Decision Trees

EE16ML

Basic terminology

- **Data point**: a sample from our dataset, e.g. a participant in the Covid-19 survey
- **Features**: quantifiable/qualifiable attributes of a data point, e.g. participant’s age
- **Class/label**: the “category” assigned to a data point, e.g. participant classified as “at-risk”
- **Classification (method)**: a type of model whose purpose is to classify data points into a class based on some explicitly (or implicitly) chosen features, e.g. our decision tree model
- **Predict(ion)**: the class our model thinks a data point belongs to, based on its features

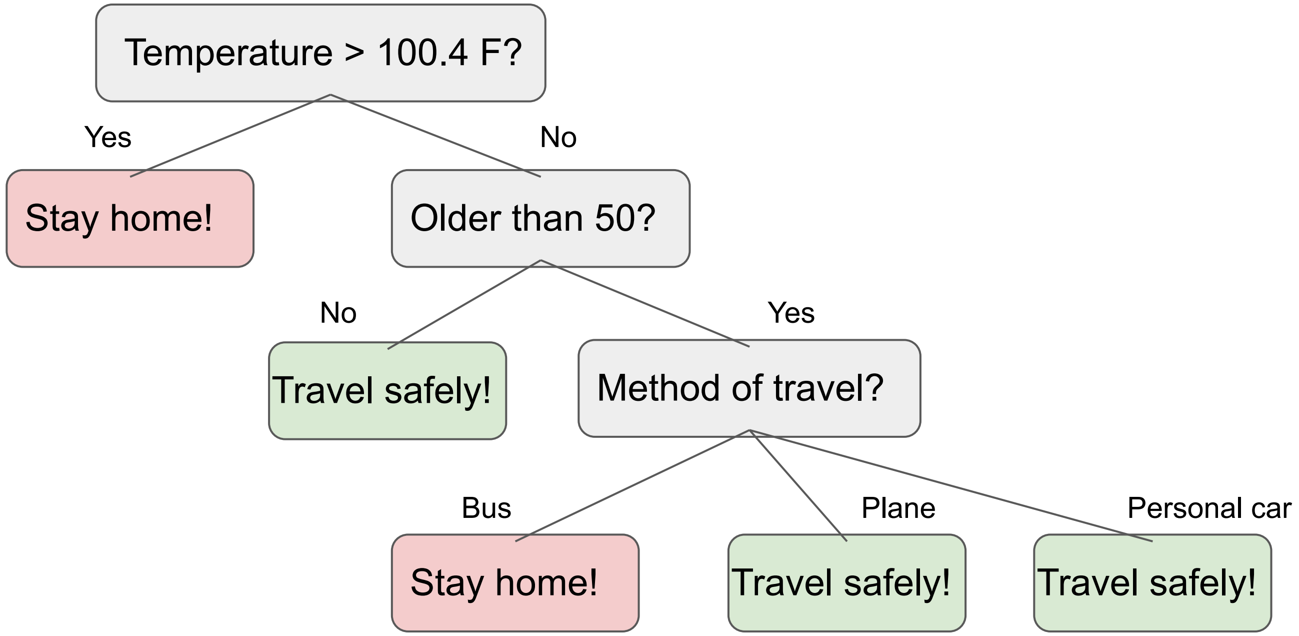
Motivation (1)

Suppose you want to visit family over winter break but are worried about your risk of contracting Covid-19. You try to assess your risk based on various features:

- Method of travel
- Travel time
- Travel distance
- Age
- Symptoms

Motivation (2)

Your process of evaluating whether or not you should go might look like this:



Motivation (3)

Why decision trees?

- We can ask questions about both categorical and numerical features!
  - Numerical features: travel time, travel distance, age
  - Categorical features: method of travel, symptoms
- Decision trees are visually easy for humans to interpret
- Recall binary trees: binary tree search is similar to DT classification, but we don't have to ask only "binary" questions

Definition

A decision tree (DT) is a classification method that models functions of features.

Input: data point (specifically the features)

Output: prediction of data point’s true class label

Goal: get a “good” classifier--at least better than uniform distribution (why?)

- **Uniform distribution** is when all outcomes are equally likely, analogous to picking an outcome at random.

Structure of a DT

- **Internal nodes**: nodes where you ask a question (usually about some feature(s))
- **Terminal nodes**: node where you have determined which class a data point belongs to
- The branches that connect the nodes represent the answers to the questions (e.g. true/false values, categorical values, etc.)
- Recall example tree from before: internal nodes were grey, terminal nodes were green and red.

A “good” decision tree (1)

What makes a good DT?

- Limit space complexity
  - How much space to store DT?
- Limit time complexity
  - How long to classify a point? On average? Worst case?
  - Recall binary search is O(log N). How long will classification take?
- Question: how could we improve our example DT?

A “good” decision tree (2)

Overfitting: a phenomenon where your model follows training data too “closely”, so the model performs poorly on test data; your model has a “narrow”/“inflexible” view of the world. Overfitting is bad!

Larger trees have a **higher** chance of **overfitting** training data, while shorter trees have a **lower** chance of **overfitting** training data.

Occam’s razor: “the simplest explanation is usually the right one”

DT classification boundaries

Classification boundaries specify which regions belong to which class, according to the classifier.

- Points are represented by their true label (+ or -)
- Green region is where DT thinks + class are
- Red region is where DT thinks - class are
- Notice the regions defined by DT are **axis-aligned**
  - How will DT represent diagonal decision boundaries?

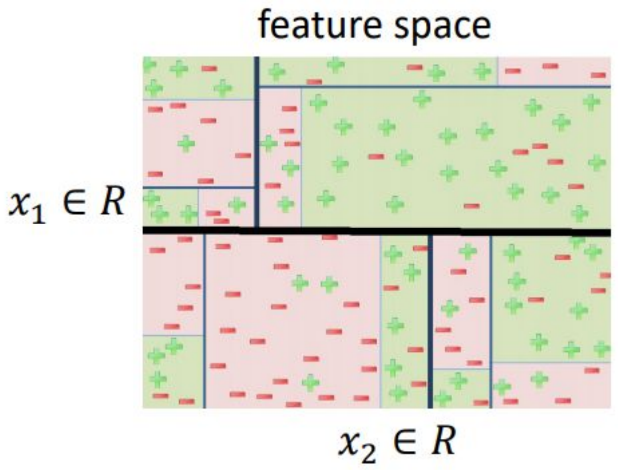
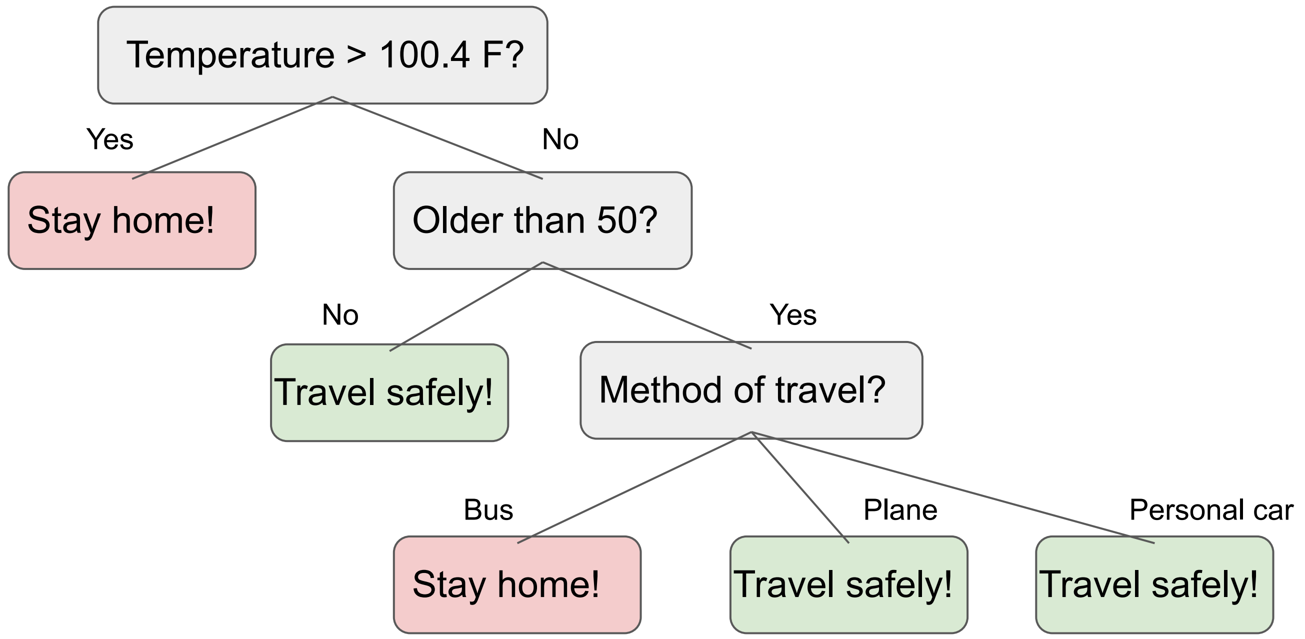


Image: Yisong Yue

Splitting



We can choose features to **split** on. Here, we split on temperature, age, and method of travel.

A “good” split

What makes a “good” split?

Think about the game 20 questions--the fewer questions you ask, the better you are at the game. Same idea here--less questions, smaller tree. But what question to ask?

Probability Basics

Random variable (RV): a variable that takes on certain values according to some probability distribution

Probability distribution: a function that tells you the likelihood of a RV taking on a value x, denoted P(x)

Entropy

A measure of **uncertainty/surprise** in a random variable, denoted by **H(X)** where X is a RV. Also known as amount of **information** in a RV:

$$H(X) = -\sum_{i=1}^{n} P(x_i) \, log \, P(x_i)$$

Aside: this quantity is derived from taking the **expectation** of -log P(x). See https://en.wikipedia.org/wiki/Expected_value for more details.

Conditional Entropy

The conditional entropy of a random variable Y given a random variable X is

$$H(Y|X) = \sum_{i=1}^{n} P(x_i)H(Y|x_i)$$

We can apply conditional entropy to tree splits where we split the data into subsets where $x_j < v$, $x_j \geq v$ repre x_j nts a data point and v is the value that we are splitting the data on:

$$H(Y|X_{j,v}) := P(X_{j,v} = 1)H(Y|X_{j,v} = 1) + P(X_{j,v} = 0)H(Y|X_{j,v} = 0)$$

Properties of Entropy

- H is non-negative
- $H(X,Y) = H(X|Y) + H(Y) = H(X) + H(Y|X)$
- If X is independent, then $H(Y|X) = H(Y)$
- $H(Y|Y) = 0$
- $H(Y|X) \leq H(Y)$
    - In other words, if we know X, we can only have less uncertainty about Y (in the worst case, X tells you nothing about Y)
- You don’t need to mathematically prove these, but think about why these properties “make sense”

Information Gain (aka Mutual Information)

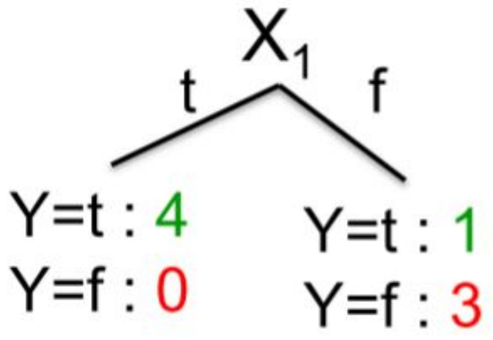
If you knew some information *a*, how much more will you know about RV X?

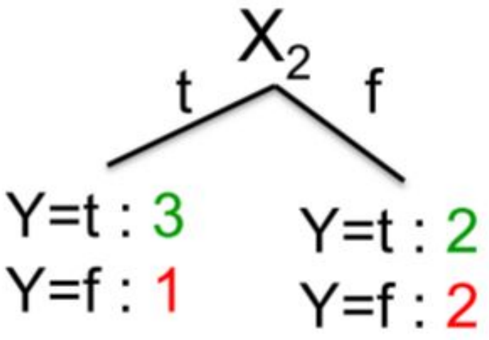
- ***IG(X,a) = H(X) - H(X|a)***
- **Nonnegative**: information gain is always greater than or equal to zero. Why? Since $H(X|a) \leq H(X)$, it follows that $H(X) - H(X|a) \geq 0$. This makes sense since we can never lose information about X if we know a, we can only gain information about X! In the worst case, we don't gain any additional information about X at all!

Purity (and impurity)

A **pure** split is a split that cleanly divides data points into their correct classes. So we want to find splits that maximize purity (i.e. minimize impurity).

Which tree would you prefer? Hint: notice purity of each split.





Example: J. Listgarten, Nov. 16, 2020 lecture

Gini impurity (aka Gini index)

- Gini impurity measures how often a data point would be incorrectly labeled if it were randomly labeled according to the distribution of the labels in the test set. The formula is as follows:

$$G(Y) = \sum_k P(Y=k) \sum_{j \neq k} P(Y=j) = \sum_k P(Y=k)(1 - P(Y=k)) = 1 - \sum_k P(Y=k)^2$$

- P(Y=k) represents probability that the data point's true label is k, $\sum_{j \neq k} P(Y=j)$ represents the chance that you mislabel the point (i.e. don't choose k), and finally sum over all possible choices of k to get G(Y)
- Properties:
  - max(G(Y)) = k / k-1
  - min(G(Y)) = 0 (**nonnegativity**)
  - G(X, Y) = G(Y, X)

Decision Tree Induction

- **Greedy algorithms** maximize some value at every iteration
  - Because of this, they tend to be **short-sighted** and are usually not optimal
- Greedy attempt #1: Split on maximum information gain.
  - Why? Can think of maximizing the amount of information you know with the fewest number of questions possible
- Greedy attempt #2: split on minimum Gini index.
  - Why? We split on purest splits first.
- When do we stop splitting?

Recursive Decision Tree Induction (Pseudocode)

```
BuildTree(DataSet,Output)
•  If all output values are the same in DataSet, return a leaf node
   that says "predict this unique output"
•  If all input values are the same, return a leaf node that says
   "predict the majority output"
•  Else find attribute $X$ with highest Info Gain
•  Suppose $X$ has $n_X$ distinct values (i.e. X has arity $n_X$).
     – Create a non-leaf node with $n_X$ children.
     – The $i$'th child should be built by calling
                 BuildTree($DS_i$,Output)
       Where $DS_i$ contains the records in DataSet where X = $i$th value of X.
```

Code: J. Listgarten, Nov. 16, 2020 lecture

Recursive Decision Tree Induction: Base Cases

See else statement: can we instead stop splitting when information gain is zero?

- Greedy is not always optimal--in this case, we should continue splitting even if information gain is zero

What else is wrong with our algorithm?

- It keeps splitting until the data points are identical or the labels are identical--perfect recipe for overfitting

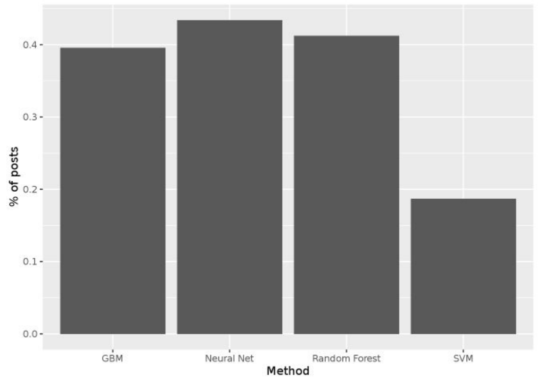
Gini index vs. entropy

Which value should we use?

- Although using the Gini index produces similar decision trees to entropy, it is faster because we don’t have to take logs to calculate it.
- Recall the **overfitting** problem--if we let our tree get too deep, we are more likely to overfit. So when to stop splitting?
  - If information gain below some threshold
  - If purity of remaining points above some threshold
  - If depth over some threshold
- Picking these thresholds: **validation**
  - Can train our DT with a separate set of data points called **validation set** to help us tune our threshold values (called **hyperparameters**)

Improvements

- Single decision tree can easily overfit!
  - High **variance**, i.e. don't expect DT to perform well consistently on test set
- Use multiple decision trees -- **ensemble methods**
  - **Bagging (bootstrap aggregating)**: train each tree using random sample of data points and average all results
  - **Boosting**: iteratively improve model by training trees that fix current model's mis-classifications
    - Ex: AdaBoost (adaptive boosting)



https://www.kaggle.com/msjgriffiths/r-what-algorithms-are-most-successful-on-kaggle/report