Part I

Classification & Decision Trees
Classification

Problem description

- We are given a data matrix $\mathbf{X}$ with either continuous or discrete variables such that each row $X_i \in \mathcal{F}$ and a set of labels $\mathbf{Y} \in \mathcal{L}$.
- For a $k$-class problem, $\#\mathcal{L} = k$ and we can think of $\mathcal{L} = \{1, \ldots, k\}$.
- Our goal is to find a classifier

$$f : \mathcal{F} \rightarrow \mathcal{L}$$

that allows us to predict the label of a new observation given a new set of features.
Classification

A supervised problem

- Classification is a supervised problem.
- Usually, we use a subset of the data, the *training set* to learn or estimate the classifier yielding $\hat{f} = \hat{f}_{\text{training}}$.
- The performance of $\hat{f}$ is measured by applying it to each case in the *test set* and computing

$$\sum_{j \in \text{test}} L(\hat{f}_{\text{training}}(X_j), Y_j)$$
Classification

<table>
<thead>
<tr>
<th>Tid</th>
<th>Attrb1</th>
<th>Attrb2</th>
<th>Attrb3</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Yes</td>
<td>Large</td>
<td>125K</td>
<td>No</td>
</tr>
<tr>
<td>2</td>
<td>No</td>
<td>Medium</td>
<td>100K</td>
<td>No</td>
</tr>
<tr>
<td>3</td>
<td>No</td>
<td>Small</td>
<td>70K</td>
<td>No</td>
</tr>
<tr>
<td>4</td>
<td>Yes</td>
<td>Medium</td>
<td>120K</td>
<td>No</td>
</tr>
<tr>
<td>5</td>
<td>No</td>
<td>Large</td>
<td>95K</td>
<td>Yes</td>
</tr>
<tr>
<td>6</td>
<td>No</td>
<td>Medium</td>
<td>85K</td>
<td>No</td>
</tr>
<tr>
<td>7</td>
<td>Yes</td>
<td>Large</td>
<td>220K</td>
<td>No</td>
</tr>
<tr>
<td>8</td>
<td>No</td>
<td>Small</td>
<td>85K</td>
<td>Yes</td>
</tr>
<tr>
<td>9</td>
<td>No</td>
<td>Medium</td>
<td>75K</td>
<td>No</td>
</tr>
<tr>
<td>10</td>
<td>No</td>
<td>Small</td>
<td>95K</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Training Set

<table>
<thead>
<tr>
<th>Tid</th>
<th>Attrb1</th>
<th>Attrb2</th>
<th>Attrb3</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>No</td>
<td>Small</td>
<td>65K</td>
<td>?</td>
</tr>
<tr>
<td>12</td>
<td>Yes</td>
<td>Medium</td>
<td>80K</td>
<td>?</td>
</tr>
<tr>
<td>13</td>
<td>Yes</td>
<td>Large</td>
<td>110K</td>
<td>?</td>
</tr>
<tr>
<td>14</td>
<td>No</td>
<td>Small</td>
<td>95K</td>
<td>?</td>
</tr>
<tr>
<td>15</td>
<td>No</td>
<td>Large</td>
<td>67K</td>
<td>?</td>
</tr>
</tbody>
</table>

Test Set
Classification

Examples of classification tasks

- Predicting whether a tumor is benign or malignant.
- Classifying credit card transactions as fraudulent or legitimate.
- Predicting the type of a given tumor among several types.
- Categorizing a document or news story as one of \{finance, weather, sports, etc.\}
Classification

Common techniques

- Decision Tree based Methods
- Rule-based Methods
- Discriminant Analysis
- Memory based reasoning
- Neural Networks
- Naïve Bayes
- Support Vector Machines
Classification trees

Training Data

Model: Decision Tree
Classification trees

There could be more than one tree that fits the same data!
Applying a decision tree rule

Training Set

<table>
<thead>
<tr>
<th>Tid</th>
<th>Attrib1</th>
<th>Attrib2</th>
<th>Attrib3</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Yes</td>
<td>Large</td>
<td>126K</td>
<td>No</td>
</tr>
<tr>
<td>2</td>
<td>No</td>
<td>Medium</td>
<td>100K</td>
<td>No</td>
</tr>
<tr>
<td>3</td>
<td>No</td>
<td>Small</td>
<td>70K</td>
<td>No</td>
</tr>
<tr>
<td>4</td>
<td>Yes</td>
<td>Medium</td>
<td>120K</td>
<td>No</td>
</tr>
<tr>
<td>5</td>
<td>No</td>
<td>Large</td>
<td>95K</td>
<td>Yes</td>
</tr>
<tr>
<td>6</td>
<td>No</td>
<td>Medium</td>
<td>80K</td>
<td>No</td>
</tr>
<tr>
<td>7</td>
<td>Yes</td>
<td>Large</td>
<td>220K</td>
<td>No</td>
</tr>
<tr>
<td>8</td>
<td>No</td>
<td>Small</td>
<td>85K</td>
<td>Yes</td>
</tr>
<tr>
<td>9</td>
<td>No</td>
<td>Medium</td>
<td>75K</td>
<td>No</td>
</tr>
<tr>
<td>10</td>
<td>No</td>
<td>Small</td>
<td>90K</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Test Set

<table>
<thead>
<tr>
<th>Tid</th>
<th>Attrib1</th>
<th>Attrib2</th>
<th>Attrib3</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>No</td>
<td>Small</td>
<td>55K</td>
<td>?</td>
</tr>
<tr>
<td>12</td>
<td>Yes</td>
<td>Medium</td>
<td>85K</td>
<td>?</td>
</tr>
<tr>
<td>13</td>
<td>Yes</td>
<td>Large</td>
<td>110K</td>
<td>?</td>
</tr>
<tr>
<td>14</td>
<td>No</td>
<td>Small</td>
<td>95K</td>
<td>?</td>
</tr>
<tr>
<td>15</td>
<td>No</td>
<td>Large</td>
<td>87K</td>
<td>?</td>
</tr>
</tbody>
</table>
Applying a decision tree rule

Start from the root of tree.

- **Refund**
  - Yes: NO
  - No: **MarSt**
    - Single, Divorced: **TaxInc**
      - $\leq 80K$: NO
      - $> 80K$: YES
    - Married: NO

**Test Data**

<table>
<thead>
<tr>
<th>Refund</th>
<th>Marital Status</th>
<th>Taxable Income</th>
<th>Cheat</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>Married</td>
<td>80K</td>
<td>?</td>
</tr>
</tbody>
</table>
Applying a decision tree rule

Test Data

<table>
<thead>
<tr>
<th>Refund</th>
<th>Marital Status</th>
<th>Taxable Income</th>
<th>Cheat</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>Married</td>
<td>80K</td>
<td>?</td>
</tr>
</tbody>
</table>

Refund

Yes

NO

No

MarSt

Single, Divorced

Married

TaxInc

< 80K

NO

> 80K

NO

YES
Applying a decision tree rule

Test Data

Refund | Marital Status | Taxable Income | Cheat
---|---|---|---
No | Married | 80K | ?

Refund

Yes

NO

MarSt

Single, Divorced

TaxInc

< 80K

NO

> 80K

YES

NO

Married

NO
Applying a decision tree rule
Applying a decision tree rule
Applying a decision tree rule

Test Data

<table>
<thead>
<tr>
<th>Refund</th>
<th>Marital Status</th>
<th>Taxable Income</th>
<th>Cheat</th>
</tr>
</thead>
<tbody>
<tr>
<td>No</td>
<td>Married</td>
<td>80K</td>
<td>?</td>
</tr>
</tbody>
</table>

Assign Cheat to “No”
Decision boundary for tree
Decision tree for iris data using all features

- Petal length < 2.45
  - Iris-setosa

- Petal width < 1.75
  - Petal length < 4.95
    - Sepal length < 5.15
      - Iris-versicolor
    - Iris-versicolor
  - Iris-virginica
Decision tree for iris data using petal.length, petal.width
Regions in petal.length, petal.width plane
Figure: Trees have trouble capturing structure not parallel to axes
Learning the tree

Training Set

<table>
<thead>
<tr>
<th>Tid</th>
<th>Attr1</th>
<th>Attr2</th>
<th>Attr3</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Yes</td>
<td>Large</td>
<td>125K</td>
<td>No</td>
</tr>
<tr>
<td>2</td>
<td>No</td>
<td>Medium</td>
<td>100K</td>
<td>No</td>
</tr>
<tr>
<td>3</td>
<td>No</td>
<td>Small</td>
<td>70K</td>
<td>No</td>
</tr>
<tr>
<td>4</td>
<td>Yes</td>
<td>Medium</td>
<td>120K</td>
<td>No</td>
</tr>
<tr>
<td>5</td>
<td>No</td>
<td>Large</td>
<td>69K</td>
<td>Yes</td>
</tr>
<tr>
<td>6</td>
<td>No</td>
<td>Medium</td>
<td>60K</td>
<td>No</td>
</tr>
<tr>
<td>7</td>
<td>Yes</td>
<td>Large</td>
<td>220K</td>
<td>No</td>
</tr>
<tr>
<td>8</td>
<td>No</td>
<td>Small</td>
<td>85K</td>
<td>Yes</td>
</tr>
<tr>
<td>9</td>
<td>No</td>
<td>Medium</td>
<td>75K</td>
<td>No</td>
</tr>
<tr>
<td>10</td>
<td>No</td>
<td>Small</td>
<td>90K</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Test Set

<table>
<thead>
<tr>
<th>Tid</th>
<th>Attr1</th>
<th>Attr2</th>
<th>Attr3</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>No</td>
<td>Small</td>
<td>65K</td>
<td>?</td>
</tr>
<tr>
<td>12</td>
<td>Yes</td>
<td>Medium</td>
<td>90K</td>
<td>?</td>
</tr>
<tr>
<td>13</td>
<td>Yes</td>
<td>Large</td>
<td>110K</td>
<td>?</td>
</tr>
<tr>
<td>14</td>
<td>No</td>
<td>Small</td>
<td>95K</td>
<td>?</td>
</tr>
<tr>
<td>15</td>
<td>No</td>
<td>Large</td>
<td>67K</td>
<td>?</td>
</tr>
</tbody>
</table>

Tree Induction algorithm

Induction

Learn Model

Model

Decision Tree

Apply Model

Deduction
Learning the tree

Hunt’s algorithm (generic structure)

- Let $D_t$ be the set of training records that reach a node $t$
- If $D_t$ contains records that belong to the same class $y_t$, then $t$ is a leaf node labeled as $y_t$.
- If $D_t = \emptyset$, then $t$ is a leaf node labeled by the default class, $y_d$.
- 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.
- This splitting procedure is what can vary for different tree learning algorithms . . .
Learning the tree

Statistics 202: Data Mining

© Jonathan Taylor
Learning the tree

**Issues**

**Greedy strategy:** Split the records based on an attribute test that optimizes certain criterion.

**What is the best split:** What criterion do we use? Previous example chose to split on Refund.

**How to split the records:** Binary or multi-way? Previous example split Taxable Income at $\geq 80K$.

**When do we stop?** Should we continue until each node is completely homogeneous?
Different splits: ordinal / nominal

Figure: Binary or multi-way?
Different splits: continuous

Figure: Binary or multi-way?
Choosing a variable to split on

Figure: Which should we start the splitting on?
Learning the tree

Choosing the best split

- Need some numerical criterion to choose among possible splits.
- Criterion should favor *homogeneous or pure* nodes.
- Common cost functions:
  - Gini Index
  - Entropy / Deviance / Information
  - Misclassification Error
Choosing a variable to split on

Gain = M0 – M12 vs M0 – M34
Learning the tree

GINI Index

- Suppose we have $k$ classes and node $t$ has frequencies $p_t = (p_{1,t}, \ldots, p_{k,t})$.
- Criterion

$$GINI(t) = \sum_{(j,j') \in \{1,\ldots,k\}: j \neq j'} p_{j,t} p_{j',t} = 1 - \sum_{j=1}^{l} p_{j,t}^2.$$

- Maximized when $p_{j,t} = 1/k$ with value $1 - 1/k$
- Minimized when all records belong to a single class.
- Minimizing $GINI$ will favour pure nodes . . .
Learning the tree

**Gain in GINI Index for a potential split**

- Suppose $t$ is to be split into $j$ new child nodes $(t_l)_{1 \leq l \leq j}$.
- Each child node has a count $n_l$ and a vector of frequencies $(p_{1,t_l}, \ldots, p_{k,t_l})$. Hence they have their own GINI index, $\text{GINI}(t_l)$.
- The gain in GINI Index for this split is

$$\text{Gain}(\text{GINI}, t \rightarrow (t_l)_{1 \leq l \leq j}) = \text{GINI}(t) - \frac{\sum_{l=1}^{j} n_l \text{GINI}(t_l)}{\sum_{l=1}^{j} n_l}.$$ 

- Greedy algorithm chooses the biggest gain in GINI index among a list of possible splits.
Decision tree for iris data using all features with GINI
Learning the tree

Entropy / Deviance / Information

- Suppose we have $k$ classes and node $t$ has frequencies $p_t = (p_{1,t}, \ldots, p_{k,t})$.
- Criterion
  \[
  H(t) = - \sum_{j=1}^{k} p_{j,t} \log p_{j,t}
  \]
- Maximized when $p_{i,t} = 1/k$ with value $\log k$
- Minimized when one class has no records in it.
- Minimizing entropy will favour pure nodes . . .
Decision tree for iris data using all features with Entropy
Learning the tree

**Gain in entropy for a potential split**

- Suppose $t$ is to be split into $j$ new child nodes $(t_l)_{1 \leq l \leq j}$.
- Each child node has a count $n_l$ and a vector of frequencies $(p_{1,t_l}, \ldots, p_{k,t_l})$. Hence they have their own entropy $H(t_l)$.
- The gain in entropy for this split is

$$
\text{Gain}(H, t \rightarrow (t_l)_{1 \leq l \leq j}) = H(t) - \frac{\sum_{l=1}^{j} n_l H(t_l)}{\sum_{l=1}^{j} n_l}.
$$

- Greedy algorithm chooses the biggest gain in $H$ among a list of possible splits.
Learning the tree

Misclassification Error

- Suppose we have $k$ classes and node $t$ has frequencies $p_t = (p_{1,t}, \ldots, p_{k,t})$.
- The mode is
  $$\hat{k}(t) = \arg\max_k p_{k,t}.$$  
- Criterion
  $$\text{Misclassification Error}(t) = 1 - p_{\hat{k}(t),t}$$
- Not smooth in $p_t$ as $GINI, H$, can be more difficult to optimize numerically.
Different criteria: \( GINI, H, MC \)
Learning the tree

Misclassification Error

- Example: suppose parent has 10 cases: \( \{7D, 3R\} \)
- A candidate split produces two nodes: \( \{3D, 0R\} \), \( \{4D, 3R\} \).
- The gain in MC is 0, but gain in GINI is \( 0.42 - 0.342 > 0 \).
- Similarly, entropy will also show an improvement . . .
Choosing the split for a continuous variable

<table>
<thead>
<tr>
<th>Cheat</th>
<th>No</th>
<th>No</th>
<th>No</th>
<th>Yes</th>
<th>Yes</th>
<th>Yes</th>
<th>Yes</th>
<th>No</th>
<th>No</th>
<th>No</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Taxable Income</th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>55</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Yes</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>No</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Gini</td>
</tr>
</tbody>
</table>
Learning the tree

**Stopping training**

- As trees get deeper, or if splits are multi-way the number of data points per leaf node drops very quickly.
- Trees that are too deep tend to overfit the data.
- A common strategy is to “prune” the tree by removing some internal nodes.
Learning the tree

Figure: Underfitting corresponds to the left-hand side, overfit to the right
Learning the tree

Cost-complexity pruning (tree library)

- Given a criterion $Q$ like $H$ or $GINI$, we define the cost-complexity of a tree with terminal nodes $(t_j)_{1 \leq j \leq m}$

$$C_\alpha(T) = \sum_{j=1}^{m} n_j Q(t_j) + \alpha m$$

- Given a large tree $T_L$ we might compute $C_\alpha(T)$ for any subtree $T$ of $T_L$.
- The optimal tree is defined as

$$\hat{T}_\alpha = \arg\min_{T \leq T_L} C_\alpha(T).$$

- Can be found by “weakest-link” pruning. See *Elements of Statistical Learning* for more . . .
Learning the tree

Pre-pruning (**rpart library**)

- These methods stop the algorithm before it becomes a fully-grown tree.

- Examples
  - Stop if all instances belong to the same class (kind of obvious).
  - Stop if number of instances is less than some user-specified threshold. Both `tree`, `rpart` have rules like this.
  - Stop if class distribution of instances are independent of the available features (e.g., using $\chi^2$ test)
  - Stop if expanding the current node does not improve impurity measures (e.g., Gini or information gain). This relates to `cp` in `rpart`. 
Training and test error as a function of $cp$
Evaluating a classifier

### Metrics for Performance Evaluation

The most widely-used metric is the confusion matrix, which compares the predicted class against the actual class. The confusion matrix is a 2x2 table where:

- `a` (TP): True Positive, where the prediction is Yes and the actual class is Yes.
- `b` (FN): False Negative, where the prediction is No and the actual class is Yes.
- `c` (FP): False Positive, where the prediction is Yes and the actual class is No.
- `d` (TN): True Negative, where the prediction is No and the actual class is No.

The table below summarizes the confusion matrix:

<table>
<thead>
<tr>
<th>ACTUAL CLASS</th>
<th>PREDICTED CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class=Yes</td>
<td>a (TP)</td>
</tr>
<tr>
<td>Class=No</td>
<td>c (FP)</td>
</tr>
</tbody>
</table>
Evaluating a classifier

Measures of performance

- Simplest is accuracy

\[
\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = \text{SMC(Actual, Predicted)} = 1 - \text{Misclassification Rate}
\]
Evaluating a classifier

Accuracy isn’t everything

- Consider an unbalanced 2-class problem with \# 1’s=10, \# 0’s=9990.
- Simply labelling everything 0 yields 99.9% accuracy.
- But, this classifier misses all class 1.
Evaluating a classifier

<table>
<thead>
<tr>
<th>ACTUAL CLASS</th>
<th>PREDICTED CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class=Yes</td>
<td>C(Yes</td>
</tr>
<tr>
<td>Class=No</td>
<td>C(Yes</td>
</tr>
</tbody>
</table>

$C(i|j)$: Cost of misclassifying class $j$ example as class $i$
Learning the tree

Measures of performance

- Classification rule changes to

\[
\text{Label}(p, C) = \arg\min_i \sum_j C(i|j)p_j
\]

- Accuracy is the same as cost if \( C(Y|Y) = C(N|N) = c_1, \)
\( C(Y|N) = C(N|Y) = c_2. \)
Evaluating a classifier

<table>
<thead>
<tr>
<th>ACTUAL CLASS</th>
<th>PREDICTED CLASS</th>
<th>Model $M_1$</th>
<th>PREDICTED CLASS</th>
<th>Model $M_2$</th>
<th>PREDICTED CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>

\[
\text{Accuracy} = 80\%
\]
\[
\text{Cost} = 3910
\]

\[
\text{Accuracy} = 90\%
\]
\[
\text{Cost} = 4255
\]
Evaluating a classifier

Measures of performance

- Other common ones

\[ \text{Precision} = \frac{TP}{TP + FP} \]

\[ \text{Specificity} = \frac{TN}{TN + FP} = TNR \]

\[ \text{Sensitivity} = \text{Recall} = \frac{TP}{TP + FN} = TPR \]

\[ F = \frac{2 \cdot \text{Recall} \cdot \text{Precision}}{\text{Recall} + \text{Precision}} \]

\[ = \frac{2 \cdot TP}{2 \cdot TP + FN + FP} \]
Evaluating a classifier

Measures of performance

- Precision emphasizes $P(p = Y, a = Y) \& P(p = Y, a = N)$.
- Recall emphasizes $P(p = Y, a = Y) \& P(p = N, a = Y)$.
- $FPR = 1 - TNR$
- $FNR = 1 - TPR$. 
Evaluating a classifier

Measure of performance

- We have done some simple training / test splits to see how well our classifier is doing.
- More accurately, this procedure measures how well our algorithm for *learning the classifier* is doing.
- How well this works may depend on
  - **Model:** Are we using the right type of classifier model?
  - **Cost:** Is our algorithm sensitive to the cost of misclassification?
  - **Data size:** Do we have enough data to learn a model?
Evaluating a classifier

Figure: As data increases, our estimate of accuracy improves, as does the variability of our estimate...
Evaluating a classifier

Estimating performance

Holdout: Split into test and training (e.g. 1/3 test, 2/3 training).

Random subsampling: Repeated replicates of holdout, averaging results.

Cross validation: Partition data into $K$ disjoint subsets. For each subset $S_i$, train on all but $S_i$, then test on $S_i$.

Stratified sampling: May be helpful to sample so Y/N class is roughly equal in training data.

0.632 Bootstrap: Combine training error and bootstrap error...