01.112 Machine Learning
Finals Revision

Tey Siew Wen

06 Feb 2020

Preface

Beware notes for these concepts were **not written**, as I felt that they were pretty self-explanatory when you learn them.

- Neural Networks
- Generative Models

Contents

1 Background Information

1.1 What is Machine Learning

Machine learning revolves around **models**, which are trained via certain **techniques** and **learning algorithms** that are based on specific **loss functions**. They produce outputs based on **predictor functions**.

1.2 Recap on Convex Functions

A convex function typically has a U-shape (there are boundary cases where the surface can be flat, though. In that case the function is both convex and concave).

Properties:

- $\frac{f(x_1) + f(x_2)}{2} \geq f\left(\frac{x_1 + x_2}{2}\right)$
- Its local optimum is also a global optimum.
- Sum of convex functions is also convex.

Some examples:

- $f(x) = (x-1)^2$
- $f(x) = \max(x^2, 2^x)$
- $f(x) = |x| - x$

The convexity of empirical risk allows us to find the minimum even in non-realizable case.

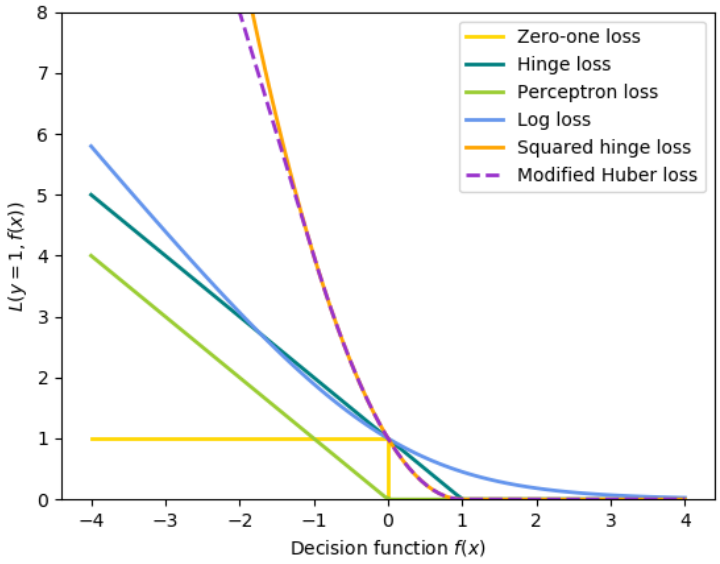
1.3 Empirical Risk

Defined as the average loss on the training examples.

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

1.4 Loss Functions

Loss function	**Loss**$_h(z)$
Squared Euclidean/ Least Squares	$\frac{1}{2}z^2$
Zero-one loss	
Hinge Loss	$\max(1-z, 0)$



1.5 Types of Machine Learning

> Notice the resemblance in the problems from each category of tasks.

Supervised Tasks

- Classification $f : R^n \to 1, \ldots, k$
  - LR, SVM, Perceptron NB
- Regression: $f : R^n \to R^m$
  - Ridge Regression, Logistic Regression
- Structured Prediction: $f : S_1 \to S_2$ where $\text{size}(S_1) > \text{size}(S_2)$
  - HMM, CRF, Structured Perceptron, Structural SVM

Unsupervised Tasks

- Clustering: $f : R^n \to 1, \ldots, k$
  - K-means, EM
- Dimensionality Reduction: $f : R^n \to R^m$
  - PCA, LDA, Autoencoder
- Structured Prediction: $f : S_1 \to S_2$
  - Unsupervised HMM

2 Introduction to Machine Learning

algorithms | tasks | performance | experience

Definition of Machine Learning: ML is programming computers to optimize a **performance criterion** using example data or past experience.

ML has been studied since computers came out, but back then the amount of existing data is insufficient to do any useful ML, so it died quickly by 1970s. It is only now in the Big Data, IOT age that data is ever so abundant,and with enhanced computational power, ML can be applied meaningfully to these data to solve real-world problems that were previously too complex to be analyzed/ computationally difficult. Using ML, we aim to build models that are useful **approximations** of the relationship between input data and output data. After all, we can never tell if there is ever an absolute relationship between any sets of data.

We usually denote each feature (column) vector of dimension d as such:

$$x = x[x_1, \ldots, x_d]^T \subset R^d$$

$$x^{(1)}, x^{(2)}, \ldots, x^{(n)} : \text{training examples}$$

$$y^{(1)}, y^{(2)}, \ldots, y^{(n)} : \text{the corresponding labels}$$

If x is used to denote the original object (e.g. image), then we would use the following notation to denote the feature vector:

$$\phi(x) \subset R^d$$

2.1 Key Aspects of Learning Problems

1. Data: could be Biased
2. Modelling: Set of Classifiers $H(x)$
3. Optimizing: Learning Algorithm / Performance Criterion
    - Each algorithm comes with its assumptions

3 Types of Machine Learning

3.1 Based on Learner's role

- Passive Learning
  - Traditionally, learning algorithms have been passive learners in which they just take the data to produce hypothesis/model.
- Active Learning
  - Active Learners can query the environment by asking questions/ performing experiments.
  - But it is difficult to account for the cost of queries/ query the environment optimally.

3.2 Based on Information/Dataset Available

1. ***Supervised Learning:*** using labelled data
   - Classification (discrete values)
   - Regression (real values)
2. ***Unsupervised Learning:*** using unlabelled data
   - Clustering
   - Probability Distribution Estimation
   - Finding Association In Features
   - Dimension Reduction
3. ***Semi-supervised Learning:*** using a small subset of labelled data with a large amount of unlabelled data for learning
4. ***Reinforcement Learning:*** Using state spaces
   - Decision Making (robotics, games like DOTA)
5. ***Transfer Learning:*** Reusing trained filters for other models

3.2.1 Supervised Learning

Classification
Steps to solving a supervised learning problem

1. Decide what the input-output pairs
2. Decide how to encode inputs and outputs
3. Choose a class of hypothesis (modelling)
4. Choose error function to define best hypothesis
5. Choose algorithm for searching efficiently through the space of hypotheses (optimizing)

3.2.2 Unsupervised Learning

Clustering
Need to know approximately how many clusters of data is expected.

Dimensionality Reduction: Sub-space Learning

The importance of Dimensions

Say Boy A is training for IPPT and he records all the timings he takes to train 2.4km. It will result in 1D plot on the number line. But if he adds in the readings by his friend and have his records be differentiated, the result will be a 2D plot. If he adds a 3rd friend, the graph will become a 3D plot. The 4th friend onwards will be a Nth-D plot but it cannot be visualized anymore.

For a fitness trainer that sees data like this, sometimes he doesn't really care who ran how long but he just wants to see the duration of each practice run that has taken place and how the data varies. So for such a problem, we can perform **Principal Component Analysis** to reduce the dimensions involved and still get useful information out it.

Principal Component Analysis

Definition of PCA: A statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables (entities each of which takes on various numerical values) into a set of values of **linearly uncorrelated** variables called *principal components*.

> PCA works optimally only in the situation where the correlations are linear, which is most of the time an approximation. Otherwise do something to transform the data to linear variables or perform Nonlinear dimensionality reduction.

- As described in the IPPT example, PCA can be used to convert the Original Data Space to Component Space.
- These principal components mark the axes for the new plot with reduced dimensions. The first principal component (PC1) is the axis that spans the most variation in data, followed by PC2 with the 2nd most variation and so on.

Orthogonal transformation is performed via Eigenvalue decomposition of a data covariance/ correlation matrix (must be diagonalizable):

Let A be a square n × n matrix with n linearly independent eigenvectors q_i (where i = 1, ... , n).
Then A can be factorized as

$$Av = \lambda v$$

$$Q = Q$$

$$A = Q\lambda Q^{-1}$$

Q: square n × n matrix whose ith column is the eigenvector of A

A: the diagonal matrix whose diagonal elements are the corresponding eigenvalues $A_{ii} = \lambda_i$.

3.2.3 Reinforcement Learning

- *Curse of Dimensionality:* all data points appear to be sparse and dissimilar in many ways, which prevents common data organization strategies from being efficient. This results in more complicated control problems to not fit enumerable states even if we discretize them.

- *Assumes Markov Property:* The future does not depend on the past. The conditional probability distribution of future states of the process (conditional on both past and present values) depends only upon the present state.
    - If there are consistent unknown factors that influence result, and could logically be deduced - maybe from history of state or actions - but are excluded from the state representation, the agent may fail to learn.

4 Why Study ML

4.1 Engineering reasons for studying ML

- Improving existing programs via:
    - Instruction scheduling and register allocation in compilers
    - Combinatorial Optimization Problems
- Solving tasks that require adaptive systems
    - where humans can perform the task but cannot explain how e.g. speech/handwriting recognitiion, intelligent UIs
    - desired function changes frequently e.g. predict stock prices
    - each user needs a customized function e.g. news filtering
- We don't want to hand code a lot of programs
- Performing tasks where there is no human expert

4.2 Scientific reasons for studying ML

- Discover knowledge & patterns in highly dimensional, complex data
- Understanding the process of learning

5 Linear Classification

A classifier h partitions space into decision regions that are separated by decision boundaries. In each region, all the points map to the same label. For linear classifiers, these regions are half spaces.

A linear classifier h takes the following form

$$h(x;\theta) = \text{sign}(\theta^T x) = \text{to add}$$

5.1 Algorithms

5.1.1 Perceptron Update Rule (Mistake-Driven Updates)

Theorem
The perceptron update rule converges after a finite number of mistakes when the training examples are linearly separable through origin. This implies that zero training error can be achieved using perceptron update rule for linearly separable training examples.

> The algorithm will not converge for non-linearly separable data! Refer to Stochastic (Sub)-Gradient Descent instead for such data.

How it works
Initialize weight $\theta = 0$. For each training sample t in S_n, classify the instance with mistake driven updates. The Perceptron Update Rule will terminate when:

- Training error is 0 (Realizable)
- Predetermined number of iterations is completed (Non-realizable)

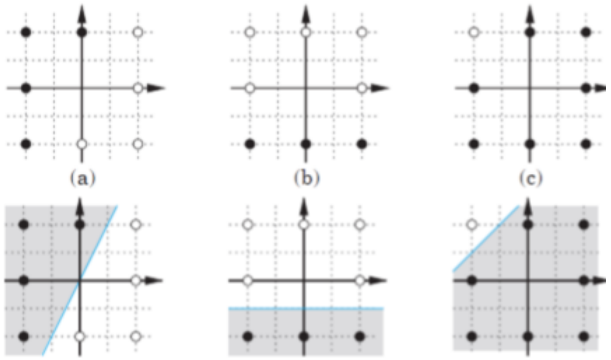
A mistaken is recognized with the following condition and the weight is updated as such:

$$\text{Mistake Condition: } y^t(\theta \cdot x^t) \leq 0 \quad \text{Update Rule:} \quad \theta^{(k+1)} = \theta^{(k+1)} = y^{(t)}x^{(t)}$$

$y^t(\theta \cdot x^t)$ then becomes more positive and eventually becomes > 0 (Realizable). If the prediction is correct, continue.

Offset

- Set default prediction to +1/-1.
- The hyper-plane $\theta \cdot x + \theta_0$ is parallel to
- $\theta \cdot x = 0$. θ is still orthogonal to the decision boundary
- For training examples that are linearly separable through origin, they would also be linearly separable with offset. However, the converse is not true.



5.1.2 Gradient Descent

Instead of checking if the value is classified wrongly, gradient descent aims to minimize the loss $R_n(\theta)$.

$$\nabla_\theta R_n(\theta) = \frac{1}{n}\sum_{t=1}^{n} \max\{1 - y^{(t)}(\theta \cdot x^{(t)}), 0\}$$

We minimize the loss by finding $\nabla_\theta R_n(\theta)$.

$$\begin{aligned}\nabla_\theta R_n(\theta) =& \nabla_\theta \text{Loss}_h(y^{(t)}\theta \cdot x^{(t)}) \\ =& \nabla_\theta(1 - y^{(t)}\theta \cdot x^{(t)}) \\ =& -y^{(t)}x^{(t)}\end{aligned}$$

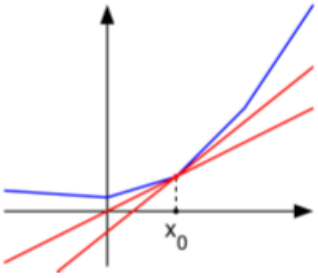
For values that are classified wrongly, the gradient will point in the direction where the error $R_n(\theta)$ increases, so we update the weight in the opposite direction and descend from the point accordingly. We will also add a learning rate α to control the rate of descent.

Thus the weight will be updated as such

$$\theta^{(k+1)} = \theta^{(k)} - \alpha_k \nabla_\theta R_n(\theta)_{\theta=\theta^{(k)}} \quad \theta^{(k+1)} = \quad \theta^{(k)} + \alpha_k y^{(t)} x^{(t)}$$

Limitations

- However, $R_n(\theta)$ is not differentiable everywhere as hinge loss functions are piece-wise linear.
- There are several possible gradients at the kinks, which are collectively defined as sub-differential.
- To minimize $R_n(\theta)$, we randomly select one possible gradient.



If there are many training examples, it is better to use Stochastic Gradient Descent (SGD).

5.1.3 Stochastic (Sub)-Gradient Descent

- Random sample avoid oscillations
- Keep track of best solution (the weights where the loss is minimum).
- Near mistakes are penalized
- Decreasing learning rate, for smaller updates, $\alpha_k = 1/(k+1)$

How it works

1. Initialize weight $\theta = 0$
2. Select random data point, t.
3. If $y^{(t)}\theta^{(k)} \cdot x^{(t)} \leq 1$ then update weight.
4. Repeat steps 2-3 until stopping criterion is met.

5.1.4 Closed Form Solution

Using least squares criterion, we can also minimize empirical risk directly by setting gradient to zero.

$$\nabla_\theta R_n(\theta) = -b + A\theta$$

$$\text{where } b = \frac{1}{n}\sum_{t=1}^{n} y^{(t)}x^{(t)} = \frac{1}{n}X^T Y, \quad A = \frac{1}{n}\sum_{t=1}^{n} x^{(t)} \cdot (x^{(t)})^T = \frac{1}{n}X^T X$$

If A is invertible, we can find the weight directly:

$$\theta = A^{-1}b \quad = \left(\frac{1}{n}X^T X\right)^{-1}\left(\frac{1}{n}X^T \vec{y}\right)$$

- Note the cost of inverting matrix A is usually $O(d^3)$ for normal gradient descent.

If A is not invertible, the training data will not be able to guide the model on setting parameter directions. Hence, we can choose to modify the estimation criterion, the mean squared error in this case, by adding a regularizatioon term. The regularization term λ shifts emphasis away from the training data.

- Larger values of λ will result in larger training error, but lower generalization error $J_n(\theta)$ as we are less swayed by noisy data. So, it is harder to over-fit to training data.
- But as λ increases too much, it will bias the parameters too strongly towards zero.

Let's choose $\frac{||\theta||^2}{2}$ as the penalty for easier solvability. This means that we will aim to minimize

$$J_{n,\lambda} = \frac{\lambda}{2}||\theta||^2 + R_n(\theta)$$

This is known as Ridge Regression.

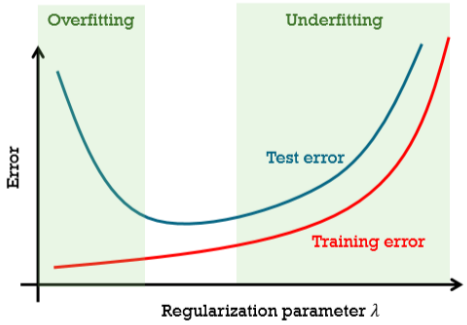
5.1.5 Ridge Regression

Using Gradient Descent

Using GD/SGD for learning parameters in this technique, the weight will be updated as such

$$\theta^{k+1} = (1 - \lambda\alpha_k)\theta^k + \alpha_k(y^{(t)} - \theta \cdot x^{(t)})x^{(t)}$$

The new factor $(1 - \lambda\alpha_k)$ helps to shrink the parameters $\theta^{(k)}$ towards zero during each update. Without regularization i.e. $\lambda = 0$, it will look the same as the original gradient descent.



Closed Form Solution

The regularization term only modifies the matrix $A = \lambda I + \left(\frac{1}{n}\right) X^T X$, where I is the identity matrix.

$$\theta = (n\lambda I + X^T X)^{-1} X^T Y$$

The identity matrix is always invertible if $\lambda > 0$.

6 Clustering

- Unsupervised learning
- Cluster: collection of similar data points, and is sufficiently different from other groups.
- Criterion for selecting clusters & representatives
  - Cosine Similarity: $\frac{x^{(i)} \cdot x^{(j)}}{||x^{(i)}|| \cdot ||x^{(j)}||}$
  - Euclidean Distance: $||x^{(i)}|| - ||x^{(j)}||$

6.1 K-means

6.1.1 Cost of Clustering

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

$$\text{cost}(z^{(1)} \dots z^{(k)}) = \sum_{i=1\dots n} \min_{j=1\dots k} ||x^{(i)} - z^{(j)}||^2$$

6.1.2 How it works

1. Choose the number of clusters k
2. Place the centroids $c_1, c_2, \dots c_k$ randomly
3. For each data point x_i:
   - Find the nearest centroid $(c_1, c_2 \dots c_k)$
   - Assign the point to that cluster
4. For each cluster j = 1..k
   - New centroid = mean of all points assigned to that cluster

$$z^{(j)} = \frac{1}{|C_j|} \sum_{i \in C_j} x^{(i)}$$

5. Repeat steps 4 and 5 until
   - convergence (when we find a clustering that minimizes the cost of clustering)
   - or until the end of a fixed number of iterations.

Each iteration requires $O(kn)$ operations, and necessarily lowers the cost of clustering, the sum of costs of individual clusters.

6.2 K-medoids

6.2.1 Choosing k

We should choose the value of k that results in the highest relative drop in the cost (which corresponds to an "elbow" in the graph capturing as a function of k).

6.2.2 How it works

1. Choose the number of clusters k
2. Choose the exemplars $z_1, z_2, \ldots, z_k$ randomly out of the data points (or given otherwise)
3. For each data point x_i, other than the centroids:
   - Calculate the distance to each exemplar $z_1, z_2, \ldots, z_k$
   - Add the data point to the cluster that contains $z^{(j)}$ which is nearest to the point x_i.
4. Set new exemplar $z^{(j)}$ to be the point in C^j that minimizes $\sum_{i \in C^j} d(x^{(i)}, z^{(j)})$
5. Repeat steps 3-4 until no further change in cost.

6.3 Difference between K-means and K-medoids

K-medoids	K-means
Applicable to arbitrary objects and distance functions (e.g. categorical data)	Not applicable for arbitrary objects and distance functions
Less sensitive to noisy data $\Rightarrow$ Mediod is less influenced by outlier than a mean.	More sensitive to noisy data
Applicable when true data points are unavailable and only their pair-wise distances are provided	Not applicable if true data points are not available
Applicable to both continuous and discrete domains	Only applicable to the continuous domain since the mean is not necessarily a data point.
Longer run time, especially for large and random datasets. Harder than computing the average.	Shorter run time
Has no statistical meaning; Neither a median nor geometric median	Has a true geometrical and statistical meaning

6.3.1 Similarities between K-means and K-medoids

- Each iteration decreases the cost
- The algorithm always converges
- Different starts gives different final answers
- It does not achieve the global minimum

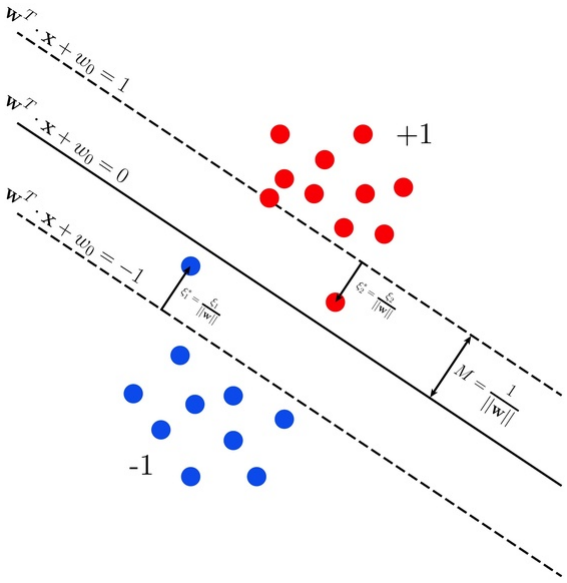
7 Support Vector Machines (SVMs)

7.1 Adding Slack Variables

Sometimes we get data which has two classes that are mostly separated except some small training data where the two categories overlap. In such cases, we would like to allow some points to intentionally be misclassified so as classify the rest correctly.

To do so, for each training data point we can define a variable that measures the distance of the point to its marginal hyperplane, ξ_t, in terms of the hinge loss function.

$$\begin{aligned}\xi_t &= \text{Loss}_h(y^{(t)}(\theta \cdot x^{(t)} + \theta_0)) \\ &= \max\{1 - y^{(t)}(\theta \cdot x^{(t)}), 0\}\end{aligned}$$



We can define a separating hyperplane by minimizing such errors ξ_i:

$$\min_{\theta,\theta_0,\xi} \sum_{i=1}^{N} \xi_i$$

Each individual training point has a different but parallel marginal hyperplane.

$$\begin{aligned}\theta \cdot x + \theta_0 &\geq 1 - \xi_i \quad \text{if } y_i = +1 \\ \theta \cdot x + \theta_0 &\leq -1 + \xi_i \quad \text{if } y_i = -1\end{aligned}$$

Combining these 2 equations we yield a single equation for the constraint of primal problem.

$$y^{(i)}(\theta \cdot x^{(i)} + \theta_0) \geq 1 - \xi_i$$

7.1.1 Updated Primal Problem

$$\min \frac{\lambda}{2}||\theta||^2 + \sum_{t=1}^{n} \xi_t$$

$$\text{such that } y^{(i)}(\theta \cdot x^{(i)} + \theta_0) \geq 1 - \xi_i,\ \xi \geq 0,\ t = 1 \ldots n$$

Here, the slack variables are simply encoding the **hinge loss in the primal formulation**.

7.1.2 Updated Dual Formulation

The Dual Problem remains the same, except that we limit how large the Langrange multipliers α_t can become.

- Also the larger the value of λ (the more we want to expand the margin at the expense of the constraints), the smaller the resulting α_t must be.

$$\max \sum_{i=1}^{n} \alpha_t - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y^{(i)} y^{(j)} (x^{(i)})$$

$$\text{subject to } 0 \leq \alpha_t \leq \frac{1}{\lambda},\ \sum_{i=1}^{n} \alpha_i y^{(i)} = 0$$

7.1.3 Complementary slackness constraints

$$\begin{aligned}
&\hat{\alpha}_i = 0 \rightarrow && \text{non-support vectors: } y^{(i)}(\sum j = 1^n \hat{\alpha}_j y^{(j)} (x^{(j)} \cdot x^{(i)}) + \hat{\theta}_0) \geq 1 \\
&\hat{\alpha}_i \Subset \left(0, \frac{1}{\lambda}\right) \rightarrow && \text{support vectors: } y^{(i)}(\sum j = 1^n \hat{\alpha}_j y^{(j)} (x^{(j)} \cdot x^{(i)}) + \hat{\theta}_0) = 1 \\
&\hat{\alpha}_i = \frac{1}{\lambda} \rightarrow && \text{margin violations: } y^{(i)}(\sum j = 1^n \hat{\alpha}_j y^{(j)} (x^{(j)} \cdot x^{(i)}) + \hat{\theta}_0) \leq 1
\end{aligned}$$

We can use α_i that lies in the interior of possible values to reconstruct $\hat{\theta}_0$.

8 Mixture Models and Expectation Minimization

8.1 Spherical Gaussian

Spherical Gaussian Distribution is a simple spherically symmetric distribution around the centroid/mean μ. Given:

- x: Point
- μ: Mean
- σ: Variance
- d: Dimension

The distribution can be represented by a circle centered at mean μ with radius σ. We can generate points from the following spherical Gaussian Distribution, each with a likelihood of

$$\begin{aligned} P(x|\mu,\sigma^2) &= N(x;\mu,\sigma^2 I) \\ &= \frac{1}{(2\pi\sigma^2)^{d/2}}\exp\left(-\frac{1}{2\sigma^2}||x-\mu||^2\right) \end{aligned}$$

8.1.1 Overall Objective Function

We try to find the best Gaussian that fits the data with the criterion of maximum likelihood (ML). The likelihood of the training data is calculated as follows:

$$\begin{aligned} \ell(S_n|\mu,\sigma^2) &= \prod_{t=1}^{n} p(x^{(t)}|\mu,\sigma^2) \\ &= \sum_{t=1}^{n} \log p(x^{(t)}|\mu,\sigma^2) \\ &= \sum_{t=1}^{n} [-\frac{d}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}||x^{(t)}-\mu||^2] \\ &= -\frac{dn}{2}\log(2\pi\sigma^2) - \frac{1}{2\sigma^2}\sum_{t=1}^{n}||x^{(t)}-\mu||^2 \end{aligned}$$

To minimize this function and the corresponding Maximum Likelihood Estimators, we find the gradients respective to the parameters μ, σ^2 and equate them to 0.

$$\frac{\partial \ell(S_n|\mu,\sigma^2)}{\partial \mu} = 0$$
$$\frac{\partial \ell(S_n|\mu,\sigma^2)}{\partial \sigma^2} = 0$$

Deriving maximum likelihood estimator of μ, $\hat{\mu}$:

$$\begin{aligned} \frac{\partial \ell(S_n|\mu,\sigma^2)}{\partial \mu} &= -\frac{1}{2\sigma^2}\cdot 2\cdot\sum_{t=1}^{n}\vec{\mu}-\vec{x}^{(t)} \\ &= \vec{0} \end{aligned}$$

$$\vec{\mu} = \frac{1}{n}\sum_{t=1}^{n}\vec{x}^{(t)}$$

Deriving maximum likelihood estimator of σ^2, $\hat{\sigma^2}$:

$$\frac{\partial \ell(S_n|\mu,\sigma^2)}{\partial \sigma^2} = -\frac{dn}{2}\cdot\frac{1}{\sigma^2}\cdot 2\pi - \frac{1}{2\sigma^4}\sum_{t=1}^{n}||x^{(t)}-\mu||^2$$

$$\frac{dn}{2}\cdot\frac{1}{\sigma^2} = \frac{1}{2\sigma^4}\sum_{t=1}^{n}||x^{(t)}-\mu||^2$$

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

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

8.2 Mixture of Gaussians

When the data are best described by multiply clusters instead of one, we need to specify multiple Gaussians, each for one cluster. Assuming k clusters is optimal for describing the data,

$$P(x|\mu^{(i)},\sigma_i^2), \text{for } i = 1,\ldots,k$$

But we don't have the respective $\mu^{(i)},\sigma_i^2$, so we have to somehow evaluate the probability each data point x could come as sample from our mixture model, and adjust the model parameters so as to increase this probability. Each x could be generated from any cluster, just with different probabilities.
For easier representation, we let θ specify all parameters for the mixture model:

$$\theta = \{\mu^{(1)},\ldots,\mu^{(k)},\sigma_1^2,\ldots,\sigma_k^2,p_1,\ldots,p_k\}$$

where $p_1,\ldots,p_k$ specify the frequency of points we would expect to see in each cluster.

8.2.1 Labelled Case

If data points are already labelled (assigned to a single cluster), we could estimate the Gaussian models same as before, and even evaluate the cluster sizes based on the actual number of points.

Let $\delta(i|t)$ be an indicator that tells us whether $x^{(t)}$ should be assigned to cluster i.

- $\delta(i|t) = 1$, if $x^{(t)}$ is assigned to i
- $\delta(i|t) = 0$, otherwise

Maximum Likelihood Objective

$$\sum_{t=1}^{n}\left[\sum_{i=1}^{k}\delta(i|t)\log\left(p_i\cdot p(x^{(t)}|\mu^i,\sigma_i^2)\right)\right]$$

Here, the inner summation selects the Gaussian that we should use to generate the corresponding data point consistent with the assignments.

We can exchange the summations to demonstrate that the Gaussians can be solved separately from each other. The terms inside the bracket now represents the objective for all points in i-th cluster.

$$\sum_{i=1}^{k}\left[\sum_{t=1}^{n}\delta(i|t)\log(p_i\cdot p(x^{(t)}|\mu^i,\sigma_i^2))\right]$$

Updating Model Parameters:

$$\text{Number of points assigned to cluster i: } \hat{n}_i = \sum t = 1^n \delta(i|t)$$

$$\text{Fraction of points in cluster i: } \hat{p}_i = \frac{\hat{n}_i}{n}$$

$$\text{Mean of points in cluster i: } \hat{\mu}_i^{(t)} = \frac{1}{n} \sum_{t=1}^{n} \delta(i|t) x^{(t)}$$

$$\text{Mean squared spread in cluster i: } \hat{\sigma^2}_i = \frac{1}{d\hat{n}_i} \sum t = 1^n \delta(i|t) ||x^{(t)} - \hat{\mu}^{(i)}||^2$$

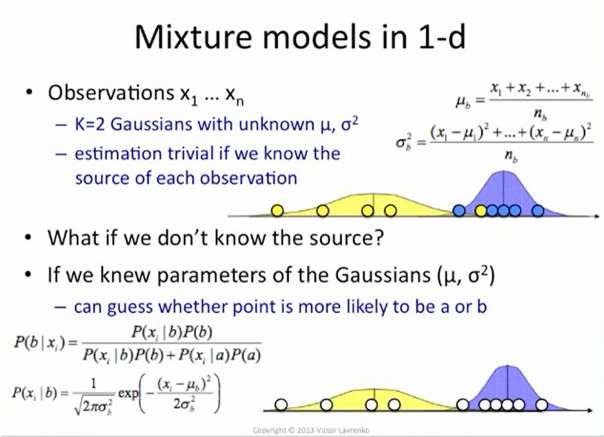
8.2.2 Unlabelled Case

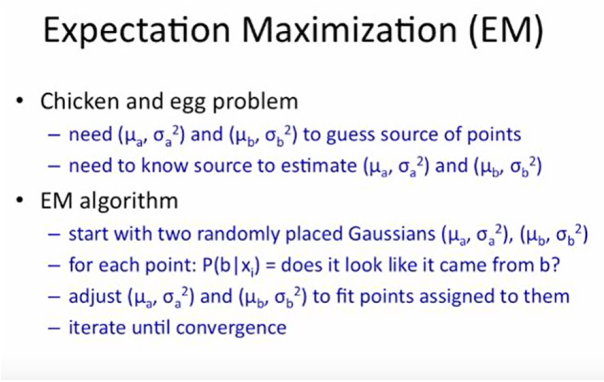
Now $\delta(i|t)$ is not given, but we can apply the EM-algorithm to derive the respective $\delta(i|t)$.

8.3 Estimating Mixtures: the EM-Algorithm

We can use an iterative algorithm known as Expectation Maximization algorithm.

Here are some relevant slides from a good video series by Victor Lavrenko on EM that I recommend you to watch.





8.4 EM Algorithm Process

1. **Initialization:** Randomly initialize model parameters $p_i, \mu^{(i)}, \delta_i^2$
2. **Expectation** + Evaluation(Testing): Find new assignments of $p(i|t)$
3. **Maximization** + Parameter Estimation (Supervised Learning): Update model parameters $p_i, \mu^{(i)}, \sigma_i^2$

We now substitute $\delta(i|t)$ with $p(i|t)$, the probability that $x^{(t)}$ is assigned to i.

$$\sum_{i}^{k} p(i|t) = 1$$

$$\sum_{i=1}^{k}[\sum_{t=1}^{n} p(i|t)\log(p_i \cdot p(x^{(t)}|\mu^i, \sigma_i^2)]$$

This simple algorithm is guaranteed to monotonically increase the log-likelihood of the data under the mixture model (cf. k-means). Just as in k-means, however, it may only find a locally optimal solution.

9 Hidden Markov Model (HMM)

9.1 Structured Prediction

$$f : S_1 \to S_2$$

where $dim(S_1) > dim(S_2)$

S_1 is a structured space consisting of word/observation sequences

S_2 is a structured space consisting tag/state sequences

9.2 Computing Joint Likelihood of a HMM

To calculate the probabilities of the sequence of words $(x_0...x_n)$ generated given the tags $(y_0...y_{n+1})$

$$p(x_1...x_n, y_0...y_{n+1}) = p(y_0...y_{n+1}) \cdot p(x_1...x_n|y_0...y_{n+1})$$

We assumed strong independence between the variables such that

$$p(y_2|y_1, y_0) \approx p(y_2|y_1)$$

Expanding the joint probability terms individually,

$$p(y_0...y_{n+1}) = \prod_{j=0}^{n} p(y_n + 1|y_n)$$

$$p(x_1...x_n|y_0...y_n + 1) = \prod_{j=1}^{n} p(x_j|y_j)$$

$$\text{Hence, } p(x_1...x_n, y_0...y_{n+1}) = \prod_{j=0}^{n} p(y_n + 1|y_n) \cdot \prod_{j=1}^{n} p(x_j|y_j)$$

For simplicity, we rewrite the notation of individual transmission and emission probabilities.

$$\text{Transmission Probability} \qquad a_{u,v} = \frac{\text{count}(u, v)}{\text{count}(u)}$$

$$\text{Emission Probability} \qquad b_{u(o)} = \frac{\text{count}(u \to o)}{\text{count}(u)}$$

$$\prod_{j=0}^{n} a_{y_j, y_j+1} = \prod_{j=0}^{n} p(y_n + 1 | y_n)$$

$$\prod_{j=1}^{n} b_{y_j(x_j)} = \prod_{j=1}^{n} p(x_j | y_j)$$

$$\text{Joint Probability} \leftarrow \prod_{j=0}^{n} a_{y_j, y_j+1} \cdot \prod_{j=1}^{n} b_{y_j(x_j)}$$

9.3 Supervised Learning

9.3.1 Decoding

Finding most probable label sequence y given the word sequence x

Brute Force Enumeration

$$\begin{aligned} y* &= \arg\max_y p(y|x) \\ &= \arg\max_y \frac{p(x,y)}{p(x)} \\ &= \arg\max_y p(x,y) \end{aligned}$$

This method is not feasible once there are too many label sequences. There will be $O(|T|^n)$ possible sequences.

Viterbi Algorithm
Since the HMM has a simple dependence structure, we can exploit this in a dynamic programming algorithm.

1. We initialize $\pi(0, u)$
   - 1 if $u = START$
   - 0 otherwise
2. For $j = 0...n-1$,

$$\pi(j+1, u) = \max_v \pi(j, v) \cdot b_u(x_{j+1}) \cdot a_{v,u} y_n^* = \arg\max_u \pi(j, u) \cdot a_u, y_{j+1}^*$$

3. Finally,

$$\pi(n+1, STOP) = \max_v \pi(n, v) \cdot a_{v,STOP})$$

Time complexity: $O((n-1)T^2 + T = T) = O(nT^2)$

- T is the number of nodes in each column, and we carry out n-1 operations for each column.
- Calculation for each node is $O(T)$, and do it for $n * t$ nodes so it is $O(T^2)$.
- $O(T)$ calculation at start and stop nodes.

Space complexity: $O(nT)$

9.4 Unsupervised Learning

We can use EM algorithms to learn model parameters in an unsupervised manner.

> Recall that EM Algorithms seek to maximize likelihood that
> the data is generated by a mixture model.

9.4.1 Hard EM

E-step

1. Assign each observed sequence of outputs (data point) as $x^{(1)}...x^{(m)}$
2. Assign each x to a single state sequence/tag sequence y (*no partial membership*).
3. *Decoding:* For each observation sequence, use Viterbi algorithm to find the most probable state sequence $\mathbf{y}^{(i)^*}$

$$Y^* = \arg\max_Y P(Y|X)$$

M-step

Parameter estimation using MLE with labelled data from the E-step

$$\text{Transmission Probability} \qquad a_{u,v} = \frac{count(u,v)}{count(u)} \qquad \text{for any } u, v \in 0, 1, 2...N+1$$

$$\text{Emission Probability} \qquad b_{u(o)} = \frac{count(u \to o)}{count(u)} \qquad \text{for any } u \in 0, 1, 2...N+1, o \in \sum$$

9.4.2 Soft EM

For soft EM, we must evaluate a posterior probability over possible tag sequences, so we cannot use Viterbi algorithm.

- Viterbi algorithm will result in hard membership (a specific y with max probability) for each observed sequence of outputs.

Inference

To compute the posterior possibilities efficiently for any $u \in 1...N, j \in 1...n$,

$$p(y_j = u|\mathbf{x}; \theta) = \sum_{\mathbf{y}:y_j=u} p(\mathbf{y}|\mathbf{x}; \theta)$$

These probabilities is conditioned on the observed sequence $\mathbf{x}$ and the model parameters θ.
Thereafter,

$$\text{Given that } \sum_B P(A, B) = P(A)$$

$$\alpha_u(j) = p(x_1...x_{j-1}, y_j = u)$$

$$\beta_u(j) = p(x_j...x_n|y_j = u)$$

$$p(y_j = u|\mathbf{x};\theta) = \frac{\alpha_u(j)\beta_u(j)}{\sum_v \alpha_v(k)\beta_v(k)}$$
$$p(y_j = u, y_{j+1} = v|\mathbf{x};\theta) = \frac{\alpha_u(j)\cdot b_u x_j \cdot a_{u,v} \cdot \beta_v(j+1)}{\sum_v \alpha_v(k)\beta_v(k)}$$

Finding the fractional count

$$\begin{aligned}
count(u,v) &= \sum_{i=1}^{m} count^{(i)}(u,v) \\
&= \sum_{i=1}^{m}\sum_{y} p(y|x^{(i)})count(\mathbf{x}^{(i)}, y, u \to v) \\
&= \sum_{i=1}^{m}\sum_{j=0}^{n} p(y_j = u, y_{j+1} = v|\mathbf{x}^{(i)}) \\
&= \sum_{i=1}^{m}\sum_{j=0}^{n} \frac{\alpha_u(j)\cdot b_u x_j \cdot a_{u,v} \cdot \beta_v(j+1)}{\sum_v \alpha_v(k)\beta_v(k)}
\end{aligned}$$

9.5 Forward-Backward Algorithm

9.5.1 Going Forward

- $\alpha_u(j)$ = the sum of the scores of all paths from START to node u at j
- For the first node, there is no emission probability.

$$\begin{aligned}
\alpha_u(1) &= \alpha_{START,u} \\
&= p(x_0, y_1 = u)
\end{aligned}$$

- For the other nodes,

$$\text{Given that } \sum_B P(A,B) = P(A)$$

$$\begin{aligned}
\alpha_u(j+1) &= p(x_1...x_j, y_{j+1} = u) \\
&= \sum_v p(x_1...x_j, y_{j+1} = u, y_j = v) \\
&= \sum_v p(x_1...x_{j-1}, y_j = v, x_j, y_{j+1} = u) \\
&= \sum_v p(x_1...x_{j-1}, y_j = v)\cdot b_v(x_j)\cdot a_{v,u} \\
&= \sum_v \alpha_v(j)\cdot b_v(x_j)\cdot \alpha_{v,u}
\end{aligned}$$

9.5.2 Going Backward

- $\beta_u(j)$ = the sum of the scores of all paths from node u at j to STOP.

$$\begin{aligned}\beta_u(j) &= p(x_j...x_n|y_j = u)\\ &= \sum_v p(x_j...x_n, y_{j+1} = v|y_j = u)\\ &= \sum_v p(x_j, y_{j+1} = v, x_{j+1}...x_n|y_j = u)\\ &= \sum_v b_u(x_j) \cdot a_{u,v} \cdot p(x_{j+1}...x_n|y_{j+1} = v)\\ &= \sum_v b_u(x_j) \cdot a_{u,v} \cdot \beta_v(j+1)\end{aligned}$$

Time Complexity: $O(nT^2)$

- $O(T)$ operations for T nodes at n positions

Time complexity of 1 EM iteration: $O(mnT^2)$

Forward Backward Algorithm is a type of Inside-Outside Algorithm, which the Prof will only cover in NLP course.

10 Forward-Backward Algorithm

10.1 Going Forward

- $\alpha_u(j)$ = the sum of the scores of all paths from START to node u at j
- For the first node, there is no emission probability.

$$\begin{aligned}\alpha_u(1) &= \alpha_{START,u}\\ &= p(x_0, y_1 = u)\end{aligned}$$

- For the other nodes,

$$\text{Given that } \sum_B P(A, B) = P(A)$$

$$\begin{aligned}\alpha_u(j+1) &= p(x_1...x_j, y_{j+1} = u)\\ &= \sum_v p(x_1...x_j, y_{j+1} = u, y_j = v)\\ &= \sum_v p(x_1...x_{j-1}, y_j = v, x_j, y_{j+1} = u)\\ &= \sum_v p(x_1...x_{j-1}, y_j = v) \cdot b_v(x_j) \cdot a_{v,u}\\ &= \sum_v \alpha_v(j) \cdot b_v(x_j) \cdot \alpha_{v,u}\end{aligned}$$

10.2 Going Backward

- $\beta_u(j)$ = the sum of the scores of all paths from node u at j to STOP.

x_{j+1} cannot be generated before x_j

$$\begin{aligned}\beta_u(j) &= p(x_j...x_n|y_j = u) \\ &= \sum_v p(x_j...x_n, y_{j+1} = v|y_j = u) \\ &= \sum_v p(x_j, y_{j+1} = v, x_{j+1}...x_n|y_j = u) \\ &= \sum_v b_u(x_j) \cdot a_{u,v} \cdot p(x_{j+1}...x_n|y_{j+1} = v) \\ &= \sum_v b_u(x_j) \cdot a_{u,v} \cdot \beta_v(j+1)\end{aligned}$$

Time Complexity: $O(nT^2)$

- $O(T)$ operations for T nodes at n positions

11 Bayesian Networks

Bayesian networks are generative probabilities models that are developed for representing and using probabilistic information. Bayesian networks subsume mixture models, HMMs and many others.

1. Bayesian Networks are **Directed Acyclic Graphs** (DAG) over the variables.
    - We usually generate the graph from x_1 first, and each node x_i has a Conditional Probability Distribution $P(X_i|\arg P(X_i))$.
    - If there is a directed edge from $x_1 \to x_3$, then x_1 is a parent of x_3. And once we know the parents, we can write the probability distribution over all the variables with chain rule as

$$P(X_1 = x_1, ..., X_d = x_d) = \prod_{i=1}^{d} P(X_i = x_i|\arg P(X_i))$$

Cannot have cyclic graphs, as it will result in weird probabilities like $p(x_3\|x_3)$. This is simply not valid!! Generative Models typically have DAGs.

2. The network will have the **associated probability distribution**.
    i. For each node i, we will attach a table of probabilities containing the combination of assignment of values that has been calculated up to node i.
    ii. As i increases, the table of probabilities attached to node i expands.
    iii. Each row in the table must add up to 1.
    iv. This is similar to $b_u(o)$.

11.1 Independence in Bayesian Networks

If there is a direct edge from A to B, influence can flow from X to Y regardless of whether other variables are observed. They would never be independent.

Basic Independence Rule

If A and B are conditionally independent, $P(A, B) = P(A) \cdot P(B)$ $P(A|B) = P(A)$

To prove dependence, just show a case where $P(A|B) \neq P(A)$.

11.1.1 Summary: X and Z dependence

For the examples below,

- ◯ refers to known variable,
- ● refers to unknown variable,
- →: causal direction
- ←: evidential direction

Relationship	Y is not given	Y is given
X→Y→Z	✔: ◯→◯→◯	✘: ◯→●→◯
X←Y→Z	✔: ◯←◯→◯	✘: ◯←●→◯
X→Y←Z	✘: ◯→◯←◯	✔: ◯→●←◯

Just remember one column and you can derive the other column.

11.1.2 Chain: X→Y→Z

◯→●→◯: to prove X, Z are independent. (This is hard to prove)

$$P(X, Z) = P(X) \cdot P(Z)$$

◯→◯→◯: to prove X, Z are independent.

$$P(X, Z|Y) = P(X|Y) \cdot P(Z|Y)$$

To prove that the conditional dependence of Z and Y is not affected regardless of Y having a parent,

$$P(Z|Y,X) = P(Z|Y)$$

We can calculate as follows:

$$\begin{aligned} P(X,Y,Z) &= P(X) \cdot P(Y|X) \cdot P(Z|Y) \\ P(Z|Y,X) &= \frac{P(X,Y,Z)}{P(X,Y)} \\ &= \frac{P(X) \cdot P(Y|X) \cdot P(Z|Y)}{P(X,Y)} \\ &= \frac{P(X) \cdot P(Y|X) \cdot P(Z|Y)}{P(X) \cdot P(Y|X)} \\ &= P(Z|Y) \end{aligned}$$

- Hence, Y is known as an open gate.

11.1.3 Common Parent (Common Cause)

Y has a directed edge to both X, Z.

- ◯←●→◯: we cannot prove that X, Z are independent.
- ◯←◯→◯: X, Z are independent.
- Here, Y is an **open gate**.

11.1.4 Common Child (Explaining Away)

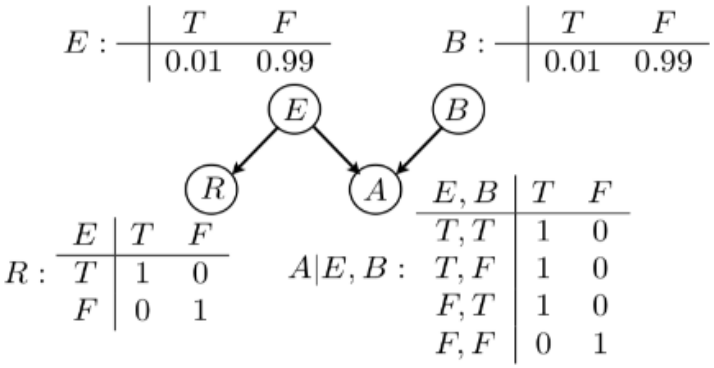
> Explaining away is a common pattern of reasoning in which the confirmation of one cause of an observed or believed event **reduces the need to invoke alternative causes**. The opposite of explaining away also can occur, where the confirmation of one cause increases belief in another.

Both X, Z have a directed edge to Y.

- ◯→●←◯: X, Z are independent.
- ◯→◯←◯: X, Z are not independent.
- Hence, Y is a **open gate**.

11.1.5 Examples

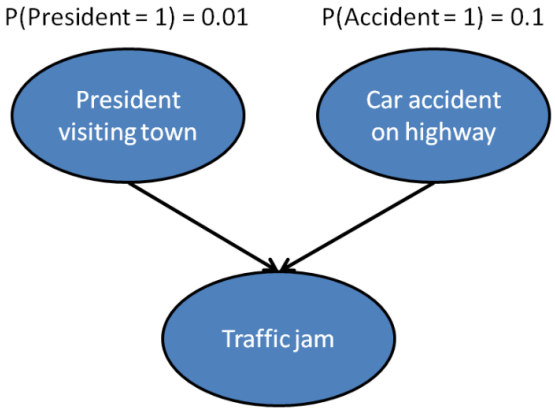
Earthquake + Burglary



Here, E and B have a common child A, and E is the common parent of R and A.

$$\begin{aligned} P(B = T|A = T) &= \frac{P(B = T, A = T)}{\sum_{b \in T,F} P(B = b, A = T)} \\ &= 0.5025 \\ P(B = b, A = T, E = T) &= P(E = T) \cdot P(A = T|E = T, B = T) \\ &= 0.99 \cdot 0.01 \cdot 1 \\ P(B = b|A = T, E = T) &= \frac{P(B = b, A = T, E = T)}{P(A = T, E = T)} \\ &= \frac{0.01 \cdot 0.01}{0.99 \cdot 0.01 + 0.01 \cdot 0.01} \\ &= 0.01 \end{aligned}$$

President and Traffic Accident



P(Traffic = 1 | President = 0, Accident = 0) = 0.1
P(Traffic = 1 | President = 0, Accident = 1) = 0.5
P(Traffic = 1 | President = 1, Accident = 0) = 0.6
P(Traffic = 1 | President = 1, Accident = 1) = 0.9

	President = 0, Accident = 0	**President = 1, Accident = 0**	**President = 0, Accident = 1**	**President = 1, Accident = 1**
Traffic = 1	0.1	0.6	0.5	0.9
Traffic = 0	0.9	0.4	0.5	0.1

11.2 Bayes Ball Algorithm

As long we can find a path to connect X and Y that are open gates, we can conclude X and Y are dependent.

11.2.1 Markov Net

A Markov Net comprises of a particular node x_i's parents, co-parents and children. Node x_i is independent of all other nodes outside this Markov net.

11.3 Bayesian Information Criterion (BIC)

BIC is a model selection criterion for comparing 2 different Bayesian networks, where

- $dim(G)$ = the number of independent parameters in the model
- m = the number of data points in the training set
- $r_j \forall j \in pa_i$ = the size of the probability table $\theta_i(x_i|\mathbf{x}_{pa_i})$ − the number of associated normalization constraints

$$BIC(D;\theta,G) = l(D;\theta,G) - \frac{dim(G)}{2}\log(m)$$

We then search for the graph G that **maximizes** $BIC(D;\theta,G)$.

1. With the complete data, find conditional estimates for each variable and the parent selection scores for each variable.
$$\theta_i(x_i|x_{pa_i}), score(i|pa_i;D)$$
2. Find the decomposable scoring function for graphs.
$$score(G;D) = \sum_{i=1}^{d} score(i|pa_i,D)$$
3. Find highest scoring **acyclic graph**.
$$\arg\max_G score(G;D)$$

12 Reinforcement Learning

Problem: $f : S \to A$

- S: Set of States
- A: Set of Actions

Transition Probability

- Moving from one state to another state is not definite
- $T(s, a, s') = p(s'|s, a)$

Reward

- Incentivize a target state to be reached / not reached.
- Can be positive (reward) / negative (penalty)
- $R(s, a, s')$: Generally
- $R(s')$: if the system only requires the target state to be reached.

12.1 Markov Decision Process

Given

- a set of states S
- a set of actions A
- a transition probability function $T(s, a, s') = p(s'|s, a)$
- a reward function $R(s, a, s')$ or $R(s')$

12.1.1 Utility (Long Term Reward)

- Maximizing the rewards alone is not sufficient to incentivise the program to reach the final state that you want. The program may infinitely loop around a few states.
- Hence we apply a **discount** γ to long-term rewards.

$$\begin{aligned} U([s_1, s_2, ...s_n]) &= R(s_1) + \gamma R(s_2) + \gamma_2 R(s_3) + \dots \\ &= \sum_{t=0}^{\infty} R(s_t) \end{aligned}$$

12.1.2 Value Iteration Policy

We first define the following:

- $\pi(s)$: a particular policy that specifies the action we should take in state s
- $V^{\pi}(s)$: The value of state s under policy π
- $Q^{\pi}(s,a)$: The Q-value of state s and action a under policy π

and the optimal version of them:

- $\pi^*(s)$: a particular policy that specifies the action we should take in state s
- $V^*(s)$: The value of state s under the optimal policy π^*
- $Q^*(s,a)$: The Q-value of state s and action a under policy π^*

$$
\begin{aligned}
Q^*(s,a) &= \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')] \\
V^*(s) &= Q^*(s,\pi^*(s)) \\
&= \max_a Q^*(s,a) \\
&= \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')] \\
\pi^*(s) &= \arg\max_a Q^*(s,a)
\end{aligned}
$$

12.1.3 Value Iteration Algorithm

1. Start with $V_0^*(s) = 0 \ \forall s \in S$
2. Given V_i^*,
   i. calculate the values for all states $s \in S$ and
   ii. keep track of the best actions to formulate the best policy

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

3. Repeat until convergence.

12.1.4 Q-value Iteration Algorithm

Similar to Value Iteration Algorithm.