Lecture 4 Machine Learning in Passive Investment
Smart Beta Strategy

Step 1. Rank all assets using exposure to each of the four metrics: β

1. MOMENTUM
2. SIZE
   - book value (Book)
3. VALUE
   - trailing five-year average cash flow (Cash Flow)
   - trailing five-year average revenue (Revenue)
   - trailing five-year average gross sales (Sales)
   - trailing five-year average gross dividends (Dividends)
4. MACRO
   - total employment (Employment)
   - inflation (CPI or PCE)

Step 2. Select the 1,000 largest by each metric. Each of these 1,000 largest was included in the index at its relative metric weight β, in which way, we create the Fundamental index for that metric.

Step 3. The portfolio were rebalanced once every quarter, using only data available on the last trading day of the prior quarter.

Smart Beta v.s. Active Alpha

Smart Beta is often characterized as passive investing for the following reasons:

- It uses rule-based selection and weighting
- It rebalances at predetermined intervals
- It does not attempt to make explicit forecasts of returns and risk for individual securities.
- It requires additional active decisions to identify the specific factors to target, and to define the factors, the selection universe, the weighting method, and the rebalancing rules.
- All smart beta portfolios are concentrated in terms of the sources of excess return they seek to exploit, such as size, value, momentum, low volatility, or some combination of a few factors. This concentration leaves them susceptible to periods of poor returns to the chosen factors, which can lead to inconsistent performance.

Active Alpha strategy is active in several ways:

- It identifies numerous potentially profitable relationships between factors and prices.
- It forecasts returns to factors and individual securities, estimates associated risks, and revises forecasts based on the changing market environment.
- regularities in price response to a wide range of independent fundamental and behavioral factors. Portfolios are diversified across exposures to numerous factors.
- Risk models take on the same form as the alpha models. The pricing model interprets the equation by expectation, but the risk model interprets the equation by variance. The main challenge is that some factors may not be priced or rewarded unconditionally through time, but they do differentiate cross-sectional security returns. **These risk factors are mainly from three categories:**
  1. Macroeconomic factors (interest rates),
  2. Fundamental factors
  3. Statistical factors (principal component analysis).
- One can actively uses the risk budget to construct portfolios instead of passively monitoring portfolio risk contribution (risk budgeting).

Unsupervised Learning and Smart Beta

Smart beta strategy only selects one factor (metric) each time when the portfolio rebalances. In order to incorporate more than one factors (metrics, say momentum + value), one may use unsupervised learning methods to group stocks by their βs. In this part, we will discuss three main methods for classification and clustering.

For classification, we consider:

- Linear Classifiers: Logistic Regression, Naive Bayes Classifier
- Nearest Neighbor
- Support Vector Machines
- Decision Trees
- Boosted Trees
- Random Forest
- Neural Networks

For clustering, we are going to introduce:

- hierarchical clustering
- *k*-mean
- model-based clustering

Classification

Logistic Regression

Logistic regression is a calculation used to predict a binary outcome: either something happens, or does not. This can be exhibited as Yes/No, Positive/Negative, Rise/Fall, etc. Independent variables are analyzed to determine the binary outcome with the results falling into one of two categories. The independent variables can be categorical or numeric, but the dependent variable is always categorical. Written like this:

$$P(Y = 1|X) or\ P(Y = 0|X)$$

For example, we can predict whether it will rain today or not, based on the current weather conditions. Two of the important parts of logistic regression are **Hypothesis and Sigmoid Curve**. With the help of this hypothesis, we can derive the likelihood of the event. The data generated from this hypothesis can fit into the log function that creates an S-shaped curve known as "sigmoid". The logistic sigmoid function, a.k.a. the inverse logit function, is

$$g(x) = \frac{e^x}{1 + e^x}$$

Its outputs range from 0 to 1, and are often interpreted as probabilities (in, say, logistic regression).

We write the equation for logistic regression as follows:

$$y = \frac{\exp(\beta_0 + \beta_1 x)}{1 + \exp(\beta_0 + \beta_1 x)}$$

In the above equation, β_0 andβ_1 are the two coefficients of the input x. We estimate these two coefficients using Maximum Likelihood Estimation.

Using this log function, we can further predict the category of class.

Naïve Bayes Algorithm

Naive Bayes calculates the possibility of whether a data point belongs within a certain category or does not. In text analysis, it can be used to categorize words or phrases as belonging to a preset "tag" (classification) or not. For example:

Text	Tag
“A great game”	Sports
“The election was over”	Not sports
“Very clean match”	Sports
“A clean but forgettable game”	Sports
“It was a close election”	Not sports

To decide whether or not a phrase should be tagged as “sports,” you need to calculate:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

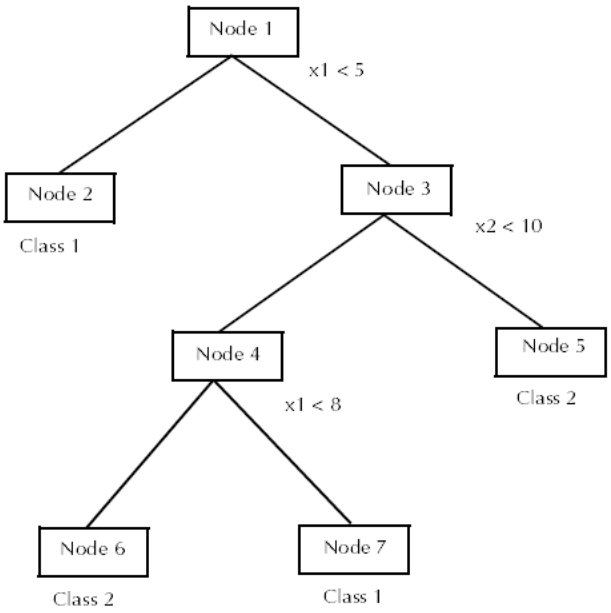
Or… the probability of A, if B is true, is equal to the probability of B, if A is true, times the probability of A being true, divided by the probability of B being true.

K-nearest Neighbors

K-nearest neighbors (k-NN) is a pattern recognition algorithm that uses training datasets to find the k closest relatives in future examples. When k-NN is used in classification, you calculate to place data within the category of its nearest neighbor. If k = 1, then it would be placed in the class nearest 1. K is classified by a plurality poll of its neighbors.

Decision Tree

A decision tree is a supervised learning algorithm that is perfect for classification problems, as it’s able to order classes on a precise level. It works like a flow chart, separating data points into two similar categories at a time from the “tree trunk” to “branches,” to “leaves,” where the categories become more finitely similar. This creates categories within categories, allowing for organic classification with limited human supervision. To continue with the sports example, this is how the decision tree works:



Notations that will be used to describe classification trees:

- L denotes a learning set made up of observed feature vectors and their class label.
- J denotes the number of classes.
- T is a classification tree.
- t represents a node in the tree.
- t_L and t_R are the left and right child nodes.
- $\{t_1\}$ is the tree containing only the root node.
- T_t is a branch of tree T starting at node t
- $\tilde{T}$ is the set of terminal nodes in the tree.
- $|\tilde{T}|$ is the number of terminal nodes in tree T.
- n is the total number of observations in the learning set.
- n_j is the number of observations in the learning set that belong to the j-th class $\omega_j = 1, \dots, J$.
- $n(t)$ is the number of observations that fall into node t
- $n_j(t)$ is the number of observations at node t that belong to class ω_j
- π_j is the prior probability that an observation belongs to class ω_j. This can be estimated from the data as

$$\pi_j = \frac{n_j}{n}$$

- $p(\omega_j, t)$ represents the joint probability that an observation will be in node t and it will belong to class ω_j. It is calculated using

$$p(\omega_j, t) = \frac{\pi_j n_j(t)}{n_j}$$

- $p(t)$ is the probability that an observation falls into node t and is given by

$$p(t) = \sum_{j=1}^{J} p(\omega_j, t)$$

- $p(\omega_j|t)$ denotes the probability that an observation is in class ω_j given it is in node t. This is calculated from

$$p(\omega_j|t) = \frac{p(\omega_j, t)}{p(t)}$$

We now discuss the splitting rule in more detail. When we split a node, our goal is to find a split that reduces the impurity in some manner. So, we need a measure of impurity $i(t)$ for a node t. Breiman, et al. [1984] discuss several possibilities, one of which is called the Gini diversity index. This is the one we will use in our implementation of classification trees. The Gini index is given by

$$i(t) = \sum_{i \neq j}^{J} p(\omega_i|t)p(\omega_j|t)$$

, which can also be rewritten as

$$i(t) = 1 - \sum_{j=1}^{J} p^2(\omega_j|t)$$

The reason the tree didn't continue growing is because Decision Trees always a growth-stop condition configured, otherwise they would grow until each training sample was separated into its own leaf node. These stop conditions are **maximum depth of the tree**, **minimum samples** in leaf nodes, or **minimum reduction in the error metric** (Hyperparameters).

Decision trees are built by recursively splitting our training samples using the features from the data that work best for the specific task. This is done by evaluating certain metrics, like the Gini index or the Entropy for categorical decision trees, or the Residual or Mean Squared Error for regression trees.

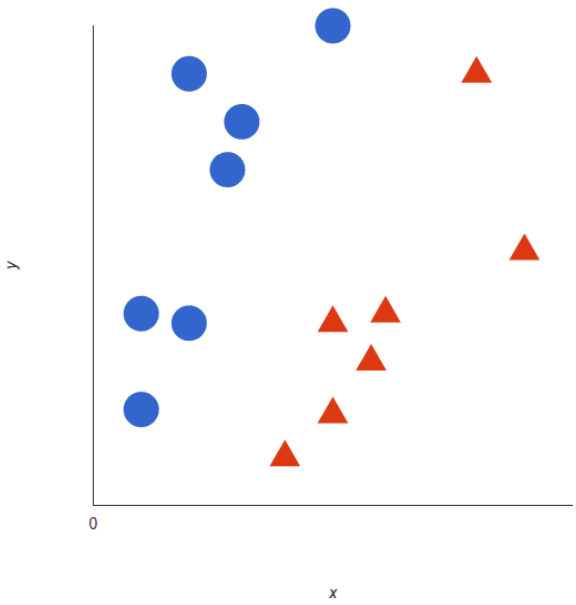
Random Forest

Motivation: a single decision tree normally does not make great predictions, so multiple trees are often combined to make 'forests' to give birth to stronger ensemble models.

The random forest algorithm is an expansion of decision tree, in that, you first construct some-axis real-world decision trees with training data, then fit your new data within one of the trees as a "random forest." It, essentially, averages your data to connect it to the nearest tree on the data scale. Random forest models are helpful as they remedy for the decision tree's problem of "forcing" data points within a category unnecessarily.

Support Vector Machines

A support vector machine (SVM) uses algorithms to train and classify data within degrees of polarity, taking it to a degree beyond X/Y prediction. For a simple visual explanation, we'll use two tags: red and blue, with two data features: X and Y, then train our classifier to output an X/Y coordinate as either red or blue.



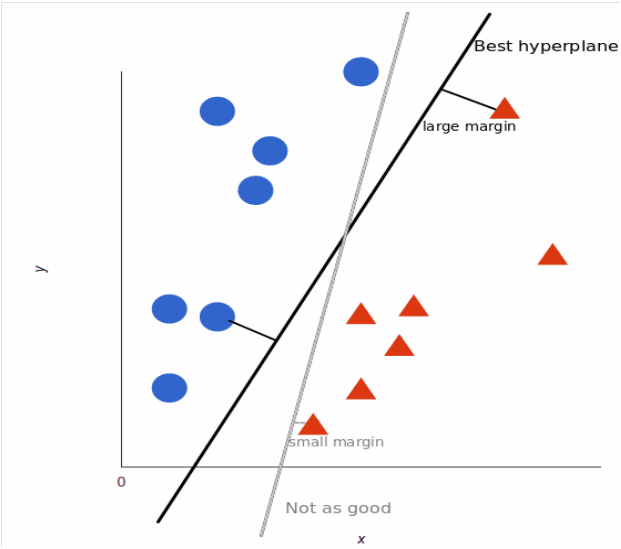
The SVM then assigns a hyperplane that best separates the tags. In two dimensions this is simply a line. Anything on one side of the line is red and anything on the other side is blue. In sentiment analysis, for example, this would be positive and negative. To minimize

$$J(w) = \frac{1}{2} w' w$$

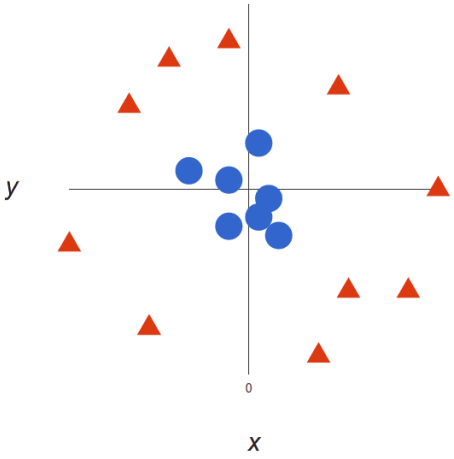
subject to all residuals having a value less than ε; or, in equation form:

$$\forall i: |y_i - wx_i| \leq \varepsilon|$$

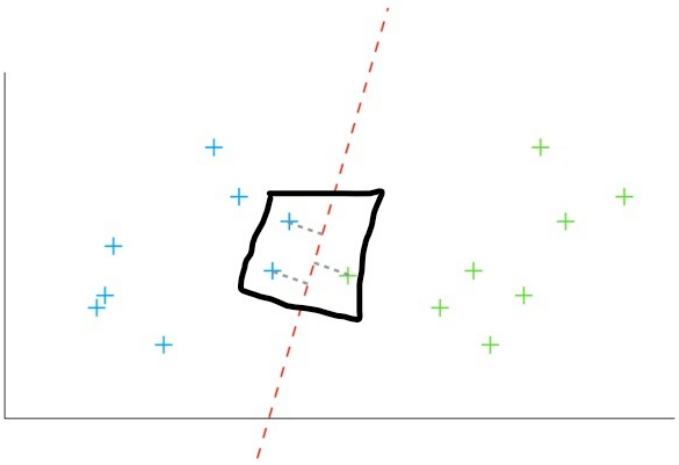
The best hyperplane is the one with the largest distance between each tag:



However, as data sets become more complex, it may not be possible to draw a single line to classify the data into two camps:



SVM is multidimensional. The model complexity is of O(n-features * n^2 samples) so it's perfect for working with data where the number of features is bigger than the number of samples. SVM does not use all the data points (vectors) to make this boundary. It only uses a few of these vectors that support (help) it to separate the points. These points (vectors) that support SVM are called Suport Vectors.



The data points in the black sqaure are Support Vectors. If you remove the other points from the feature space, SVM can still draw that hyperplane, making it work brilliantly for high-dimensional data.

Clustering

The choice of distance measures is a critical step in clustering. It defines how the similarity of two elements (x, y) is calculated and it will influence the shape of the clusters.

The classical methods for distance measures are Euclidean and Manhattan distances, which are defined as follow:
Euclidean distance:

$$d_{euc}(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

Manhattan distance:

$$d_{man} = \sum_{i=1}^{n}|x_i - y_i|$$

where, x and y are two vectors of length n.

Hierarchical Clustering
There are two types of hierarchical clustering methods: **agglomerative and divisive.**

- ***Divisive methods*** start with one large group and successively split the groups until there are *n* groups with some observations per group.

- ***Agglomerative methods*** are just the opposite; we start with *n* groups (one observation per group) and successively merge the two most similar groups until we are left with only one group. There are five commonly used methods for merging clusters in agglomerative clustering. These are single linkage, complete linkage, average linkage, centroid linkage and Ward's method.
    1. **The single linkage method** uses minimum distance, where the distance between clusters is defined as the distance between the closest pair of observations. Pairs consisting of one case from each group are used in the calculation. The first cluster is formed by merging the two groups with the shortest distance. Then the next smallest distance is found between all of the clusters (keep in mind that an observation is also a cluster). The two clusters corresponding to the smallest distance are then merged.
    2. The process for the **complete linkage method** is similar to single linkage, but the clustering criterion is different. The distance between groups is defined as the most distant pair of observations, with one coming from each group. The logic behind using this type of similarity criterion is that the maximum distance between observations in each cluster represents the smallest sphere that can enclose all of the objects in both clusters. Thus, the closest of these cluster pairs should be grouped together. The complete linkage method does not have the chaining problem that single linkage has.
    3. **The average linkage method** for clustering starts out the same way as single and complete linkage. In this case, the cluster criterion is the average distance between all pairs, where one member of the pair comes from each cluster. Thus, we find all pairwise distances between observations in each cluster and take the average. This linkage method tends to combine clusters with small variances and to produce clusters with approximately equal variance.
    4. **Centroid linkage** calculates the distance between two clusters as the distance between the centroids. The centroid of a cluster is defined as the *d*-dimensional sample mean for those observations that belong to the cluster. Whenever we merge clusters together or add an observation to a cluster, the centroid is recalculated.
    5. The distance between two clusters using **Ward's linkage method** is defined as the incremental sum of the squares between two clusters. To merge clusters, the within-group sum-of-squares is minimized over all possible partitions obtained by combining two clusters. The within-group sum-of-squares is defined as the sum of the squared distances between all observations in a cluster and its centroid. This method tends to produce clusters with approximately the same number of observations in each one.

If the clustering is a valid one, then there should be a strong correlation between them. We can measure this using the ***cophenetic correlation coefficient***. A cophenetic matrix is defined using the results of the linkage procedure. The ij-th entry of the cophenetic matrix is the fusion level at which the i-th and j-th objects appear together in the same cluster for the first time. The correlation coefficient between the distances and the corresponding cophenetic entries is the

cophenetic correlation coefficient. Large values indicate that the linkage provides a reasonable clustering of the data.

k-Means Clustering

The goal of *k*-means clustering is to partition the data into *k* groups such that the within-group sum-of-squares is minimized. One way this technique differs from hierarchical clustering is that we must specify the number of groups or clusters that we are looking for. We briefly describe two algorithms for obtaining clusters via *k*-means.

One of the basic algorithms for *k*-means clustering is a two step procedure. First, we assign observations to its closest group, usually using the Euclidean distance between the observation and the cluster centroid. The second step of the procedure is to calculate the new cluster centroid using the assigned objects. These steps are alternated until there are no changes in cluster membership or until the centroids do not change. This algorithm is sometimes referred to as HMEANS or the basic ISODATA method.

PROCEDURE - HMEANS ALGORITHM

1. Specify the number of clusters *k*.
2. Determine initial cluster centroids. These can be randomly chosen or the user can specify them.
3. Calculate the distance between each observation and each cluster centroid.
4. Assign every observation to the closest cluster.
5. Calculate the centroid (i.e., the *d*-dimensional mean) of every cluster using the observations that were just grouped there.
6. Repeat steps 3 through 5 until no more changes are made.

There are two problems with the HMEANS algorithm. The first one is that this method could lead to empty clusters, so users should be aware of this possibility. As the centroid is recalculated and observations are reassigned to groups, some clusters could become empty. The second issue concerns the optimality of the partitions. With *k*-means, we are searching for partitions where the within-group sum-of-squares is minimum. It can be shown that in some cases, the final *k*-means cluster assignment is not optimal, in the sense that moving a single point from one cluster to another may reduce the sum of squared errors. The following procedure helps address the second problem.

PROCEDURE - K-MEANS

1. Obtain a partition of *k* groups, possibly from the HMEANS algorithm.
2. Take each data point x_i and calculate the Euclidean distance between it and every cluster centroid.
3. Here x_i is in the *r*-th cluster, n, is the number of points in the *r*-th cluster, and d_{ir}^2 is the Euclidean distance between x_j and the centroid of cluster *r*. If there is a group *s* such that

$$\frac{n_r}{n_r - 1} d_{ir}^2 > \frac{n_s}{n_s + 1} d_{is}^2$$

then move x_i to cluster *s*.

4. If there are several clusters that satisfy the above inequality, then move the x_i to the group that has the smallest value for

$$\frac{n_s}{n_s + 1} d_{is}^2$$

5. Repeat steps 2 through 4 until no more changes are made.

There are also many algorithms for *k*-means clustering described in the literature.

Model-Based Clustering

Model-based clustering combines several ideas:

(1) ***Finite Mixtures Models and EM Algorithm*** is used to estimate the parameters of the finite mixture. Each component density of the finite mixture will represent a group or cluster.

(2) ***Model-based agglomerative clustering*** is used to get a reasonable partition of the data. Each partition is then used to get initial starting values for the EM algorithm.

(3) The ***Bayes Information Criterion (BIC)*** is used to choose the best grouping and number of clusters, given the data. The BIC is an approximation to Bayes factors.

Finite Mixture Models and the EM Algorithm

Finite Mixture Model

We define the multivariate finite mixture model as the weighted sum of multivariate component densities:

$$f(x, p, \theta) = \sum_{k=1}^{c} p_k g_k(x, \theta_k)$$

The component density is denoted by $g_k(x, \theta_k)$, with parameters represented by θ_k. The weights are given by p_k, with the constraint that they are nonnegative and sum to one. These weights are also called the mixing proportions or mixing coefficients.

The component density can be any bona fide probability density, but one of the most commonly used ones is the multivariate normal. This yields the following equation for a multivariate Gaussian finite mixture

$$f(x, p, \mu, \Sigma) = \sum_{k=1}^{c} p_k \phi(x, \mu_k, \Sigma_k)$$

where ϕ represents a multivariate normal probability density function given by

$$\phi(x_i; \mu_k, \Sigma_k) = \frac{\exp\{-\frac{1}{2}(x - x_i)^T \Sigma_k^{-1}(x_i - \mu_k)\}}{(2\pi)^{d/2}\sqrt{|\Sigma_k|}}$$

μ_k is a d-dimensional vector of means, and Σ_k is a $d \times d$ covariance matrix. To cluster the data we need to estimate the weights p_k, the d-dimension means for each component density and the covariance matrices. We have in total $c \times d$ parameters for the means and $c \times \frac{d(c+1)}{2}$ parameters for the component variance matrices.

EM Algorithm for Estimating Parameters

The problem of estimating the parameters in a finite mixture has been studied extensively in the literature. The book by Everitt and Hand (1981) provides an excellent overview of this topic and offers several methods for parameter estimation. The technique we present here is called the Expectation-Maximization (EM) method. This is a general method for optimizing likelihood functions and is useful in situations where data might be missing or simpler optimization methods fail. The seminal paper on this topic is by Dempster, Laird and Rubin (1977), where they formalize the EM algorithm and establish its properties. Redner and Walker (1984) apply it to mixture densities. The EM methodology is now a standard tool for statisticians and is used in many applications.

In this section, we discuss the EM algorithm as it can be applied to estimating the parameters of a finite mixture of normal densities. To use the EM algorithm, we must have a value for the number of terms c in the mixture. This is usually obtained using prior knowledge of the application (the analyst expects a certain number of groups), using graphical exploratory data analysis(looking for clusters or other group structure) or using some other method of estimating the number of terms. The approach called adaptive mixtures[Priebe, 1994] offers a way to address the problem of determining the number of component densities to use in the finite mixture model. This approach is discussed later.

Besides the number of terms, we must also have an initial guess for the value of the component parameters. Once we have an initial estimate, we update the parameter estimates using the data and the equations given below. These are called the iterative EM update equations, and we provide the multivariate case as the most general one. The univariate case follows easily.

We wish to estimate

$$\theta = p_1, \cdots, p_{c-1}, \mu_1, \cdots \mu_c, \Sigma_1 \cdots, \Sigma_c$$

by maximizing the log-likelihood given by

$$\ln[L(\theta|x_1, \cdots, x_n)] = \sum_{i=1}^{n} \ln[\sum_{k=1}^{c} p_k \phi(x, \mu_k, \Sigma_k)]$$

Note that we don't have to estimate all c of the weight because

$$p_c = 1 - \sum_{i=1}^{c-1} p_i$$

we assume that the components exist in a fixed proportion in the mixture, given by the p_k.
The first step is to determine the posterior probabilities given by

$$\hat{\tau}_{ik}(x_i) = \frac{\hat{p}_k \phi(x_i; \hat{\mu}_k, \hat{\Sigma}_k)}{\hat{f}(x_i; \hat{p}, \hat{\mu}, \hat{\Sigma})}; \qquad k = 1, \cdots, c; \;\; i = 1, \cdots, n,$$

where $\phi(x_i; \hat{\mu}_k, \hat{\Sigma}_k)$ is the multivariate normal density for the *i*-th term evaluated at x_j, and

$$\hat{f}(x_i; \hat{p}, \hat{\mu}, \hat{\Sigma}) = \sum_{j=1}^{c} \hat{p}_j \phi(x_i; \hat{\mu}_j, \hat{\Sigma}_j)$$

is the finite mixture estimate at point x_j.

The posterior probability tells us the likelihood that a point belongs to each of the separate component densities. We can use this estimated posterior probability to obtain a weighted update of the parameters for each component. This yields the iterative EM update equations for the mixing coefficients, the means and the covariance matrices. These are

$$\hat{p}_k = \frac{1}{n} \sum_{i=1}^{n} \hat{\tau}_{ik}$$

$$\hat{\mu}_k = \frac{1}{n} \sum_{i=1}^{n} \frac{\hat{\tau}_{ik} x_i}{\hat{p}_k}$$

$$\hat{\Sigma}_k = \frac{1}{n} \sum_{i=1}^{n} \frac{\hat{\tau}_{ik} (x_i - \hat{\mu}_k)(x_i - \hat{\mu}_k)'}{\hat{p}_k}$$

Note that if d = 1 then the update equation for the variance is

$$\hat{\sigma}_k^2 = \frac{1}{n} \sum_{i=1}^{n} \frac{\hat{\tau}_{ik} (x_i - \hat{\mu}_k)^2}{\hat{p}_k}$$

The EM algorithm is a two step process, consisting of an E-step and an M-step, as outline below. These two steps are repeated until the estimated value converge.

FINITE MIXTURES - EM PROCEDURE

1. Determine the number of terms or component densities c in the mixture.
2. Determine an initial guess at the component parameters. These are the mixing coefficients, means and covariance matrices for each normal density.
3. **(E-Step)** For each data point , calculate the posterior probability using

$$\hat{\tau}_{ik}(x_i) = \frac{\hat{p}_k \phi\left(x_i; \hat{\mu}_k, \hat{\Sigma}_k\right)}{\hat{f}\left(x_i; \hat{p}, \hat{\mu}, \hat{\Sigma}\right)}; \qquad k = 1, \cdots, c; \;\; i = 1, \cdots, n,$$

4. **(M-Step)** Update the parameter estimates using the posterior probability from the E-step and equations

$$\hat{p}_k = \frac{1}{n}\sum_{i=1}^{n} \hat{\tau}_{ik}$$

$$\hat{\mu}_k = \frac{1}{n}\sum_{i=1}^{n} \frac{\hat{\tau}_{ik} x_i}{\hat{p}_k}$$

$$\hat{\sigma}_k^2 = \frac{1}{n}\sum_{i=1}^{n} \frac{\hat{\tau}_{ik}(x_i - \hat{\mu}_k)^2}{\hat{p}_k}$$

5. Repeat steps 3 through 4 until the estimates converge.

Typically, **step** 5 is implemented by continuing the iteration until the changes in the estimates at each iteration are less than some pre-set tolerance. Note that with the iterative EM algorithm, we need to use the entire data set to simultaneously update the parameter estimates. This imposes a high computational load when dealing with massive data sets.

Model-based agglomerative clustering uses the same general ideas as the agglomerate clustering we described earlier. In particular, all observations start out in their own singleton group, and two groups are merged at each step until we have just one group. It also provides a partition of the data for any given number of clusters between one and n. The difference between the two methods has to do with the criterion used to decide which two groups to merge at each step. With regular agglomerate clustering, the two closet groups (in terms of selected distance and type of linkage) are merged. In the case of model-based agglomerate clustering, we do not use a distance. Instead, we use the ***classification likelihood*** as our objective function, which is given by

$$L_{CL}(\theta_k, \gamma_i; x_i) = \prod_{i=1}^{n} f_{\gamma_i}(x_i; \theta_{\gamma_i})$$

where γ_i is a label indicating a classification for the i-th observation. we have $\gamma_i = k$, if x_i belongs to the k-th component. In the mixture approach, the number of observations in each component has a multinomial distribution with the sample size of n and probability parameters given by $p_1, \cdots, p_c$. The exact form of classification likelihood will change depending on the model chosen for our finite mixture.

In the model based agglomerative clustering we seek to find partition that maximize (approximately) the classification likelihood. To accomplish this, the two clusters producing the largest increase in the classification likelihood are merged at each step. This process continues until all observations are in one group.

Bayesian Information Criterion

We now turn our attention to the third major component in the model-based clustering, which is choosing the model that best fits our data according to some criterion.

The Bayesian Information Criterion is given by

$$BIC = 2\log\left[L_M\left(X, \hat{\theta}\right)\right] - mlog(n),$$

where L_M is the likelihood given that data, the model M, and the estimated parameters $\hat{\theta}$. The number of independent parameters to be estimated for model *M* is given by *m*.