Machine learning

focus on prediction
pattern recognition and computational learning theory in artificial intelligence
creation and use of models that are learned from data

- often used as a *black box*
- evaluation often based on testing data set

algorithms divided into: supervised learning and unsupervised learning.

Supervised learning

variables:
- input variables X_1, X_2, ..., X_p (or $\mathbf{X} \in \Re^p$), set of p features, measured on n observations
- outcome variable, output: Y (response also measured on those same n observations)
objectives: predict Y using X_1, X_2, ..., X_p

Unsupervised learning

variables:
no outcome variable
objectives: (not insterested in prediction, because no associated response variable Y)
-visualizing $X_i \in \Re^p$ -discover interesting things about the measurements on X_1, X_2, ..., X_p
-discover subgroups among the variables or among the observations more challenging!

challenge!!!: hard to evaluate how well you are doing. No way to check our work, because we have no true answer.
-more subjective, no clear objective
-often part of explorative data analysis
-difficult to evaluate results (no clear objective, difficult to evaluate based on testing set)

Methods

Supervised
-linear regression,
-generalized linear regression,
-classification,
-resampling methods

Unsupervised learning
-principal component analysis,
-clustering

Unsupervised learning: Clustering

among other methods:

- K-means clustering (divide in a specific number of groups; minimize a criterion)
- hierarchical clustering (merge two and two groups in an iteractive procedure; tree-based representation of the observations, called a *dendogram*)

Clustering

class of methods to group data

- observations inside a group are quite similar
- observations from different groups are more different

K-means clustering

we seek to partition our observations into a pre-specified number of clusters (K). These K clusters are distinct and non-overlapping. Details Let C_1, ..., C_K denote the sets containing the indices of the observations in each cluster. These sets satisfy the properties:

1. $C_1 \cup C_2 \cup ... \cup C_K = \{1,\ldots,n\}$. Each observation belongs to at least one of the K clusters.
2. $C_k \cup C_{k'} = \emptyset$ for all $k \neq k'$. The clusters are non-overlapping: no observation belongs to more than one cluster.
3. Variation inside a group lowest possible

$$\begin{aligned} &\underset{C_1,\ldots,C_K}{\text{minimize}} \quad \{\sum_{k=1}^{K} W(C_k)\} \\ &\text{where} \qquad W(C_k) = \frac{1}{|C_k|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{i,j} - x_{i',j})^2 \end{aligned} \tag{1}$$

   where $|C_k|$ denotes the number of observations in the kth cluster. The within-cluster variation for the kth cluster ($W(C_k)$) is the sum of all of the pairwise squared Euclidean distances between the observations in the kth cluster, divided by the total number of observations in the kth cluster.

K-means clustering - characteristics

- one of the most popular iterative descent clustering methods,
- when all variables are of quantitative type,
- squared Euclidean distance (straight-line distance between two points in Euclidean space):

K-means clustering - algorithm

There are almost K^n ways to partition n observations into K clusters.

Wish algorithm which minimizes $\sum_{k=1}^{K} W(C_k)$:

1. Allocate at random each observation to one group $\{1,\ldots,K\}$.
2. Iterate until no more changes

   2-1 For each of the K clusters, compute the centroid. The kth cluster centroid is the vector of the p feature means for the observations in the kth cluster.

   2-2 Assign each observation to one group (cluster) whose centroid is closest (where closest is defined using Euclidean distance).

The algorithm will decrease the value of the objective Eq. (1) at each step. The algorithm converges to a local optimum.

meteos - Distance measure

Euclidean distance does not apply in this problem.

The haversine formula determines the great-circle distance between two points on a sphere given their longitudes and latitudes.

Hierarchical Clustering

agglomerative (upside-down tree, bottom-up clustering) dendogram built starting from the leaves and combining clusters up to the trunk.

divisive

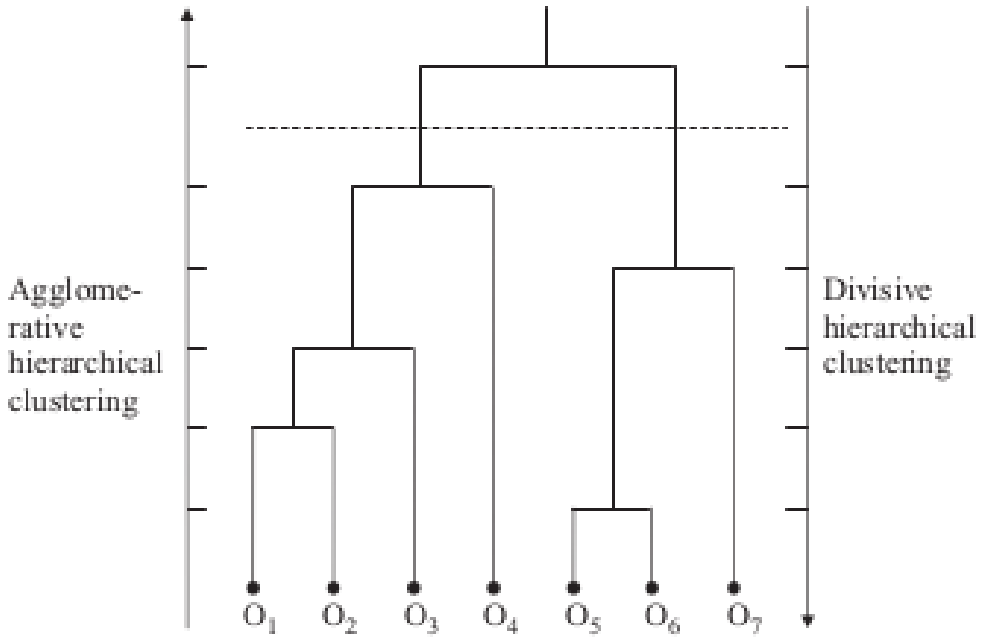

Figure 1: Hierarchiecal Clustering - DENDOGRAM (from ref. (5))

Hierarchical Clustering: Agglomerative

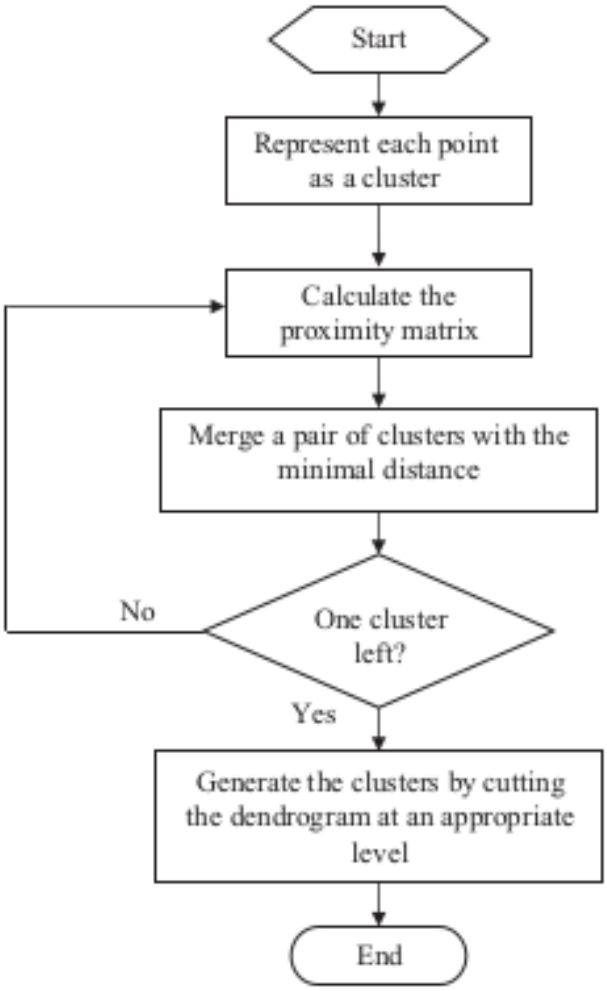

Figure 2: Hierarchiecal Clustering - Agglomerative (bottom-up, upside-down tree) algorithm (from ref. (5))

Proximity measures

Clusters are groups containing data objects that are similar to each other, while data objects in different clusters are not.

How to measure the closeness, or the distance (dissimilarity) or similarity between a pair of objects, an object and a cluster, or a pair of clusters?

A data object is described by a set of **features** or **variables**, usually represented as a multidimensional vector.

Features (variables) are lat (latitude), lon (longitude), IVT (integrated water vapor transport):

```
lat=infile.variables['lat']
lon=infile.variables['lon']
ivt=infile.variables['IVT']
```

IVT

Before performing hierarchiecal clustering, the tree size is decreased by considering the threshold on IVT of 250 kg/m/s. (`https://www.esrl.noaa.gov/psd/psd2/coastal/satres/ar_detect.html`)

```
# use a boolean condition to find where pixel values are > 250
blobs=(ivt[tt,:,:]>=250.)*1.0
```

Haversine

distance matrix computation:

```
import scipy.spatial.distance as distance
```

for distance computations (scipy.spatial.distance.cdist) computes distance between each pair of the collections of inputs.

```
dist = distance.cdist(coordsOut, coordsIn, lambda u, v: getDistanceByHaversine(u,v))
```

The Haversine distance defines the dissimilarity measure used.

Clustering linkage - Linkage metrics

How do we define the dissimilarity between two clusters if one or both of the clusters contains multiple observations?

In order to extend the concept of dissimilarity between a pair of observations to a pair of *groups of observations*, the notion of **linkage** is developed.

The merge of pair of clusters is dependent on the definition of the distance function between two clusters.

```
from scipy import ndimage
```

for multi-dimensional image processing.

```
# label connected regions that satisfy this condition
labels, nlabels = ndimage.label(blobs)
```

From the default of ndimage.label, for a 2-D input array (blobs), the default structuring element is:

$$\begin{Bmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{Bmatrix}$$

labels is XXX

nlabels is a tuple of (labeled-array, num-features).

test.py algorithm

Practical issues

- Presence of outliers?

  Clustering forces every observation into a cluster, the clusters may be heavily distorted due to outliers that do not belong to any cluster.

- Sensitive to perturbations

References

(1) Geir Storvik, STK2100 - Maskinlæring og statistiske metoder for prediksjon og klassifikasjon, 2017.

(2) T. Hastie et al., The Elements of Statistical Learning, Second Edition, Springer, 2009.

(3) Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani, An Introduction to Statistical Learning with Applications in R, Springer, 2015.

(4) Joel Grus, Data Science do Zero, Alta Books Editora, 2016.

(5) Rui. Xu, Clustering, IEEE Press Series on Computational Intelligence, 2008.

test.py

```
#!/usr/bin/env python

from netCDF4 import Dataset
import matplotlib.pyplot as plt
from scipy import ndimage
from mpl_toolkits.basemap import Basemap
import numpy as np
import scipy.spatial.distance as distance
from optparse import OptionParser
from numpy import sin,cos,arctan2,sqrt,pi # import from numpy

# earth's mean radius = 6,371km
EARTHRADIUS = 6371.0
West= [4.49,58.875,7,61.75]
South = [6.4,57.9,9.5,59.65]
North = [11.25,65.0,14.5,66.8]
W = 1
S = 2
N = 3

def Norway(lat,lon):
if lon >= West[0] and lon <= West[2] and lat >= West[1] and lat <= West[3]:
region = W
elif lon >= South[0] and lon <= South[2] and lat >= South[1] and lat <= South[3]:
region = S
elif lon >= North[0] and lon <= North[2] and lat >= North[1] and lat <= North[3]:
region = N
else:
region = -1
return region

def getDistanceByHaversine(loc1, loc2):
'''Haversine formula - give coordinates as a 2D numpy array of
(lat_denter link description hereecimal,lon_decimal) pairs'''
#
# "unpack" our numpy array, this extracts column wise arrays
lat1 = loc1[1]
lon1 = loc1[0]
lat2 = loc2[1]
lon2 = loc2[0]
#
# convert to radians ##### Completely identical
lon1 = lon1 * pi / 180.0
lon2 = lon2 * pi / 180.0
lat1 = lat1 * pi / 180.0
lat2 = lat2 * pi / 180.0
#
# haversine formula #### Same, but atan2 named arctan2 in numpy
dlon = lon2 - lon1
dlat = lat2 - lat1
a = (sin(dlat/2))**2 + cos(lat1) * cos(lat2) * (sin(dlon/2.0))**2
c = 2.0 * arctan2(sqrt(a), sqrt(1.0-a))
km = EARTHRADIUS * c
return km
```

```
def main():
usage = """usage: %prog --input=inputfile"""
parser = OptionParser(usage=usage)
parser.add_option("-i", "--input", dest="inputfile",
help="netCDF inputfile", metavar="inputfile" )

(options, args) = parser.parse_args()
if not options.inputfile:
parser.error("a netCDF input file is required!")
else:
inputfile=options.inputfile


lsmfile = Dataset('file_lsm.nc','r')
lsm=lsmfile.variables['LSM']
infile = Dataset(inputfile,'r')
lat=infile.variables['lat']
lon=infile.variables['lon']
ivt=infile.variables['IVT']

ntime = ivt[:,0,0].size

found_point=False
for tt in range(0,ntime):
# use a boolean condition to find where pixel values are > 250
blobs=(ivt[tt,:,:]>=250.)*1.0

img = ivt[tt,:,:]

# label connected regions that satisfy this condition
labels, nlabels = ndimage.label(blobs)

if tt==0:
xlon, ylat = np.meshgrid(lon[:], lat[:])

# plot

nclusters=labels.max()

for ic in range(1,nclusters):
lx=(labels==ic)
# split in two parts: potentially inside Norway and outside Norway
xylatlonIn = np.logical_and((ylat>=58.0),(xlon >= 4.49))
xylatlonInandOn = np.logical_and((lsm > 0.25), xylatlonIn)

r1 =  np.logical_and((xlon>=4.49), (ylat <58.0))
xylatlonOut = np.logical_or(r1,(xlon < 4.49))

lxIn  =  np.logical_and(xylatlonInandOn,lx)
lxOut = np.logical_and(xylatlonOut,lx)

# For points outside Norway take contour only
xd=ndimage.binary_erosion(lxOut).astype(lxOut.dtype)
x=lxOut-xd
lon_validOut = xlon[np.nonzero(x)]
```

```
lat_validOut = ylat[np.nonzero(x)]
coordsOut=[]
for i in range(0,lat_validOut.size):
coordsOut.append((lon_validOut[i],lat_validOut[i]))

# For point inside Norway take all points
lon_validIn = xlon[np.nonzero(lxIn)]
lat_validIn = ylat[np.nonzero(lxIn)]
coordsIn=[]
for i in range(0,lat_validIn.size):
coordsIn.append((lon_validIn[i],lat_validIn[i]))


if len(coordsOut)>0 and len(coordsIn)>0 :
dist = distance.cdist(coordsOut, coordsIn, lambda u, v: getDistanceByHaversine(u,v))
# index where we have the maximum
# we keep distance if they are greater than 2000 only
distVALID=(dist>2000.)*dist
distMAX=distVALID.max()
# skip if we don't have anything > 2000
while distMAX > 0:
idxList = np.where(distVALID==distMAX)
idxR=idxList[0]
idxC=idxList[1]
# number of max distances (usually two but several points may have the same max distance...)
nb_dist = len(idxR)
is_over_region=False
for dd in range(nb_dist):
# point on Norway
idx_lon1=np.where(lon==coordsIn[idxC[dd]][0])
idx_lat1=np.where(lat==coordsIn[idxC[dd]][1])
# reset to 0 as we have distMAX
distVALID[idxR[dd],idxC[dd]]=0
# The point is over Norway but it belongs to which region?
region = Norway(ylat[idx_lat1[0],idx_lon1[0]], xlon[idx_lat1[0],idx_lon1[0]])
if region > 0:
is_over_region=True
print "lat,lon: ", ylat[idx_lat1[0],idx_lon1[0]], xlon[idx_lat1[0],idx_lon1[0]]
break

if is_over_region:
print "We found it"
print "distance = ", distMAX, " Region = ", region, " time = ", tt
fig, ax = plt.subplots(1, 3, sharex=True, sharey=True, figsize=(15, 5))
ax[0].imshow(img)
ax[1].imshow(lsm[0,:,:]>0)
ax[1].imshow(np.ma.masked_array((labels==ic), 1-blobs), cmap=plt.cm.rainbow)
ax[2].hold(True)
ax[2].imshow(labels==ic , cmap=plt.cm.rainbow)
found_point=True
break
# Compute new distanceMAX
distMAX=distVALID.max()
```

```
plt.show()
if found_point:
break
infile.close()
lsmfile.close()

if __name__ == "__main__":
main()
```