My ML studies

Babak Poursartip

June 10, 2023

Contents

Chapter 1

Introduction to ML

1.1 General

- Supervised learning (SL): functional approximation from labeled data.
- Unsupervised learning (UL): functional description. Learning from unlabeled data.
- Reinforcement learning (RL): Learning from delayed reward.

Chapter 2

Supervised learning (SL)

There are two types of supervised learning:

- classification: taking some inputs and mapping it to some discrete labels. (true of false; male or female; SUV or sedan or truck; class1, class2, class3.)
- regression: returns continuous valued function. Mapping from input to some **real number**. Something like age is more like discrete number, so, it is also classification.

2.1 Terms

- Instances: Inputs, such as pictures, pixels, etc.
- Concept: A set of *functions* that maps inputs to outputs.
- Target concept: The actual function that can map inputs to outputs. This is the actual answer.
- Hypothesis class: Set of all the functions that we are going to think about.
- Sample (Training set): A set of instances with correct labels.
- Candidate: The best approximation of the target concept.

- Testing set: A set of instances with correct labels that was not visible to the learning algorithm during the training phase. It's used to determine the algorithm performance on novel data. we can argue that we learned something by memorization, but indeed, we just memorized the concepts. What we want is generalization.

2.2 SL: Decision tree

2.2.1 Introduction

Decision tree is a classification learning in the form of a sequence of decisions based on the attributes/features/questions (which forms the nodes) applied to every instance to assign it to a specific class. Answers to the questions would be the edges of the trees. For example, deciding to eat at a restaurant or not, or classification of vehicles into sedans, SUVs, trucks, etc.

2.2.2 Representation of the decision tree:

A decision tree is a structure of nodes, edges and leaves that can be used to *represent* the data. We always start at the root of the tree, otherwise we don't look at any other attribute in the tree.

- Nodes represent attributes where you ask a question about it. (vehicle length, vehicle height, number of doors, etc.).
- Edges represent values/answers, a specific value for each attribute/question.
- Leaves represent the output (or final answer). For example, vehicle's type.

2.2.3 Algorithm to build a decision tree

Algorithm means how to create the decision tree. A decision tree *algorithm* is a sequence of steps that will lead you to the desired output. To form a decision tree, we need to come up with the attributes that can split the space, roughly in half.

- Pick the best attribute (the one that can split the data roughly in half). If this attribute added no valuable information (not a good split), it might cause overfitting.

- Ask a question about this attribute.
- Follow the correct answer path.
- Loop back to (1) till you narrow down the possibilities to one answer (the output).

Note that the usefulness of a question depends upon the answers we got to previous questions.

2.2.4 Expressiveness

- Decision trees can basically express any function using the input attributes.
- For Boolean functions (AND, OR, XOR, etc.), the truth table row will be translated into a path to a leaf.
- How many decision trees we can generate for a specific function/problem? For n boolean attributes with boolean output, the decision tree hypothesis space is very huge ($2^{(2^n)}$). (Babak note: this number of decision tree is not really distinct, see my notebook). This is why we need to design algorithms to efficiently search the hypothesis space for the best decision tree.

Decision trees with AND and OR are linear (they need linear number of nodes-ANY type problems). But XOR is a hard problem (requires 2^n nodes-PARITY type of problems).

2.2.5 ID3 algorithm to create a decision tree

We need to find the best attributes for each node, to narrow down the space with each question. ID3 (Inducing Decision Tree (3)) builds decision trees using a top-down, greedy approach. The greedy part of the approach comes from the fact that it will decide which attribute should be at the root of the tree by looking just one move ahead. It compares all available attributes to find which one classifies the data the best, but it doesn't look ahead (or behind) at other attributes to see which combinations of them classify the data the best. ID3 algorithm returns an optimal decision tree, but as the size of the training data and the number of attributes increase, it becomes more likely that running ID3 on it will return a suboptimal decision tree.

Pseudo code:

1. Pick the best attribute A (the definitions of best attribute comes later). To select this attribute, a statistical test is used to determine for each attribute how well it alone classifies the training examples.

2. Split the data into subsets that correspond to the possible values of the best attribute.

3. Assign A as a decision attribute for a node.

4. For each value of A, create a descendant node.

5. Sort training examples to leaves.

6. If examples perfectly classified (means all data point are of the same class) or there is no more attributes – Stop – else –Iterate over leaves

How to find the best attribute for the decision tree at each level:
There are a couple of options. The most common method is called information gain: a measure that expresses how well an attribute splits the data into groups based on classification. For example, an attribution returns *low information gain*, if it divides samples to two classes with an even yes and no, but the information gain is high, if each class has more of yeses or nos.

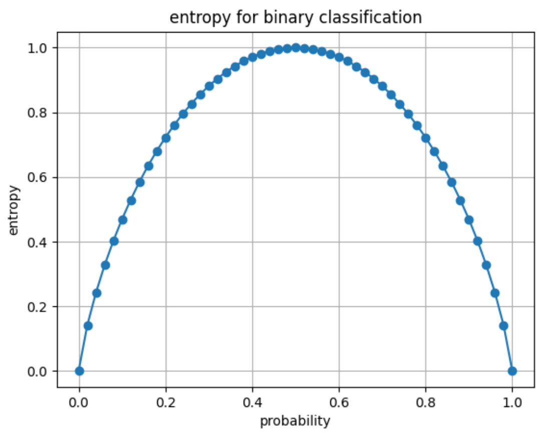
It quantifies the reduction in randomness (Entropy) over the labels we have with a set of data, based upon knowing the value of a particular attribute. A mathematical way to measure the gain, is entropy. It measures the homogeneity of a data set S's classifications. *Entropy ranges from 0, which means that all of the classifications in the data set are the same (either yes or no), to* log_2 *of the number of different classifications, which means that the classifications are equally distributed within the data set.* For a binary classification the entropy is only $log_2 1 = 1$ (if yes and no samples are equally divided). entropy is calculated as follows:

$$Entropy(S) = \sum_{i=1}^{c} -p_i \, log_2(p_i) \tag{2.1}$$

where:

- c corresponds to the number of different classifications (either for the attribute or the final output)
- p_i corresponds to the proportion of the data with the classification i

Here is the plot of entropy for a binary classifier, as the proportional of yes and no samples change:



Information gain measures the reduction in entropy that results from partitioning the data on an attribute A, which is another way of saying that it represents how effective an attribute is at classifying the data. Given a set of training data S and an attribute A, the formula for information gain is:

$$Gain(S, A) = Entropy(S) - \sum_{v} \frac{|S_v|}{|S|} Entropy(S_v) \tag{2.2}$$

where v is all the possible values of attribute the attribute (for example, sunny, cloudy, rainy), and S_v is the total number of samples corresponding to this value of attribute (for example sunny).

For *if I play tennis game* (yes and no is the final output) with attributes weather (sunny, cloudy, rainy), temperature (hot, cold, mild), humidity (normal, high), first, we calculate the entropy of the entire set (Entropy(S)) (wrt the the final output classification, yes and no). For each attribute, we calculate the the entropy corresponding to each value of entropy, then we calculate the final entropy.

We want to maximize information gain, so we want the entropies of the partitioned data to be as low as possible, which explains why attributes that exhibit high information gain split training data into relatively heterogeneous groups.

As the size of the training data and the number of attributes increase, it becomes likelier that running ID3 on it will return a sub-optimal decision tree.

2.2.6 Inductive bias

Two types of bias for classifiers:

- Restriction Bias: Describes the hypothesis set space (H) that we will consider for learning the tree. Complete hypothesis space (low Restriction Bias). Instead of looking at infinitely many functions, we only consider those that can be represented by the decision tree.
- Preference Bias: Describes the subset of hypothesis n ($n \in H$), from the hypothesis set space H, that we prefer.

Inductive bias of ID3 algorithm:

- it prefers the decision tree with good splits at top (even if a bad split generates the same outcome). This is because ID3 is greedy using info gain.
- it prefers correct outcome over incorrect because ID3 repeats until the labels are correctly classified.
- it prefers shorter trees (comes from the fact that we use good splits from the top).

2.2.7 Extending ID3 algorithm-other consideration

For continuous attributes, we can classify the attributes based on thresholds (that exists in the data set). The continuous data attribute provides the most information gain, while giving a decision tree that is not generalize well.

Does it make sense to repeat an attribute along a path in a decision tree? No, it does not make sense for discrete values, but for continuous values, it makes sense, because indeed we are asking a different question (for a different range).

If the data for some attributes is missing, we can choose the most popular data for the missing item. Two options here: does not take the classification into account, or choose the most popular value for the same classification. Another method to deal with the missing data, is assigning the probability of each value of the attribute to the missing entries.

When do we stop? When everything classified correctly or if there is no more attribute. What if we have noise in data (different answer for the same instance)? Causes an infinite loop. What would be the stopping criterion, then? We want generalization, and we should avoid **over-fitting**. If the tree is too big/complicated, there is a chance that it overfits. There are two popular approaches to avoid over-fitting in decision trees: stop growing the tree before it becomes too large or prune the tree after it becomes too large. Typically, a limit to a decision tree's growth will be specified in terms of the maximum number of layers, or depth, it's allowed to have.

One option to avoid over-fitting, is that we can grow the trees, and use cross-validation to prevent over-fitting. The data available to train the decision tree will be split into a training set and validation/test set and we keep growing the tree with various maximum depths will be created based on the training set and then test it against the test set. The best will be selected (when the error grows, we stop growing the tree).

Pruning the tree, on the other hand, involves testing the original tree against pruned versions of it. Leaf nodes are taken away from the tree as long as the pruned tree performs better against test data than the larger tree.

Regression: how to adapt decision trees for regression type of problems (continuous outputs)? Decision trees as we've defined them here don't transfer directly to regression problems. We no longer have a useful notion of information gain, so our approach at attribute sorting falls through. Instead, we can rely on purely statistical methods (like variance and correlation) to determine how important an attribute is. For leaves, too, we can do averages, local linear fit, or a host of other approaches that mathematically generalize with no regard for the meaning of the data.

2.2.8 Changing the information gain formula is another option

The information gain formula used by the ID3 algorithm treats all of the variables the same, regardless of their distribution and their importance. This is a problem when it comes to continuous variables or discrete variables with many possible values because training examples may be few and far between for each possible value, which leads to low entropy and high information gain by virtue of splitting the data into small subsets, but results in a decision tree that might not generalize well.

One successful approach to deal with this is using a formula called GainRatio in the place of information gain. GainRatio tries to correct for information gain's natural bias toward attributes with many possible values by adding a denominator to information gain called SplitInformation. SplitInformation attempts to measure how broadly partitioned an attribute is and how evenly distributed those partitions are. In general, the SplitInformation of an attribute with n equally distributed values is $log_2 n$. These relatively large denominators significantly affect an attribute's chances of being the best attribute after an iteration of the ID3 algorithm and help to avoid choices that perform particularly well on the training data but not so well outside of it.

$$GainRatio(S, A) = \frac{Gain(S, A)}{SplitInformation(S, A)}$$

$$SplitInformation(S, A) = -\sum_{i=1}^{c} \frac{|S_i|}{S} \, log_2 \frac{|S_i|}{|S|}$$

2.2.9 Advantages and Disadvantages of Decision Trees (copied from a text)

Advantages:

- Simple to understand and to interpret.
- Requires little data preparation. Other techniques often require data normalization, dummy variables need to be created and blank values to be removed.

- The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree.
- Able to handle both numerical and categorical data. Other techniques are usually specialized in analyzing datasets that have only one type of variable.
- Able to handle multi-output problems.
- Uses a white box model. If a given situation is observable in a model, the explanation for the condition is easily explained by Boolean logic. By contrast, in a black box model (e.g., in an artificial neural network), results may be more difficult to interpret.
- Possible to validate a model using statistical tests. That makes it possible to account for the reliability of the model.
- Performs well even if its assumptions are somewhat violated by the true model from which the data were generated.

Disadvantages:

- Can create over-complex trees that do not generalize the data well. This is called overfitting. Mechanisms such as pruning (setting the minimum number of samples required at a leaf node or setting the maximum depth of the tree) are necessary to avoid this problem.
- Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This problem is mitigated by using decision trees within an ensemble.
- The problem of learning an optimal decision tree is known to be NP-complete under several aspects of optimality and even for simple concepts. Consequently, practical decision-tree learning algorithms are based on heuristic algorithms such as the greedy algorithm where locally optimal decisions are made at each node. Such algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees in an ensemble learner, where the features and samples are randomly sampled with replacement.

- There are concepts that are hard to learn because decision trees do not express them easily, such as XOR, parity or multiplexer problems.
- Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the dataset prior to fitting with the decision tree.

2.2.10 Random forest

The random forest uses many trees, and it makes a prediction by averaging the predictions of each component tree. It generally has much better predictive accuracy than a single decision tree and it works well with default parameters. If you keep modeling, you can learn more models with even better performance, but many of those are sensitive to getting the right parameters.

2.3 SL: Regression and classification

Regression is the mapping of continuous inputs to continuous outputs, as opposed to the classification where we had mapping from discrete input to discrete output. Regression is similar to function approximation. Back then, regression meant falling back to the average/mean, but now, we mean, using functional form to approximate a bunch of points.

We use regression because the data itself is not accurate and has noise. If the data is exact, we need to use interpolation and spline methods, to exactly fit the line with the data.

2.3.1 Linear regression

Finding a linear function/relationship between the input data and the output. It might not be a good fit though. We want to find a line (linear function) such that it minimizes the deviation, or to be more precise, the squared error between the data points and the line. This is least square regression. Basically, this is fitting data with a curve, so that we can predict the results based on the model. Here we do not try to intersect every point, rather the curve is designed to follow the pattern of points. (The other approach is interpolation that we try to pass the line/curve through each data point.)

In summary, this is an approximate function that fits the shape or general trend of the data w/o necessarily matching the individual points. To find the line:

$$\begin{aligned} \text{data points: } & \{(x_i, y_i) : i = 1...n\} \\ \text{find a and b for the linear fit: } & y_i = ax_i + b \\ \text{such that minimizes error: } & e = \sum_{i=1}^{n}(y_i - (ax_i + b))^2 \end{aligned} \tag{2.3}$$

To find the the two unknowns of the linear fit, a and b, take the derivative of the error with respect to a and b and set it zero. That would be a system of equations with two unknowns. After solving the equation you have the fit. To have a nonlinear fit, we can use a polynomial model:

2.3.2 Non-linear regression

$$f(x_i) = a_0 + a_1x_i + a_2x_i^2 + ... \tag{2.4}$$

The unknowns are $a_0, a_1, ...$. See my class notes for more details.

2.3.3 Error

Training data has errors due to various reasons such as reading errors, sensor errors, transcription error, maliciously given data, etc. This is the reason we are using the regression not the interpolation.

2.3.4 Cross validation

To better fit the model, we can use higher order polynomial, but it leads to over-fitting. The goal is always to generalize the model to fit the real world data. The test set is just a representation of the data that we may see in future. The model should be complex enough to model the training set without overfitting.

The data points that we are using for training are assumed to be Independent and Identically Distributed (IID), which means that there's no inherent difference between training, testing and real-world data and all the data is coming from the same source. This is the *Fundamental Assumption of Supervised Learning.*

The idea is that we select part of the training set as the test set, not touching the actual test during the training set. This is indeed the cross validation set.

Cross Validation steps:

- Randomly partition the training data into k folds of equal size.
- Train the model on all the folds except for one (k-1).
- Validate the model's performance using the fold that was not used in training.
- Repeat steps 2 and 3 using different combinations of these folds.
- Average the error in predicting the polynomial trained on the training folds.

2.4 SL: Neural network

2.5 SL: Instance based learning

2.6 SL: Ensemble B&B

2.7 SL: Kernel methods and SVMs

2.8 SL: Comp Learning Theory

2.9 SL: VC dimensions

2.10 SL: Bayesian Learning

2.11 SL: Bayesian inference

Chapter 3

Unsupervised learning (UL)

3.1 Random optimization

3.2 Clustering

Chapter 4

Reinforced Learning (RL)

4.1 Markov Decision Process

Chapter 5

Miscellaneous topics

5.1 One hot encoding

source1 and source2

A one hot encoding is a representation of categorical variables as binary vectors. This first requires that the categorical values be mapped to integer values. Then, each integer value is represented as a binary vector that is all zero values except the index of the integer, which is marked with a 1.

Example: assume we have a sequence of labels with the values *red* and *green*. We can assign *red* an integer value of 0 and *green* the integer value of 1. As long as we always assign these numbers to these labels, this is called an integer encoding. Consistency is important so that we can invert the encoding later and get labels back from integer values, such as in the case of making a prediction. Next, we can create a binary vector to represent each integer value. The vector will have a length of 2 for the 2 possible integer values. The *red* label encoded as a 0 will be represented with a binary vector [1, 0] where the zeroth index is marked with a value of 1. In turn, the *green* label encoded as a 1 will be represented with a binary vector [0, 1] where the first index is marked with a value of 1. If we had the sequence: *red, red, green*, we could represent it with the integer encoding: 0, 0, 1. And the one hot encoding of: [1, 0], [1, 0], and [0, 1].

Why Use a One Hot Encoding? A one hot encoding allows the representation of categorical data to be more expressive. Many machine learning algorithms cannot work with categorical data directly. The categories must be converted into numbers. This is required for both input and output variables that are categorical. We could use an integer encoding directly, rescaled where needed. This may work for problems where there is a natural ordinal relationship between the categories, and in turn the integer values, such as labels for temperature *cold, warm, and hot*.

There may be problems when there is no ordinal relationship and allowing the representation to lean on any such relationship might be damaging to learning to solve the problem. An example might be the labels dog and cat

In these cases, we would like to give the network more expressive power to learn a probability-like number for each possible label value. This can help in both making the problem easier for the network to model. When a one hot encoding is used for the output variable, it may offer a more nuanced set of predictions than a single label.

We can use libraries such is scikit-learn and keras to encode a categorical feature for ML.

5.2 F1 score, precision, and recall

Refer to this source.

Precision (P) is defined as the number of true positives (T_p) over the number of true positives plus the number of false positives (F_p): $P = \frac{T_p}{T_p+F_p}$.

Recall (R) is defined as the number of true positives (T_p) over the number of true positives plus the number of false negatives (F_n): $R = \frac{T_p}{T_p+F_n}$.

These quantities are also related to the ($F1$) score, which is defined as the harmonic mean of precision and recall: $F1 = 2\frac{R*P}{P+R}$.

The F1 score can be interpreted as a harmonic mean of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0.

5.3 Overfitting vs underfitting

This is a phenomenon called overfitting, where a model matches the training data almost perfectly, but does poorly in validation and other new data. On the flip side, if we make our tree very shallow, it doesn't divide up the houses into very distinct groups.

At an extreme, if a tree divides houses into only 2 or 4, each group still has a wide variety of houses. Resulting predictions may be far off for most houses, even in the training data (and it will be bad in validation too for the same reason). When a model fails to capture important distinctions and patterns in the data, so it performs poorly even in training data, that is called underfitting.

Since we care about accuracy on new data, which we estimate from our validation data, we want to find the sweet spot between underfitting and overfitting. Visually, we want the low point of the (red) validation curve in the figure below.