Learning From Examples

Michael Rose

In which we describe agents that can improve their behavior through diligent study of their own experiences

18.1 | Forms of Learning

Any component of an agent can be improved by learning from data. The improvements, and techniques used to make them, depend on four major factors:

- Which component is to be improved
- What prior knowledge the agent already has
- What representation is used for the data and the component
- What feedback is available to learn from

18.2 | Supervised Learning

The task of supervised learning is this:

Given a training set of N example input-output pairs $(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)$, where each y_j was generated by an unknown function $y = f(x)$, discover a function h that approximates the true function f.

In general, there is a tradeoff between complex hypotheses that fit the training data well and simpler hypotheses that may generalize better. We say that a learning problem is **realizable** if the hypothesis space contains the true function. Supervised learning can be done by choosing the hypothesis h^* that is most probable given the data:

$$h^* = \arg\max_{h \in \mathcal{H}} P(h|data) = \arg\max_{h \in \mathcal{H}} P(data|h)p(h)$$

There is a tradeoff between the expressiveness of a hypothesis space and the complexity of finding a good hypothesis within that space. For example, fitting a straight line to data is an easy computation; fitting high degree polynomials is somewhat harder.

18.3 | Learning Decision Trees

The greedy search used in decision tree learning is designed to approximately minimize the depth of the tree. The idea is to pick the attribute that goes as far as possible toward providing an exact classification of the other examples. We need a formal measure of good and bad in order to understand the concept of importance in the choices we make. We will use the notion of information gain, which is defined in terms of **entropy**.

Entropy is a measure of the uncertainty of a random variable; acquisition of information corresponds to a reduction in entropy. In general, the entropy of a random variable V with values v_k, each with probability $P(v_k)$, is defined as:

$$\text{Entropy: } H(V) = \sum_k P(v_k) \log_2 \frac{1}{P(v_k)} = -\sum_k P(v_k) \log_2 P(v_k)$$

A randomly chosen example from the training set has the kth value for the attribute with probability $\frac{(p_k+n_k)}{(p+n)}$, so the expected entropy remaining after testing attribute A is

$$\text{Remainder}(A) = \sum_{k=1}^{d} \frac{(p_k+n_k)}{(p+n)} B(\frac{(p_k)}{(p_k+n_k)})$$

Then the **information gain** from the attribute test on A is the expected reduction in entropy:

$$\text{InformationGain}(A) = B(\frac{p}{p+n}) - \text{Remainder}(A)$$

For decision trees, a technique called **decision tree pruning** combats overfitting. Pruning works by eliminating nodes that are not clearly relevant. In order to detect if a node is irrelevant, we check information gain. In order to ascertain how large a gain should be required in order to split on a particular attribute, we can use a statistical significance test.

In this case, the null hypothesis is that the attribute is irrelevant and the information gain for an infinitely large sample would be zero. We can measure the deviation by comparing actual numbers of positive and negative examples in each subset, p_k and n_k, with the expected numbers, $\hat{p_k}$ and $\hat{n_k}$, assuming true irrelavance:

$$\hat{p_k} = p \times \frac{(p_k+n_k)}{(p+n)} \qquad \hat{n_k} = n \times \frac{(p_k+n_k)}{(p+n)}$$

A convenient measure of total deviation is given by

$$\Delta = \sum_{k=1}^{d} \frac{(p_k-\hat{p_k})^2}{p+n} + \frac{(n_k=\hat{n_k})^2}{p+n}$$

Under the null hypothesis, the value of Δ is distributed according to the χ^2 distribution with $v-1$ degrees of freedom. This form of pruning is known as χ^2 pruning.

18.3.6 | Broadening the Applicability of Decision Trees

To extend the decision tree induction to a wider variety of problems, a number of issues must be addressed.

- Missing data : How should we classify an example that is missing one of the test attributes? How should we modify the information gain formula when some examples have unknown values for the attribute
- Multivalued attributes : When an attribute has many possible values, the information gain measure gives an inappropriate indication of the attributes usefulness. One solution is to use the gain ratio.
- Continuous and integer-valued input attributes : Continuous or integer-valued attributes (like height weight) have an infinite set of possible values. Rather than generating infinite branches, decision trees typically find the split point that gives the highest information gain. Splitting is the most expensive part of real world decision tree learning applications.
- Continuous valued output attributes: If we are trying to predict a numerical output, we need a regression tree rather than a classification tree.

18.4 | Evaluating and Choosing the Best Hypothesis

We wish to learn a hypothesis that fits the future data best. In order to do this, we make the **stationarity assumption**: that there is a probability distribution over examples that remains stationary over time. Essentially, we want iid values.

The next step is define best fit. We define the **error rate** of a hypothesis as the proportion of mistakes that it makes.

18.4.1 | Model Selection: Complexity vs. Goodness of Fit

We can think of the task of finding the best hypothesis as two tasks: model selection defines the hypothesis space and then optimization finds the hypothesis within that space.

18.4.2 | From Error Rates to Loss

In machine learning it is traditional to express utilities by means of a **loss function**. The loss function $L(x, y, \hat{y})$ is defined as the amount of utility lost by predicting $h(x) = \hat{y}$ when the correct answer is $f(x) = y$. In general small errors are better than large ones. Two functions that implement this idea are the absolute value of the different (L_1 loss) and the square of the difference (L_2 loss). If we are content with the idea of minimizing error rate we can use the $L_{0/1}$ loss function, which has a loss of 1 for an incorrect answer and is appropriate for discrete-valued outputs:

$$\text{Absolute Value Loss: } L_1(y, \hat{y}) = |y - \hat{y}|$$
$$\text{Squared Error Loss: } L_2(y, \hat{y}) = (y - \hat{y})^2$$
$$\text{0/1 Loss: } L_{0/1}(y, \hat{y}) = 0 \quad \text{if} \quad y = \hat{y}, \quad \text{else} \quad 1$$

The learning agent can theoretically maximize its expected utility by choosing the hypothesis that minimizes the expected loss over all input-output pairs it will see. It is meaningless to talk about expectation without defining a prior probability distribution $P(X, Y)$ over examples. Let Ξ be the set of all possible input-output examples. Then the expected **generalization loss** for a hypothesis h with respect to loss function L is

$$\text{GenLoss}_L(h) = \sum_{(x,y)\in\Xi} L(y, h(x))P(x, y)$$

and the best hypothesis, h^*, is the one with the minimum expected generalization loss:

$$h^* = \arg\min_{h\in\mathcal{H}} \text{GenLoss}_L(h)$$

Since $P(x, y)$ is not known, the learning agent can only estimate generalization loss with **empirical loss** on a set of examples, E:

$$\text{EmpLoss}_{L,E}(h) = \frac{1}{N} \sum_{(x,y)\in E} L(y, h(x))$$

where the estimated best hypothesis $\hat{h^*}$ is the one with the minimum empirical loss:

$$\hat{h^*} = \arg\min_{h\in\mathcal{H}} EmpLoss_{L,E}(h)$$

18.5 | The Theory of Learning

How do we know that the hypothesis h is close to the target function f if we don't know what f is? How many examples do we need to get a good h? What hypothesis space should we use? If the hypothesis space is very complex, can we even find the best h or do we have to settle for a local maximum in the space of hypotheses? How complex should h be? How do we avoid overfitting? Questions like these are addressed by **computational learning theory**.

The underlying principle is that any hypothesis that is seriously wrong will almost certainly be found out with high probability after a small number of examples, because it will make an incorrect prediction. Thus, any hypothesis that is consistent with a sufficiently large set of training examples is unlikely to be seriously wrong: it must be **probablity approximately correct**. Any learning algorithm that returns hypotheses that are probably approximately correct is called a **PAC learning algorithm**. We can use this approach to provide bounds on the performance of various learning algorithms.