Decision Trees

Florian Hartmann, Yu He

July 17, 2017

Abstract

TODO: Abstract

1 Introduction

(Give an overview of the report)

General

- Repeatedly split the space (divide and conquer)
- Make a decision by following a path down to a leaf
- Nonlinear is possible
- Nonparametric

Two types of nodes:

1. Leafs
2. Internal nodes

Advantages

- Interpretability: In contrast to neural networks, it's very clear why a decision was made. This is mostly true for individual predictions, interpreting the entire decision tree is harder
- Little preprocessing needed: Depending on the algorithm, missing values don't need to be handled by the user of the algorithm. Features don't need to be normalized (0-centered and 1-variance)
- Can deal with nonlinearities without kernel tricks
- Predictions are very quick: Just follow a path to a leaf
- Describes given data very well
- Does feature selection for you
- Easy to parallelize

Disadvantages

- Very easy to overfit
- Many linear decisions (each node) can't necessarily approximate curves well
- Still have to do feature engineering, e.g. for computer visison
- Too many dimensions are bad (Optimal: Low dimension, many training examples)

Regression
(Just shortly explain how this could work but mention that we focus on classification here)

2 Constructing a decision tree

2.1 ID3

ID3 (Iterative Dichotomiser 3) is a straight-forward approach for building decision trees. We decided to start with this algorithm because it allows for a gentle introduction to the fundamental ideas behind building decision trees. The fact that it's quite simple also means that it has some serious drawbacks, and generally shouldn't be used in production. We will explain these shortcomings and address the most important ones in later sections of the report.

Generally, ID3 implements the ideas introduced in the previous section. There are two types of nodes: Internal nodes that make the decision which node to visit next, and leafs which make a constant prediction. To build a decision tree using ID3, a recursive strategy is followed: The training data is repeatedly split until a new node gets training data that is suitable for making a constant prediction.

In the following, we will look at both of these type of nodes, starting with internal nodes.

2.1.1 Internal nodes: Making decisions

In ID3, the input data is always split based on a single feature. For each possible value of that feature a new child node is created. In other words, if a split is made on a feature that allows k different values, then the node will have exactly k child nodes. The child node then follows the recursive strategy based on the subset of the training data used in the parent node that has the respective value in the selected feature.

Here the first serious drawback of ID3 becomes apparent: Splitting on all possible values only works well for categorical features. For a continuous variable like *age* this approach would work terribly. This is also the major shortcoming of ID3. In Section 3, we will see an algorithm that can deal a lot better with continuous features.

2.1.2 Leafs: Stopping the recursion

At some point, the recursion has to end, which means a node is turned into a leaf. The leaf then constantly predicts the same answer. Which answer to predict is based on the training data it received.

There are various reasons why a node might be turned into a leaf. The simplest one is that all data points it received have the same label. In this case, there is no reason to split the data further and the node can be turned into a leaf which predicts this label.

Another case is that there is just no training data available. We always create a new node for each possible value of a feature. If we're deep in the tree and split on a feature that can have many different values, there might not exist training data for each case. If this happens, the most common label of the parent node is predicted.

There are other cases where we might want to stop the recursion. We will come back to these in a later section.

2.1.3 Selecting the feature to split on

Of course, now the major remaining question is how find the best feature to split on. ID3 follows a greedy strategy for this. For each feature, it evaluates how well the classes would be divided in the next step if a split based on this feature would be performed. Then, the feature where this worked best is chosen. Of course, this greedy strategy could mean that a very good combination of several subsequent splits is missed, because we were too greedy in the beginning.

To find the feature which produces the best division in the next step, different heuristics can be used. The easiest one is misclassification: We just assume that the next node will predict the most common label and calculate how accurate it would be by doing this. Then, the feature with the lowest misclassification rate is chosen.

A more popular heuristic that's commonly used in ID3 is entropy and information gain. In a nutshell, entropy measures the uncertainty, i.e. if a node receives data points that are mostly in the same class, uncertainty of what to predict is very low. If it receives data points in equal number from two classes, then the uncerainty is very high. Information gain aggregates the entropy over the different values that a feature can have. In the end, the feature which yields the largest information gain is chosen. A more thorough description of the heuristics to use for splitting is given in section 4.

Another important idea to mention is that in ID3 each feature should only be used once for splitting. Because we split based on all possible values, the data of each child only has one kind of value for each feature that was already split on. Hence, it doesn't make sense to split on these features again.

2.1.4 Limiting the tree depth

Overfitting is a huge problem with decision trees. It can happen very easily that the training data is perfectly described. However, this description might contain so much noise that the predictions don't generalize well for previously unseen data points.

In most Machine Learning algorithms a very common way to combat overfitting is regularization: By adding additional constraints, the model is forced to be simpler. A simpler model means there is less capacity for remembering information from the training data, and hopefully the more important information is remembered.

For decision trees, the primary way to add regularization is to limit the tree depth. Because we split on each feature exactly once, limiting the tree depth in ID3 is equivalent to limiting the maximum number of features that should be used for one prediction. If only a low tree depth is allowed, it means that only the most important features are considered. Instead of splitting, a leaf is then created that predicts the most common label.

Of course, the optimal tree depth highly depends on the dataset being used. Because of this, tree depth is a hyperparameter that needs be optimized by a method like cross validation.

Another way to limit the tree depth is to only perform a split if the information gain is high enough. The intuition behind this is that splits with low information gain tend to introduce more model complexity without directly improving the predictive power of the model.

However, sometimes a split with low information gain is necessary to get a good split afterwards. One extreme example for this is the xor dataset. In the first layer, all possible ID3 splits yield an information gain of 0. If we stop here, we could only get an accuracy of 50%, which is equivalent to randomly guessing. However, if we perform the split without a direct information gain, we can perform a perfect split afterwards, which yields an information gain of 1 and improves accuracy to 100%.

(image of decision tree and xor table here)

2.2 C4.5

- From the same creator as ID3 and was designed to fix ID3 shortcomings
- We focus on the three features we consider to be the most important / interesting ones
- There's an entire book on C4.5, so of course there are many more ideas to try but that's beyond the scope of this project
- (Maybe: Mention weighting features is also interesting)

2.2.1 Continuous features

- Probably the biggest improvement
- Simple bruteforce solution: Consider each binary split ($\leq h, > h$) and calculate the information gain. The maximum information gain is then compared to the ones of the other features. We now need to save one more information about the split (h)
- Actually, not all h that occur in the training data need to be considered, many lead to the same information gain
- Idea: Add a function that finds all splits that should be considered
- TODO: Show a table with example values for this

2.2.2 Missing values

- Most algorithms don't know how to deal with missing features, so one usually imputes missing values, e.g. with the median or mean, and then uses normal algorithms
- C4.5 has a built-in way to deal with this problem, so no imputing needs to be done by the user
- Missing values are ignored when computing entropy or information gain
- We remember how common the individual values of a split are
- When predicting and a feature is missing, we randomly choose a child to visit next, using the probability distribution we get by considering how common the individual values were
- When training, the data points with the missing value can be divided the same random way. Alternatives are giving them to certain children that don't get a lot of data, so that there's enough data for all children. Of course this can also introduce wrong biases, so it's a matter of trying and seeing what works best
- Sometimes it's important to always predict the same answer. So the deterministic variant is to always choose the most common option

2.2.3 Pruning

- As explained earlier, we don't want super deep trees that overfit
- Stopping the splitting when the information gain is too low is a bad idea, because there might be a great split coming afterwards (see xor)
- Because of this, we first want to grow the tree very far. Afterwards we can go through the tree and prune all nodes that didn't lead to a good information gain
- (Based on confidence intervals and binomial probability distributions with a user specified threshold)

3 Metrics

(Show plots for the 2d case)

(Measures of impurity)

(For pure data: 0, where everything is equally likely: 1)

3.1 Misclassification rate

3.2 Entropy and information gain

3.3 Gini

4 Random Forest

- Bagging to convert many weak learners into a strong learner
- Easy to parallelize

5 Evaluating datasets

5.1 Titanic

5.2 Income census

(Maybe check with the teacher if we can really use this)