Introduction to Machine Learning (by Implementation)

Lecture 9: Training Decision Trees
The ID3 Algorithm

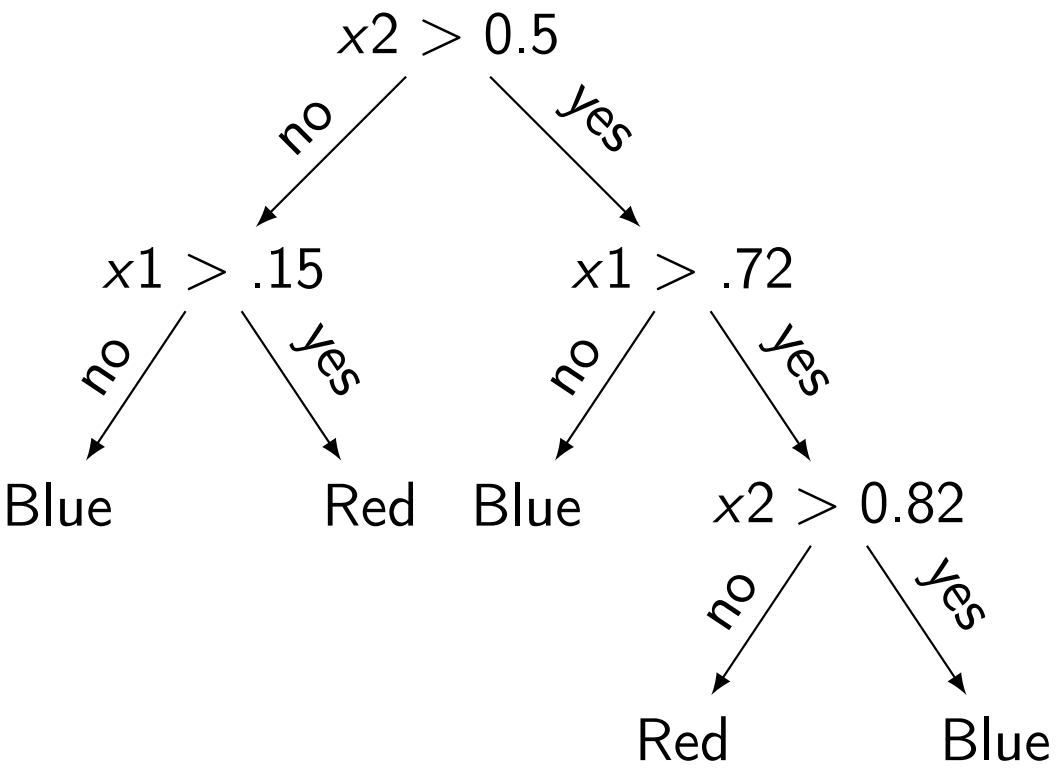
Ian J. Watson

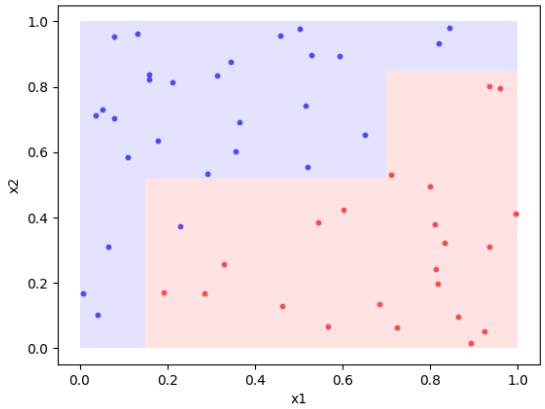
University of Seoul

University of Seoul Graduate Course 2019

KRF KOREA RESEARCH FELLOWSHIP
해외우수신진연구자 유치사업

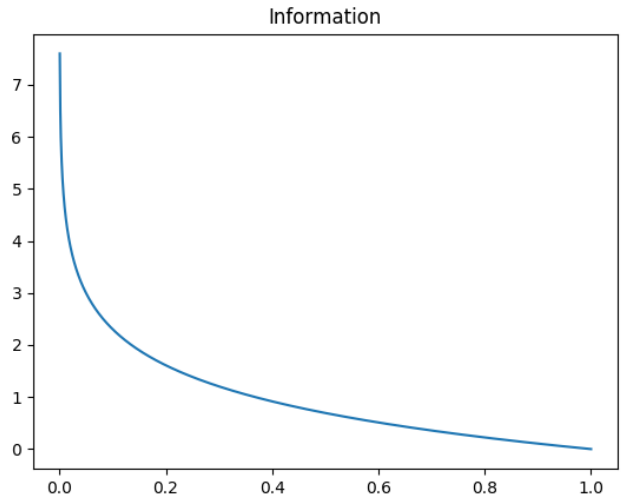
Introduction

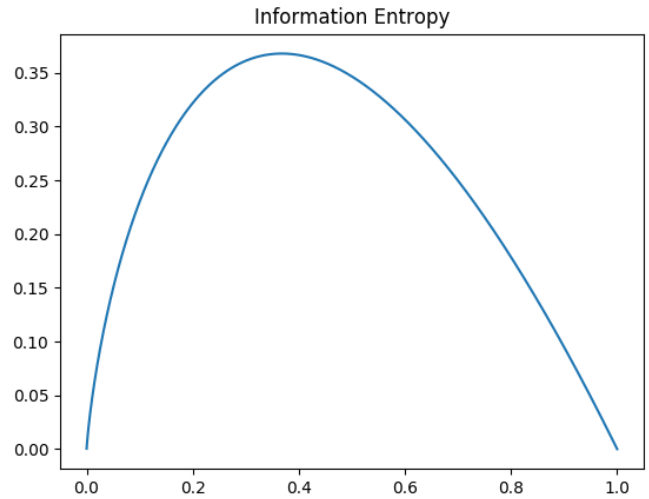




- Given a set of data we want to split into red and blue spaces
- The decision tree will partition the problem space into discrete regions
- We looked at writing code to take the tree and use it to classify last time, today we will look at training a tree

Shannon's Information Entropy





- Given a dataset with labels indexed by j, we define the information from observing label j as $I_j = -\log_2 p_j$
  - where p_j represents the probability of label j to be in the dataset (i.e. the fraction of data with label j)
    - If you put all the data in a hat and randomly picked one, what's the chance its in the j category
  - A low probability event "carries more information" than a high probability event (the theory was developed for communication)
- Then, we define entropy as $S = -\sum_j p_j \log_2 p_j$
  - The average information expected from sampling the data once
- As category prob. goes to 0 or 1, entropy goes to 0

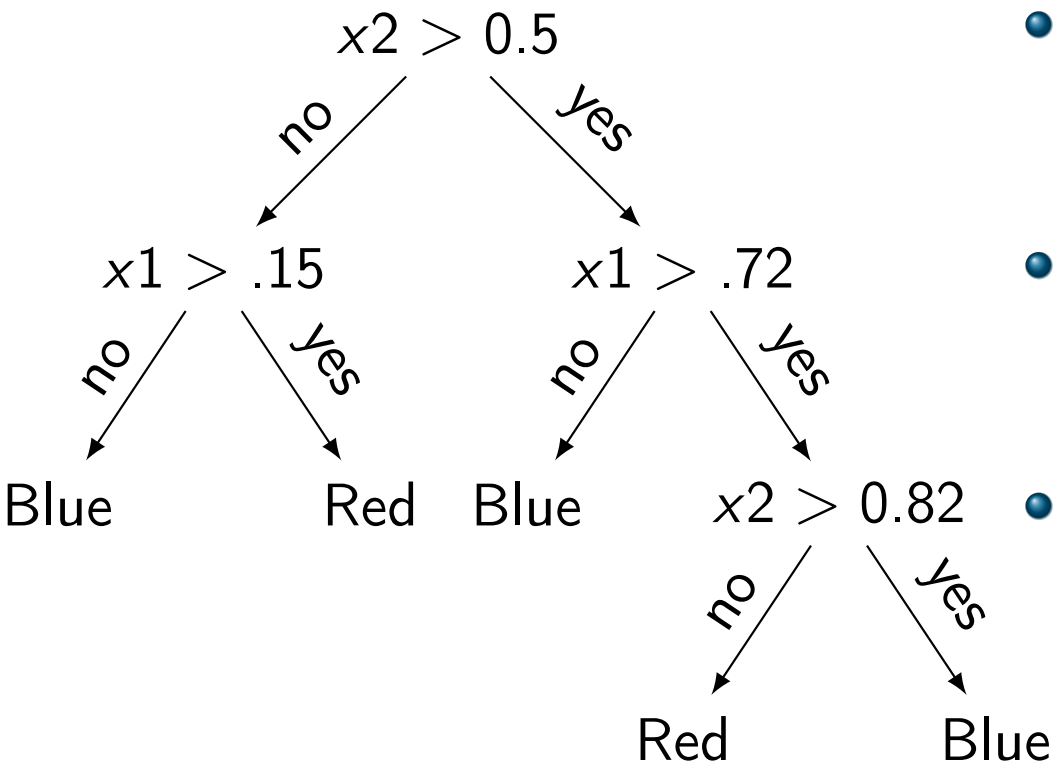
Partition Entropy

- What's this to do with decision trees?
- Well, (next week) we will start off with the full dataset, then begin partitioning the data via our cutoffs
  - That is introduce branches to separate the data
- We need a measure of how much better we separate the categories after some new branch, we want to go high entropy to low entropy
  - But taking the entropy of the whole dataset always results in the same entropy
- Instead, we will test this by checking the *partition entropy*
- After partitioning the dataset Ω into subsets $\Omega_1, \Omega_2, \ldots$ (think, the data at each of the leaves of the tree), containing $q_1, q_2, q_3, \ldots$ fraction of the data
  - $S = q_1 S(\Omega_1) + q_2 S(\Omega_2) + \ldots$ is the partition entropy
  - i.e. the weighted average entropy of the subsets

The ID3 Algorithm

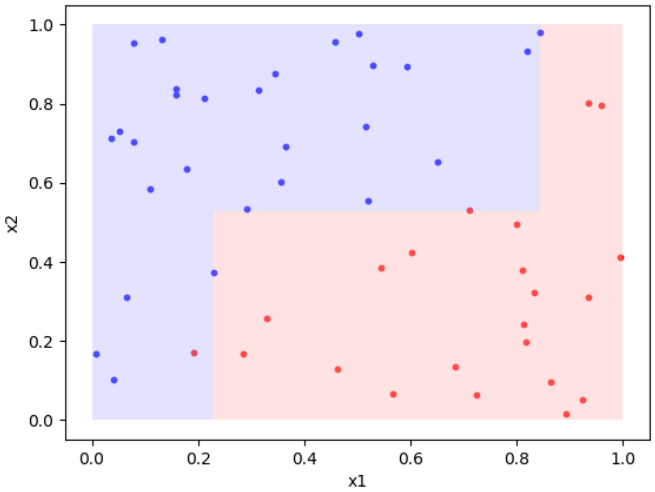
- Starting with the whole sample:
  - If all samples are in a single category, return a leaf node of that category
  - Go through each variable, test each possible branch point (based on the data), find the variable/cutoff pair with the highest information gain $I(\Omega, A) = S(\Omega) - S(\Omega|A)$
    - Where $S(\Omega|A)$ is our partition entropy based on some cutoff A
    - Equivalently, the partition with smallest partition entropy
  - Split into left/right, then, individually for left and right, go back to the start of the algorithm
- Possible branch points are infinite for real variables, so choose by taking each datapoint's x_j value when testing for best cutoff
- Can stop early if we reach a maximum *depth* (vertical size), in which case the node returned is a leaf node with the category with the most elements

Reminder: Representation of a cut-offs in Python



- Notice we can always write the cuts as $x_i > c$ for some $i \in \mathbb{Z}$ and $c \in \mathbb{R}$
- Our input will be a list of numbers, `x = [x0, x1, x2, x3, ...]`
- We will therefore only represent the i and c in our python code `(i,c)` will represent a node in the tree requiring $x_i > c$ at the node: `x[i] > c`

Example ID3 run



- Take the example dataset from last week
  - Let the blue outputs be represented by 1 and red by 0
- The tree the ID3 algorithm found is:

```
[(1, 0.529855996206679),
  [(0, 0.2298684687678438 4),
    [(0, 0.06450727283476698),
      1,
      [(0, 0.1907295070114163 3), 0, 1]],
    0],
  [(0, 0.8427165440886971), 1, 0]]
```

Exercises

- Starting with your functions written last week, write:
- `partition_by(inputs, attribute, threshold)`
  - Return a tuple of `(left, right)` partitions that would be created from the cutoff `x[atrribute] > thresold` (assume `inputs` is a list of (x, y) tuples)
- `partition_entropy_by(inputs, attributes, thresold)`
  - Returns `(left, right), partition_entropy` where partition_entropy is the `partition_entropy` from the split into `left, right`
- `best_partition_entropy(inputs)`
  - Find the cut with the best partition entropy
  - Try each attribute in turn, using all the data points for test thresholds
- `build_tree_id3(inputs, max_height, current_height=0)`
  - Implements the ID3 algorithm
  - If theres only 1 label type, or `current_height` = maxheight=, returns the label of inputs with the most samples
  - Otherwise, it finds the best partition (based on entropy) and creates a new split, recursing on the left and right returned by the split

Exercises

- The test will run the test sample from last week, and the Fisher data
- Check also that the accuracy is 100% when the max_height is high enough
- How does the accuracy evolve with height?

Final Project

- Extend one of the projects we've look at this semester in a new direction
- Similar to how we extended multinomial logistic regression with ridge regression, or add some new algorithm that can give more information
- Consider the ideas on the next page, or ones your interested in yourself, and let me know
- Next week, instead of a regular lecture, we can all meet, and I'll help everyone with understanding and starting their projects
- Take one of the datasets we've looked at or create a dataset you expect your new technique to work well on, and apply your new technique to it
- Write up a comparison with the old and new techniques

Final Project Ideas

- Random Forest: generate several decision trees randomly (pick a few test cutoffs randomly instead of looking over the whole sample), and have them "vote" on the output (take average of the outputs)
  - Should reduce overtraining
  - Can be further extended with boosting: currently misclassified data points are given bigger weights in subsequent trees
- Bootstrap distributions: run logistic regression several times, where new datasets are created by randomly choosing a new dataset of the same size as the old but *allowing repeats*
  - Can find the variance of the β in the "bootstraps", which acts as an estimate of the variance of the parameter
  - when finding e.g. β for logistic regression, we don't see how "important" the various variables are. With this, we can see if they are significantly away from 0
- Momentum in SGD: add a momentum parameter in gradient descent
- Simple SVM (support vector machine): take multiple logisitic regression-like analysis and add additional variables derived from the input, run over the extended dataset
  - E.g. if one input x, create the dataset (x, x^2) and run a multiple logistic regression over the (x, x^2) dataset. Can you come up with a dataset that would benefit?
- Simple CNN (ambitious?): create a simple CNN filter for MNIST