14 PQHS 471 Notes Week 14

14.1 Week 14 Day 1

14.1.1 ESL 14.2

The goals of **market basket analysis** (MBA) are:

- Find items that are frequently purchased together. (Density estimation)
- Find (simple) prediction models that have both good prediction (i.e., high confidence and/or lift) and good impact (i.e., not too low frequency/support).

Supermarket transaction data may contain billions of transactions (i.e., $n \sim 10^9$) and many features. There may be various levels of interest: product categories (e.g., peanut butter; $p \sim 10^4$), brands (e.g., Jif; $p \sim 10^5$), UPCs (e.g., Jif Extra Crunchy 16 oz; $p \sim 10^6$), etc.

Feature coding:

- Binary: $X_j = 1$ if item j is purchased as part of a transaction and $X_j = 0$ if not.
- Numerical: X_j is the number of item j purchased, or in number categories (e.g., "0", "1", "2-4", ">4").
  - This allows us to evaluate events like "2 or more Jif Extra Crunchy 16 oz are bought".
  - *Conjunctive rules* may be used to simplify the problem: Only focus on $\bigcap_j (X_j \in s_j)$, where s_j is a subset of S_j, which is the set of all possible values for X_j. Under conjunctive rules, the scenario in the left panel of ESL Figure 14.1 will be ignored.
- Dummy variable: For example, $Z_k = 1$ if the customer bought "2-4" "Jif Extra Crunchy 16 oz"; $Z_k = 0$ otherwise. (ESL Figure 14.1 middle and right panels)
  - This is equivalent to requiring that s_j either contains a single value or is the whole set S_j.

The dummy variable coding is currently the standard formulation of the market basket problem. With the dummy variable coding, we have the concept of **item set**, which is a set defined as $K = \{Z_j : Z_j = 1\}$. The **support** of an item set K is the probability that the items in K are bought together; it is denoted as $T(K)$.

An **association rule** is just a rule $A \to B$, where the item sets A and B do not overlap. Here, A is called the *antecedent* and B is the called the *consequent*. Some example association rules are in ESL 14.2.3.

- The **support** of the rule is $T(A \to B) = T(A \text{ and } B) = P(A \text{ and } B)$.
- The **confidence** of the rule is $C(A \to B) = T(A \text{ and } B)/T(A) = P(B|A)$.
- The **expected confidence** is $T(B) = P(B)$.
- The **lift** of the rule is $L(A \to B) = C(A \to B)/T(B) = P(A \text{ and } B)/P(A)P(B)$.

Notes:

- The lift of the rule $A \to B$ is the same as the lift of the rule $B \to A$.
- The concept of lift is similar to the concept of "relative risk" in genetic epidemiology. For example, the sibling relative risk for type II diabetes is defined as the ratio of Pr(type II diabetes | a sibling has type II diabetes) and Pr(type II diabetes) in general population.
- The lift is the ratio of the real joint probability to that under independence. (A similar concept is the mutual information between two random variables.)

Association rule mining: Identify all association rules that have "good" support, confidence and/or lift. Specifically, $T(A \to B) > t$, $C(A \to B) > c$, and $L(A \to B) > l$, where $t > 0$, $c > 0$ and $l \geq 1$.

Any rule $A \to B$ with support $> t$ must have $T(K) > t$, where K is the union of the item sets A and B. Thus we first need to identify all candidate item sets K that have support $> t$. The **Apriori algorithm** (Agrawal and Srikant, 1994) achieves exactly this. Let L_m be the set of all size-m item sets that have support $> t$. Then for any item set in L_m, its size-k subset must be a member of L_k. The Apriori algorithm is the following:

1. Search all singletons (item sets of size 1) to create L_1.
2. For $m = 2, 3, \ldots$, generate L_m:
   a. Join step: For any two sets $p, q \in L_{m-1}$ that share $m-2$ items, create their union c;
   b. Prune step: If c has a size-$(m-1)$ subset that is not in L_{m-1}, drop c;
   c. Check support: If c has support $> t$, keep it.

3. For every item set c with support $> t$, consider every possible split of c into two subsets, A and B, and if the confidence (and/or lift) of $A \to B$ is above a predetermined threshold, keep it.

R has package arules for mining association rules. The package arulesViz provide visualization tools. The arules package comes with a small dataset Groceries, in which the features are 169 product categories in binary coding.

```
library(arules)
?Groceries
data(Groceries)  ## load the dataset
summary(Groceries)

dim(Groceries)  ## 9835 x 169  ## dim(Groceries@data) is 169 x 9835
length(Groceries)  ## 9835
class(Groceries)  ## "transactions"
str(Groceries)  ## Groceries@data is stored as a sparse matrix
itemInfo(Groceries)  ## Groceries@itemInfo , a data frame
unique(Groceries@itemInfo$level2)  ## 55 categories in level 2
unique(Groceries@itemInfo$level1)  ## 10 categories in level 1

inspect(head(Groceries, 3))
size(head(Groceries, 3))
LIST(head(Groceries, 3))

table(size(Groceries))
mean(size(Groceries)) / 169  ## density of data: mean fraction of items per transaction
head(sort(table(unlist(LIST(Groceries))), decreasing=T))  ## most frequent items
library(magrittr)
Groceries %>% LIST %>% unlist %>% table %>% sort(decreasing=T) %>% head

itemFrequency(Groceries)  ## relative frequency of items (probability)
itemFrequency(Groceries, type="absolute")  ## absolute frequency of items (count)
itemFrequencyPlot(Groceries, topN=10, main="Item Frequency")
itemFrequencyPlot(Groceries, topN=10, type="absolute", main="Item Frequency")

## check itemFrequency(Groceries)
aa = sort(as.numeric(table(unlist(LIST(Groceries)))))
all.equal(sort(itemFrequency(Groceries)), aa / 9835, check.attributes=F)  ## True
all.equal(sort(itemFrequency(Groceries, type='absolute')), aa, check.attributes=F)  ## True
```

The arules package provides two main functions: eclat() for mining frequent item sets, and apriori() for mining frequent item sets or association rules. To turn off the printout of eclat() and apriori(), specify control=list(verbose=F). To mine frequent item sets, use either of the two functions:

```
freqitems = eclat(Groceries, parameter=list(supp=0.05, maxlen=10))
#freqitems = eclat(Groceries, parameter=list(supp=0.05, maxlen=10, minlen=2))
inspect(freqitems)

freqitems2 = apriori(Groceries, parameter=list(supp=0.05, maxlen=10, target="frequent itemsets"))
inspect(freqitems2)

library(arulesViz)
plot(freqitems)
```

To mine association rules:

```
rules = apriori(Groceries, parameter=list(supp=0.01, conf=0.5, maxlen=10))
plot(rules)

rules_supp = sort(rules, by="support", decreasing=T)
```

```
inspect(head(rules_supp))
rules_conf = sort(rules, by="confidence", decreasing=T)
inspect(head(rules_conf))
rules_lift = sort(rules, by="lift", decreasing=T)
inspect(head(rules_lift))

## get rules that lead to buying 'whole milk'
rules = apriori(Groceries, parameter=list(supp=0.001, conf=0.8),
                appearance=list(default="lhs", rhs="whole milk"))
## get rules that start from buying 'whole milk'; notice minlen=2
rules = apriori(Groceries, parameter=list(supp=0.001, conf=0.1, minlen=2),
                appearance=list(default="rhs", lhs="whole milk"))
```

Here is another MBA demonstration (code) using the Online Retail Data Set.

Market basket analysis can be used on other types of data. For example, the `arules` package has the `Adult` dataset, which is the UCI `Adult` data in the "transactions" format for use with `arules`; `AdultUCI` is the original data in a data frame. Dummy variable coding is used in the `Adult` dataset. Numerical variables are converted to categories; e.g., `age` has 4 categories: Young, Middle-aged, Senior, Old. There are 115 dummy variables; the variables `fnlwgt` and `education-num` in `AdultUCI` are not in `Adult`.

```
library(arules)
data("Adult"); data("AdultUCI")
str(Adult)
table(Adult@itemInfo$variables)
str(AdultUCI)

LIST(Adult[1])  ## first person
size(Adult)  ## those with size<13 have missing data
table(size(Adult))

which(size(Adult) == 9)  ## take a look at an example
LIST(Adult[34722])
AdultUCI[34722,]
```

Other datasets in the `arules` package.

```
data(package="arules")
```

Generalized association rule (ESL 14.2.4–14.2.7)

14.1.2 Density estimation in high dimensions

In machine learning, models for density estimation are called **generative models**. (In contrast, models for prediction are called **discriminative models**.) Methods for density estimation fall in two broad types:

- Specify a model (i.e., a family of distributions indexed by parameters), and maximize the likelihood (or equivalently, minimize the cost $C = -\log L$) to estimate the parameters. This is a typical way in classical statistics. For high-dimensional data, a popular method is the restricted Boltzmann machines (RBMs).
- Transform the problem into a supervised learning problem and apply the wide variety of supervised learning tools to attack the problem. ESL 14.2.4 describes one approach (below). For high-dimensional data, a popular deep learning method is the generative adversarial networks (GANs).

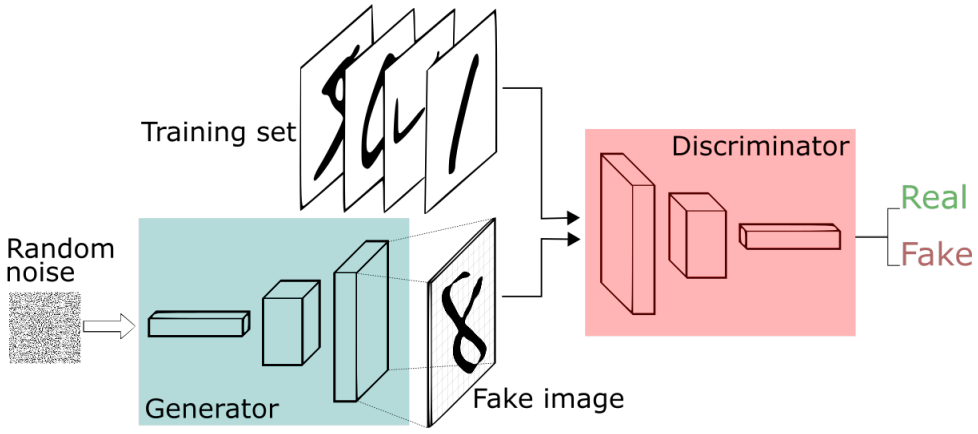
The approach in ESL 14.2.4: Let $g(x)$ be the density function of an unknown distribution, and N be the number of observations from that distribution. Let $g_0(x)$ be the density function of a known reference distribution over the same sample space. For example, $g_0(x)$ could be a uniform distribution over the sample space. Consider a mixture distribution with density function $(g(x) + g_0(x))/2$. If we know $\mu(x) = \frac{g(x)}{g(x)+g_0(x)}$, then $g(x) = g_0(x)\frac{\mu(x)}{1-\mu(x)}$.

To learn $\mu(x)$, we can generate a large number, N_0, of observations from $g_0(x)$, and then assign weight $N_0/(N+N_0)$ to the observations in the real data, and weight $N/(N+N_0)$ to the observations in the simulated data. The pooled dataset has equal total weight from the two distributions. We assign $Y=1$ to the observations in the real data and $Y=0$ to those in the simulated data. Now we can apply ML tools to estimate $\mu(x)$.

14.1.3 Generative adversarial networks (GANs)

GANs (Goodfellow et al., 2014) are for high-dimensional density estimation. In GANs, the unsupervised learning problem is transformed to an iteration of solving supervised learning problems.

Goodfellow et al., (2014) nicely describes the idea behind GANs: "In the proposed adversarial nets framework, the generative model is pitted against an adversary: a discriminative model that learns to determine whether a sample is from the model distribution or the data distribution. The generative model can be thought of as analogous to a team of counterfeiters, trying to produce fake currency and use it without detection, while the discriminative model is analogous to the police, trying to detect the counterfeit currency. Competition in this game drives both teams to improve their methods until the counterfeits are indistiguishable from the genuine articles." The image below is from https://deeplearning4j.org/generative-adversarial-network



The method has two components:

- The **generator** is a model $G_\theta(z)$ for generating fake data $x_0 = G(z)$, where z is from a known distribution.
- The **discriminator** is a classification model $D_\delta(x)$ learned from the input $\{x_1,\ldots,x_N,x_{01},\ldots,x_{0N}\}$ and output $\{1,\ldots,1,0,\ldots,0\}$. $D_\delta(x)$ is a value between 0 and 1 to represent the probability that x is from the real data. The function $D_\delta(x)$ provides a score to every x, and this score is used to drive the learning of G_θ.

The method works in the following way:

1. Initialize the generator G_θ with $\theta=\theta_0$.
2. Iterate between the following two steps:
   - Given the current function G_θ, generate fake data x_0. Train the discriminator $D_\delta(x)$ to tell apart the real and fake data.
   - Given the current function D_δ, train the generator to maximize $D_\delta(G_\theta(z))$ over θ, or equivalently, to minimize the cost $\log(1-D_\delta(G_\theta(z)))$.

Goodfellow et al. (2014) also describes a stochastic version of the algorithm to be used for large datasets.

A famous application is the generation of fake human faces (video) by a Karras et al., 2018, a team from NVIDIA who trained a GAN progressively using 30,000 pictures of celebrity.

Since the publication of the original GAN paper, many variants of GANs have been developed. Here is an example of applying a variant of GAN, Auxiliary Classifier GAN (ACGAN), to the MNIST dataset using the R `keras` package.

14.1.4 Assignment

1. Reading for next lecture: ESL 14.8–14.9, 14.4

14.2 Week 14 Day 2

MDS and SOM are dimension reduction techniques, mostly for the purpose of visualization. Unlike PCA, the target dimension needs to be specified in MDS and SOM.

14.2.1 ESL 14.8

In **multi-dimensional scaling** (MDS), we seek to identify a representation (called a **configuration**), $\{z_1, \ldots, z_n\}$ in R^k, of the original data such that their pairwise dissimilarity (or similarity) matrix $M' = \{d'_{ij}\}$ is as close as possible to that for the original data, $M = \{d_{ij}\}$. For similarity, we use the notation $M' = \{s'_{ij}\}$ and $M = \{s_{ij}\}$.

Applications of MDS:

- *Proximity data* (e.g., pairwise similarity or dissimilarity, confusion matrix);
- Visualization of data (an alternative to principal components) ($k = 2, 3$);
- Visualization of networks ($k = 2, 3$);
- In chemistry, identification of the spatial structure of molecules ($k = 3$).

For MDS, we need to define:

1. a measure of pairwise dissimilarity d_{ij} (or similarity s_{ij}) between observations in the original data;
2. a measure of pairwise dissimilarity d'_{ij} (or similarity s'_{ij}) between observations in R^k;
3. a **stress function**, which is a measure of closeness between matrices M and M'.

The goal of MDS is to minimize the stress function. Gradient descent is often used for this purpose.

Classical metric scaling is on pairwise similarity: The measure of similarity in R^k is defined as the centered inner product $s'_{ij} = \langle z_i - \bar{z}, z_j - \bar{z} \rangle$. The stress function is $S_M = \sum_{i \neq j} (s_{ij} - s'_{ij})^2$.

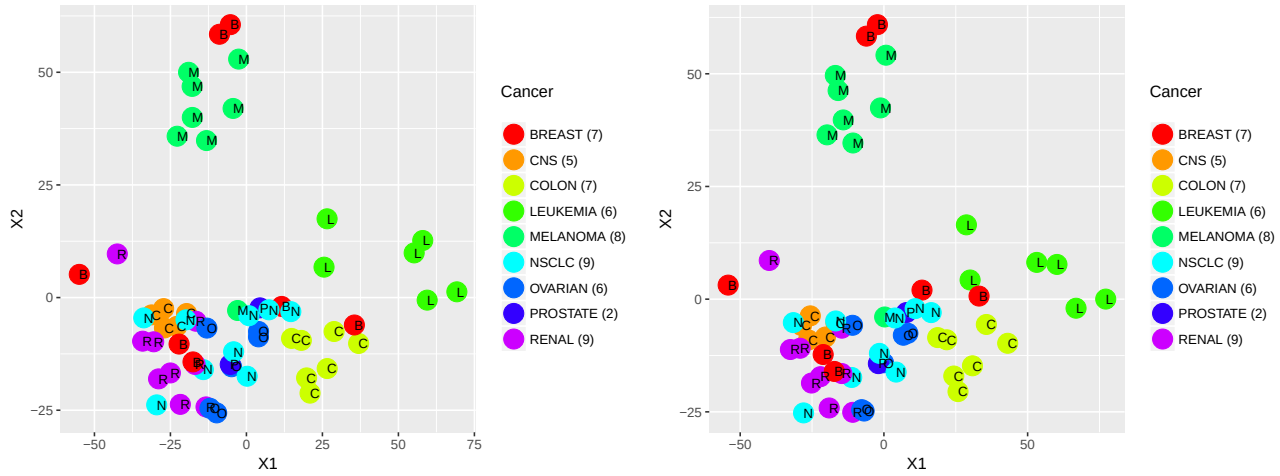
Notes:

- If the original data are observed, $\{x_1, \ldots, x_n\}$, and their similarity measure is also the centered inner product $s_{ij} = \langle x_i - \bar{x}, x_j - \bar{x} \rangle$, the classical scaling is effectively the same as principal components.
- Suppose the data is already centered. Then $s((4,0),(1,1)) = s((1,0),(4,4)) = s((2,0),(2,2))$. These three pairs of vectors have the same "similarity" as measured by inner product, but their distances are quite different.
- This method is also called the *Principal Coordinate Analysis* in ecology.

The base R has a function `cmdscale()` for the classifical MDS. It takes a symmetric matrix or a "dist" object. The matrix is a dissimilarity matrix, which is internally first converted to an inner product matrix B.

```
library(ggplot2)
library(ISLR)
nci.labs=NCI60$labs
nci.data=NCI60$data
dim(nci.data)  ## 64 x 6830
table(nci.labs)
data.dist1 = dist(scale(nci.data))

## define the subset of the 9 known cancers
ncitypes = c("BREAST", "CNS", "COLON", "LEUKEMIA", "MELANOMA", "NSCLC", "OVARIAN", "PROSTATE", "RENAL")
nci.data2 = NCI60$data[NCI60$labs %in% ncitypes, ]
nci.labs2 = NCI60$labs[NCI60$labs %in% ncitypes]
dim(nci.data2)  ## 59 x 6830
data.dist2 = dist(scale(nci.data2))

## labels for the plots
aa = table(nci.labs2); cancerlabels = paste(names(aa), ' (', aa, ')', sep='')

## Using all 64 samples
cmds2a = cmdscale(data.dist1, k=2, add=T, list.=T)
```

```
apply(cmds2a$points, 2, mean)  ## zero mean (not exactly due to rounding errors)
aa2a = data.frame(cmds2a$points)

ggplot(subset(aa2a, nci.labs %in% ncitypes), aes(X1, X2)) +
  geom_point(aes(colour=nci.labs2), size=6) +
  labs(color="Cancer\n") +
  scale_color_manual(labels=cancerlabels, values=rainbow(10)[1:9]) +
  geom_text(aes(label=substr(nci.labs2, 1, 1)), hjust=0, size=3)

## Using 59 samples (excluding unclassified ones)
cmds2b = cmdscale(data.dist2, k=2, add=T, list.=T)
aa2b = data.frame(cmds2b$points)

ggplot(aa2b, aes(X1, X2)) +
  geom_point(aes(colour=nci.labs2), size=6) +
  labs(color="Cancer\n") +
  scale_color_manual(labels=cancerlabels, values=rainbow(10)[1:9]) +
  geom_text(aes(label=substr(nci.labs2, 1, 1)), hjust=0, size=3)
```



Now the 3D classical MDS.

```
cmds3a = cmdscale(data.dist1, k=3, add=T, list.=T)
cmds3b = cmdscale(data.dist2, k=3, add=T, list.=T)
x = cmds3b$points[,1]
y = cmds3b$points[,2]
z = cmds3b$points[,3]

library(rgl)
plot3d(x, y, z, size=10, col=as.numeric(factor(nci.labs2)))
text3d(x, y, z, nci.labs2, cex=.7)
```

Alternative metric scaling methods:

- *Kruskal–Shepard metric scaling* focuses on dissimilarity and uses the stress function $\sqrt{\sum_{i\neq j}(d_{ij}-d'_{ij})^2}$. It is also called *least squares scaling*. This is different from the classical scaling.
- *Sammon mapping* uses the stress function is $\sum_{i<j}\frac{(d_{ij}-d'_{ij})^2}{d_{ij}}$. For a pair of observations with a small d_{ij}, the "stress" $\frac{(d_{ij}-d'_{ij})^2}{d_{ij}}$ is relatively larger than that in the least squares scaling. As a result, d'_{ij} is pushed to be close to d_{ij} even when d_{ij} is small. In contrast, in the classical and least squares scaling, the results are driven mostly by large s_{ij} or d_{ij}.

In **non-metric scaling**, we do not focus directly on the original dissimilarity values d_{ij}, but on any possible

increasing monotonic function of them, $\theta(d_{ij})$.

- *Shepard–Kruskal non-metric MDS*: The stress function is $\sqrt{\frac{\sum_{i<j}[\theta(d_{ij})-d'_{ij}]^2}{\sum_{i<j} d'^2_{ij}}}$.
- *Sammon non-metric MDS*: The stress function is $\sum_{i<j} \frac{(d_{ij}-d'_{ij})^2}{d_{ij}} / \sum_{i<j} d_{ij}$.

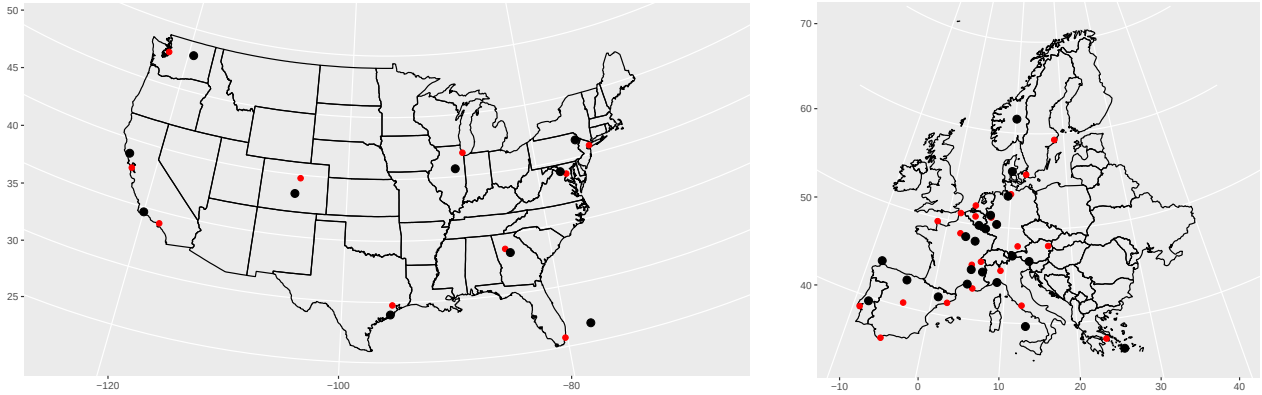
The R `MASS` library has functions `isoMDS()`, `Shepard()`, and `sammon()` for Kruskal–Shepard and Sammon non-metric MDS. For the NCI60 dataset, `isoMDS()` gives very similar results as `cmdscale()`.

```
library(MASS)
mds2 = isoMDS(data.dist1, k=2)
all.equal(cmds2a$points, mds2$points)

plot(mds2$points)
text(mds2$points, labels = nci.labs, cex=.8)

## Now the 3D MDS
mds3 = isoMDS(data.dist1, k=3)
all.equal(cmds3a$points, mds3$points)
range(mds3$points)

library(rgl)
x = mds3$points[,1]
y = mds3$points[,2]
z = mds3$points[,3]
plot3d(mds3$points)
text3d(x,y,z, nci.labs, cex=.7)
```

Example: Pairwise distances between cities. R has datasets `UScitiesD` and `eurodist`, which are "dist" objects containing pairwise distances between some cities in the US and Europe. According to their help page, "`eurodist` gives the road distances (in km) between 21 cities in Europe," and "`UScitiesD` gives 'straight line' distances between 10 cities in the US." This example shows how much 2D relative positions can be recovered by the classical MDS. Because flipping and rotatation do not change pairwise distances, the results may be flipped or rotated or both.

```
## Define a rotation transformation given angle in degrees counterclockwise
rotate = function(angle) {
  aa = angle/180*pi
  matrix(c(cos(aa), -sin(aa), sin(aa), cos(aa)), 2)
}

USdist.cmds = cmdscale(UScitiesD)
USdist.cmds = -USdist.cmds  ## flip both coordinates
USdist.cmds = USdist.cmds %*% rotate(-10)  ## rotate 10 degrees clockwise
plot(USdist.cmds, xlab="", ylab="", asp=1, axes=F, xlim=c(-1425, 1350))
text(USdist.cmds, rownames(USdist.cmds), adj=c(-.2, 1), cex=.8)

eurodist.cmds = cmdscale(eurodist)
eurodist.cmds[,2] = -eurodist.cmds[,2]  ## flip the latitude coordinate
eurodist.cmds = eurodist.cmds %*% rotate(10)  ## rotate 10 degrees counterclockwise
plot(eurodist.cmds, xlab="", ylab="", asp=1, axes=F, xlim=c(-1900, 2800))
text(eurodist.cmds, rownames(eurodist.cmds), adj=c(-.2, 1), cex=.8)
```

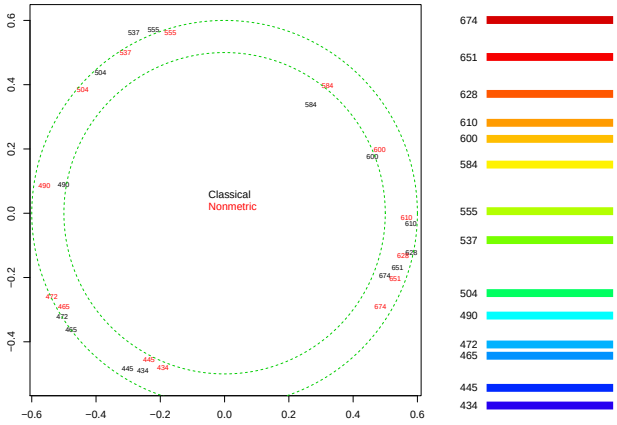
Now we put the MDS results on the real maps, with the correct locations in red and the MDS coordinates in black dots. The results for European cities are not as good probably because the distances are road distances. The distances for US cities are straight line distances. Note that the "spherical" straight line distance between two cities is often shorter than their distance on a 2D map. This effect is stronger when the two cities are closer to the poles.



Example: Ekman data. The data are average perceived similarity between 14 colors. The help page of `ekman` has more details. We apply MDS methods to obtain a 2-dimensional representation of the 14 colors. The wavelengths of the 14 colors cover most of the spectrum of a full rainbow. The surprisingly good circular shape of the MDS results indicates that the methods yield a good representation.

```
library(smacof)
library(MASS)
ekman
ekmandis = sim2diss(ekman, method=1)  ## convert to dissimilarity = 1 - similarity

ekman.cmds2 = cmdscale(ekmandis, k=2, add=T, list.=T)
ekman.mds2 = isoMDS(ekmandis, k=2)

## Draw the MDS results
plot(ekman.mds2$points, type="n", asp=1, xlab="", ylab="")
text(ekman.cmds2$points, labels=rownames(ekman.cmds2$points), cex=.7)
text(ekman.mds2$points, labels=rownames(ekman.mds2$points), cex=.7, col=2)
legend("center", text.col=1:2, legend=c('Classical', 'Nonmetric'), bty='n')

## Draw circles for reference
theta = seq(0, 2*pi, length=180)
for(i in seq(0.5, 0.6, 0.1)) lines(x=i*cos(theta), y=i*sin(theta), lty=2, col=3)

## Draw the 14 colors
source("wavelength_to_rgb.r") ## From https://gist.github.com/friendly/67a7df339aa999e2bcfcfec88311abfc
aa = as.numeric(rownames(ekman.cmds2$points))
```

```
wavecol = NULL
for(i in 1:length(aa)) wavecol[i] = wavelength_to_rgb(aa[i])

plot(rep(0, length(aa)), aa, axes=F, xlab='', ylab='', type='n', xlim=c(-.2, 1))
segments(rep(0, length(aa)), aa, rep(1, length(aa)), aa, col=wavecol, lwd=12, lend=1)
text(rep(0, length(aa)), aa, adj=1.5, labels=aa)
```



In addition to the MDS functions above,

- The R package `smacof` provides a suite of functions for MDS. It also has some classical proximity datasets: `ekman`, `morse`, `winedat`, etc.
- The R package `vegan` has functions `monoMDS()` and `wcmdscale()`.
- GGobi has a plugin, GGvis, for MDS. But its R version `rggobi` seems not to have the MDS features. The R package `ggvis` is for interactive graphics and is not relevant to `rggobi`. [Here the first "G" refers to GTK. Earlier versions were called XGobi and XGvis, with "X" referring to the X Window system.]

14.2.2 ESL 14.9

ESL Figure 14.44 shows a scenario where the traditional MDS may fail.

Local MSD (Chen and Buja, 2008): Its stress function is (14.106). The focus is on small d_{ij}'s. An example is shown in ESL Figure 14.44.

Local linear embedding (LLE) (Roweis and Saul, 2000): Every observation has a "local structure" with its neighboring observations. Find a low-dimensional representation that keeps this local structure. ESL Figure 14.45 shows an application of LLE to 1,965 pictures of faces in a resolution of 20×28.

Isometric feature mapping (ISOMAP) (Tenenbaum et al., 2000)

14.2.3 Packages for visualization

The R package `ggplot2` implements static graphics. The "gg" stands for the "Grammar of Graphics", the title of a book by Leland Wilkenson.

The R package `ggvis` implements interactive graphics. The graphics are rendered in a web browser using Vega, a visualization language build on the D3 JavaScript library.

Plotly is also built on the D3 library. Its graphics rendering requires a 'plotly' account.

The R package `googleVis` is an interface to Google Charts to create interactive charts. Here are some examples.

The R package `ggmap` is for drawing maps. The current CRAN version 2.6 is from more than 2 years ago. To install version 2.7, use `devtools::install_github("dkahle/ggmap")`. Some demonstrations are at https://github.com/dkahle/ggmap .

14.2.4 ESL 14.4

Self-organizing maps (SOM) is another method for dimension reduction and visualization. The online version of the SOM algorithm is:

- Start with a grid of **prototypes** in R^p, where p is the number of features. Let m_j be the initial position of prototype l_j.
- Let data gradually pull the prototypes towards them. Process data one by one (online algorithm).
  - For observation x_i, find the prototype m_j that is closest to it, move m_j and its "neighbors" m_k towards x_i. A simple version is in (14.46): $m_k \leftarrow m_k + a(x_i - m_k)$, where a is the *learning rate*. (14.47) is a slightly more sophisticated version.

Notes:

- The initial positions of the prototypes can be on the hyperplane of the first 2 principal components. The number of prototypes is often high. A 2D grid can be rectangular or hexagonal (triangular).
- Neighboring prototypes are defined according to their positions on the original grid, not on m_j's.
- As we move along, the learning rate and the neighbor "radius" may be gradually reduced (similar to simulated annealing).
- Using the Euclidean distance requires feature standardization to make sense.
- The result can be influenced by the order of observations. Online algorithms are also sensitive to outliers.
- Presenting the final result in the original grid can be misleading because the final locations of the prototypes can be highly disfigured, as shown in ESL Figure 14.17.
- If the neighbor of a prototype is always itself, this algorithm becomes an online version of K-means clustering.

Examples:

- ESL Figures 14.15–14.17. Note that the size of the grid is $25 \gg 3$.
- ESL Figure 14.19. The data are 12,088 articles in a newsgroup. The data are first processed into a *term-document matrix*, which is often very sparse.

The **batch version of SOM** is similar to K-means clustering. Iterate the following:

1. map all observations to their closest prototypes;
2. update prototype m_k by a weighted average of x_i that are mapped to the neighborhood prototypes of m_k.

If the neighbor of a prototype is always itself, then this batch version becomes the K-means clustering algorithm. In general, SOM can be viewed as a constrained version of K-means clustering.

R packages:

- `kohonen`
  - https://www.r-bloggers.com/self-organising-maps-for-customer-segmentation-using-r/
  - SOM applied to NBA data: https://clarkdatalabs.github.io/soms/SOM_NBA
- `som`