supervised learning framework

given pairs $(x_1, y_1), \ldots, (x_n, y_n)$ - input-output pairs; learn a function $f(x)$ that accurately predicts $y_i \approx f(x_i)$ on this data.
objective: use function $f(x)$ to predict new y_0 given x_0

probabilistically:

- (x,y) is a random variable with joint distribution $p(x,y)$
- supervised learning seeks to learn conditional distribution $p(y|x)$
- directly e.g. logistic regression $p(y|x) = \sigma(\cdot)$
- indirectly e.g. Bayes classifier $y = \underset{k}{\operatorname{argmax}}\ p(x|y=k)\,p(y=k)$

build each 'piece' and model with distribution

unsupervised learning - motivation, framework

motivation:

- Bayes classifier factorises joint density as $p(x,y) = p(x|y)p(y)$ i.e. as product of conditional probability of x given y multiplied by prior on y
- also, joint density $p(x,y) = p(y|x)p(x)$; joint likelihood $y|x$ multiplied by prior on x
- unsupervised learning focuses on term $p(x)$; ~~ler~~ estimating $p(x|y)$ on a class specific-subset has same feel.
estimating a distribution on data $p(x)$ that captures underlying qualities

unsupervised learning:

Given: A data set $x_1, \ldots, x_n$ where $x_i \in X$ e.g. $X \in \mathbb{R}^d$
Define: A model of the data (probabilistic or non-probabilistic) on x
Goal: learn/explore structure of the data set as defined by the model.
Points: ★ No clear performance metric / outputs to base metric on
★ More subjective

types of unsupervised learning

1) Clustering models
- Partition data into $x_1, \ldots, x_n$ groups
▶ image segmentation, data quantisation, pre-processing

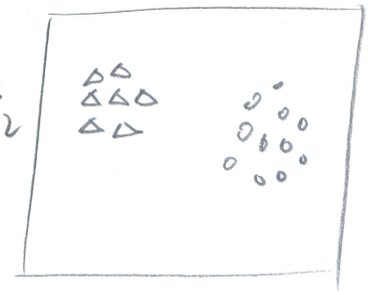
2) matrix factorisation
- estimate underlying dot-product representations
▶ user preference modelling, topic modelling
(movie prediction) (themes in text)

3) sequential models
- estimate model based on sequential information
▶ learn how to rank objects, target tracking

- Unsupervised models can be interpreted as a supervised model or indeed one
- Supervised learning methods have started informing unsupervised methods

Clustering

Problem :-
- Given data $x_1, \dots, x_n$; partition it into clusters
- Find clusters given data and modelling assumption
- Observations in same cluste → 'similar' (/different)
- Requires us to define similarity/difference
- Set how many clusters to use



- △/○ represents cluste assignment

cluste assignment representation

For K clusters, encode cluste assignments as an indicator c
& $c \in \{1, \dots, K\}$ (multi-class indicator)

$c_i = k \iff x_i$ is assigned to cluster k

Similarity to classification in that we 'label' an observation by its cluster assignment, except no ground truth (y_i)
c is a latent/auxiliary variable indexing cluste $\{1, \dots, K\}$ observation x_i belongs to.
Each cluste has label ; but we do not know labels (and we do not know clusters.

means clustering - setup, objective, optimisation

k-means is the simplest, most fundamental clustering algorithm
Input : $x_1, \dots, x_n$ $x \in \mathbb{R}^d$
Output : vector c of cluste assignments over n data points, with each $c_i \in \{1, \dots, K\}$
and K mean vectors μ

- $c = (c_1, \dots, c_n)$ $c_i \in \{1, \dots, K\}$
  - If $c_i = c_j = k$ then x_i and x_j are clustered together in cluste k
- $\mu = (\mu_1, \dots, \mu_K)$ $\mu_k \in \mathbb{R}^d$ (same space as x_i)
  - Each μ_k is called a centroid and defines a cluster.

Goal : Define an objective function ; choose one that
1. Tells us what are good c and μ
2. Easy to optimise

Find an assignment of data points to clusters $c = (c_1, \dots, c_n)$ and set of vectors $\{\mu_k\}$ (centroids)

Objective function:

- K-means objective function:-

$$\underline{\mu}^*, \underline{c}^* = \underset{\underline{\mu}, \underline{c}}{\operatorname{argmin}} \sum_{i=1}^{n} \sum_{k=1}^{K} \mathbb{1}\{c_i = k\} \|x_i - \mu_k\|_2^2$$

Comments

- squared Euclidean distance of x_i to centroid μ_k
- only penalises distance of x_i to the centroid its assigned by c_i

$$L = \sum_{i=1}^{n} \sum_{k=1}^{K} \mathbb{1}\{c_i = k\} \|x_i - \mu_k\|_2^2 = \sum_{k=1}^{K} \sum_{i : c_i = k} \|x_i - \mu_k\|^2 \quad (*)$$

- objective function L is non-convex (cannot find global optima $\underline{\mu}^*$ and $\underline{c}^*$); only drive an algorithm for local optimum
- sum over all data indexed by i; for each i, we sum over all clusters, ~~the~~ an indicator on whether $c_i = k$ i.e. x_i is in cluste k multiplied by a Euclidean squared distance of i^{th} data point to k^{th} cluster.
- Indicator $\mathbb{1}\{c_i = k\}$ selects the squared distance to the cluster that the i^{th} data point is actually assigned to: $0 \times \|x_i - \mu_k\|_2^2$ for $(K-1)$ clusters
  $1 \times \|x_i - \mu_k\|_2^2$ for 1 cluster (i^{th} point assigned to)

(*) For a particular assignment vector c_i, we sum for each value k, the total sum of squared distance of all the data assigned to k^{th} cluster

(*) can see L as aggregating over indexes sets $\{k\}$ which is K clusters or $\{i\}$ which are data points; with selection carried out by an indicator fn.
- which summation operator to aggregate over first defines how you think about it

Optimisation

- We can't optimise K-means objective function analytically ⇒ algorithm
- Gradient descent? i.e. $\underline{w} \leftarrow \underline{w} - \eta \nabla_w L$?
- i.e. update parameter "w" ~~we~~ by moving in direction that decreases the objective? 1) Will almost certainly not move to best value for that param (non-convex)
  2) If η too big ~~neg~~ (step size) may not move to better value
  3) Parameter w needs to be continuous valued: c_i is discrete i.e. $c_i \in \{1, \ldots, K\}$ to allow derivatives

K-means clustering - co-ordinate descent, assignment, update

- implement a new and widely used optimisation procedure in this context
- minimise over $\underline{c}$ i.e. all $\{c_1, \ldots, c_n\}$ and $\underline{\mu}$ i.e. all vectors $\{\mu_1, \ldots, \mu_K\}$ the loss

$$L = \sum_{i=1}^{n} \sum_{k=1}^{K} \mathbb{1}\{c_i = k\} \|x_i - \mu_k\|_2^2$$

- split variables one is optimising over into unknown sets $\underline{\mu}$ and $\underline{c}$
- Find optimal values $\underline{\mu}^*$ and $\underline{c}^*$ by fixing $\underline{\mu}$ to find best $\underline{c}$; then fix $\underline{c}$ to find best $\underline{\mu}$ (co-ordinate descent)
- Hold one set of parameters fixed and then optimise other set; then switch

co-ordinate descent in context of K-means:-

Input: $x_1, \dots, x_n$ where $x_i \in \mathbb{R}^d$. Randomly initialise $\underline{\mu} = (\mu_1, \dots, \mu_K)$

- Iterate back-and-forth between following steps:
  1. Given $\underline{\mu} = \bar{\underline{\mu}}$ (fixed), find best value $c_i \in \{1, \dots, K\}$ for $i = 1, \dots, n$
  2. Given $\underline{c} = \bar{\underline{c}}$ (fixed), find best vector $\mu_k \in \mathbb{R}^d$ for $k = 1, \dots, K$
- circularity involved due to dependence of $\underline{\mu}$ and $\underline{c}$ on each other in objective function L (frequent in unsupervised)
- Carry out until they 'equilibrate' / stop changing; not guaranteed (I guess)

Assignment step - updating $\underline{c}$

- Given $\bar{\underline{\mu}} = (\bar{\mu}_1, \dots, \bar{\mu}_K)$, update $\underline{c} = (c_1, \dots, c_n)$; rewrite L, noticing independence of each c_i given $\underline{\mu}$:
- Expanding summation over each data point (loss fn)

$$L = \sum_{i=1}^{n} \sum_{k=1}^{K} \mathbb{1}\{c_i = k\} \|x_i - \mu_k\|_2^2$$

$$= \left(\sum_{k=1}^{K} \mathbb{1}\{c_1 = k\} \|x_1 - \bar{\mu}_k\|_2^2 \right) + \dots + \left(\sum_{k=1}^{K} \mathbb{1}\{c_n = k\} \|x_n - \bar{\mu}_k\|_2^2 \right)$$

distance of x_1 to assigned centroid

— " — x_n — " —

- Solution:-

Assign x_i to the closest centroid;

$$c_i = \underset{k}{\operatorname{argmin}} \|x_i - \bar{\mu}_k\|_2^2$$

Comments

- Each bracket is independent: n independent problems, optimisation over c_i, given setting $\underline{\mu} = (\bar{\mu}_1, \dots, \bar{\mu}_K)$ fixed
- Setting c_i ~~as~~ no impact on c_n's value
- K options for each $c_i \Rightarrow$ no derivatives; given $\bar{\mu}_1, \dots, \bar{\mu}_K$, calculate all possible values for c_i and choose best

update step - updating μ

- Given $\underline{c} = (c_1, \dots, c_n)$, update $\mu = (\mu_1, \dots, \mu_K)$
- For a given $\underline{c}$ (~~i.e.~~), we can break $\mathcal{L}$ into K clusters defined by c so each μ_i is independent.

$$\mathcal{L} = \sum_{i=1}^{n} \sum_{k=1}^{K} \mathbb{1}\{c_i = k\} \|x_i - \mu_k\|_2^2$$

Expanding summation over K clusters

$$= \left(\sum_{i=1}^{n} \mathbb{1}\{c_i = 1\} \|x_i - \mu_1\|_2^2 \right) + \dots + \left(\sum_{i=1}^{n} \mathbb{1}\{c_i = K\} \|x_i - \mu_K\|_2^2 \right)$$

sum of squared distance of data in cluster #1

sum of squared distance of data in cluster #K

solution:-

- For each k, we optimise.
- Define $n_k = \sum_{i=1}^{n} \mathbb{1}\{c_i = k\}$
- Then $\mu_k = \underset{\mu}{\operatorname{argmin}} \sum_{i=1}^{n} \mathbb{1}\{c_i = k\} \|x_i - \mu\|_2^2 \longrightarrow \mu_k = \frac{1}{n_k} \sum_{i=1}^{n} x_i \mathbb{1}\{c_i = k\} = \frac{\sum_{i=1}^{n} x_i \mathbb{1}\{c_i = k\}}{\sum_{i=1}^{n} \mathbb{1}\{c_i = k\}}$

comments

- Each bracket is independent: there are K optimisation problems and updating μ_i is separate to the other values $\mu_2, \dots, \mu_n$ can take
- c_i is equal to one value out of the set $\{0, \dots, K\}$
- μ_k is the mean of the data assigned to cluster k.
  - → for a particular c_i, $\mathbb{1}\{c_i = k\} = 0$ for $K-1$ values and 1 for 1 value as x_i has already been assigned to a cluster at beginning of this round
- (☒) This was trickier to understand because of the indicator function

Example

- Assume there are 2 clusters ie. $K=2 \Rightarrow$ 2 centroids μ_1 and μ_2
- Assume 4 data points x_1, x_2, x_3, x_4
- Assignment $\underline{c} = (c_1, c_2, c_3, c_4) = (1, 2, 1, 1)$

$$\mathcal{L} = \left(\sum_{i=1}^{4} \mathbb{1}(c_i = 1) \|x_i - \mu_1\|_2^2 \right) + \left(\sum_{i=1}^{4} \mathbb{1}(c_i = 2) \|x_i - \mu_2\|_2^2 \right)$$

$$= \left(\|x_1 - \mu_1\|_2^2 + \|x_3 - \mu_1\|_2^2 + \|x_4 - \mu_1\|_2^2 \right) + \left(\|x_2 - \mu_2\|_2^2 \right)$$

SSD of data in cluster #1

SSD data in cluster #2

• 24
• To minimise L, we want to set μ_i so that the sum of squared distances of the data points selected by indicator function / assigned to that cluster is minimised. Intuitively we need to set μ_i accounting for the values of all the data points in that cluster; do this for all K clusters. So set μ_K = mean of the data assigned to cluster k.

• more formally, we can find the derivative of each bracket representing one of K optimisation problems and solve for a minimum μ_K

(24) • we want to minimise the sum of squared distance of data in cluster #K

SSD for cluster #K: $\left(\sum_{i=1}^{n} \mathbb{1}\{c_i = k\} \|x_i - \mu_K\|_2^2\right)$ (give a fixed cluster assignment $c = (\bar{c}_1, \bar{c}_2, \ldots, \bar{c}_n)$)

• Recall $\sum_{i=1}^{n} \mathbb{1}\{c_i = k\} \|x_i - \mu_K\|_2^2$ can be seen as $\sum_{i : c_i = k} \|x_i - \mu_K\|_2^2$

• Taking gradients wrt μ_K and setting to 0 vector :-

$$\nabla_{\mu_K} \left(\sum_{i=1}^{n} \mathbb{1}\{c_i = k\} \|x_i - \mu_K\|_2^2\right) = \underline{0}$$

$$\Rightarrow \nabla_{\mu_K} \left(\sum_{i=1}^{n} \mathbb{1}\{c_i = k\} \left(x_i^T x_i - 2 x_i \mu_K^T + \mu_K^T \mu_K\right)\right) = \underline{0}$$

$$\Rightarrow \sum_{i=1}^{n} \mathbb{1}\{c_i = k\} \nabla_{\mu_K} \left(x_i^T x_i - 2 x_i \mu_K^T + \mu_K^T \mu_K\right) = \underline{0}$$

$$\Rightarrow \sum_{i=1}^{n} \mathbb{1}\{c_i = k\} \left(-2 x_i + 2 \mu_K\right) = \underline{0}$$

$$\Rightarrow -2 \sum_{i=1}^{n} \mathbb{1}\{c_i = k\} x_i + 2 \sum_{i=1}^{n} \mathbb{1}\{c_i = k\} \mu_K = \underline{0}$$

$$\Rightarrow \mu_K = \frac{\sum_{i=1}^{n} x_i \mathbb{1}\{c_i = k\}}{\sum_{i=1}^{n} \mathbb{1}\{c_i = k\}} = \frac{1}{n_K} \sum_{i=1}^{n} x_i \mathbb{1}\{c_i = k\}$$ as required

with n_K - no. of data points assigned to cluster k

K-means

K-means clustering - algorithm, example

Algorithm:

Given: $x_1, x_2, x_3, \dots, x_n$ where each $x \in \mathbb{R}^d$

Goal: Minimise $L = \sum_{i=1}^{n} \sum_{k=1}^{K} \mathbb{1}\{c_i = k\} \|x_i - \mu_k\|_2^2$

- Randomly initialise $\underline{\mu} = (\mu_1, \dots, \mu_K)$
- Iterate until $\underline{c}$ and $\underline{\mu}$ stop changing:
  1. update each c_i: $c_i = \underset{k}{\operatorname{argmin}} \|x_i - \mu_k\|_2^2$
  2. update each μ_k: $\mu_k = \frac{1}{n_k} \sum_{i=1}^{n} x_i \mathbb{1}\{c_i = k\}$ where $n_k = \sum_{i=1}^{n} \mathbb{1}\{c_i = k\}$

Comments

1. set c_i to be index of cluster x_i is closest to in Euclidean sense
2. update cluster centroids μ_k by taking average of all data assigned to k^{th} centroid

Example

(see diagram)

- The mechanics of the diagram is good for visualising the iteration steps! ☺

Convergence of K-means algorithm

- see diagram showing convergence of the K-means algorithm after each assignment of $\underline{c}$ and update $\underline{\mu}$, showing decrease of objective L after iterations
- converges because each update to c_i and μ_k decreases L
- L is monotically decreasing in no. of iterations
- $L \geq 0$ due to non-negativity of squared l_2 norm → converges (but often not to 0)
- proof → (Q)

(??) When $\underline{c}$ and $\underline{\mu}$ stop changing: algorithm converges to a local optimum

- results from non-convexity of L in $\underline{c}$ and $\underline{\mu}$ simultaneously
- convexity ⇒ any initialisation of an optimisation procedure will converge to the same and the best solution
- non convexity ⇒ different initialisations will give different results

(??) In practice, run algorithms many times with different initialisations → select one with lowest L.

Selecting K

- selecting K is tricky

(??) - Kmeans objective function decreases as K increases

$L = \sum_{i=1}^{n} \sum_{k=1}^{K} \mathbb{1}\{c_i = k\} \|x_i - \mu_k\|_2^2$ → set $K = n$ and $\mu_k = x_k$ ⇒ $L = 0$ (minimum)

- Different settings of $K \Rightarrow$ different models ~~comparison ?~~
- Same setting of $K \Rightarrow$ can compare different runs of K-means objective

Methods for selecting K

- Advanced (domain) knowledge from outside the model
- Heuristic argument: Examine relative decrease in $\mathcal{L}$; if K^* is optimal increasing K when $K \leq K^*$ should decrease $\mathcal{L}$ much more than when $K > K^*$ (?)
- Part of a large application e.g. pre-processing; discretising continuous data
- Bayesian non-parametric methods

Applications of K-means

- Lossy data compression (see diagram)
- Image is 1024×1024 i.e. as matrix $X \in \mathbb{R}^{1024 \times 1024}$ with each value $x_{ij} \in [0, 255]$
- vectorise 2×2 patches i.e. $x \in \mathbb{R}^{2\times 2}$ so that data is $x \in \mathbb{R}^4$
- Learn K-means on the vectorised data in $\mathbb{R}^4$ e.g. $K = 200$, $K = 2$; replace each vectorised patch with its centroid μ_K.

Extension - K-medoids

Algorithm:

Input: Data $x_1, \ldots, x_n$ and distance measure $D(x, \mu)$. Randomly initialise μ

- Iterate until c is no longer changing

1. For each c_i: set $c_i = \underset{k}{\operatorname{argmin}}\, D(x_i, \mu_k)$
2. For each μ_K: set $\mu_K = \underset{\mu}{\operatorname{argmin}} \sum_{i: c_i = K} D(x_i, \mu)$

Comments

- Here x not necessarily in $\mathbb{R}^d$
- Replace squared L_2 norm/~~with~~ Euclidean distance with $D(x, \mu)$ in K-means
- e.g. $D(x, \mu) = \|x - \mu\|_1 \Rightarrow$ robust to outliers/sparse/LASSO; resistant to outlier values

1. Assign c_i equal to index of cluster x_i is closest to according to a distance metric
2. Update centroids possibly via another algorithm as no recourse to derivatives

- $x \notin \mathbb{R}^d \rightarrow D(x, \mu)$ more complex; e.g. if data is histogram of counts. (discrete)

L15

(*)

Approaches to data modelling

- Modelling data → probabilistic or non-probabilistic
- Probability distributions - defined on data or no probability distributions
- Probabilistic - e.g. Bayes classifiers, logistic regression, least squares and ridge (ML/MAP/mtlp) Bayesian LR
- Non-probabilistic - e.g. perceptron, SVM, decision trees, k-means
- Common to both approaches → an objective function being optimised
- There may be different forms of optimisation and corresponding types.

Maximum likelihood and co-ordinate ascent

- ML - probabilistic objective (I)
- setup involves 4 ingredients: 1) set of model parametes θ 2) set of data $\{x_1, \dots, x_n\}$ 3) Probability distri $p(x|\theta)$ 4) iid assumption $x_i \overset{iid}{\sim} p(x|\theta)$
- Maximum likelihood seeks to find θ that maximises likelihood:-

$$\theta_{ML} = \underset{\theta}{\operatorname{argmax}}\, p(x_1, \dots, x_n|\theta) \overset{(a)}{=} \underset{\theta}{\operatorname{argmax}} \hat{\prod_{i=1}} p(x_i|\theta) \overset{(b)}{=} \underset{\theta}{\operatorname{argmax}} \hat{\sum_{i=1}} \ln p(x_i|\theta)$$

(a) - iid (b) - $f(y) > f(x) \Rightarrow \ln f(y) > \ln f(x)$ and properties of argmax

- least squares LR and Bayes classifier involve analytic solutions under assumptions made
- If $x_i \overset{iid}{\sim} N(\mu, \Sigma)$ with $\theta = (\mu, \Sigma)$ then $\nabla_\theta \ln \hat{\prod_{i=1}} p(x_i|\theta) = 0 \Rightarrow \begin{cases} \mu_{ML} = \frac{1}{n} \hat{\sum_{i=1}} x_i \\ \Sigma_{ML} = \frac{1}{n} \hat{\sum_{i=1}} (x_i - \mu_{ML})(x_i - \mu_{ML})^T \end{cases}$
- More complicated situation (II):-
- Split parametes into two groups θ_1 and θ_2; try to maximise likelihood ove both:-

$$\theta_{1,ML}, \theta_{2,ML} = \underset{\theta_1, \theta_2}{\operatorname{argmax}} \hat{\sum_{i=1}} \ln p(x_i|\theta_1, \theta_2)$$

- We can solve one, given another, but not 'simultaneously' analytically and in abstract

Co-ordinate ascent:-

- For iteration $t = 1, 2, \dots$
  1. optimise $\theta_1^{(t)} = \operatorname{argmax}_{\theta_1} \hat{\sum_{i=1}} \ln p(x_i|\theta_1, \theta_2^{(t-1)})$; i.e. fixing $\theta_2 = \overline{\theta_2^{(t-1)}}$
  2. optimise $\theta_2^{(t)} = \operatorname{argmax}_{\theta_2} \hat{\sum_{i=1}} \ln p(x_i|\theta_1^{(t)}, \theta_2)$; i.e. fixing $\theta_1 = \overline{\theta_1^{(t)}}$
- Will converge to a local optimum if non-convex

Motivation for expectation-maximisation (EM) algorithm

- We want to find:-

$$\theta_{1,ML} = \underset{\theta_1}{\operatorname{argmax}} \hat{\sum_{i=1}} \ln p(x_i|\theta_1)$$

- (SI) but we cannot optimise this directly, instead we add a second variable and work with: (OII): $\hat{\sum_{i=1}} \ln p(x_i, \theta_2|\theta_1)$

- θ_2 on left-side of conditioning $\Rightarrow$ prior on θ_2
- θ_2 is a 'latent' variable; essentially finding a marginal distribution on observed x_i; via integrating over a joint distribution over observed x_i and latent θ_2
- EM algorithm finds $\theta_{1,ML}$ using above method
- In context of missing and observed data
- Let $x_i \in \mathbb{R}^d$ be a vector with missing data; split into 2 parts:-
  1. x_i^o - observed portion (sub-vector of x_i that is measured)
  2. x_i^m - missing portion (——— " ——— that is still unknown)
  3. missing dimensions can be different for different x_i
- (07) Assume $x_i \overset{iid}{\sim} N(\mu, \Sigma)$ and we want to solve

$$\mu_{ML}, \Sigma_{ML} = \underset{\mu,\Sigma}{\operatorname{argmax}} \sum_{i=1}^{n} \ln p(x_i^o | \mu, \Sigma)$$

- This is difficult; in particular as each x_i^o has different dimensionality; and each dimension can correspond to different subsets of dimensions in μ and Σ.
- However; if we know x_i^m (and therefore x_i) then following optimisation is tractable:- (?) (*)

$$\mu_{ML}, \Sigma_{ML} = \underset{\mu,\Sigma}{\operatorname{argmax}} \sum_{i=1}^{n} \ln \underbrace{p(x_i^o, x_i^m | \mu, \Sigma)}_{= p(x_i | \mu, \Sigma)}$$

EM algorithm - setup, objective function

(07) - A method for optimising $\sum_{i=1}^{n} \ln p(x_i^o | \mu, \Sigma)$ and imputing missing values $\{x_i^m, \dots, x_n^m\}$, a very general technique:-

setup:

- Imagine we have two parameter sets θ_1 and θ_2 and that:

$$p(x|\theta_1) = \int p(x, \theta_2 | \theta_1) \, d\theta_2 \quad \text{(marginal distribution)}$$

- In context of earlier example:-

$$p(x_i^o | \mu, \Sigma) = \int p(x_i^o, x_i^m | \mu, \Sigma) \, dx_i^m = N(\mu_i^o, \Sigma_i^o) \quad \text{(useful derivation)}$$

where μ_i^o - subvector of μ
Σ_i^o - submatrix of Σ
} defined by x_i^o

- We need to define a general objective function that fulfils following criteria:-
  i) lets us optimise marginal $p(x|\theta_1)$ over θ_1
  ii) uses $p(x, \theta_2 | \theta_1)$ purely for computational convenience.

Objective function

The objective function is :-

$$\ln p(x|\theta_1) = \int q(\theta_2) \ln \frac{p(x,\theta_2|\theta_1)}{q(\theta_2)} d\theta_2 + \int q(\theta_2) \ln \frac{q(\theta_2)}{p(\theta_2|x,\theta_1)} d\theta_2$$

(i) (ii) (iii)

Comments

i) $\ln p(x,\theta_2|\theta_1)$ - Joint likelihood of data (x) and additional variable θ_2

ii) $q(\theta_2)$ - probability distribution over additional variable; can in principle be anything but EM gives specific prescriptions on this

iii) $p(\theta_2|x,\theta_1)$ - conditional posterior distribution of additional variable θ_2 given data (x) and fixed $\hat{\theta}_1$; assumed we know/can calculate this; sometimes in non-Bayesian settings merely called the conditional distribution of θ_2 given x and current estimates of parameter θ_1.

Derivation: Showing that this equality holds

$$\ln p(x|\theta_1) = \int q(\theta_2) \ln \frac{p(x,\theta_2|\theta_1)}{q(\theta_2)} d\theta_2 + \int q(\theta_2) \ln \frac{q(\theta_2)}{p(\theta_2|x,\theta_1)} d\theta_2$$

$$= \int q(\theta_2) \ln \frac{p(x,\theta_2|\theta_1) q(\theta_2)}{p(\theta_2|x,\theta_1) q(\theta_2)} d\theta_2$$

$$= \int q(\theta_2) \ln p(x|\theta_1) d\theta_2$$

$$= \ln p(x|\theta_1)$$

- via the 'distributive' property of operator $\int (a+b)dx = \int a\,dx + \int b\,dx$
- $p(a,b|c) = p(a|b,c)\,p(b|c)$
  $\Rightarrow p(b|c) = \frac{p(a,b|c)}{p(a|b,c)}$
- $a = \theta_2$
  $b = x$ $c = \theta_1$; as $\ln p(x|\theta_1)$ does not contain θ_2 and $\int q(\theta_2) d\theta_2 = 1$ (unity of prob.)

More comments

$$\ln p(x|\theta_1) = \int q(\theta_2) \ln \frac{p(x,\theta_2|\theta_1)}{q(\theta_2)} d\theta_2 + \int q(\theta_2) \ln \frac{q(\theta_2)}{p(\theta_2|x,\theta_1)} d\theta_2$$

(i) (ii)

(i): $\mathcal{L}$: Assuming integral in $\mathcal{L}$ can be calculated/evaluated, rendering this merely a function of θ_1 (and the data x). i.e. we can integrate out additional variable θ_2. It is for a particular setting of the distribution $q(\theta_2)$.

(ii) Kullback-Liebler divergence: Always ≥ 0 (non-trivial proof) with equality when $q = p$ i.e. distributions are the same. Is akin to a 'distance' measure in a non-strict sense between 2 probability distributions. More overlap $\Rightarrow$ KL ↓
less overlap $\Rightarrow$ KL ↑

EM Algorithm

Q: what does it mean to iteratively optimise $\ln p(x|\theta_1)$ wrt θ_1?

A: We want an algorithm (i.e. an effectively calculable procedure) for generating:-

1. A sequence of values for θ_1 such that $\ln p(x|\theta_1^{(t)}) \geq \ln p(x|\theta_1^{(t-1)})$
2. We want $\theta_1^{(t)}$ to converge to a <u>local maximum</u> of $\ln p(x|\theta_1)$

- Doesn't matter how we generate sequence $\{\theta_1^{(t)}\} = \{\theta_1^{(1)}, \theta_1^{(2)}, \theta_1^{(3)}, \ldots, \theta_1^{(t)}\}$
- Lecture shows how to generate #1; #2 is stated to be satisfied, but further development requires investigating convex/non-convex optimisation.

EM objective:-

$$\ln p(x|\theta_1) = \underbrace{\int q(\theta_2) \ln \frac{p(x,\theta_2|\theta_1)}{q(\theta_2)} d\theta_2}_{\mathcal{L}(x,\theta_1)} + \underbrace{\int q(\theta_2) \ln \frac{q(\theta_2)}{p(\theta_2|x,\theta_1)} d\theta_2}_{\text{KL divergence}}$$

definition: EM algorithm

Given value for $\theta_1^{(t)}$, find the value for $\theta_1^{(t+1)}$ as follows:- (t: indexes time step ~~iteration~~)

E-step: Set $q_t(\theta_2) = p(\theta_2|x,\theta_1^{(t)})$ and calculate:- (i)

$$\mathcal{L}_t(x,\theta_1) = \int q_t(\theta_2) \ln \frac{p(x,\theta_2|\theta_1)}{q_t(\theta_2)} d\theta_2 \quad \text{(ii)}$$

$$= \int q_t(\theta_2) \ln p(x,\theta_2|\theta_1) d\theta_2 - \underbrace{\int q_t(\theta_2) \ln q_t(\theta_2) d\theta_2}_{\text{ignored}} \quad \text{(iii)}$$

(iv)

M-step: Set $\theta_1^{(t+1)} = \underset{\theta_1}{\operatorname{argmax}}\, \mathcal{L}_t(x,\theta_1)$

Comments

- Index $q(\theta_2)$ and $\mathcal{L}(x,\theta_1)$ by time-step t, each iteration $t \to t+1$

(i) At time step t, set $q_t(\theta_2)$ to be equal to the conditional posterior of θ_2 (additional variable); given the data$^{(x)}$ and current estimates of parametes $\theta_1^{(t)}$; in non-Bayesian framework, this is conditional distribution.

(ii) Take the expected (complete-data) joint$_{\wedge}^{\log}$ likelihood of x and latent variable θ_2 under the posterior of latent variable θ_2; $q_2 = p(\theta_2|x,\theta_t^{(1)})$ ie. using q-distribution - integrating out θ_2

(iii) $\int q_t(\theta_2) \ln q_t(\theta_2) d\theta_2$ is constant - ignore

(iv) M-step:- Now we are free to vary θ_1 and it is no longer fixed at $\theta_1 = \theta_1^{(t)}$,

set $\theta_1^{(t+1)} = \underset{\theta_1}{\text{argmax}}\, \mathcal{L}_t(x, \theta_1)$ i.e. set new parameters $\theta_1^{(t+1)}$ to maximise

$\mathcal{L}_t(x, \theta_1)$ where θ_1 is free to vary

EM algorithm - proof of monotonic improvement, illustration

- We want to show $\ln p(x|\theta_1^{(t)}) \leq \ln p(x|\theta^{(t+1)})$
- Essentially, each EM update over 1 iteration increases the (incomplete-data) log likelihood.

★ Derivation:-

$$\ln p(x|\theta_1^{(t)}) = \mathcal{L}(x, \theta_1^{(t)}) + KL(q(\theta_2) \| p(\theta_2|x_1, \theta_1^{(t)})$$ → this equality holds regardless of of iteration

$$= \mathcal{L}_t(x, \theta_1^{(t)})$$

- E step:- set $q_t(\theta_2) = p(\theta_2|x, \theta_1^{(t)})$ calculate $\mathcal{L}(x, \theta_1^{(t)})$ using q_t ⇒ $\mathcal{L}_t(x, \theta_1^{(t)})$, i.e. $\mathcal{L}$ evaluated with q_t

$$\leq \mathcal{L}_t(x, \theta_1^{(t+1)})$$

- M-step: Allow θ_1 to vary; we ~~do not~~ use same $q_t(\theta_2)$ (posterior conditional) but do not enforce evaluation at $\theta_1^{(t)}$, i.e. θ_1 is no longer fixed at $\theta_1^{(t)}$. ∴ $\mathcal{L}_t(x, \theta_1^{(t+1)}) \geq \mathcal{L}_t(x, \theta_1^{(t)})$ via maximisation

$$\leq \mathcal{L}_t(x, \theta_1^{(t+1)}) + KL(q_t(\theta_2) \| p(\theta_2|x_1, \theta_1^{(t+1)})$$

> 0 as $q \neq p$

- Can arbitrarily add a non-negative constant to RHS without altering inequality relation. Choose to add KL divergence between $q_t(\theta_2)$ and posterior conditional $p(\theta_2|x, \theta_1^{(t+1)})$ evaluated with updated parameter estimate $\theta_1^{(t+1)}$, $q \neq p$ assuming change in value of $\theta_1^{(t)}$ to $\theta_1^{(t+1)}$

$$\leq \mathcal{L}(x, \theta_1^{(t+1)}) + KL(q(\theta_2) \| p(\theta_2|x_1, \theta_1^{(t+1)})$$

- Now let q_t vary again, remove subscripts

$$\leq \ln p(x|\theta^{(t+1)})$$

Illustration:-

(see/draw diagram)

- Start: current setting of θ_1 and $q(\theta_2)$
- E-step: set $q(\theta_2) = p(\theta_2|x, \theta_1)$ and update $\mathcal{L}$, setting $KL = 0 \Rightarrow \ln p(x|\theta_1) = \mathcal{L}$
- M-step: Maximise $\mathcal{L}$ wrt $\theta_1 \Rightarrow q \neq p \Rightarrow KL > 0$ and $\mathcal{L} \uparrow$
- $\Delta \ln p(x|\theta_1) = \Delta \mathcal{L}$ and ΔKL
- Reference:- $\ln p(x|\theta_1) = \mathcal{L} + KL$ $\quad \mathcal{L} = \int q(\theta_2) \ln \frac{p(x, \theta_2|\theta_1)}{q(\theta_2)} d\theta_2$ $\quad KL = \int q(\theta_2) \ln \frac{q(\theta_2)}{p(\theta_2|x, \theta_1)} d\theta_2$

Missing data - problem, EM algorithm application, implementation

Problem:

(see diagram and draw)

- data matrix with missing entries; model columns as $x_i \sim N(\mu, \Sigma)$ (iid)
- **Goal:-**
  1. Learn μ and Σ using maximum likelihood
  2. Fill in missing values intelligently
  3. Both using EM algorithm
- **Strategy**
  1. maximise log-likelihood only considering observed points x_i^o over unknown parameters μ and Σ
  2. use q-distribution $q(x_i^m)$ (conditional posterior on missing data points x_i^m $p(x_i^m | x, \theta)$ given data, and parameter estimates $\Rightarrow$ probabilistic statements about conjectured values for x_i^m.

EM algorithm application - single Gaussian with missing data

Setup:-

Generic EM objective:-

$$\ln p(x|\theta_1) = \int q(\theta_2) \ln \frac{p(x, \theta_2 | \theta_1)}{q(\theta_2)} d\theta_2 + \int q(\theta_2) \ln \frac{q(\theta_2)}{p(\theta_2 | x, \theta_1)} d\theta_2$$

contextualised EM objective:- i.e. set $x = x_i^o$, $\theta_1 = \mu, \Sigma$, $\theta_2 = x_i^m$ and consider $\sum_{i=1}^{n}$ (over n data points)

$$\sum_{i=1}^{n} \ln p(x_i^o | \theta_1) = \sum_{i=1}^{n} \int q(x_i^m) \ln \frac{p(x_i^o, x_i^m | \mu, \Sigma)}{q(x_i^m)} dx_i^m + \sum_{i=1}^{n} \int q(x_i^m) \ln \frac{q(x_i^m)}{p(x_i^m | x_i^o, \mu, \Sigma)} dx_i^m$$

- All can be calculated. $\sum_{i=1}^{n} \ln p(x_i^o | \theta_1)$ is the incomplete-data log likelihood
- $\sum_{i=1}^{n} \ln p(x_i^o, x_i^m | \mu, \Sigma)$ is complete-data log likelihood

E-step pt. I: setting $q(x_i^m)$

- set $q(x_i^m) = p(x_i^m | x_i^o, \mu, \Sigma)$ using current μ, Σ
- Let x_i^o and x_i^m be observed and missing dimensions of x_i respectively

$$x_i = \begin{bmatrix} x_i^o \\ x_i^m \end{bmatrix} \sim N\left(\begin{bmatrix} \mu_i^o \\ \mu_i^m \end{bmatrix}, \begin{bmatrix} \Sigma_i^{oo} & \Sigma_i^{om} \\ \Sigma_i^{mo} & \Sigma_i^{mm} \end{bmatrix} \right)$$

- We can show via standard MVG calculations not derived on this course that

$$p(x_i^m | x_i^o, \mu, \Sigma) = N(\hat{\mu}_i, \hat{\Sigma}_i) \text{ with } \hat{\mu}_i = \mu_i^m + \Sigma_i^{mo} (\Sigma_i^{oo})^{-1} (x_i^o - \mu_i^o)$$

$$\hat{\Sigma}_i = \Sigma_i^{mm} - \Sigma_i^{mo} (\Sigma_i^{oo})^{-1} \Sigma_i^{om}$$

- Key point is that we can calculate $\hat{\mu}_i$ and $\hat{\Sigma}_i$ for each x_i^m from μ, Σ to get $p(x_i^m | x_i^o, \mu, \Sigma)$

- we have only showed this for one particular value of x_i^m i; missingness will be different for different i.

E-step pt II - evaluating $\mathbb{E}_{q(x_i^m)}[\ln p(x_i^o, x_i^m | \mu, \Sigma)]$

- In words, we evaluate the expected joint log-likelihood of the complete dataset under the distribution q; which is the conditional (posterior) distribution of missing data x_i^m given observed data x_i^o and current parameter estimates μ, Σ.
- For each i, we need to calculate (without full derivations):

$$\mathbb{E}_q[(x_i-\mu)^T\Sigma^{-1}(x_i-\mu)] = \mathbb{E}_q[\text{trace}\{\Sigma^{-1}(x_i-\mu)(x_i-\mu)^T\}]$$

$$= \text{trace}\{\Sigma^{-1}\underbrace{\mathbb{E}_q[(x_i-\mu)(x_i-\mu)^T]}_{(A)}\}$$

(A) is calculated by integrating with respect to the distribution $q(x_i^m) = p(x_i^m | x_i^o, \mu, \Sigma)$; only x_i^m portion of the complete vector x_i will be integrated.

Recall $q(x_i^m) = N(\hat{\mu}_i, \hat{\Sigma}_i)$

- Define:-
  1. $\hat{x}_i$ - vector where we replace missing values in x_i with $\hat{\mu}_i$
  2. $\hat{V}_i$ - matrix of 0s plus sub-matrix $\hat{\Sigma}_i$ in missing dimensions.

M-step: maximise $\sum_{i=1}^{n} \mathbb{E}_q[\ln p(x_i^o, x_i^m | \mu, \Sigma)]$ by updating μ, Σ

- Derivation omitted
- Note that we have 'calculated' $\mathbb{E}_{q(x_i^m)}[\ln p(x_i^o, x_i^m | \mu, \Sigma)]$; even though this isn't explicitly shown.
- Now

$$\mu_{up}, \Sigma_{up} = \underset{\mu,\Sigma}{\text{argmax}} \sum_{i=1}^{n} \mathbb{E}_q[\ln p(x_i^o, x_i^m | \mu, \Sigma)]$$

and we find:

$$\mu_{up} = \frac{1}{n}\sum_{i=1}^{n} \hat{x}_i$$

$$\Sigma_{up} = \frac{1}{n}\sum_{i=1}^{n} \{(\hat{x}_i - \mu_{up})(\hat{x}_i - \mu_{up})^T + \hat{V}_i\}$$

Then return to E-step to calculate new $p(x_i^m | x_i^o, \mu_{up}, \Sigma_{up})$

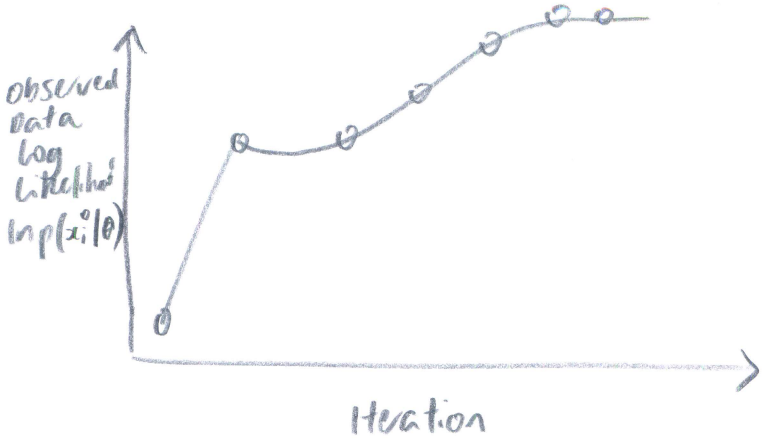
- Note that μ_{up} and Σ_{up} are similar to μ_{ML} and Σ_{ML}:

  μ_{up} - replace x_i in sample mean to with $\hat{x}_i$ (which is complete data vector imputed with missing values $\hat{\mu}_i$)

  Σ_{up} - replace x_i with $\hat{x}_i$ (as above); μ with μ_{up} (as above) and add an additional $\hat{V}_i$ corresponding to matrix of 0s with sub-matrix $\hat{\Sigma}_i$ in missing dimensions.

Implementation

- Each EM update results in a monotonic increase in the log-likelihood of the observed i.e. incomplete data set $\ln p(x_i^o|\theta)$



- Small marginal improvement $\Rightarrow$ in $\ln p(x_i^o|\theta) \Rightarrow$ algorithm converged.
- Initialise μ and Σ e.g. setting missing values $x_i^m = 0$ and using initial μ_{ML}, Σ_{ML}; or random initialisation.
- EM objective calculated after each update μ and Σ looks like above

(EM): Output μ_{ML} and Σ_{ML} that maximise $\sum_{i=1}^{\wedge} \ln p(x_i^o|\mu,\Sigma)$ and a conditional posterior $q(x_i^m) = p(x_i^m|x_i^o, \mu_{ML}, \Sigma_{ML})$ on missing dimensions $\Rightarrow$ which gives mean and uncertainty of missing data; i.e. a probability distribution that tells us what x_i^m might be. From this we can impute $\{x_1^m, \ldots x_n^m\}$.

// . Tricky harmonising this with other presch. — 1. Notation
2. Likelihood vs probability
3. "Assume we know x_i^m" ~~!!~~

#

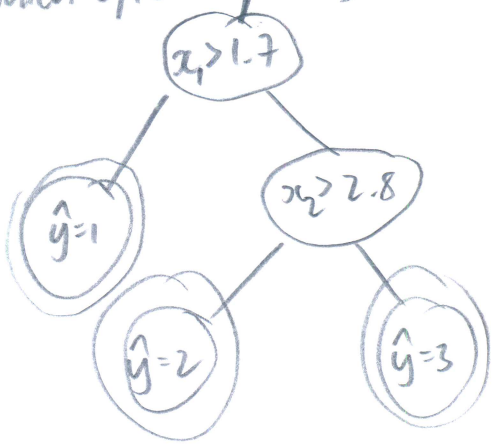
L12

Decision tree (binary)

- A decision tree maps an input $x \in \mathbb{R}^d$ to an output y using binary decision rules
- Note output may be $y \in \mathbb{R}$ (regression); $y \in \{-1, +1\}$ (binary classification) or $y \in \{1, \dots, K\}$ (multi-class classification)
- Each node in the tree contains a splitting rule
- each leaf node is associated with an output value (which often repeats)
- each splitting rule takes the form:-

$$h(x) = \mathbb{1}\{x_{(j)} > t\}$$ for some dimension j of x and some $t \in \mathbb{R}$

- using these transition rules, a path to a leaf node gives a prediction.
- decision tree represents a series of (conditional) IF statements
- 1-level tree - decision stump

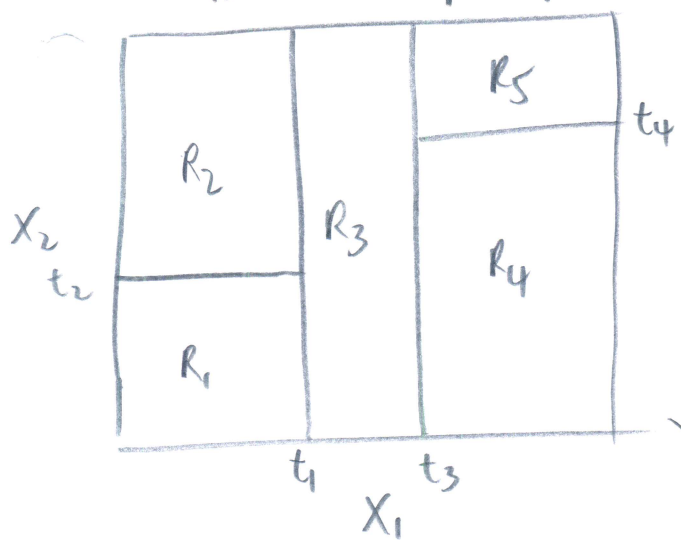


$y \in \{1, 2, 3\}$

Regression tree

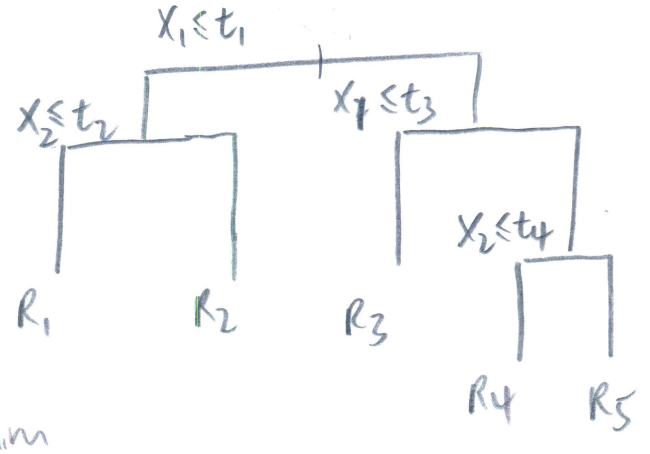
- For a feature/input space X, we want to partition the space so that the data in a region have the same prediction.
- some partitionings of input spaces cannot be achieved via binary decision trees (see diagram) and a recursive splitting rule
- We are concerned with partitions that can be achieved in context of above.

Partitioned input space

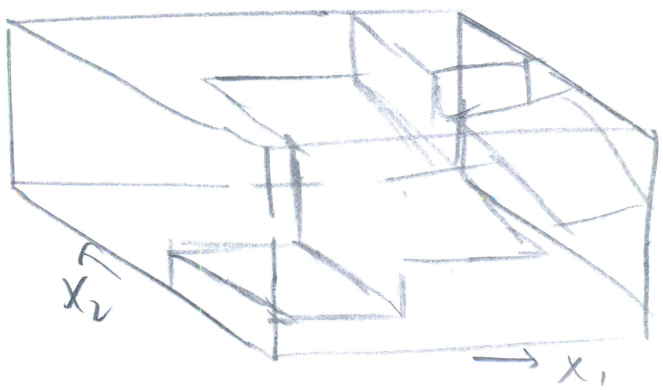


add output dim

Decision tree



- Regression trees learn a step function approximation to data.



Classification trees

- classify irises using sepal/petal widths
- $x \in \mathbb{R}^2$ $y \in \{1, 2, 3\}$ corresponding to 3 types of irises
- perform feature extraction, informed by expert ad apply 'pre-processing'
- Let x_1 = ratio of sepal length to width
  x_2 = " petal —— " ——

(1) check you understand the mechanics of decision stump - 1-decision tree - 2 decision tree
- key idea is most prevalent class, how will a region is already classified?

Basic decision tree algorithm (for classification and regression)

Basic method for "learning" trees is a top-down greedy algorithm

(2) 1) start with a leaf node containing all data

loop
- pick the leaf node to split that reduces uncertainty the most } start with node, split into two children via decision. choose which leaf node to split
- figure out a decision rule on one of the dimensions

2) stopping rule - later

label/response of the leaf is majority-vote/average of data assigned to it.

owing a regression tree

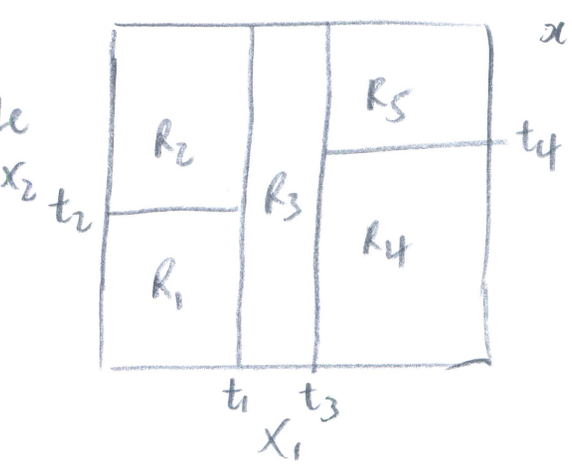
How do we grow a regression tree?

For M regions/partitions of space $R_1, \dots, R_M$ the prediction function is :-

) $f(x) = \sum_{m=1}^{M} c_m \mathbb{1}\{x \in R_m\}$

(2)-(2) (1)
r fixed M, we require R_m and c_m

oal: minimise $\sum_i (y_i - f(x_i))^2$ (RSS)



$x \in \mathbb{R}^2$
$y \in \mathbb{R}$

Note for clarity $f(x) = \sum_{m=1}^{M} c_m \mathbb{1}\{x \in R_m\} = c_1 \mathbb{1}\{x \in R_1\} + c_2 \mathbb{1}\{x \in R_2\} + \dots + c_M \mathbb{1}\{x \in R_M\}$

for each x_i (data point), $f(x_i) = \sum_{m=1}^{M} c_m \mathbb{1}\{x_i \in R_m\}$ will return a prediction c_m based which region x_i inhabits (summed over all regions)

Find c_m given R_m so that RSS is minimised:
- minimising RSS $\sum_i (y_i - f(x_i))^2$ conditioned on partition being considered breaks down into m different problems, depending on region being considered
- Finding c_m for corresponding R_m, we are summing the squared residuals for all the data restricted to that particular region R_m

(on) optimal c_m is just the average of corresponding outputs y_i for which $x_i \in R_m$

2. Finding R_m

- Define $R^-(j,s) = \{x_i \in R \mid x_i(j) \leq s\}$ and $R^+(j,s) = \{x_i \in R \mid x_i(j) > s\}$
- For each dimension j, calculate splitting point s for that dimension
- Do this for each region (leaf node). Pick the one that reduces objective the most

(*)

Comments

i) Brute force method/exhaustive search

ii) Some narrative:- conside splitting a particular region R (may be entire data set) at/along dimension j at a particular value s. Eg in figure R_1; look at $j=1$ or 2 and split at a particular value s along those two dimensions; then we have a proposed j and proposed s for a particular region R. Then ask, how does this partition the data in region R. Construct set R^- (all of data in region R with value of dim j less than s) and R^+ which is the resulting partition putting data in negative or positive group depending on value of j^{th} dim and its comparison with s.

(on) Update proposed predictions c_m i.e. proposed value of C had we made that split and examine how much it reduces resulting objective.

- Do this for every single region, every single dimension in that region, and every possible splitting point s that would lead to a unique partition
- For each of these proposals we get a new possible regression function f and we evaluate SSR for each of them.

Growing classification trees - quality measures

- Regression trees use sum of squared residuals to define splitting rule
- For classification, we need a measure of how badly a region classifies data and how much it can improve if split (no natural performance measure)

K-class problem :- $y \in \{1, \ldots, K\}$ For all $x \in R_m$, let p_k be empirical fraction labelled k in the region R_m i.e. construct an empirical distribution on labels (y) for data in that region

To measure quality of prediction of R_m :-

(on) Prediction $\hat{y}$ for a region is just most prevalent label

1. Classification error = $1 - \max_k p_k$
2. Gini index = $1 - \sum_k p_k^2$
3. Entropy = $-\sum_k p_k \ln p_k$

· All measures of R_m maximised with p_k uniform over label classes K in R_m

· — " — minimised with $p_k = 1$ for some k (R_m only contains one class)

· See diagram

Growing classification tree example:-

(see diagram) ② which region to split, along which dim j, and where to split s or (t)

· search R_1 and R_2 for splitting options (j,s) $y \in \{1,2,3\}$

$u(R_1)=0$

R_1: $y=1$: leaf classifies perfectly (all of points on that region are class 1 so $p_k=1$ for class 1 in that R_2)

R_2: $y=3$: leaf has Gini index

$$u(R_2) = 1 - \left(\frac{1}{101}\right)^2 - \left(\frac{50}{101}\right)^2 - \left(\frac{50}{101}\right)^2$$

$$= 0.5098$$

· R_2 has Gini coefficient of 0.5098

· $1 - \sum_k$ fraction of that class k in region R_2

Gini) Improvement from split R_m to R_m^- and R_m^+:

$$u(R_m) - \left(p_{R_m^-} \cdot u(R_m^-) + p_{R_m^+} \cdot u(R_m^+)\right)$$

$p_{R_m^+}$ - fraction of data in R_m split into R_m^+

$u(R_m^+)$: new quality measure on R_m^+

which region R_m to split, along which dim j, where to split s

the diagram, we have chosen to split R_2 as the region as R_1 had $u(R_1)=0$;

each dimension $j=1$ ~~and 2~~; we evaluate the reduction in uncertainty (new

solute value) over all splitting points $t \in (1.7, 3)$ corresponding to rule

$\{x_1 > t\}$ for $j=1$

dimension $j=2$, we evaluate reduction in uncertainty over all

tting points $t \in [2,6]$ corresponding to rule $\mathbb{1}\{x_2 > t\}$ for $j=2$

$j=1$, ~~not~~ RID is maximised at $t=2.25$, leading to RID of ≈ 0.018;

esponding classification is $\mathbb{1}\{x_1 > 2.25\}$

$=2$, RID is maximised at $t=2.8$, leading to RID of ≈ 0.2; corres (*)

sification $\mathbb{1}\{x_2 > 2.8\}$

efore select (j,t) to be $(2, 2.8)$ as RID is greate

iew this as 1. evaluate $u(R_m)$ for M regions and select the region R_m to partition with highest $u(R_m)$

2. Given a selected region R_m, choose a pair (j,t) from a possible (j,t)s. Each (j,t) defines a corresponding $R^-(j,t)$ and $R^+(j,t)$. Select (j,t) to maximise $\left(p_{R_m^-} \cdot u(R_m^-) + p_{R_m^+} \cdot u(R_m^+)\right)$

be of

and evaluating $(p_{R_m^-} \cdot u(R_m^-) + p_{R_m^+} u(R_m^+))$ over all t for $j = \bar{j}$; then selecting the highest over possible js.

Stopping rules - pruning

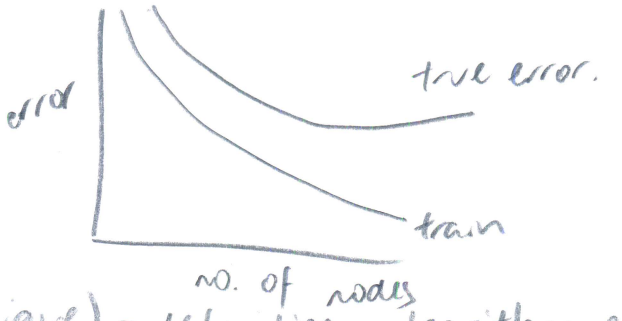
(Q) when should we stop growing a tree? (stopping rule)

(A) uncertainty reduction not best way

- Example shows any split of x_1 or $x_2 \Rightarrow$ zero reduction in uncertainty
- But perfect tree on data can be learnt by partitioning in quadrants (all splits made → same amount of class 1 and 2 on both splits)

Pruning often use pruning ie. grow tree to a large size and use algorithm to trim it back (non-trivial algorithm) → (9)

Overfitting issues :



Bootstrap (resampling technique) - definition, algorithm, example

Bootstrap is a statistical technique used as the basis for ensemble classifiers
t 'improves' estimators
esampling - sampling from an empirical distribution of data
plied to ensemble methods; resampling generates many mediocre classifiers
→ bootstrap / subsamples of the data
ey are then bagged to improve performance

rithm:-

ut: A sample of data $(x_1, \ldots, x_n)$ drawn from an unknown underlying distribution
An estimation rule/estimator $\hat{S}$ of a statistic S eg.
e.g. $\hat{S} = \text{med}(x_{1:n})$ estimates the true (population) median S of the unknown distribution on x.

rithm: 1. Generate B lots of bootstrap samples $B_1, \ldots B_B$
- Create B_b by picking points from sample data $\{x_1, \ldots, x_n\}$ randomly n times with replacement
- A particular x_i can appear in B_b many times or be omitted

2. Evaluate estimator on each B_b by pretending it is the dataset

$$\hat{S}_b := \hat{S}(\mathcal{B}_b)$$

3. Estimate the mean and variance of the estimator $\hat{S}$ by 'averaging' over the B bootstrap estimators $\{\hat{S}_1, \dots, \hat{S}_B\}$ to find:-

$$\mu_B = \frac{1}{B}\sum_{b=1}^{B} \hat{S}_b \qquad \sigma_B^2 = \frac{1}{B}\sum_{b=1}^{B} (\hat{S}_b - \mu_B)^2$$

Bootstrap example - variance estimation of median

- The median of $x_1, \dots, x_n$ for $x \in \mathbb{R}$ is found by sorting from smallest to highest and taking middle or average of 2 middle ones.

Q: How can we be confident can we be in the estimate median$(x_1, \dots, x_n)$

- Find variance
- BUT only one possible value for estimate of median, given our original sample.

A: 1. Generate B, lots of bootstrap data sets $\mathcal{B}_1, \dots, \mathcal{B}_B$

2a. calculate median$(\mathcal{B}_b)$ for all B bootstrap data sets B times.

2b. Calculate :-

$$\hat{S}_{mean} = \frac{1}{B}\sum_{b=1}^{B} \text{median}(\mathcal{B}_b) \qquad \hat{S}_{var} = \frac{1}{B}\sum_{b=1}^{B} \left(\text{median}(\mathcal{B}_b) - \hat{S}_{mean}\right)^2$$

in this case

- Here $S = \text{med}(\cdot)$ and $\hat{S}_{mean}$ is the mean of the estimator, which is the median.

- Lots of theory behind it ⇒ see Wasserman

Bagging (bootstrap aggregation) - motivation, algorithm, example

- Bagging stands for bootstrap aggregation; uses bootstrap for regression and class

Algorithm

For $b = 1, \dots, B$:-

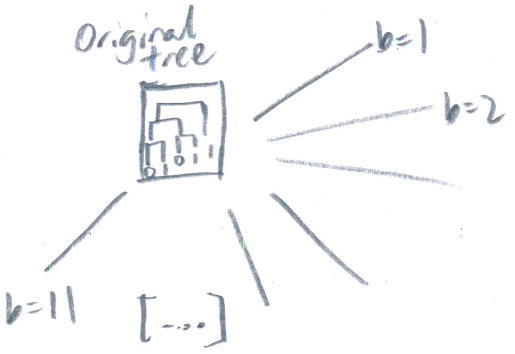
1. Draw a bootstrap sample of size n from training data
2. Train a classifier or regression model f_b on bootstrap data sub-sample $\mathcal{B}_b$, yielding B regression models $f_1, \dots, f_B$

- For a new point x_0 compute

$$f_{avg}(x_0) = \frac{1}{B}\sum_{b=1}^{B} f_b(x_0)$$

i.e. average of prediction (using model f) ~~after each~~ all individual bootstrap data sets

- Regression: $f_{avg}(x_0)$ is new prediction
- Classification: $f_{avg}(x_0)$ is an average over B votes → pick class label ~~on~~ from majority vote



Example (bagging trees)

- Binary classification $y \in \{0,1\}$ and $x \in \mathbb{R}^5$
- Each tree generated according to bootstrap sample B_b with $b \in [1,11]$
- Variation among trees as they are estimated from re-sampled data
- Bagging - each tree can be 'mediocre', improves results when function is non-linear (?)

- Bagging on trees works by exploiting bias-variance trade off
- Bagged trees are correlated however
- In general, when bootstrap samples are correlated (explain mechanism pls), bagging benefits decrease.

Random Forests - Motivation, algorithm, example

A modification of bagging where trees are designed to reduce correlation
It is a simple modification and still estimate trees over bootstrap sets B_b.
However to partition a region, we only consider a random subset of dimensions of $x \in \mathbb{R}^d$ and partition along one of these dimensions as usual

gorithm :-

Input parameter : m - numbe of dimensions considered in each split with $m > 0$ $m < d$ $m \approx \sqrt{d}$ i.e. smaller than no. of dimensions in $\mathbb{R}$

or $b = 1, \ldots, B$:-

. Draw a bootstrap sample B_b of size n from training data

. Train a tree classifier/(regression) on B_b with each split computed as follows

:- Randomly select m dimensions of $x \in \mathbb{R}^d$, newly chosen for each b
- make the best split restricted to that subset of dimensions

Comment :-

ch tree considers a small subset of dimensions in dataset

gging for trees : Bag trees using original algorithm on bootstrapped dataset

ndom Forests : Bag trees estimated using algorithm over subset of original dimensions for each tree.

de

lassification, forest size: few hundred trees

tree predicts orange/blue class, majority vote leeds ~~to~~ of prediction ⇒

ion boundary

nated decision boundaries can be complex

Bagging binary classifiers

Algorithm

Given $(x_1, y_1), \ldots, (x_n, y_n)$, $x \in \mathcal{X}$ $y \in \{-1, +1\}$

- For $b = 1, \ldots, B$
  - sample a bootstrap dataset $\mathcal{B}_b$ of size n. For each entry in $\mathcal{B}_b$, select with probability $(\frac{1}{n})$ (uniform). Some (x_i, y_i) will repeat and some in $\mathcal{B}_b$
  - Learn a classifier f_b using data in bootstrap dataset $\mathcal{B}_b$
- Define classification rule:-

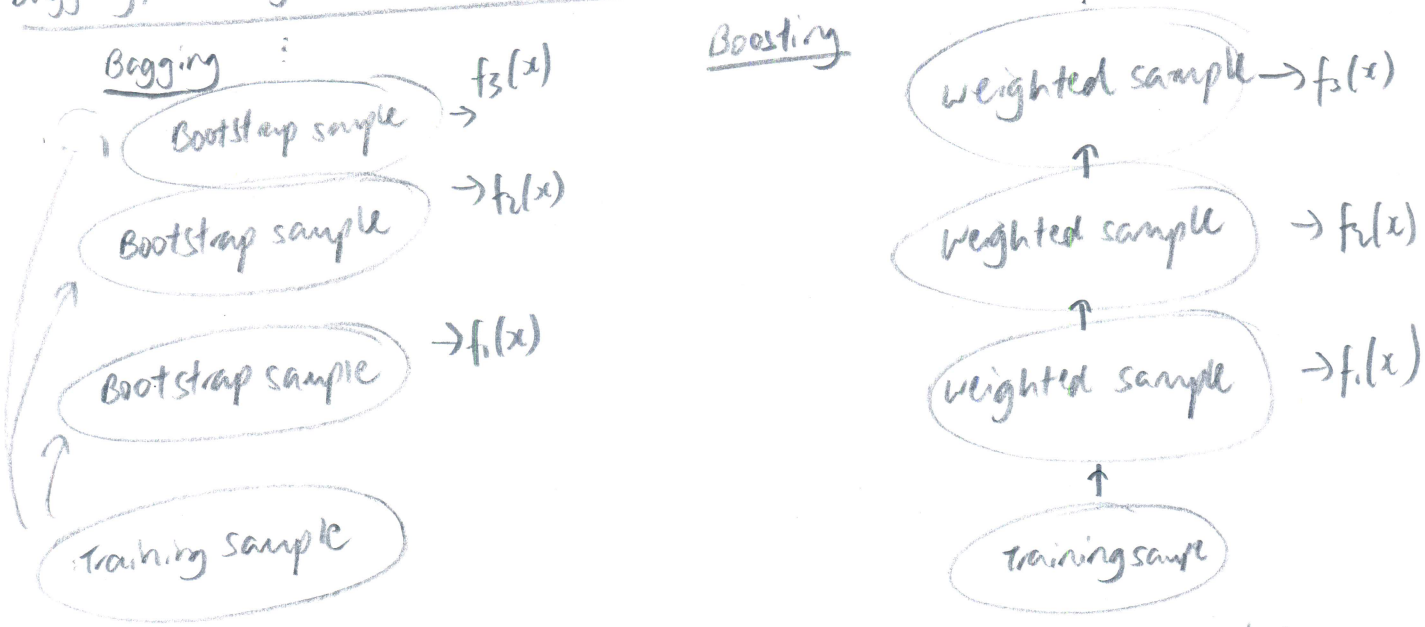
$$f_{bag}(x_0) = \text{sign}\left(\sum_{b=1}^{B} f_b(x_0)\right)$$

- Bagging: A committee of classifiers votes on a label with each classifier being learnt/estimated on a bootstrap sample from the data set. Learning a collection of classifiers → ensemble method

Boosting - context

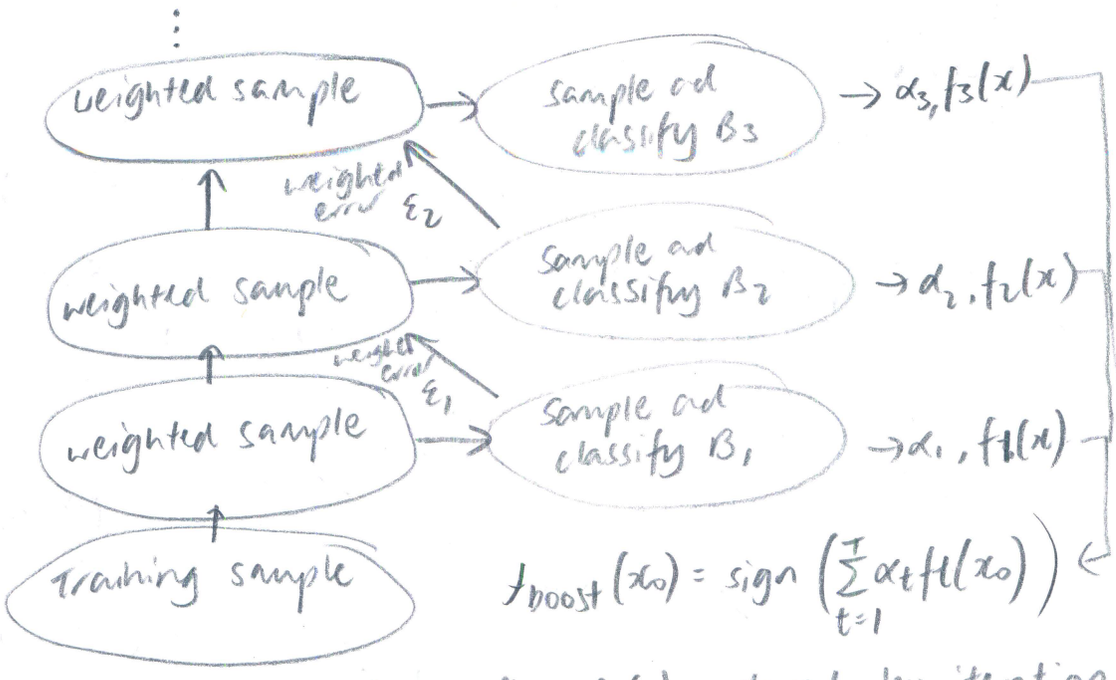
- Another method for ensemble learning; similar to bagging as we combine a set of class.
- Key figures are Leslie Valiant, Michael Kearns, Robert Schapire and Yoav Freund
- Works for any classifier, but weak one to learn usually chosen
- Weak - better than random guessing

Bagging, boosting, Adaboost (sampling version) - illustrations



- new weighted sample generated from previous weighted samples

Adaboost



- AdaBoost outputs a sequence of classifies $f_t(x)$ indexed by iteration t together with an associated α_t that weights the contribution of each classifier $f_t(x)$.
- The resulting boosted classifier f_{boost} then predicts for new x_0 by taking the weighted sum of predictions over all iterations and outputs a classification; this prediction is a majority WEIGHTED vote

Adaboost algorithm (sampling version)

Algorithm

Given $(x_1, y_1), \ldots, (x_n, y_n)$; $x \in X$ $y \in \{-1, +1\}$ $w \in \mathbb{R}^n$ (n-dimensional weight vector)

set $w_1(i) = \frac{1}{n}$

- For $t = 1, \ldots, T$
  1. Sample a bootstrap dataset B_t of size n according to distribution w_t. choose each (x_i, y_i) with probability $w_t(i)$, not $\frac{1}{n}$.
  2. Estimate/learn a classifier f_t using data in bootstrap sample B_t
  3. set $\varepsilon_t = \sum_{i=1}^{n} w_t(i) \mathbb{1}\{y_i \neq f_t(x_i)\}$ and $\alpha_t = \frac{1}{2} \ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$
  4. Scale $\hat{w}_{t+1}(i) = w_t(i) e^{-\alpha_t y_i f_t(x_i)}$ and set $w_{t+1}(i) = \frac{\hat{w}_{t+1}(i)}{\sum_j \hat{w}_{t+1}(j)}$
- Set classification rule (for new data point x_0) to be

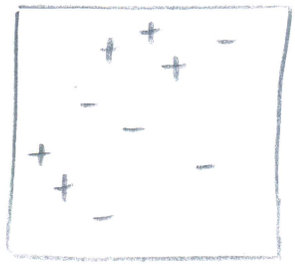
$$f_{boost}(x_0) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t f_t(x_0)\right)$$

- Note that B_t is now indexed by iteration t rather than number of bootstrap samples b
- T is large and arbitrary
- Initialise with distribution $w_{t=1}(i)$: $w_{t=1}(i)$ is uniform with probability $\frac{1}{n}$
- Calculate probability of error at iteration t (ϵ_t) according to distribution on data at iteration t
- ϵ_t = sum of probability weights w_t on misclassified data according to classifier learned at iteration t [weighted classification error]
- Reasons for particular setting α_t transpires later
- $\sum_j \hat{w}_{t+1}(j)$ is a normalisation constant that renders $w_{t+1}(i)$ a probability weight
- We have T pairs of classifiers f_t and associated weights α_t which are united to yield a boosted classifier f_{boost} which for a new data point x_0 gives the sign of each individual classifier f_t's prediction weighted by corresponding α_t. [α_t determines votes to assign to new classifier f_t]
- majority weighted vote

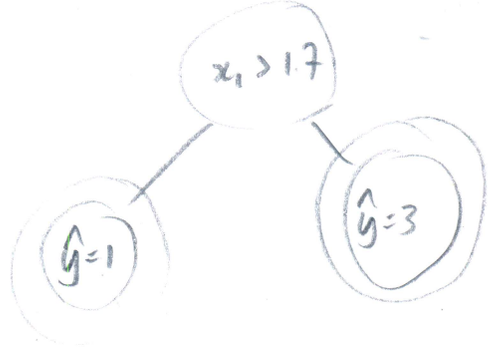
Boosting decision stump - example

(see diagram / draw to nearest intuition)

- 1st diagram reproduced, followed by exposition:-



original data
- uniform distribution w_1
- learn weak classifier
- use decision stump →



- one root node, 2 children

- Each point corresponds to covariate vector x
- + or − indicates class
- weights represented by size of symbol

Initial $\epsilon_1 = \sum_{i=1}^{10} w_1(i)\,\mathbb{1}\{y_i \neq f_1(x_i)\} = (0.1\times1) + (0.1\times1) + (0.1\times1) = 0.3$ (3 points misclassified)

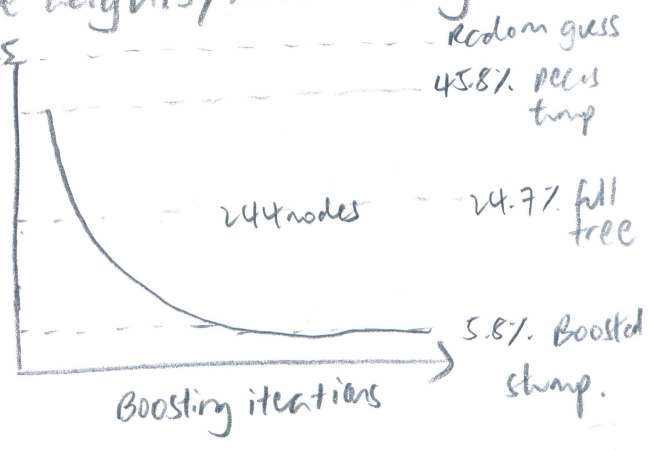
$$\alpha_1 = \frac{1}{2}\ln\left(\frac{1-0.3}{0.3}\right) \approx 0.42$$

(on)
- After 1 round/iteration; all the points that were misclassified receive an up-weighting in the next round; and all the points that were classified correctly have weights decreased. As iterations proceed, observations which are difficult to classify correctly receive increasing influence; each successive classifier

forced to concentrate on training observations that are missed by previous ones in sequence.

(A) - see mechanics of update rule at the end (OTI): upweighting/downweighting

- After 3 rounds, we have 3 classifiers (weak) (decision stumps) f_1, f_2, f_3 and corresponding weights $\alpha_1, \alpha_2, \alpha_3$
- This results in a net non-linear decision boundary; more complex decision boundaries with more classifiers/more weights/more boosting rounds.
- Plotting test error against boosting iterations
- Example of boosting on one dataset
- Examples of boosting on multiple datasets, varying by choice of algorithm (see diagram) complexity (stump vs C4.5)
  - ↳ location represents error rate
  - Above 45° line ⇒ boosting improves error



- C4.5 is a more complex decision tree
- Boosting improves performance for classifiers in general, but significantly more in the case of weak classifiers
- Boosting a bad classifier (e.g. decision stumps) is better than not boosting a good classifier (e.g. C4.5)
- Boosting a good classifier is often better, but takes more time ie. computational requirements at play (greater for more complex algorithms)
- No discussion of time complexity → O(?); (Q)

Boosting as a feature mapping

(Q) Why does boosting work so well? (Does it?)

(A) well-studied. One analysis presented later.

- We can provide an intuition using existing understanding of feature mappings
- For new x_0,

$$f_{boost}(x_0) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t f_t(x_0)\right)$$

- Define $\phi(x) = [f_1(x), f_2(x), \ldots, f_T(x)]^T$ with each $f_t(x) \in \{-1, +1\}$
- The 'adaptively' generated classifiers generated by the Adaboost algorithm over T iterations can be stacked into a vector (high dimensional given large T), so ϕ can be seen as a high-dimensional feature map of x.

$\underline{\alpha} = [\alpha_1, \ldots, \alpha_T]^T$, corresponding to a hyperplane and defining a linear decision boundary

(VII) classifier f_{boost} can be written:- (i.e. find linear classifier that perfectly classifies training set)

$$f_{boost}(x_0) = \text{sign}\left(\phi(x_0)^T \underline{\alpha}\right)$$

- Boosting learns a feature mapping and hyperplane simultaneously

Face-detection application (Viola and Jones 2001)

(VIII) Boosting used to filter through all info and find which combinations of dimensions/features and which weak classifiers to combine to declare that there is an image in the data set

- AdaBoost used to both select features and train classifier
- Allows us not to have to try to define the perfect classifier

Boosting training error theorem - (Freund & Schapire)

use analysis to make a statement about the accuracy of boosting on training data.

(IX) Theorem

under the Adaboost framework, if ϵ_t is the weighted error of the classifier f_t, then for the classifier $f_{boost}(x_0) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t f_t(x_0)\right)$, we then have

$$\text{training error} = \frac{1}{n}\sum_{i=1}^{n} \{y_i \neq f_{boost}(x_i)\} \leq \exp\left(-2\sum_{t=1}^{T}\left(\frac{1}{2} - \epsilon_t\right)^2\right)$$

ments

aining error - take boosted classifier evaluated on all points in our data set, n no. misclassified data points and divide by no. of observations (using T ds to construct f_{boost})

ch round of boosting yields a corresponding ϵ_t

ak classifier assumption: ea better than random guessing $\Rightarrow \epsilon_t < \frac{1}{2}$ $(\Rightarrow \alpha_t > 0)$

n if ϵ_t is a little better than random guessing; sum over T classifiers

ds to large negative values in exponent with large no. iterations T

. $\epsilon_t = 0.45$ $T = 1000 \Rightarrow$ training error ≤ 0.0067 (!)

Proof of theorem

Set up

- Set a = training error $= \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}\{y_i \neq f_{boost}(x_i)\}$ (and $b = \prod_{t=1}^{T} z_t$)
- Set c = bound $= \exp\left(-2\sum_{t=1}^{T}\left(\frac{1}{2}-\epsilon_t\right)^2\right)$

Strategy

- Show $a \leq c$ finding an expression for b such that $a \leq b$ and $b \leq c$ (transitivity argument)

1) Find an expression for 'b', the intermediate term
2) Derive a lower and upper bound for b, involving 'training error' and 'bound'
3) Involves manipulating expressions, use of inequalities

Key expressions

i) $\hat{w}_{t+1}(i) = w_t(i)e^{-\alpha_t y_i f_t(x_i)}$ (update rule)

ii) $w_{t+1}(i) = \frac{\hat{w}_{t+1}(i)}{\sum_j \hat{w}_{t+1}(j)}$ (weight normalisation)

iii) $z_t = \sum_j \hat{w}_{t+1}(j)$ (normalisation constant)

Step 1: Rewrite recursive update rule ($w_{T+1}(i)$)

- using (i) and (ii) and (iii), we have :-

$$w_{t+1}(i) = \frac{\hat{w}_{t+1}(i)}{\sum_j w_{t+1}(j)} = \frac{w_t(i)e^{-\alpha_t y_i f_t(x_i)}}{z_T}$$

- unfolding update rule to initial point from T+1 (T times)

- $w_2(i) = \frac{w_1(i)\,e^{-\alpha_1 y_i f_1(x_i)}}{z_1}$

- Each unfolding yields a factor $\frac{e^{-\alpha_s y_i f_s(x_i)}}{z_s}$

- Note that $w_1(i) = \frac{1}{n}$ (uniform at initialisation)

$$\Rightarrow w_{T+1}(i) = w_1(i)\cdot\frac{e^{-\alpha_1 y_i f_1(x_i)}}{z_1}\cdot\frac{e^{-\alpha_2 y_i f_2(x_i)}}{z_2}\cdots\frac{e^{-\alpha_t y_i f_t(x_i)}}{z_T}$$

$$\Rightarrow w_{T+1}(i) = \frac{1}{n}\frac{e^{-\sum_{t=1}^{T}\alpha_t y_i f_t(x_i)}}{\prod_{t=1}^{T} z_t} = \frac{1}{n}\frac{e^{-y_i\sum_{t=1}^{T}\alpha_t f_t(x_i)}}{\prod_{t=1}^{T} z_t}$$

- And that $e^{-y_i\sum_{t=1}^{T}\alpha_t f_t(x_i)} = e^{-y_i f^{(T)}_{boost}(x_i)}$

- Note $f^{(T)}_{boost}(x_i)$ is classifier boosted after T iterations; before it has been signed!

OR:

$$\Rightarrow w_{T+1}(i) = \frac{1}{n}\frac{e^{-y_i f^{(T)}_{boost}(x_i)}}{\prod_{t=1}^{T} z_t} \Rightarrow w_{T+1}(i)\prod_{t=1}^{T} z_t = \frac{1}{n}e^{-y_i f^{(T)}_{boost}(x_i)}$$

Step 2: Training error of $f_{BOOST}^{(T)}$ after T steps $\leq \prod_{t=1}^{T} z_t$ (lower bound on $\prod_{t=1}^{T} z_t$)

- using inequality $0 < e^{z_1} \; 1 < e^{z_2}$ for any $z_1 < 0 < z_2$:-

$$\frac{1}{n}\sum_{i=1}^{n} \mathbb{1}\{y_i \neq f_{BOOST}^{(T)}(x_i)\} \leq \frac{1}{n}\sum_{i=1}^{n} e^{-y_i f_{BOOST}^{(T)}(x_i)} \quad (*)$$

- using $n\, w_{T+1}(i) \prod_{t=1}^{T} z_t = e^{-y_i f_{BOOST}^{T}(x_i)}$

$$\leq \frac{1}{n}\sum_{i=1}^{n}\left(n\, w_{T+1}(i) \prod_{t=1}^{T} z_t\right)$$

$$\leq \sum_{i=1}^{n}\left(w_{T+1}(i) \prod_{t=1}^{T} z_t\right)$$

- As $\prod_{t=1}^{T} z_t$ is constant wrt to index i

$$\leq w_{T+1}(1)\prod_{t=1}^{T} z_t + w_{T+1}(2)\prod_{t=1}^{T} z_t \ldots$$

$$\leq \left(\sum_{i=1}^{n} w_{T+1}(i)\right)\left(\prod_{t=1}^{T} z_t\right)$$

- As $\sum_{i=1}^{n} w_{T+1}(i) = 1$ (probability weights over n data points sum to unity)

(OT)

$$\Rightarrow \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}\{y_i \neq f_{BOOST}^{T}(x_i)\} \leq \prod_{t=1}^{T} z_t$$

(*): $\mathbb{1}\{y_i \neq f_{BOOST}^{T}(x_i)\} \leq e^{-y_i f_{BOOST}^{(T)}(x_i)} \quad \forall i$:

$y_i = f_{BOOST}^{T}(x_i)$	INCORRECT
0 or 1	e^{-1} or e^{1}

- For i^{th} data point : LHS can take on values 0 or 1
  RHS can take on values e^{-1} or e^{1}

Step 3: $\prod_{t=1}^{T} z_t \leq \exp\left(-2\sum_{t=1}^{T}\left(\frac{1}{2} - \epsilon_t\right)^2\right)$

- Note from iii) and i); using same indexing letter

$$z_t = \sum_i \hat{w}_{t+1}(i) = \sum_i w_t(i) e^{-\alpha_t y_t f_t(x_i)}$$

$$= \sum_{i: y_i = f_t(x_i)} e^{-\alpha_t} w_t(i) + \sum_{i: y_i \neq f_t(x_i)} e^{\alpha_t} w_t(i)$$

$$= e^{-\alpha_t} \sum_{i: y_i = f_t(x_i)} w_t(i) + e^{\alpha_t} \sum_{i: y_i \neq f_t(x_i)} w_t(i)$$

(OT)

$$z_t = e^{-\alpha_t}(1 - \epsilon_t) + e^{\alpha_t}\epsilon_t$$

- ~~using sa~~
- Partition z_t into sets depending on whether (x_i, y_i) is misclassified according to classifier at t^{th} iteration

	y_i	$f_t(x_i)$	—	Net
· correctly classify :	(+)	(+)	(−)	(−)
	(−)	(−)	(−)	(−)
· Incorrectly " :	(−)	(+)	(−)	(+)
	(+)	(−)	(−)	(+)

- $\alpha_t \geq 0$
- $\epsilon_t = \sum_{i=1}^{n} w_t(i)\, \mathbb{1}\{y_i \neq f_t(x_i)\} =$

$$\sum_{i: y_i \neq f_t(x_i)} w_t(i)$$

- We want to minimise <u>training error</u> ⇒ choose α_t to minimise Z_t (2)
- ϵ_t is determined by the data set ; α_t as a free parameter

- $\frac{dZ_t}{d\alpha_t} = -(1-\epsilon_t)e^{-\alpha_t} + e^{\alpha_t}\epsilon_t = 0$

$$\Rightarrow \quad e^{-\alpha_t}(1-\epsilon_t) = e^{\alpha_t}\epsilon_t$$

$$\Rightarrow \quad \left(\frac{1-\epsilon_t}{\epsilon_t}\right) = e^{2\alpha_t}$$

$$\Rightarrow \quad \alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right) \rightarrow \text{subs into } Z_t$$

- Value of α_t used by Adaboost

$$Z_t = e^{-\alpha_t}(1-\epsilon_t) + e^{\alpha_t}\epsilon_t = e^{-\frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)}(1-\epsilon_t) + e^{\frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)}\epsilon_t$$

$$= \left(e^{\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)}\right)^{-\frac{1}{2}}(1-\epsilon_t) + \left(e^{\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)}\right)^{\frac{1}{2}}\epsilon_t$$

$$= \left(\frac{\epsilon_t}{1-\epsilon_t}\right)^{\frac{1}{2}}(1-\epsilon_t) + \left(\frac{1-\epsilon_t}{\epsilon_t}\right)^{\frac{1}{2}}\epsilon_t$$

$$= \left(\epsilon_t(1-\epsilon_t)\right)^{\frac{1}{2}} + \left(\epsilon_t(1-\epsilon_t)\right)^{\frac{1}{2}}$$

$$\Rightarrow \boxed{Z_t = 2\sqrt{\epsilon_t(1-\epsilon_t)}}$$

- Rewriting Z_t

$$Z_t = \sqrt{4\epsilon_t - 4\epsilon_t^2} = \sqrt{1-1+4\epsilon_t-4\epsilon_t^2} = \sqrt{1-4\left(\frac{1}{4}+\epsilon_t-\epsilon_t^2\right)}$$

$$= \sqrt{1-4\left(\frac{1}{2}-\epsilon_t\right)^2}$$

$$Z_t = \left(1-4\left(\frac{1}{2}-\epsilon_t\right)^2\right)^{\frac{1}{2}} \leq \left(e^{-4\left(\frac{1}{2}-\epsilon_t\right)^2}\right)^{\frac{1}{2}}$$

- using inequality $\boxed{1-x \leq e^{-x}}$
- setting $x = 4\left(\frac{1}{2}-\epsilon_t\right)^2$
- Assuring sqrt(·) > 0
- As LHS or RHS > 0 (postive)
- Apply product operator over T iterations

$$\Rightarrow \quad Z_t \leq e^{-2\left(\frac{1}{2}-\epsilon_t\right)^2}$$

$$\Rightarrow \quad \prod_{t=1}^{T} Z_t \leq \prod_{t=1}^{T} e^{-2\left(\frac{1}{2}-\epsilon_t\right)^2}$$

(or)

$$\Rightarrow \quad \boxed{\prod_{t=1}^{T} Z_t \leq \exp\left(-2\sum_{t=1}^{T}\left(\frac{1}{2}-\epsilon_t\right)^2\right)}$$

Finally:

(VII) $$\frac{1}{n}\sum_{i=1}^{n}\hat{\mathbb{1}}\{y_i \neq f_{BOOST}(x_i)\} \leq \prod_{t=1}^{T} Z_t \leq \exp\left(-2\sum_{t=1}^{T}\left(\frac{1}{2}-\epsilon_t\right)^2\right)$$

Comments

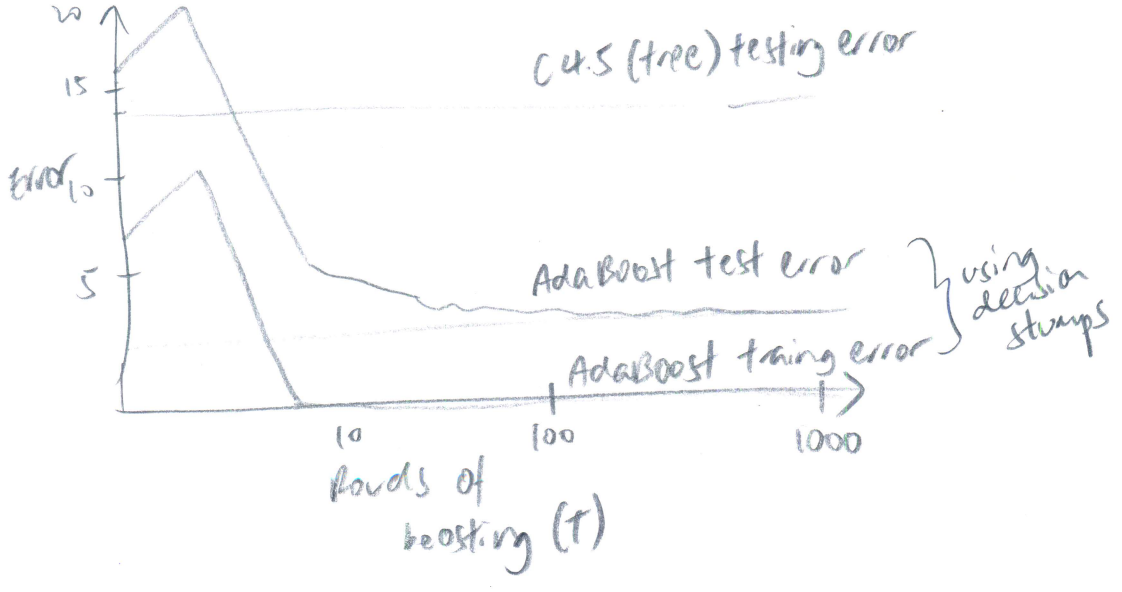
- Bound on this theorem is an inequality that holds regardless of the dataset and its subsequent inducement on possible realisations of ϵ_t
- Rules on calculating $\epsilon_t = \sum_{i=1}^{n} w_t(i)\,\mathbb{1}\{y_i \neq f_t(x_i)\}$ based on performance of classifier $f_t(x_i)$ over data points at iteration t.
- Add rule for calculation of $\alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
- Following these on any data set → inequality holds
- This is a ~~com~~ theoretical guarantee that bounds <u>training error</u>

Training and test error on Adaboost

(Q) Driving training error to zero makes one ask if boosting overfits?

(A) Sometimes, but very often doesn't (Q) → See research papers by Freund, Schapire

(Q) → Generalisation error bounds on Adaboost
→ unclear why it works so well with test error (empirical reports of test error continuing to decline); generally should increase due to overfitting
→ AdaBoost → Godel Prize in theoretical CS
→ Boosting ~~is not~~ empirical performance and theoretical explanation (equally test error not fully understood) (Friedman)
→ See papers, bibliographic notes in Hastie et al.
→ As an ergodic dynamical system (Breiman conjecture)



Hard clustering models - K-means algorithm

- Review of K-means algorithm (see L11)

Algorithm:

Given: Data $x_1, \dots, x_n$ where $x \in \mathbb{R}^d$

Goal: minimise $\mathcal{L} = \sum_{i=1}^{n} \sum_{k=1}^{K} \mathbb{1}\{c_i = k\} \|x_i - \mu_k\|_2^2$

- Iterate until values no longer change

1. update c: For each i, set $c_i = \operatorname{argmin}_k \|x_i - \mu_k\|_2^2$
2. update μ: For each k, set $\mu_k = \sum_i x_i \{c_i = k\} / \sum_i \{c_i = k\}$

} co-ordinate descent

- $\mathcal{L}$: we minimise an objective where we sum over all data points $x_1, \dots, x_n$, the squared distance to the cluster centroid it was assigned to encoded by parameters c_i
$\Rightarrow x_i$ is assigned to one of K clusters and c_i indexes the cluster
- K-means is an example of a hard clustering algorithm as observations are assigned to only one cluster.
- In other words $c_i = k$ for some $k \in \{1, \dots, K\}$, no accounting for boundary cases through a probabilistic hedging on c_i.
- We can encode uncertainty about a particular data point on 'boundary' with a 'probability weight'.

Soft-clustering models - weighted K-means algorithm

- Soft clustering algorithms breaks data across clusters 'intelligently'.
- See diagram → shaded dots based on probability

Weighted K-means algorithm:-

Given: Data $(x_1, \dots, x_n)$ where $x \in \mathbb{R}^d$

Goal: minimise $\mathcal{L} = \sum_{i=1}^{n} \sum_{k=1}^{K} \phi_i(k) \frac{\|x_i - \mu_k\|_2^2}{\beta}$ over ϕ_i and μ_k

Conditions: $\phi_i(k) > 0$ and $\sum_{k=1}^{K} \phi_i(k) = 1$, set parameter $\beta > 0$

- Iterate the following:-

1. update ϕ: For each i, update the word allocation weights

$$\phi_i(k) = \frac{\exp\{-\frac{1}{\beta}\|x_i - \mu_k\|_2^2\}}{\sum_j \exp\{-\frac{1}{\beta}\|x_i - \mu_k\|_2^2\}} \quad \text{for } k = 1, \dots, K$$

2. update μ: For each k, update μ_k with the weighted average

$$\mu_k = \frac{\sum_i x_i \phi_i(k)}{\sum_i \phi_i(k)}$$

Comments:-

- Instead of $\mathbb{1}\{c_i = k\}$ (K-means), substitute $\phi_i(k) = \frac{\exp\{-\frac{1}{\beta}\|x_i - \mu_k\|_2^2\}}{\sum_j \exp\{-\frac{1}{\beta}\|x_i - \mu_k\|_2^2\}}$ and hyperparam. β.

- Notice functional similarity with RBF kernel $K(x,x') = a\exp\{-\frac{1}{b}\|x-x'\|_2^2\}$ as a proximity measure
- Note $\sum_j \exp\{-\frac{1}{\beta}\|x_i-\mu_j\|_2^2\}$ is a normalisation constant
- x_i close to μ_k: large $\phi_i(k)$; x_i far from μ_k: small $\phi_i(k)$
- Setting $\phi_i(k) = \{0,1\} \Rightarrow$ hard K-means clustering.
- See diagram illustrating 'pabability weights' ϕ_i.

Mixture models - general

- Probabilistic vs non-probabilistic soft clustering:-
  - Previously we have likened ϕ_i to a probability of x_i being assigned to each cluster
- A mixture model is a probabilistic model where ϕ_i is actually a probability distrib. according to the model
- Mixture models work by explicitly defining:-
  - A prior distribution on the cluster assignment indicator c_i $p(c_i)$
  - A likelihood distribution on observation x_i given the assignment c_i $p(x_i|c_i)$
- Intuitively, a mixture model is the unsupervised analogue of the Bayes classifier
  - class priors and class conditional likelihood become cluster priors and cluster-
  - we do not know 'label'
- Mixture models - key features:-
  1) Generative model (defines a probability distribution on the data)
  2) A weighted combination of simpler distributions where each distribution is in the same distribution family and the weighting is defined by a discrete probability distrib.
- Example shows 3 MVGs combined; with z-axis corresponding to density at a given point in $\mathbb{R}^2$.

(21) Mixture models :-

- To define a mixture model, we have to define a generative process on the data in Bayesian style.

Data:- $x_1, x_2, \ldots, x_n$ where each $x_i \in X$ (imagine $X = \mathbb{R}^d$)

Model parameters: A K-dimensional distribution π and parameters $\theta_1, \ldots, \theta_K$ due to there being K-clusters

(22) π - mixing weights

Generative process: For observation $i = 1, \ldots, n$

1. Generate cluster assignment $c_i \overset{iid}{\sim} \text{Discrete}(\pi) \Rightarrow \text{Prob}(c_i = k|\pi) = \pi_k$
2. Generate observation $x_i \sim p(x|\theta_{c_i})$

↳ Parameter used is picked out by indicator c_i

comments

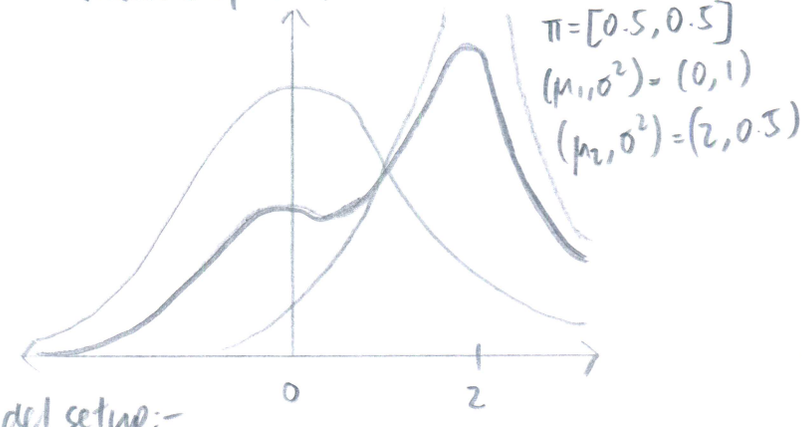
- Each x_i is randomly assigned to a cluster a priori using distribution π
- c_i indexes cluster assignment for x_i
  - Picks out the index of the parameter θ used to generate x_i
  - If two x's share a parameter → they are clustered together; i.e. if x_i, x_j are assigned to the same cluster then $c_i = c_j$ (both generated from the same probability distribution and they share same parameters to that distribution)
- Assume $p(x|\theta_{c_i})$ distribution family is the same for all clusters (MVG); differing only by parametrisation

(see diagram) → shows impacts of mixing weights that is drawn from the discrete probability distribution. Gaussians define region data falls within; weights define 'density' of data in terms of allocation to each region

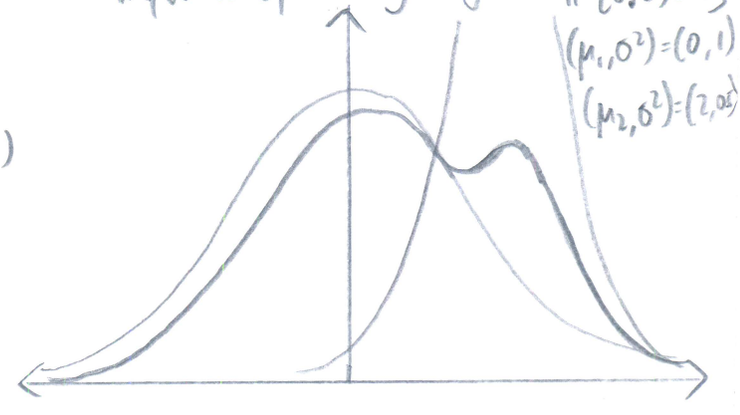
Gaussian mixture models (GMMs) - illustration, setup

- Gaussian mixture models are models where $p(x|\theta)$ is Gaussian:-

Mixture of 2 Gaussians



Influence of mixing weights



model setup:-

- generative model ⇒ joint likelihood on the data given parameters and distribution family (I)
- parameters: Let π be a K-dimensional probability distribution and (μ_k, Σ_k) be the mean and covariance of the k^{th} Gaussian in $\mathbb{R}^d$
- generate data: For the i^{th} observation,
  1. Assign the i^{th} observation to cluster $c_i \sim \text{Discrete}(\pi)$ $p(c_i|\pi_i)$
  2. Generate the value of the observation $x_i \sim N(\mu_{c_i}, \Sigma_{c_i})$ $(p(x|\theta_{c_i}))$
     - parameter (μ_k, Σ_k) and respective Gaussian used 'selected' by indicator c_i
- definitions: $\underline{\mu} = \{\mu_1, \dots, \mu_K\}$ and $\underline{\Sigma} = \{\Sigma_1, \dots, \Sigma_K\}$
- goal: Learn π, $\underline{\mu}$ and $\underline{\Sigma}$
- Given (I); what are the parameters that generated the data under that distribution family?

EM Algorithm for MLE in context of GMMs - motivation, setup, algorithm, illustration

Motivation for using EM algorithm:-

Objective: Maximise likelihood over model parameters π, μ, Σ by treating values of cluster assignments c_i as auxiliary/latent variables using EM algorithm.

$$p(x_1, \dots, x_n | \pi, \mu, \Sigma) \overset{(i)}{=} \prod_{i=1}^{n} p(x_i | \pi, \mu, \Sigma) \overset{(ii)}{=} \prod_{i=1}^{n} \sum_{k=1}^{K} p(x_i, c_i = k | \pi, \mu, \Sigma)$$

(i) - iid (ii) Rewrite $p(x_i | \pi, \mu, \Sigma)$ i.e. marginal distribution of x_i as joint distribution (likelihood) over observed x_i and cluster assignments summed over all possible discete outcomes $c_i = k$ i.e. $\sum_{k=1}^{K} p(x_i, c_i = k | \mu, \Sigma)$.

Note: summation over values of c_i as it is discrete, rather than an integral for continuas.
Analytic methods using derivatives/gradients of π, μ, Σ not possible
could use iterative gradient methods, but EM is cleaner.

Q: Why not include each c_i and maximise $\prod_{i=1}^{n} p(x_i, c_i | \pi, \mu, \Sigma)$ since we can do this via co-ordinate ascent?

Why not keep c_i in the model as something we want to infer and do a point estimate of c_i in addition to point estimate of π, μ, Σ

End up with hard clustering model; updating a particular cluster assignment and maximising c_i would mean assigning it to the most probable cluster; so eximising likelihood solution or MAP solution → hard clustering.
we want soft-clustering (integrate out effect of the cluster assignment) which is ore accurate and better than hard clustering because we've integrated out knowns → fewer potential local optima for algorithm convergence.

up:
t deriving all from first principles
eat $c_1, \dots, c_n$ as auxiliary data that is integrated out
se EM to

maximise $\sum_{i=1}^{n} \ln p(x_i | \pi, \mu, \Sigma)$ by using $\sum_{i=1}^{n} \ln p(x_i, c_i | \pi, \mu, \Sigma)$

ic can objective:-

$$p(x|\theta_1) = \underbrace{\int q(\theta_2) \ln \frac{p(x, \theta_2 | \theta_1)}{q(\theta_2)} d\theta_2}_{L(x, \theta_1)} + \underbrace{\int q(\theta_2) \ln \frac{q(\theta_2)}{p(\theta_2 | x, \theta_1)} d\theta_2}_{KL(q||p)}$$

bjective for GMM:

$$p(x_i | \pi, \mu, \Sigma) = \sum_{i=1}^{n} \sum_{k=1}^{K} q(c_i = k) \ln \frac{p(x_i, c_i = k | \pi, \mu, \Sigma)}{q(c_i = k)} + \sum_{i=1}^{n} \sum_{k=1}^{K} q(c_i = k) \ln \frac{q(c_i = k)}{p(c_i = k | x_i, \pi, \mu, \Sigma)}$$

g $\theta_2 = c_i$, $\theta_1 = \pi, \mu, \Sigma$, $x = x_i$; set ∧ individual KL divergences = 0 via settings on q

EM Iteration

- set $q(c_i=k) \Leftarrow p(c_i=k|x_i,\pi,\underline{\mu},\underline{\Sigma})$ ~~$\forall i$~~ using Bayes rule :-

$$p(c_i=k|x_i,\pi,\underline{\mu},\underline{\Sigma}) \propto p(c_i=k|\pi)\,p(x_i|c_i=k,\underline{\mu},\underline{\Sigma})$$

- Solve the posterior of c_i given $\pi,\underline{\mu},\underline{\Sigma}$ for each i

$$q(c_i=k) = \frac{\pi_k N(x_i|\mu_k,\Sigma_k)}{\sum_j \pi_j N(x_i|\mu_j,\Sigma_j)} \Rightarrow \phi_i(k)$$

- $p(c_i=k|x_i,\pi,\underline{\mu},\underline{\Sigma})$ is the conditional posterior distribution of the i^{th} data point being assigned to the k^{th} cluster given x_i, given parameters $(\pi,\underline{\mu},\underline{\Sigma})$ of GMM at current iteration
- $q(c_i=k)$ returns the probability that the i^{th} data point x_i lies in the k^{th} cluster. For each data point x_i, it will return K cluster probabilities that x_i will lie in the k^{th} cluster.

- Recall that this setting of q renders the KL divergence term $KL(q||p)=0$ for a particular x_i and that there is a summation operator $\sum_{i=1}^{n}$ over n data points.
- In this context, we will set n KL divergences to be equal to 0, indexed by data point x_i.
- More explicitly:-

$$\sum_{i=1}^{n} KL(q||p) = \sum_{i=1}^{n}\left(\sum_{k=1}^{K} q(c_i=k)\ln\frac{q(c_i=k)}{p(c_i=k|x_i,\pi,\underline{\mu},\underline{\Sigma})}\right)$$

$$= \left(\sum_{k=1}^{K} q(c_1=k)\ln\frac{q(c_1=k)}{p(c_1=k|x_1,\pi,\underline{\mu},\underline{\Sigma})}\right) + \left(\sum_{k=1}^{K} q(c_2=k)\ln\frac{q(c_2=k)}{p(c_2=k|x_2,\pi,\underline{\mu},\underline{\Sigma})}\right)$$

$$+ \ldots + \left(\sum_{k=1}^{K} q(c_n=k)\ln\frac{q(c_n=k)}{p(c_n=k|x_n,\pi,\underline{\mu},\underline{\Sigma})}\right)$$

- So this step involves setting n q-distributions over the auxiliary variables $c_1,\ldots,c_n$ as n conditional posterior distributions $p(c_i=k|x_i,\pi,\underline{\mu},\underline{\Sigma})$ indexed by data point x_i

E-step:

- As n-individual KL-divergences = 0, we are left with:-

$$\sum_{i=1}^{n}\ln p(x_i|\pi,\underline{\mu},\underline{\Sigma}) = \sum_{i=1}^{n}\sum_{k=1}^{K} q(c_i=k)\ln\frac{p(x_i,c_i=k|\pi,\underline{\mu},\underline{\Sigma})}{q(c_i=k)}$$

(EM objective for GMM with $KL(q||p)=0$)

$$= \sum_{i=1}^{n}\sum_{k=1}^{K} q(c_i=k)\ln p(x_i,c_i=k|\pi,\underline{\mu},\underline{\Sigma}) + \underbrace{\sum_{i=1}^{n}\sum_{k=1}^{K} q(c_i=k)\ln q(c_i=k)}_{\text{constant wrt } \pi,\underline{\mu},\underline{\Sigma}}$$

- Recall $q(c_i=k)=\phi_i(k)$; taking expectation using n updated q distributions over auxiliary variables and data x_i:-

$$\mathcal{L} = \sum_{i=1}^{n}\sum_{k=1}^{K}\phi_i(k)\ln p(x_i,c_i=k|\pi,\mu_k,\Sigma_k) + \text{constant wrt } \pi,\underline{\mu},\underline{\Sigma}$$

M-step :-

- Maximise $\mathcal{L}$ with respect to π and each μ_K, Σ_K for each K

EM made this easier?

original objective function:-

$$\mathcal{L} = \sum_{i=1}^{n} \ln p(x_i|\pi, \mu_K, \Sigma_K) = \sum_{i=1}^{n} \ln \left(\sum_{K=1}^{K} p(x_i, c_i = K|\pi, \mu, \Sigma) \right) = \sum_{i=1}^{n} \ln \sum_{K=1}^{K} \pi_K N(x_i|\mu_K, \Sigma_K)$$

rewriting marginal likelihood as joint likelihood over auxiliary summed over outcomes

- Optimisation difficulty → log-sum form for π and each μ_K, Σ_K for $K=1,\dots,K$ tricky

EM- Mstep :-

$$\mathcal{L} = \sum_{i=1}^{n} \sum_{K=1}^{K} \phi_i(k) \{ \ln \pi_k + \ln N(x_i|\mu_k, \Sigma_k) \} + \text{constant wrt } \pi, \mu, \Sigma$$

$$= \ln p(x_i, c_i = k|\pi, \mu_K, \Sigma_k)$$

- sum-log form easier to optimise
- can find $\nabla \mathcal{L}$ wrt π, μ, Σ, and solve for updates; with constraint that π is a probability distribution

EM Algorithm - maximum likelihood EM for GMM

Given: $x_1, \dots, x_n \in \mathbb{R}^d$

Goal: Maximise $\mathcal{L} = \sum_{i=1}^{n} \ln p(x_i|\pi, \mu, \Sigma)$ - no cluster assign as integrated out

- Iterate until incremental improvement to $\mathcal{L}$ is small :-

1. E-step : For $i = 1, \dots, n$ set

$$\phi_i(k) = \frac{\pi_K N(x_i|\mu_K, \Sigma_K)}{\sum_j \pi_j N(x_i|\mu_j, \Sigma_j)} \quad \text{for } k = 1, \dots, K$$

2. M-step : for $k = 1, \dots, K$, define $n_K = \sum_{i=1}^{n} \phi_i(k)$ and update the values

$$\pi_K = \frac{n_K}{n} \qquad \mu_K = \frac{1}{n_K} \sum_{i=1}^{n} \phi_i(k) x_i \qquad \Sigma_k = \frac{1}{n_K} \sum_{i=1}^{n} \phi_i(k)(x_i - \mu_K)(x_i - \mu_K)^T$$

Comment: updated value for μ_K is used when updating Σ_K

(+) Initialise means μ_K, covariances Σ_K, mixing coefficients π_K

Comments:-

- $\phi_i(k)$: probility (distribution) of i^{th} data point coming from k^{th} cluster
- $n_k = \sum_{i=1}^{n} \phi_i(k)$: sum over each data point of the probability of that data point coming from the k^{th} cluster; expected no. of points that we're going to see coming from k^{th} cluste at current iteration.
- $\pi_k = \frac{n_k}{n}$: prior probability of the k^{th} cluster; empirical distribution ~~of~~ on clusters according to our soft assignments
- $\mu_k = \frac{1}{n_k} \sum_{i=1}^{n} \phi_i(k) x_i$: mean for k^{th} cluste
  Take each data point and multiply by probability of it coming from k^{th} cluster, summed over data points and divide by expected/weighted total no. of points coming from that cluster
  - similar to K-means, except in the calculation of $\phi_i(k)$
- $\Sigma_k = \frac{1}{n_k} \sum_{i=1}^{n} \phi_i(k)(x_i - \mu_k)(x_i - \mu_k)^T$: covariance matrix for k^{th} cluste
  - weighted empirical covariance of data coming from k^{th} cluster using updated μ_k (mean of k^{th} cluste); calculate outer product, weight by probability that i^{th} point came from k^{th} cluster, sum over data points and divide by expected total no. of points
  - soft covariance matrix

expectations/sample averages

Illustration:-

(see diagram)

- Random initialisation
- 2 initial Gaussians i.e. π is a 2-dimensional probability distribution ~~and~~ with $K=$
- (μ_1, Σ_1) as mean and covariance of 1st Gaussian in $\mathbb{R}^2$ as $d=2$
- (μ_2, Σ_2) —— " —— of 2nd Gaussian in $\mathbb{R}^2$ as $d=2$
- weights not shown but $\pi = [\frac{1}{2}, \frac{1}{2}]$ ~~is~~ $\pi = [\pi_1, \pi_2]$ ($K \neq 2$ clustes)
- E-step (Iteration 1) - Assign data to clustes
- Take each point as x_i and calculate posterior probability $p(c_i = k | x_i, \pi, \mu, \Sigma)$
- colour code based on the weight of that probability (2)
- soft clustering on E-step; whereas K-means - line drawn instead of probabilistic colouring
- M-step (Iteration 1) - update Gaussians
- update Gaussians i.e. each π_k, μ_k, Σ_k for $K=2$

i.e. calculate new prior probability of k^{th} cluste (π_k)

~~Error~~ —"— empirical weighted mean of k^{th} cluste (μ_k)

—"— —"— covariance —"— (Σ_k)

- 1st Gaussian with $\mu_{1,ML}$ and $\Sigma_{1,ML}$ with probability π_1
- 2nd Gaussian with $\mu_{2,ML}$ and $\Sigma_{2,ML}$ with probability π_2 → based on empirical proportion of data from that cluster $\pi_k = \frac{n_k}{n}$
- Each Gaussian corresponding to a cluster has a probability, π_k based on empirical proportion of data coming from that cluster $\left(\frac{n_k}{n}\right)$

GMM and the Bayes classifier

- GMM feels like an unsupervised analogue of the Bayes classifier, where label of x_i is:-

$$\text{label}(x_i) = \underset{k}{\text{argmax}}\ \pi_k N(x_i | \mu_k, \Sigma_k)$$

→ π_k - class/cluster prior and $N(x_i|\mu_k, \Sigma_k)$ is the class/cluster-conditional density function
→ π, μ, Σ estimated via maximum likelihood (no EM algorithm)
Bayes classifier ; could find π, μ, Σ as cluster assignment/class label was known

GMM update: $\pi_k = \frac{n_k}{n}$ $\mu_k = \frac{1}{n_k}\sum_{i=1}^{n} \phi_i(k) x_i$ $\Sigma_k = \frac{1}{n_k}\sum_{i=1}^{n} \phi_i(k)(x_i - \mu_k)(x_i - \mu_k)^T$

Note similarity with estimation of Bayes classifiers via MLE and plug-in classifiers using available data to approximate $P(Y=y)$ and $P(X=x|Y=y)$)

$$\hat{\pi}_y = \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}\{y_i = y\} \quad \hat{\mu}_y = \frac{1}{n_y}\sum_{i=1}^{n} \mathbb{1}\{y_i = y\} x_i \quad \hat{\Sigma}_y = \frac{1}{n_y}\sum_{i=1}^{n} \mathbb{1}\{y_i = y\}(x_i - \hat{\mu}_y)(x_i - \hat{\mu}_y)^T$$

Almost identical expressions; recall multi-class class.
n_y (no. of observations labelled y) replaced with n_k (no. of observations clustered in k)
$\mathbb{1}\{y_i = y\}$ (indicator for whether example has class label y) replaced with $\phi_i(k)$ (probability of i^{th} data point being assigned to cluster k).
(k) as our example is constantly changing as we update values.
Bayes classifier "$\phi_i(k)$" encodes the label and was already known
GMM update: we iterate between updating cluster assignment and updating model param.; iterative algorithm as simultaneous optimisation not possible.
Inferring 'labels' and cluster-specific densities simultaneously

Picking number of clusters K

How many clusters in my dataset?
Maximum likelihood for Gaussian mixture model will learn/estimate parameters as many as it's given → responsibility on statistician
Will we assign each point 'softly' via a conditional posterior to K no. of Gaussians result in overfitting
techniques based on Dirichlet distribution (a multivariate generalisation of beta)
Dirichlet prior is used on π which encourages many Gaussians to disappear.

EM Algorithm for a generic mixture model

Algorithm:-

Given: Data $x_1, \ldots, x_n$ where $x \in \mathcal{X}$

Goal: Maximise $\ell = \sum_{i=1}^{n} \ln p(x_i | \pi, \underline{\theta})$ where $p(x|\theta_k)$ is problem-specific

- Iterate until incremental improvement in ℓ is small:-

1. E-step: For $i = 1, \ldots, n$ set

$$\phi_i(k) = \frac{\pi_k p(x_i|\theta_k)}{\sum_j \pi_j p(x_i|\theta_j)} \quad \text{for } k = 1, \ldots, K$$

2. M-step: For $k = 1, \ldots, K$ define $n_k = \sum_{i=1}^{n} \phi_i(k)$ and set

$$\pi_k = \frac{n_k}{n} \qquad \theta_k = \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{n} \phi_i(k) \ln p(x_i|\theta)$$

Comment → Similar to generalisation of Bayes classifier for any $p(x|\theta_k)$

Comments

- $p(x|\theta_k)$ - cluster specific likelihood/probability distribution (family) over a set of parametes θ_k.
- E-step - for each i observation, we calculate conditional posterior probabilty of i^{th} obs. coming from k^{th} cluster, using parametes of k^{th} cluste to evaluate likelihood via Bayes rule: same as prior probability of a point coming from k^{th} cluste multiplied by likelihood x_i comes from k^{th} cluster; normalised
- M-step for each k^{th} cluster, find expected no. of points from k^{th} cluster (by summing probabilities of all data coming from that cluster) & ; and update prior probability of point coming from that cluster (i.e. mixing weight)

updating θ_k: generic and may involve calling a different algorithm.