Notes for CSE6740 Lecture 7: Feature Selection and Decision Tree

Fred Yang (GTID: 904140528)
fred.yang@gatech.edu

CONTENTS

1 SUPERVISED LEARNING

In ***Unsupervised Learning***, we are given input features $\mathbf{x} \in \mathbb{R}^d$ but no labels. The goal of unsupervised learning is discover the hidden patterns or structures in the data, such as **Clustering** and **Dimension Reduction**, which were discussed in previous lessons. The model learns to make sense of the data without being told what the "right answer" is.

Oppositely, in ***Supervised Learning***, we are given both input features $\mathbf{x} \in \mathbb{R}^d$ and labels $\mathbf{y}$ (aka the right answer). The goal turns to predict y for a new feature x from the original mapping relationship $f\ )\ X \rightarrow Y$. **Classification Problem** is one of the representative examples for supervised learning:

- Given a dataset $D = \{(x^1, y^1), (x^2, y^2), \ \ldots, \ (x^n, y^n)\}, x \in \mathbb{R}^d, y \in \{1, 2, \ldots, K\}$.
- Given a new feature x, infer the corresponding y from the function mapping $f\ )\ X \rightarrow Y$.

2 FEATURE SELECTION

Instead of using all available features (which may include irrelevant, redundant, or noisy ones), people try to identify the most informative features. Considering this, we often utilized **feature selection**, which is the process of choosing a subset of relevant features from the dataset that contribute the most to predicting the output (label), to help the model learn better, run faster and generalize more effectively. Two examples for feature selection are shown in Figure 1.

People are interested in features, and we want to know which are relevant. Our model should be interpretable. Basically, these are the reasons for using feature selection.

2.1 Information Gain

2.1.1 *Informativeness of a Feature*

Before we observe any features, we have some uncertainty about the label Y. For example, if we're predicting whether an email is spam, and we haven't seen any words yet, we might say we're 50/50 unsure. This uncertainty can be quantified using **entropy**, written as $H(Y)$.

Once we look at a particular feature X_i (say, whether the word "Viagra" appears in the email), our uncertainty about Y may decrease. This updated uncertainty is measured by conditional entropy, $H(Y \mid X_i)$.

The **informativeness of the feature** is how much the uncertainty is reduced. In information theory, this is called mutual information, and it's calculated as:

$$I(X_i; Y) = H(Y) - H(Y \mid X_i)$$

In other words, the more a feature reduces our uncertainty, the more useful it is for classification.

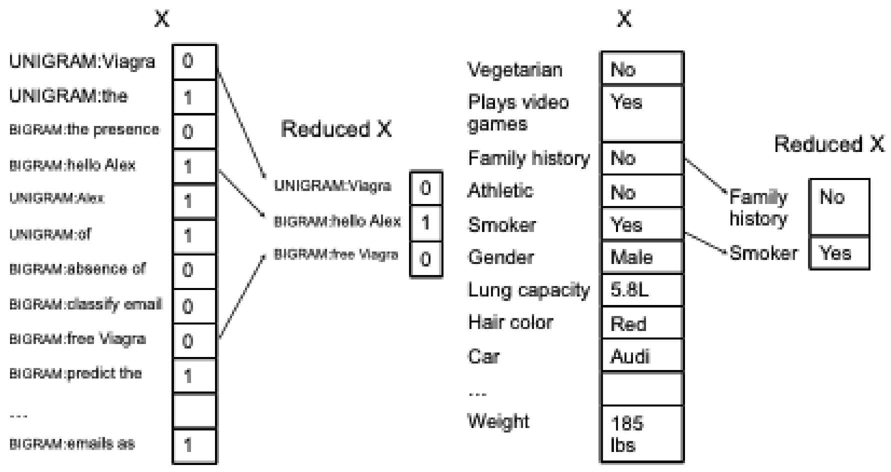

Figure 1—Examples of feature selection in supervised learning: On the left, an email classification task reduces a large set of word presence features (unigrams and bigrams) to a smaller subset of informative terms; On the right, predicting lung disease risk from a medical survey reduces many personal and lifestyle attributes to only the most relevant features (e.g., family history, smoking status).

2.1.2 *Entropy: Quantify Uncertainty*

Now, let's make this precise: we use **entropy** to quantify uncertainty. Entropy H(Y) is the **expected number of bits** needed to encode the outcome of Y, using the most efficient code. If something is very uncertain, it takes more bits to describe. If it's predictable, it takes fewer bits. This gives us a clean, mathematical way to measure uncertainty.

For a discrete random variable Y, entropy is defined as:

$$H(Y) = -\sum_{k=1}^{K} P(y=k)\,\log_2 P(y=k)$$

This is called **Shannon entropy**.

For continuous random variables, we replace the sum with an integral:

$$H(Y) = \mathbb{E}_y[-\log_2 P(y)] = -\int_{y \in D} P(y)\,\log_2 P(y)\,dy$$

This is known as **differential entropy**.

Per the Information Theory, the "most efficient code" to represent an outcome

$Y = k$ requires $-\log_2 P(Y = k)$ bits, which is the only expression that satisfies the natural requirements for an "information measure". It has the right properties:

- If $P(Y = k) = 1$ (certain event), then $-\log_2(1) = 0$ bits (no info needed).
- If $P(Y = k) = 0.5$, then $-\log_2(0.5) = 1$ bit (like a coin flip).
- If $P(Y = k) = 0.25$, then $-\log_2(0.25) = 2$ bits (you need 2 yes/no questions to identify it).

In coding, think of a code as a sequence of yes/no questions:

- If there are 4 equally likely outcomes, you need 2 bits (00, 01, 10, 11).
- If one outcome is much more likely than others, you can design a shorter code for that outcome and longer codes for rare ones (this is what Huffman coding does).

So the entropy is the average code length across all possible values of Y.

Intuitively, high entropy represents more randomness, more surprise and less predictability:

- Y is from a uniform like distribution.
- Flat histogram.
- Values sampled from it are less predictable.

while low entropy represents more order, less surprise, more predictability:

- Y is from a varied (peaks and valleys) distribution.
- Histogram has many lows and highs.
- Values sampled from it are more predictable.

2.1.3 *Examples for Computing Entropy*

A concrete example is coin-flip. Imagine you flip a coin 6 times and count how many are heads vs. tails. The entro

2.1.4 *Conditional Entropy*

Conditional entropy quantifies how much uncertainty remains in Y after observing X_i:

$$H(Y \mid X) = -\int_{x \in D_X} \left(\sum_{k=1}^{K} P(y = k \mid x) \log_2 P(y = k \mid x) \right) p(x) dx \quad \text{(Discrete Case)}$$

Table 1—Step-by-step entropy calculations for different head–tail distributions out of 6 coin flips.

Heads	**Tails**	$P(\text{head}), P(\text{tail})$	**Calculation**	$H(S)$
0	6	$0,\ 1$	$-0\log_2 0 - 1\log_2 1$	0
1	5	$\frac{1}{6},\ \frac{5}{6}$	$-\frac{1}{6}\log_2\frac{1}{6} - \frac{5}{6}\log_2\frac{5}{6}$	0.65
2	4	$\frac{2}{6},\ \frac{4}{6}$	$-\frac{2}{6}\log_2\frac{2}{6} - \frac{4}{6}\log_2\frac{4}{6}$	0.92

$$H(Y \mid X) = -\int_{x\in D_X}\left(\int_{y\in D_Y} P(y \mid x)\log_2 P(y \mid x)\,dy\right)p(x)\,dx \quad \text{(Continuous Case)}$$

$H(Y \mid X)$ is the expected number of bits needed to encode a randomly drawn value of Y given X_i, and average over the likelihood of seeing particular value of X_i.

Table 2 gives a worked-out example of conditional/joint entropy.

Table 2—Joint probability distribution $P(X,Y)$.

$X\backslash Y$	0	1
0	$\frac{1}{4}$	$\frac{1}{4}$
1	$\frac{1}{2}$	0

The joint entropy of X and Y is defined as:

$$H(X,Y) = -\sum_{x,y} p(x,y)\log_2 p(x,y)$$

This measures the total uncertainty when we consider the random pair $Z = (X,Y)$ together:

$$H(X,Y) = -\left(\tfrac{1}{4}\log_2\tfrac{1}{4} + \tfrac{1}{4}\log_2\tfrac{1}{4} + \tfrac{1}{2}\log_2\tfrac{1}{2}\right) = 1.5$$

The conditional entropy $H(Y \mid X)$:

$$H(Y \mid X) = \mathbb{E}_x[H(Y \mid X = x)] = \frac{1}{2}H(Y \mid X = 0) + \frac{1}{2}H(Y \mid X = 1) = \frac{1}{2}H(\frac{1}{2},\frac{1}{2}) + \frac{1}{2}H(1,0)$$
$$= \frac{1}{2}$$

The conditional entropy $H(X \mid Y)$:

$$H(X \mid Y) = \mathbb{E}_y[H(X \mid Y = x)] = \frac{3}{4}H(X \mid Y = 0) + \frac{1}{4}H(X \mid Y = 1) = \frac{3}{4}H(\frac{2}{3}, \frac{1}{3}) + \frac{1}{4}H(1, 0)$$
$$= 0.6887 \neq H(Y \mid X)$$

Mutual information is the formal way to measure how much information one random variable gives us about another. In other words, mutual information is **the reduction in uncertainty** of Y after seeing feature X_i:

$$I(X_i, Y) = H(Y) - H(Y \mid X_i)$$

The more the reduction in entropy, the more information a feature contains. Mutual information has a very nice symmetry property:

$$I(X_i; Y) = I(Y; X_i)$$

This means the amount of information X provides about Y is exactly the same as the amount of information Y provides about X. Contrast this with conditional entropy, where $H(Y|X) \neq H(X|Y)$ (remember that $H(Y|X)$ represents **the remaining uncertainty** about Y after knowing X).

2.2 Feature Selection Algorithm

- Given a dataset $D = \{(x^1, y^1), (x^2, y^2), \ldots, (x^n, y^n)\}, \quad x \in \mathbb{R}^d,\ y \in \{1, 2, \ldots, K\}$
- Estimate density $p(y = k)$ for each value of the label $y = k$.
- For the i-th feature entry x_i (e.g. attendance) where $i \in \{1, 2, \ldots, d\}$
  - Estimate its density $p(x_i)$ using density estimation
  - For each value of the label $y = k$
    - Estimate the density $p(x_i \mid y = k)$

- Score feature x_i using:

$$
\begin{aligned}
I_i &= I(Y; X_i) \\
&= H(X_i) - H(X_i \mid Y) \\
&= -\int p(x_i) \log_2 p(x_i) dx_i + \sum_y \int p(x_i \mid y = k) \log_2 p(x_i \mid y = k) p(y = k) dx_i \\
&= \sum_y \int p(x_i, y = k) \log_2 \frac{p(x_i, y = k)}{p(y = k)} dx_i - \int p(x_i) \log_2 p(x_i) dx_i \\
&= \sum_y \int p(x_i, y = k) \log_2 \frac{p(x_i, y = k)}{p(y = k)} dx_i - \sum_y \int p(x_i, y = k) \log_2 p(x_i) dx_i \\
&= \sum_{k=1}^{K} \int p(x_i, y = k) \log_2 \frac{p(x_i, y = k)}{p(y = k) p(x_i)} dx_i \\
&= \int \sum_{k=1}^{K} p(x_i \mid y = k) p(y = k) \log_2 \frac{p(x_i \mid y = k)}{p(x_i)} dx_i
\end{aligned}
$$

- Choose those feature x_i with high score I_i.

3 DECISION TREE

A **decision tree** is a supervised learning model used for classification (and sometimes regression). It works by asking a sequence of questions about the input features, then following the answers down a tree until a final class label is assigned.

3.1 How Decision Tree Works

Figure 2 illustrates the structure of an canonical decision tree. Each internal node tests an attribute/feature x_i, and one branch for each possible attribute value (e.g. $x_i \le v$ or $x_i > v$). Each leaf assigns a class y. To classify input x, traverse the tree from the root to leaf, output the labeled y.

Decision trees are among the most intuitive and interpretable models in machine learning nevertheless constructing them is not as straightforward as it may first appear. In fact, the task of learning the smallest possible decision tree that perfectly classifies a dataset has been proven to be **NP-complete** Hyafil and Rivest, 1976. This result implies that, in the general case, no efficient algorithm exists that can guarantee finding the globally optimal tree in polynomial time. Exhaustively searching through all possible tree structures is computationally infeasible, as the number of candidate trees grows exponentially with the number of features

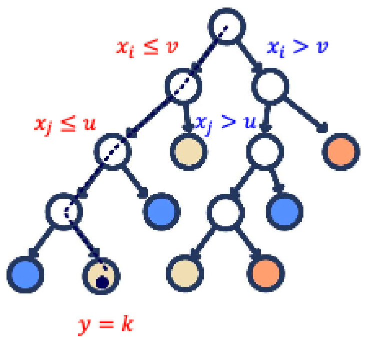

Figure 2—Example of a decision tree for classification. Each internal node tests a feature value (e.g., $x_i \leq v$ or $x_j \leq u$), and each branch corresponds to the outcome of that test. Leaves assign class labels $y = k$. A classification for a new input is made by traversing from the root to a leaf following the test outcomes.

and data points.

Given this complexity, practical decision tree learning algorithms turn to heuristics rather than exact optimization. A widely used strategy is a greedy recursive algorithm:

- Start from an empty decision tree
- Split on next best **attribute (feature)**
- Recurse

A good example shows how a decision tree can be built is predicting Atlanta apartment rental prices from housing features (see Figure 3):

- At the root node, the tree first splits on the feature X_i, representing the number of bedrooms. The dataset is divided into three subsets: apartments with 0, 1, or 2 bedrooms ($D_{(X_i=0)}, D_{(X_i=1)}, D_{(X_i=2)}$).
- For the subset with $X_i = 2$ (two bedrooms), the tree further splits on another feature X_j, the number of bathrooms. This creates additional branches for apartments with 0, 1, or 2 bathrooms.
- At the leaves, the tree outputs a predicted rental price. For example, one leaf assigns a rental price of $Y = 2000$.

Here are the canonical progresses for learning decision trees:

- **Initialization**: start from empty decision tree with dataset D

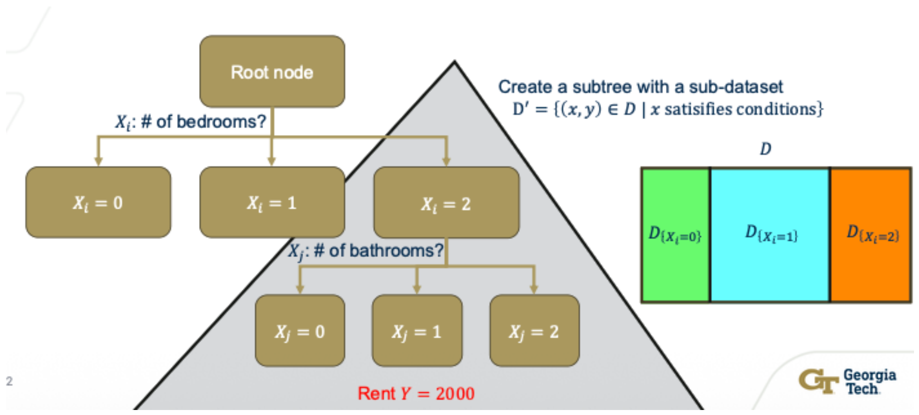

Figure 3—Decision Tree Example: predicting Atlanta apartment rental prices from housing features

- Based on D, compute information gain of each attribute (feature) X_i:

$$I(X_i, Y) = H(Y) - H(Y \mid X_i)$$

- Pick **the best attribute (feature)**:

$$i\ ) = \arg\max_i I(X_i; Y) = \arg\max_i H(Y) - H(Y \mid X_i)$$

- Split the tree and the dataset at node X_i
- Create subtrees and recurse through all subtrees

However, the odds are some features are numeric/continuous in a real word setting(e.g., room size, distance to school). It cannot be split naively. Decision trees need heuristics (like **threshold splits**) to avoid infinite branching and overfitting.

3.2 Threshold Splits

Instead of branching on every possible numeric value, use a binary split at a threshold t: one brach $X_i < t$ and other brach $X_i \geq t$. This method allows repeated splits on **same variable** along a path.

When dealing with continuous features in decision trees, the challenge is choosing the right threshold t for splitting. Although there are infinitely many possible values, *only a finite number of t's are important*. The standard approach is to sort the feature values in the dataset and evaluate splits between consecutive distinct values. Each candidate threshold is tested by computing information gain (or another criterion), and the one that maximizes the separation of classes is chosen. In this way, threshold selection becomes computationally feasible while still capturing the most meaningful splits.

3.3 Decision Tree Algorithm

Considering the threshold splits, we can now update the previous process of learning decision trees:

- **Initialization**: start from empty decision tree with dataset D
- Based on D, compute information gain of **each attribute (feature)** X_i **and split t**:

$$I(X_i > t, Y) = H(Y) - H(Y \mid X_i > t)$$

- Pick **the best attribute (feature**) and **split** t:

$$i, t\ ) = \arg\max_{i,t} I(X_i > t; Y) = \arg\max_{i,t} H(Y) - H(Y \mid X_i > t)$$

- Split the tree and the dataset at node $\mathbf{X_i > t}$
- Create subtrees and recurse through all subtrees

Another critical question for learning decision trees is when to step:

- **Base Case One:** If all data within the subtree have the same outputs, then don't recurse (no further split needed).
- **Base Case Two:** If all data within the subtree have exactly the same input features, then don't recurse further splits won't separate them).
- **Base Case Three (BAD IDEA):** If all attributes have small information gain, then don't recurse. An exapmle is $Y = X_1 XOR X_2$, where each of the splits X_1 and X_2 has 0 information gain but splitting both can perfectly learn Y.

4 REFERENCES

[1] Hyafil, Laurent and Rivest, Ronald L. (1976). "Constructing optimal binary decision trees is NP-complete". In: *Information Processing Letters* 5.1, pp. 15–17. ISSN: 0020-0190. DOI: https://doi.org/10.1016/0020-0190(76)90095-8. URL: https://www.sciencedirect.com/science/article/pii/0020019076900958.