf.pr = predict(model4,type = "prob",newdata=nba2[-training,])
rf.pred = prediction(rf.pr[,2], labels=nba2[-training,4])
#performance in terms of true and false positive rates
rf.perf = performance(rf.pred,"tpr","fpr")

#plot the curve
plot(rf.perf,main="ROC Curve for Random Forest",col=2,lwd=2)
abline(a=0,b=1,lwd=2,lty=2,col="gray")

#compute area under curve
auc <- performance(rf.pred,"auc")
Validation in R

What is the AUC?
What is the AUC calculated on the training data?

Is the accuracy on the training and testing set similar?
ML for large noisy heterogeneous data

Don’t use the same data to train and to assess. Sometimes it’s evident, sometimes it’s not

Use cross-fold validation

Data redundancy can be an issue

Don’t let your negative dataset be too negative
References


• http://see.stanford.edu/see/courseinfo.aspx?coll=348ca38a-3a6d-4052-937d-cb017338d7b1
library(ROCR) -> ROC curves and assessment

library(randomForest) -> random Forest

General idea:
train the machine,
use the predict function to perform new predictions
Exercise

load the epitope data
define the binders as the epitopes with a value <100
1) train a DT on the whole set in order to predict
   the binders from the sequence. calculate FPR and TPR
2) do the same with a 2-fold cross-validation.
3) What are the variables used in the tree?
4) train a RF on the whole set in order to predict
   the value from the sequence. calculate FPR, TPR, roc and auc
5) do the same with a 2-fold cross-validation.

Are the results comparable?
Exercise

6) What are the important variables without and with cross-validation?
7) Try to train the rf excluding residue 2. Does the performance drop?