UNIT III

Ashok Kumar Yadav

Mar 2018

1 Unsupervised learning

Unsupervised learning is a type of machine learning algorithm used to draw inferences from datasets consisting of input data without labelled responses. The most common unsupervised learning method is cluster analysis, which is used for exploratory data analysis to find hidden patterns or grouping in data. Unsupervised learning is the training of machine using information that is neither classified nor labelled and allowing the algorithm to act on that information without guidance. Here the task of machine is to group unsorted information according to similarities, patterns and differences without any prior training of data. Unlike supervised learning, no teacher is provided that means no training will be given to the machine. Therefore machine is restricted to find the hidden structure in unlabelled data by our-self. Unsupervised learning classified into two categories of algorithms:

Clustering: A clustering problem is where you want to discover the inherent groupings in the data, such as grouping customers by purchasing behavior.
Association: An association rule learning problem is where you want to discover rules that describe large portions of your data, such as people that buy X also tend to buy Y.

2 Clustering K means

Cluster analysis or clustering is the task of grouping a set of objects in such a way that objects in the same group (called a cluster) are more similar (in some sense) to each other than to those in other groups (clusters).
K-means clustering is a method of vector quantization, originally from signal processing, that is popular for cluster analysis in data mining. k-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. This results in a partitioning of the data space into Voronoi cells.

The problem is computationally difficult (NP-hard); however, efficient heuristic algorithms converge quickly to a local optimum. These are usually similar to

the expectation-maximization algorithm for mixtures of Gaussian distributions via an iterative refinement approach employed by both k-means and Gaussian mixture modeling. They both use cluster centers to model the data; however, k-means clustering tends to find clusters of comparable spatial extent, while the expectation-maximization mechanism allows clusters to have different shapes.

The algorithm has a loose relationship to the k-nearest neighbor classifier, a popular machine learning technique for classification that is often confused with k-means due to the name. Applying the 1-nearest neighbor classifier to the cluster centers obtained by k-means classifies new data into the existing clusters. This is known as nearest centroid classifier or Rocchio algorithm.

2.1 K means Algorithm

k-means is one of the simplest unsupervised learning algorithms that solve the well known clustering problem. The procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters) fixed apriori. The main idea is to define k centers, one for each cluster. These centers should be placed in a cunning way because of different location causes different result. So, the better choice is to place them as much as possible far away from each other. The next step is to take each point belonging to a given data set and associate it to the nearest center. When no point is pending, the first step is completed and an early group age is done. At this point we need to re-calculate k new centroids as barycenter of the clusters resulting from the previous step. After we have these k new centroids, a new binding has to be done between the same data set points and the nearest new center. A loop has been generated. As a result of this loop we may notice that the k centers change their location step by step until no more changes are done or in other words centers do not move any more. Finally, this algorithm aims at minimizing an objective function know as squared error function given by:

$$J(V) = \sum_{i=1}^{c} \sum_{j=1}^{c_i} (\|x_i - v_j\|)^2$$

where
$\|x_i - v_j\|$ is the Euclidean distance between x_i and v_j
c_i is the number of data points in i^{th} cluster.
c is the number of cluster centers.

Let $X = \{x_1, x_2, x_3, \ldots, x_n\}$ be the set of data points and $V = \{v_1, v_2, \ldots, v_c\}$ be the set of centers.

1 Randomly select c cluster centers.

2 Calculate the distance between each data point and cluster centers.

3 Assign the data point to the cluster center whose distance from the cluster center is minimum of all the cluster centers

4 Recalculate the new cluster center using:

$$v = \frac{1}{c_i} \sum_{j=1}^{c_i} x_i$$

Where c_i represents the number of data points in i^{th} cluster.

5 Recalculate the distance between each data point and new obtained cluster centers.

6 If no data point was reassigned then stop, otherwise repeat from step 3)

2.2 Pros and Cons

Pros

- Fast, robust and easier to understand
- Relatively efficient: $O(tknd)$, where n is number of objects, k is number of clusters, d is number of dimension of each object, and t is number of iterations. Normally, $k, t, d << n$.
- Gives best result when data set are distinct or well separated from each other.

Cons

- The learning algorithm requires apriori specification of the number of cluster centers.
- The use of Exclusive Assignment - If there are two highly overlapping data then k-means will not be able to resolve that there are two clusters.
- The learning algorithm is not invariant to non-linear transformations i.e. with different representation of data we get different results (data represented in form of cartesian co-ordinates and polar co-ordinates will give different results).
- Euclidean distance measures can unequally weight underlying factors.
- The learning algorithm provides the local optima of the squared error function.
- Randomly choosing of the cluster center cannot lead us to the fruitful result.

- Applicable only when mean is defined i.e. fails for categorical data.
- Unable to handle noisy data and outliers.
- Algorithm fails for non-linear data set.

3 Expectation-Maximization (EM) Algorithm

The Expectation-Maximization (EM) algorithm is a way to find maximum-likelihood estimates for model parameters when your data is incomplete, has missing data points, or has unobserved (hidden) latent variables. It is an iterative way to approximate the maximum likelihood function. While maximum likelihood estimation can find the "best fit" model for a set of data, it doesn't work particularly well for incomplete data sets. The more complex EM algorithm can find model parameters even if you have missing data. It works by choosing random values for the missing data points, and using those guesses to estimate a second set of data. The new values are used to create a better guess for the first set, and the process continues until the algorithm converges on a fixed point. The Expectation-Maximization (EM) Algorithm is an iterative method to find the Maximum Likelihood Estimation (MLE) or Maximum a-posteriori estimates (MAP) estimate for models with latent variables. This is a description of how the algorithm works:

1 **Initialization:** Initialization: Get an initial estimate for parameters θ^0 (e.g. all the μ_k, μ_k^2 and π variables). In many cases, this can just be a random initialization.

2 **Expectation Step:** Assume the parameters $(\theta^t - 1)$ from the previous step are fixed, compute the expected values of the latent variables (or more often a function of the expected values of the latent variables)

3 **Maximization Step:** Given the values you computed in the last step (essentially known values for the latent variables), estimate new values for θ^t that maximize a variant of the likelihood function.

4 **Exit Condition:** If likelihood of the observations have not changed much, exit; otherwise, go back to Step 1.

3.1 Limitations

The EM algorithm can very very slow, even on the fastest computer. It works best when you only have a small percentage of missing data and the dimensionality of the data isn't too big. The higher the dimensionality, the slower the E-step; for data with larger dimensionaloty, you may find the E-step runs extremely slow as the procedure approaches a local maximum.

4 Mixture of Gaussians

Gaussian mixture models are a probabilistic model for representing normally distributed subpopulations within an overall population. Mixture models in general don't require knowing which subpopulation a data point belongs to, allowing the model to learn the subpopulations automatically. Since subpopulation assignment is not known, this constitutes a form of unsupervised learning.

For example, in modeling human height data, height is typically modeled as a normal distribution for each gender with a mean of approximately 5'10" for males and 5'5" for females. Given only the height data and not the gender assignments for each data point, the distribution of all heights would follow the sum of two scaled (different variance) and shifted (different mean) normal distributions. A model making this assumption is an example of a Gaussian mixture model (GMM), though in general a GMM may have more than two components. Estimating the parameters of the individual normal distribution components is a canonical problem in modeling data with GMMs.

GMMs have been used for feature extraction from speech data, and have also been used extensively in object tracking of multiple objects, where the number of mixture components and their means predict object locations at each frame in a video sequence.

A Gaussian mixture model is parameterized by two types of values, the mixture component weights and the component means and variances/covariances. For a Gaussian mixture model with K components, the component has a mean of μ_k and variance of σ_k for the univariate case and a mean of $\vec{\mu}_k$ and covariance matrix of $\sum_k$ for the multivariate case. The mixture component weights are defined as ϕ_k for component C_k, with the constraint that $\sum_{i=1}^{K} \phi_i = 1$ so that the total probability distribution normalizes to 1. If the component weights aren't learned, they can be viewed as an a-priori distribution over components such that $p(x \text{ generated by component } C_k) = \phi_k$. If they are instead learned, they are the a-posteriori estimates of the component probabilities given the data.

One-dimensional Model

$$p(x) = \sum_{i=1}^{K} \phi_i N(x|\mu_i, \sigma_i)$$

$$N(x|\mu_i, \sigma_i) = \frac{1}{\sigma_i \sqrt{2\pi}} exp\left(-\frac{(x-\mu_i)^2}{2\sigma_i^2}\right)$$

$$\sum_{i=1}^{K} \phi_i = 1$$

Multi-dimensional Model

$$p(\vec{x}) = \sum_{i=1}^{K} \phi_i N(\vec{x}|\vec{\mu}_i, \sum_i)$$

$$N(\vec{x}|\vec{\mu}_i, \sum_i) = \frac{1}{\sqrt{(2\pi)^K |\sum_i|}} exp\left(-\frac{1}{2}(\vec{x} - \vec{\mu}_i)^T \sum_i^{-1} (\vec{x} - \vec{\mu}_i)\right)$$

$$\sum_{i=1}^{K} \phi_i = 1$$

4.1 EM for Gaussian Mixture Models

Expectation maximization for mixture models consists of two steps. The first step, known as the expectation step or E step, consists of calculating the expectation of the component assignments C_k for each data point $x_i \in X$ given the model parameters ϕ_k, μ_k and σ_k.

The second step is known as the maximization step or M step, which consists of maximizing the expectations calculated in the E step with respect to the model parameters. This step consists of updating the values ϕ_k, μ_k and σ_k.

The entire iterative process repeats until the algorithm converges, giving a maximum likelihood estimate. Intuitively, the algorithm works because knowing the component assignment C_k for each x_i makes solving for ϕ_k, μ_k and σ_k easy, while knowing ϕ_k, μ_k and σ_k makes inferring $P(C_k|x_i)$ easy. The expectation step corresponds to the latter case while the maximization step corresponds to the former. Thus, by alternating between which values are assumed fixed, or known, maximum likelihood estimates of the non-fixed values can be calculated in an efficient manner.

4.2 Algorithm for Univariate Gaussian Mixture Models

The expectation maximization algorithm for Gaussian mixture models starts with an initialization step, which assigns model parameters to reasonable values based on the data. Then, the model iterates over the expectation (E) and maximization (M) steps until the parameters' estimates converge, i.e. for all parameters θ_t at iteration $t, |\theta_t - \theta_{t-1}| \leq \epsilon$ for some user-defined tolerance ϵ. The EM algorithm for a univariate Gaussian mixture model with K components is described below. A variable denoted $\hat{\theta}$ denotes an estimate for the value θ. All equations can be derived algebraically by solving for each parameter as outlined in the section above titled EM for Gaussian Mixture Models.

Initialization Step:

- Randomly assign samples without replacement from the dataset $X = \{x_1, \ldots, x_N\}$ to the component mean estimates $\hat{\mu}_1, \ldots, \hat{\mu}_K$ E.g. for $K = 3$ and $N = 100$, set $\hat{\mu}_1 = x_{45}, \hat{\mu}_2 = x_{32}, \hat{\mu}_3 = x_{10}$

- Set all component variance estimates to the sample variance $\hat{\sigma}_1^2, \ldots, \hat{\sigma}_K^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2$ where $\bar{x}$ is the sample mean $\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$.
- Set all component distribution prior estimates to the uniform distribution $\hat{\phi}_1, \ldots, \hat{\phi}_K = \frac{1}{K}$

Expectation (E) Step:

- Calculate $\forall i, k$

$$\hat{\gamma}_{ik} = \frac{\hat{\phi}_k N(x_i|\hat{\mu}_k, \hat{\sigma}_k)}{\sum_{j=1}^{K} \hat{\phi}_j N(x_i|\hat{\mu}_j, \hat{\sigma}_j}$$

  where $\hat{\gamma}_{ik}$ is the probability that x_i is generated by component C_k. Thus, $\hat{\gamma}_{ik} = p(C_k|x_i, \hat{\phi}, \hat{\mu}, \hat{\sigma})$.

Maximization (M) Step:
Using the $\hat{\gamma}_{ik}$ calculated in the expectation step, calculate the following in that order $\forall k$:

- $\hat{\phi}_k = \sum_{i=1}^{N} \frac{\hat{\gamma}_{ik}}{N}$
- $\hat{\mu}_k = \frac{\sum_{i=1}^{N} \hat{\gamma}_{ik} x_i}{\sum_{i=1}^{N} \hat{\gamma}_{ik}}$
- $\hat{\sigma}_k^2 = \frac{\sum_{i=1}^{N} \hat{\gamma}_{ik}(x_i - \hat{\mu}_k)^2}{\sum_{i=1}^{N} \hat{\gamma}_{ik}}$

When the number of components K is not known a priori, it is typical to guess the number of components and fit that model to the data using the EM algorithm. This is done for many different values of K. Usually, the model with the best trade-off between fit and number of components (simpler models have fewer components) is kept. The EM algorithm for the multivariate case is analogous, though it is more complicated and thus is not expounded here.

5 Factor Analysis

Factor analysis is a statistical method used to describe variability among observed, correlated variables in terms of a potentially lower number of unobserved variables called factors. For example, it is possible that variations in six observed variables mainly reflect the variations in two unobserved (underlying) variables. Factor analysis searches for such joint variations in response to unobserved latent variables. The observed variables are modelled as linear combinations of the potential factors, plus "error" terms. Factor analysis aims to find independent latent variables.

It is a theory used in machine learning and related to data mining. The theory behind factor analytic methods is that the information gained about the interdependencies between observed variables can be used later to reduce the

set of variables in a dataset. Factor analysis is commonly used in biology, psychometrics, personality theories, marketing, product management, operations research, and finance. It may help to deal with data sets where there are large numbers of observed variables that are thought to reflect a smaller number of underlying/latent variables. It is one of the most commonly used interdependency techniques and is used when the relevant set of variables shows a systematic inter-dependence and the objective is to find out the latent factors that create a commonality

5.1 Types of factoring:

Principal component analysis:
This is the most common method used by researchers. PCA starts extracting the maximum variance and puts them into the first factor. After that, it removes that variance explained by the first factors and then starts extracting maximum variance for the second factor. This process goes to the last factor.
Common factor analysis:
The second most preferred method by researchers, it extracts the common variance and puts them into factors. This method does not include the unique variance of all variables. This method is used in SEM.
Image factoring:
This method is based on correlation matrix. OLS Regression method is used to predict the factor in image factoring.
Maximum likelihood method:
This method also works on correlation metric but it uses maximum likelihood method to factor.
Other methods of factor analysis:
Alfa factoring outweighs least squares. Weight square is another regression based method which is used for factoring.

6 Principal Component Analysis

Principal component analysis (PCA) is 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. If there are n observations with p variables, then the number of distinct principal components is min(n-1,p). This transformation is defined in such a way that the first principal component has the largest possible variance (that is, accounts for as much of the variability in the data as possible), and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components. The resulting vectors (each being a linear combination of the variables and containing n observations) are an uncorrelated orthogonal basis set. PCA is sensitive to the relative scaling of the original variables.

PCA is the simplest of the true eigenvector-based multivariate analyses. Often, its operation can be thought of as revealing the internal structure of the data in a way that best explains the variance in the data. If a multivariate dataset is visualised as a set of coordinates in a high-dimensional data space, PCA can supply the user with a lower-dimensional picture, a projection of this object when viewed from its most informative viewpoint. This is done by using only the first few principal components so that the dimensionality of the transformed data is reduced. PCA is closely related to factor analysis. Factor analysis typically incorporates more domain specific assumptions about the underlying structure and solves eigenvectors of a slightly different matrix.

6.1 Properties of Principal Component

Technically, a principal component can be defined as a linear combination of optimally-weighted observed variables. The output of PCA are these principal components, the number of which is less than or equal to the number of original variables. Less, in case when we wish to discard or reduce the dimensions in our dataset. The PCs possess some useful properties which are listed below:
1. The PCs are essentially the linear combinations of the original variables, the weights vector in this combination is actually the eigenvector found which in turn satisfies the principle of least squares.
2. The PCs are orthogonal.
3. The variation present in the PCs decrease as we move from the 1st PC to the last one, hence the importance.
The least important PCs are also sometimes useful in regression, outlier detection, etc.

6.2 Implementing PCA on a 2-D Dataset

1 **Normalize the data**
First step is to normalize the data that we have so that PCA works properly. This is done by subtracting the respective means from the numbers in the respective column. So if we have two dimensions X and Y, all X become $x^{'}$ and all Y become $y^{'}$. This produces a dataset whose mean is zero.

2 **Calculate the covariance matrix**
Since the dataset we took is 2-dimensional, this will result in a 2×2 Covariance matrix

$$Matrix(Covariance) = \begin{bmatrix} Var[X_1] & Cov[X_1, X_2] \\ Cov[X_2, X_1] & Var[X_2] \end{bmatrix}$$

Please note that $Var[X_1] = Cov[X_1, X_1]$ and $Var[X_2] = Cov[X_2, X_2]$.

3 **Calculate the eigenvalues and eigenvectors**
Next step is to calculate the eigenvalues and eigenvectors for the covariance

matrix. The same is possible because it is a square matrix. λ is an eigenvalue for a matrix A if it is a solution of the characteristic equation: $|\lambda I - A| = 0$.
Where, I is the identity matrix of the same dimension as A which is a required condition for the matrix subtraction as well in this case and $||$ is the determinant of the matrix. For each eigenvalue λ, a corresponding eigen-vector v, can be found by solving:
$(\lambda I - A)v = 0$

4 **Choosing components and forming a feature vector**
We order the eigenvalues from largest to smallest so that it gives us the components in order or significance. Here comes the dimensionality reduction part. If we have a dataset with n variables, then we have the corresponding n eigenvalues and eigenvectors. It turns out that the eigenvector corresponding to the highest eigenvalue is the principal component of the dataset and it is our call as to how many eigenvalues we choose to proceed our analysis with. To reduce the dimensions, we choose the first p eigenvalues and ignore the rest. We do lose out some information in the process, but if the eigenvalues are small, we do not lose much.
Next we form a feature vector which is a matrix of vectors, in our case, the eigenvectors. In fact, only those eigenvectors which we want to proceed with. Since we just have 2 dimensions in the running example, we can either choose the one corresponding to the greater eigenvalue or simply take both.
$FeatureVector = (\lambda_1, \lambda_2)$

5 **Forming Principal Components**
This is the final step where we actually form the principal components using all the math we did till here. For the same, we take the transpose of the feature vector and left-multiply it with the transpose of scaled version of original dataset.
$NewData = FeatureVector^T \times ScaledData^T$
Here, $NewData$ is the Matrix consisting of the principal components, $FeatureVector$ is the matrix we formed using the eigenvectors we chose to keep, and $ScaledData$ is the scaled version of original dataset ('T' in the superscript denotes transpose of a matrix which is formed by interchanging the rows to columns and vice versa. In particular, a 2×3 matrix has a transpose of size 3×2) If we go back to the theory of eigenvalues and eigenvectors, we see that, essentially, eigenvectors provide us with information about the patterns in the data. In particular, in the running example of 2-D set, if we plot the eigenvectors on the scatter plot of data, we find that the principal eigenvector (corresponding to the largest eigenvalue) actually fits well with the data. The other one, being perpendicular to it, does not carry much information and hence, we are at not much loss when deprecating it, hence reducing the dimension.
All the eigenvectors of a matrix are perpendicular to each other. So, in PCA, what we do is represent or transform the original dataset using these

orthogonal (perpendicular) eigenvectors instead of representing on normal x and y axes. We have now classified our data points as a combination of contributions from both x and y. The difference lies when we actually disregard one or many eigenvectors, hence, reducing the dimension of the dataset. Otherwise, in case, we take all the eigenvectors in account, we are just transforming the co-ordinates and hence, not serving the purpose.

7 Independent Component Analysis

Independent component analysis (ICA) is a method for finding underlying factors or components from multivariate (multi-dimensional) statistical data. What distinguishes ICA from other methods is that it looks for components that are both statistically independent, and non-Gaussian. Independent component analysis (ICA) is a statistical and computational technique for revealing hidden factors that underlie sets of random variables, measurements, or signals.

ICA defines a generative model for the observed multivariate data, which is typically given as a large database of samples. In the model, the data variables are assumed to be linear mixtures of some unknown latent variables, and the mixing system is also unknown. The latent variables are assumed nongaussian and mutually independent, and they are called the independent components of the observed data. These independent components, also called sources or factors, can be found by ICA.

The data analyzed by ICA could originate from many different kinds of application fields, including digital images, document databases, economic indicators and psychometric measurements. In many cases, the measurements are given as a set of parallel signals or time series; the term blind source separation is used to characterize this problem. Typical examples are mixtures of simultaneous speech signals that have been picked up by several microphones, brain waves recorded by multiple sensors, interfering radio signals arriving at a mobile phone, or parallel time series obtained from some industrial process.

7.1 Definition of ICA

In practical situations, we cannot in general find a representation where the components are really independent, but we can at least find components that are as independent as possible. This leads us to the following definition of ICA. Given a set of observations of random variables $(x_1(t), (x_2(t), \ldots, (x_n(t))$, where t is the time or sample index, assume that they are generated as a linear mixture

of independent components:

$$\begin{pmatrix} x_1(t) \\ x_2(t) \\ \vdots \\ x_n(t) \end{pmatrix} = A \begin{pmatrix} s_1(t) \\ s_2(t) \\ \vdots \\ s_n(t) \end{pmatrix} \tag{1}$$

where A is some unknown matrix. Independent component analysis now consists of estimating both the matrix A and the $si(t)$, when we only observe the $xi(t)$. Note that we assumed here that the number of independent components s_i is equal to the number of observed variables; this is a simplifying assumption that is not completely necessary.
Alternatively,we could define ICA as follows: find a linear transformation given by a matrix W, so that the random variables $y_i, i = 1, \ldots, n$ are as independent as possible. Since after estimating A its inverse gives W.
It can be shown that the problem is well defined, that is, the model in eq. 1 can be estimated if and only if the components s_i are nongaussian. This is a fundamental requirement that also explains the main difference between ICA and factor analysis, in which the nongaussianity of the data is not taken into account. In fact, ICA could be considered as nongaussian factor analysis, since in factor analysis, we are also modeling the data as linear mixtures of some underlying factors.

7.2 Applications

Due to its generality the ICA model has applications in many different areas. Some examples are:

- In brain imaging, we often have different sources in the brain emit signals that are mixed up in the sensors outside of the head, just like in the basic blind source separation model.
- In econometrics, we often have parallel time series, and ICA could decompose them into independent components that would give an insight to the structure of the data set.
- A somewhat different application is in image feature extraction, where we want to find features that are as independent as possible

7.3 ICA estimation principles

- Nonlinear decorrelation: Find the matrix W so that for any $i \neq j$, the components y_i and y_j are uncorrelated, and the transformed components $g(y_i)$ and $h(y_j)$ are uncorrelated, where g and h are some suitable nonlinear functions.
- Maximum nongaussianity: Find the local maxima of nongaussianity of a linear combination $y = Wx$ under the constraint that the variance of x is constant.

- Each local maximum gives one independent component.
- Assumptions have many like Source signals are statistically independent knowing the value of one of the components does not give any information about the others ICs have non-gaussian distributions. There are initial distributions unknown. Currently at most one Gaussian source only the recovered sources can be permutated and scaled.

8 Latent Semantic Indexing

Latent semantic indexing, sometimes referred to as latent semantic analysis, is a mathematical method developed in the late 1980s to improve the accuracy of information retrieval. It uses a technique called singular value decomposition to scan unstructured data within documents and identify relationships between the concepts contained therein. In essence, it finds the hidden (latent) relationships between words (semantics) in order to improve information understanding (indexing). It provided a significant step forward for the field of text comprehension as it accounted for the contextual nature of language. Earlier technologies struggled with the use of synonyms that characterizes natural language use, and also the changes in meanings that come with new surroundings.

For example, the words 'hot' and 'dog' may seem easy to understand, but both have multiple definitions based on how they are used. Put both of them together and you have a whole new concept altogether.

So how can we train a machine to adapt to these nuances? This is a problem that has troubled great minds for centuries and LSI has helped computers to start understanding language in use. It works best on static content and on small sets of documents, which was great for its initial purposes. LSI also allows documents to be clustered together based on their thematic commonalities, which was a very useful capability for early search engines.

8.1 Basic concepts

Latent Semantic Indexing is a technique that projects queries and documents into a space with "latent" semantic dimensions. In the latent semantic space, a query and a document can have high cosine similarity even if they do not share any terms - as long as their terms are semantically similar in a sense to be described later. We can look at LSI as a similarity metric that is an alternative to word overlap measures like tf.idf.

The latent semantic space that we project into has fewer dimensions than the original space (which has as many dimensions as terms). LSI is thus a method for dimensionality reduction. A dimensionality reduction technique takes a set of objects that exist in a high-dimensional space and represents them in a low

dimensional space, often in a two-dimensional or three-dimensional space for the purpose of visualization.

Latent semantic indexing is the application of a particular mathematical technique, called Singular Value Decomposition or SVD, to a word-by-document matrix. SVD (and hence LSI) is a least-squares method. The projection into the latent semantic space is chosen such that the representations in the original space are changed as little as possible when measured by the sum of the squares of the differences.

SVD takes a matrix A and represents it as $\hat{A}$ in a lower dimensional space such that the "distance" between the two matrices as measured by the 2-norm is minimized:

$$\Delta = \|A - \hat{A}\|_2 \tag{2}$$

The 2-norm for matrices is the equivalent of Euclidean distance for vectors. SVD project an n-dimensional space onto a k-dimensional space where $n >> k$. In our application (word-document matrices), n is the number of word types in the collection. Values of k that are frequently chosen are 100 and 150. The projection transforms a document's vector in n-dimensional word space into a vector in the k-dimensional reduced space.

There are many different mappings from high dimensional to low-dimensional spaces. Latent Semantic Indexing chooses the mapping that is optimal in the sense that it minimizes the distance Δ. This setup has the consequence that the dimensions of the reduced space correspond to the axes of greatest variation. The SVD projection is computed by decomposing the document-by-term matrix $A_{t\times d}$ into the product of three matrices, $T_{t\times n}, S_{n\times n}, D_{d\times n}$:

$$A_{t\times d} = T_{t\times n} S_{n\times n} \left(D_{d\times n}\right)^T$$

where t is the number of terms, d is the number of documents, $n = min(t, d)$, T and D have orthonormal columns, i.e. $TT^T = DD^T = I$, $rank(A) = r$, $S = diag(\sigma_1, \sigma_2, \ldots, \sigma_n)$, $\sigma_i > 0$ for $1 \leq i \leq r, \sigma_j = 0$ for $j \geq r + 1$.

We can view SVD as a method for rotating the axes of the n-dimensional space such that the first axis runs along the direction of largest variation among the documents, the second dimension runs along the direction with the second largest variation and so forth. The matrices T and D represent terms and documents in this new space. The diagonal matrix S contains the singular values of A in descending order. The i^{th} singular value indicates the amount of variation along the i^{th} axis. By restricting the matrices T, S and D to their first $k < n$ rows one obtains the matrices $T_{t\times n}, S_{n\times n}, D_{d\times n}$. Their product $\hat{A}$ defined in Eq. 3 is the best square approximation of A by a matrix of rank k in the sense defined in the equation Eq. 2.

$$\hat{A}_{t\times d} = T_{t\times k} S_{k\times k} \left(D_{d\times k}\right)^T \tag{3}$$

Choosing the number of dimensions k for $\hat{A}$ is an interesting problem. While a reduction in k can remove much of the noise, keeping too few dimensions or

factors may loose important information. LSI performance can improve considerably after 10 or 20 dimensions, peaks between 70 and 100 dimensions, and then begins to diminish slowly. This pattern of performance (initial large increase and slow decrease to word-based performance) is observed with other datasets as well. Eventually performance must approach the level of performance attained by standard vector methods, since with $k = n$ factors $\hat{A}$ will exactly reconstruct the original term by document matrix A. That LSI works well with a relatively small (compared to the number of unique terms) number of dimensions or factors k shows that these dimensions are, in fact, capturing a major portion of the meaningful structure.

One can also prove that SVD is unique, that is, there is only one possible decomposition of a given matrix. That SVD finds the optimal projection to a low dimensional space is the key property for exploiting word co-occurrence patterns. It is important for the LSI method that the derived $\hat{A}$ matrix does not reconstruct the original term document matrix A exactly. The truncated SVD, in one sense, captures most of the important underlying structure in the association of terms and documents, yet at the same time removes the noise or variability in word usage that plagues word-based retrieval methods. Intuitively, since the number of dimensions, k is much smaller than the number of unique terms t, minor differences in terminology will be ignored. Terms which occur in similar documents, for example, will be near each other in the k-dimensional factor space even if they never co-occur in the same document. This means that some documents, which do not share any words with a user's query, may nonetheless be near it in k-space. This derived representation, which captures term-term associations, is used for retrieval.

8.2 Advantages and Disadvantages

Advantages

1 **True (latent) dimensions**
The assumption in LSI (and similarly for other forms of dimensionality reduction like principal component analysis) is that the new dimensions are a better representation of documents and queries. The metaphor underlying the term "latent" is that these new dimensions are the true representation. This true representation was then obscured by a generation process that expressed a particular dimension with one set of words in some documents and a different set of words in another document. LSI analysis recovers the original semantic structure of the space and its original dimensions.

2 **Synonymy**
Synonymy refers to the fact that the same underlying concept can be described using different terms. Traditional retrieval strategies have trouble discovering documents on the same topic that use a different vocabulary. In LSI, the concept in question as well as all documents that are related to it are all likely to be represented by a similar weighted combination of indexing variables.

3 **Polysemy**
Polysemy describes words that have more than one meaning, which is common property of language. Large numbers of polysemous words in the query can reduce the precision of a search significantly. By using a reduced representation in LSI, one hopes to remove some "noise" from the data, which could be described as rare and less important usages of certain terms. (Note however that this would work only when the real meaning is close to the average meaning. Since the LSI term vector is just a weighted average of the different meanings of the term, when the real meaning differs from the average meaning, LSI may actually reduce the quality of the search).

4 **Term Dependence**
The traditional vector space model assumes term independence and terms serve as the orthogonal basis vectors of the vector space. Since there are strong associations between terms in language, this assumption is never satisfied. While term independence represents the most reasonable first-order approximation, it should be possible to obtain improved performance by using term associations in the retrieval process. Adding common phrases as search items is a simple application of this approach. On the other hand, the LSI factors are orthogonal by definition, and terms are positioned in the reduced space in a way that reflects the correlations in their use across documents. It is very difficult to take advantage of term associations without dramatically increasing the computational requirements of the retrieval problem. While the LSI solution is difficult to compute for large collections, it need only be constructed once for the entire collection and performance at retrieval time is not affected.

Disadvantages

1 **Storage**
One could also argue that the SVD representation is more compact. Many documents have more than 150 unique terms. So the sparse vector representation will take up more storage space than the compact SVD representation if we reduce to 150 dimensions. In reality, the opposite is actually true. For example, the document by term matrix for the Cranfield collection used in Hull's experiments had 90,441 non-zero entries (after stemming and stop word removal). Retaining only 100 of the possible 1399 LSI vectors requires storing 139,900 values for the documents alone. The term vectors require the storage of roughly 400,000 additional values. In addition, the LSI values are real numbers while the original term frequencies are integers, adding to the storage costs. Using LSI vectors, we can no longer take advantage of the fact that each term occurs in a limited number of documents, which accounts for the sparse nature of the term by document matrix. With recent advances in electronic storage media, the storage requirements of LSI are not a critical problem, but the loss of sparseness has other, more serious implications.

2 **Efficiency**

One of the most important speed-ups in vector space search comes from using an inverted index. As a consequence, only documents that have some terms in common with the query must be examined during the search. With LSI, however, the query must be compared to every document in the collection. There are, however, several factors that can reduce or eliminate this drawback. If the query has more terms than its representation in the LSI vector space, then inner product similarity scores will take more time to compute in term space. For example, if relevance feedback is conducted using the full text of the relevant documents, the number of terms in the query is likely to grow to be many times the number of LSI vectors, leading to a corresponding increase in search time. In addition, using a data structure such as the k-d tree in conjunction with LSI would greatly speed the search for nearest neighbors, provided only a partial ordering of the documents is required. Most of the additional costs come in the preprocessing stage when the SVD and the k-d tree are computed, and actual search time should not be significantly degraded. Other query expansion techniques suffer even more heavily from the difficulties described above, and LSI performs relatively well for long documents due to the small number of context vectors used to describe each document. However, implementation of LSI does require an additional investment of storage and computing time.

3 **LSI and normally-distributed data**

Another obection to SVD is that, along with all other least-squares methods, it is really designed for normally-distributed data, but such a distribution is inappropriate for count data, and count data is what a term-by-document matrix consists of. The link between least squares and normal distribution can be easily seen by looking at the definition of the normal distribution.

4 **Toward a theoretical foundation**

Although (little) empirical improved performance has been observed, there is very little in the literature in the way of a mathematical theory that predicts this improved performance. In this session I briefly describe one paper that is an attempt at using mathematical techniques to rigorously explain the empirically observed improved performance of LSI, Papadimitriou starts citating an interesting mathematical fact due to Eckart and Young, often cited as an explanation of the improved performance of LSI, that states, informally, that LSI retains as much as possible the relative position (and distances) of the document vectors while projecting it to a lower-dimensional space. This may only provide an explanation of why LSI does not deteriorate too much in performance over conventional vector-space methods; it fails to justify the observed improvement in precision and recall.

8.3 Applications of LSI

1 **Information retrieval**
The application of Singular Value Decomposition to information retrieval was originally proposed by a group of researchers at Bellcore and called Latent Semantic Indexing in this context. At this point it should be clear how to use LSI for IR. Regarding the performances, reports that for several information science test collections, the average precision using LSI ranged from comparable to 30% better than that obtained using standard keyword vector methods. The LSI method performs best relative to standard vector methods when the queries and relevant documents do not share many words, and at high levels of recall.

2 **Relevance Feedback**
Most of the tests of Relevance Feedback using LSI have involved a method in which the initial query is replaced with the vector sum of the documents the users has selected as relevant. The use of negative information has not yet been exploited in LSI; for example, by moving the query away from documents which the user has indicated are irrelevant. Replacing the users' query with the first relevant document improves performance by an average of 33% and replacing it with the average of the first three relevant documents improves performance by an average of 67%. Relevance feedback provides sizeable and consistent retrieval advantages. One way of thinking about the success of these methods is that many words (those from relevant documents) augment the initial query that is usually quite impoverished. LSI does some of this kind of query expansion or enhancement even without relevance information, but can be augmented with relevance information.

3 **Information Filtering**
Applying LSI to information filtering applications is straightforward. An initial sample of documents is analyzed using standard LSI/SVD tools. A users' interest is represented as one (or more) vectors in this reduced-dimension LSI space. Each new document is matched against the vector and if it is similar enough to the interest vector it is recommended to the user. Learning methods like relevance feedback can be used to improve the representation of interest vectors over time. Performances studies are encouraging.

4 **Cross-Language Retrieval**
It is important to note that the LSI analysis makes no use of English syntax or semantics. This means that LSI is applicable to any language. In addition, it can be used for cross-language retrieval - documents are in several languages and user queries (again in several languages) can match documents in any language. What is required for cross-language applications is a common space in which words from many languages are represented.

5 **Matching People Instead of Documents**
In a couple of applications, LSI has been used to return the best matching people instead of documents. In these applications, people were represented by articles they had written. In one application, known as the Bellcore Advisor, a system was developed to find local experts relevant to users' queries. A query was matched to the nearest documents and project descriptions and the authors' organization was returned as the most relevant internal group. In another application, LSI was used to automate the assignment of reviewers to submitted conference papers. Several hundred reviewers were described by means of texts they had written, and this formed the basis of the LSI analysis. Hundreds of submitted papers were represented by their abstracts, and matched to the closest reviewers. These LSI similarities were used to assign papers to reviewers for a major human-computer interaction conference. Subsequent analyses suggested that these completely automatic assignments (which took less than 1 hour) were as good as those of human experts.

6 **Noisy Input**
Because LSI does not depend on literal keyword matching, it is especially usefulwhen the text input is noisy, as in OCR (Optical Character Reader), open input, or spelling errors. If there are scanning errors and a word (Dumais) is misspelled (as Duniais), many of the other words in the document will be spelled correctly. If these correctly spelled context words also occur in documents that contained a correctly spelled version of Dumais, then Dumais will probably be near Dunials in the k-dimensional space determined by $\hat{A}$.

9 Spectral clustering

Clustering is a widely used unsupervised learning method. The grouping is such that points in a cluster are similar to each other, and less similar to points in other clusters. Thus, it is up to the algorithm to find patterns in the data and group it for us and, depending on the algorithm used, we may end up with different clusters. There are 2 broad approaches for clustering Fig. 1:

1. **Compactness Points** that lie close to each other fall in the same cluster and are compact around the cluster center. The closeness can be measured by the distance between the observations. E.g. K-Means Clustering.

2. **Connectivity Points** that are connected or immediately next to each other are put in the same cluster. Even if the distance between 2 points is less, if they are not connected, they are not clustered together. Spectral clustering is a technique that follows this approach.

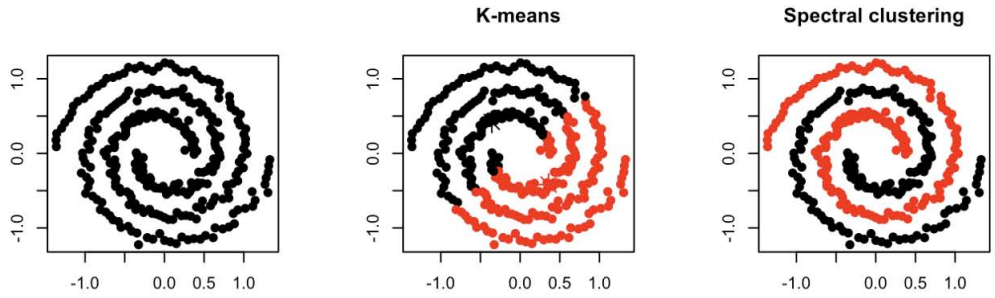

Fig. 1 Difference between K-Means and Spectral Clustering

9.1 Spectral Clustering Algorithm

In spectral clustering, the data points are treated as nodes of a graph. Thus, clustering is treated as a graph partitioning problem. The nodes are then mapped to a low-dimensional space that can be easily segregated to form clusters. An important point to note is that no assumption is made about the shape/form of the clusters. Spectral clustering involves 3 steps:

1. Compute a similarity graph:
   We first create an undirected graph $G = (V, E)$ with vertex set $V = v1, v2, \ldots, vn = 1, 2, \ldots, n$ observations in the data. This can be represented by an adjacency matrix which has the similarity between each vertex as its elements . To do this, we can either compute: The ϵ-neighborhood graph, KNN Graph or Fully connected graph.

2. Project the data onto a low-dimensional space
   As we can see in Fig. 1, data points in the same cluster may also be far away–even farther away than points in different clusters. Our goal then is to transform the space so that when the 2 points are close, they are always in same cluster, and when they are far apart, they are in different clusters. We need to project our observations into a low-dimensional space. For this, we compute the Graph Laplacian, which is just another matrix representation of a graph and can be useful in finding interesting properties of a graph.

3. Create clusters
   We use the eigenvector corresponding to the 2nd eigenvalue to assign values to each node. To get bipartite clustering (2 distinct clusters), we first assign each element of v2 to the nodes. We then split the nodes such that all nodes with value ¿ 0 are in one cluster, and all other nodes are in the other cluster. It is important to note that the 2nd eigenvalue indicates how tightly connected the nodes are in the graph. For good, clean partitioning, lower the 2nd eigenvalue, better the clusters. For k clusters, we have to modify our Laplacian to normalize it.

Normalized Spectral Clustering Algorithm

1. Input: Similarity matrix $S \in \mathbb{R}^{n \times n}$, number k of clusters to construct.
2. Construct a similarity graph by one of the ways described above. Let W be its weighted adjacency matrix.
3. Compute the normalized Laplacian L_{sym}.
4. Compute the first k eigenvectors $u_1, u_2, \ldots, u_k$ of L_{sym}.
5. Let $U \in \mathbb{R}^{n \times k}$ be the matrix containing the vectors $u_1, u_2, \ldots, u_k$ as columns.
6. Form the matrix $T \in \mathbb{R}^{n \times k}$ from U by normalizing the rows to norm 1, that is set $t_{ij} = \frac{u_{ij}}{\sqrt{\sum_k u_{ik}^2}}$
7. For $i = 1, \ldots, n$, let $y_i \in \mathbb{R}^k$ be the vector corresponding to the i^{th} row of T.
8. Cluster the points (y_i) $i = 1, ..., n$ with the k-means algorithm into clusters $C_1, C_2, \ldots, C_k$.
9. Output: Clusters $A_1, A_2, \ldots, A_k$ with $A_i = \{j | y_j \in C_i\}$.

9.2 Advantages and Disadvantages

Advantages:

- Does not make strong assumptions on the statistics of the clusters. Clustering techniques like K-Means Clustering assume that the points assigned to a cluster are spherical about the cluster center. This is a strong assumption to make, and may not always be relevant. In such cases, spectral clustering helps create more accurate clusters.
- Easy to implement and gives good clustering results. It can correctly cluster observations that actually belong to the same cluster but are farther off than observations in other clusters due to dimension reduction.
- Reasonably fast for sparse data sets of several thousand elements.

Disadvantages:

- Use of K-Means clustering in the final step implies that the clusters are not always the same. They may vary depending on the choice of initial centroids.
- Computationally expensive for large datasets. This is because eigenvalues and eigenvectors need to be computed and then we have to do clustering on these vectors. For large, dense datasets, this may increase time complexity quite a bit.

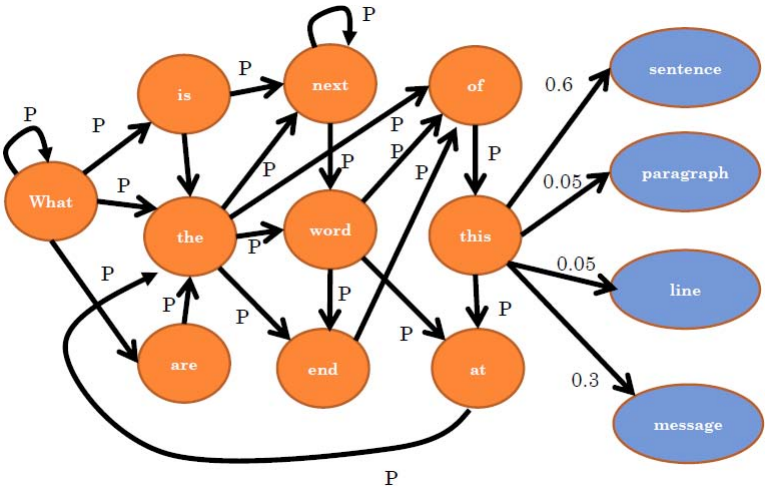

Fig. 2 Markov Chain

10 Markov Model

In probability theory, a Markov model is a stochastic model used to model randomly changing systems. It is assumed that future states depend only on the current state, not on the events that occurred before it (that is, it assumes the Markov property). Generally, this assumption enables reasoning and computation with the model that would otherwise be intractable. For this reason, in the fields of predictive modelling and probabilistic forecasting, it is desirable for a given model to exhibit the Markov property.

10.1 Markov Process

A Markov process is a process that is capable of being in more than one state, can make transitions among those states, and in which the states available and transition probabilities depend only upon what state the system is currently in. In other words, there is no memory in a Markov process.

10.2 Markov Chain

A Markov Chain is a statistical model of a system that moves sequentially from one state to another Fig. 2. The probabilities of transition from one state to another are dependent only on the current state (not on previous states). Generally modeled as a stochastic process. A Markov chain can be described by a transition matrix.

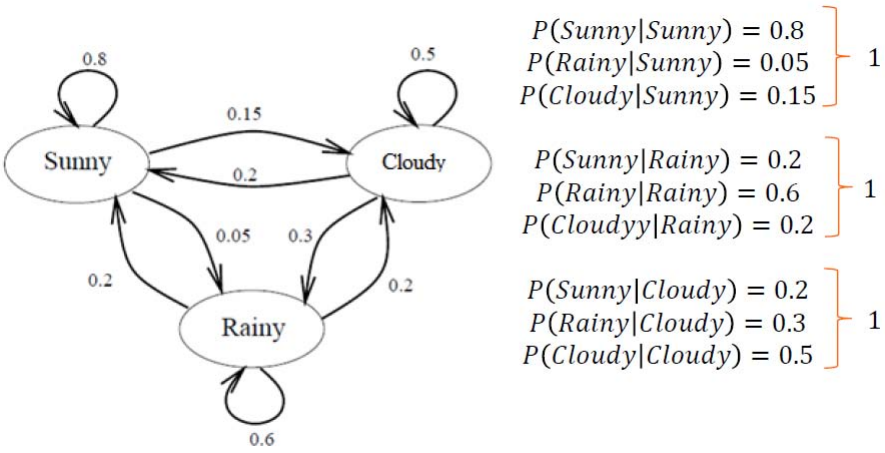

Fig. 3 Markov Chain for weather

10.3 Example of Markov Chain

Design a Markov Chain to predict the weather of tomorrow using previous information of the past days.

- Our model has only 3 states: $\{S_1, S_2, S_3\}$ and the name of each state is $S_1 = Sunny, S_2 = Rainy, S_3 = Cloudy$.
- To establish the transition probabilities relationship between states we will need to collect data.
- Assume the data produces the following transition probabilities Fig. 3:
- Let's say we have a sequence: $Sunny, Rainy, Cloudy, Cloudy, Sunny, Sunny, Sunny, Rainy, \ldots$; so, in a day we can be in any of the three states.
- We can use the following state sequence notation: $q_1, q_2, q_3, q_4, \ldots$ where $q_i \in \{Sunny, Rainy, Cloudy\}$.
- In order to compute the probability of tomorrow's weather we can use the Markov property: $p(q_1, \ldots, q_n) = \Pi_{i=1}^{n} P(q_i|q_{i-1})$

Exercise 1: Given that today is Sunny, what's the probability that tomorrow is Sunny and the next day Rainy?
Solution:

$$
\begin{aligned}
P(q_2, q_3|q_1) &= P(q_2|q_1)P(q_3|q_1, |q_2) \\
&= P(q_2|q_1)P(q_3|q_2) \\
&= P(Sunny|Sunny)|P(Rainy|Sunny) \\
&= 0.8 \times 0.05 \\
&= 0.04
\end{aligned}
$$

Exercise 2: Assume that yesterday's weather was Rainy, and today is Cloudy, what is the probability that tomorrow will be Sunny?
Solution:

$$
\begin{aligned}
P(q_3|q_1, q_2) &= P(q_3|q_2) \\
&= P(Sunny|Cloudy) \\
&= 0.2
\end{aligned}
$$

10.4 Hidden Markov Model

Hidden Markov Model (HMM) is a statistical Markov model in which the system being modelled is assumed to be a Markov process with unobservable (i.e. hidden) states. The hidden Markov model can be represented as the simplest dynamic Bayesian network. The mathematics behind the HMM were developed by L. E. Baum and co-workers. In simpler Markov models (like a Markov chain), the state is directly visible to the observer, and therefore the state transition probabilities are the only parameters, while in the hidden Markov model, the state is not directly visible, but the output (in the form of data or "token" in the following), dependent on the state, is visible. Each state has a probability distribution over the possible output tokens. Therefore, the sequence of tokens generated by an HMM gives some information about the sequence of states; this is also known as pattern theory, a topic of grammar induction.

Hidden Markov models are especially known for their application in reinforcement learning and temporal pattern recognition such as speech, handwriting, gesture recognition, part-of-speech tagging, musical score following, partial discharges and bio-informatics. A hidden Markov model can be considered a generalization of a mixture model where the hidden variables (or latent variables), which control the mixture component to be selected for each observation, are related through a Markov process rather than independent of each other. Recently, hidden Markov models have been generalized to pairwise Markov models and triplet Markov models which allow consideration of more complex data structures and the modelling of non-stationary data.

10.5 HMM Terminology

A HMM Model is specified by:

- The set of states $S = \{s_1, s_2, \ldots, s_N\}$ and a set of parameters $\Theta = \{\pi, A, B\}$

- The prior probabilities $\pi_i = P(q_1 = s_i)$ are the probabilities of s_i being the first state of a state sequence collected in a vector π.

- The transition probabilities are the probabilities to go from state i to state j: $a_{i,j} = P(q_{n+1} = s_j | q_n = s_i)$. They are collected in the matrix A.

- The emission probabilities characterize the likelihood of a certain observation x, if the model is in state s_i. Depending on the kind of observation x we have:

  - for discrete observations, $x_n \in \{v_1, \ldots, v_K\} : b_{i,k} = P(x_n = v_k | q_n = s_i)$, the probabilities to observe v_k if the current state is $q_n = s_i$. The numbers $b_{i,k}$ can be collected in a matrix B.
  - for continuous valued observations, e.g., $x_n \in \mathbb{R}^D$: A set of functions $b_i(x_n) = p(x_n | q_n = s_i)$ describing the probability densities functions over the observation space for the system being in state s_i. Collected in the vector $B(x)$ of functions.

The operation of a HMM is characterized by

- The (hidden) state sequence $Q = \{q_1, q_2, \ldots, q_N\}$, $q_n \in S$.

- The observation sequence $X = \{x_1, x_2, \ldots, x_N\}$.

A HMM allowing for transitions from any emitting state to any other emitting state is called an ergodic HMM. The other extreme, a HMM where the transitions only go from one state to itself or to a unique follower is called a left-right HMM. **Useful formula:**

- **Probability of a state sequence**: the probability of a state sequence $Q = \{q_1, q_2, \ldots, q_N\}$ coming from a HMM with parameters Θ corresponds to the product of the transition probabilities from one state to the following:

$$P(Q|\Theta) = \pi_{q_1} \Pi_{n=1}^{N-1} a_{q_n, q_{n+1}} = \pi_{q_1} . a_{q_1, q_2} . a_{q_2, q_3} . \ldots . a_{q_{N-1}, q_N} .$$

- **Likelihood of an observation sequence given a state sequence, or likelihood of an observation sequence along a single path**: given an observation sequence $X = \{x_1, x_2, \ldots, x_N\}$ and a state sequence $Q = \{q_1, q_2, \ldots, q_N\}$ (of the same length) determined from a HMM with parameters Θ, the likelihood of X along the path Q is equal to:

$$P(X|Q, \Theta) = \Pi_{n=1}^{N} P(x_n | q_n, \Theta) = b_{q_1, x_1} . b_{q_2, x_2} . \ldots . b_{q_N, x_N}$$

  i.e., it is the product of the emission probabilities computed along the considered path.

- **Joint likelihood of an observation sequence X and a path Q**: it is the probability that X and Q occur simultaneously, $P(X, Q|\Theta)$, and decomposes into a product of the two quantities defined previously:

$$P(X, Q|\Theta) = P(X|Q, \Theta).P(Q|\Theta) \text{ (Bayes)}$$

- **Likelihood of a sequence with respect to a HMM**: the likelihood of an observation sequence X = x1, x2, . . . , xN with respect to a Hidden Markov Model with parameters Θ expands as follows:

$$P(X|\Theta) = \sum_{all\ Q} P(X, Q|\Theta)$$

  i.e., it is the sum of the joint likelihoods of the sequence over all possible state sequences Q allowed by the model.

10.6 Applications of HMM

- Speech recognition
- Handwriting recognition
- Gesture recognition
- Speech tagging
- Musical score following
- Bioinformatics
- Data compression
- Computer vision applications