Data Mining:
Classification/Supervised Learning

Definitions

**Data.** Consider a set \( A = \{A_1, \ldots, A_n\} \) of attributes, and an additional categorical attribute \( C \), which we call a **class attribute** or **category attribute**. 
\[
\text{dom}(C) = \{c_1, \ldots, c_k\}
\] 
We call each value \( c_i \) a **class label** or a **category label**.

The **learning dataset** is a relational table \( D \).

Two formats:

1. **Training (data)set.** \( D \) has schema \( (A_1, \ldots, A_n, C) \), i.e.,
   
   *for each element of the dataset we are given its class label.*

2. **Test (data)set.** \( D \) has schema \( (A_1, \ldots, A_n) \), i.e.,
   
   *the class labels of the records in \( D \) are not known.*

**Classification Problem.** Given a (training) dataset \( D \), construct a **classification/prediction function** that correctly predicts the class label for every record in \( D \).

**Classification function = prediction function = classification model = classifier.**

**Supervised learning** because training set contains class labels. Thus we can compare (supervise) predictions of our classifier.

**Classification Methodology**

**Naïve Bayes.** Estimation of probability that a record belongs to each class.

**Neural Networks.** Graphical models that construct a "separation function" based on the training set data.
Support Vector Machines (SVMs). Linear models for two-class classifiers.

Association Rules. Infer association rules with class label on the right side.

Decision Trees. Build a tree-like classifier. (key advantage: human-readable!)

Decision Trees

Decision tree-based classifiers are simple and efficient.

Decision trees. Let $A = \{A_1, \ldots, A_k\}$ are the dataset attributes and $C$ is a class label. Let $\text{dom}(C) = \{c_1, \ldots, c_k\}$. A decision tree over $A$ and $C$ is a tree $T = (V, E)$ such that,

1. Each non-leaf node $v \in V$ is labeled with some $A_i \in A$.
2. Each leaf node $v_l \in V$ is labeled with some class label $c_i \in \text{dom}(C)$.
3. Each edge $E = (v, v')$, where $\text{label}(v) = A_i$ is labeled with some value $a \in \text{dom}(A_i)$.
4. No attribute $A_i \in A$ can appear more than once on each path from root to leaf.

A decision tree can be used as a classifier as follows:

• Consider a record $t = (a_1, a_2, \ldots, a_n)$.

• Start at the root node $r$ of the decision tree $T$. Let $\text{label}(r) = A_i$. Find the edge $e = (r, v)$, such that $\text{label}(e) = t(A_i) = a_i$, i.e., follow the outgoing edge from $r$ that is labeled with the value of $A_i$ in $t$.

• For node $v$ visited next, continue the same process: follow the outgoing edge labeled with the value of the $\text{label}(v)$ attribute found in $t$.

• When you reach leaf node $l$, the label $\text{label}(l)$ will be the $\text{class}(t)$.

C4.5 Algorithm: Induction of Decision Trees

The C4.5. Algorithm for decision tree induction was originally proposed by Quinlan in [1].

Input/Output The C4.5 algorithm for decision tree induction has three parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>I/O</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D$</td>
<td>input</td>
<td>the training dataset</td>
</tr>
<tr>
<td>$A$</td>
<td>input</td>
<td>the list of attributes</td>
</tr>
<tr>
<td>$T$</td>
<td>output</td>
<td>the constructed decision tree</td>
</tr>
</tbody>
</table>
Algorithm idea.  The C4.5 Algorithm is a recursive decision tree induction algorithm. The algorithm has the following three main steps:

1. Termination conditions. The algorithm has two termination conditions:
   (a) $D$ contains records with the same class label $c$. In this case, the algorithm creates a tree that consists of a single node, and assigns to it the class label $c$.
   (b) $A = \emptyset$: there are no more attributes left to consider. In this case, the algorithms creates a tree that consists of a single node, and assigns to it the label of the plurality records in $D$.

2. Selection of the splitting attribute. The algorithm chooses the attribute $A_i$ to be used to split the dataset.

3. Tree construction. The algorithm does the following:
   (a) Creates a tree node $r$ labeled $A_i$.
   (b) Splits the dataset $D$ into $\text{dom}(A_i)$ subsets $D_1, \ldots, D_{\text{dom}(A_i)}$, and recursively calls itself for each subset $D_j$, with the reduced list of attributes $A - \{A_i\}$.
   (c) Creates $\vert \text{dom}(A_i) \vert$ edges from $r$ to the roots for trees $T_1, \ldots, T_{\vert \text{dom}(A_i) \vert}$ returned by the recursive calls. Labels each edge with the appropriate value from $\text{dom}(A_i)$.
   (d) Returns the constructed tree.

The pseudocode for the C4.5 Algorithm is shown in Figure 1.

Selection of the Splitting Attribute

The C4.5. Algorithm relies on an external function to identify the splitting attribute on each step. In this section we discuss how to find splitting attributes.

Information Entropy. Consider a relational dataset $D$ over a list of attributes $A = \{A_1, \ldots, A_n, C\}$, where $C$ is the class attribute of $D$. Let $\text{dom}(C) = \{c_1, \ldots, c_k\}$. Let $D_i = \{t \in D \vert \text{class}(D) = c_i\}$. Thus, $D = D_1 \cup D_2 \cup \ldots \cup D_n$. As $Pr(C = c_i)$ we denote the probability that a randomly chosen record $t \in D$ will have the class label of $c_i$. We can see that

$$Pr(C = c_i) = \frac{\vert D_i \vert}{\vert D \vert}.$$

The entropy of the dataset $D$ w.r.t. $C$ is defined as follows:

$$\text{entropy}(D) = - \sum_{i=1}^{k} Pr(C = c_i) \cdot \log_2(Pr(C = c_i)).$$

Entropy is measured in bits.

(Note: In this computation, we assume that $0 \cdot \log_2(0) = 0$.)
Algorithm C4.5(D, A, T, threshold);
begin  // Step 1: check termination conditions
if for all d ∈ D: class(d) = c then
create leaf node r;
label(r) := c;
T := r;
else if A = ∅ then
c := find_most_frequent_label(D);
create leaf node r;
label(r) := c;
else  //Step 2: select splitting attribute
A_g := selectSplittingAttribute(A, D, threshold);
if A_g = NULL then  //no attribute is good for a split
create leaf node r;
label(r) := find_most_frequent_label(D);
T := r;
else  // Step 3: Tree Construction
create tree node r;
label(r) := A_g;
for each v ∈ dom(A_g) do
D_v := {t ∈ D| A_g(t) = v};
if D_v ≠ ∅ then
C4.5(D_v, A - {A_g}, T_v);  //recursive call
append T_v to r with an edge labeled v;
endif
endfor
endif
endif
end

Figure 1: C4.5 algorithm for decision tree induction.

function selectSplittingAttribute(A, D, threshold);  //uses information gain
begin
p0 := entropy(D);
for each A_i ∈ A do
p[A_i] := entropyA_i(D);
Gain[A_i] = p0 - p[A_i];  //compute info gain
endfor
best := arg(findMax(Gain[]));
if Gain[best] > threshold then return best
else return NULL;
end

function selectSplittingAttribute(A, D, threshold);  //uses information gain ratio
begin
p0 := entropy(D);
for each A_i ∈ A do
p[A_i] := entropyA_i(D);
Gain[A_i] := p0 - p[A_i];  //compute info gain
gainRatio[A_i] := Gain[A_i]/entropy(A_i);  //compute info gain ratio
endfor
best := arg(findMax(gainRatio[]));
if Gain[best] > threshold then return best
else return NULL;
end

Figure 2: selectSplittingAttribute() functions using information gain and information gain ratio measures.
Properties of entropy. Entropy of a random variable defined the predictability of the observed results. If certain values are more likely, it is easier to predict the outcomes. If all values are equally likely, it is much harder to do so.

The higher the entropy, the more unpredictable are the outcomes.

The entropy of a homogenous dataset in which each class label has the same probability of occurring is \( \log_2 k \), i.e., the number of bits necessary to represent \( k \).

\[
\text{entropy}(D) = - \sum_{i=1}^{k} \frac{1}{k} \cdot \log_2 \left( \frac{1}{k} \right) = - \log_2 \left( \frac{1}{k} \right) \cdot \sum_{i=1}^{k} \frac{1}{k} = \log_2 k
\]

The entropy of a dataset where only one class label out of \( k \) is present is 0.

\[
\text{entropy}(D) = - \sum_{i=1}^{k-1} 0 \cdot \log_2 0 - 1 \cdot \log_2 1 = 0.
\]

Entropy measures the impurity of data. The higher the entropy, the more impure the data is.

Information Gain. Idea: we want to select the attribute that splits the dataset \( D \) into most pure subsets. We introduce information gain measure. Given a dataset \( D \) over the list \( A = \{A_1, \ldots, A_k\} \) of attributes, the entropy of \( D \) after being split using attribute \( A_i \) with domain \( \text{dom}(A_i) = \{v_1, \ldots, v_s\} \) is defined as:

\[
\text{entropy}_{A_i}(D) = \sum_{j=1}^{s} \frac{|D_j|}{|D|} \cdot \text{entropy}(D_j),
\]

where \( D_j = \{t \in D | t[A_i] = v_j\} \).

The information gain achieved by the split is the difference between the entropy of \( D \) before and after the split:

\[
\text{Gain}(D, A_j) = \text{entropy}(D) - \text{entropy}_{A_j}(D).
\]

Information Gain Ratio. Information Gain Ratio is the normalized version of the information gain measure:

\[
\text{gainRatio}(D, A_j) = \frac{\text{Gain}(D, A_j)}{- \sum_{j=1}^{s} \left( \frac{|D_j|}{|D|} \cdot \log_2 \left( \frac{|D_j|}{|D|} \right) \right)}
\]

(essentially, we normalize information gain by the “entropy” of the split itself.)

Using Information Gain and Information Gain Ratio to select splitting attributes

Figure 2 shows the two versions of the selectSplittingAttribute() function. The first version uses the information gain measure to determine the splitting attribute, while the second version uses the information gain ratio.

Both algorithms do the following:
1. Compute the entropy of the current dataset.

2. Compute the entropy after splitting the dataset using each of the available attributes.

3. Find the attribute with the best information gain/information gain ratio.

4. If the information gain/information gain ratio exceed the threshold, the attribute is returned. Otherwise, NULL is returned, as no attribute leads to a significant improvement in the entropy.

Classifier Evaluation

Accuracy Measures

**Notation.** Let $T$ be a classifier constructed by any supervised learning algorithm given a training set $D$.

Let $D'$ be a test set, drawn from the same data/distribution as $D$.

Let $t \in D'$. As $T(t)$ we denote the class label supplied for $t$ by the classifier $T$.

As $\text{class}(t)$ we denote the actual class label of $t$.

As $D_{\text{true}}$ we denote the set of all test cases for which our classifier provides correct prediction:

$$D_{\text{true}} = \{ t \in D' | T(t) = \text{class}(t) \}$$

As $D_{\text{error}}$ we denote the set of all test cases for which our classifier provides incorrect prediction:

$$D_{\text{error}} = \{ t \in D' | T(t) \neq \text{class}(t) \}$$

**Accuracy.** The accuracy of the classifier $T$ is:

$$\text{accuracy}(T) = \frac{|D_{\text{true}}|}{|D'|}.$$ 

**Error rate.** The error rate of the classifier $T$ is:

$$\text{errorRate}(T) = 1 - \text{accuracy}(T) = \frac{|D_{\text{error}}|}{|D'|}.$$ 

Accuracy Measures for Binary Classification

**Binary Classifiers.** Many classifiers are binary: i.e., the class variable $C$ has only two values. A classification problem with $\text{dom}(C) = \{c_1, \ldots, c_k\}$, $k > 2$ can be transformed into $k$ classification problems with class variables $C_1, C_2, \ldots, C_k$, such that, $\text{dom}(C_i) = \{0, 1\}$. $C_i = 1$ means $C = c_i$. 
**Classification Errors.** Consider a binary classification problem with the class variable $C$, $\text{dom}(C) = \{0, 1\}$, where $C = 1$ is interpreted as "record belongs to class $C$" and $C = 0$ is interpreted as "record does not belong to class $C$.

Let $T$ be a classifier for $C$. Let $D'$ be a test dataset. Given $t \in D$, we can observe four possibilities:

1. **True Positive**: $T(t) = \text{class}(t) = 1$;
2. **True Negative**: $T(t) = \text{class}(t) = 0$;
3. **False Positive**: $T(t) = 1; \text{class}(t) = 0$;
4. **False Negative**: $T(t) = 0; \text{class}(t) = 1$;

There are **two types of errors of classification**:

1. **Type I error**: a.k.a. **error of commission** a.k.a. **false positive**: classifier incorrectly classifies a tuple as belonging to class $C$.
2. **Type II error**: a.k.a. **error of omission** a.k.a. **false negative**: classifier incorrectly classifies a tuple as NOT belonging to class $C$.

**Notation.** Consider the following notation:

1. $D_{TP}$: set of all **true positives** in $D'$; $TP = |D_{TP}|$;
2. $D_{TN}$: set of all **true negatives** in $D'$; $TN = |D_{TN}|$;
3. $D_{FP}$: set of all **false positives** in $D'$; $FP = |D_{FP}|$;
4. $D_{FN}$: set of all **false negatives** in $D'$; $FN = |D_{FN}|$;

**Confusion Matrix.** The information about the accuracy of a **binary classifier** is usually arranged in a form of **confusion matrix**:

<table>
<thead>
<tr>
<th>Actual positive</th>
<th>Classified Positive</th>
<th>Classified Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual positive</td>
<td>$TP$</td>
<td>$FN$</td>
</tr>
<tr>
<td>Actual negative</td>
<td>$FP$</td>
<td>$TN$</td>
</tr>
</tbody>
</table>

**Precision.** **Precision** of the classifier is the percentage of the correctly positively classified records in the set of all positively classified records:

$$\text{precision}(T) = \frac{TP}{TP + FP}.$$  

Precision measures *how accurately the classifier selects positive examples*, it reaches 100% when the classifier *admits no false positives*.

**Recall.** **Recall** of the classifier is the percentage of all correctly positively classified records in the set of all actual positive records:

$$\text{recall}(T) = \frac{TP}{TP + FN}.$$  

Recall measures *how successful the classifier is in correctly identifying all positive records*. It reaches 100% when the classifier *admits no false negatives*. 


**Note:** Precision and recall make sense only when combined together.

It is easy to build a classifier with 100% precision: $T(t) = 0$ for all $t \in D'$ guarantees that. **But this classifier will have recall of 0.** It is easy to build a classifier with 100% recall: $T(t) = 1$ for all $t \in D'$ guarantees that. **But this classifier will have small precision.**

**PF.** The PF measure is defined as:

$$PF(T) = \frac{FP}{FP + TN}.$$  

PF measures the *misclassification rate*: the percentage of records not in class $C$ that was incorrectly classified.

**F-measure.** The F-measure is the harmonic mean of precision and recall:

$$F(T) = \frac{2}{\frac{1}{\text{precision}(T)} + \frac{1}{\text{recall}(T)}} = \frac{2 \cdot \text{precision}(T) \cdot \text{recall}(T)}{\text{precision}(T) + \text{recall}(T)}.$$  

F-measure combines precision and recall into a single number by balancing them against each other.

In some situations, one of the two measures (precision or recall) is more important than the other. F-measure can be skewed to favor each. The $F_2$-measure below assumes recall is twice as valuable as precision. The $F_{0.5}$-measure below assumes precision is twice as valuable as recall.

$$F_2(T) = \frac{5 \cdot \text{precision}(T) \cdot \text{recall}(T)}{4 \cdot \text{precision}(T) + \text{recall}(T)}.$$  

$$F_{0.5}(T) = \frac{1.25 \cdot \text{precision}(T) \cdot \text{recall}(T)}{0.25 \cdot \text{precision}(T) + \text{recall}(T)}.$$  

The formula for $F_\beta$, where $\beta$ represents the relative importance of recall over precision is:

$$F_\beta(T) = \frac{(1 + \beta^2) \cdot \text{precision}(T) \cdot \text{recall}(T)}{\beta^2 \cdot \text{precision}(T) + \text{recall}(T)}.$$  

**Evaluation Techniques**

In a typical situation, you are given a **training set** $D$, and are asked to produce a classifier for it.

If all records from $D$ are used to create a classifier, there will be no way to independently test its accuracy.
**Holdout set.** Divide $D$ into two sets: $D = D_{train} \cup D_{test}; D_{train} \cap D_{test} = \emptyset$. $D_{test}$ is called the **holdout set**.

Create a classifier $T$ using $D_{train}$ as the training set. **Test** $T$ using $D_{test}$.

**Holdout set** selection:

- **Random sampling.** Select a fraction $x$. Randomly sample $x\%$ of records from $D$, put them in $D_{test}$.

  Commonly, you use around 90% of $D$ as the training set, reserving the remaining 10% for the holdout set.

- **Time slices.** If $D$ consists of "old" data and "new" data, then, the training set can include all of the "old" data, while the holdout set can include the "new" data. (e.g., in situations where new records appear every day).

**Multiple random sampling.** This technique is used when $D$ is small.

- Select some number $M$ of repetitions.

- Perform $M$ random samplings of a **holdout set** from $D$. Run classifier construction on the remaining set $D_{train}$. Compute the **accuracy** of the classifier for the current sample.

- Compute the final accuracy as the mean accuracy over all samples.

**Multiple random sampling** allows us to avoid **flukes** (or, at least, to downgrade their effects).

**Cross-Validation.** This is a variant of multiple random sampling that uses only one random assignment of records, but performs multiple classifications.

- Select $n$ – the number of **slices** of data in $D$.

- Using **random sampling** split $D$ into $n$ slices of equal (or almost equal) size.

- Perform $n$ classification procedures. On step $i$, use slice $D_i$ as the **holdout set**, while using all other $n - 1$ slices as the **training set**.

**Note:** Standard cross-validations used in practice are 10-fold, 5-fold and leave-one-out cross-validations.

**References**