SL00: Supervised Learning

- Supervised learning → Function approximation.
- Unsupervised learning → Function description.
- Reinforcement learning → Learning from delayed reward.
- Algorithms and data are co-equal.

In a supervised learning setting, the learning algorithm will be provided with instances and labels as training data.

- Induction: Going from specifics to generalities.
- Classification: The classification task can be summarized as taking input x (e.g. images of traffic signs) and assign it to a label y that belongs to a discrete number of labels Y (e.g. STOP, Speed Limit, Keep Right, etc.).
- Regression: The regression task can be summarized as taking input x (e.g. employee's years of experience) and mapping it to a real continuous value (e.g. expected salary).

Terminology:

- Instances: Inputs (Features).
- Concept: A set of functions that make good candidates to map inputs to outputs.
- Target concept: The actual function that can map inputs to outputs. In other words, the concept that we're willing to model.
- Hypothesis class: A subset of the concept set of functions that is easier to search through for the target concept.
- 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.

SL01: Decision Trees:

A sequence of decision making points (nodes) applied to every instance to assign it to a specific class.
Example: Classifying vehicles into Sedans and Trucks.

Representation vs Algorithm:

- A decision tree is a structure of nodes, edges and leaves that can be used to represent the data.
  1. Nodes represent attributes (vehicle length, vehicle height, number of doors, etc.).
  2. Edges represent values. (A specific value for each attribute)
  3. Leaves represent the output (A classification decision).

- A decision tree algorithm is a sequence of steps that will lead you to the desired output.
  1. 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 be cause overfitting.
  2. Ask a question about this attribute.
  3. Follow the correct answer path.
  4. 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.

Expressiveness:

- Decision trees can basically express any function using the input attributes.
- For Boolean functions (AND, OR, etc.), the truth table row will be translated into a path to a leaf.
- Decision tree hypothesis space is very huge ($2^{(2^n)}$), this why we need to design algorithms to efficiently search the hypothesis space for the best decision tree.

ID3 Algorithm:

- ID3 builds decision trees using a top-down, greedy approach.
- Pseudocode:
  1. Pick the best attribute A.
  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 descendent of node.
  5. Sort training examples to leaves.
  6. If (examples perfectly classified) {
     Stop
     } else {
     Iterate over leaves
     }
- Finding the best attribute:

  We can find the best attribute by calculating the information gain of each particular attribute, which is the amount of information gained by picking this attribute.

  Mathematically, the information gain expresses how well an attribute splits the data into groups based on classification. 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.

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

  - S is the set of training examples.
  - A is an attribute.
  - v is possible values for the attribute.
  - H is the entropy (Randomness) defines as:

$$Entropy(S) = -\sum_{v} P(v) \log P(v)$$

  - $P(v)$ is the probability of seeing the value v.

  So ID3 is an algorithm of maximizing over the information gain (minimizing Entropy).

- ID3 Bias:
  We have two kind of biases:
  1. Restrictive bias: Describes the hypothesis set space H that we will consider for learning.
  2. Preference bias: Describes the subset of hypothesis n, from the hypothesis set space H $(n \in H)$, that we prefer.

  The ID3 algorithm will prefer these decision trees:
  1. Good splits near the top of the tree.
  2. Trees that gives the correct answers.
  3. Prefers shorter trees vs. longer trees, which is a natural result of preferring good splits near the top.
- We say that ID3 uses a greedy approach because 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. This means that while the algorithm will, in most cases, come up with a good decision tree, a better one may exist.
- ID3 algorithm returns an optimal decision tree, but as the size of the training data and the number of attributes increase, it becomes likelier that running ID3 on it will return a suboptimal decision tree.

Other considerations:

- What if we have continuous attributes?
  - Information gain for a continuous attribute would be the full entropy of S. This is because there is only one observation per attribute value, which means that for each value of the attribute, the training data agrees unanimously on a classification and the entropy for each attribute value is 0.
  - To solve this issue, we split the attribute range into equal intervals.
  - A modified ID3 can be used to find the best split.
- Does it make sense to repeat an attribute along a path in the tree?
  - No if this attribute has a finite value.
  - Continuous attributes can be tested with different questions.
- When do we stop?
  - When everything is classified correctly.
  - No more attributes.
  - When you have noise, don't overfit:
    - → Not too big tree.
    - → We can use cross-validation to prevent overfitting.
    - → Use a validation set and stop when the error is low enough.
    - → Build the whole tree, the prune it. Pruning means removing sections of the tree that provide little power in classifying instances.
- What if it's a regression problem?
  - Information Gain won't be efficient because it's not clear how to measure information over continuous values.
  - We can measure variance, which is the deviation of a variable from its mean value.
  - We need to decide what to report on the leaves (output). Average might be an option.

- If we have continuous variables or discrete variables with many possible values, we might end up assigning high information gain to these variables but producing a decision tree that might not generalize well.
  - A solution to this would be to use the GainRatio instead of information gain:

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

  → $SplitInformation(S,A)$ measures how broadly partitioned an attribute is and how evenly partitioned those partitions are.

$$SplitInformation(S,A) = -\sum_{v} \frac{|S_v|}{|S|} \log \frac{|S_v|}{|S|}$$

  → If we have an attribute with many possible values, each of which occurs few times, the fraction $\frac{|S_v|}{|S|}$ will have a small value. This will affect the attribute's chances of being the best attribute after an iteration of the ID3 algorithm and help to avoid overfitting.

Decision Trees Complexity:

- The run time cost to construct a balanced binary tree is $O(n_{samples} n_{features} \log n_{samples})$.
- The query time is $O(\log n_{samples})$.
- Assuming that the subtrees remain approximately balanced, the cost at each node consists of searching through $O(\log n_{features})$ to find the feature that offers the largest reduction in entropy. This has a cost of $O(n_{samples} n_{features} \log n_{samples})$ at each node, leading to a total cost over the entire trees (by summing the cost at each node) of $O(n^2{}_{samples} n_{features} \log n_{samples})$.

Advantages and Disadvantages of Decision Trees:

- 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.