Non-metric methods for pattern classification

-Slides compiled by Sanghamitra De

Non numeric data or nominal data

- We have considered pattern recognition based on feature vectors of real-valued and discrete-valued numbers, and in all cases there has been a natural measure of distance between such vectors.
- We concentrate now on specifying the values of a fixed number of properties by a property d-tuple For e.g.- property, consider describing a piece of fruit by the four properties of color, texture, d-tuple taste and smell.
- Then a particular piece of fruit might be described by the 4-tuple {*red, shiny, sweet, small*}, which is a shorthand for color = red, texture = shiny, taste = sweet and size = small.
- Such nominal data can be used for classification & categories be learnt from such non-metric data by moving beyond the notion of continuous probability distributions and metrics toward discrete problems that are addressed by rule-based or syntactic pattern recognition methods.

Decision Trees: Basics

- It is natural and intuitive to classify a pattern through a sequence of questions, in which the next question asked depends on the answer to the current question.
- Such a sequence of questions is displayed in a directed decision tree or simply tree, where by convention the first or root node is displayed at the top, connected by successive (directional) links or branches to other nodes.
- These are similarly connected until we reach terminal or leaf nodes, which have no further links.
- The classification of a particular pattern begins at the root node, which asks for the value of a particular property of the pattern.
- The different links from the root node correspond to the different possible values.

Decision Tree: Basics

- Based on the answer we follow the appropriate link to a subsequent or descendent node.
- In these trees the links must be mutually distinct and exhaustive, i.e., one and only one link will be followed.
- The next step is to make the decision at the appropriate subsequent node, which can be considered the root of a sub-tree.
- We continue this way until we reach a leaf node, which has no further question.
- Each leaf node bears a category label and the test pattern is assigned the category of the leaf node reached.
- The simple decision tree has one benefit of trees over many other classifiers such as neural networks: interpretability.

Decision Tree: Basics

- It is a straightforward matter to render the information in such a tree as logical expressions.
- Such interpretability has two manifestations. First, we can easily interpret the decision for any particular test pattern as the conjunction of decisions along the path to its corresponding leaf node.
- Thus if the properties are {taste, color, shape, size}, the pattern x = {sweet, yellow, thin, medium} is classified as **Banana** because it is (color = yellow) AND (shape = thin).
- Second, we can occasionally get clear interpretations of the categories themselves, by creating logical descriptions using conjunctions and disjunctions. For instance the tree shows **Apple** = (green AND medium) OR(red AND medium).
- Thirdly, these trees lead to rapid classification, employing a sequence of typically simple queries.
- Finally, the trees provide a natural way to incorporate prior knowledge from human experts.

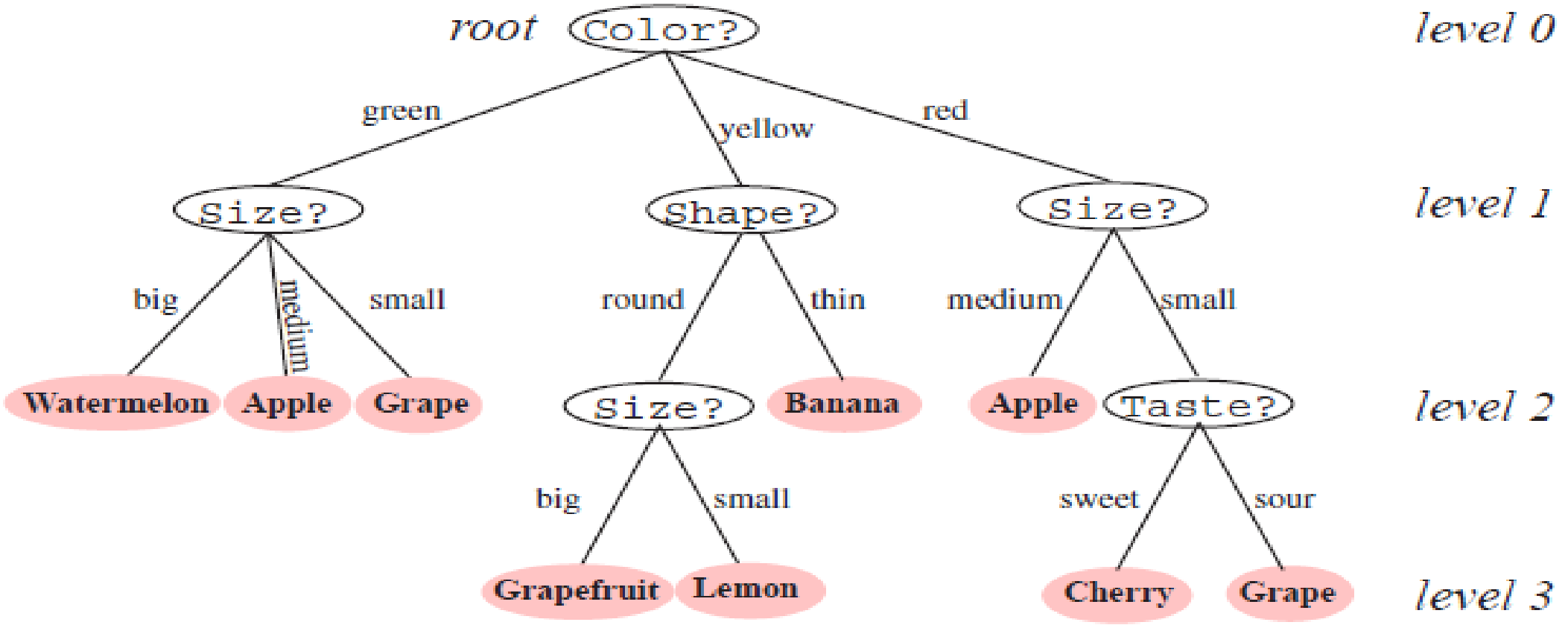

Figure 8.1: Classification in a basic decision tree proceeds from top to bottom. The questions asked at each node concern a particular property of the pattern, and the downward links correspond to the possible values. Successive nodes are visited until a terminal or leaf node is reached, where the category label is read. Note that the same question, `Size?`, appears in different places in the tree, and that different questions can have different numbers of branches. Moreover, different leaf nodes, shown in pink, can be labeled by the same category (e.g., **Apple**).

CART (Classification and Regression Trees)

Current question: how to use training data to create or "grow" a decision tree.

- Any decision tree will progressively split the set of training examples into smaller and smaller subsets.
- It would be ideal if all the samples in each subset had the same category label.
- In that case, we would say that each subset was pure, and could terminate that portion of the tree.
- Usually, however, there is a mixture of labels in each subset, and thus for each branch we will have to decide either to stop splitting and accept an imperfect decision, or instead select another property and grow the tree further.
- We use a recursive tree-growing process: given the data represented at a node, either declare that node to be a leaf (and state what category to assign to it), or find another property to use to split the data into subsets.
- A generic tree-growing methodology know as CART (Classification and Regression Trees) is to be used.

CART: Six questions

- CART provides a general framework that can be instantiated in various ways to produce different decision trees.
- In the CART approach, six general kinds of questions arise:
  1) *Should the properties be restricted to binary-valued or allowed to be multivalued? That is, how many decision outcomes or splits will there be at a node?*
  2) *Which property should be tested at a node?*
  3) *When should a node be declared a leaf?*
  4) *If the tree becomes "too large," how can it be made smaller and simpler, i.e., pruned?*
  5) *If a leaf node is impure, how should the category label be assigned?*
  6) *How should missing data be handled?*

Q1: Number of splits

- Each decision outcome at a node is called a *split*, since it corresponds to splitting a subset of the training data.
- The root node splits the full training set; each successive decision splits a proper subset of the data. The number of splits at a node is closely related to question 2, specifying which particular split will be made at a node.
- In general, the **number of splits is set by the designer**, and could vary throughout the tree.
- The number of links descending from a node is sometimes called the node's *branching factor* or *branching ratio*, denoted B.
- However, every decision (and hence every tree) can be represented using just binary decisions.

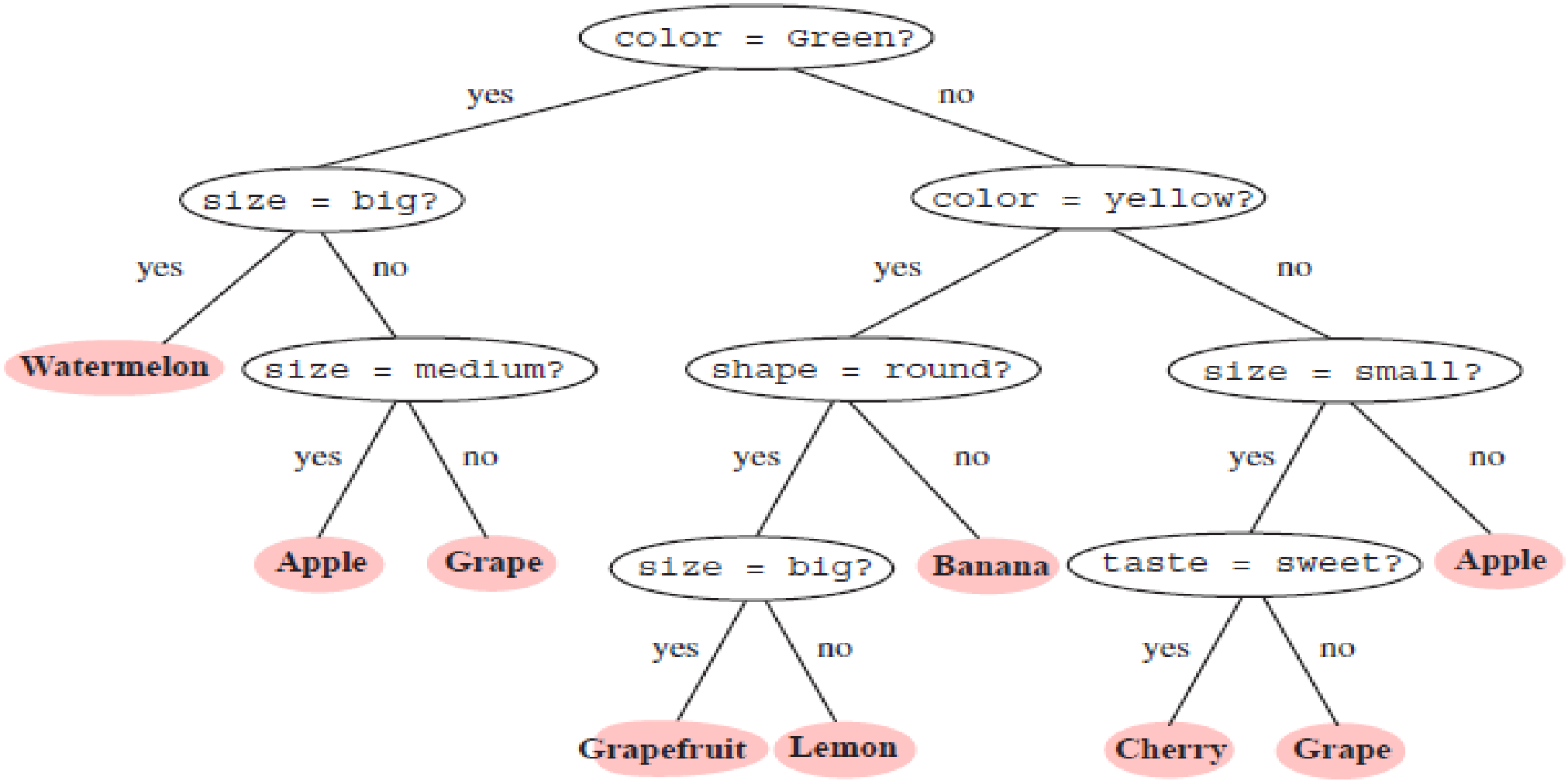

Figure 8.2: A tree with arbitrary branching factor at different nodes can always be represented by a functionally equivalent binary tree, i.e., one having branching factor $B = 2$ throughout. By convention the "yes" branch is on the left, the "no" branch on the right. This binary tree contains the same information and implements the same classification as that in Fig. 8.1.

Q2: Test selection and node impurity

- Much of the work in designing trees focuses on deciding which property test or query should be performed at each node.
- With non-numeric data, there is no geometrical interpretation of how the test at a node splits the data.
- For numerical data, there is a simple way to visualize the decision boundaries that are produced by decision trees.
- The fundamental principle underlying tree creation is that of simplicity: we prefer decisions that lead to a simple, compact tree with few nodes.
- To this end, **we seek a property test T at each node N that makes the purity data reaching the immediate descendent nodes as "pure" as possible.**

Q2: Test selection and node impurity

- Different mathematical measures of impurity have been proposed, all of which have basically the same behavior.
- Let $i(N)$ denote the impurity of a node N.
- In all cases, we want $i(N)$ to be 0 if all of the patterns that reach the node bear the same category label, and to be large if the categories are equally represented.
- The most popular measure is the ***entropy impurity*** (or occasionally *information impurity*):

$$i(N) = -\sum_j P(\omega_j) \log_2 P(\omega_j),$$

where $P(\omega_j)$ is the fraction of patterns at node N that are in category ω_j . By the well-known properties of entropy, if all the patterns are of the same category, the impurity is 0; otherwise it is positive, with the greatest value occurring when the different classes are equally likely.

Q2: Test selection and node impurity

- Another definition of impurity is particularly useful in the two-category case.
- Given the desire to have zero impurity when the node represents only patterns of a single category, the simplest polynomial form is:

$$i(N) = P(\omega_1)P(\omega_2).$$

- This can be interpreted as a **variance impurity** since under reasonable assumptions it is related to the variance of a distribution associated with the two categories.
- A generalization of the variance impurity, applicable to two or more categories, is the ***Gini impurity*.**
- This is just the expected error rate at node N if the category label is selected randomly from the class distribution present at N.
- Another impurity measure, the ***misclassification impurity*** measures the minimum probability that a training pattern would be misclassified at N.

Q3: When to stop splitting

Problem: when to stop splitting during the training of a binary tree.

- If we continue to grow the tree fully until each leaf node corresponds to the lowest impurity, then the data has typically been overfit.
- In the extreme but rare case, each leaf corresponds to a single training point and the full tree is merely a convenient implementation of a lookup table; it thus cannot be expected to generalize well in (noisy) problems having high Bayes error.
- Conversely, if splitting is stopped too early, then the error on the training data is not sufficiently low and hence performance may suffer.
- One traditional approach is to **use techniques of validation and cross-validation**.

Q3: When to stop splitting

- In ***validation***, the tree is trained using a subset of the data (for instance 90%), with the remaining (10%) kept as a validation set.
- We continue splitting nodes in successive layers until the error on the validation data is minimized.
- ***Cross-validation*** relies on several independently chosen subsets.
- Another method is to set a (small) ***threshold value*** in the reduction in impurity; splitting is stopped if the best candidate split at a node reduces the impurity by less than that pre-set amount.
- Merits:
  - ✓ tree is trained directly using all the training data.
  - ✓ leaf nodes can lie in different levels of the tree, which is desirable whenever the complexity of the data varies throughout the range of input.
- Demerit: it is often difficult to know how to set the threshold

Q3: When to stop splitting

Another method is to trade complexity for test accuracy by **splitting until a minimum in a new, global criterion function, is reached**.

$$\alpha \cdot size + \sum_{leaf\ nodes} i(N),$$

Here size could represent the number of nodes or links and α is some positive constant.

Q3: When to stop splitting

- An alternative approach is to use a **stopping criterion based on the statistical significance of the reduction of impurity**.
- During tree construction, we estimate the distribution of all the Δi for the current collection of nodes; we assume this is the full distribution of Δi.
- For any candidate node split, we then determine whether it is statistically different from zero, for instance by a chi-squared test.
- ***If a candidate split does not reduce the impurity significantly, splitting is stopped***.

Q4: Pruning

- Occasionally, **stopped splitting** suffers from the lack of sufficient look ahead, a phenomenon called the *horizon effect*.
- The determination of the optimal split at a node effect *N* is not influenced by decisions at *N*'s descendent nodes, i.e., those at subsequent levels.
- In stopped splitting, node *N* might be declared a leaf, cutting off the possibility of beneficial splits in subsequent nodes; as such, a stopping condition may be met "too early" for overall optimal recognition accuracy.
- Informally speaking, the stopped splitting biases the learning algorithm toward trees in which the greatest impurity reduction is near the root node.
- The principal alternative approach to stopped splitting is *pruning*.
- In pruning, a tree is grown fully, that is, until leaf nodes have minimum impurity — beyond any putative "horizon."

Q4: Pruning

- Then, all pairs of neighboring leaf nodes (i.e., ones linked to a common antecedent node, one level above) are considered for elimination.
- **Any pair whose elimination yields a satisfactory** (small) **increase in impurity is eliminated, and the common antecedent node declared a leaf.** (This antecedent, in turn, could itself be pruned.)
- Clearly, such merging or joining of the two leaf nodes is the inverse of splitting.
- It is not unusual that after such pruning, the leaf nodes lie in a wide range of levels and the tree is unbalanced.
- Although it is most common to prune starting at the leaf nodes, this is not necessary: cost-complexity pruning can replace a complex subtree with a leaf directly.

Q4: Pruning

- Benefits of pruning:

  It avoids the horizon effect; further, since there is no training data held out for cross-validation, it directly uses all information in the training set.

- Naturally, this comes at a greater computational expense than stopped splitting, and for problems with large training sets, the expense can be prohibitive.
- For small problems, though, these computational costs are low and pruning is generally to be preferred over stopped splitting.
- What we have been calling stopped training and pruning are sometimes called pre-pruning and post-pruning, respectively.

Q5: Assignment of leaf node labels

- Assigning category labels to the leaf nodes is the simplest step in tree construction.
- If successive nodes are split as far as possible, and **each leaf node corresponds to patterns in a single category (zero impurity), then of course this category label is assigned to the leaf.**
- In the more typical case, where either stopped splitting or pruning is used and the **leaf nodes have positive impurity, each leaf should be labeled by the category that has most points represented.**
- An extremely small impurity is not necessarily desirable, since it may be an indication that the tree is overfitting the training data.

Q6: Missing attributes

- Classification problems might have missing attributes during training, during classification, or both.
- Consider first training a tree classifier despite the fact that some training patterns are missing attributes.
- A naive approach would be to delete from deficient consideration any such *deficient patterns*; however, this is quite wasteful and should be employed only if there are many complete patterns.
- A better technique is calculate impurities at a node N using only the attribute information present.
- Suppose there are n training points at N and that each has three attributes, except one pattern that is missing attribute x_3.
- To find the best split at N, we calculate possible splits using all n points using attribute x_1, then all n points for attribute x_2, then the $n-1$ non-deficient points for attribute x_3.
- Each such split has an associated reduction in impurity, calculated with different numbers of patterns.

Q6: Missing attributes

- Consider how to create and use trees that can classify a deficient pattern.
- If we suspect that such deficient test patterns will occur, we must modify the training procedure.
- The basic approach during classification is to use the traditional ("primary") decision at a node whenever possible (i.e., when the queries involves a feature that is present in the deficient test pattern) but to use alternate queries whenever the test pattern is missing that feature.
- During training then, in addition to the primary split, each non-terminal node N is given an ordered set of *surrogate splits*, consisting of an attribute label and a surrogate rule.
- The first such surrogate split maximizes the "predictive association" with the primary split. A simple measure of the predictive association of two splits s_1 and s_2 is merely the numerical count of patterns that are sent to the "left" by both s_1 and s_2 plus the count of the patterns sent to the "right" by both the splits.
- The second surrogate split is defined similarly, being the one which uses another feature and best approximates the primary split in this way.
- A method closely related to surrogate splits is that of *virtual values*, in virtual which the missing attribute is assigned its most likely value.

Computational complexity

Nevertheless, the result that for fixed dimension d the training is $O(dn^2 \log n)$ and classification $O(\log n)$ is a good rule of thumb; it illustrates how training is far more computationally expensive than is classification, and that on average this discrepancy grows as the problem gets larger.

Resources

- Pattern Classification- Duda, Hart & Stork.

End