1 Linear Classifier

Feature vectors x, labels y

$$x \in \mathbb{R}^d$$
$$y \in \{-1, 1\}$$

Training set

$$S_n = \{(x^{(i)}, y^{(i)}), i = 1, ..., n\}$$

Classifier

$$h : \mathbb{R}^d \to \{-1, 1\}$$
$$\chi^+ = \{x \in \mathbb{R}^d : h(x) = 1\}$$
$$\chi^- = \{x \in \mathbb{R}^d : h(x) = -1\}$$

Training error

$$\varepsilon_n(h) = \frac{1}{n}\sum_{i=1}^n \mathbf{1}\{h(x^{(i)}) \neq y^{(i)}\}$$

Test error (over disjoint set of examples)

$$\varepsilon(h)$$

Set of classifiers

$$h \in H$$

1.1 Linear classifiers through origin

Set of all points that satisfies a line through the origin.

$$\theta = \begin{bmatrix}\theta_1\\ \theta_2\end{bmatrix}$$
$$X = \begin{bmatrix}x_1\\ x_2\end{bmatrix}$$

Decision Boundary

$$\{x : \theta_1 x_1 + \theta_2 x_2 = 0\}$$
$$\{x : \theta \cdot X = 0\}$$

Linear Classifier through origin

$$h(x, \theta) = sign(\theta \cdot X)$$
$$\Theta \in \mathbb{R}^d$$

1.2 Linear classifiers

General linear Classifier (with Intercept)

$$\theta = \begin{bmatrix}\theta_1\\ \theta_2\end{bmatrix}$$
$$X = \begin{bmatrix}x_1\\ x_2\end{bmatrix}$$

Decision Boundary

$$\{x : \theta \cdot X + \theta_0 = 0\}$$

Linear Classifier through origin

$$h(x, \Theta, \theta_0) = sign(\theta \cdot X + \theta_0)$$
$$\theta \in \mathbb{R}^d$$
$$\theta_0 \in \mathbb{R}$$

1.3 Linear Separation

Traning examples $S_n = \{(x^{(i)}, y^{(i)}), i = 1, ..., n\}$ are linear separable if there exists a parameter vector $\hat{\theta}$ and offset parameter $\hat{\theta}_0$ such that $y^{(i)}(\hat{\theta} \cdot x^{(i)} + \hat{\theta}_0) > 0$ for all $i = 1, \cdots, n$.

$$(\hat{\theta} \cdot x^{(i)}) > 0 \begin{cases} y^{(i)} > 0 \text{ and } \theta \cdot x^{(i)} > 0 \\ y^{(i)} < 0 \text{ and } \theta \cdot x^{(i)} < 0 \end{cases}$$

$y^{(i)}(\theta \cdot x^{(i)}) > 0$ if label and classified result match. This leads to a new definition of the **Training error**:

$$\varepsilon_n(\theta) = \frac{1}{n}\sum_{i=1}^n \mathbf{1}\{y^{(i)}(\theta \cdot x^{(i)}) \leq 0\}$$
$$\varepsilon_n(\theta, \theta_0) = \frac{1}{n}\sum_{i=1}^n \mathbf{1}\{y^{(i)}(\theta \cdot x^{(i)} + \theta_0) \leq 0\}$$

1.4 Perceptron through Origin

Perceptron$\left(\left\{(x^{(i)}, y^{(i)}), i = 1, ..., n\right\}, T\right)$:

initialize $\theta = 0$ (vector);
for $t = 1, \cdots, T$ do
for $i = 1, \cdots, n$ do
if $y^{(i)}(\theta \cdot x^{(i)}) \leq 0$ then
update $\theta = \theta + y^{(i)}x^{(i)}$

1.5 Perceptron with Offset

Perceptron$\left(\left\{(x^{(i)}, y^{(i)}), i = 1, ..., n\right\}, T\right)$:

initialize $\theta = 0$ (vector); $\theta_0 = 0$ (scalar)
for $t = 1, \cdots, T$ do
for $i = 1, \cdots, n$ do
if $y^{(i)}(\theta \cdot x^{(i)} + \theta_0) \leq 0$ then
update $\theta = \theta + y^{(i)}x^{(i)}$
update $\theta_0 = \theta_0 + y^{(i)}$

1.6 Margin Boundary

The Margin Boundary is the set of points x which satisfy $\theta \cdot x + \theta_0 = \pm 1$. So, the signed distance from the decision boundary to the margin boundary is $\frac{1}{\|\theta\|}$.

$\frac{y^{(i)}(\theta \cdot x^{(i)} + \theta_0)}{\|\theta\|} = \frac{1}{\|\theta\|}$.

Hinge Loss (agreement)

$$Agreement = z = y^{(i)}(\theta \cdot x^{(i)} + \theta_0)$$
$$Loss_h(z) = \begin{cases} 0 \text{ if } z \geq 1 \\ 1 - z \text{ if } z < 1 \end{cases}$$

Regularization means pushing out the margin boundaries by adding $max(\frac{1}{\|\theta\|})$ or $min(\frac{1}{2}\parallel \theta \parallel^2)$ to the objective function.

Objective Function

Objective function = average loss + regularization

Objective function is minimized, learning becomes an optimization problem. Using hinge loss and margin boundaries is called **Support Vector Machine** or **Large margin linear classification**:

$$J(\theta, \theta_0) = \frac{1}{n}\sum_{i=1}^n \text{Loss}_h(z) + \frac{\lambda}{2}\parallel \theta \parallel^2.$$

Where $\lambda > 0$ is called the regularization parameter that regulates how important the margin boundaries are in comparison to the average hinge loss.

1.7 Gradient Descent

Assume $\theta \in \mathbb{R}$ the goal is to find θthat minimizes $J(\theta, \theta_0) = \frac{1}{n}\sum_{i=1}^n \text{Loss}_h(y^{(i)}(\theta \cdot x^{(i)} + \theta_0)) + \frac{\lambda}{2}\parallel \theta \parallel^2$ through gradient descent.

In other words, we will

- Start θ at an arbitrary location: $\theta \leftarrow \theta_{start}$
- Update θ repeatedly with $\theta \leftarrow \theta - \eta\frac{\partial J(\theta, \theta_0)}{\partial \theta}$ until θ does not change significantly.

Where $\eta > 0$ is called the stepsize or **learning parameter**.

1.8 Stochastic Gradient Descent

$$J(\theta, \theta_0) = \frac{1}{n}\sum_{i=1}^n \text{Loss}_h(z) + \frac{\lambda}{2}\parallel \theta \parallel^2$$
$$= \frac{1}{n}\sum_{i=1}^n \left[\text{Loss}_h(z) + \frac{\lambda}{2}\parallel \theta \parallel^2\right]$$

With stochastic gradient descent, we choose $i \in \left\{1, ..., n\right\}$ at random and update θ such that

$$\theta \leftarrow \theta - \eta \nabla_\theta\left[\text{Loss}_h(z) + \frac{\lambda}{2}\parallel \theta \parallel^2\right]$$

2 Linear Algebra

2.1 Distance

Consider a line L in $\mathbb{R}^2$ given by the equation $L : \theta \cdot x + \theta_0 =$ where θ is a vector normal to the line L. Let the point P be the endpoint of a vector x_0(so the coordinates of P equal the components of x_0).

The shortest distance d between the line L and the point P is:

$$d = \frac{|\theta \cdot x_0 + \theta_0|}{\|\theta\|}$$

3 Regression and classification

Classification:

$$S_n = \{(x^{(t)}, x^{(t)})|t = 1, \cdots, n\}$$
$$x^{(t)} \in \mathbb{R}^d, y^{(t)} \in \{-1, 1\}$$

Regression:

$$y^{(t)} \in \mathbb{R}$$
$$f(x, \theta, \theta_0) = \sum_{i=1}^d (\theta_i x_i + \theta_0) =$$
$$= \theta \cdot x + \theta_0$$

3.1 Objective for linear regression

The empirical risk R_n is defined as

$$R_n(\theta) = \frac{1}{n}\sum_{t=1}^n \text{Loss}(y^{(t)} - \theta \cdot x^{(t)})$$

where $(x^{(t)}, y^{(t)})$ is the tth training example (and there are n in total), and Loss is some loss function, such as hinge loss.

Possible to get **closed form solution** for gradient because function is concave. Only possible if the dxd matrix A is invertible. Computationally expensive if dimensions are very high like in bag of words approach.

$$\nabla R_n(\theta) = A\theta - b (= 0)$$
$$= A^{-1}b$$

where

$$A = \frac{1}{n}\sum_{t=1}^n x^{(t)}(x^{(t)})^T$$
$$b = \frac{1}{n}\sum_{t=1}^n y^{(t)}x^{(t)}$$

b is a vector with dimensionality d.

3.2 Gradient based Approach

Nudge gradient in the opposite direction to find (local) minima.

$$\nabla_\theta(y^{(t)} - \theta x^{(t)})^2/2 =$$
$$= (y^{(t)} - \theta x^{(t)})\nabla_\theta(y^{(t)} - \theta x^{(t)}) =$$
$$= -(y^{(t)} - \theta x^{(t)}) \cdot x^{(t)}$$

- initialize $\theta = 0$
- randomly pick $t = \{1, \cdots, n\}$
- $\theta = \theta + \eta(y^{(t)} - \theta x^{(t)}) \cdot x^{(t)}$

Where η is the learning rate (steps) and the learning rate gets smaller the closer you get $\eta_k = \frac{1}{1+k}$

3.3 Ridge Regression

Regularization is trying to push away from perfect fit.

$$J_{n,\lambda}(\theta, \theta_0) = \frac{1}{n}\sum_{t=1}^n \frac{(y^{(t)} - \theta \cdot x^{(t)} - \theta_0)^2}{2} + \frac{\lambda}{2}\|\theta\|^2$$
$$\nabla_\theta(J_{n,\lambda}) = \lambda\theta - (y^{(t)} - \theta x^{(t)})x^{(t)}$$

- initialize $\theta = 0$
- randomly pick $t = \{1, \cdots, n\}$
- $\theta = \theta + \eta\lambda\theta - (y^{(t)} - \theta x^{(t)})x^{(t)} = (1 - \eta\lambda)\theta + \eta(y^{(t)} - \theta \cdot x^{(t)})$

3.4 Kernels

$$\phi(x) = \left[x_1, x_2, {x_1}^2, \sqrt{2}x_1x_2, {x_2}^2\right]$$
$$\phi(x') = \left[x_1', x_2', {x_1'}^2, \sqrt{2}x_1'x_2', {x_2'}^2\right]$$
$$\phi(x) \cdot \phi(x') = x_1x_1' + x_2x_2' + x_1{x_1'}^2 + 2x_1x_1'x_2x_2' + x_2{x_2'}^2$$
$$= \left(x_1x_1' + x_2x_2'\right) + \left(x_1x_1' + x_2x_2'\right)^2$$
$$= \left(x_1x_1' + x_2x_2'\right) + \left(x_1x_1' + x_2x_2'\right)^2$$

3.5 Kernel Perceptron

The parameter vector of a preceptron algorithm can also be written as:

$$\theta = \sum_{j=1}^n \alpha_j y^{(j)}\phi\left(x^{(j)}\right)$$

Where α_j represents the number of classification mistakes the perceptron algo made. Every time a missclassification happens the parameter vector is updated with the product of the label and the feature vector $\theta = \theta + y^{(i)}x^{(i)}$.

The goal of the Kernel Perceptron algo is to find the vector α with the counts of the missclassifications.

Kernel Perceptron$\left(\left\{(x^{(i)}, y^{(i)}), i = 1, ..., n, T\right\}\right)$

initialize $\alpha_1, \alpha_2, ..., \alpha_n$; to some values
for $t = 1, \cdots, T$ do
for $i = 1, \cdots, n$ do
if $y^{(i)}\sum_{j=1}^n \alpha_j y^{(j)}K(x^j, x^i) \leq 0$ then
update $\alpha_i = \alpha_i + 1y^{(i)}$

3.6 Radial basis Kernel

$$K(x, x') = e^{-\frac{1}{2}\|x - x'\|^2}$$

4 Recommender Systems

4.1 K nearest neighbors

The K-Nearest Neighbor method makes use of ratings by K other "similarüsers when predicting Yai .

Let $KNN(a)$ be the set of K users "similar toüser a , and let $sim(a, b)$ be a similarity measure between users a and $b \in KNN(a)$. The K -Nearest Neighbor method predicts a ranking Yai to be :

$$\widehat{Y}_{ai} = \frac{\sum_{b \in \text{KNN}(a)} \text{sim}(a, b)Y_{bi}}{\sum_{b \in \text{KNN}(a)} \text{sim}(a, b)}.$$

The similarity measure $sim(a, b)$ could be any distance function between the feature vectors xa and x_b of users a and b , e.g. the euclidean distance $\|x_a - x_b\|$ and the cosine similarity $c\cos\theta = \frac{x_a \cdot x_b}{\|x_a\|\|x_b\|}$.

4.2 Collaborative Filtering

Matrix Y with n rows (users) and m columns (Movies) is sparse (entries missing), (a, i)th entry $Y_a i$ is the rating by user a of movie i if this rating has already been given, and blank if not. Goal is to predict matrix X with no missing entries.

Let D be the set of all (a, i) 's for which a user rating $Y_a i$ exists, i.e. $(a, i) \in D$ if and only if the rating of user a to movie i exists.

$$J = \sum_{(a,i) \in D} \frac{(Y_{ai} - \left[UV^T\right]_{ai})^2}{2} + \frac{\lambda}{2}\left(\sum_{a,k} U_{ak}^2 + \sum_{i,k} V_{ik}^2\right)$$

$$u = \begin{bmatrix}u_1\\ u_2\end{bmatrix}; v = \begin{bmatrix}v_1\\ v_2\\ v_3\end{bmatrix}$$

$$v = \begin{bmatrix}2\\ 7\\ 8\end{bmatrix}$$

$$uv^T = \begin{bmatrix}2u_1 7u_1 8_u 1\\ 2u_2 7_u 18_u 2\end{bmatrix}$$

Take derivative of Objective function J with respect to every user, set it to zero and find respective u_i value:

$$\frac{d}{du_1}\left(\frac{(7 - 8u_1)^2}{2} + \frac{\lambda}{2}u^2\right) = 0$$
$$\frac{d}{du_2}(J) = 0$$
$$u_1 = \frac{66}{\lambda + 68}; u_2 = \frac{16}{\lambda + 53}$$

Use resulting values for u to compute uv^T compare resulting matrix X with matrix Y and start again. Continue until convergence.

5 Clustering

Two Views:

Clustering input: $S_n = \{x^{(i)}|n = 1, \cdots, n\}$

Clustering output are indexes for the data that partition the data: $C_1, \cdots, C_k$; where $C_1 \cup C_2 \cup ... \cup C_K = \left\{1, 2, ..., n\right\}$ and the union of all C_j 's is the original set and the intersection of any C_i and C_j is an empty set.

Representatives of clusters: $z^{(1)}, \cdots, z^{(k)}$.

Cost of partitioning is the sum of costs of individual clusters: $cost(C_1, \cdots, C_k) = \sum_{j=1}^k cost(C_j)$.

Cost of cluster is sum of distances from data points to the representative of the cluster:$Cost(C, z) = \sum_{i \in C} = distance(x^{(i)}, z)$

Cosine similarity: $cos(x^{(i)}, x^{(j)}) = \frac{x^{(i)} \cdot x^{(j)}}{\|x^{(i)}\|\|x^{(j)}\|}$ is not sensitive of magnitude of vector (will not react to length).

Euclidean square distance: $dist(x^{(i)}, x^{(j)}) = \|x^{(i)} - x^{(j)}\|^2$. Will react to length.

$cost(C_1, \cdots, C_k; Z^{(1)}, \cdots, Z^{(1)}) = \sum_{j=1}^k \sum_{c \in C_j} \|x^{(i)} - z^{(j)}\|^2$

5.1 The K-Means Algorithm

Only works with Euclidean square distance.

Given a set of feature vectors $S_n = \left\{x^{(i)}|i = 1, ..., n\right\}$ and the number of clusters K we can find cluster assignments $C_1, \cdots, C_K$ and the representatives of each of the K clusters $z_1, \cdots, z_K$:

1. Randomly select $z_1, \cdots, z_K$
2. Iterate
   (a) Given $z_1, \cdots, z_K$, assign each data point $x^{(i)}$ to the closest z_j, so that $\text{Cost}(z_1, ... z_K) = \sum_{i=1}^n \min_{j=1,...,K}\left\|x^{(i)} - z_j\right\|^2$
   (b) Given $C_1, \cdots, C_K$ find the best representatives $z_1, \cdots, z_K$, i.e. find $z_1, \cdots, z_K$ such that $z_j = \text{argmin}_z \sum_{i \in C_j} \|x^{(i)} - z\|^2$

The best representative is found by optimization (gradient with respect to $z^{(j)}$, setting to zero and solving for $z^{(j)}$).

It is the centroid of the cluster: $z^{(j)} = \frac{\sum_{i \in C_j} x^{(i)}}{|C_j|}$

The clustering output that the K-Means algorithm converges to depends on the intialization.

5.2 K-Medoids Algorithm

Finds the cost-minimizing representatives $z_1, \cdots, z_K$ for any distance measure. Uses real data points for initialization.

1. Randomly select $\left\{z_1, ..., z_K\right\} \subseteq \left\{x_1, ..., x_n\right\}$
2. Iterate
   (a) Given $z_1, \cdots, z_K$, assign each data point $x^{(i)}$ to the closest z_j, so that $\text{Cost}(z_1, ... z_K) = \sum_{i=1}^n \min_{j=1,...,K}\left\|x^{(i)} - z_j\right\|^2$
   (b) Given $C_j \in \left\{C_1, ..., C_K\right\}$ find the best representative $z_j \in \left\{x_1, ..., x_n\right\}$ such that $\sum_{x^{(i)} \in C_j} \text{dist}(x^{(i)}, z_j)$ is minimal

6 Generative Models

Understand structure of data probabilistically.

6.1 Multinominal Models

Fixed Vocabulary W

Multinomial model M to generate text in documents.

Document D

Likelihood of generating certain word $w \in W$: $p(w|\theta) = \theta_w$ where $\theta_w \geq 0$ and $\sum_{w \in W}\theta_w = 1$.

Likelihood function:

$$P(D|\theta) = \prod_{i=1}^n \theta_{wi}$$
$$= \prod_{w \in W}\theta_w^{count(w)}$$

Toy Example:

$$\theta_1 : \theta_{cat} = 0.3; \theta_{dog} = 0.7$$
$$\theta_2 : \theta_{cat} = 0.9; \theta_{dog} = 0.1$$
$$D = \{cat, cat, dog\}$$
$$P(D|\theta_1) = 0.3^2 \cdot 0.7 = 0.063$$
$$P(D|\theta_2) = 0.9^2 \cdot 0.1 = 0.081$$

Maximum likelihood:

$$max_\theta P(D|theta) = max_{theta}\prod_{w \in W}\theta_w^{count(w)}$$
$$log\prod_{i=1}^n \theta_w^{count(w)} = \sum_{w \in W} count(w)log(\theta_w)$$
$$W = \{0, 1\}; \theta_0 = \theta; \theta_1 = (1 - \theta)$$
$$\frac{d}{d\theta}(count(0)log(\theta) + count(1)log(1 - \theta) =$$
$$= \frac{count(0}{\theta} - \frac{count(1}{1 - \theta} = 0$$
$$\hat{\theta} = \frac{count(0)}{count(1) + count(0)}$$

For any length of W:

$$\hat{\theta} = \frac{count(w)}{\sum_{w' \in W} count(w)}$$

6.2 Prediction

Goal: categorize between minus and plus class. Both classes have a associated parameter θ^+ and θ^-

Class conditional distribution:

$$log\left(\frac{P(D|\theta^+)}{P(D|\theta^-)}\right) = \begin{cases} \geq 0, + \\ < 0, - \end{cases}$$

Model is the same as a linear classifier through origin:

$$log(P(D|\theta^+)) - log(P(D|\theta^-)) =$$
$$= log\prod_{w \in W}\theta_w^{+count(w)} - log\prod_{w \in W}\theta_w^{-count(w)} =$$
$$= \sum_{w \in W} count(w)log(\theta_w^{+count(w)}) - \sum_{w \in W} cou$$
$$= \sum_{w \in W} count(w)log\frac{\theta_w^{+count(w)}}{\theta_w^{-count(w)}}$$
$$= \sum_{w \in W} count(w)\tilde{\theta}_w$$

6.3 Prior, Posterior and Likelihood

From bayes rule $P(A|B) = \frac{P(B|A)P(A)}{P(B'B)}$ we get:

$$P(y=+|D) = \frac{P(D|\theta^+)P(y=+)}{P(D)}$$

Where $P(y=+|D)$ is the posterior distribution and $P(y=+)$ is the prior distribution while $P(D|\theta+)$ is the likelihood of document D given parameter θ^+. This yields (after some work) a linear separator with offset:

$$log(\frac{P(y=+|D)}{P(y=-|D)} = \frac{P(D|\theta^+)P(y=+)}{P(D|\theta^-)P(y=-)}$$
$$= log(\frac{P(D|\theta^+}{P(D|\theta^-}) + log(\frac{P(y=+)}{P(y=-)})$$
$$= \sum_{w \in W} count(w)\tilde{\theta}_w + \tilde{\theta}_0$$

6.4 Gaussian Generative models

Vectors in $x \in R^d$ "cloudöf data in which μ (average over all points) is the center of the cloud and σ^2 (square of average distance) the radius.
Probability of x generated by gaussian cloud:

$$P(x|\mu,\sigma^2) = \frac{1}{(2\pi\sigma^2)^{d/2}} exp(-\frac{1}{2\sigma^2}||x-\mu||^2)$$

Likelihood of the training data: $S_n = \{x^{(t)}|t = 1, \cdots, n\}$ given the gaussian model $p(S_n|\mu,\sigma^2) = \prod_{t=1}^n P(x^{(t)|\mu,\sigma^2}$.
To get the MLE calculate likelihood, take the log and massage:

$$log(\prod_{t=1}^n \frac{1}{(2\pi\sigma^2)^{d/2}} exp(-\frac{1}{2\sigma^2}||x-\mu||^2)) =$$
$$= \sum_{t=1}^n log\frac{1}{2\sigma^2} + \sum_{t=1}^n log(exp(-\frac{1}{2\sigma^2}||x-\mu||^2))$$
$$= \sum_{t=1}^n(-\frac{d}{2}log(2\pi\sigma^2)) + \sum_{t=1}^n(-\frac{1}{2\sigma^2}||x-\mu||^2)$$
$$= -\frac{nd}{2}log(2\pi\sigma^2) + \frac{1}{2\sigma^2}\sum_{t=1}^n ||x-\mu||^2)$$
$$= L$$

Differentiate loglikelihood with respect to μ and σ^2 set to zero and solve for the respective parameters yields:

$$\hat{\mu} = \frac{\sum_{t=1}^n x^{(t)}}{n}$$
$$\hat{\sigma}^2 = \frac{\sum_{t=1}^n ||x^{(t)}-\mu||^2}{nd}$$

6.5 Gaussian Mixture Models

Is called SSoft Clustering"because it deals with probabilities not hard classification.

We have K clusters, each with own gaussian cloud $N(x, \mu^{(j)}, \sigma^2_{(j)}), j = 1, \cdots, K$.
Each Cluster gets own mixture-weight $j \sim Multinomial(p_1, \cdots, p_k)$
Parameters of the mixture model are parameters of Multinomials and gaussians:

$$\theta = p_1, \cdots, p_k; \mu^{(1)}, \cdots, \mu^{(k)}; \sigma^2_{(j)}), \cdots, \sigma^2_{(j)})$$

Conditional probability of data-point given gaussian mixture:

$$P(x|\theta) = \sum_{i=1}^K p_j N(x, \mu^{(j)}, \sigma^2_{(j)}$$

Conditional Likelihood of Training set S_n given gaussian mixture:

$$P(S_n|\theta) = \prod_{j=1}^n \sum_{j=1}^k N(x, \mu^{(j)}, \sigma^2_{(j)}$$

Observed Case:
We know to which mixture x belongs.

Indicator Variable is used to count the cases in which observation is part of a cluster $\delta(j|i) = \mathbf{1}(x^{(i)}$is assinged to j).

$$\sum_{i=1}^n [\sum_{j=1}^k \delta(j|i) log(p_j N(x, \mu^{(j)}, \sigma^2_{(j)})] =$$
$$= \sum_{j=1}^k [\sum_{i=1}^n \delta(j|i) log(p_j N(x, \mu^{(j)}, \sigma^2_{(j)})]$$

Optimizing (according to MLE principle) yields:

$$\hat{n}_j = \sum_{i=1}^n \delta(j|i)$$
$$\hat{p}_j = \frac{\hat{n}_j}{n}$$
$$\hat{\mu}^{(j)} = \frac{1}{n}\sum_{i=1}^n \delta(j|i) \cdot x^{(i)}$$
$$\hat{\sigma}^2 = \frac{1}{\hat{n}_j}\sum_{i=1}^n \delta(j|i)||x^{(i)} - \mu^{(j)}||^2$$

EM Algorithm (Unobserved Case):
We don't know to which mixture x belongs.

1. Randomly initialize $\theta = p_1, \cdots, p_k; \mu^{(1)}, \cdots, \mu^{(k)}; \sigma^2_{(j)}), \cdots, \sigma^2_{(j)})$
2. E-Step:
   (a) Calculate the softcount of a point (the probability of a cluster j given the point i: $p(j \mid i) = \frac{p_j\ \mathcal{N}(x^{(i)}; \mu^{(j)}, \sigma_j^2)}{p(x^{(i)}|\theta)}$, where $P(x|\theta) = \sum_{j=1}^K p_j N(x, \mu^{(j)}, \sigma^2_{(j)}$
3. M-Step
   (a) Use softcounts to calculate new parameters.

$$\hat{n}_j = \sum_{i=1} p(j|i)$$
$$\hat{p}_j = \frac{\hat{n}_j}{n}$$
$$\hat{\mu}^{(j)} = \frac{1}{n}\sum_{i=1}^n p(j|i) \cdot x^{(i)}$$
$$\hat{\sigma}^2_j = \frac{1}{\hat{n}_j}\sum_{i=1}^n p(j|i)(x^{(i)} - \mu^{(j)})^2$$

6.6 Reinforcement Learning

A **Markov decision process (MDP)** is defined by
a set of states $s \in S$ a set of actions $a \in A$;
Action dependent transition probabilities $T(s,a,s') = P(s'|s,a)$, so that for each state s and action a , $\sum_{s' \in S} T(s,a,s') = 1$.
Reward functions $R(s,a,s')$ representing the reward for starting in state s , taking action a and ending up in state s' after one step. (The reward function may also depend only on s , or only s and a .)
Therefore a Markov decision process is defined by $MDP =< S, A, T, R >$ MDPs satisfy the Markov property in that the transition probabilities and rewards depend only on the current state and action, and remain unchanged regardless of the history (i.e. past states and actions) that leads to the current state.
Rewards collected after the nth step do not depend on the previous states $s_1, s_2, \cdots, s_{n-1}$
Markov properties:

Rewards collected after the nth step do not depend on the previous actions $a_1, a_2, \cdots, a_n$
(Infinite horizon) discounted reward based utility

$$U[s_0, s_1, \ldots] = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) \ldots =$$
$$= \sum_{t=0}^\infty \gamma^t R(s_t) \text{where } 0 \leq \gamma < 1$$
$$\leq \frac{R_max}{\gamma}$$

Bellman Equations

$$V^*(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

$$Q^*(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$$

Q-value: $Q(s,a)$ in state s take action a and act optimally afterwards.

Policy $\pi^* : s \to a$ is set of actions to maximize the expected reward for every state s.

$$\pi^*(s) = argmax_a(Q^*(s,a)$$
$$Q^*(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma max_a Q(s',a')]$$

To find the policy two algos: Value iteration and Q-value iteration (look online).

6.7 Q value iteration by sampling