Week 10: Clustering and Modeling

Ben Schmidt

4/2/2015

Last week we looked at **supervised learning and comparison**: techniques that use *known* metadata to compare groups.

This week, the techniques we'll discuss are what it known as **unsupervised learning**; for inferring patterns out of data where you *don't* necessarily have data.

Clustering

Clustering means putting sets into groups.

We're going to talk about using a very large number of dimensions–the words in a document. But you can do it on a comparatively smaller number of features, as well. The simplest case is clustering on a single dimension. Let's go back to our list of cities as an example.

```
cities = read.csv("data/CESTACityData.csv")

cities = cities %>% filter(X2010>250000) %>%
  select(CityST,LAT,LON,X2010,X2000) %>% filter(LON<0)

ggplot(cities) + geom_point(aes(x=LON,y=1))
```

How would you group these points together?

Hierarchical clustering uses something called a *dendrogram* to do this. It finds the two closest points, and joins them together, pretending they're a single point [1]; then finds the next two closest points, and joins *them* together; and keeps on going until there's a single group, encompassing everything.

First, we'll just cluster on longitude.

This sort of makes sense, but there are some weird pairings. Do Anchorage and Honolulu really have that much in common?

Also note how the final ordering isn't east-to-west; that information is *lost*.

```
distances = dist(cities %>% select(LON))

clustering = hclust(distances)

plot(clustering,labels=cities$CityST)
```

A *two-dimensional* clustering could use latitude *and* longitude.

```
distances = dist(cities %>% select(LAT,LON))

plot(hclust(distances),labels=cities$CityST)
```

[1] This is a simplification–in fact, there are a number of different ways to treat the amalgamated clusters

But you could also do a three-dimensional clustering, that relates both location and population change.

The scales will be very different on these, so you can use the R function `scale` to make them all follow the same distribution. (Try fiddling around with some numbers to see what it does, or just put in `?scale`.)

That starts to cluster cities together based on things like both location and demographic characteristics: for instance, New Orleans ends up connected to the Rust Belt because it, too, experienced population decreases.

```
distances = dist(cities %>%
                   mutate(LON=scale(LON),LAT=scale(LAT),change=scale(log(X2010/X2000)))%>%
                   select(LAT,LON,change)
                 )

plot(hclust(distances),labels=cities$CityST)
```

Here's an example of doing this with texts, we'll pull the Federalist papers for analysis. There's a new function defined in the `commonFunctions.R` file online, so download or cut and paste that in as necessary.

```
FederalistPapers =
  bookwormTDMatrix(groups=list("author","fedNumber","*unigram"),database="federalist",cutoff=15) %>%
  arrange(meta_fedNumber)

# Select everything that doesn't start with the term "meta_: this lets us get only the words."
# Just the words

justWords = FederalistPapers %>% select(-starts_with("meta_"))
```

To make a dendrogram, we once again take the distances and plot a hierarchical clustering.

By making the labels be the authors, we can see whether the clusters are grouping similar people together.

```
relations = dist(justWords)

plot(hclust(relations),labels=FederalistPapers$meta_author)
```

K-means clustering

Sometimes, it makes sense to choose a fixed number of clusters. Hierarchical clustering *can* provide this, but cutting the tree at any particular height. But an expecially widely used form of find the best group of 5 (or 10, or whatever) is k-means clustering. A nice example video is online here.

Let's go back to our data to see how kmeans handles the cities. We could run kmeans on just one dimension, but let's just straight to two:

```
cities = read.csv("data/CESTACityData.csv")

cities = cities %>% filter(X2010>250000) %>% select(CityST,LAT,LON,X2010,X2000) %>% filter(LON<0)

kMeansModel = kmeans(cities %>% select(LAT,LON),centers=6)
```

```
names(kMeansModel)

cities$cluster = kMeansModel$cluster

ggplot(cities) + aes(x=LON,y=LAT,color=factor(cluster)) + geom_point(size=10)
```

You can use this clustering to pull out similar groups of words. You'd have to adjust to the size of the words, though, or else the clusters will just be in terms of how common the words are. This could be a hard exercise, or something for office hours.

But worth knowing is that it's easy to switch to terms of analysis if you are working with roughly a "square" matrix.

```
justWords = FederalistPapers %>% select(-starts_with("meta_"))

transposed = t(justWords)

wordGroups = kmeans(transposed,centers=10)
```

Topic Modeling

Clustering and Modeling

To do topic modelling at home, you'll install the "mallet" package for R.

Note: this depends on the rJava package, which some people may have trouble installing. If `install.packages("mallet")` doesn't work on your machine, write the list for help.

```
library(mallet)

federalistPapers = read.table("data/federalist-mallet.tsv",
                              sep="\t",quote="",
                              comment.char="")

input=federalistPapers

n.topics=32

mallet.instances <- mallet.import(input$id, input$text,
                                  token.regexp = "\\w+",preserve.case=F)

topic.model <- MalletLDA(num.topics=n.topics)
topic.model$loadDocuments(mallet.instances)

#Look at the word frequencies sorted in order.
vocabulary <- topic.model$getVocabulary()
word.freqs <- mallet.word.freqs(topic.model)
head(word.freqs[order(-word.freqs$term.freq),],20)

#Some preferences. Inside baseball: see Wallach and Mimno for what's going on.
topic.model$setAlphaOptimization(20, 50)
topic.model$train(300)
#Increase the fit without changing the topic distribution; optional
topic.model$maximize(10)

#Gets a list of the documents and topics
doc.topics <- mallet.doc.topics(topic.model, smoothed=T, normalized=T)
#Changes the orientation of that matrix to be horizontal:
topic.docs <- t(doc.topics)
```

```
#Gets a list of the top words.
topic.words <- mallet.topic.words(topic.model, smoothed=T, normalized=T)


#Assign some labels to the topics
topics.labels <- rep("", n.topics)
for (topic in 1:n.topics) {
  topics.labels[topic] <- paste(
    mallet.top.words(topic.model, topic.words[topic,], num.top.words=5)$words, collapse=" "
    )}

#to look at the labels, type "topics.labels"


rownames(doc.topics) = input$id
colnames(doc.topics) = topics.labels

library(reshape2)
allcounts = melt(doc.topics)
names(allcounts) = c("doc","topic","proportion")
```