Lecture 11

Starting to look at classification, which is our first supervised technique.

Given a collcetion of records (the **training set**), where each record contains a set of **attributes**, and one of the attributes is the **class**. Then, find a model for class attribute as a function of the values of other attributes.

The goal here is for previously unseen records should be assigned to a class as accurately as posible. A **test set** is used to determine the accuracy of the model. Usually, the given data set is divided into training and test sets, with training set used to build the model and test set used to validate it.

Can think of this as a two step process: learn a model from the training data, the apply the model on the test set.

This is a bit different than clustering - in clustering, we wanted to find the k clusters given the structure of the data. Here, we have data labeled, and we want to understand why a each data point should be assigned to a given class.

Some examples:

- predicting tumor cells as benign or malignant
- classifying credit card transactions as legitimate or fraudulent
- etc.

We'll cover decision trees, neural networks, Bayes models, and support vector machines. We'll also cover some regression techniques.

Let's approach this problem from a lazy perspective, given the tools we have so far. We have a tool for distance - we can compare distances between points. Maybe we say "if the attributes are similar, then the class is similar." We store the training set, and then for each test data, compare distances to each one, and output the class of the closest one.

Examples of such classifiers (called instance based classifiers:)

- Rote-learner: memorizes the entire training data, and performs classification only if attributes of record match one of the training examples exactly.
- Nearest Neighbor: Uses the k "closest" points (the nearest neighbors) for performing classification. The basic idea: if it walks like a duck, it quacks like a duck, then it's a duck.

Vornoi Diagram: Finding the single nearest neighbor, and how well the space is separated.

Computing the distance between points: use the Euclidean distance between points. Then, to determine the class, take the majority of vote fo class labels among the k-nearest beighbor.s Or, weigh the vote according to distance: weight factor $w = 1/d^2$

Here, if we choose k that is too large, then the neighbord may include points from other classes. If we choose k that is too small, then we may be too sensitive to noise points.

There are also some scaling issues with this approach: attributes may have to be scaled to prevent distance measures from being dominated by one of the attributes.

Let's consider the volume of a sphere: as the dimensions increase, the volume becomes concentrated on the shell (surface). Basically, we're saying that the data may not be concentrated where we expect it to be.

High dimensionality:

- hyper-sphere volume of unit radius goes to 0 as dimensionality goes to infinity.
- All data in the shell
- In high dimensions, kNN can be problematic

k-NN classifiers are "lazy learners" - it does not build model explicitly, and unlike eager learners such as decision tree induction and rule based systems. However, classifying unknown records are relatively expensive.

Decision Trees

We basically build a tree, that maps decisions - start at the root, and we split left or right at each node, depending on some condition.

How do we build this tree? We'll look at Hunt's algorithm, which is the most intuitive.

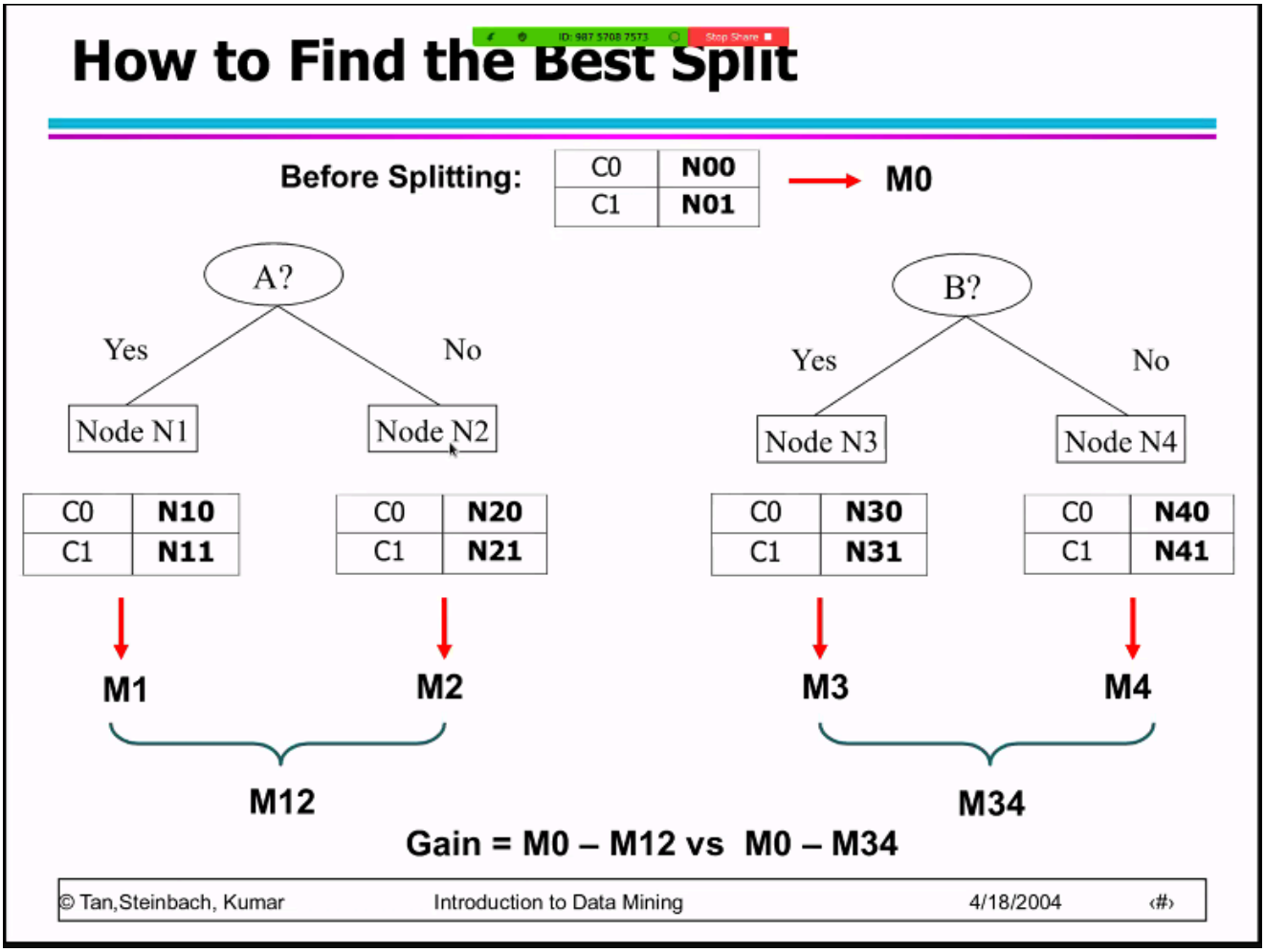
General structure of Hunt's algorithm:

- Let D_t be the set of training records that reach a specific node t
- General procedure:
    - If D_t contains records that belong to the same class y_t, then t is a leaf node, and labeled as y_t (the first base case scenario)
    - If D_t is an empty set, then t is a leaf node labeled by the default class y_d (the second base case scenario)
    - If D_t contains records that belong to more than one class, use an attribute test to split the data into smaller subsets. Recursively apply the procedure to each subset.

How do we determine the best split? Let's say before splitting we have 10 records in class 0, and 10 records of class 1. We might ask different things: own car? car type? student id? In the first one, we have about and even split, which doesn't help. In the last one, everyone splits into their own bucket. The one in the middle makes a good differentiation between class 0 and class 1. So how do we go about finding that split?

We want a measure of node impurity - non-homogenous, high degree of impurtiy, or homogenous, low degree of impurity. In a greedy approach, nodes with homogenous class distribution are preferred.

Measures of Node Impurity: gini index, entropy, misclassification error.



Based on the answer to the question of gain here, we choose which attribute to split on.

Gini index for a given node t: $GINI(t) = 1 - \sum_j [p(j|t)]^2$ Note that $p(j|t)$ is the relative frequency of class j at node t.

- Max $(1 - 1/n_c)$: when records are equally distributed among all classes, implying the least interesting information. Note that n_c is the number of classes.
- Minimum (0.0): when all records belong to one class, implying th emost interesting information

When a node p is split into k partitions, the quality of the split is computed as:
$GINI_{split} = \sum_{i=1}^{k} \frac{n_i}{n} GINI(i)$, when n_i is the number of records at child i, and n is the number of records at node p.

Alternatives measures for splitting:

- Entropy: measures the homogeneity of a node. These computations are similar to gini index computations. Follow a similar process: compute the entropy of each node, then compute the overall gain.