Lecture 6:
Stochastic optimization
EM algorithm

Stochastic and combinatorial optimization

- Unconstrained optimization:
  - Input variables are continuous
  - Response is differentiable

We could apply *Steepest descent, Newton, BFGS, CG*

- Now:
  - Variables can be discrete (scheduling problem, traveling salesman)
  - Outcome can be discrete, noisy (typical in statistics) or having multiple local minima

Stochastic and combinatorial optimization

Given set of states S, objective to minimize *f(s),* typically S is large.

- Sometimes, exhaustive search is possible (shortest path algorithm)
- Often exhaustive search is computationally expensive, sometimes NP-hard (traveling salesman)
- Alternative – a solution (sometimes exact) can be obtained by *stochastic methods* (ex: simulated annealing, genetic algorithms)

Simulated annealing

Idea comes from physics (melted metal is being cooled)

- Parameters:
  - Energy of the metal (decreasing, but not monotonic)
  - Temperature (decreasing)

- How to find minimum energy (global one)?

Simulated annealing

0. Set $k = 1$ and initialize state s.
1. Compute the temperature $T(k)$.
2. Set $i = 0$ and $j = 0$.
3. Generate a new state r and compute $\delta f = f(r) - f(s)$.
4. Based on δf, decide whether to move from state s to state r.
   If $\delta f \leq 0$,
   accept state r;
   otherwise,
   accept state r with a probability $P(\delta f, T(k))$.
   If state r is accepted, set $s = r$ and $i = i + 1$.
5. If i is equal to the limit for the number of successes at a given temperature, go to step 1.
6. Set $j = j + 1$. If j is less than the limit for the number of iterations at given temperature, go to step 3.

7. If $i = 0$,
   deliver s as the optimum; otherwise,
   if $k < k_{\max}$,
   set $k = k + 1$ and go to step 1;
   otherwise,
   issue message that
   'algorithm did not converge in $k_{\max}$ iterations'.

Simulated annealing

Comments

- Check https://www.youtube.com/watch?v=iaq_Fpr4KZc
- How to generate new state?
  - Continuous inputs: choose new point at some distance *r* from the current point *(r* can be a random variable..*)*
  - Discrete inputs: same idea or rearrangements
- How to choose selection probability
  - Sometimes chosen as exp(-δf/T)
- How to choose temperature function?
  - continuous or noisy functions, taken constant
  - Another choice

$$T(k+1) = b(k)T(k),\ b(k) = (\log(k))^{-1}$$

Simulated annealing

Example: Traveling salesman

- Assume constant temperature

1. Choose initial configuration ($A_1 \ldots A_n$)
2. Generate new configuration by 2-rearrangement:

$$(1, \underline{2, 3, 4, 5, 6}, 7, 8, 9) \rightarrow (1, \underline{6, 5, 4, 3, 2}, 7, 8, 9).$$

$$1, \underline{2, 3, 4, 5, 6}, 7, 8, {}_{\uparrow}\, 9) \rightarrow (1, 7, 8, \underline{2, 3, 4, 5, 6}, 9)$$

3. Measure difference in path length δf between new and old configuration
4. If shorter path found, accept it otherwise accept it with probability P(δf)
5. Repeat until maximum iteration condition fulfilled

Genetic algorithm

- Idea comes from biology (the fittest survives)

- Variables=genotypes
- Observation=organism, characterized by genetic code
- State space= Population of organisms
- Objective function=fitness of organism

New points are obtained from old points by crossover and mutation, the population retains only fittest organisms (with better objective function)

Genetic algorithm

- How to code the points
  1. Enumerate each point in *S*
  2. Code for observation *i* is presented by binary representation of *i*
  - Other encodings are possible

- Mutation and recombination rules

Generation k		Generation $k+1$
Crossover		
$x_i^{(k)}$ 11001001		
	$\rightarrow x_i^{(k+1)}$	11011010
$x_j^{(k)}$ 00111010		
Inversion		
$x_i^{(k)}$ 11101011	$\rightarrow x_i^{(k+1)}$	11010111
Mutation		
$x_i^{(k)}$ 11101011	$\rightarrow x_i^{(k+1)}$	10111011
Clone		
$x_i^{(k)}$ 11101011	$\rightarrow x_i^{(k+1)}$	11101011

Genetic algorithm

0. Determine a representation of the problem, and define an initial population, $x_1^{(0)}, x_2^{(0)}, \ldots, x_n^{(0)}$. Set $k = 0$.
1. Compute the objective function (the "fitness") for each member of the population, $f(x_i^{(k)})$ and assign probabilities p_i to each item in the population, perhaps proportional to its fitness.
2. Choose (with replacement) a probability sample of size $m \leq n$. This is the reproducing population.
3. Randomly form a new population $x_1^{(k+1)}, x_2^{(k+1)}, \ldots, x_n^{(k+1)}$ from the reproducing population, using various mutation and recombination rules (see Table 6.2). This may be done using random selection of the rule for each individual of pair of individuals.
4. If convergence criteria are met, stop, and deliver $\arg\min_{x_i^{(k+1)}} f(x_i^{(k+1)})$ as the optimum; otherwise, set $k = k + 1$ and go to step 1.

Genetic algorithm

Traveling salesman problem

Encoding and crossover

First idea - encode tours as $A_1 \ldots A_n$. Problem:

Parent 1	F A B | E C G D
Parent 2	D E A | C G B F
Child 1	F A B | C G B F
Child 1	D E A | E C G D

1. Instead: Remove FAB from DEACGBF -> DECG. Obtain first child by appending: FABDECG
2. Second child is obtained by taking prefix from parent 2

Genetic algorithm

Traveling salesman problem

Mutation

- If taking small population and using only crossover – the input domain becomes limited, may converge to local solution
- Taking large initial population may be computationally heavy
- Mutation allows to investigate entire input domain
- In traveling salesman, mutation =moving a city in the tour to another position

Genetic algorithm

Traveling salesman problem

Other issues

- Reproduction: Among *m* tours selected at step 2, two best are selected for reproduction, two worst replaced by children
- Neighborhood size: large *m* – some tours are never parents, global solution may not be attained
- Mutation probalility should be fixed

Stochastic gradient descent

- Machine learning models minimize empirical risk

$$R(\theta, \hat{\theta}) = \frac{1}{N}\sum_{i=1}^{N} L(\theta, \hat{\theta})$$

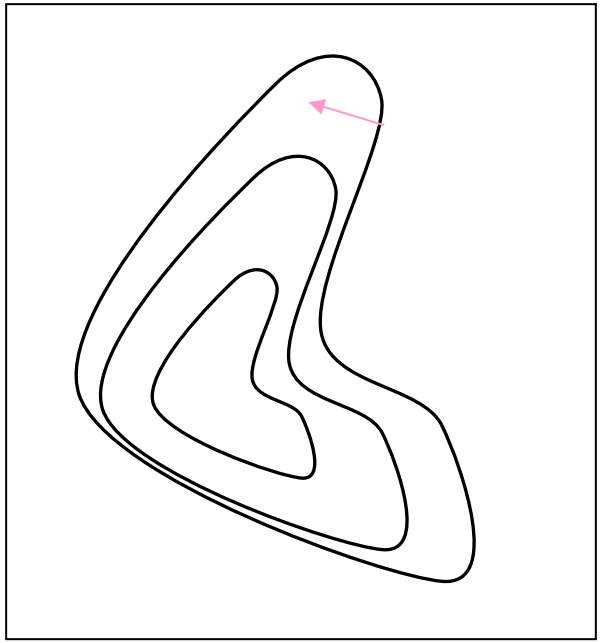
  - Regression models: $R(Y, \hat{Y}) = (Y - \hat{Y})^2$

- Usual gradient descent

$$\theta_{k+1} = \theta_k - \alpha_k \nabla R(\theta, \theta_k)$$

$$\theta_{k+1} = \theta_k - \alpha_k \frac{1}{N}\sum_{i=1}^{N} \nabla L(\theta, \theta_k)$$

- Takes a lot of time per iteration for large N



Stochastic gradient descent

- Main idea: $L(\theta_i, \hat{\theta}) \approx \frac{1}{N}\sum_{i=1}^{N} \nabla L(\theta, \theta_k)$ for any i
  - Update formula $\theta_{k+1} = \theta_k - \alpha_k \nabla L(\theta, \theta_k)$
  - Second order gradient descent $\theta_{k+1} = \theta_k - \alpha_k \Gamma_k \nabla L(\theta, \theta_k)$

- Converges to a local minimum but may require a lot more iterations than gradient descent
- Less time per iteration (by factor N)
- Converges to the same kind of optimal point as the gradient descent
- Can be used in **online learning**

Stochastic gradient descent

- Use package **sgd** in R:
  - Linear model, GLM, Cox model,…

r.dioxide	density	pH	sulphates	alcohol	quality
170.0	1.0010	3.00	0.45	8.8	1
132.0	0.9940	3.30	0.49	9.5	1
186.0	0.9956	3.19	0.40	9.9	1
186.0	0.9956	3.19	0.40	9.9	1
97.0	0.9951	3.26	0.44	10.1	1
136.0	0.9949	3.18	0.47	9.6	1
170.0	1.0010	3.00	0.45	8.8	1

```
sgd.theta <- sgd(quality ~ ., data=dat,
                 model="glm",
model.control=binomial(link="logit"),
                 sgd.control=list(reltol=1e-5,
npasses=200))
sgd.theta
```

```
> sgd.theta
                [,1]
 [1,] -0.067246995
 [2,] -0.498257484
 [3,] -0.170646610
 [4,]  0.079520738
 [5,] -0.007496517
 [6,] -0.010871955
 [7,]  0.080419024
 [8,] -0.055379999
 [9,] -0.067940438
[10,] -0.104186087
[11,]  0.052921863
[12,]  0.766583940
> sgd.theta$converged
[1] TRUE
```

EM algorithm

- Model depends on the data which are known **Y** and data that can not be observed **Z** (latent).
- Data depend on some parameters $\boldsymbol{\theta}$

How to compute the ML of model parameters θ?

- All data known: Apply unconstrained optimization (for ex. Gradient descent)
  - Can not be used because gradient contains unknown data (latent variables)
- Now: Use **EM algorithm**

EM algorithm

Let

$$Q(\theta,\theta^t)=\int \log p(Y,Z\mid\theta)p(Z\mid\theta^t,Y)dz = E_{Z|\theta^t,Y}\text{loglik}(\theta\mid Y,Z)$$

EM algorithm

1. Choose starting point θ^t,
2. *E-step* : Derive $Q(\theta^t,\theta)$
3. *M-step*: $\theta^{t+1}=\arg\max_{\theta} Q(\theta^t,\theta)$, set $t=t+1$
4. Repeat until convergence

Example: Normal data with missing values

R: EM Algorithm

Example: Normal data with some missing observations

```
em.norm <- function(Y){
Yobs <- Y[!is.na(Y)]
Ymiss <- Y[is.na(Y)]
n <- length(c(Yobs, Ymiss))
r <- length(Yobs)
# Initial values
mut <- 1
sit <- 0.1
# Define log-likelihood function
ll <- function(y, mu, sigma2, n){
-0.5*n*log(2*pi*sigma2)-0.5*sum((y-mu)^2)/sigma2
}
# Compute the log-likelihood for the initial values
lltm1 <- ll(Yobs, mut, sit, n)
```

R: EM Algorithm

Example: Normal data with some missing observations

```
repeat{
# E-step
EY <- sum(Yobs) + (n-r)*mut
EY2 <- sum(Yobs^2) + (n-r)*(mut^2 + sit)
# M-step
mut1 <- EY / n
sit1 <- EY2 / n - mut1^2
# Update parameter values
mut <- mut1
sit <- sit1
# Compute log-likelihood using current estimates
llt <- ll(Yobs, mut, sit, n)
# Print current parameter values and likelihood
cat(mut, sit, llt, "\n")
# Stop if converged
if ( abs(lltm1 - llt) < 0.001) break
lltm1 <- llt
}
}
```

R: EM Algorithm

Example: Normal data with some missing observations

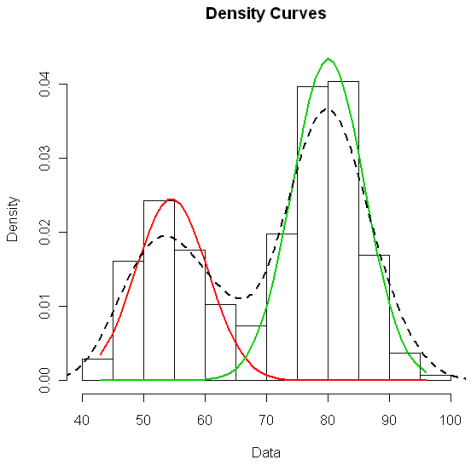
```
# Generate complete data N(5,1) and set 5 last as missing values
x <- rnorm(20,5)
x[16:20] <- NA
# Run the EM-algorithm function
em.norm(x)
3.856777 3.279395 -33.95757
4.570972 1.523877 -26.37452
4.74952 0.9255991 -23.4036
4.794157 0.7660673 -22.6868
4.805317 0.7255617 -22.53068
4.808106 0.7153964 -22.49403
4.808804 0.7128526 -22.48504
4.808978 0.7122165 -22.4828
4.809022 0.7120575 -22.48225

# Note that the function only generates parameter values
```

EM algorithm

- Applications
  - Mixture models (π_k is a latent variable)
    - In regression and classification
      - Mixed data comes from different sources
    - Clustering
      - Density in each cluster is normally distributed
      - Cluster label is latent

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$



Maximizing likelihood directly leads to numerical problems→ latent class variables are introduced and EM is used

EM algorithm for gaussian mixtures

1. Initialize the means $\boldsymbol{\mu}_k$, covariances $\boldsymbol{\Sigma}_k$ and mixing coefficients π_k, and evaluate the initial value of the log likelihood.
2. **E step**. Evaluate the responsibilities using the current parameter values

$$\gamma(z_{nk}) = \frac{\pi_k \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}. \tag{9.23}$$

3. **M step**. Re-estimate the parameters using the current responsibilities

$$\boldsymbol{\mu}_k^{\text{new}} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) \mathbf{x}_n \tag{9.24}$$

$$\boldsymbol{\Sigma}_k^{\text{new}} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) (\mathbf{x}_n - \boldsymbol{\mu}_k^{\text{new}}) (\mathbf{x}_n - \boldsymbol{\mu}_k^{\text{new}})^{\text{T}} \tag{9.25}$$

$$\pi_k^{\text{new}} = \frac{N_k}{N} \tag{9.26}$$

where

$$N_k = \sum_{n=1}^{N} \gamma(z_{nk}). \tag{9.27}$$

4. Evaluate the log likelihood

$$\ln p(\mathbf{X}|\boldsymbol{\mu}, \boldsymbol{\Sigma}, \boldsymbol{\pi}) = \sum_{n=1}^{N} \ln \left\{ \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\} \tag{9.28}$$

and check for convergence of either the parameters or the log likelihood. If the convergence criterion is not satisfied return to step 2.

$$E z_{nk} = \gamma(z_{nk})$$

Source: Pattern recognition by Bishop