8 Clustering/Support Vector Machines

The EM algorithm (Expectation-Maximization)

Consider a missing data problem where $x = (x_o, x_m)$; x_o is observered, x_m is missing data, and the distribution of the whole thing is $f(x_o, x_m; \theta)$. We want to maximize the likelihood given the observed data

$$L(\theta \mid x_o) = \int_{x_m} f(x_0, x_m; \theta) dx_m.$$

Letting $\ell = \log L$ denote the log-likelihood. Suppose we have an estimate θ^t for the argmax of ℓ, and we want to produce a better estimate θ^{t+1} (and so on, iteratively). We can write

$$\ell(\theta \mid x_0) = \mathbb{E}_{x_m \sim f(\cdot \mid x_o, \theta_t)}[\ell(\theta \mid x_o, x_m)] + \mathbb{E}_{x_m \sim f(\cdot \mid x_0, \theta_t)}[-\log f(x_m \mid x_o, \theta)]$$

Then by relative entropy/Jensen's inequality

$$\ell(\theta \mid x_0) \geq \mathbb{E}_{x_m \sim f(\cdot \mid x_o, \theta_t)}[\ell(\theta \mid x_o, x_m)] + \mathbb{E}_{x_m \sim f(\cdot \mid x_0, \theta_t)}[-\log f(x_m \mid x_o, \theta^t)]$$

It therefore makes sense to choose

$$\theta^{t+1} = \arg\max_{\theta} \mathbb{E}_{x_m \sim f(\cdot \mid x_o, \theta_t)}[\ell(\theta \mid x_o, x_m)]$$

That is the EM-algorithm.

The Hard-EM algorithm makes a simplification. Iteratively, to go from θ^t to θ^{t+1}:

- Pick x_m^t to maximize $f(x_o, x_m \mid \theta^t)$.
- Pick θ^{t+1} to maximize $f(x_o, x_m^t \mid \theta^{t+1})$.

Mixture models

Consider

- a family of probability measures $f_1, \ldots f_K$, and
- a discrete probability distribution on the set $\{1, 2, ..., k\}$ given by $w_1 + \cdots + w_K = 1$.

Then a mixture probability distribution is defined by $f(\cdot) = \sum_{i=1}^K w_i f_i(\cdot)$. For example

$$f(x) = \frac{1}{2}\frac{1}{\sqrt{2\pi}}\exp(-x^2/2) + \frac{1}{3}\frac{1}{\sqrt{2\pi}}\exp(-(x-1)^2/2) + \frac{1}{6}\frac{1}{\sqrt{2\pi}}\exp(-(x-2)^2/2)$$

is a mixture of $N(0,1)$, $N(1,1)$ and $N(2,1)$ with $(w_1, w_2, w_3) = (\frac{1}{2}, \frac{1}{3}, \frac{1}{6})$.

K-means clustering

- Consider a distance function d (perhaps $d(x,y) = \|x-y\|_2^2$ or $d(x,y) = \|x-y\|_1^1$)
- Consider a collection of K points in $\mathbb{R}^d$ denoted $\theta=(c_1, \ldots c_K) \in (\mathbb{R})^K$. They parameterize the mixture distribution $\frac{1}{K}\sum_i \exp(-d(x,c_i))$.
- Consider a sample of size n from the mixture model. Let $x_o \in (\mathbb{R})^n$ denote the points picked, and let $x_m \in \{1,\ldots,K\}^n$ denote the cluster numbers corresponding to the samples.

Applying the Hard-EM algorithm produces the K-means algorithm (when $d(x,y) = \|x-y\|_2^2$) or the K-medians algorithm (when $d(x,y) = \|x-y\|_1^1$).

Gaussian Mixture models

To specify a K-component Gaussian mixture model, you need to specify, the size of the components (the $w_1 + \cdots + w_K = 1$), the means of the components, and the variances. Let I denote the identity matrix.. The different components can either

- all have the same covariance matrix with the form λI (EII) , or
- all have a covariance matrix of the form $\lambda_i I$ (VII), or
- ..., or
- they can all be allowed to totally arbitrary (VVV).

The more freedom you allow the covariance matrices to have, the better the fit for any given number of clusters, but the higher the risk of overfitting the data. The danger of overfitting is probably fairly small as long as the number of clusters is low.

Gaussian mixutre models can be fit using the EM-algorithm. It is implemented by the function `Mclust` in the R package `mclust`.

Questions

1. Derive the K-means algorithm from the Hard-EM algorithm. Code the Hard-EM algorithm in R and run it on the faithful dataset; who should you initialize the cluster centres? Compare your function with the `kmeans` R function.
2. Derive the EM-algorithm for a Gaussian mixuture model where the covariance matrix for all of the components is the identity matrix.
3. The e1071 library in R allows you to build support vector machines. i.e.

```
library(e1071)
distance=1
#Generate some training and test data.
y.train=rep(c(1,2),100)
x.train=matrix(y.train/sqrt(2),100,2)+rnorm(200,)
y.test=rep(c(1,2),100)
x.test=matrix(y.test/sqrt(2),100,2)+rnorm(200)
plot(x.train,col=y.train)
s=svm(x.train,y.train,type="C",kernel="l") #Linear SVM
summary(s)
#Measure accuracy
predicted=(y.test==predict(s,x.test))
mean(predicted)
plot(x.test,pch=4-3*predicted,col=y.test)
```

(a) Try changing `distance` and see how (i) the number of support vectors changes and (ii) how the prediction accuracy changes.

(b) Download mnist.small.RData. It cotains 10,000 training images (28x28 pictures of the digits 0,1,...,9 in train.X), the class labels (in train.labels), 1,000 test images (test.X) and the class labels (test.label). Divide train.X and test.X by 255 to fit make the entries fit the unit interval (SVMs prefer normalized data). Train a support vector machine using the training data, then test it using the test data.

(c) The `knn` function in the class library implements k-th nearest neighbour matching: it predicts the class of test labels by looking at the k-nearest neighbors (k is generally a small integer, i.e. 1) in the training set, and looking at the corresponding labels (using majority voting). Again using mnist.small.RData, compare SVM with k-nearest neighbour classification.