Decision Tree Learning

An implementation and improvement of the ID3 algorithm.

CS 695 - Final Report

Presented to

The College of Graduate and Professional Studies

Department of Computer Science

Indiana State University

Terre Haute, Indiana

In Partial Fullfilment

of the Requirements for the Degree

Master of Science in Computer Science

By

Rahul Kumar Dass

May 2017

Keywords: Machine Learning, Decision Tree Learning, Supervised Learning, ID3, Entropy, Information Gain.

Decision Tree Learning

An implementation and improvement of the ID3 algorithm.

Rahul Kumar Dass

CS 695 - supervised by Professor László Egri.

Abstract

The driving force behind the evolution of computing from automating automation [3] to automating learning [8] can be attributed to Machine Learning algorithms. By being able to generalize from examples, today's computers have the ability to perform autonomously and take critical decisions in almost any given task. In this report, we briefly discuss some of the foundations of what makes Machine Learning so powerful. Within the realm of supervised learning, we explore one of the most widely used classification technique known as Decision Tree Learning. We develop and analyze the ID3 algorithm, in particular we demonstrate how concepts such as Shannon's Entropy and Information Gain enables this form of learning to yield such powerful results. Furthermore, we introduce avenues through which the ID3 algorithm can be improved such as Gain ratio and the Random Forest Approach [4].

Acknowledgements

There are many people that I would like to express my sincerest gratitude, without which this research course would not have been completed. Firstly, I would like start by thanking my research advisor Professor László Egri for his support, guidance and giving me the opportunity to study Machine Learning without any prior education or experience in the field. I would also like to thank some my faculty members Dr. Geoffery Exoo, Dr. Jeffery Kinne and Dr. Arash Rafiey for their constant support and feedback, especially during the implementation phase of research module at Indiana State University. I would like to also thank my peers Samuel Rider-Leck and Ali Salman for their advice and input especially during the coding phase of ID3 algorithm. I also wish to thank the College of Graduate and Professional Studies for providing me with a graduate assistantship to support my studies, without which my education at Indiana State University would not be possible. Finally, I would like to thank my mother, Mita Dass and my girlfriend Suchismita Dutta for their unconditional support, love and patience.

To my parents

Contents

Introduction

In order to motivate the concept of machine learning (ML), consider three seemingly distinct situations: 1. While uploading a new picture onto the Facebook app on our phone, the app automatically "suggests" tagging people that are in our friend list, with their respective names hovering over their faces. 2. While browsing an e-commerce website such as Amazon or Netflix for products, the system in turn "recommends" new, unseen products based on a persons' preferences. 3. Self-driving cars being able to drive autonomously along highways, and is able to adapt according to an unpredictable environment. What do all these three have in common? They all incorporate ML algorithms that learn from data. The notion of a computer learning, i.e. the ability to program itself, has rapidly become attractive that in addition to the above-mentioned examples, applications such as financial forecasting, medical diagnosis, search engines, robotics, gaming, credit evaluation etc. have now become tremendously dependent on ML.

In Chapter 1, we establish a general overview of the foundations of ML; by stating the learning problem, the three key components to every ML algorithm, and the three types of learning. As there are many different types of ML, in this report we will focus on the most popular and extensively studied: classification [3, 2]. In Chapter 2, we introduce one of the ways to represent a ML program, i.e. a classifier known as Decision Trees. We dive deeper into Decision Tree Learning by developing and analysing the ID3 algorithm, in particular key concepts such as Shannon's Entropy and Information Gain, in Chapter 3. As part of CS 695, we have also implemented the ID3 algorithm in Python3, and obtained a correctly working Decision Tree model that is able to predict if a person plays tennis or not, using a toy data set [5].

In Chapter 4, we discuss two methods that can be used to improve the effectiveness of the ID3 algorithm: Gain ratio and the Random Forest Technique. Finally, we conclude this report by discussing future goals that we would like to pursue with regard to Decision Tree Learning and ML, in general.

Chapter 1

Machine Learning in a nutshell

In this chapter we provide an overview to ML. First, we discuss how ML is different to the traditional form of every ML algorithm boils down into three components. We state the three major types of learning algorithms.

1.1 Is Machine Learning magic?

What makes computers so powerful is the ability to automate tasks that were done by people. This resulted to computers becoming drastically cheaper, and we are able to accomplish tasks that were once thought of as impossible. But while programming has given us these capabilities, we continue to face several challenges such as debugging, implementation, scaling, to name a few. Yet consider at a higher level, what if we were able to tell a computer what to do and it will *learn* and be able to program by itself by just looking at the data? Now, that seems quite a proposition.

But, is this possible? Pedro Domingos [3], one of the leading researchers in ML, illustrates the fundamental differences between traditional programming and machine learning. Consider the figure below:

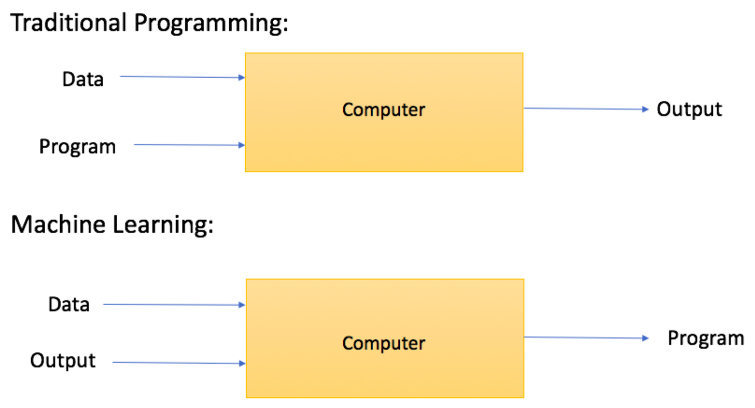


Figure 1.1: Traditional programming vs machine learning

Within the traditional context as we all know, as input to the computer we provide the data and the program. The computer provides the programmer with an output after computation. But, within the ML paradigm, the data and the output are now inputs to the computer. And, the output of a ML algorithm is *another algorithm.* Think of ML as being the ”inverse” of traditional programming.

So to answer the question: no, ML is not magic. Domingos [3] provides a compelling analogy to farming! Consider the following:

1. Seeds = learning algorithms. A seed is simple in nature, yet a whole tree can grow out of it. In ML, learning algorithms are quite simple, but yields very powerful, complex results.
2. Fertilizer + water = Data. Just as fertilizers and water provide crops with nutrients and help them to grow. In ML, the data is what enables learning algorithms to grow.
3. Farmer = Programmer
4. Crops = Programs (ML output)

1.2 Every ML algorithm: three keys

Given its success in practice and relevance in so many different applications, it should not be surprising that there are thousands of ML algorithms. However, the fundamental ingredients to every ML algorithm remains constant, it is the manner by which they are combined together based on the particular problem at hand that results in there being so many of them [2, 3]. The key components are: representation, evaluation and optimization.

1.2.1 Representation

If a learning algorithm outputs a program, first thing we must choose is what the language is that the program will be written in. For Human programmers, the choice could be: C, Python, Java etc. For ML, the choice could be:

1. Decision Trees (nested if/else statements) - simple.
2. Instances – simplest ML program: “remember what you saw” (memory-based learning)
3. Graphical models (Bayes/Markov networks) – inside the brain of self-driving cars; Google ads uses Bayesian networks for user prediction (if advert will be clicked on or not).

4. Support Vector Machines – related to instances; kernel = measures the similarity between data points

5. Model ensembles = take a bunch of the above-mentioned ML programs or variations of it and combine them. Eg: Netflix prize winner, Kaggle competitions.

These are only a few examples of the many different ML representations that are available. To go through them all would be beyond the scope of this report.

1.2.2 Evaluation

Say we choose Decision Trees as the ML program. Need to ask, "what is the *best* one that will model the phenomenon that I'm interested in?" "What is the best decision tree to decide if a person is a good credit risk or not?" To do so, we need to find a way to "score" our programs and in the case for Decision Trees, typically we would be after *accuracy* as a good measure.

1.2.3 Optimization

Once we have chosen a representation model and an evaluation measure, now there is the search process by which we optimize that measure: *how do we find the most optimal decision tree?* Naively, we could try all possible decision trees – brute force. But, there are more decision trees than atoms in the universe so that's not going to work. Thus, we do optimization (or in AI, "search"). There are three types: combinatorial, convex and constrained. For the purposes of this report, we'll be focusing on combinatorial optimization. For example, doing a greedy search - try a bunch of things, pick the best one. Keep going, until the single biggest gain is found, irrespective of what might happen in the long run. This is the approach used in discrete models like decision trees (which are essentially graphs with discrete sections).

1.3 Learning: three types

Next, we'll briefly give an overview of the three major types of learning: supervised, unsupervised and reinforcement learning.

1.3.1 Supervised learning

The basic principle is *the training data includes desired outputs.* Think of this as a "learning with a teacher" - somebody has already labeled what the "right" answer is. Eg: Somebody has already labeled which emails are spam and which aren't; Somebody labeled which x-rays show cancer and which don't. Thus, we know what to learn. Supervised or inductive learning is the most mature (widely studied and used in practice) kind of learning, and forms the basis of decision trees learning.

1.3.2 Unsupervised learning

In sharp contrast to inductive learning, *the training data does not include desired outputs* within an unsupervised learning environment. This is a much harder but the most important kind of learning, in the long run. As a useful analogy, think of how babies learn how to walk on its own - this would be an instance of unsupervised learning. If a parent tells the baby, "that's a chair" or "that's a table" – that's supervised learning.

1.3.3 Reinforcement learning

The idea is *rewards from sequence of actions*. Out of the three types of learning, this is the most ambitious - closest to how humans learn. Inspired by psychology, you are an agent in the world, going around and doing things, no one's telling you what is the right thing or wrong – but every now and then, the agent gets a reward. According to Domingos [3] reinforcement learning will be greatly implemented in order to solve problems using AI (i.e. solve problems that humans cannot). Although not widely used in industry, in 2016, it achieved major success with Google's DeepMind AI program AlphaGo defeating the Go World Champion Lee Sedol 4-1 in a 5-match series [1, 10].

1.4 The learning problem: an overview

Let us revisit Inductive learning and describe it more formally. Given examples of a function $(X, F(X))$ pair where X is the input of vector values that are either continuous or discrete. Eg: Symptoms of a patient (temp, blood pressure, glucose levels etc. And, $F(X)$ is the value of the function for that element. Eg: Diagnosis of the patient - "yes, the patient has diabetes" or "no, the patient does not have diabetes". The crux is how can we predict $F(X)$ or "generalize" for new examples - data that we have not seen before?

There three types of supervised learning that needs to be mentioned:

1. Discrete $F(X)$: classification. Eg: Computer vision system that wants to label the object that is seen (chair or table, say). As we are predicting the <u>class</u> of the object, hence this is known as a classification problem.
2. Continuous $F(X)$: regression. Eg: Predicting the gas mileage of a car given its characteristics?

3. $F(X)$ = Probability(X): probability estimation (if that is what we are predicting). Eg: Google might be interested in learning the probability of a user clicking on a particular ad? A harder problem would be predicting several things at the same time.

This brings us to the conclusion this chapter by stating the quintessence of the *learning problem* via the simple example below:

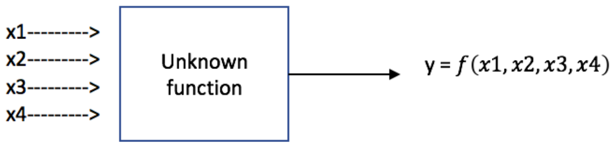


Figure 1.2: The Learning Problem

Suppose we are given a black box that represents a Boolean function. The input to this function are four Boolean inputs, $x1, x2, x3, x4$, each with a value of either 0 or 1. The output is y a Boolean function of four Boolean inputs, where $y = f(x1, x2, x3, x4)$. The *goal is to figure out what that function y is.* One possible solution is we can check all possible values for the four-Boolean values and read off the respective value for y.

Example	x_1	x_2	x_3	x_4	y
1	0	0	1	0	0
2	0	1	0	0	0
3	0	0	1	1	1
4	0	0	0	1	1
5	0	1	1	0	0
6	0	1	0	0	0
7	0	1	0	1	0

Table 1.1: An unknown Boolean function with 7 examples, each comprising of four Boolean inputs.

Each set of 4-values is known as an example. For these seven examples, we know what the outputs are - so what is y? If we are given an additional example set, what would the corresponding value for y be? This, in essence, the inductive learning problem.

Chapter 2

Decision Tree Learning

2.1 Problem description and goals

As stated earlier, decision trees is a way to represent a ML program. It is easiest to get started with a concrete example. We are tasked with predicting (classifying) if Roger plays tennis at any given day. In order to do so, we have observed Roger for two weeks and is outlined using the following table:

Day	Outlook	Humidity	Wind	Play
D1	Sunny	High	Weak	No
D2	Sunny	High	Strong	No
D3	Overcast	High	Weak	Yes
D4	Rain	High	Weak	Yes
D5	Rain	Normal	Weak	Yes
D6	Rain	Normal	Strong	No
D7	Overcast	Normal	Strong	Yes
D8	Sunny	High	Weak	No
D9	Sunny	Normal	Weak	Yes
D10	Rain	Normal	Weak	Yes
D11	Sunny	Normal	Strong	Yes
D12	Overcast	High	Strong	Yes
D13	Overcast	Normal	Weak	Yes
D14	Rain	High	Strong	No

Table 2.1: Observations of Roger playing Tennis

We have observed that Roger plays nine times and does not play five times. The attributes in the data set contribute to whether Roger plays on a given day.

Goal: Build a mechanism such that on Day 15 (unseen example), it would be able to classify if Roger plays or not?

2.2 How does it work?

Decision trees works in a ”divide and conquer” approach and while implementing recursion. The general idea is:

1. Split the data set (eg: 2.1) into disjoint subsets based on all values per attribute.
2. check if those subsets are pure (i.e. if the target value are all ”yes” or ”no”) or not?
   (a) If yes, then STOP! We do not need to make any further decisions.
   (b) If no, then we repeat the process, minus the attribute that we just considered.
3. Continue the recursive call, until we are left with pure sets.

As a side note, it is acceptable use this approach considering that the toy dataset model in question only has 14-examples. In the real-world, the training datasets that practitioners interact with would be orders of magnitude in size. In those situations, the standard practice is to partition the datasets into two subsets:

- A larger portion: training dataset (used for training the decision tree).
- A smaller portion: testing dataset (used for evaluating the accuracy of decision tree).

Overall goal: Once we have a working decision tree (data structure), when we are given an unseen example/instance, we need to be able to predict the target. This is accomplished by traversing through the tree.

2.2.1 Example - Outlook attribute

Let us look at an example. From 2.1, consider the ”Outlook” attribute. It has three distinct values: Sunny, Overcast and Rain.

1. First, we split the training dataset into three-disjoint subsets.
2. Look at the target attribute to see if Roger plays consistently or not?
   (a) If yes, we can conclude that for that value of the attribute, it leads to John playing or not playing.
   (b) If no, we repeat the process of splitting those disjoint subsets further, but removing the “Outlook” attribute value from those examples.
3. Continue this recursively such that there is no uncertainty of whether John plays or not – left with pure sets in the end.

Chapter 3

ID3 algorithm

3.1 Building the decision tree

The Iterative Dichotomiser 3 (ID3) algorithm was discovered by Ross Quinlan [6], which he later extended as a generalized version to the C4.5 algorithm [7]. The ID3 is a ML algorithm that is used for building the (decision) tree data structure. As with most tree algorithms, it a recursive algorithm and is delineated as follows:

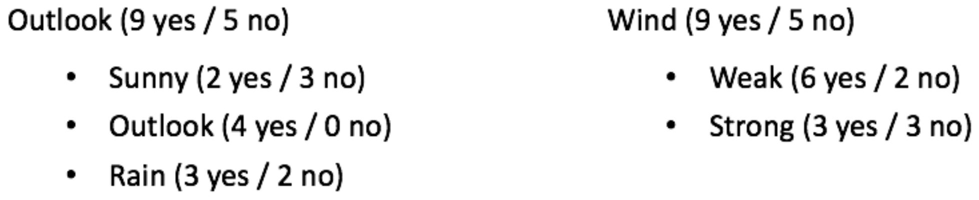
1. Split (node, examples)*
2. A $\leftarrow$ the "best" attribute for splitting the examples
3. Decision attribute for this node $\leftarrow$ A
4. For each value of A, create a new child_node
5. Split training examples to child nodes
6. For each child_node:
    - If subsetexamples is pure: STOP
    - else Split(child_node, subset)

*Each node in the tree contains the decision node (attribute), and children (subset of training examples based on the values of the decision node).

Notice that in the second step of the algorithm there is a reference to a quantity called the "best" attribute that is used to split the training set. In the next section, we formally discuss how this quantity is computed through a statistics measure of purity known as Shannon's entropy.

3.2 Shannon's Entropy

Naively, we can consider splitting the training dataset on any attribute, this will however yield to a different partitioning of the dataset compared with any other attribute. *Crux*: How to decide if one partitioning is "better" than the other? Let us look at an example, keep in mind that according to 2.1, prior to any splitting, Roger played tennis on 9 instances and did not play on 5:



Outlook (9 yes / 5 no)

- **Sunny (2 yes / 3 no)**
- **Outlook (4 yes / 0 no)**
- **Rain (3 yes / 2 no)**

Wind (9 yes / 5 no)

- **Weak (6 yes / 2 no)**
- **Strong (3 yes / 3 no)**

Figure 3.1: Splitting 2.1 on two different attributes.

Looking at 3.1, we can immediately make several observations. Between Outlook and Wind, Outlook appears to be the better split (overcast = pure set). However, if we only consider Wind on its own, we can deduce that as the results from the value "Weak" has a lot more "yes's" than "no's", thus there is a higher probability for Roger to play. But with the value "Strong", the split is 50-50, gives us no information as to which target value is more likely.

In general, we want splits that are *heavily biased* towards a particular target value. The reason being for a subset that is mostly positive or negative, we can then predict what is the mostly likely outcome. So, for two distinct cases:

- Outlook – Overcast: (4 yes/0 no) $\Rightarrow$ 100% certain (pure set).
- Wind – Strong: (3 yes/3 no) $\Rightarrow$ completely uncertain 50% (impure/mixed set)

Thus, with a pure set Outlook emerged as the "better" attribute to split on then Wind. So, the first step towards determining which attribute, in a real-world dataset with potentially N different attributes, would be considered the "best" - is figuring out a good measure of the *purity* of subsets. Fortunately, in the field of statistics there is such a measure called Shannon's entropy [9] that will help us accomplish this. Its definition:

$$H(S) = -p_+ log_2 p_+ - p_- + log_2 p_- \tag{3.1}$$

where S: subset of examples
p_+/p_-: proportion of $S_{examples}$ that have positive/negative target values.

With the log_2 representation, it enables us to interpret the value as the number of bits needed to overcome uncertainty in determining if an item is positive or negative.

- Given a subset $S_{examples}$ that we know is completely positive. If we pick any example at random - we would need 0 bits of information to determine the target value for that particular example.

  Eg: Consider the pure subset (4 yes/0 no):

$$H(S) = -\frac{4}{4} log_2 \frac{4}{4} - \frac{0}{4} log_2 \frac{0}{4} = 0 \tag{3.2}$$

- Given a subset $S_{examples}$ that we know is perfectly impure $(3p_+/3p_-)$. If we pick any example at random - we would need 1 full bit of information, as we have no idea if that particular example's target value is positive or negative.

  Eg: Consider the pure subset (4 yes/0 no):

$$H(S) = -\frac{3}{6} log_2 \frac{3}{6} - \frac{3}{6} log_2 \frac{3}{6} = 1 \tag{3.3}$$

In conclusion, if we have a pure subset, we obtain the lowest entropy value of zero. If we have a perfectly impure subset, we get the maximum entropy value of 1. Depending on how impure the subset is, the entropy value can be in a *range* of $H(S) \leq 1$. While the notion of a "fractional bit" is unusual in Computer Science, it justifies the uncertainty of the target value for a given subset.

3.3 Information Gain

So far, we have computed the entropy for any one given subset. Note that for each node in a decision tree, there may be several children $\Rightarrow$ multiple subsets. Thus, we need to take an <u>average</u> of the entropy for all subsets per node. We define information gain as:

$$Gain(S, A) = H(S) - \sum_{V \in Values(A)} \frac{|S_V|}{|S|} H(S_V) \tag{3.4}$$

where A: attribute (node)
V: possible values of A
S: set of examples X
S_V: subset of $X_A = V$

In Computer Science, the <u>difference</u> between the entropy before the split $H(S)$ and after the split $\sum_V H(S_V)$ is the information gain. Note that the quantity $\frac{|S_V|}{|S|}$ represents the weight of the subset with value V with respect to the initial dataset before any splitting. As we are computing an average of the entropies for the subsets per node, equation 3.4 includes the weight in order to give <u>more weight</u> to larger subsets. This makes sense because say $S_{examples}$ has only one example (singleton set), this is pure by definition. However, in real-world large datasets, singletons represent a small fraction of the data. Instead, the formulation of information gain rewards splits that produce pure subsets for a large number of examples.

Bring it all together now, by calculating the information gain for every attribute, the one with the maximum gain after splitting S from the root node is the "best" attribute.

In conclusion, by taking every A that we have in our data, we compute the information gain for that A. We select the A that has the highest information gain. We implement this approach because that A will reduce our uncertainty the most and it will lead to the purest possible split out of all the As. If we have mixed sets as children, we then recursively compute the information gain (minus the A we just looked at from S_V) [4].

Chapter 4

Improving ID3

4.1 Always get a perfect solution?

With the methodology described in Chapter 3, one may ask if this approach always finds a perfect solution? Yes and no!

Yes: if we run the ID3 algorithm to build a decision tree, it will always partition the dataset perfectly. This is guaranteed, as the algorithm will continue splitting the data until we get singletons - (pure, by definition).

No: obtaining singletons more often than not implies that we are traversing too deep into the tree and this is something that we do not want. When splitting the data to the point of getting singleton subsets – we don't really have a lot of confidence in the predictions. Any unseen example that falls into a branch with a singleton leaf, will be pure, only because there was a single instance in the training data that followed the same branch. This "lack of confidence" is noticed when we see a graph detailing the performance (accuracy) of decision trees with respect to their size.

With the recursive nature of the ID3 algorithm, the tree will continue growing deeper and deeper. We could consider "cutting the tree" or "stopping the algorithm" at a certain point, this is the notion of pruning and is an area where the author would like to investigate further as a future project. As an example, consider the figure :

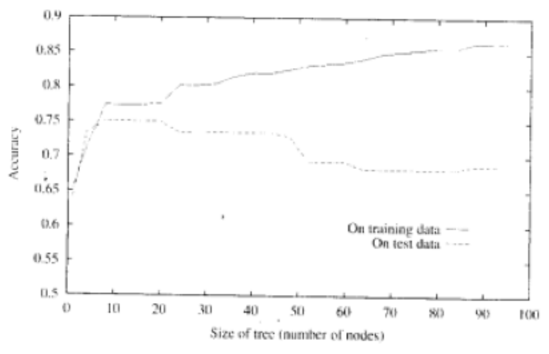


Figure 4.1: Size of tree (nodes) vs Tree Accuracy [5]

From 4.1 we can see that as the tree grows deeper with more training datasets, the greater its accuracy - at some point, it will reach 100%. With test data, initially the performance does quite well. But, after some point, the tree drops in performance – it is too biased to the training dataset and is unable to correctly predict unseen instances (over-fitting). By capturing the *noise* in the training data, this noise is not repeated in the test data, resulting to drop in the tree's accuracy.

In the next two sections, we will investigate two methods by which the ID3 algorithm can be improved upon: Gain ratio and the Random Forest Technique.

4.2 Gain ratio

In Chapter 3, we introduced the notion of information gain. While as a measure in determining the "best" attribute, there is a potential flaw in its logic. Notice that for 2.1 dataset, when we investigated entropy and information gain we did not include "Day" as one of the attributes. If we had done that, then for this dataset, we would have found the "holy grail" of decisions.

This would be the case as we would have found an optimal split because by choosing "Day", each of its values are singletons and therefore are perfectly pure! If we pick any random example, we would not even need to consider any of the other attributes. However, this would not be particularly useful because when the tree is faced with a new, unseen example, eg: Day15 - the tree does not have a branch for that instance.

In general, when dealing with very large datasets, as information gain by nature is greedy it would latch on to such attributes and put them as high nodes in the tree. As a consequence, the resulting predictions from such a tree would be poor for any test data. In order to resolve this issue, we introduce the concept of Gain ratio.

$$SplitEntropy(S, A) = -\sum_{V \in Values(A)} \frac{|S_V|}{|S|} log_2 \frac{|S_V|}{|S|} \tag{4.1}$$

$$GainRatio(S, A) = \frac{Gain(S, A)}{SplitEntropy(S, A)} \tag{4.2}$$

The general idea is we take the information gain of A as normal, but then we normalize it using the SplitEntropy. 4.2 is essentially informing us if A gives a lot of small subsets or not? Comparing this with 3.4, 4.2 does not look the positive and negative values of p, instead it looks at the split itself. After computation, if we obtain high values then it implies lots of small subsets or low values implying bigger subsets, the latter being much more desirable [4].

4.3 Random Forest Algorithm

Another method that is deceptively simple yet very effective is using the Random Forest approach [4] in conjunction with the ID3 algorithm. The general idea is as follows:

1. Instead of growing a single tree for S, we grow K-different trees, by randomizing the input. So from S, take a random subset S_r
2. Use S_r to learn/build a decision tree (run ID3, with no pruning)
3. When splitting, pick a subset of the total number of attributes: A_r
4. Compute gain based on S_r instead of the full dataset S.
5. Repeat K-times as we are using S_r different subsets of data, and A_r different subsets of attributes.
6. So for new example, run it through all K-trees; each tree will give a classification; take the majority vote!

Chapter 5

Conclusion

In conclusion, in this report we first introduced the incredibly versatile and powerful area in Computer Science known as Machine Learning. We explored the key difference between the traditional programming and ML paradigms. Namely that while in the former, a program is one of the inputs to a computer and after computation, we get an output from the computer, in ML, the computer takes in both the data and the output as inputs and gives another program as the output. We explored that with an continual increasing number of applications where ML algorithms are becoming not just useful but the standard practice; every ML algorithm has three components: representation, evaluation and optimization. For the purposes of this report, decision tree learning is our chosen representation that is used to evaluate classification problems and its effectiveness is based upon how accurate it is able to do so. Decision trees implement a form of combinatorial optimization, namely doing a greedy search of which attribute in a dataset yields the maximum gain. We also explored the three primary types of learning: supervised, unsupervised and reinforcement learning and ended our introduction to ML by discussing the learning problem.

We introduced decision tree learning using Mitchell's [5] classic prediction problem of determining if a person plays tennis or not, depending on weather conditions. Despite being no where near as complicated as datasets used in practice today, the author found it particularly effective in motivating the ID3 algorithm that builds the decision tree data structure.

We delineated how the ID3 algorithm is formulated and went into great detail regarding concepts such as Shannon's Entropy and Information Gain. Finally, upon understanding how a decision tree is formulated, we then further investigate if the output of the ID3 algorithm always yields perfect solutions. We learned that according to 4.1 when it came to training a decision tree, due to the recursive nature of the algorithm the tree would continue splitting until singletons were obtained. While this does fit the bill when asking for a perfect solution, this may not be what we want as singletons do not help us when making predictions for massive datasets.

Finally, we concluded this report by outlining two approaches that could help improve the ID3 algorithm. First, by understanding that while information gain helps us identify the "best" attribute to split our datasets, it may not give us desirable results based on its inherent greedy nature. We resolve this issue by introducing the notion of Grain ratio. Secondly, we outline the Random Forest Algorithm that extends the ID3 algorithm by creating K-trees, that are created/trained using subsets of examples/attributes from initial dataset, instead of just one. When making a prediction we input the unseen example into all K-trees and classify the example based on the majority output of all K-trees.

Bibliography

[1] DEEPMIND. Alphago, 2017.

[2] DOMINGOS, P. A few useful things to know about machine learning. *Communications of the ACM*, 10 (2012), 78–87.

[3] DOMINGOS, P. University of washington - csep 546: Data mining and machine learning, 2016.

[4] LAVRENKO, V., AND GODDARD, N. University of edinburgh - introductory applied machine learning, 2015.

[5] MITCHELL, T. *Machine Learning.* McGraw-Hill Education; 1 edition, 1997.

[6] QUINLAN, J. R. Induction of decision trees. *Machine Learning 1*, 1 (1986), 81–106.

[7] QUINLAN, J. R. C4.5: Programs for machine learning. *Morgan Kaufmann Publishers* (1993).

[8] SHALEV-SHWARTZ, S., AND BEN-DAVID, S. *Understanding Machine Learning: From theory to algorithms.* Cambridge University Press; 1 edition, 2014.

[9] SHANNON, C. E. A mathematical theory of communication. *Bell System Technical Journal*, 27 (3): (1948), 379–423.

[10] SILVER, D., AND HUANG, A. Mastering the game of go with deep neural networks and tree search. *Nature*, 529 (2016), 484–489.