Machine Learning

Lecture 10

Ensemble Learning and Other Topics

Justin Pearson [1]

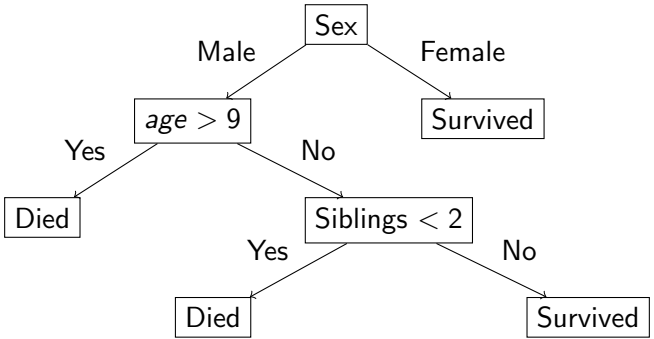
[1] https://justinkennethpearson.github.io/teaching/courses/intro_ml/

Today's plan

- Decision trees for regression with a simple greedy learning algorithm.
- The idea of weak learners with some intuition from probability.
- Random Forests , combining shallow decision trees.
- Very broad introduction (without too much detail, because it requires too much mathematics) to:
  - Boosting
  - Bagging
  - AdaBoost
  - Gradient Boosting
  - XGBoost (eXtreme Gradient Boosting)

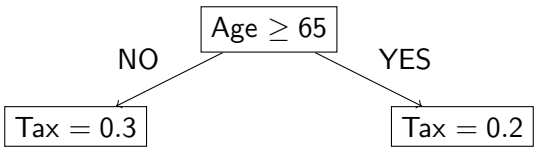
Decision trees for Regression

In lecture 8 we talked about decision trees, and came up with a learning algorithm for classification of categorical data. We hinted that it is possible to include non-categorical data:



Decision trees for Regression

It is also possible to use decision trees to produce a value:



The question is how do we learn the decision boundaries, such as 65 in this example.

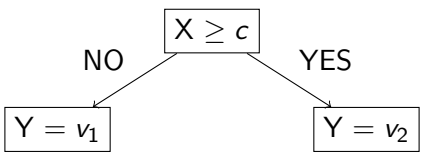
Decision trees for Regression

- The problem is as least as complex for learning decision trees for classification using categorical data.
- Any algorithm that we use is going to be a heuristic.
- As with ID3 the algorithm is recursive. So if we know how to split one node, then we just keep going recursively.

Measuring Error

Since we are trying to do regression then we should use mean squared error.

Suppose that our tree is trying to predict the value Y from the variable X.



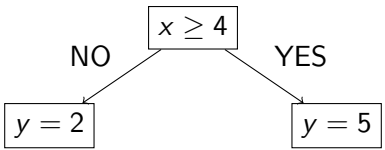
What are the correct values of v_1,v_2 and c?

Simple Example

Suppose that we have the following data set:

x	y
1	2
2	5
3	4
7	3
8	4

If we build the tree:



Simple Example

Running through our example we need to separate our data into two sets $x < 4$ and $x \geq 4$.

- $x < 4$ is the set $\{(1, 2), (2, 5), (3, 4)\}$ elements of the set are (x, y) pairs.
- $x \geq 4$ is the set $\{(7, 3), (8, 4)\}$.

Simple Example

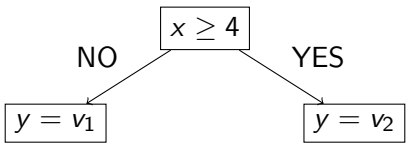
Running through our example we need to separate our data into two sets $x < 4$ with predicted value $y = 2$ and $x \geq 4$ with predicted value $y = 5$. So we can calculate the squared error.

- $x < 4$ is the set $\{(1, 2), (2, 5), (3, 4)\}$ gives the error $(2-2)^2 + (5-2)^2 + (4-2)^2$ equals $0 + 9 + 4 = 13$.
- $x \geq 4$ is the set $\{(7, 3), (8, 4)\}$ gives the error $(3-5)^2 + (4-5)^2$ equals $4 + 1 = 5$

So the total mean squared error is : $1/5 * (13 + 5) = 3.6$.

Splitting on c

Suppose we are given c then what are the sensible choices for v_1 and v_2?

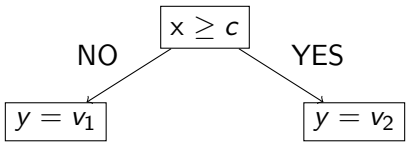


We have the two sets: $\{(1,2),(2,5),(3,4)\}$ and $\{(7,3),(8,4)\}$. We want to minimise the resulting mean squared error. Picking the average value in each set minimises the mean squared error.

- $v_1 = \frac{2+5+4}{3} = 3.67$
- $v_2 = \frac{3+4}{2} = 3.5$

If you run through the calculations you'll get an error of about 1.033

Finding c

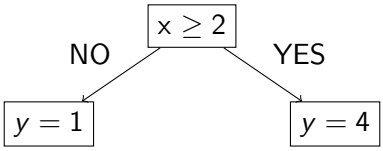


Given c we now know how to calculate the best values of y_1 and y_2.

To find the best value of c we simply sort our data set by y, go through all the possible values of c for our data set and pick the one that gives the minimum error.

Simple Example Continued

If you run through the calculations you'll get this tree:



Multiple inputs

The obvious recursive greedy algorithm is to go through all variables, look at all the splitting points and pick the one that gives the minimum error.

Rinse and repeat recursively.

If you are trying to predict something from multiple input values say $x_1, x_2, \ldots, x_n$. Then for each variable x_i you are ignoring the other variables when you do the sorting.

Ensemble Learning

Decision trees are great:

- They can learn anything.
- You can understand what they have learnt.

But

- They are prone to overfitting
- Small changes in the data set can give you different trees.

Because they can learn anything, they tend to overfit your data set.

Ensemble Learning

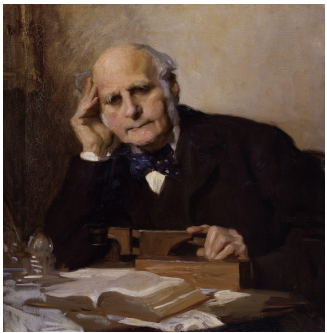
Basic idea.

Combine multiple weak learners (see later) and take majority votes.

Various techniques such as boosting and bagging to get more statistical mileage out of your data.

Wisdom of the crowds

At a 1906 country fair in Plymouth (UK), 800 people participated in a contest to estimate the weight of a slaughtered ox. Francis Galton observed that the median guess, 1207 pounds, was accurate within 1% of the true weight of 1198 pounds.



Sir Francis Galton (1822 - 1911). Again a founder of modern statistics, but again had ideas that we find abhorrent. He was a founder of eugenics.

Wisdom of the Crowds

General idea. Take lots of incorrect observation. In the average the errors cancel out.

Probabilistic Warm Up — Triple Modular Redundancy

This is an old idea in hardware design.

- Assume that hardware failures are independent.
- Run three devices in parallel and take a majority vote on a decision or an action.

Probabilistic Warm Up — Triple Modular Redundancy

If a single device has probability p of failing, then the whole system fails if 2 or 3 out of the 3 devices fail. So the following failure scenarios (strike through represents failure):

- ~~123~~ $= p^3$
- ~~123~~ $= p^2(1-p)$
- ~~123~~ $= p(1-p)p$
- ~~123~~ $= (1-p)p^2$

So the total probability of failure is $p^3 + 3p^2(1-p)$. We are doing this by hand, but you can calculate this directly using some elementary combinatorics.

So even if you have a system with a probability of failure of say 0.2 you could reduce this to about 0.1.

Weak Learners

You can turn the idea on its head.

Suppose you have independent classifiers which are slightly better than random guessing.

If you have enough of them, then the probability of making a wrong decision is when a majority of them get the wrong answer.

Weak Learners

Suppose there are 25 **independent** classifiers:

- Each with an error rate $\epsilon = 0.35$.

Then the probability that the whole ensemble makes a wrong prediction is:

$$P(X \geq 13) = \sum_{i=13}^{25} \binom{25}{i} \epsilon^i (1-\epsilon)^{25-i} \approx 0.06$$

Weak Learners

$$P(X \geq 13) = \sum_{i=13}^{25} \binom{25}{i} \epsilon^i (1-\epsilon)^{25-i} \approx 0.06$$

- The ϵ^i term is probability of i incorrect predictions.
- The $(1-\epsilon)^{25-i}$ is the probability $25-i$ correct predictions.
- The $\binom{25}{i}$ term gives the number of ways of choosing a subset of size i from 25 items.
- You need at least 13 of the classifiers to be wrong.
- You sum of all the terms.

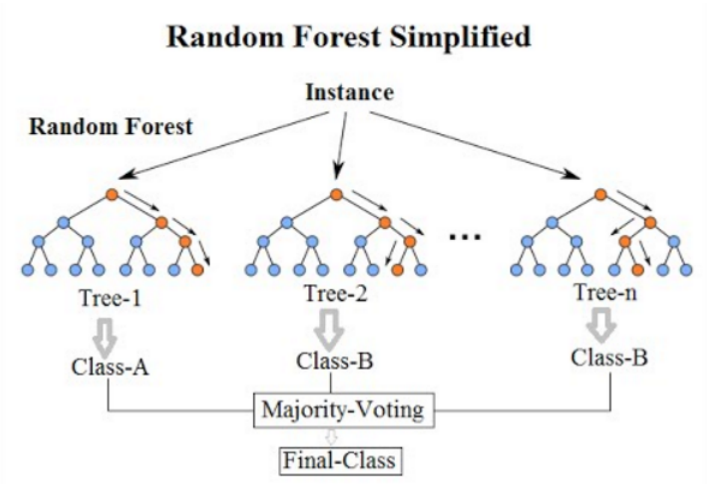
So we have transformed a probability of $1 - 0.35 = 0.65$ of being correct into a probability of $1 - 0.06 = 0.94$ of being correct.

Important: The analysis only works if the weak learners are independent.

They can make mistakes, but the mistakes that they make have to be independently correlated.

All of the rest of the techniques in this lecture are ways of training **independent weak learners.**

Random Forests[2]



[2]Picture taken from Wikipedia

Random Forests

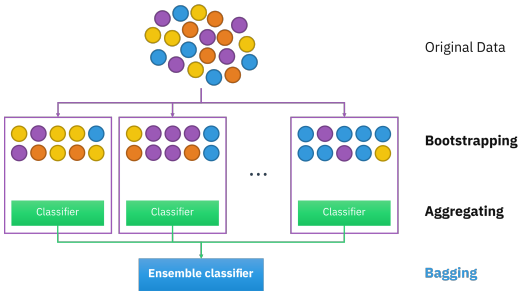
Majority voting with decision trees:

- Build weak learners by picking random subsets of the features.
- The learners will be weak, because you won't use all the features.
- The learners will be independent because they work on different features.

Often works very well in practice.
Lots of hyper-parameters to tune, the number of trees, the number of features in each tree.

Bootstrap aggregating — Bagging[3]

A way of sampling the original data set to make a number of statistically independent data sets.



We will not worry too much about the details. You sample with replacement, you can use the same item twice in each subset. This gives statistically similar but independent samples of the same size, that are manufactured from your original data set.

[3] Picture taken from Wikipedia

Boosting

Bagging you train everything in parallel. Boosting you train a sequence of weak learners. You can learn from the mistakes that previous weak learners make, or you can work out some sort of weighting scheme.

Boosting — or Adaboost

The details are complicated and require a lot of mathematics to understand what is going on, but the idea is something like:
Initially you give each item in your training set an equal weight. Then repeat the following steps:

- Train a new weak learner on your weighted set. Note that weight tell you how important the item is, so you have to modify your learning algorithm a bit.
- Work out where the weak learner miss classified and update the weights of those training samples.

Combine all the weak learners that you trained.

Gradient Boosting

Again train things sequentially, but

- Try to pick the next weak classifier that improves the overall training error.

Extreme gradient boosting or XGBoost is an efficient parallel version of gradient boosting that works well on large data sets.

Boosting

- Most of these algorithms originated in decision trees, but the techniques can be used with other learning algorithms.
- It is all about reducing variance. Weak learners cannot be too biased, and the various ways of combining them has good probabilistic reasons for working well.

Take away from this lecture

- Training decision trees for regression.
- Random Forests.
- What are the ideas behind boosting and bagging.

If you want to win a Kaggle competition without using deep learning, then try XGBoost[4].

[4] https://xgboost.readthedocs.io/en/stable/