Classification, Decision Trees, and a Generalization Theorem

Yufei Tao

Department of Computer Science and Engineering
Chinese University of Hong Kong

In **classification**, we are given a training set containing objects of two **classes**, and want to learn a **classifier** to predict the class of an object outside the training set. This course will cover several techniques to perform classification effectively. We will start with one such technique: the **decision tree** method.

Classification

Let $A_1, ..., A_d$ be d **attributes**.

Define the **instance space** as $\mathcal{X} = dom(A_1) \times dom(A_2) \times ... \times dom(A_d)$ where $dom(A_i)$ represents the set of possible values on A_i.

Define the **label space** as $\mathcal{Y} = \{-1, 1\}$ (the elements in $\mathcal{Y}$ are called the **class labels**).

Each **instance-label pair** (a.k.a. **object**) is a pair $(\boldsymbol{x}, y)$ in $\mathcal{X} \times \mathcal{Y}$.

- $\boldsymbol{x}$ is a vector; we use $\boldsymbol{x}[A_i]$ to represent the vector's value on A_i $(1 \leq i \leq d)$.

Denote by $\mathcal{D}$ a probabilistic distribution over $\mathcal{X} \times \mathcal{Y}$.

Classification

Goal: Given an object $(\boldsymbol{x}, y)$ drawn from $\mathcal{D}$, we want to predict its label y from its attribute values $\boldsymbol{x}[A_1], ..., \boldsymbol{x}[A_d]$.

We will find a function

$$h : \mathcal{X} \rightarrow \mathcal{Y}$$

which is referred to as a **classifier** (sometimes also called a **hypothesis**). Given an instance $\boldsymbol{x}$, we predict its label as $h(\boldsymbol{x})$.

The error of h on $\mathcal{D}$ — denoted as $err_{\mathcal{D}}(h)$ — is defined as:

$$err_{\mathcal{D}}(h) \quad = \quad \boldsymbol{Pr}_{(\boldsymbol{x},y)\sim\mathcal{D}}[h(\boldsymbol{x}) \neq y]$$

namely, if we draw an object $(\boldsymbol{x}, y)$ according to $\mathcal{D}$, what is the probability that h mis-predicts the label?

Classification

Ideally, we want to find an h to minimize $err_{\mathcal{D}}(h)$, but this in general is not possible without the precise information about $\mathcal{D}$.

Instead, We would like to learn a classifier h with small $err_{\mathcal{D}}(h)$ from a **training set** S where each object is drawn independently from $\mathcal{D}$.

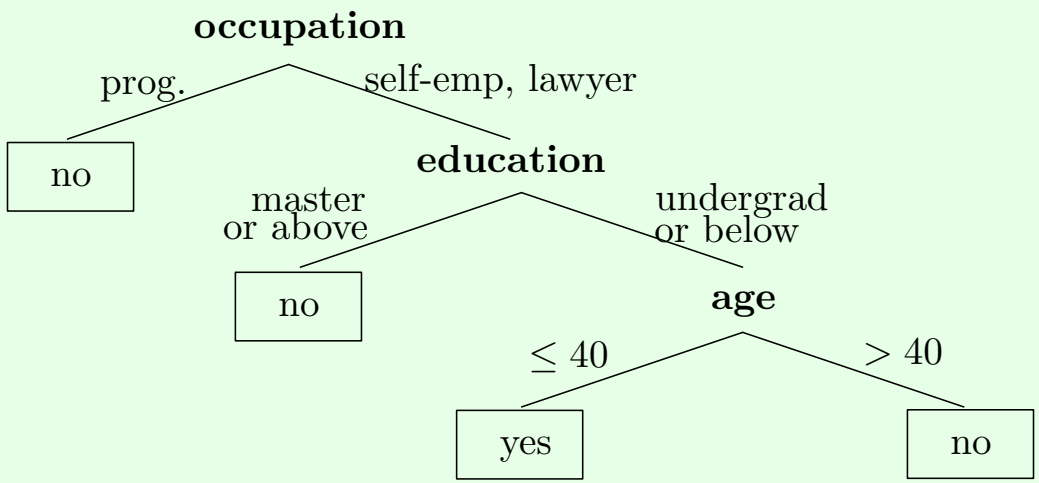
Example: Suppose that we have the following training set:

age	education	occupation	loan default
28	high school	self-employed	yes
32	master	programmer	no
33	undergrad	lawyer	yes
37	undergrad	programmer	no
40	undergrad	self-employed	yes
45	master	self-employed	no
48	high school	programmer	no
50	master	laywer	no
52	master	programmer	no
55	high school	self-employed	no

Now, given a new customer (50, high school, self-employed), how should we predict whether s/he would default?

The **decision tree method** represents a classifier h as a tree.

Example:



Given an instance (50, high school, self-employed), the above tree returns the class label "no" by descending a root-to-leaf path to the rightmost leaf.

Formally, a decision tree T is a binary tree where

- each leaf node carries a class label: yes or no (namely, 1 or -1);
- each internal node u has two child nodes, and carries a predicate P_u on an attribute A_u.

Given an instance $\boldsymbol{x}$, T predicts its label as follows:

1. $u \leftarrow$ the root of T.
2. If u is a leaf, return the class label associated with u.
3. If u is an internal node, check whether $\boldsymbol{x}[A_u]$ satisfies P_u:
   - if so, $u \leftarrow$ the left child of u;
   - otherwise, $u \leftarrow$ the right child of u.

Our objective is to produce a good decision tree from the training set S. Next, we will describe a simple algorithm called the **Hunt's algorithm** which achieves the purpose reasonably well in practice.

Given a node u in T, define $S(u)$ as follows:

- If u is the root of T, $S(u) = S$.
- Recursively, consider now u as an internal node whose $S(u)$ has been defined. Let v_1, v_2 be the left and right child nodes of u, respectively.
  - $S(v_1)$ is the set of objects in $S(u)$ that satisfy $P(u)$;
  - $S(v_2) = S(u) \setminus S(v_1)$.

Think: What is $S(u)$ for each node u in the decision tree on Slide 7?

Hunt's Algorithm

The algorithm builds a T in a top-down and greedy manner. At each node u, it finds the "best" way to split $S(u)$ according to a certain quality metric.

algorithm Hunt(S)

/* S is the training set; the function returns the root of a decision tree */

1. if all the objects in S belong to the same class
2. return a leaf node with the value of this class
3. if all the objects in S have the same attribute values
4. return a leaf node whose class label is the majority one in S
5. find the **"best"** split attribute A^* and predicate P^* /* details next slide */
6. $S_1 \leftarrow$ the set of objects in S satisfying P^*; $S_2 \leftarrow S \setminus S_1$
7. $u_1 \leftarrow$ Hunt(S_1); $u_2 \leftarrow$ Hunt(S_2)
8. create a root u with left child u_1 and right child u_2
9. set $A_u \leftarrow A^*$, and $P_u \leftarrow P^*$
10. return u

Implementing Line 5 requires resolving the following issues:

1. What are the possible ways to perform a split?
2. How to evaluate the quality of a split?

We will provide a way to resolve these issues in the next few slides.

Candidate Split

A split concerns a single attribute A. We distinguish two types of A:

- **Ordinal**: there is an ordering on A.
- **Nominal**: no ordering makes sense on A.

Example: In the training set of Slide 6, age and education are ordinal attributes, whereas occupation is nominal.

Candidate Split

For an ordinal attribute A, a **candidate split** is a condition of the form $A \leq v$, where v is a value of A appearing in S such that S has at least one object satisfying the condition, and has at least one object that does not.

For a nominal attribute A, a **candidate split** is be a condition of the form $A \in S$, where S is a subset of the values of A appearing in S such that S has at least one object satisfying the condition, and has at least one object that does not.

Example: In the training set of Slide 6, "age $\leq$ 40", "education $\leq$ undergrad", and "occupation $\in$ {self-employed, lawyer}" are all candidate split predicates. But "age $\leq$ 41", "age $\leq$ 55", "education $\leq$ elementary", and "occupation $\in$ {professor, lawyer}" are not.

Quality of a Split

Next, we tackle the second issue of Slide 11 by resorting to **GINI index**.

In general, let S be a set of objects whose class labels are known. Define:

$$\begin{aligned} n &= |S| \\ n_y &= \text{number of objects in } S \text{ with label yes} \\ p_y &= n_y/n \\ p_n &= 1 - p_y \end{aligned}$$

The GINI index of S is:

$$GINI(S) = 1 - (p_y^2 + p_n^2)$$

Quality of a Split

Example:

- If $p_y = 1$ and $p_n = 0$ (i.e., **maximum purity**), then $GINI(R) = 0$.
- If $p_y = 0.75$ and $p_n = 0.25$, then $GINI(R) = 0.375$.
- If $p_y = 0.5$ and $p_n = 0.5$ (i.e., **maximum impurity**), then $GINI(R) = 0.5$.

It is rudimentary to verify:

Lemma: $GINI(R)$ ranges from 0 to 0.5. It increases as $|p_y - p_n|$ decreases.

Quality of a Split

We are ready to resolve the second issue on Slide 11. Suppose that S has been split into S_1 and S_2. We define the **GINI index of the split** as

$$GINI_{split} = \frac{|S_1|}{|S|} GINI(S_1) + \frac{|S_2|}{|S|} GINI(S_2).$$

The **smaller** $GINI_{split}$ is, the **better** the split quality.

At this point, we have completed the description of Hunt's algorithm on Slide 10. An important issue has been left out: **overfitting**, i.e., although a tree may fit the training set well, its error on the distribution $\mathcal{D}$ is actually rather bad.

Next, we will discuss understand what causes overfitting, and then fix the issue by modifying the algorithm slightly.

Let $\mathcal{P}$ be the set of people in the world. Given a random person, we want to predict whether s/he drinks.

Suppose that there are no attributes (i.e., $\mathcal{X} = \emptyset$). Given a training set $S \subseteq \mathcal{P}$, Hunt's algorithm returns a decision tree T that has only a single node (i.e., a leaf). Let c be the label at that leaf; clearly, T will predict the label of every person in $\mathcal{P}$ as c.

Which value of c is ideal for $\mathcal{P}$? This depends on how many people in $\mathcal{P}$ belong to the yes class. Specifically, let

$$\pi_y = \frac{\text{number of people in } \mathcal{P} \text{ of yes}}{|\mathcal{P}|}$$

$$\pi_n = \frac{\text{number of people in } \mathcal{P} \text{ of no}}{|\mathcal{P}|}$$

The optimal choice is to set c to yes if $\pi_y > \pi_n$, or to no otherwise.

Example: Suppose $\pi_y = 0.7$ and $\pi_n = 0.3$. If $c =$ yes, we err with probability 0.3; if $c =$ no, we err with probability 0.7.

However, π_y and π_n are unknown.

We rely on S to guess the relationship between π_y and π_n. If S has more yes objects, we guess $\pi_y > \pi_n$ and, hence, set c to yes; otherwise, we set c to no. This is precisely what Hunt's algorithm does.

How to make sure we obtain a good guess? Obviously we need S to be **sufficiently large**.

Without enough training data, you should not hope to build a reliable decision tree (lack of statistical significance).

As Hunt's algorithm builds a decision tree T, the $|S(u)|$ of the current node u continuously decreases as we go deeper. When $|S(u)|$ becomes too small, statistical significance is lost such that the subtree of u becomes unreliable: even though the subtree may fit the training set well, it does not accurately predict the label of an unknown object falling into the subtree. Therefore, **overfitting** occurs.

Hunt's Algorithm (Modified)

We now add a heuristic to the algorithm to alleviate overfitting.

algorithm Hunt(S)

/* S is the training set; the function returns the root of a decision tree */

1. if all the objects in S belong to the same class
2. return a leaf node with the value of this class
3. if (all the objects in S have the same attribute values) or ($|S|$ is too small)
4. return a leaf node whose class value is the majority one in S
5. find the "best" split attribute A^* and predicate P^*
6. $S_1 \leftarrow$ the set of objects in R satisfying P^*; $S_2 \leftarrow S \setminus S_1$
7. $u_1 \leftarrow$ Hunt(R_1); $u_2 \leftarrow$ Hunt(R_2)
8. create a root u with left child u_1 and right child u_2
9. set $A_u \leftarrow A^*$, and $P_u \leftarrow P^*$
10. return u

Next, we will provide a theoretical explanation about overfitting.

Given a classifier h, define its **error on S** — denote as $err_S(h)$ — to be:

$$err_S(h) = \frac{|\{(\boldsymbol{x}, y) \in S \mid h(\boldsymbol{x}) \neq y\}|}{|S|}.$$

namely, the percentage of objects in S whose labels are incorrectly predicted by h.

Remark:

- $err_S(h)$ is often called the **empirical error** of h.
- $err_{\mathcal{D}}(h)$ is often called the **generalization error** of h.

Generalization Theorem

Theorem: Let $\mathcal{H}$ be the set of classifiers that can be described in b bits. Given any value $0 < \delta \leq 1$ and any $h \in \mathcal{H}$, it holds with probability at least $1 - \delta$ that:

$$err_{\mathcal{D}}(h) \leq err_S(h) + \sqrt{\frac{\ln(1/\delta) + b \ln 2}{2|S|}}.$$

Implications: we should

- look for a decision tree that is both accurate on the training set and small in size;
- increase the size of S as much as possible.

To prove the generalization theorem, we need:

Theorem (Hoeffding Bounds): Let $X_1, ..., X_n$ be independent Bernoulli random variables satisfying $\boldsymbol{Pr}[X_i = 1] = p$ for all $i \in [1, n]$. Set $s = \sum_{i=1}^{n} X_i$. Then, for any $0 \leq \alpha \leq 1$:

$$\begin{aligned} \boldsymbol{Pr}[s/n > p + \alpha] &\leq e^{-2n\alpha^2} \\ \boldsymbol{Pr}[s/n < p - \alpha] &\leq e^{-2n\alpha^2}. \end{aligned}$$

The proof of the theorem is beyond the scope of this course.

We will also need:

Lemma (Union Bound): Let $E_1, ..., E_n$ be n arbitrary events such that event E_i happens with probability p_i. Then,

$$\boldsymbol{Pr}[\text{at least one of } E_1, ..., E_n \text{ happens}] \leq \sum_{i=1}^{n} p_i.$$

The proof is rudimentary and left to you.

Proof of the Generalization Theorem

Let $\mathcal{H}$ be the set of classifiers that can be described with b bits. $|\mathcal{H}| \leq 2^b$.

Fix any classifier $h \in \mathcal{H}$.

Let S be the training set; set $n = |S|$. For each $i \in [1, n]$, define $X_i = 1$ if the i-th object in S is incorrectly predicted by h, or 0 otherwise. Hence:

$$err_S(h) = \frac{1}{n} \sum_{i=1}^{n} X_i.$$

Proof of the Generalization Theorem

Since each object in S is drawn from $\mathcal{D}$ independently, for every $i \in [1, n]$:

$$\textbf{\textit{Pr}}[X_i = 1] = err_{\mathcal{D}}(h).$$

By the Hoeffding bounds, we have:

$$\textbf{\textit{Pr}}[err_S(h) < err_{\mathcal{D}}(h) - \alpha] \quad \leq \quad e^{-2n\alpha^2}$$

which is at most $\delta/2^b$ by setting $e^{-2n\alpha^2} = \delta/2^b$, namely

$$\alpha = \sqrt{\frac{\ln(1/\delta) + b \ln 2}{2n}}.$$

We say that h **fails** if $err_S(h) < err_{\mathcal{D}}(h) - \alpha$.

Proof of the Generalization Theorem

The above analysis shows that every classifier in $\mathcal{H}$ fails with probability at most $\delta/2^b$. By the Union Bound, the probability that at least one classifier in $\mathcal{H}$ fails is at most δ. Hence, the probability that no classifiers fail is at least $1 - \delta$. $\square$

Our proof of the generalization theorem did not use any properties from decision trees. Indeed, the theorem holds for any type of classifiers. It will allow us to understand the rationales behind other classification techniques.