Classification, Decision Trees, and A Theorem on Generalization Errors

Yufei Tao

Department of Computer Science and Engineering
Chinese University of Hong Kong

The first data mining topic we will study in this course is classification which, in fact, a classical topic machine learning (data mining has a large overlap with machine learning). At a high level, in classification, we are given a training set which contains objects of two classes, and want to learn a classifier from the training set that allows us to predict accurately the class of an object outside the training set.

In this course, we will discuss several techniques to perform classification effectively. This lecture will start with one such technique: the decision tree method.

Classification

Let $A_1, ..., A_d$ be the **attributes** of a d-dimensional universe U, i.e.:

$$U = dom(A_1) \times dom(A_2) \times ... \times dom(A_d)$$

where $dom(A_i)$ represents the set of possible values on A_i.

Each **object** is an element of $U \times \{0, 1\}$, i.e., it takes a value $e[A_i]$ on every attribute A_i $(1 \le i \le d)$, and a **class label** $e[C]$ either 0 or 1.

Denote by D as a probabilistic distribution on $U \times \{0, 1\}$.

Classification

Goal: Given an object e drawn from D, we want to predict its label $e[C]$ from its attribute values $e[A_1], ..., e[A_d]$.

We do so by constructing a function

$$M : dom(A_1) \times dom(A_2) \times ... \times dom(A_d) \rightarrow \{0, 1\}$$

which we refer to as a **classifier**. Given any object e, we predict its class label as $M(e[A_1], ..., e[A_d])$.

We define error of M on D—denoted as $err_D(M)$—as:

$$err_D(M) \quad = \quad \boldsymbol{Pr}_{e \sim D}[M(e[A_1], ..., e[A_d]) \neq e[C]].$$

Namely, if we draw an object e according to D, what is the probability that M makes a mistake about the class label of e?

Ideally, we want to find an M to minimize $err_D(M)$.

Classification

In training, we are given a sample set R of D, where each object in R is drawn independently according to D. We refer to R as the **training set**.

We would like to learn our classifier M from R. A main research topic is to study how to do so effectively.

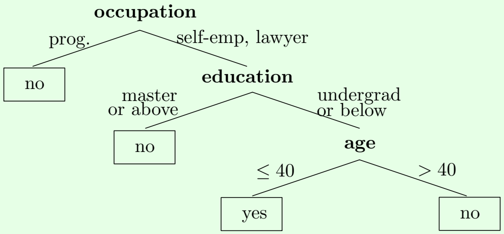
Example: Suppose that we have the following traning set:

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 we are given a new customer (50, high school, self-employed) with an unknown "default" value. How should we predict this value?

The decision tree method represents a classifier M as a tree.

Example:



Given an object (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, we define a decision tree T to be a complete binary tree where:

- each leaf node carries a class label: yes or no.
- each internal node u:
  - has two child nodes
  - carries a predicate P_u on an attribute A_u.

Given an object e with attributes $e[A_1], ..., e[A_d]$, the classifier M decides its class label $e[C]$ as follows:

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

Think: What are the predicates associated with the internal nodes in the decision tree of Slide 7?

Notice that the decision tree of Slide 7 correctly classifies all the objects in the training set of Slide 6.

Think: Give another decision tree that can correctly classify all the objects in the training set of Slide 6.

In general, our objective is to produce a good decision tree from the training set R. In the next few slides, we will discuss an algorithm called the Hunt's algorithm which achieves the purpose reasonably well in practice.

Let us first introduce a notation. Given a node u in a decision tree T, we recursively define a set $R(u)$ of objects as follows:

- If u is the root of T, $R(u) = R$.
- Let u be an internal node whose $R(u)$ has been defined, and denote by v_1, v_2 the left and right child nodes of u, respectively.
  Then, $R(v_1)$ is the set of objects in R that satisfy the predicate of $P(u)$, and $R(v_2)$ is the set of objects in R that do not satisfy the predicate of $P(u)$,.

We say that $R(u)$ is split into $R(v_1)$ and $R(v_2)$.

Think: For each node u in the decision tree on Slide 7, explain what is its $R(u)$.

Hunt's Algorithm

The algorithm builds a decision tree T in a top-down and greedy manner. More specifically, at each node u of T, it adopts the "best" way to split $R(u)$ according to an appropriate metric.

```
algorithm Hunt($R$)
/* $R$ is the training set; the function returns the root of a decision tree */
1.  if all the objects in $R$ belong to the same class
2.      return a leaf node with the value of this class
3.  if all the objects in $R$ have the same attribute values
4.      return a leaf node whose class label is the majority one in $R$
5.  find the "best" split attribute $A^*$ and predicate $P^*$ /* details next slide */
6.  $R_1 \leftarrow$ the set of objects in $R$ satisfying $P^*$; $R_2 \leftarrow R \setminus R_1$
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 $u$
```

It remains to explain how to implement Line 5 of the pseudocode. We aim to resolve the following two issues:

1. What are the candidate ways to a define a split predicate on a training set R?
2. How to evaluate the quality of a candidate split?

After this, we can implement Line 5 by setting A^* and P^* according to the candidate split of the best quality.

Candidate Split Predicate

Candidate splits can be defined in various methods (which give different variants of Hunt's algorithm). Next, we present an intuitive method that works well in practice.

A split predicate concerns one attribute A. We distinguish two types of A:

- Ordinal: there is an ordering on the values of A.
- Nominal: no ordering makes sense on the values of A.

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

Candidate Split Predicate (cont.)

For an ordinal attribute A, we define a candidate split predicate to be a condition of the form $A \leq v$, where v is a value of A that appears in R, such that R has at least one object satisfying the condition, and has at least one object violating the condition.

For a nominal attribute A, we define a candidate split predicate to be a condition of the form $A \in S$, where S is a subset of the values of A that appear in R, such that R has at least one object satisfying the condition, and has at least one object violating the condition.

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", "education $\leq$ elementary", and "occupation $\in$ {professor, lawyer}" are not. Also, "age $\leq$ 55" is not either (why?).

Quality of a Split

Now we proceed to tackle the second issue of Slide 12. Let us first introduce a key concept called GINI index.

Let R be a set of objects whose class labels are known. Define:

$$\begin{aligned} n &= |R| \\ n_y &= \text{number of objects in } R \text{ in the yes class} \\ p_y &= \frac{n_y}{n} \\ p_n &= 1 - p_y \end{aligned}$$

Then, the GINI index of R, denoted as $GINI(R)$, to be:

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

Quality of a Split (cont.)

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 the next lemma:

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

Quality of a Split (cont.)

We are now ready to quantify the quality of a split. Suppose that we have fixed a split predicate P (and hence, also the corresponding split attribute A). The split breaks the training set R into

- R_1: the set of objects in R satisfying P.
- $R_2 := R \setminus R_1$.

We define the split's GINI index as

$$GINI_{split} = \frac{|R_1|}{|R|} GINI(R_1) + \frac{|R_2|}{|R|} GINI(R_2)$$

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

At this point, we have completed the description of Hunt's algorithm on Slide 25. We have, however, intentionally left out an important issue: the algorithm often suffers from a phenomenon called overfitting, which adversely affects the accuracy of the resulting decision tree.

Fortunately, this issue can be adequately dealt with by introducing a small heuristic to the algorithm. However, to appreciate why the heuristic makes sense, we need to gain a better understanding on what is overfitting and what causes it. We will achieve the purpose by taking a probabilistic view into the essence of classification.

Let us look at a simplified classification task. Suppose that there are no attributes at all. Let $\mathcal{P}$ be the set of people in the whole world. We want to learn a classifier that, given a random person, predicts whether s/he drinks.

For this purpose, we are given a training set $R \subseteq \mathcal{P}$. Using Hunt's algorithm, we obtain from R a decision tree T.

It is easy to see that T has only a single leaf. Let c be the class value of this leaf (i.e., $c =$ either yes or no). Then, for every object in $\mathcal{P}$, we will predict its class value as c.

Which value of c would be good for $\mathcal{P}$? This ought to be related to how many people in $\mathcal{P}$ belong to the yes class, and how many to the no class. Specifically, let

$$\begin{aligned}\pi_y &= \frac{\text{number of people in } \mathcal{P} \text{ of yes}}{|\mathcal{P}|} \\ \pi_n &= \frac{\text{number of people in } \mathcal{P} \text{ of no}}{|\mathcal{P}|}\end{aligned}$$

To minimize the error in predicting a random person in $\mathcal{P}$, we should set c to yes if $\pi_y > \pi_n$, or to no otherwise.

Example 1.

Suppose $\pi_y = 0.7$ and $\pi_n = 0.3$. Then, if we set c to yes, we will be correct 70% of the time. On the other hand, if we set c to no, we will be correct only 30% of the time.

However, we do not know the real values of π_y and π_n. Hence, we rely on R to guess the relationship between those two values. If there are more yes objects in R than no 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 is doing.

How to make sure we obtain a good guess? Obviously we need, very crucially, the size s of R is large.

This is very intuitive: if you do not have enough training data, you should not hope to build a reliable decision tree. In such a case, your training data have no statistical significance.

With the above insight, we are now ready to explain the issue of overfitting. As Hunt's algorithm builds the decision tree T in a top-down manner, the size $R(u)$ of the current node u continuously decreases as we go deeper into T. When $|R(u)|$ has become too small, statistical significance is lost, such that the subtree of u we grow according to Hunt's algorithm becomes unreliable. The consequence is that even though the subtree may fit the training set very well, it does not predict well the classes of objects outside the training set. Therefore, overfitting occurs.

Think: Use the probabilistic view in the previous few slides to explain why it is a bad idea to grow the subtree of u when $|R(u)|$ is small.

Hunt's Algorithm (Modified)

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

```
algorithm Hunt($R$)
/* $R$ is the training set; the function returns the root of a decision tree */
1.  if all the objects in $R$ belong to the same class
2.      return a leaf node with the value of this class
3.  if (all the objects in $R$ have the same attribute values)
    or ($|R|$ is too small)
4.      return a leaf node whose class value is the majority one in $R$
5.  find the "best" split attribute $A^*$ and predicate $P^*$
6.  $R_1 \leftarrow$ the set of objects in $R$ satisfying $P^*$; $R_2 \leftarrow R \setminus R_1$
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 $u$
```

Remark: Judging whether $|R|$ is too small is application dependent. A simple heuristic is to introduce a threshold τ such that $|R|$ is deemed too small if $|R| < \tau$.

Heuristic Warning

Hunt's algorithm is a heuristic algorithm that does not promise strong theoretical guarantees. In other words, it is designed based on some reasonable idea that "seem to work", but offers no assurance on the quality of the obtained decision tree on a new object.

Heuristic Warning (cont.)

In fact, the algorithm itself can be modified—heuristically—in many different ways, some of which may offer better performance in certain scenarios. The following is just a short list of the possible modifications:

- Definition of a candidate split.
  - To alleviate overfitting, it may make sense to require a candidate split to generate only sufficiently large subsets. Currently, a candidate split may create subsets of size 1.
- Quality assessment of a candidate split.
- Stopping condition (i.e., when do we want to make a node a leaf).

Next, we will provide a theoretical explanation about the overfitting phenomenon using a concrete formula that allows us to bound $err_D(M)$, namely, the error of a decision tree M on an unseen object drawn from D.

Denote by $err_R(M)$ the error of M on R to be the percentage of objects in R whose labels are incorrectly predicted by M, namely:

$$err_R(M) = \frac{|\{e \in R \mid M(e[A_1], ..., e[A_d]) \neq e[C]\}|}{|R|}.$$

$err_R(M)$ is usually called the training error.

Now, fix an encoding scheme of a decision tree.

Note that for every decision tree M, you must somehow represent it as a sequence of bits, according to certain encoding conventions that you settle for. The encoding scheme explains all your conventions so that any one else can look at the scheme and encode any M in precisely the same way as you do.

Generalization Theorem

Theorem: Suppose that M can be described in b bits (according to the chosen encoding scheme). Given any value $0 < \delta \le 1$, it holds with probability at least $1 - \delta$ that:

$$err_D(M) \le err_R(M) + \sqrt{\frac{\ln(1/\delta) + b \ln 2}{2|R|}}.$$

We will refer to the above as the **generalization theorem**. Before proving the theorem in the next slide, let us observe its implications:

- We should look for a decision tree that is both accurate on the training set (i.e., reducing $err_R(M)$) and small in size (i.e., reducing b).
- Increase the size of R 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 $0 \le \alpha \le 1$:

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

We will skip the proof of the theorem which is beyond the scope of this course.

We will also need the following obvious fact:

Lemma (Union Bound): Let $\Lambda_1, ..., \Lambda_n$ be n arbitrary events such that event Λ_i happens with probability p_i. Then the probability that at least one of $\Lambda_1, ..., \Lambda_n$ happens is at most $\sum_{i=1}^{n} p_i$.

The proof is rudimentary and left to you.

Proof of the Generalization Theorem

Let $\mathcal{H}$ be the set of possible classifiers that can be described with b bits according to fixed encoding scheme. Clearly, $|\mathcal{H}| \le 2^b$.

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

Let R be the training set; set $n = |R|$. For each $i \in [1, n]$, define X_i to take the value 1 if the i-th object in the training set is incorrectly predicted by M, and 0 otherwise. Hence:

$$err_R(M) = \frac{1}{n} \sum_{i=1}^{n} X_i.$$

Proof of the Generalization Theorem

On the other hand, as each object in R is drawn from D independently, we know for every $i \in [1, n]$:

$$\boldsymbol{Pr}[X_i = 1] = err_D(M).$$

By the Hoeffding bounds, we have:

$$\boldsymbol{Pr}[err_R(M) > err_D(M) + \alpha] \quad \leq \quad e^{-2n\alpha^2}$$

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

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

When $err_R(M) > err_D(M) + \alpha$ happens, we say that M fails.

Proof of the Generalization Theorem

Remember that we have fixed M to be one, but an arbitrary one, classifier in $\mathcal{H}$. The above analysis shows that any one 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 classifier in $\mathcal{H}$ fails is at least $1 - \delta$. This completes the proof. $\square$

We close this lecture by pointing out that 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 be a very useful fact that will allow us to understand better the rationales behind other classification techniques.