CLASSIFIER

Consider an $n \times d$ data matrix $\underline{D}$, where n is the # of instances and d is the number of features, and an $n+1$ class vector $\underline{C}$ which contains the various classes $c_1, \dots c_{n+1}$.

Note that,

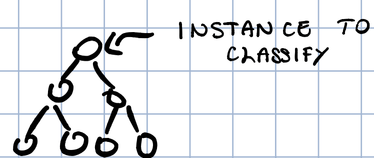
$\underline{D}$ → can contain either categorical features or numerical features.

$\underline{C}$ → contains a single categorical feature

A CLASSIFIER is a statistical approach that is able to assign to every row of $\underline{D}$ a unique class in $\underline{C}$.

There are various methods to build classifiers models such as:

- DECISION TREES,



- BAYES CLASSIFIERS,

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

Q: What is the meaning of classification for a given instance?

R: It means using the dataset and class label to build a classifier model that assigns to the given instance, which does not have to be contained in the original dataset, the class label with the highest probability.

After we build a classifier, we have to test the quality of the results obtained. This is important, since different classifiers may yield different results depending on the model used.

To test the ACCURACY of a classifier we have to know the correct class label associated with a feature. This is implemented in practice by splitting the original dataset.

$$\underline{D} \begin{cases} n_1 \times d & \text{matrix used for TRAINING to build the classifier.} \\ n_2 \times d & \text{matrix used to TEST the previously built classifier.} \end{cases}, \quad n_1 + n_2 = n$$

Having done this, the testing phase is trivial:

For each instance of the TEST_MATRIX,

compute the class label assigned by the classifier C_1 with the above class label C_2.

If $C_1 \neq C_2$, then we increment the # of errors made by the classifier.

Return $\dfrac{\#\text{ of errors}}{\#\text{ of test instances}}$ → AVERAGE NUMBER OF ERRORS

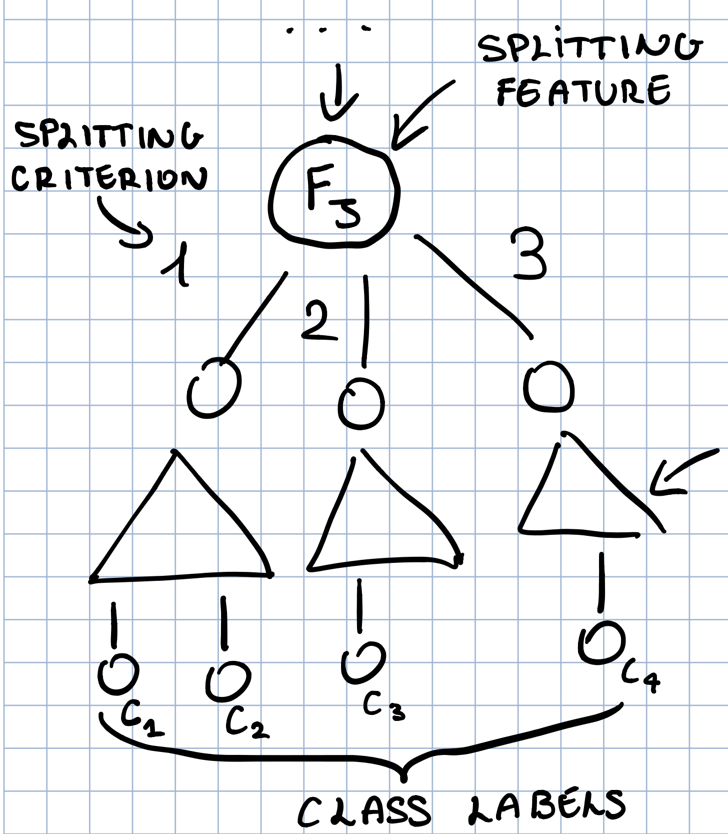
Note that,

1 − AVG. ERROR = AVG. ACCURACY

Decision Trees

A **DECISION TREE** is a finite, oriented tree, with a root. It is a model that can be used to build classifiers.

In a decision tree takes in input the instance to classify and uses the values of the intermediate nodes, called the **SPLITTING FEATURES**, to traverse the tree and arrive at one of the leaves, which represent class labels.

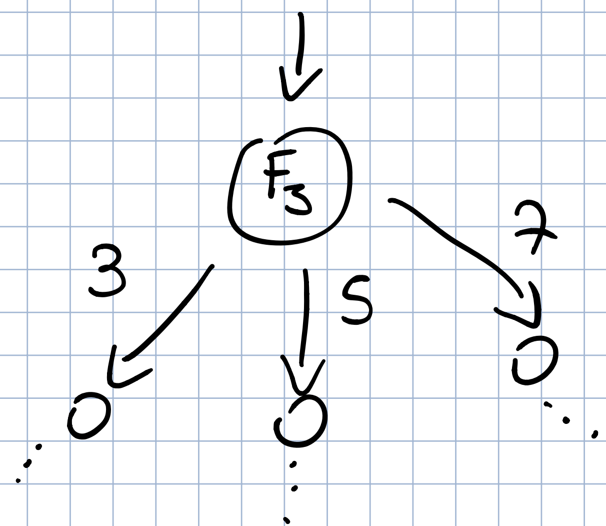


Based on the value of F_3 we chose where to go.

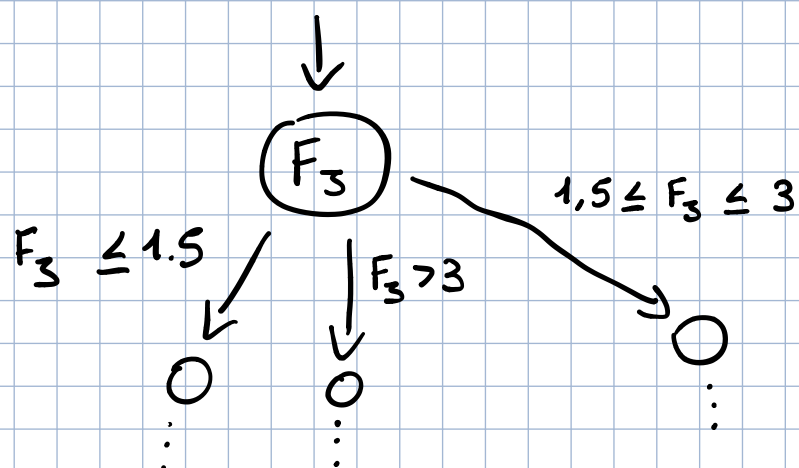
If $F_3 = 3$, then we go into this sub tree

The SPLITTING CRITERION depends on the nature of F_3. If F_3 is a categorical feature then we split based on specific values of F_3

$$S_{F_3} = \{3, 5, 7\}$$



If F_3 is a numerical feature, then we split based on ranges of values of F_3



It may happen to have decision trees in which both criteria are used, since its possible to have datasets with both categorical and numerical features.

iD3

An example of a decision tree based on information theory is iD3 which stands for "iterative Dicotomizer".

The iD3 algorithm builds the decision tree by computing the MUTUAL INFORMATION between the various features F_j and the class vector C.

In particular it does the following:

- In the root node we put the feature F_j that maximizes the mutual information with the class vector C. That is,

$$F_j = \underset{F_i}{\text{ARG MAX}} \ I(F_i, C)$$

Since we assume F_j to be discrete, the splitting criteria for this node is trivial: for every possible value of F_j add an edge labeled with that value and add a node.

- Suppose in the previous step F_3 could assume 3 possible values

To fill the new nodes we have to eliminate the F_3 column from the original dataset and partition the new dataset into three sub-datasets based on the value of F_3.

F_3	F_{3+1}	...	C
1	D_1		
1			
1			
2	D_2		
2			
2			
3	D_3		
3			
3			

D_1 := Dataset which contains the rows from the original dataset where $F_3 = 1$.

After having done this split we can proceed recursively in the new nodes.

- This step is repeated until the dataset consist of only the class label and the values of the class label are all equal.

  At this point we add a leaf to the tree in which we insert the remaining class label.

Remember that we assumed that each feature was a discrete r.v. This comes from the fact that, in its original form, iD3 could be applied only to categorical features. If we have numerical features we have to discretize them first.

OSS: We use the mutual information because it is a measure of correlation of two r.v.

Since we are building a classifier, we are interested in high correlation between features and class labels.

Bayes Classifier ~ 03:00/3

Consider a test instance, also called **EVIDENCE**,

$$\underline{x} = [x_1, x_2, \dots, x_d]$$

Once again, we want to assign to $\underline{x}$ a class label. To do so we would like to compute $P(C_j | \underline{x})$. If we know that for every class label C_j we can assign to $\underline{x}$ the one that maximizes said probability. That is, if we define

$\tilde{c}(\underline{x})$ = class assigned to instance $\underline{x}$.

Then we can do the following,

$$\tilde{c}(\underline{x}) = \underset{C_j}{\text{ARG MAX}}\ P(C_j | \underline{x})$$

OSS: The function $\tilde{c}$ is our classifier built using the training data D_t.

In this way to build a classifier we only have to compute the various probabilities $P(C_j | \underline{x})$.

Let us use the **BAYES RULE** to rewrite $P(c_j \mid \underline{x})$ as follows

$$\underbrace{P(c_j \mid \underline{x})}_{\text{POSTERIOR PROBABILITY}} = \frac{\overbrace{P(\underline{x} \mid c_j)}^{\text{LIKELIHOOD}} \cdot \overbrace{P(c_j)}^{\text{PRIOR PROBABILITY}}}{P(\underline{x})}$$

Since we are only interested in the argmax values, we can do the following semplifications,

$$\tilde{c} = \underset{c_j}{\text{ARGMAX}} \; P(c_j \mid \underline{x})$$

$$= \underset{c_j}{\text{ARG MAX}} \; \frac{P(\underline{x} \mid c_j) \cdot P(c_j)}{P(x)}$$

$P(x)$ does not depend on c_j, and thus has no effect on the argmax.

$$= \underset{c_j}{\text{ARG MAX}} \; P(\underline{x} \mid c_j) \cdot P(c_j)$$

To build a BAYES CLASSIFIER we thus need to compute the following probabilities:

- $P(\underline{x} \mid c_j)$ (LIKELIHOOD)
- $P(c_j)$ (PRIOR PROBABILITY)

To commute $P(C_j)$ we simply have to estimate the pmf of the class features using the training data. In particular we have,

$$P(C_j) := \frac{\text{\# of instances in train data with class } C_j}{\text{\# of intances in train data.}}$$

To commute the likelihood, that is $P(\underline{x} | C_j)$, we have to compute a multi-variate probability conditioned to C_j. In practice this means that, fixed a C_j, we have to estimate the multi-vaiate pmf or pdf in the sub-dataset defined by C_j.

If $\underline{x}$ is a vector of continuous random variables, we can estimate its pdf using the KERNEL METHOD FOR MUTI-VARIATE R.V.

Naïve Bayes Classifier

By introducing a particular assumption called the **naïve assumption** we can simplify the construction of the Bayes Classifier.

Naïve Assumption : features in the dataset are independent.

This simplifies the computation of the likelihood, since we now have that

$$P(\underline{x} \mid C_j) = \prod_{i=1}^{d} P(x_i \mid C_j)$$

We now simply need to estimate the pmf for every feature F_j.

Usually we also have the following,

Gaussian Assumption : features are independent and gaussian distributed.

With this other assumption we are only interested in computing the mean μ and variance σ^2 for each feature.

ASSIGNMENT n° 5 ~ 06:00/4

- NAIVE BAYES + GAUSSIAN

Split the training dataset D into $D_1, \dots, D_m$ where m = # of classes.
d = # of features.

For each sub-dataset D_j associated with class C_j compute

$\hat{\mu}_{ij}$:= sample mean of feature F_i in D_j

$\hat{\sigma}^2_{ij}$:= sample variance of feature F_i in D_j

Then, to classify a test instance $\underline{x}$, we can compute the likelihood as follows,

$$P(\underline{x} \mid C_j) = \prod_{i=1}^{d} P(x_i \mid C_j)$$

$$= \prod_{i=1}^{d} \frac{1}{\sqrt{2 \cdot \pi \cdot \hat{\sigma}_{i,s}^2}} \cdot e^{-\frac{(x_i - \hat{\mu}_{i,s})^2}{2 \hat{\sigma}_{i,s}^2}}$$