Clustering

Nuno Vasconcelos

ECE Department, UCSD
Statistical learning

- goal: given a function

\[ x \rightarrow f(.) \rightarrow y = f(x) \]

and a collection of example data-points, learn what the function \( f(.) \) is.

- this is called training.

- two major types of learning:
  - unsupervised: only \( X \) is known, usually referred to as clustering;
  - supervised: both are known during training, only \( X \) known at test time, usually referred to as classification or regression.
Clustering

why need to learn without supervision?

- in many problems labels are not available or impossible to get
- in the digit example, someone sat in front of the computer for hours to label all those examples
- for other problems the classes depend on the application
- a good example is image segmentation
  - if you want to know if this is an image of the wild or of a big city, there is probably no need to segment
  - if you want to know if there is animal, then you do
  - the segmentation mask usually not available
Supervised learning

- today we focus on clustering
- let’s start by recalling what we have learned so far
- we started with the optimal decision rule for a supervised learning problem

- given
  - a training set $D = \{(x_1, y_1), \ldots, (x_n, y_n)\}$
  - $x_i$ is a vector of observations, $y_i$ is the label

- we start by
  - estimating a probability model for each class
  - this includes $P_{X|Y}(x|i)$, $P_Y(i)$, for all i
Supervised learning

- this is done by maximum likelihood estimation

- which has two steps:
  1) we choose a parametric model for all probabilities

\[ P_{X|Y}(x \mid i; \Theta) \]

- 2) we select the parameters of class \( i \) to be the ones that maximize the probability of the data from that class

\[
\Theta_i = \arg \max_{\Theta} P_{X|Y}(D^{(i)} \mid i; \Theta)
\]

\[ = \arg \max_{\Theta} \log P_{X|Y}(D^{(i)} \mid i; \Theta) \]
Maximum likelihood

we have also seen that this is a relatively straightforward step

the solutions are the parameters such that

\[
\nabla_{\Theta} P_{X|Y}(x | i; \Theta) = 0
\]

\[
\theta' \nabla_{\Theta}^2 P_{X|Y}(x | i; \theta) \theta \geq 0, \quad \forall \theta \in \mathbb{R}^n
\]

note that you always have to check the second-order condition

we must also do this for the class probabilities \( P_Y(i) \)

- but here there is not much choice of probability model
- Bernoulli: ML estimate is the percent of training points in the class
Maximum likelihood

we have worked out the Gaussian case in detail

- \( \mathcal{D}^{(i)} = \{x_1^{(i)}, \ldots, x_n^{(i)}\} \) set of examples from class \( i \)
- the ML estimates are

\[
\begin{align*}
\mu_i &= \frac{1}{n} \sum_j x_j^{(i)} \\
\Sigma_i &= \frac{1}{n} \sum_j (x_j^{(i)} - \mu_i)(x_j^{(i)} - \mu_i)^T
\end{align*}
\]

\( P_Y(i) = \frac{n}{N} \)

there are many other distributions for which we can derive a similar set of equations

but the Gaussian case is particularly relevant for clustering (more on this later)
Supervised learning

- this gives probability models for all classes

we have also seen that

- assuming the zero/one loss,
- the optimal decision rule is

$$i^* (x) = \arg \max_i P_{Y|X}(i | X)$$

- which can also be written as

$$i^* (x) = \arg \max_i [\log P_{X|Y}(X | i) + \log P_Y(i)]$$

- this completes the process, we have a rule to classify any unseen point
Gaussian classifier

In the Gaussian case this can be written as

\[
i^* (x) = \arg \min_i [d_i (x, \mu_i) + \alpha_i]
\]

with

\[
d_i (x, y) = (x - y)^T \Sigma_i^{-1} (x - y)
\]

\[
\alpha_i = \log (2\pi)^d |\Sigma_i| - 2 \log P_Y (i)
\]

and can be seen as nearest neighbors with a funny metric

- each class has its own distance
- this is the sum of Mahalanobis for that class plus the \( \alpha \) constant
- we effectively have different metrics in different regions of the space
Gaussian classifier

A special case of interest is when

- all classes have the same $\Sigma_i = \Sigma$

$$i^*(x) = \arg \min_i [d(x, \mu_i) + \alpha_i]$$

with

$$d(x, y) = (x - y)^T \Sigma^{-1} (x - y)$$

$$\alpha_i = -2 \log P_Y(i)$$

Note:

- $\alpha_i$ can be dropped when all classes have equal probability
- this really becomes close to the NN classifier with Mahalanobis distance
- instead of finding the nearest neighbor, it looks for the nearest “prototype” $\mu_i$
Gaussian classifier

- $\Sigma_i = \Sigma$, two classes

  - one important property of this case is that decision boundary is a hyper-plane
  - can be shown by computing the set of points $x$ such that
    \[ d(x, \mu_0) + \alpha_0 = d(x, \mu_1) + \alpha_1 \]
    
    - and showing that they satisfy
    \[ \mathbf{w}^T (x - x_0) = 0 \]
    
    - this is the equation of a plane with normal $\mathbf{w}$ that passes through $x_0$
Gaussian classifier

- if all the covariances are the identity $\Sigma_i = I$

$$i^* (x) = \arg \min_i [d(x, \mu_i) + \alpha_i]$$

with

$$d(x, y) = \| x - y \|^2$$

$$\alpha_i = -2 \log P_y(i)$$

- this is just template matching with class means as templates
  - e.g. for digit classification
  - compare complexity to nearest neighbors!
Clustering

- In a clustering problem, we do not have labels in the training set.
- However, we can try to estimate these labels as well.
- Here is a strategy:
  - We start with random class distributions.
  - We then iterate between two steps:
    - 1) Apply the optimal decision rule for these distributions.
    - 2) Update the distributions by doing parameter estimation within each class.
Clustering

a natural question is: what model should we assume?

- well, let’s start as simple as possible
- assume Gaussian with identity covariances and equal \( P_Y(i) \)
- each class has a prototype \( \mu_i \)

the clustering algorithm becomes

- start with some initial estimate of the \( \mu_i \) (e.g. random)
- then, iterate between
- 1) classification rule:

\[
 i^* (x) = \arg \min_i \| x - \mu_i \|^2
\]

- 2) re-estimation:

\[
 \mu_i^{\text{new}} = \frac{1}{n} \sum_j x_j^{(i)}
\]
K-means (thanks to Andrew Moore, CMU)

1. Ask user how many clusters they'd like. *(e.g. k=5)*
K-means (thanks to Andrew Moore, CMU)

K-means

1. Ask user how many clusters they’d like. *(e.g. k=5)*

2. Randomly guess k cluster centers locations
K-means (thanks to Andrew Moore, CMU)

K-means

1. Ask user how many clusters they’d like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it’s closest to. (Thus each Center “owns” a set of datapoints)
K-means (thanks to Andrew Moore, CMU)

K-means
1. Ask user how many clusters they’d like. 
   *(e.g. k=5)*
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it’s closest to.
4. Each Center finds the centroid of the points it owns
K-means (thanks to Andrew Moore, CMU)

K-means

1. Ask user how many clusters they’d like. 
   (e.g. $k=5$)
2. Randomly guess $k$ cluster Center locations
3. Each datapoint finds out which Center it’s closest to.
4. Each Center finds the centroid of the points it owns...
5. ...and jumps there
6. ...Repeat until terminated!
K means

- the name comes from the fact that we are trying to learn “k” means

- it is optimal if you want to minimize the expected value of the squared error between vector \( x \) and template to which \( x \) is assigned

problems:

- how many clusters?
  - various methods available, Bayesian information criterion, Akaike information criterion, minimum description length
  - guessing can work pretty well
- local minimum only
- how do I initialize?
  - random can be pretty bad
  - mean splitting can be significantly better
mean splitting

- for $K = 1$ we just need the mean of all points ($\mu^1$)
- to initialize means for $K = 2$ perturb the mean randomly
  - $\mu_1^2 = \mu^1$
  - $\mu_2^2 = (1+\epsilon) \mu^1 \quad \epsilon << 1$
- then run $K$ means with $K = 2$
- initial means for $K = 4$
  - $\mu_1^4 = \mu_1^2$
  - $\mu_2^4 = (1+\epsilon) \mu_1^2$
  - $\mu_3^4 = \mu_2^2$
  - $\mu_4^4 = (1+\epsilon) \mu_2^2$
- then run $K$ means with $K = 4$
- etc ....
Empty clusters

- can be a source of headaches
- at the end of each iteration of K means
  - check the number of elements in each cluster
  - if too low, throw the cluster away
  - reinitialize the mean with a perturbed version of that of the most populated cluster

- note that there are alternative names:
  - in the compression literature this is known as the generalized Loyd algorithm
  - this is actually the right name, since Loyd was the first to invent it
  - it is used in the design of vector quantizers
Vector quantization

is a popular data compression technique

- find a codebook of prototypes for the vectors to compress
- instead of transmitting each vector, transmit the codebook index
- image compression example
  - each pixel has 3 colors (3 bytes)
  - we find the optimal 256 color prototypes
Vector quantization

we now have an image compression scheme

- each pixel has 3 colors (3 bytes)
- we find the nearest neighbor template in codebook
- we transmit its index
- since there are only 256 templates I only need one byte
- the decoder looks up the prototype in its table
- by sacrificing a little bit of distortion, I saved 2 bytes per pixel!
K-means

- but there are many other applications
  - e.g. image segmentation: decompose each image into component objects
  - we run k-means on the colors and look at the assignments
  - the pixels assigned to the red cluster tend to be from the house
K-means

- we can also use texture information in addition to color
  - many methods for doing this
  - here are some results

- note that this is not the state-of-the-art in image segmentation
- but gives a good idea of what k-means can do
Extensions

there are many extensions to the basic k-means algorithm

- it turns out that one of the most important applications is supervised learning
- remember that the optimal decision rule

$$i^*(X) = \arg \max_i [\log P_{X|Y}(X | i) + \log P_Y(i)]$$

is optimal insofar the probabilities $P_{X|Y}(x|i)$ are correctly estimated

- this usually turns out to be impossible, by definition, when we use parametric models like the Gaussian
- although these models are good local approximations, there are usually multiple clusters when we take a global view
- this can be captured with recourse to mixture distributions
Mixture distributions

Consider the following problem:

- Certain types of traffic banned from a bridge
- We need a classifier to see if the ban is holding
- The sensor measures vehicle weight
- Need to classify each car in OK vs banned class
- We know that in each class there are multiple clusters
- E.g., OK = {compact, station wagon, SUV}
- Banned = {truck, bus}
- Each of these is close to Gaussian, but for the whole class we get this
Mixture distributions

- this distribution is a mixture
  - the overall shape is determined by a number of sub-classes
  - we introduce a random variable $Z$ to account for this
  - given the value of $Z$ (the cluster) we have a parametric mixture component
  - e.g. a Gaussian

$$P_X(x) = \sum_{c=1}^{C} P_{X|Z}(x|c)P_Z(c)$$

$C$th “mixture component”

$C$th “component weight”

# of mixture components
Mixture distributions

- learning a mixture density is itself a clustering problem
  - for each training point we need to figure out from which component it was drawn
  - once we know the point assignments we need to estimate the parameters of the component
- this could be done with k-means
- a more generic algorithm is expectation-maximization
  - the only difference is that we never assign the points
  - in expectation step we compute the posterior class probabilities
  - but we do not pick the max
  - in the maximization step, the point contributes to all classes
  - weighted by its posterior class probability
Expectation-maximization

- **in summary**
  1. start with an initial parameter estimate $\mathcal{Y}^{(0)}$
  2. **E-step**: given current parameters $\Theta^{(i)}$ and observations in $D$, “guess” what the values of the $z_i$ are
  3. **M-step**: with the new $z_i$, we have a complete data problem, solve this problem for the parameters, i.e. compute $\Theta^{(i+1)}$
  4. go to 2.

- **in a graphical form**
Expectation-maximization

- mathematically we have

- expectation:

\[
h_{ij} = P_{Z|X}(e_j|x_i; \Psi^{(i)}) = \frac{G\left(x_i, \mu_j^{(i)}, \sigma_j^{(i)}\right) \pi_j^{(i)}}{\sum_{k=1}^{C} G\left(x_i, \mu_k^{(i)}, \sigma_k^{(i)}\right) \pi_k^{(i)}}
\]

- maximization:

\[
\mu_j^{(i+1)} = \frac{\sum_i h_{ij} x_i}{\sum_i h_{ij}} \quad \pi_j^{(i+1)} = \frac{1}{n} \sum_i h_{ij}
\]

\[
\sigma_j^{2(i+1)} = \frac{\sum_i h_{ij} (x_i - \mu_j)^2}{\sum_i h_{ij}}
\]
Expectation-maximization

- note that the difference to k-means is that
  - in E-step $h_{ij}$ would be thresholded to 0 or 1
  - this would make the M-step exactly the same
  - plus we get the estimate of the covariances and class probabilities automatically
  - k-means can be seen as a greedy version of EM
  - at each point we make the optimal decision
  - but this does not take into account other points or future iterations
  - if the hard assignment is best, EM can choose it too

- to get a feeling for EM you can use
  - [http://www-cse.ucsd.edu/users/ibayrakt/java/em/](http://www-cse.ucsd.edu/users/ibayrakt/java/em/)
Any Questions?