Changelog:
-- 8 Nov 2016

Att: The following text is transcribed from the recordings about DTs, that were made in 2015 and that you find on Scalable Learning. The slides on the course websites have been updated in 2016. You might experience a mismatch between recordings 2015 and slides 2016. Students who attended the Math course (spring 2016) have already studied entropy and surprisal (same slides, same recordings). Slides 2016 will be used for new recordings in the future.

Lecture 3: Transcripts - Decision Trees (1)

Decision Trees 1

1. **Part 1a**: In this video clip we are going to talk about a simple and intuitive learning model: the decision tree.
2. There will be 2 lectures on decision trees. In today's lecture, I will explain how a decision tree works and I will cover some basic characteristics of this model, such as Greediness, the Divide and Conquer notion , the Inductive Bias, the Loss function, the Expected loss, the Empirical error, and at the end we will summarize the induction.
3. We said previously that we could simplify the concept on learning in the field of ML by saying that we want to make informed guesses about the future. The past is represented by the examples stored in the training set, and the future is represented by the unseen examples. We can evaluate the generalization ability of our learner by using a test set. In the figure we have classified examples of iris flowers divided into three classes (setosa, versicolor, virginica), each example is represented by measurements. The purpose of a machine learning model would then be to guess correctly the class of a previously unseen iris flower based only on its measurements, that might differ in some respects from the measurements stored in the training set. So we want to make a good prediction based on our previous experience of irises. Our experience is formalized in the dataset.
4. Now, let's make a more specific example by using the same problem that is presented in Daume's book. Our problem is now to predict if a student will like a course or not based on his/her ratings on previous courses. We could make predictions by asking yes-no questions to the student. For instance, does the new course belong to the Systems program? Has the student liked most previous Systems' courses, etc.? And we could build a diagram, a tree-like diagram like the one you see on the slide.
5. When we build our supervised decision tree learning model, we do not ask questions directly to the students. Instead, we use training data in order to answer the questions. Essentially, we have a dataset similar to that on the screen where each row is an example paired with the correct answer. In the dataset on the screen, the column Rating is the class. Interpret the classes as Like (meaning that the student liked the course) if the rating is 0, +1, +2, and Hate if the student did not like the course and ranked it using -2 and -1.
6. So the ratings in this specific dataset are the class labels, the column names are questions and they are our features, the responses, yes and no, are the feature values. So we have here a feature representation that we assume is useful to solve our classification problem. With this data, we could build many possible trees. Since

we do not want to spend months in deciding which of these possible trees is the best tree, we proceed greedily.

7. Being greedy, in this context means: if you could ask only one question, which question would you ask? Which is the most useful question? One can start depicting the usefulness of questions in histograms. Look at the histograms on the screen. Each histogram shows the frequency of like/hate labels for each possible value of a feature. From these histograms, you can see that asking the first question (that is, is it easy or not?) is not useful because there is no clear divide between yes and no. On the contrary, asking the question "is this a Systems course" (the fourth histogram on the screen) is useful because if the value is no, you can be sure that students liked the course, if the value is yes students hated the course. Now, pick up a random example from this dataset, and ask this question. If you get the answer no, you would possibly be inclined to say that the class label of the example is "like". On the contrary, if you would get the answer yes to this questions, you would be inclined to think the class label is hate. Try and use this feature and our assumptions to make informed guesses on the examples of the dataset. You will see that you will guess right many times. So, if you choose this feature you can make reliable informed guesses. Repeat the computation for each of the available features, and score them. When you have to choose which feature to consider first, you choose the one with the highest score. In this way you choose the ROOT node of the decision tree.
8. How do we choose subsequent features? Here is where the notion of divide and conquer is applied. When you ask the first question "Is the course a Systems course"? you can partition the data into 2 sets: the no set and the yes set. This is the divide step: you get 2 partitions. In the Conquer step, repeat the same process you have applied to choose the first feature on the examples listed under the no branch and the yes branch of the tree.
9. At one point, we realize that asking additional questions becomes redundant, or that we have run out of questions. In both cases, we create a LEAF NODE and we guess the most prevalent answer based on the training data you are looking at.
10. The goal of the decision tree learning model is to figure out what questions to ask in what order, what answer to predict once you have asked enough questions. The inductive bias of decision trees assumes that the t*hings that we want to learn to predict are more like the root node and less like the other branch nodes*.
11. We will talk more about the basic characteristics of decision trees in the next video clip.
12. **Part 1b**: Welcome back to decision trees part 1
13. Let's now start with an informal definition of the decision tree model. A decision tree is a flow-chart-like structure, where each internal (non-terminal) node denotes a test on an attribute, each branch represents the outcome of a test, and each leaf (or terminal) node holds a class label. The topmost node in a tree is the root node.
14. Let's now formalize the definition. We know that the performance of a learning algorithm should be measured on unseen data. We can use a function to measure the performance and we call it Loss function: The loss function is the price paid for inaccuracy of predictions in classification problems. Loss is this case means "misclassifications or wrong predictions" How "bad" is our system's predictions in comparison to the truth? In particular, if y is the truth and y-hat is the system's prediction, then the function l(y, yˆ) is a measure of error. Note that the loss function is something that we must decide on based on the goals of learning. There are many loss functions that we could use. Let's use the simplest here: the zero-one loss. If y is

equal to y hat, the system's classification is correct so we have 0 errors. If y is not equal to y hat, the classification is incorrect, so we have to count one error.:

$$\textit{Classification:}\ \ \textbf{zero/one loss}\ \ell(y,\hat{y}) = \begin{cases} 0 & \text{if } y = \hat{y} \\ 1 & \text{otherwise} \end{cases}$$

15. Distribution: Now that we have defined our loss function, we need to consider where the data (training and test) comes from. We talked about distribution before and we focussed on normal distribution. We now know that normal distribution is a bell-shaped distribution of data. if we know a priori what your data generating distribution is, our learning problem becomes easier. In this case, we are not making any assumptions about what the distribution D looks like. We are assuming that we do not know what D is. Perhaps the hardest thing about machine learning is that we don't know what D is: all we get is a random sample from it. This random sample is our training data. We can say that the Data Generating Distribution is a probability distribution D over input/output pairs. If we write x for the input (examples/instances) and y for the output (the rating), then D is a distribution over (x, y) pairs. Remember that our problem is guess the rating of an unseen example. A useful way to think about D (Data Generating Distribution) is that it gives high probability to reasonable (x, y) pairs, and low probability to unreasonable (x, y) pairs.
16. Expected Loss: We are given access to training data, which is a random sample of input/output pairs drawn from D. Based on this training data, we need to induce a function "f" that maps new inputs to corresponding prediction. The key property that f should obey is that it should do well on future examples that are also drawn from D. Formally, its expected loss (epsilon) over the distribution (D) with respect to "l" should be as small as possible, meaning that we should minimize the expected loss, meaning that we should make as few error as possible. :

$$\epsilon \triangleq \mathbb{E}_{(x,y)\sim\mathcal{D}}\big[\ell(y, f(x))\big] = \sum_{(x,y)} \mathcal{D}(x,y)\ell(y, f(x))$$

17. Now let's read and anlyse the formulae:
Epsilon is equal by definition to blackboard-bold E sub the pair x y over script D of l of the pair y f of x. All this corresponds to: Sum (big sigma means sum) over all the pairs in script D of x and y times l of y and f of x. This is exactly the weighted average loss over all the pairs x and y in D, weighted by their probability under the distribution D. In practical terms, this formula accounts for the average loss if we draw a bunch of xy pairs for a distribution D.
18. Training error: The difficulty in minimizing our expected loss formula is that we don't know anything about the distribution D. but we know that in our training data we have certain number of xy pairs. So in order to compute our training error epsilon-hat (which is an average, hat indicated an), we divide the expected loss (the formula is explained in the previous slide) by the number of training examples, 1 over capital N.

And we get the formula that you see on the screen: the training error epsilon-hat (the hat means that it is an estimate) is equal by definition to 1 over N of the Sum from n=1 to capital N of "l" of y and f of x. That is, our training error is simply our average error over the training data. The challenge for our learned function needs to generalize beyond the training data to some future data that it might not have seen yet. the training error epsilon-hat is equal by definition to 1 over N of the Sum from n=1 to capital N of "l" of y and f of x.

$$\hat{\epsilon} \triangleq \frac{1}{N} \sum_{n=1}^{N} \ell(y_n, f(\boldsymbol{x}_n))$$

19. The training error is sometimes called empirical error. Remember that terminology can be confusing sometimes. The empirical error can be called the "training error", "test error", or "observed error" depending on whether it is the error on a training set, test set, or a more general set.
    What out! Formulae can be written using different notation styles. For example, the formula on this slide is the formula for the empirical error given by Alpaydin: the empirical error is the proportion of training instances where the predictions of h (the hypothesis = the informed guess) do not match the required values given in big X (the training set). The formula should be read in this way: the empirical error of the hypothesis h given the training set X is the sum of the training instances (small x) where the hypothesis on the class label r fails.

$$E(h|\mathcal{X}) = \sum_{t=1}^{N} 1(h(\boldsymbol{x}^t) \neq r^t)$$

20. Induction: So, putting it all together, we get a formal definition of induction machine learning: Given a loss function l and a sample small d
from some unknown distribution capital D, you must compute a function f that has low expected error epsilon over D with respect to l.

21. Ok. We stop here today. Thank you for your attention.

Termininology

DEFINITION OF 'DISCRETE DISTRIBUTION'
The statistical or probabilistic properties of observable (either finite or countably infinite) pre-defined values. Unlike a continuous distribution, which has an infinite

number of outcomes, a discrete distribution is characterized by a limited number of possible observations. Discrete distribution is frequently used in statistical modeling and computer programming. Also known as a "discrete probability distribution".

BREAKING DOWN 'DISCRETE DISTRIBUTION'
Examples of discrete probability distributions include binomial distribution (with a finite set of values) and Poisson distribution (with an countably infinite set of values). The concept of probability distrubtions and the random variables they describe are the underpinnnings of probability theory and statistical analysis.

Terminology: Ordered Pairs: And here is another way to think about functions: Write the input and output of a function as an "ordered pair", such as (4,16). They are called ordered pairs because the input always comes first, and the output second: (input, output) So it looks like this: (x, f(x)).

Decision Trees (2)

Part 2a

1. Wecome to Lecture 4. In this lecture we will continue the exploration of decision trees.
2. These are the acknowledgements and the references for this presentation.
3. Today we will talk about: Attribute selection, Entropy, Suprisal, Information Gain, Gain Ratio, Rules, Pruning.
4. We said that the problem of constructing a decision tree can be expressed recursively. First we select the root node and make a branch for each possible value. The root node splits up the training set into subsets, into partitions: one for every value of the attribute. Then we repeat the process recursively for every attribute. We stop when the data cannot split any further.
5. Let's make an example with the weather dataset. With this dataset we want to determine whether we can play outside or not given certain weather conditions.
6. There are 4 possibilities, one for each attribute as shown on the screen. Which is the best root node? Which attribute do you think is the best choice? We know that any leaf with only one single class (either yes or no) will not have to be split further and the recursive process will terminate. Since it is recommended to build trees as small as possible, we would like that the recursive process would stop as soon as possible. If we had a measure of purity, that is a measure that tells us which attribute produces fewer splits, we could choose the purest node.
7. The measure of purity that we could use is called the *information* and it is measured in bits. It is calculated based on the number of yes or no associated with each node of the tree, it represents the expected amount of information that would be needed to specify when a new instance should be classified yes or no. What does your intuition say? Which attribute is the best in this context?
8. Trust me for a second and let's say that the best attribute for the root note is outlook. One thing that you could notice is that the outlook branching is the only one in which a node is completely pure (overcast has only yes) and this gives considerable advantage over the other attributes if we want a small tree.

9. If we continue with the application of the same idea, we end up with a decision tree like the one shown on the screen. Ideally the process terminates when all leaf nodes are pure, but there are exception to this. Usually, the process terminates when the data cannot be split any further.
10. Why is the outlook attribute the best attribute to split upon? We said that we prefer small trees, so the best way to achieve this is to choose attributes that produces the "purest" node, node with no splits or fewer splits. So, the information measure measures how pure a branch is.
11. Now a popular measure that helps us choose the best attribute to split upon is a measure called information gain. Information gain increases with the average purity of the subsets. The strategy is then: choose attributes that give the greatest information gain.
12. The information measure should tell us something about the purity or impurity of a node. Now let's consider the kind of properties that an information quantity should have in our scenario. In our scenario the information quantity can be defined by 3 properties: the first property says: When the number of either yes OR no is zero (that is the node is pure) the information is zero. If a node is pure, there is no surprise, we are certain about the nature of the node, we are certain about the outcome. The second property says: When the number of yes and no is equal, the information reaches its maximum because we are very uncertain about the outcome. The 3rd property says that the measure must be useful in a complex scenario, that is the measure should be applicable to a multi-staged decision process. Essentially Entropy is the only function that satisfies all three properties!
13. Information gain is a measure of impurity and it is based on a function called entropy. The notion of entropy comes from information theory. Entropy (also called Shannon entropy) is the expected value (average) of the information contained in each message received. The entropy of the message is its amount of uncertainty; it increases when the message is closer to random, and decreases when it is less random. The idea here is that the less likely an event is, the more information it provides when it occurs. In information theory, 'information' doesn't necessarily mean useful or structured information; it simply describes the amount of randomness of the message. So, when the probability is 0.5, entropy is at its peak of 1.
14. Entropy is based on surprisal. So let's analyse the concept of surprisal first: In information theory, self-information or surprisal is a measure of the information content associated with an event in a probability space. It represents the "surprise" of seeing the outcome (a highly improbable outcome is very surprising). By definition, the amount of surprisal contained in a probabilistic event depends only on the probability of that event: the smaller its probability, the larger the surprisal associated with receiving the information that the event occurred. By definition, the measure of surprisal is positive and additive. If an event C is the intersection of two independent events A and B, then the amount of information at the proclamation that C has happened, equals the sum of the amounts of information at proclamations of event A and event B respectively: $I(A \cap B)=I(A)+I(B)$. You are already familiar with the intersection of probability events.
15. Putting all this together, we can say: that The surprisal I of an event w-sub-n is the log base 2 of 1 over the probability of w-sub-n. Since the logarithms of the fractions are negative, with minus in front of the log of the resulting value becomes positive.

16. And these are the formulas to compute entropy. The 2 formulas on the screen are perfectly equivalent. The notation style is different but the content is identical. You might notice that the minus of the surprisal formula which was placed in front of the log, in the entropy formula it has been moved at the beginning of the expression. The result of this expression will be positive, because, as we said the log of a fraction is negative. Multiplication of 2 minus(es) gives a plus, i.e a positive number. How surprising is an event? Informally, we can formulate this by saying: the lower probability you assign to an event, the more surprising it is, so surprise seems to be some kind of decreasing function of probability. If event 1 has a certain amount of surprise, and event 2 has a certain amount of surprise, and you observe them together, and they're independent, it's reasonable that the amount of surprise adds. That's why we have the big sigma in the lower formula. In the first formula we have the juxtaposition of p-sub-1 up to p-sub-n. Entropy-based measures are commonly used not only for deciding the best node, but for feature selection in general. And we are going to use them shortly in weka.
17. Let's make all this more concrete by giving an example. For the first branch of the first tree, we take the number of yes and no classes at the leaf node: in the first branch we have 2 yes and 3 nos. So we take the fraction/the proportion of the 2 yes over the 5 instances on that branch and multiply by the log of the fraction; and we do the same with the 3 no. On the screen you can see the calculation of the entropy of the sunny branch of the tree that has been built with the outlook attribute. The level of uncertainty is high, so the entropy value is high.
18. Watch out: There are many statements in the literature which say that information is the same as entropy. **Properly speaking:** entropy is a probabilistic measure of uncertainty or ignorance and information is a measure of a reduction in that uncertainty. The *information* measure is a measure of purity and represents the expected amount of information carried by a new instance. Entropy on the other hand is a measure of impurity (the opposite). However, in our context, we use entropy to (the quantity of uncertainty) to measure the purity of a node. If a node is pure, entropy is close to 0: there is no uncertainty about making decisions. If a node is impure, entropy will be higher because making decisions becomes expensive.
19. So we compute the information values of the individual branches of the outlook attribute, and we sum them up. The overall information value of the attribute outlook is 0.693. Try to compute the information values for the other attributes yourselves.
20. Now in order to compute the information gain of an attribute with respect to the other attributes, we have to compute the entropy of the training examples at the root before any tree was created. At the beginning, before any split, we have 9 yes and 5 no, corresponding to an information value of 0.940. From 0.940, we subtract the information value of the outlook branch and we get a value of 0.247. This value is the gain, the information gain that we get if we create a branch on the outlook attribute. We repeat the same procedure for all the attributes and we will find out that the outlook attribute has the highest information gain. So it is the best root node for this dataset. We continue the same process with all the other nodes until the data cannot be split any further.
21. This is the end of the first video clip.

Part 2b

1. Welcome back to the second part of Lecture 4. In the previous video clip we described how to select a node using information gain. In simple words, we could say that information gain measures the amount of information load carried by an attribute. We analysed how to compute information gain values, and calculations are not too complicated if we understand the notions of surprisal and entropy. So far so good. However, information gain has a practical drawback. More specifically, when some attributes have a large number of possible values, a problem arises with the information gain calculation. Let's see what happens if we add an extra attribute, the ID code, to our weather dataset.
2. Since the ID code identifies the instance, it determines the class without ambiguity.
3. Therefore the information gain of the ID code attribute is just the information at the root, that is 0.940 bits. This value is greater than any other attribute, so ID code will be chosen as the splitting attribute. But if we branch on the ID code, we cannot predict the class of unknown instances. This attribute has no generalization power. Basically information gain prefers attributes with a large number of possible values, which is not always good.
4. Information gain is biased towards choosing attributes with a large number of values. This may result in *overfitting* (the model learns too many specific elements and is unable to generalize when predicting a new instance).
5. To compensate for this, a modification of the measure is usually adopted. This variant is called Gain Ratio. GR takes into account the number and the sizes of nodes into which an attribute splits the dataset, disregarding the information about the class. *If we neglect the class, we get the intrinsic information of a node*. We divide the information gain by the intrinsic information and we get the GR.
6. These are the GR values for the weather dataset without ID code. And you can see that values can differ from the values computed using IG.
7. If we would add the ID code attribute, the GR for the outlook attribute would be 0.247 which would be still the highest, but much lower than the information gain value of 0.940 bits. Problem with gain ratio: it may overcompensate May choose an attribute just because its intrinsic information is very low Standard fix: only consider attributes with greater than average information gain
8. The divide-and-conquer approach to decision tree induction is also called top-down induction of decision trees. The scheme that has been described using the information gain criterion is the one known as ID3; the use of GR was one of the improvements made over the years. The creator of ID3 is Ross Quinlan. Additional improvements were made to deal with missing values, noisy data, etc. and this implementation is called C4.5. In weka we use J48 which is an open source Java implementation of the C4.5 algorithm. There are other variants that have been implemented, like CART, Cart means Classification And Regression Tree analysis and is an umbrella term that covers both classification trees and regression trees. (Do you remember the difference between classification and regression? We will see this at quiz time) Basically decision trees tend to return similar results.
9. Let' introduce the concept of pruning now. Fully expanded decision trees often contain unnecessary structures and it is generally advisable to simplify them. Now it is time to learn how to prune decision trees. There are two strategies: postpruning and prepruning. By building the complete tree and pruning it afterward we are adopting a strategy of postpruning (sometimes called backward pruning). Prepruning would involve trying to decide during the tree-building process when to

stop developing subtrees—quite an attractive prospect because that would avoid all the work of developing subtrees only to throw them away afterward. However, postpruning does seem to offer some advantages.

10. Most decision tree builders postprune. First, build full tree then, prune it. Fully-grown tree shows all attribute interactions. Problem: some subtrees might be due to chance effects. Two rather different operations have been considered for postpruning: subtree replacement and subtree raising. At each node, a learning scheme might decide whether it should perform subtree replacement, subtree raising, or leave the subtree as it is, unpruned.
11. Subtree replacement is the primary pruning operation. The idea is to select some subtrees and replace them with single leaves. For example, the whole subtree on the LHS, involving two internal nodes and four leaf nodes, has been replaced by the single leaf "bad" (RHS). When subtree replacement is implemented, it proceeds from the leaves and works back up toward the root.
12. The other postpruning strategy, subtree raising, is more complex and computationally expensive. Subtree raising is a potentially time-consuming operation. In actual implementations it is generally restricted to raising the subtree of the most popular branch. It is not always worth implementing it, but it is used in a very influential building system tree like C4.5. It simple words, it proceeds by deleting one node and redistributing the instances. If you look at the slide, you can see that the node "C" has been "raised" to subsume node "B". The daughters of the raised node are marked with primes (the single quote next to the numbers) to indicate that they are not the same as the original daughters 1, 2 and 3 but they also include examples originally covered by node B.
13. Prepruning is based on statistical significance test (like chi-square). With prepruning, the tree stops growing when there is no statistically significant association between any attribute and the class at a particular node. We will explain statistical significance in the next lectures.
14. It is possible to read a set of rules directly off a decision tree, by generating a rule for each leaf and making a conjunction of all the tests encountered on the path from the root to that leaf. This produces rules that are unambiguous and it does not matter in which order they are executed. However, these rules are more complex than necessary, and rules derived from trees are usually pruned to remove redundant tests.
15. It is more complicated to transform a set of rules into a tree because trees cannot easily express the disjunction implied among the different rules in a set.
16. A good illustration of this problem occurs when the rules have the same structure but different attributes like: if a and b then x and if c and d then x. In this case, it is necessary to break the symmetry and choose a single test for the root node. If for ex a is chosen, the second rule must be repeated twice in the tree, as shown in the tree on the right hand side. And this is known as the replicated subtree problem.
17. We end the description of decision trees here. In this lecture we continued the description of decision trees and talked about the following topics: attribute selection, entropy, surprisal, information gain, gain ratio, pruning and rules. Now a few simple quizzes. I tried to make these quizzes naively tricky, just to double check that your attention is still with me ☺

---the end -----