MATERIAL FOR DECISION TREES AND BOOSTING
MACHINE LEARNING E20

Mathias Ravn Tversted
Department of Computer Science
Aarhus University
Åbogade 34, 8200 Aarhus N
`Tversted@post.au.dk`

December 27, 2020

Contents

1 Decision Trees

It is a way of approximating discrete-valued target functions by using tree structures. This is basically spicy nested if-statements.

A tree represents a disjunction of conjunctions, where each path from the root to a leaf represents a conjunction of attributes.

They work well if the instances of the data are represented with attribute-value pairs, as these correspond nicely to the tests at each layer of the tree.

They are also pretty good at *classification*.

INDSÆT NEMT EKSEMPEL HER PÅ ET TRÆ

Finding the smallest decision tree that perfectly classifies data is *NP-Hard*. Therefore, we have to rely on heuristics in order to create them, a popular one of these is by using *entropy*.

2 Entropy and loss functions

INDSÆT NOGLE AF KASPERS NICE BILLEDER

For a c-wise classification of points, we can measure the *entropy* of such a distribution. The entropy measures *uncertainty* in a distribution/the expected number of bits needed to represent this information.

The equation for Entropy is as follows.

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

Where p_i is the proportion of points with that label.

If we want to, we can measure the entropy of an entire decision tree by measuring summing over the entropy of each leaf in the tree. If S_v is the subset of n data points ending in that leaf, then the entropy of the entire decision tree is

$$\sum_{v} \frac{|S_v|}{n} Entropy(S_v) \tag{2}$$

Where $Entropy(S_v)$ only considers the points that end in that leaf node.

This also allows us to measure the *information gain* from rearranging our tree, to make decisions on how we should build it.

Entropy weights clean cuts very high

3 Algorithma

MÅSKE SKAL DER OGSÅ VÆRE DE DER BRUTE FORCE NOGLE FRA

3.1 Greedy algorithm for small decision trees (KALD NOGET ANDET?)

ID3? Fra DT noten:

> Summary of the ID3 algorithm specialized to learning boolean-valued functions. ID3 is a greedy algorithm that grows the tree top-down, at each node selecting the attribute that best classifies the local training examples. This process continues until the tree perfectly classifies the training examples, or until all attributes have been used

Let k be a budget of internal nodes in the tree. Start with the root as a leaf. Repeat k times:

1. Try all leaves v, all features f and all split values x
2. For each choice, compute reduction in tree entropy
3. Replace with the stump that decides entropy the most

4 regression trees

DER SKAL LIGE SKRIVES NOGET KORT OM HVAD DET ER!

Her skal der stå ting om BAGGING, RANDOM FOREST & ADABOOST MÅDDERFÅKKER

5 Boosting

Boosting are meta-algorithms that take weak learners, defined as models that are better than random guessing, and combining them somehow in order to produce good learners. This can be done with things such as voting or weighing.

If we run algorithm A T times (not on same dataset) in order to produce hypotheses $h_1, \cdots, h_T \in H$, we can with weights $\alpha_1, \cdots, \alpha_T$ produce a better model

$$f(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x)) \tag{3}$$

Here the different weights and datasets enable us to tweak things in order to produce a good learner. It is also possible to have weights on training data. For example, some points could be outliers or something similar which could mess up classifiers.

5.1 AdaBoosting

AdaBoost runs in T rounds where it creates a new weak hypotesis h_t.
It then calculates the error (The sum of scenarios where $h_t(x_i) \neq y_i$).
The weight a_t for that hypothesis is larger the less errors there are: $\frac{1}{2} ln(\frac{1-er_t}{er_t})$.
Then using the previous weights for the dataset (D_t), the new hypothesis (h_t), and its weight a_t a new set of weights for the dataset is created. $D_{t+1}(i) = \frac{D_t(i) exp(-a_t y_i h_t(x_i))}{Z_t}$
The final function after the T rounds is $f(x) = sign(\sum_{t=1}^{T} a_t h_t(x))$

Problems with AdaBoost

- Minimises exp loss, not your favourite davourite loss function
- Binary classification problem, but we may want to do regression or sigmoidybois

6 Gradient boosting

6.1 Distance to target

Algorithm:
Make a regressions tree h_t in each round t that minimizes least aquares loss where the y it trains against is $y - h_{t-1} - \cdots - h_1$.
To prevent overstepping use a learning rate η.

6.2 Gradient Descent

- General template for boosting with any pointwise loss function $L(y', y)$
- Only need to be able to compute derivative
- If we can compute derivative of $L(c, y_i)$ w.r.t c, then we can minimise gradient descent in one variable
- Can focus on effective MSE regression tree algorithms
- To optimise for concrete loss function L, only need to be able to opmise for constant in a leaf

Allows for arbitrary loss functions. Uses regression trees as base learner. We will train many regression trees and take their weighted sum as the output.

Let $h_1 = c$ where c minimises $E_{in} = \frac{1}{n}\sum_{i=1}^{n} L(c, y_i)$ be the first regression tree. Here $\alpha_1 = 1$.

For $t = 2, \cdots, T$:

1. Train regression tree h_t to minimise least squares loss
2. With data $X, \hat{y} = -\begin{pmatrix} \frac{\partial L(F_{t-1}(x_1), y_1)}{\partial F_{t-1}(x_1)} \\ \vdots \\ \frac{\partial L(F_{t-1}(x_n), y_n)}{\partial F_{t-1}(x_n)} \end{pmatrix}$
3. For each leaf l in h_t
4. (a) Let l predict value c_l minimajsing
   (b) $\sum_{i \in l} L(F_{t-1}(x_i) + c_l, y_i)$
5. Let $\alpha_t = \eta$
6. Output $f(x) = \sum_{t=1}^{T} \alpha_t h_t(x)$