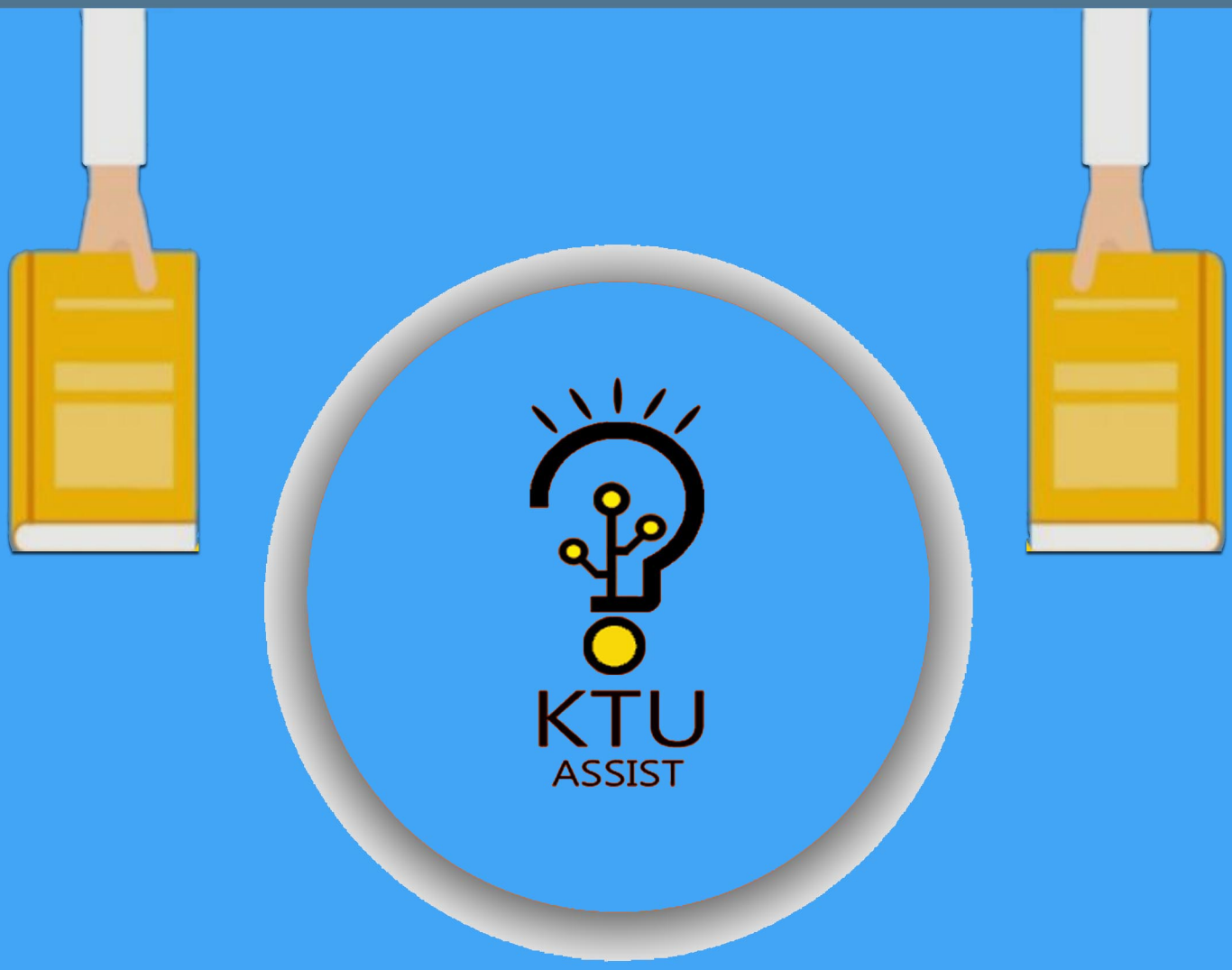
APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY

STUDY MATERIALS



Module 3

Classification Models: Introduction to Classification and Prediction, Issues regarding classification and prediction, Decision Tree- ID3, C4.5, Naive Bayes Classifier.

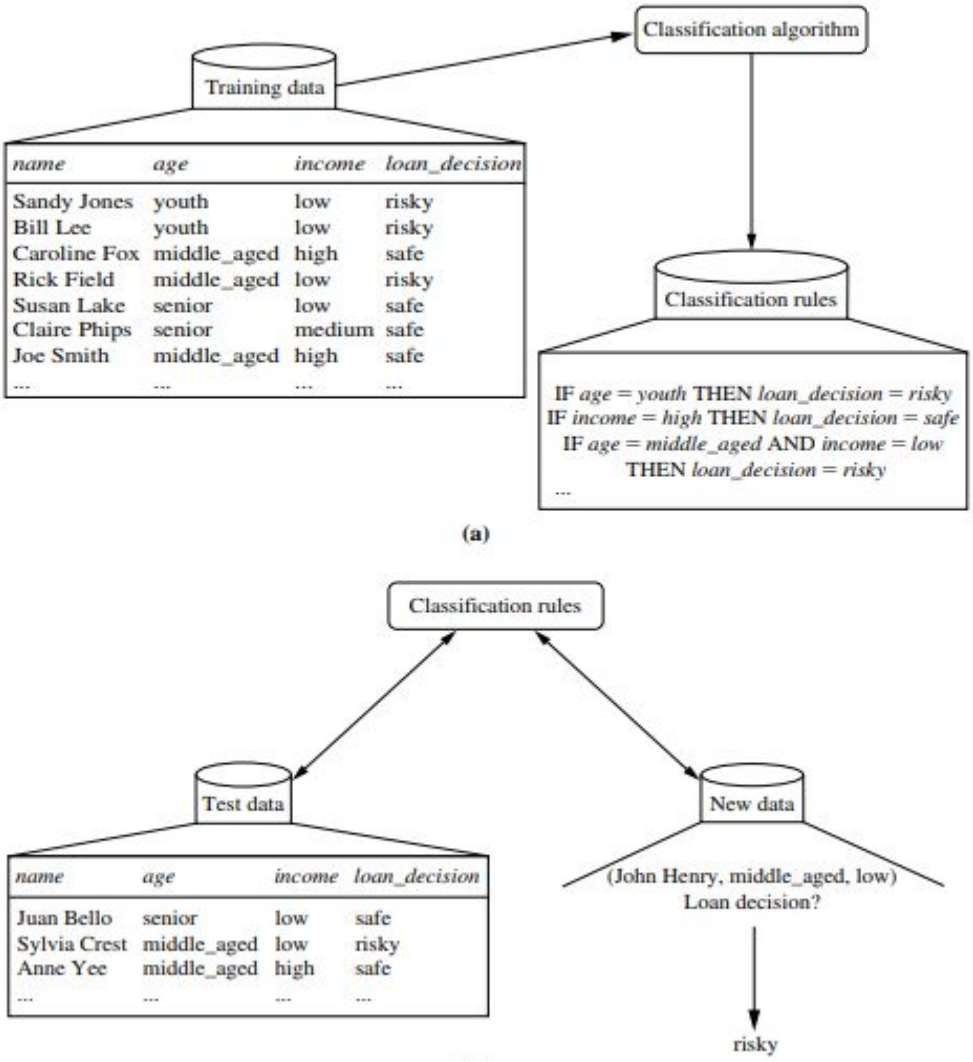
Introduction to Classification and Prediction, Issues regarding classification and prediction,

Classification and prediction are two forms of data analysis that extracts models describing important data classes or to predict future data trends. Such models, called classifiers, predict categorical (discrete, unordered) class labels. Whereas classification predicts categorical labels, prediction models continuos-valued functions. For example, we can build a classification model to categorize bank loan applications as either safe or risky, while a prediction model may be built to predict the expenditures of potential customers on computer equipment given thier income and occupation . Such analysis can help provide us with a better understanding of the data at large. Many classification and prediction methods have been proposed by researchers in machine learning, pattern recognition, and statistics. Most algorithms are memory resident, typically assuming a small data size. Recent data mining research has built on such work, developing scalable classification and prediction techniques capable of handling large amounts of disk-resident data. Classification has numerous applications, including fraud detection, target marketing, performance prediction, manufacturing, and medical diagnosis.

What Is Classification? What Is Prediction ?

Data classification is a two-step process, consisting of a learning step (where a classification model is constructed) and a classification step (where the model is used to predict class labels for given data). The process is shown for the loan application data of Figure .In the first step, a classifier is built describing a predetermined set of data classes or concepts. This is the learning step (or training phase), where a classification algorithm builds the classifier by analyzing or "learning from" a training set made up of database tuples and their associated class labels. A tuple, X, is represented by an n-dimensional attribute vector, X = (x1, x2,..., xn), depicting n measurements made on the tuple from n database attributes, respectively, A1, A2,..., An. 1 Each tuple, X, is assumed to belong to a predefined class as determined by another database attribute called the class label attribute. The class label attribute is discrete-valued and unordered. It is categorical (or nominal) in that each value serves as a category or class. The individual tuples making up the training set are referred to as training tuples and are randomly sampled from the database under analysis. In the context of classification, data tuples can be referred to as samples, examples, instances, data points, or objects.

Because the class label of each training tuple is provided, this step is also known as supervised learning (i.e., the learning of the classifier is "supervised" in that it is told to which class each training tuple belongs). It contrasts with unsupervised learning (or clustering), in which the class label of each training tuple is not known, and the number or set of classes to be learned may not be known in advance. For example, if we did not have the loan decision data available for the training set, we could use clustering to try to determine "groups of like tuples," which may correspond to risk groups within the loan application data.



The data classification process: (a) *Learning*: Training data are analyzed by a classification algorithm. Here, the class label attribute is *loan_decision*, and the learned model or classifier is represented in the form of classification rules. (b) *Classification*: Test data are used to estimate the accuracy of the classification rules. If the accuracy is considered acceptable, the rules can be applied to the classification of new data tuples.

This first step of the classification process can also be viewed as the learning of a mapping or function, y = f (X), that can predict the associated class label y of a given tuple X. In this view, we wish to learn a mapping or function that separates the data classes. Typically, this mapping is represented in the form of classification rules, decision trees, or mathematical formulae. In our example, the mapping is represented as classification rules that identify loan applications as being either safe or risky (Figure a).

The rules can be used to categorize future data tuples, as well as provide deeper insight into the data contents. They also provide a compressed data representation.

"What about classification accuracy?" In the second step (Figure b), the model is used for classification. First, the predictive accuracy of the classifier is estimated. If we were to use the training set to measure the classifier's accuracy, this estimate would likely be optimistic, because the classifier tends to overfit the data (i.e., during learning it may incorporate some particular anomalies of the training data that are not present in the general data set overall). Therefore, a test set is used, made up of test tuples and their associated class labels. They are independent of the training tuples, meaning that they were not used to construct the classifier. The accuracy of a classifier on a given test set is the percentage of test set tuples that are correctly classified by the classifier. The associated class label of each test tuple is compared with the learned classifier's class prediction for that tuple. Section 8.5 describes several methods for estimating classifier accuracy. If the accuracy of the classifier is considered acceptable, the classifier can be used to classify future data tuples for which the class label is not known. (Such data are also referred to in the machine learning literature as "unknown" or "previously unseen" data.) For example, the classification rules learned in Figure (a) from the analysis of data from previous loan applications can be used to approve or reject new or future loan applicants.

What Is Prediction ?

Prediction can be viewed as the construction and the use of a model to access the class of an unlabeled sample or to access the value or value ranges of an attribute that a given sample is likely to have. In this view , classification and regression are two major types of prediction problems, where classification is used to predict discrete or nominal values while regression is used to predict continuos or ordered values. But we refer to the use of prediction to predict the class labels as classification and the use of prediction to predict continuos values (using regression technques) as prediction. This is the most accepted view in data mining.

Classification and prediction have numerous applications including credit applications, medical diagnosis, preformance prediction and selective marketing.

Issues regarding classification and prediction

1) **Preparing data for classification and prediction**

The following preprocessing steps may be applied to the data in order to help improve the accuracy,efficiency and scalability of the classification or prediction process.

- Data Cleaning: This refers to the preprocessing of data in order to remove or reduce noise(by applying smoothing techniques for ex) and the treatment of missing values(e.g by replacing a missing value with the most commonly occuring value for that attribute, or with most probable value based on statistics).
- Relevance Analysis: Many of the attributes in the data may be irrelevant to classification or prediction task. Relevance analysis may be performed on data with the aim of removing any

irrelevant or redundant attributes from the learning process. In machine learning, this step is known as feature selection.

The time spent on relevance analysis, when time spent on learning from the resulting "reduced" feature subset, should be less than the time that would spent on learning from the original set of features.

- Data Transformation: The data can be generalized to higher-level concepts.Concept hierarchies may be used for this purpose. This is particularly useful for continuos-valued attributes. For ex. numeric values for attribute income may be generalized to discrete ranges as low, medium and high. Since generalization compress the original training data, fewer input/output operations may be involved during learning.

The data may also be normalized, particualrly for neural networks or methods involving distance measurements are used in learning step. Normalization involves scaling all values for a given attribute so that they fall with in a small specified range such as -1.0 to 1.0 or 0.0 to 1.0.

2) Comparing Classification Methods

Classification and prediction methods can be compared and evaluated according to following criteria.

- **Predictive accuracy**: This refersw to ability of the model to correctly predict the class label of new or previously unseen data.
- **Speed**: This refers to the computation costs involved in generating and using the model.
- **Robustness:** This is the ability of the model to amke correct predictions given noisy data or data with the missing values.
- **Scalability:** This refers to the ability to construct the model efficiently given large amounts of data.
- **Interpretability**: This refers to the level of understanding and insight that is provided by the model.

3) Issues in Classification

- **Missing Data**. Missing data values cause problems during both the training phase and the classification process itself. Missing values in the training data must be handled and may produce an inaccurate result. Missing data in a tuple to be classified must be able to be handled by the resulting classification scheme. There are many approaches to handling missing data:
  - Ignore the missing data.
  - Assume a value for the missing data. This may be determined by using some method to predict what the value could be.
  - Assume a special value for the missing data. This means that the value of missing data is taken to be a specific value all of its own.

Notice: the similarity between missing data in the classification problem and that of nulls in traditional databases.

- **Measuring Performance**. Table 4. 1 shows two different classification results using . two different classification tools. Determining which is best depends on the interpretation of the problem by users. The performance of classification algorithms is usually examined by evaluatmg the accuracy

TABLE 4.1: Data for Height Classification

Name	Gender	Height	Output1	Output2
Kristina	F	1.6 m	Short	Medium
Jim	M	2 m	Tall	Medium
Maggie	F	1.9 m	Medium	Tall
Martha	F	1.88 m	Medium	Tall
Stephanie	F	1.7 m	Short	Medium
Bob	M	1.85 m	Medium	Medium
Kathy	F	1.6 m	Short	Medium
Dave	M	1.7 m	Short	Medium
Worth	M	2.2 m	Tall	Tall
Steven	M	2.1 m	Tall	Tall
Debbie	F	1.8 m	Medium	Medium
Todd	M	1.95 m	Medium	Medium
Kim	F	1.9 m	Medium	Tall
Amy	F	1.8 m	Medium	Medium
Wynette	F	1.75 m	Medium	Medium

of the classification. However, since classification isoften a fuzzy problem, the correct answer may depend on the user. Traditional algorithm evaluation approaches such as determining the space and time overhead can be used, but these approaches are usually secondary.

Classification accuracy is usually calculated by determining the percentage of tuples placed in the correct class. This ignores the fact that there also may be a cost associated with an incorrect assignment to the wrong class. This perhaps should also be determined.

We can examine the performance of classification much as is done with information retrieval systems. With only two classes, there are four possible outcomes with the classification, as is shown in Figure 4.3. The upper left and lower right quadrants [for both Figure 4.3(a) and (b)] represent correct actions. The remaining two quadrants are incorrect actions. The performance of a classification could be determined by associating costs with each of the quadrants. However, this would be difficult because the total number of costs needed is m^2, where m is the number of classes.

Given a specific class, C_j, and a database tuple t_i, that tuple may or may not be assigned to that class while its actual membership may or may not be in that class. This again gives us the four quadrants shown in Figure 4.3(c), which can be described in the following ways:

- True positive (TP): t_i predicted to be in C_j and is actually in it.
- False positive (FP): t_i predicted to be in C_j but is not actually in it.
- True negative (TN): t_i not predicted to be in C_j and is not actually in it.
- False negative (FN): t_i not predicted to be in C_j but is actually in it.

An OC (operating characteristic) curve or ROC (receiver operating characteristic) curve or ROC (relative operating characteristic) curve shows the relationship between false positives and

RET REL	NOTRET REL
RET NOTREL	NOTRET NOTREL

(a) Information retrieval

Assigned Class A in Class A	Assigned Class B in Class A
Assigned Class A in Class B	Assigned Class B in Class B

(b) Classification into Class A

True positive	False negative
False positive	True negative

(c) Class prediction

FIGURE 4.3: Comparing classification performance to information retrieval.

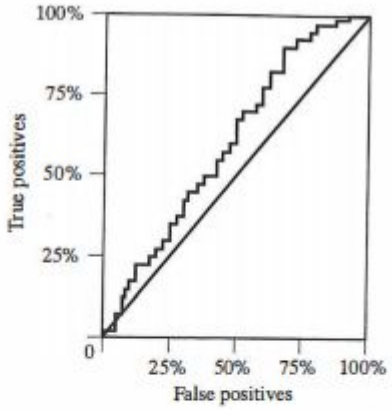


FIGURE 4.4: Operating characteristic curve.

TABLE 4.2: Confusion Matrix

Actual Membership	Assignment		
	Short	Medium	Tall
Short	0	4	0
Medium	0	5	3
Tall	0	1	2

true positives. An OC curve was originally used in the communications area to examine false alarm rates. It has also been used in information retrieval to examine fallout (percentage of retrieved that are not relevant) versus recall (percentage of retrieved that are relevant) . In the OC curve the horizontal axis has the percentage of false positives and the vertical axis has the percentage of true positives for a database sample. At thebeginning of evaluating a sample, there are none of either category, while at the end there are 100 percent of each. When evaluating the results for a specific sample, the curve looks like a jagged stair-step, as seen in Figure 4.4, as each new tuple is either a false positive or a true positive. A more smoothed version of the OC curve can also be obtained.

A confusion matrix illustrates the accuracy of the solution to a classification problem. Given m classes, a confusion matrix is an m x m matrix where entry $c_{i,j}$ indicates the number of tuples from D that were assigned to class Cj but where the correct class is Ci . Obviously, the best solutions will have only zero values outside the diagonal. Table 4.2 shows a confusion matrix for the height example in Table 4.1 where the Outputl assignment is assumed to be correct and the Output2 assignment is what is actually made.

DECISION TREE-BASED ALGORITHMS

The decision tree approach is most useful in classification problems. With this technique, a tree is constructed to model the classification process. Once the tree is built, it is applied to each tuple in the database and results in a classification for that tuple. There are two basic steps in the technique: building the tree and applying the tree to the database. Most research has focused on how to build effective trees as the application process is straightforward.

The decision tree approach to classification is to divide the search space into rectangular regions. A tuple is classified based on the region into which it falls. A definition for a decision tree used in classification is contained in Definition 4.3. There are alternative definitions; for example, in a binary DT the nodes could be labeled with the predicates themselves and each arc would be labeled with yes or no.

DEFINITION 4.3. Given a database D = {t1 , ... , tn } where ti = (t_{i1} , . .. , t_{ih}} and the database schema contains the following attributes {A1 , A2, ... , Ah }. Also given is a set of classes C = { C 1 , ... , C m}. A decision tree(DT) or classification tree is a tree associated with D that has the following properties:

• Each internal node is labeled with an attribute, Ai.

• Each arc is labeled with a predicate that can be applied to the attribute associated with the parent.

• Each leaf node is labeled with a class, C j.

Solving the classification problem using decision trees is a two-step process:

1. Decision tree induction: Construct a DT using training data.

2. For each ti E D, apply the DT to determine its class.

Based on our definition of the classification problem, Definition 4. 1, the constructed DT represents the logic needed to perform the mapping. Thus, it implicitly defines the mapping. Using the DT shown in Figure 3.5 from Chapter 3, the classification of the · sample data found in Table 4. 1 is that shown in the column labeled Output2. A different DT could yield a different classification. Since the application of a given tuple to a DT is relatively straightforward, we do not consider the second part of the problem further. Instead, we focus on algorithms to construct decision trees. Several algorithms are surveyed in the following subsections.

There are many advantages to the use of DTs for classification. DTs certainly are easy to use and efficient. Rules can be generated that are easy to interpret and understand. They scale well for large databases because the tree size is independent of the database size. Each tuple in the database must be filtered through the tree. This takes time proportional to the height of the tree, which is fixed. Trees can be constructed for data with many attributes.

Disadvantages also exist for DT algorithms. First, they do not easily handle continuous data. These attribute domains must be divided into categories to be handled. The approach used is that the domain space is divided into rectangular regions [such as is seen in Figure 4.l (a)]. Not all classification problems are of this type. The division shown by the simple loan classification problem in Figure 2.4(a) in Chapter 2 cannot be handled by DTs. Handling missing data is difficult because correct branches in the tree could not be taken. Since the DT is constructed from the training data, overfitting may occur. This can be overcome via tree pruning. Finally, correlations among attributes in the database are ignored by the DT process.

ALGORITHM 4.3

Input :

D / /Training data

Output :

T / /Decision tree

DTBuild algorithm:

//Simplistic a lgorithm to illustrate naive approach to building DT

T = Ø;

Determine best splitting criterion ;

T = Create root node node and label with splitting attribute;

T = Add arc to root node for each split predicate and label ;

for each arc do

D = Database created by applying splitting predicate to D;

if stopping point reached for this path, then

$T^{/}$ = Create leaf node and l abel with appropriate class ;

else

$T^{/}$ = DTBuild(D) ;

T= Add T/ to arc ;

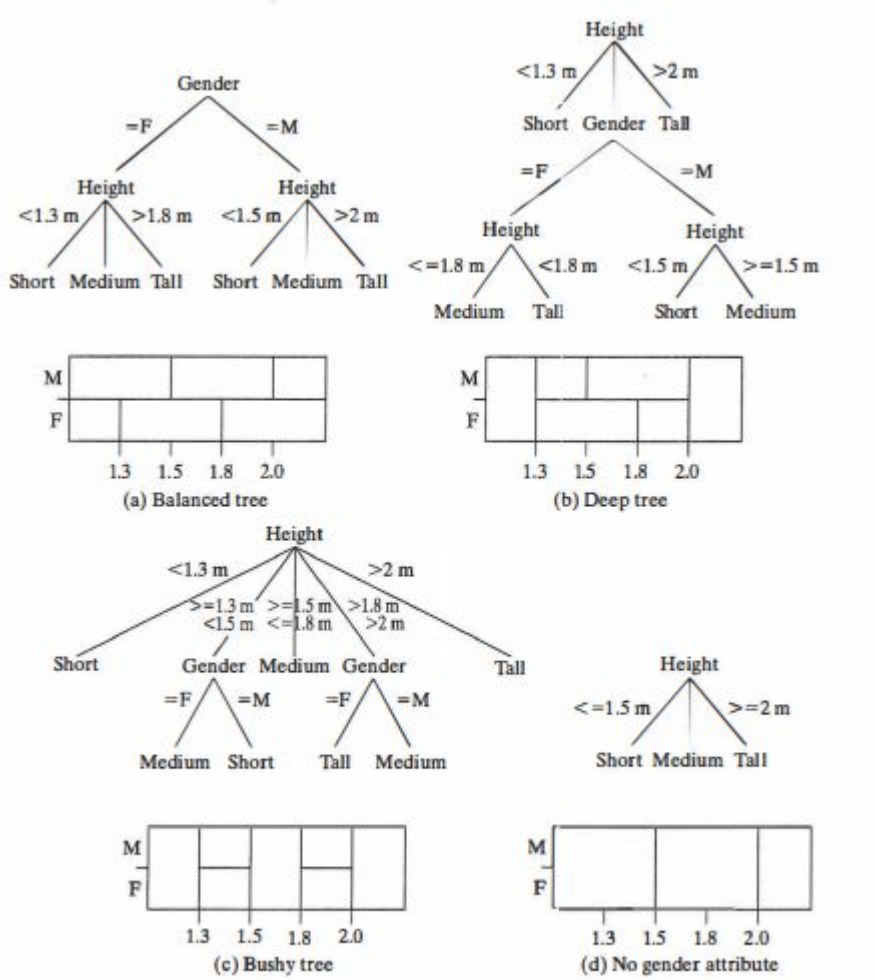


FIGURE 4.11: Comparing decision trees.

There have been many decision tree algorithms . We illustrate the tree-building phase in the simplistic DTBuild Algorithm 4.3. Attributes in the database schema that will be used to label nodes in the tree and around which the divisions will take place are called the splitting attributes. The predicates by which the arcs in the tree are labeled are called the splitting predicates. In the decision trees shown in Figure 4.11, the splitting attributes are {gender, height}. The splitting predicates for gender are {= female, = male}, while those for height include {< 1 .3 m, >1.8 m, 2 m}. The splitting predicates for height differ based on whether the tuple is for a male or a female. This recursive algorithm builds the tree in a top-down fashion by examining the training data. Using the initial training data, the "best" splitting attribute is chosen first. Algorithms differ in how they determine the "best attribute" and its "best predicates" to use for splitting. Once this has been determined, the node and its arcs are created and added to the created tree. The algorithm continues recursively by adding new subtrees to each branching arc. The algorithm terminates when some "stopping criteria" is reached. Again, each algorithm determines when to stop the tree differently. One simple approach would be to stop when

the tuples in the reduced training set all belong to the same class. This class is then used to label the leaf node created.

Note that the major factors in the performance of the DT building algorithm are the size of the training set and how the best splitting attribute is chosen. The following issues are faced by most DT algorithms:

• Choosing splitting attributes: Which attributes to use for splitting attributes impacts the performance applying the built DT. Some attributes are better than others. In the data shown in Table 4.1, the name attribute definitely should not be used and the gender may or may not be used. The choice of attribute involves not only an examination of the data in the training set but also the informed input of domain experts.

• Ordering of splitting attributes: The order in which the attributes are chosen is also important. In Figure 4.1 l(a) the gender attribute is chosen first. Alternatively, the height attribute could be chosen first. As seen in Figure 4.l l(b), in this case the height attribute must be examined a second time, requiring unnecessary comparisons.

• Splits: Associated with the ordering of the attributes is the number of splits to take. With some attributes, the domain is small, so the number of splits is obvious based on the domain (as with the gender attribute). However, if the domain is continuous or has a large number of values, the number of splits to use is not easily determined.

• Tree structure: To improve the performance of applying the tree for classification, a balanced tree with the fewest levels is desirable. However, in this case, more complicated comparisons with multiway branching [see Figure 4.11(c)] may be needed. Some algorithms build only binary trees.

• Stopping criteria: The creation of the tree definitely stops when the training data are perfectly classified. There may be situations when stopping earlier would be desirable to prevent the creation of larger trees. This is a trade-off between accuracy of classification and performance. In addition, stopping earlier may be performed to prevent overfitting. It is even conceivable that more levels than needed would be created in a tree if it is known that there are data distributions not represented in the training data.

• Training data: The structure of the DT created depends on the training data. If the training data set is too small, then the generated tree might not be specific enough to work properly with the more general data. If the training data set is too large, then the created tree may overfit.

• Pruning: Once a tree is constructed, some modifications to the tree might be needed to improve the performance of the tree during the classification phase. The pruning phase might remove redundant comparisons or remove subtrees to achieve better performance.

To illustrate some of these design decisions, Figure 4. 11 shows four different decision trees that can be used to classify persons according to height. The first tree is a duplicate of that from Chapter 3. The first three trees of this figure all perform the same classification. However, they all perform it

differently. Underneath each tree is a table showing the logical divisions used by the associated tree for classification. A nice feature of Figure 4.11(a) is that it is balanced. The tree is of the same depth for any path from root to leaf. Figures 4.11(b) and (c), however, are not balanced. In addition, the height of the tree in (b) is greater than that of any of the others, implying a slightly worse behavior when used for classification. However, all of these factors impact the time required to do the actual classification. These may not be crucial performance issues unless the database is extremely large. In that case a balanced shorter tree would be desirable. The tree shown in Figure 4.l l(d) does not represent the same classification logic as the others.

The training data and the tree induction algorithm determine the tree shape. Thus, the best-shaped tree that performs perfectly on the training set is desirable. Some algorithms create only binary trees. Binary trees are easily created, but they tend to be deeper. The performance results when applying these types of trees for classification may be worse because more comparisons usually are needed. However, since these comparisons are simpler than those that require multiway branches, the ultimate performance may be comparable.

The DT building algorithms may initially build the tree and then prune it for more effective classification. With pruning techniques, portions of the tree may be removed or combined to reduce the overall size of the tree. Portions of the tree relating to classification using an unimportant attribute may be removed. This sort of change with a node close to the root could ripple down to create major changes in the lower parts of the tree. For example, with the data in Figure 4.1, if a tree were constructed by looking at values of the name attribute, all nodes labeled with that attribute would be removed.

Lower-level nodes would move up or be combined in some way. The approach to doing this could become quite complicated. In the case of overfitting, lower-level subtrees may be removed completely. Prumng may be performed while the tree is being created, thus preventing a tree from becoming too large. A second approach prunes the tree after it Is built.

The time and space complexity of DT algorithms depends on the size of the training data, q; the number of attributes, h; and the shape of the resulting tree. In the worst case the DT that is built may be quite deep and not bushy. As the tree is built, for each of these nodes, eac h attribute will be examined to determine if it is the best. This gives a time complexity to build the tree of O (h q log q). The time to classify a database of size n is based on the height of the tree. Assuming a height ofO (log q), this is then O(n log q).

In the following subsections we examine several popular DT approaches.

ID3

The **ID3** technique to building a decision tree is based on information theory and attempts to minimize the expected number of comparisons. The bask idea of the induction algonthm is to ask questions whose answers provide the most information. This is similar to the intuitive approach taken by adults when playing the "Twenty Questions" game. The first question an adult might ask could be "Is

the thing alive?" while a child might ask "Is it my Daddy?" The first question divides the search space into two large search domains, while the second performs little division of the space. The basic strategy used by ID3 is to choose splitting attributes with the highest information gain first. The amount of information associated with an attribute value is related to the probability of occurrence. Looking at the "Twenty Questions" example, the child's question divides the search space mto two sets. One set (Daddy) has an infinitesimal probability associated with it and the other set is almost certain, while the question the adult makes divides the search space into two subsets with almost equal probability of occurring.

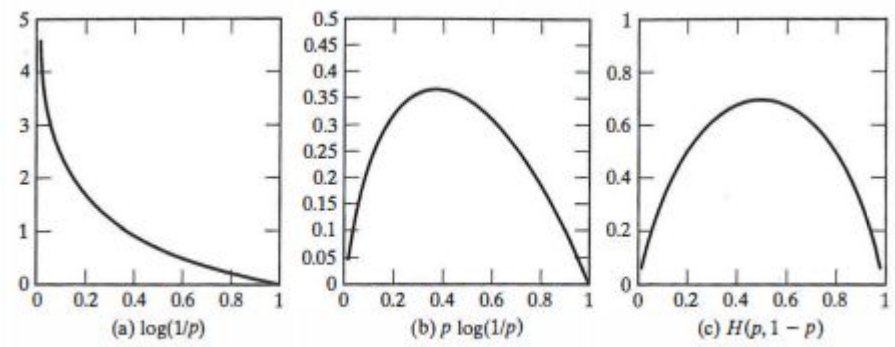


FIGURE 4.12: Entropy.

The concept used to quantify information is called entropy. Entropy is used to measure the amount of uncertainty or surprise or randomness in a set of data. Certainly, when all data in a set belong to a single class, there is no uncertainty. In this case the entropy is zero. The objective of decision tree classification is to iteratively partition the given data set into subsets where all elements in each final subset belong to the same class. In Figure 4. 12(a, b, and c) will help to explain the concept. Figure 4.12(a) shows log(l/ p) as the probability p ranges from 0 to 1. This intuitively shows the amount of surprise based on the probability. When p = J., there is no surprise. This means that if an event has a probability of 1 and you are told that the event occurred, you would not be surprised.

As p $\rightarrow$ 0, the surprise increases. When we deal with a divide and conquer approach such as that used with decision trees, the division results in multiple probabilities whose sum is 1. In the "Twenty Questions" game, the P (Daddy) < P(-Daddy) and P (Daddy) + P (-Daddy) = 1. To measure the information associated with this division, we must be able to combine the inforrnation associated with both events. That is, we must be able to calculate the average information associated with the division. This can be performed by adding the two values together and taking into account the probability that each occurs

. Figure 4.12(b) shows the function p X log(1/p), which is the expected information based on probability of an event. To determine the expected informatiori associated with two events, we add the individual values together. This function p log(l/ p) + (1 - p) log(1/(1 - p)) is plotted in Figure 4.1 2(c). Note that the maximum occurs when the two probabilities are equal. This supports our intuitive idea that the more sophisticated questions posed by the adult are better than those posed by the child.

The formal definition of entropy is shown in Definition 4.4. The value for entropy is between 0 and 1 and reaches a maximum when the probabilities are all the same.

DEFINITION 4.4. Given probabilities PI , P2, ... , Ps where $\sum_{i=1}^{s} p\ i = 1$, entropy is defined as

H(p1 , P2, ... , Ps) = $\sum_{i=1}^{s} \text{pi} \log(1/\text{pi})$ (4.16)

Given a database state, D, H (D) finds the amount of order (or lack thereof) in that state. When that state is split into s new states S = {D1 , D2 , ... , Ds}, we can again look at the entropy of those states. Each step in ID3 chooses the state that orders splitting the most.

A database state is completely ordered if all tuples in it are in the same class. ID3 chooses the splitting attribute with the highest gain in information, where gain is defined as the difference between how much information is needed to make a correct classification before the split versus how much information is needed after the split. Certainly, the split should reduce the information needed by the largest amount. This is calculated by determining the differences between the entropies of the original dataset and the weighted sum of the entropies from each of the subdivided datasets. The entropies of the split datasets are weighted by the fraction of the dataset being placed in that division.

The ID3 algorithm calculates the gain of a particular split by the followmg formula:

Gain(D, S) = H(D) - $\sum_{i=1}^{s} \text{P(Di)H(Di)}$ (4.17)

Example 4.7 and associated Figure 4.13 illustrate this process using the height example. In this example, six divisions of the possible ranges of heights are used.

This division into ranges is needed when the domain of an attribute is continuous or (as in this case) consists of many possible values. While the choice of these divisions is somewhat arbitrary, a domain expert shoulQ be able to perform the task.

EXAMPLE 4.7 The beginning state of the training data in Table 4.1 (with the Output1 classification) is that (4/ 1 5) are short, (8/1 5) are medium, and (3/15) are tall.

Thus, the entropy of the starting set is 4/ 15 log(15/4) + 8/15 log(15/8) + 3/15 log(15/3) = 0.4384

Choosing the gender as the splitting attribute, there are nine tuples that are F and six that are M. The entropy of the subset that are F is 3/9 log(9/3) + 6/9 log(9/6) = 0.2764 (4.18)

whereas that for the M subset is 1/6log(6/1) + 2/6 log(6/2) + 3/6 log(6/3) = 0.4392 (4.19)

The ID3 algorithm must determine what the gain in information is by using this split. To do this, we calculate the weighted sum of these iast two entropies to get

((9/ 15) 0.2764) + ((6/1 5) 0.4392) = 0.34152 (4.20)

The gain in entropy by using the gender attribute is thus 0.4384 - 0.34152 = 0.09688 (4.21) Looking at the height attribute, we have two tuples that are 1 .6, two are 1 .7, one is 1 .75, two are 1 .8, one is 1 .85, one is 1 .88, two are 1 .9, one is 1 .95, one is 2, one is 2.1, and one is 2.2.

Determining the split values for height is not easy. Even though the training dataset has these 11 values, we know that there will be many more. Just as with continuous data, we divide into ranges

: (0, 1 .6], (1 .6, 1 .7], (1.7, 1 .8], (1.8, 1 .9], (1.9, 2.0], (2.0,∞)

There are 2 tuples in the first division with entropy (2/2(0) + 0 + 0) = 0, 2 in (1.6, 1 .7] with entropy (2/2(0) +0+0) = 0, 3 in (1.7, 1.8] with entropy (0 + 3/3(0) + 0) = 0, 4 in (1.8, 1.9] with entropy (0+4/4(0) +0) = 0, 2 in (1.9, 2.0] with entropy (0+ 1/2(0.301)+ 1 /2(0.301)) = 0.301, and two in the last with entropy (0 + 0 + 2/2(0)) = 0. All of these states are completely ordered and thus an entropy of 0 except for the (1.9, 2.0] state.

The gain in entropy by using the height attribute is thus 0.4384 - 2/15(0.301) = 0.3983 (4.22)

Thus, this has the greater gain, and we choose this over gender as the first splitting attribute. Within this division there are two males, one medium and one tall. This has occurred because this grouping was too large. A further subdivision on height is needed, and this generates the DT seen in Figure 4.13(a).

Figure 4.13(a) illustrates a problem in that the tree has multiple splits with identical results. In addition, there is a subdivision of range (1 .9, 2.0]. Figure 4.13(b) shows an optimized version of the tree.

C4.5

The decision tree algorithm C4.5 improves ID3 in the following ways:

• Missing data: When the decision tree is built, missing data are simply ignored. That is, the gain ratio is calculated by looking only at the other records that have a value for that attribute. To classify a record with a missing attribute value, the value for thai item can be predicted based on what is known about the attribute values for the other records.

• Continuous data: The basic idea is to divide the data into ranges based on the attribute values for that item that are found in the training sample.

- Pruning: There are two primary pruning strategies proposed in C4.5
  : - With subtree replacement, a subtree is replaced by a leaf node if this replacement results in an error rate close to that of the original tree. Subtree replacement works from the bottom of the tree up to the root.
  - Another pruning strategy, called subtree raising, replaces a subtree by its most used subtree. Here a subtree is raised from its current location to a node higher up in the tree. Again, we must determine the increase in error rate for this replacement.

• Rules: C4.5 allows classification via either decision trees or rules generated from them. In addition, some techniques to simplify complex rules are proposed. One approach is to replace the left-hand side of a rule by a simpler version if all records iri the training set are treated identically. An "otherwise" type of rule can be used to indicate what should be done if no other rules apply.

• Splitting: The ID3 approach favors attributes with many divisions and thus may lead to overfitting. In the extreme, an attribute that has a unique value for each tuple in the training set would be the best because there would be only one tuple (and thus one class) for each division. An improvement can be made by taking into account the cardinality of each division. This approach uses the GainRatio as opposed to Gain. The GainRatio is defined as

$$\text{GainRatio}(D, S) = \frac{\text{Gain}(D, S)}{H\left(\frac{|D_1|}{|D|}, \ldots, \frac{|D_s|}{|D|}\right)} \tag{4.23}$$

For splitting purposes, C4.5 uses the largest GainRatio that ensures a larger than average information gain. This is to compensate for the fact that the GainRatio value is skewed toward splits where the size of one subset is close to that of the starting one. Example 4.8 shows the calculation of GainRatio for the first split in Example 4.7.

EXAMPLE 4.8

To calculate the GainRatio for the gender split, we first find the entropy associated with the split ignoring classes

$$H\left(\frac{9}{15}, \frac{6}{15}\right) = \frac{9}{15}\log\left(\frac{15}{9}\right) + \frac{6}{15}\log\left(\frac{15}{6}\right) = 0.292 \tag{4.24}$$

This gives the GainRatio value for the gender attribute as

$$\frac{0.09688}{0.292} = 0.332 \tag{4.25}$$

The entropy for the split on height (ignoring classes) is

$$H\left(\frac{2}{15}, \frac{2}{15}, \frac{3}{15}, \frac{4}{15}, \frac{2}{15}\right) \tag{4.26}$$

Naıve Bayesian Classifier

Bayes' Theorem

Bayes' theorem is named after Thomas Bayes, a nonconformist English clergyman who did early work in probability and decision theory during the 18th century. Let X be a data tuple. In Bayesian terms, X is considered "evidence." As usual, it is described by measurements made on a set of n attributes. Let H be some hypothesis such as that the data tuple X belongs to a specified class C. For classification problems, we want to determine P(H|X), the probability that the hypothesis H holds given the "evidence" or observed data tuple X. In other words, we are looking for the probability that tuple X belongs to class C, given that we know the attribute description of X.

P(H|X) is the posterior probability, or a posteriori probability, of H conditioned on X. For example, suppose our world of data tuples is confined to customers described by the attributes age and

income, respectively, and that X is a 35-year-old customer with an income of $40,000. Suppose that H is the hypothesis that our customer will buy a computer. Then P(H|X) reflects the probability that customer X will buy a computer given that we know the customer's age and income.

In contrast, P(H) is the prior probability, or a priori probability, of H. For our example, this is the probability that any given customer will buy a computer, regardless of age, income, or any other information, for that matter. The posterior probability, P(H|X), is based on more information (e.g., customer information) than the prior probability, P(H), which is independent of X.

Similarly, P(X|H) is the posterior probability of X conditioned on H. That is, it is the probability that a customer, X, is 35 years old and earns $40,000, given that we know the customer will buy a computer.

P(X) is the prior probability of X. Using our example, it is the probability that a person from our set of customers is 35 years old and earns $40,000.

"How are these probabilities estimated?" P(H), P(X|H), and P(X) may be estimated from the given data, as we shall see next. Bayes' theorem is useful in that it provides a way of calculating the posterior probability, P(H|X), from P(H), P(X|H), and P(X).

Bayes' theorem is

$$P(H|X) = \frac{P(X|H)P(H)}{P(X)}. \qquad (8.10)$$

Now that we have that out of the way, in the next section, we will look at how Bayes' theorem is used in the naıve Bayesian classifier.

Naıve Bayesian Classification

The naıve Bayesian classifier, or simple Bayesian classifier, works as follows:

1. Let D be a training set of tuples and their associated class labels. As usual, each tuple is represented by an n-dimensional attribute vector, X = (x1, x2,..., xn), depicting n measurements made on the tuple from n attributes, respectively, A1, A2,..., An.

2. Suppose that there are m classes, C1, C2,..., Cm. Given a tuple, X, the classifier will predict that X belongs to the class having the highest posterior probability, conditioned on X. That is, the naıve Bayesian classifier predicts that tuple X belongs to the class Ci if and only if

P(Ci/ X) > P(Cj/ X) for $1 \le j \le m, j \ne i$. Thus, we maximize P(Ci /X). The class Ci for which P(Ci|X) is maximized is called the maximum posteriori hypothesis. By Bayes' theorem (Eq. 8.10),

$$P(Ci/X) = \frac{P(X|Ci)P(Ci)}{P(X)}) .$$

3. As P(X) is constant for all classes, only P(X|Ci)P(Ci) needs to be maximized. If the class prior probabilities are not known, then it is commonly assumed that the classes are equally likely, that is, P(C1) = P(C2) = ⋯ = P(Cm), and we would therefore maximize P(X|Ci). Otherwise, we maximize P(X|Ci)P(Ci). Note that the class prior probabilities may be estimated by P(Ci) = |C i,D|/|D|, where |C i,D| is the number of training tuples of class Ci in D.

4. Given data sets with many attributes, it would be extremely computationally expensive to compute P(X|Ci). To reduce computation in evaluating P(X|Ci), the naıve assumption of class-conditional independence is made. This presumes that the attributes' values are conditionally independent of one another, given the class label of the tuple (i.e., that there are no dependence relationships among the attributes).

Thus, $P(X|Ci) = \prod_{k=1}^{n} P(xk|Ci)$ (8.12)

= P(x1|Ci) × P(x2|Ci) × ⋯ × P(xn|Ci).

We can easily estimate the probabilities P(x1|Ci), P(x2|Ci),..., P(xn|Ci) from the training tuples. Recall that here xk refers to the value of attribute Ak for tuple X.

For each attribute, we look at whether the attribute is categorical or continuous-valued. For instance, to compute P(X|Ci), we consider the following:

(a) If Ak is categorical, then P(xk |Ci) is the number of tuples of class Ci in D having the value xk for Ak , divided by |Ci,D|, the number of tuples of class Ci in D.

(b) If Ak is continuous-valued, then we need to do a bit more work, but the calculation is pretty straightforward. A continuous-valued attribute is typically assumed to have a Gaussian distribution with a mean μ and standard deviation σ, defined by

$$g(x, \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, \qquad (8.13)$$

so that

$$P(x_k|C_i) = g(x_k, \mu_{C_i}, \sigma_{C_i}). \qquad (8.14)$$

These equations may appear daunting, but hold on! We need to compute μCi and σCi , which are the mean (i.e., average) and standard deviation, respectively, of the values of attribute Ak for training tuples of class Ci . We then plug these two quantities into Eq. (8.13), together with xk , to estimate P(xk |Ci).

For example, let X = (35,$40,000), where A1 and A2 are the attributes age and income, respectively. Let the class label attribute be buys computer. The associated class label for X is yes (i.e., buys computer = yes). Let's suppose that age has not been discretized and therefore exists as a continuous-valued attribute. Suppose that from the training set, we find that customers in D who buy a computer are 38 ± 12 years of age. In other words, for attribute age and this class, we have μ = 38 years

and σ = 12. We can plug these quantities, along with x1 = 35 for our tuple X, into Eq. (8.13) to estimate P(age = 35|buys computer = yes).

5. To predict the class label of X, P(X|Ci)P(Ci) is evaluated for each class Ci . The classifier predicts that the class label of tuple X is the class Ci if and only if

$$P(X|Ci)P(Ci) > P(X|Cj)P(Cj) \text{ for } 1 \leq j \leq m, j \neq i. \quad (8.15)$$

In other words, the predicted class label is the class Ci for which P(X|Ci)P(Ci) is the maximum.

"How effective are Bayesian classifiers?" Various empirical studies of this classifier in comparison to decision tree and neural network classifiers have found it to be comparable in some domains. In theory, Bayesian classifiers have the minimum error rate in comparison to all other classifiers. However, in practice this is not always the case, owing to inaccuracies in the assumptions made for its use, such as class-conditional independence, and the lack of available probability data.

Bayesian classifiers are also useful in that they provide a theoretical justification for other classifiers that do not explicitly use Bayes' theorem. For example, under certain assumptions, it can be shown that many neural network and curve-fitting algorithms output the maximum posteriori hypothesis, as does the naıve Bayesian classifier.

Predicting a class label using naıve Bayesian classification.

Table 8.1 Class-Labeled Training Tuples from the *AllElectronics* Customer Database

RID	age	income	student	credit_rating	Class: buys_computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

We wish to predict the class label of a tuple using naıve Bayesian classification, The training data were shown in Table 8.1. The data tuples are described by the attributes age, income, student, and credit rating. The class label attribute, buys computer, has two distinct values (namely, {yes, no}). Let C1 correspond to the class buys computer = yes and C2 correspond to buys computer = no. The tuple we wish to classify is

X = (age = youth, income = medium, student = yes, credit rating = fair)

We need to maximize P(X|Ci)P(Ci), for i = 1, 2.

P(Ci), the prior probability of each class, can be computed based on the training tuples:

P(buys computer = yes) = 9/14 = 0.643

P(buys computer = no) = 5/14 = 0.357

To compute P(X|Ci), for i = 1, 2, we compute the following conditional probabilities:

P(age = youth | buys computer = yes) = 2/9 = 0.222

P(age = youth | buys computer = no) = 3/5 = 0.600

P(income = medium | buys computer = yes) = 4/9 = 0.444

P(income = medium | buys computer = no) = 2/5 = 0.400

P(student = yes | buys computer = yes) = 6/9 = 0.667

P(student = yes | buys computer = no) = 1/5 = 0.200

P(credit rating = fair | buys computer = yes) = 6/9 = 0.667

P(credit rating = fair | buys computer = no) = 2/5 = 0.400

Using these probabilities, we obtain P(X|buys computer = yes) = P(age = youth | buys computer = yes) × P(income = medium | buys computer = yes) × P(student = yes | buys computer = yes) × P(credit rating = fair | buys computer = yes) = 0.222 × 0.444 × 0.667 × 0.667 = 0.044.

Similarly, P(X|buys computer = no) = 0.600 × 0.400 × 0.200 × 0.400 = 0.019.

To find the class, Ci , that maximizes P(X|Ci)P(Ci), we compute

P(X|buys computer = yes)P(buys computer = yes) = 0.044 × 0.643 = 0.028

P(X|buys computer = no)P(buys computer = no) = 0.019 × 0.357 = 0.007

Therefore, the naıve Bayesian classifier predicts buys computer = yes for tuple X.

try it now

A KTU
STUDENTS
PLATFORM

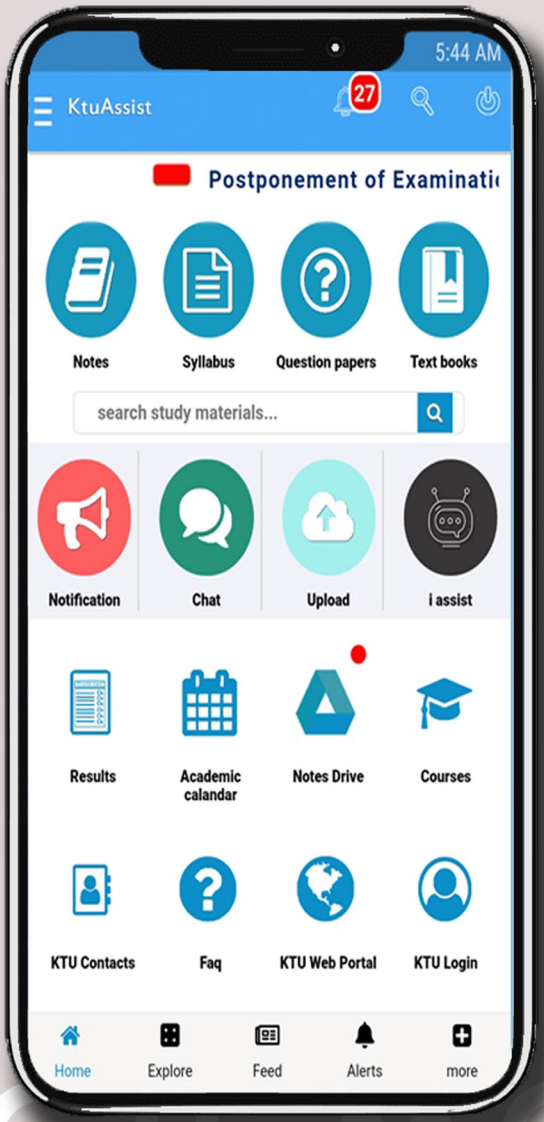
SYLLABUS

NOTES

TEXT BOOKS

QUESTION PAPERS

KTU NOTIFICATION



DOWNLOAD
IT
FROM
GOOGLE PLAY

CHAT

A FAQ

LOGIN

CALENDAR

MUCH MORE

DOWNLOAD APP

ktuassist.in

instagram.com/ktu_assist

facebook.com/ktuassist