Machine Learning Algorithms
Project Reading

Michael Painter

November 19, 2015

0 General Notation and Concise Description of Machine Learning Algorithms

Here we look at some machine learning algorithms. We will be looking at a classification problem which we define as the following. Given a feature vector $\mathbf{v} = (x_1, x_2, \ldots, x_d) \in \mathbb{R}^d$ and a set of classes $\mathcal{C} = \{c_1, c_2, \ldots, c_n\}$ we want to *learn* a function $H : \mathbb{R}^d \to \mathcal{C}$. What we mean by learning is given a *training sequence* $S = ((\mathbf{v}_1, c_1), (\mathbf{v}_2, c_2), \ldots, (\mathbf{v}_m, c_m))$, can we find an H such that $H(\mathbf{v}_i) = c_i$ for most $i \in \{1, \ldots, m\}$. We have an *ill-posed* problem, because we use the word 'most'. And we use this because there may be outliers and anomalous cases in S, that is the data is commonly noisy.

Maybe want to improve the intro, and better define what it means for a machine to 'learn' something.

1 Decision Trees

1.1 Definition of a Decision Tree

With the decision tree method we construct a binary tree from the learning sequence S, which we can intuitively think of as a flow diagram of yes/no questions to determine the class of a given feature vector.

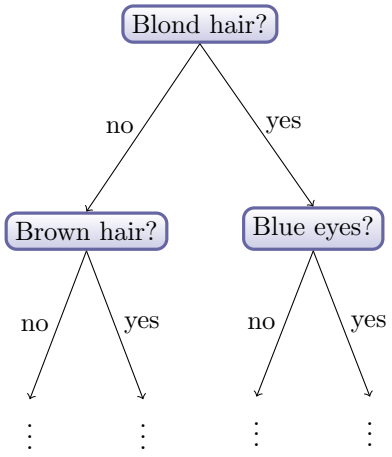

Figure 1: Example of (part of) a tree used to classify humans.

More formally we can define a decision tree T, where we have a set of states $Q \subseteq \mathbb{N}$, and we assume that if $i \in Q$ then it's children (if any) are $(2i)$ and $(2i + 1)$, both in Q. For each node $i \in Q$ we have some learned parameter settings θ_i, which is used to be used in a *split function* $h : \mathbb{R}^d \times \mathbb{R}^d \to \{0, 1\}$. The split function will give $h(\mathbf{x}; \theta_i) = 0$ if we should traverse to the 'left' sub tree of i whilst classifying $\mathbf{x}$, and traverse right if $h(\mathbf{x}; \theta_i) = 1$. Finally each root node of the tree is associated with a class in $\mathcal{C}$, which if reached is what the decision tree will output as its classification for the feature vector.

1.2 Training a Decision Tree

When we teach a decision tree a greedy approach is used. This means that we pick an optimal solution for dividing our training sequence into two at each node, which may not necessarily lead to the optimal decision tree. Creating an optimal decision tree is an NP-Complete problem.
Each leaf node of the tree will be associated with a given class in $\mathcal{C}$. We can classify $\mathbf{x} \in \mathbb{R}^n$
Lets consider how we might build a decision tree. Suppose we have constructed the tree up to some node i. At this node i let $S_i \subseteq S$ be a subset of the training sequence, containing all of the feature vectors that would reach i by traversing the tree so far.
we have a set of possible classifications $C_i \subseteq \mathcal{C}$. We use the feature vector and some learned parameter settings θ in a *split function* h, where we progress to state $2i$ if $h(\mathbf{v}, \theta) = 0$ and $2i + 1$ if $h(\mathbf{v}, \theta) = 1$. We require that the sets of possible classes satisfy

$$C_{2i} \subseteq C_i \tag{1}$$
$$C_{2i+1} \subseteq C_i \tag{2}$$

and preferably satisfies

$$C_{2i} \cap C_{2i+1} = \emptyset. \tag{3}$$

When teaching the tree, typically a greedy approach is used. By this we mean we pick an optimal solution for splitting classes at each node, which may not necessarily lead to picking the optimal tree. Consider a finite training sequence $S \subseteq \mathbb{R}^d \times \mathcal{C}$. For ease of notation we consider all feature vectors in S to be unique, however in practise they may not be. We can define the entropy of the training sequence by

$$H(S) = -\sum_{c \in \mathcal{C}} p(c) \log(c) \tag{4}$$

where we have

$$p(c) = \frac{|\{\mathbf{v} \mid c' = c \wedge (\mathbf{v}, c') \in S\}|}{|S|} \tag{5}$$

which is the empirical probability of a training sample vector in s being classified as c.

Now consider $S_j \subseteq S$ at some node j, and suppose we have split them into a left and right partition S_j^L and S_j^R. The *information gain* I can then be defined as

$$I = H(S_j) - \sum_{i \in \{L,R\}} \frac{|S_j^i|}{|S_j|} H(S_j^i). \tag{6}$$

We now consider how we can partition the training sequence for a maximal information gain. To do this we may define the split function $h(\mathbf{v}, \theta)$. To separate the sequence into two we can define a hyper-surface, the most simple of which being a hyper-plane, and the hyper-surface is described by θ. If we are using a hyper-plane model, then we let $\theta = (\phi, \psi, \tau)$. ϕ is a filter for a feature vector $\mathbf{v}$, that is $\phi(\mathbf{v}) = (v_{\phi_1}, v_{\phi_2}, \ldots, v_{\phi_{d'}}) \in \mathbb{R}^{d'}$ where $\phi_i \in \{1, \ldots, d\}$ for each i, and $d' \leq d$. Then $\psi \in \mathbb{R}^{d'}$ and $\tau \in \mathbb{R}$ defines a $(d' - 1)$ dimensional hyper-plane.

hfhfhf

Figure 2: TODO: Add 3 images, one of an axis aligned line, one of a non-axis aligned line and a quadratic curve, used to separate some data.

Let $\mathbb{I}$ be the indicator function

$$\mathbb{I}(x) = \begin{cases} 0 & \text{if } x \text{ false} \\ 1 & \text{if } x \text{ true.} \end{cases} \tag{7}$$

Now we define h as

$$h(\mathbf{v}, \theta) = \mathbb{I}(\phi(\mathbf{v}) \cdot \psi \geq \tau). \tag{8}$$

We can now define S_j^L and S_j^R in terms of S_j and θ

$$S_j^L(S_j, \theta) = \{(\mathbf{v}, c) \in S_j \mid h(\mathbf{v}, \theta) = 0\} \tag{9}$$

$$S_j^L(S_j, \theta) = \{(\mathbf{v}, c) \in S_j \mid h(\mathbf{v}, \theta) = 1\}. \tag{10}$$

So far we have arbitrarily partitioned our dataset according to some unknown parameter θ, this is what we need to learn for each node of our tree. Consider the information gain defined in equation (6), which becomes a function in S_j and **theta**

$$I(S_j, \theta) = H(S_j) - \sum_{i \in \{L,R\}} \frac{|S_j^i(S_j, \theta)|}{|S_j|} H(S_j^i(S_j, \theta)) \tag{11}$$

and then it is clear in an ideal world that we pick

$$\theta_j = \arg\max_{\theta \in \mathcal{T}} I(S_j, \theta) \tag{12}$$

as our value for θ, where $\mathcal{T}$ is the space of all possible 'split parameters'. Finding θ_j exactly in most cases will be computationally infeasible, and we have to approximate it.

So now at a given node j for some feature vector $\mathbf{v}$ we can pick the child node to move to using $h(\mathbf{v}, \theta_j)$.

We have described how to train a single node of a tree, we then recursively train nodes in a decision tree until some prescribed limit, such as tree depth.

1.3 Advantages and Disadvantages

Some (supposed) advantages:

- A decision tree is a simple concept to understand (despite some complexity in training it).
- Robustness. It tends to work well even if
- Through some clever training we can seemingly get 'dimensionality reduction' for free. We note also that the dimensions of the feature vector are explicitly stored in θ and so we can read them easily.
- Deals with large datasets well.

Some (supposed) disadvantages:

- If not careful decision tree learners can create over-complex trees and suffers from overfitting as a consequence.
- We make a greedy choice when training the tree at each node, and so we don't find an optimal decision tree, which can be shown to be NP-Complete.
- The are concepts that a decision tree struggle to learn, such as parity problems.

We note that some of the disadvantages can be overcome or reduced by using random forest models.