Decision Trees and their variants

Naeemullah Khan
naeemullah.khan@kaust.edu.sa

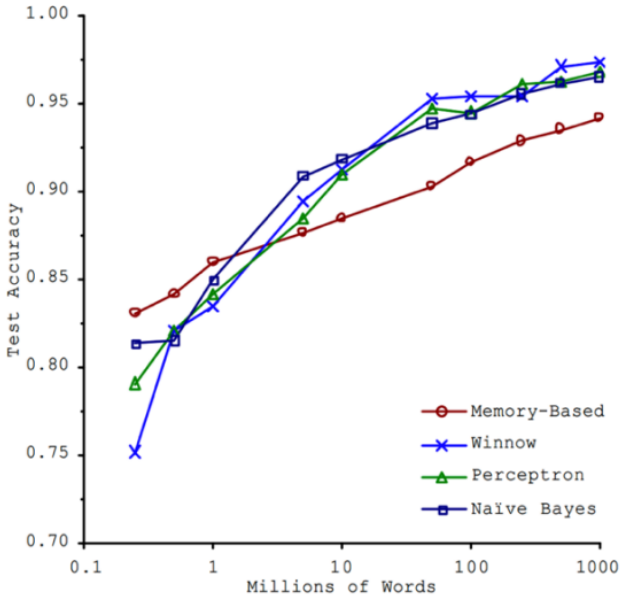
جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

KAUST Academy
King Abdullah University of Science and Technology

July 25, 2025

Why Large-Scale ML?

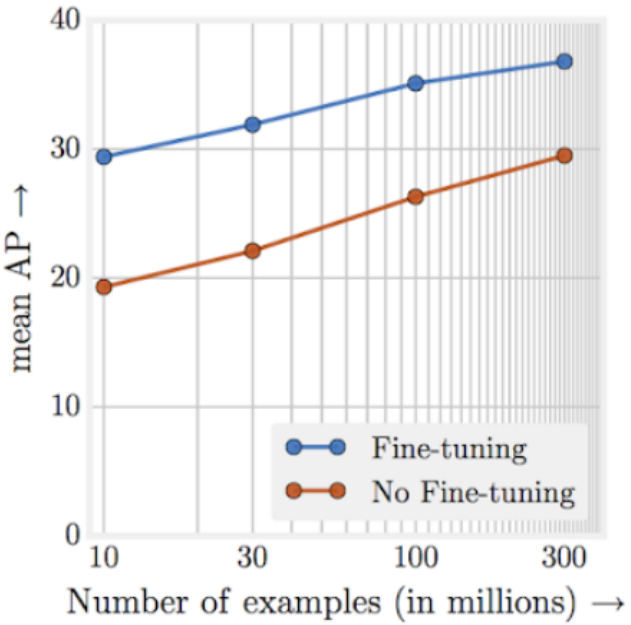
- **Brawn or Brains?**
  - In 2001, Microsoft researchers ran a test to evaluate 4 different approaches to ML-based language translation
- **Findings:**
  - Size of the dataset used to train the model mattered more than the model itself
  - As the dataset grew large, performance difference between the models became small



Banko, M. and Brill, E. (2001), "Scaling to Very Large Corpora for Natural Language Disambiguation"

Why Large-Scale ML?

- **The Unreasonable Effectiveness of Data**
  - In 2017, Google revisited the same type of experiment with the latest Deep Learning models in computer vision
- **Findings:**
  - Performance increases logarithmically based on volume of training data
  - Complexity of modern ML models (i.e., deep neural nets) allows for even further performance gains
- Large datasets + large ML models = amazing results!!



"Revisiting Unreasonable Effectiveness of Data in Deep Learning Era": https://arxiv.org/abs/1707.02968

Why Worry About Non-Deep Models?

- A few reasons why this is important:
- They outperform DL models in certain tasks.
- Deep models are often hard to scale and require lots of data. Traditional models allow you to encode prior knowledge better and give you more control.
- Combine: ideas from several ML models, e.g., GNNs
- Rule of thumb: If working on a well understood problem use deep learning. If working on a new problem use techniques we'll discuss here.

Preface: Decision Trees

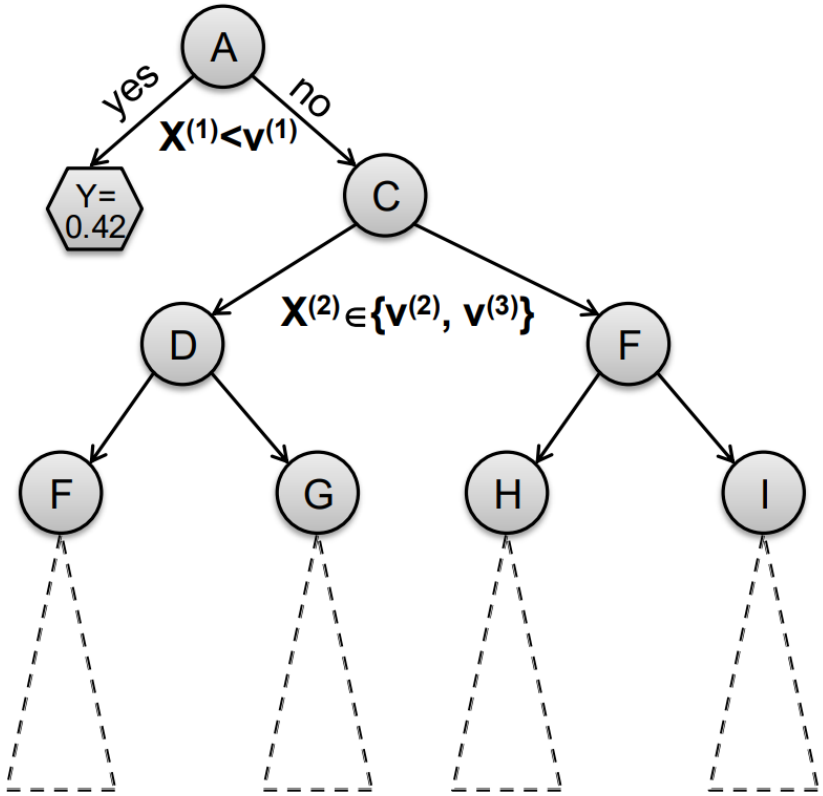
- Decision trees are part of ML since 1980s
  - Introduced by Leo Breiman in 1984
  - Notable algorithms: ID3, C4.5
- More recent innovations include:
  - Boosted decision trees (gradient boosted DT)
  - Random forest
- Even though DTs are old, hand-engineered and heuristic, they are a method of choice for tabular data and for Kaggle competitions.

Decision Tree Learning

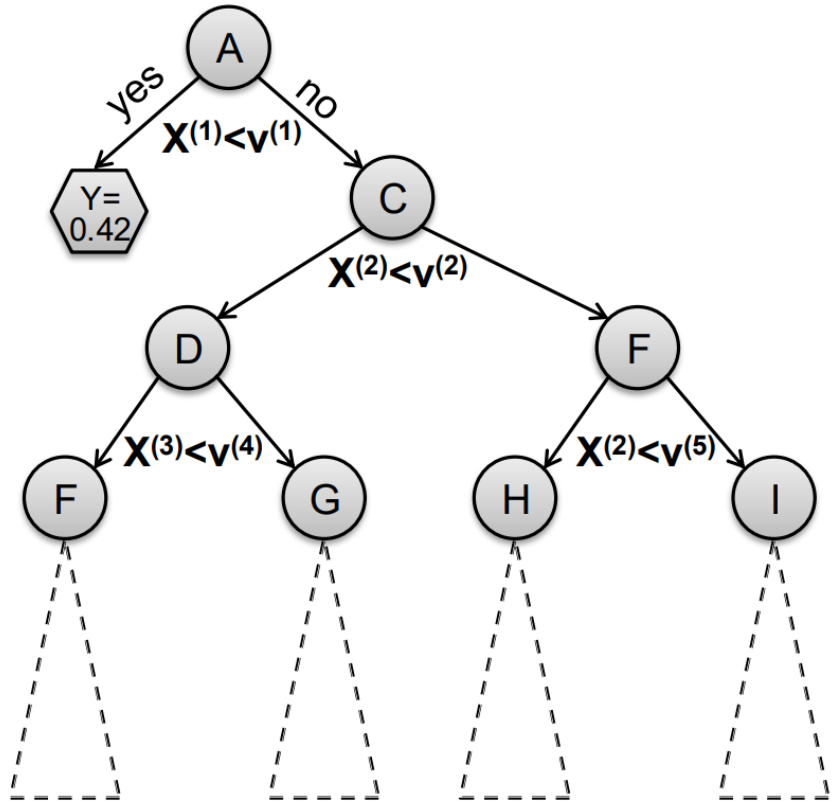
- Given one attribute (e.g., lifespan), try to predict the value of new people's lifespans by a subset of the other available attributes
- Input attributes:
  - d features/attributes: $x^{(1)}, x^{(2)}, \ldots, x^{(d)}$
  - Each $x^{(j)}$ has domain O_j
    - Categorical: $O_j = \{male, female, nonbinary\}$
    - Numerical: $H_j = (1, 200)$
  - Y is output variable with domain O_Y:
    - Categorical: Classification e.g. $Y =$ eye color
    - Numerical: Regression e.g. $Y =$ lifespan
- Data D:
  - n examples (x_i, y_i) where x_i is a d-dim feature vector, $y_i \in O_Y$ is output variable
- Task:
  - Given an input data vector x predict output label y

Decision Trees

- A Decision Tree is a tree-structured plan of a set of attributes to test in order to predict the output

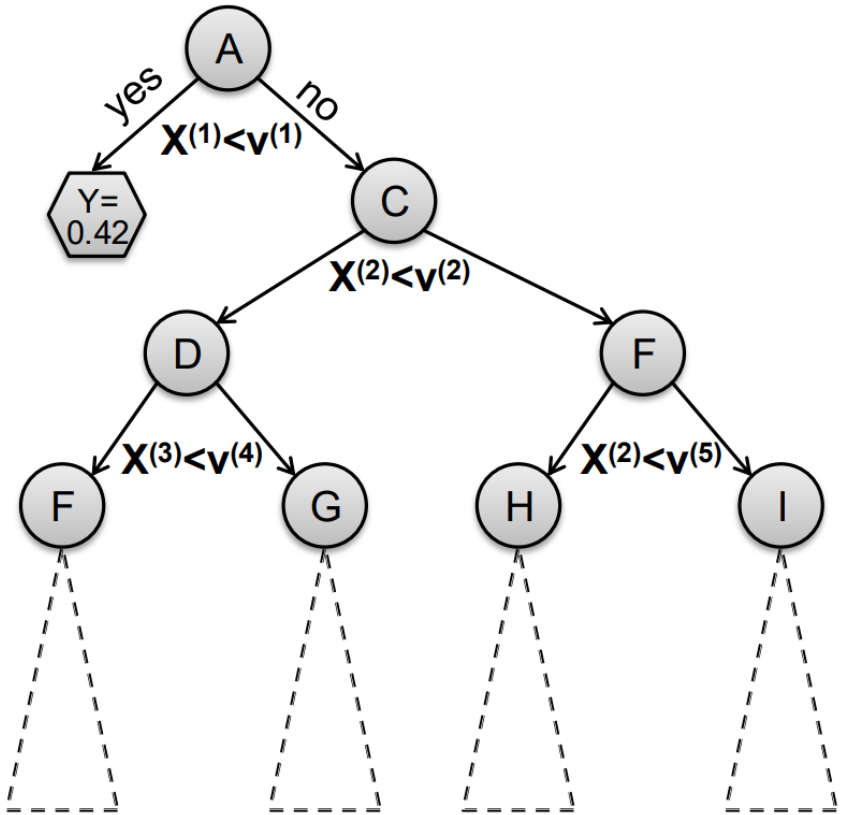


- **Decision trees:**
  - Split the data at each internal node
  - Each leaf node makes a prediction
- **Lecture today:**
  - Binary splits: $x^{(j)} < v$
  - Numerical attributes
  - Regression

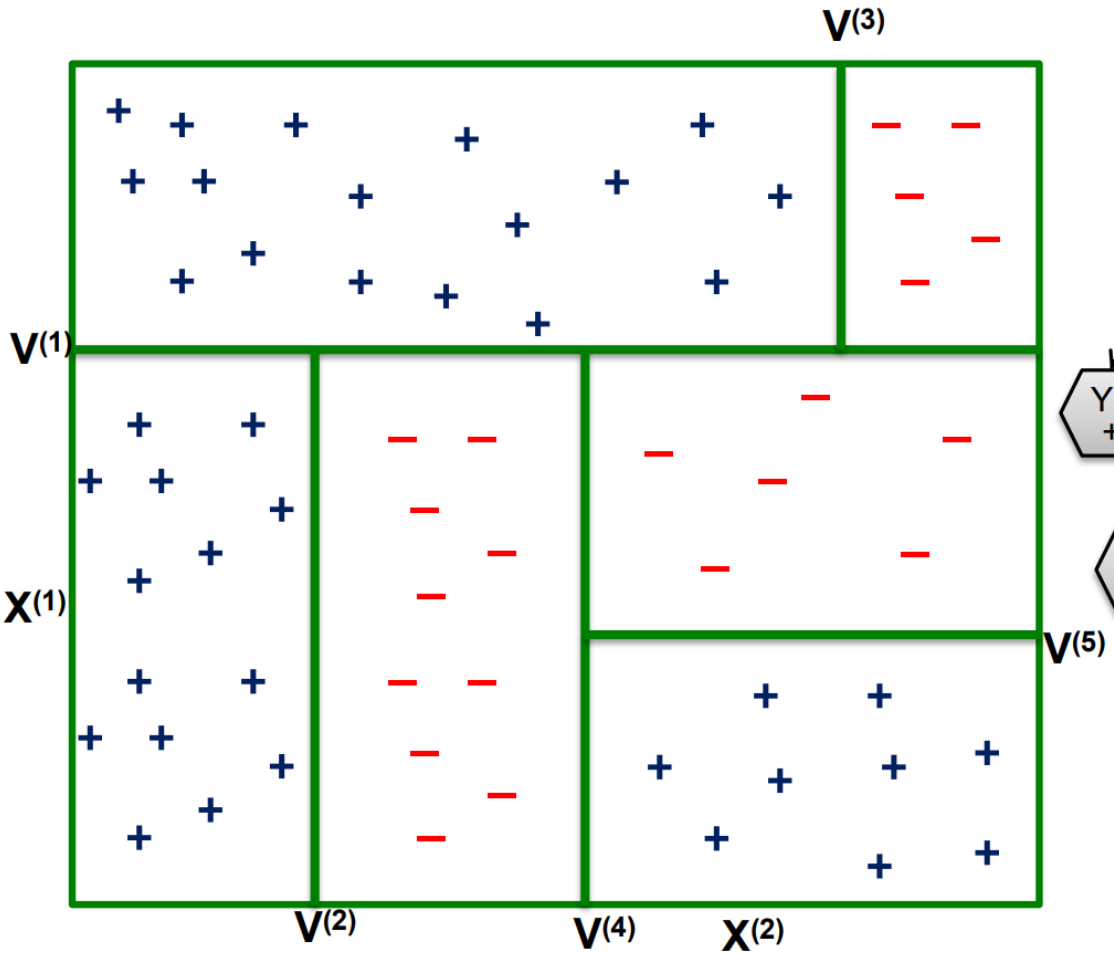


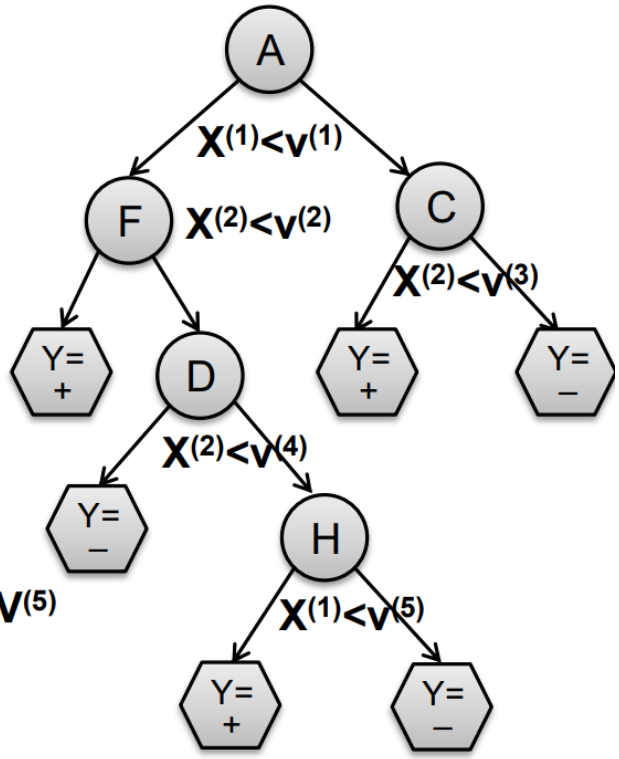
How to make predictions?

- Input: Example x_i
- Output: Predicted $\hat{y}_i$
- "Drop" x_i down the tree until it hits a leaf node
- Predict the value stored in the leaf that x_i hits

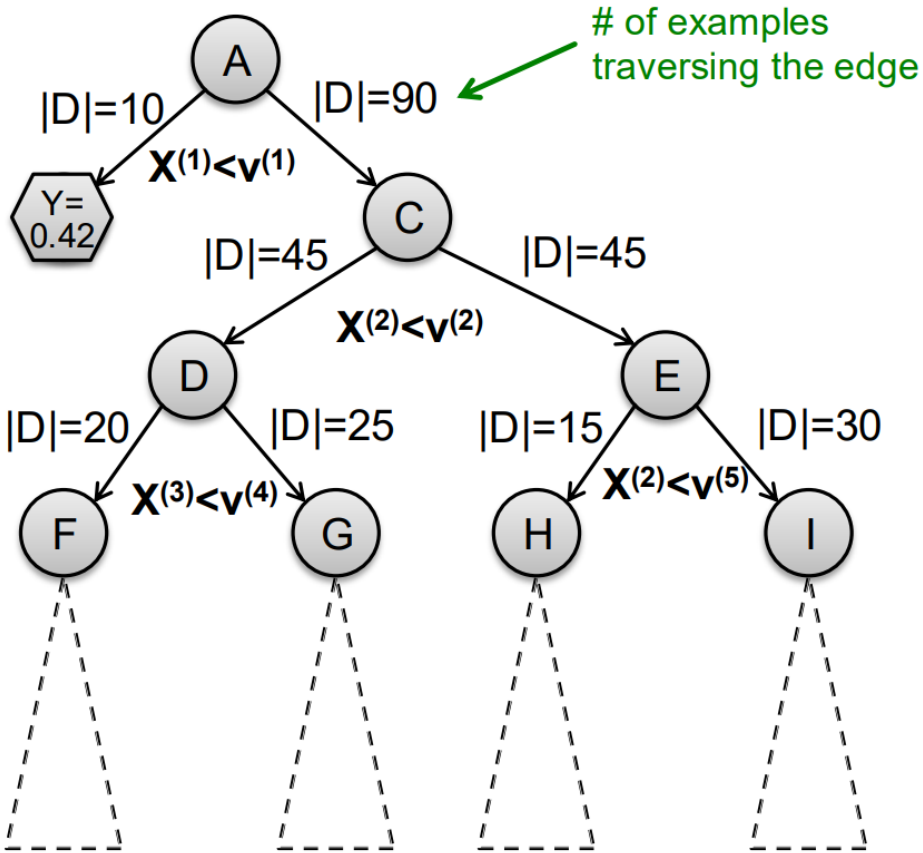


- **Alternative view:**



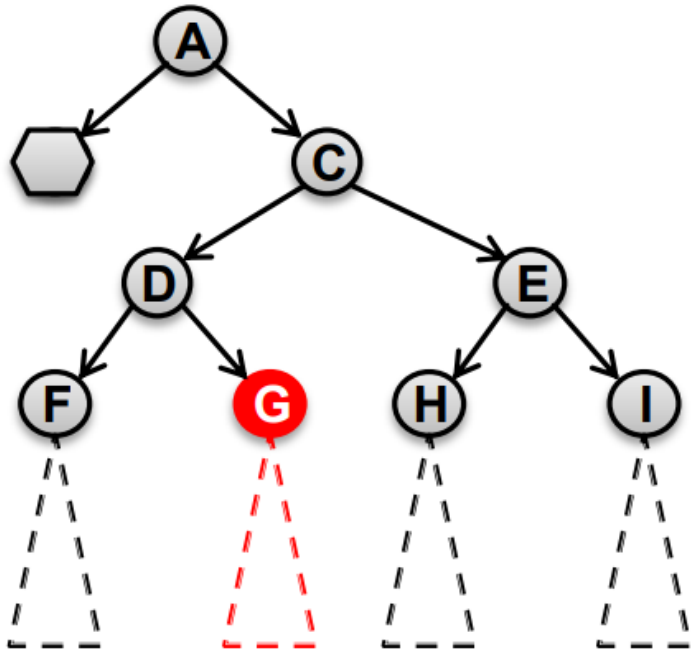


- Training dataset D^*, $|D^*| = 100$ examples



How to construct a tree?

- Imagine we are currently at some node G
  - Let D_G be the data that reaches G
- There is a decision we have to make: Do we continue building the tree?
  - If yes, which variable and which value do we use for a split?
    - Continue building the tree recursively
  - If not, how do we make a prediction?
    - We need to build a "predictor node"



3 steps in constructing a tree

Algorithm 1 **BuildSubtree**

Require: Node n, Data $D \subseteq D^*$

1: $(n \rightarrow \text{split}, D_L, D_R) = \text{FindBestSplit}(D)$ (1)
2: **if** $\text{StoppingCriteria}(D_L)$ **then** (2)
3: $n \rightarrow \text{left\_prediction} = \text{FindPrediction}(D_L)$ (3)
4: **else**
5: **BuildSubtree** $(n \rightarrow \text{left}, D_L)$
6: **if** $\text{StoppingCriteria}(D_R)$ **then**
7: $n \rightarrow \text{right\_prediction} = \text{FindPrediction}(D_R)$
8: **else**
9: **BuildSubtree** $(n \rightarrow \text{right}, D_R)$

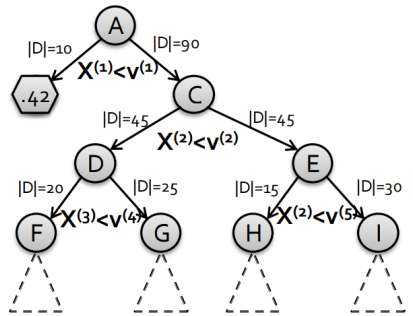
- Requires at least a single pass over the data!

How to construct a tree?

- (1) How to split? Pick attribute & value that optimizes some criterion
- Regression: Purity
  - Find split $(X^{(i)}, v)$ that creates D, D_L, D_R: parent, left, right child datasets and maximizes:

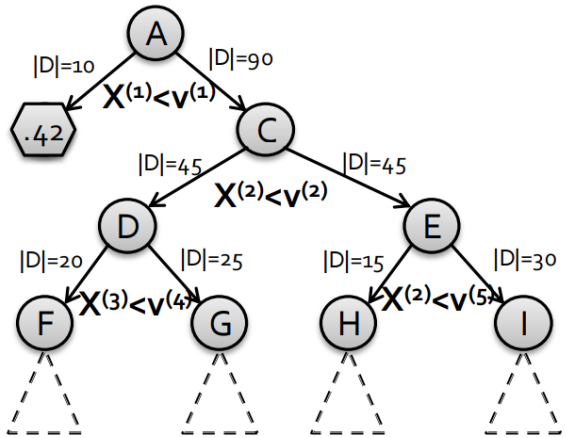
    $$|D| \cdot Var(D) - (|D_L| \cdot Var(D_L) + |D_R| \cdot Var(D_R))$$

  - $Var(D) = \frac{1}{|D|} \sum_{i \in D} (y_i - \bar{y})^2$ ... variance of y_i in D



How to construct a tree?

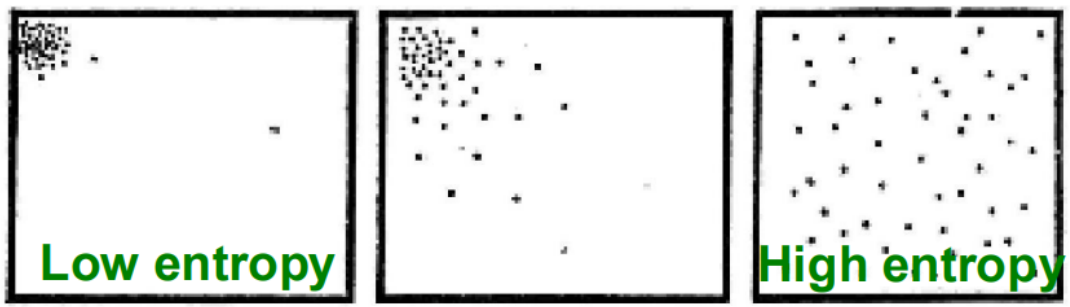
- (1) How to split? Pick attribute & value that optimizes some criterion
- Classification: Information Gain
  - Measures how much a given attribute X tells us about the class Y
  - $IG(Y \mid X)$: We must transmit Y over a binary link. How many bits on average would it save us if both ends of the line knew X?



The entropy of X:

$$H(X) = -\sum_{j=1}^{m} p(X_j) \log p(X_j)$$

- "High Entropy": X is from a uniform (boring) distribution
  - A histogram of the frequency distribution of values of X is **flat**
- "Low Entropy": X is from a varied (peaks/valleys) distribution
  - A histogram of the frequency distribution of values of X would have many lows and one or two highs



- Suppose I want to predict Y and I have input X
  - X = College Major
  - Y = Likes "Casablanca"

X	Y
Math	Yes
History	No
CS	Yes
Math	No
Math	No
CS	Yes
Math	Yes
History	No

- From this data we estimate
  - $P(Y = Yes) = 0.5$
  - $P(X = Math \wedge Y = No) = 0.25$

Why Information Gain? Entropy (cont.)

- $P(X = Math) = 0.5$
- $P(Y = Yes \mid X = History) = 0$

- Note:
  - $H(Y) = -\frac{1}{2}\log_2(\frac{1}{2}) - \frac{1}{2}\log_2(\frac{1}{2}) = 1$
  - $H(X) = 1.5$

Why Information Gain? Entropy

- Suppose I want to predict Y and I have input X
  - X = College Major
  - Y = Likes "Casablanca"
- Def: Specific Conditional Entropy
  - $H(Y \mid X = v)$ = The entropy of Y among only those records in which X has value v
- Example:
  - $H(Y \mid X = Math) = 1$
  - $H(Y \mid X = History) = 0$
  - $H(Y \mid X = CS) = 0$

Why Information Gain?

- Suppose I want to predict Y and I have input X
  - X = College Major
  - Y = Likes "Casablanca"
- Def: Conditional Entropy
  - $H(Y \mid X)$ = The average specific conditional entropy of Y
  - = the entropy of Y, conditioned on X, if you choose a record at random
  - $= \sum_j P(X = v_j) H(Y \mid X = v_j)$

Why Information Gain?

- Suppose I want to predict Y and I have input X
  - X = College Major
  - Y = Likes "Casablanca"
- $H(Y \mid X)$ = The average specific conditional entropy of Y

$$= \sum_j P(X = v_j) H(Y \mid X = v_j)$$

- Example:

v_j	$P(X = v_j)$	$H(Y \mid X = v_j)$
Math	0.5	1
History	0.25	0
CS	0.25	0

- So: $H(Y \mid X) = 0.5 \cdot 1 + 0.25 \cdot 0 + 0.25 \cdot 0 = 0.5$

Why Information Gain?

- Suppose I want to predict Y and I have input X
  - X = College Major
  - Y = Likes "Casablanca"
- Def: Information Gain
  - $IG(Y \mid X)$ = I must predict Y. How much information do I get about Y if I knew X?

$$IG(Y \mid X) = H(Y) - H(Y \mid X)$$

- Example:
  - $H(Y) = 1$
  - $H(Y \mid X) = 0.5$
  - Thus $IG(Y \mid X) = 1 - 0.5 = 0.5$

What is Information Gain used for?

- Suppose you are trying to predict whether someone is going to live past 80 years
- From historical data you might find:
  - $IG(LongLife \mid HairColor) = 0.01$
  - $IG(LongLife \mid Smoker) = 0.4$
  - $IG(LongLife \mid Gender) = 0.25$
  - $IG(LongLife \mid LastDigitOfSSN) = 0.00001$
- IG tells us how much information about Y is contained in X
  - So attribute X that has high $IG(Y \mid X)$ is a good split!

3 steps in constructing a tree

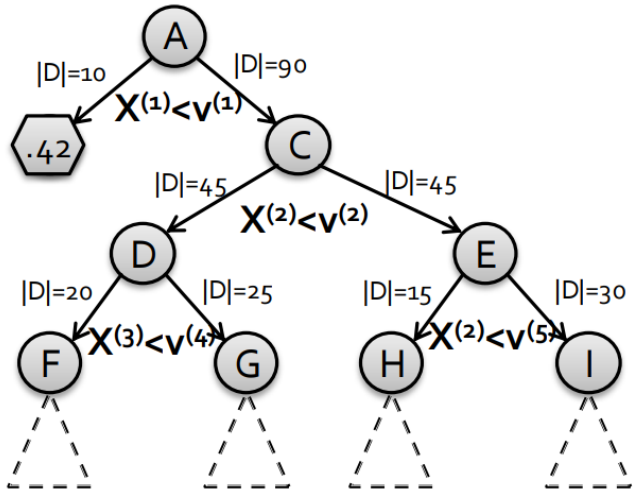
Algorithm 1 **BuildSubtree**

Require: Node n, Data $D \subseteq D^*$

1: $(n \rightarrow \text{split}, D_L, D_R) = \text{FindBestSplit}(D)$ (1)
2: **if** $\text{StoppingCriteria}(D_L)$ **then** (2)
3: $n \rightarrow \text{left\_prediction} = \text{FindPrediction}(D_L)$ (3)
4: **else**
5: **BuildSubtree** $(n \rightarrow \text{left}, D_L)$
6: **if** $\text{StoppingCriteria}(D_R)$ **then**
7: $n \rightarrow \text{right\_prediction} = \text{FindPrediction}(D_R)$
8: **else**
9: **BuildSubtree** $(n \rightarrow \text{right}, D_R)$

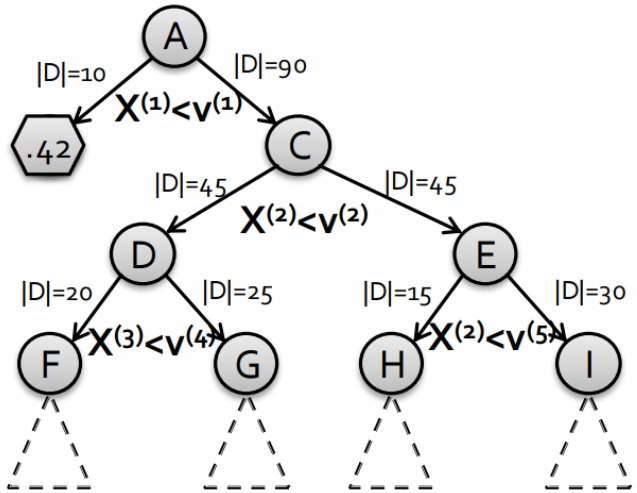
When to stop?

- **(2) When to stop?**
- Many different heuristic options
- **Two ideas:**
  - (1) When the leaf is "pure"
    - The target variable does not vary too much: $Var(y) < \varepsilon$
  - (2) When # of examples in the leaf is too small
    - For example, $|D| \leq 100$



How to predict?

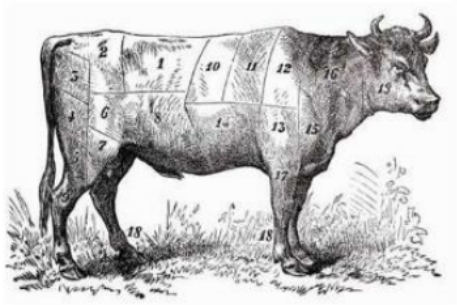
- **(3) How to predict?**
- **Many options**
  - **Regression:**
    - Predict average y_i of the examples in the leaf
    - Build a linear regression model on the examples in the leaf
  - **Classification:**
    - Predict most common y_i of the examples in the leaf



- **Characteristics**
  - Classification & Regression
    - Multiple ($\sim$10) classes
  - Real valued and categorical features
  - Few (hundreds) of features
  - Usually dense features
  - Complicated decision boundaries
    - Early stopping to avoid overfitting!
- **Example applications**
  - User profile classification
  - Landing page bounce prediction

- **Decision trees are the single most popular data mining tool:**
  - Interpretable
  - Easy to implement
  - Easy to use
  - Computationally cheap
  - It's possible to mitigate overfitting (i.e., with ensemble methods)
  - **They do classification as well as regression!**

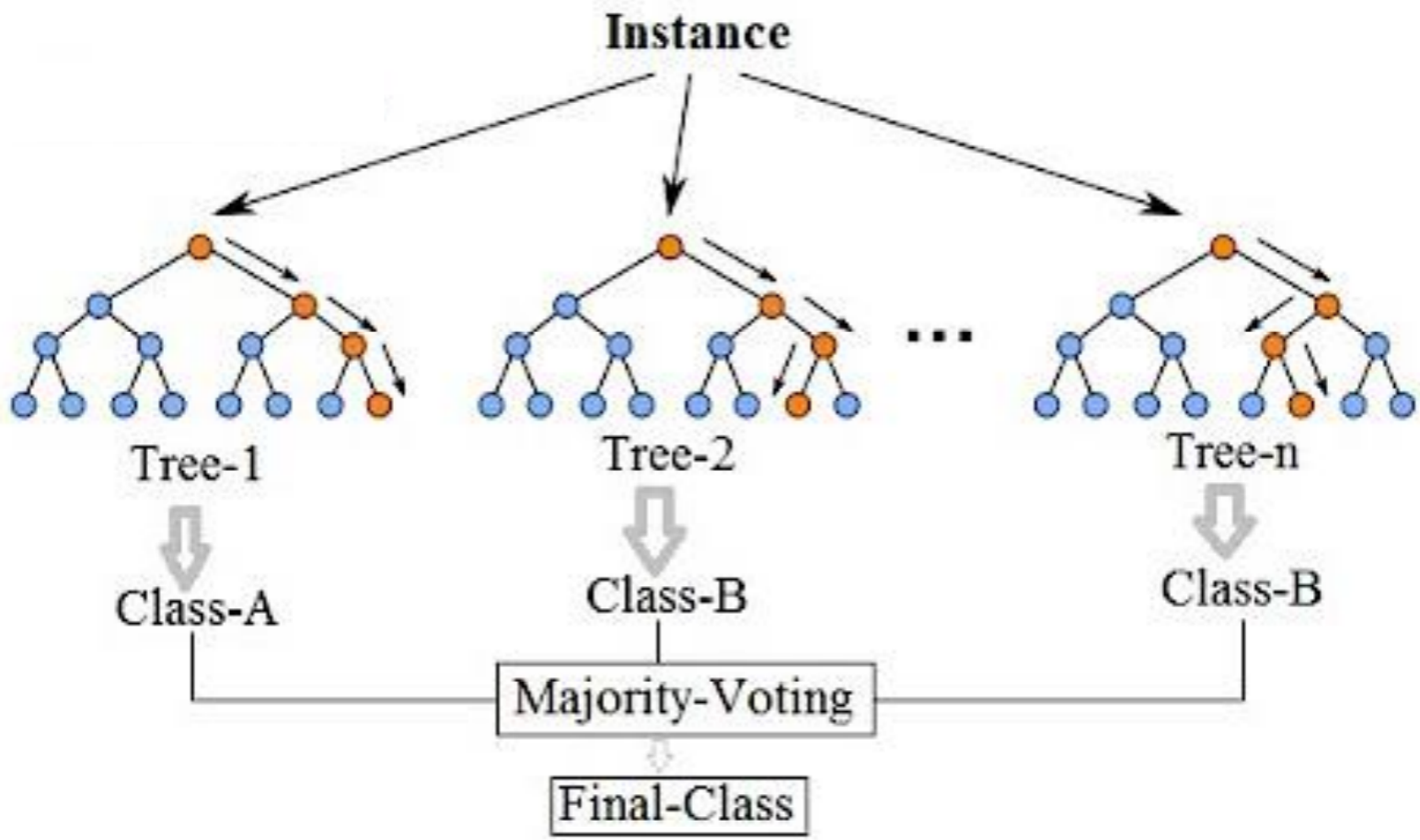
- **Learn multiple trees and combine their predictions**
  - Fix overfitting/underfitting problem in decision trees
  - Gives better performance in practice
  - The "wisdom of the crowds"
- **The parable of the ox (Sir Francis Galton, 1906)**
  - 787 people guessed the weight of an ox
  - Avg crowd guess: 1,197 pounds
  - True weight: 1,198 pounds



Bagging (bootstrap aggregation):

- Learns multiple trees in parallel over independent samples of the training data
- **1) Bootstrapping:** Given a dataset D on n data points: Create multiple datasets D' of n points by sampling from D with replacement:
  - 37% points in D' will be duplicates, 63% will be unique
- **2) Parallel training:** Train decision trees on samples independently and in parallel
- **3) Aggregation:** Depending on the task, an average or majority of the predictions are computed for a more accurate estimate

(1): Bagging Decision Trees



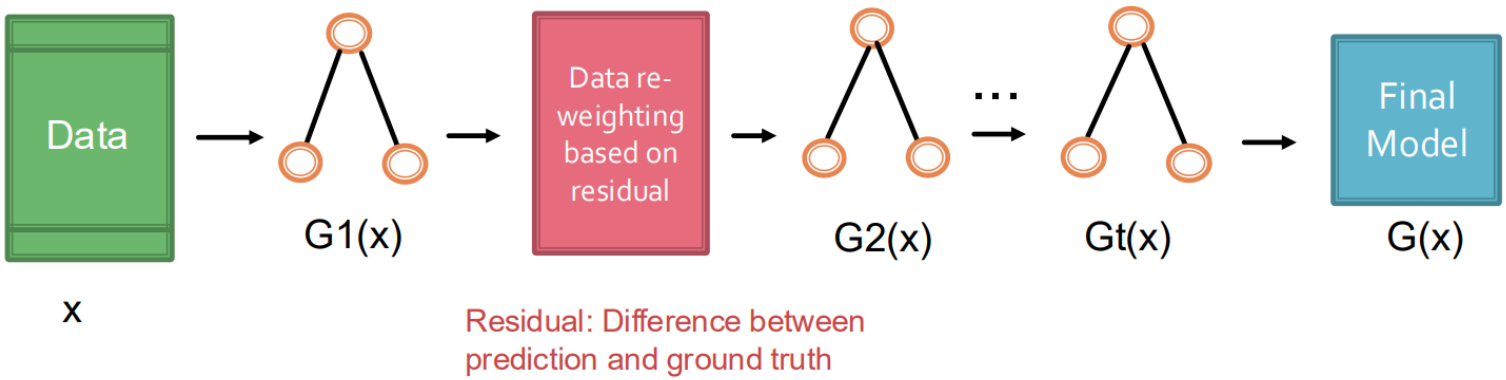
(1): Instance Bagging

- Decision trees are greedy
  - They choose which variable to split on using a greedy algorithm that maximizes purity or information gain
- Even with Bagging, the decision trees can have a lot of structural similarities and correlation in their predictions
  - If one feature is very strong predictor, then every tree will select it, causing trees to be correlated.
- But ensemble learning works best with independent predictors

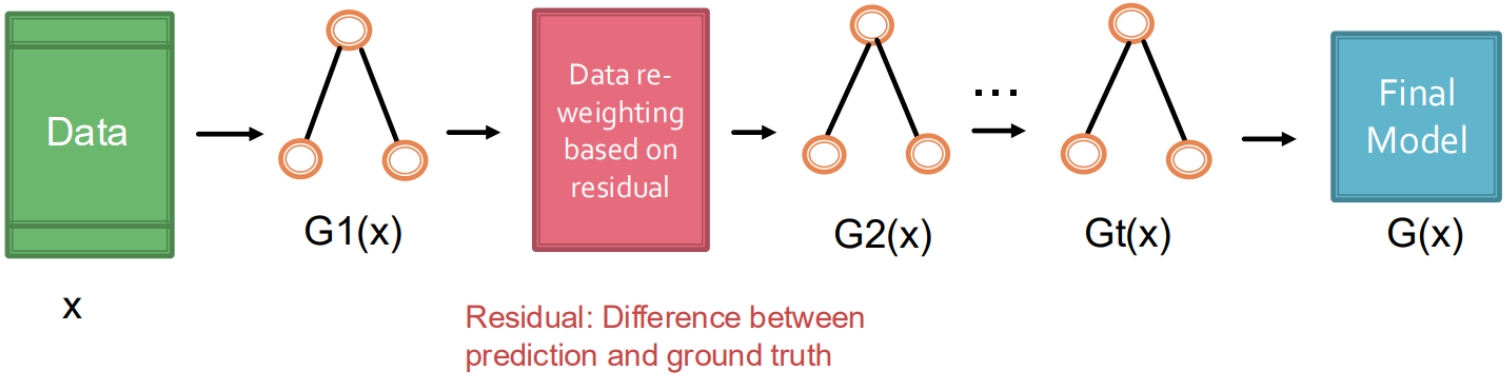
(2) Improvement: Random Forests

- Train a Bagged Decision Tree
- But use a modified tree learning algorithm that selects (at each candidate split) a random subset of features
  - If we have d features, consider $\sqrt{d}$ random features
- This is called: Feature bagging
  - Benefit: Breaks correlation between trees
- Random Forests achieve state-of-the-art results in many classification problems!

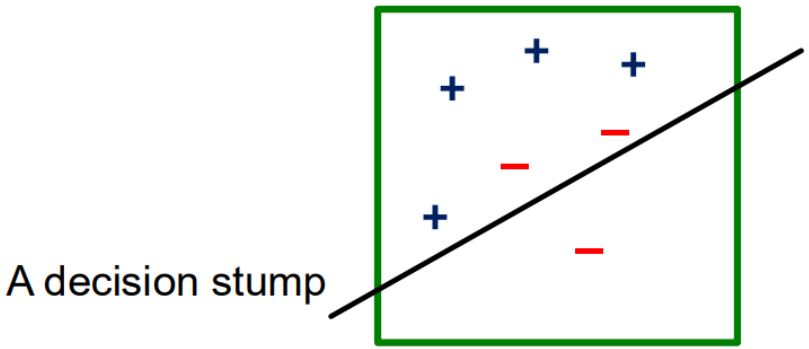
- **Boosting:** Another ensemble learning algorithm
  - Combines the outputs of many "weak" classifiers to produce a powerful "committee"
  - Learns multiple trees sequentially, each trying to improve upon its predecessor
  - Final classifier is weighted sum of the individual classifiers



- **We will show 2 examples:**
  - **Example 1: AdaBoost**
    - Where each $G_t(x)$ is a one-level decision tree
  - **Example 2: Gradient Boosted Decision Trees**
    - Where each $G_t(x)$ is a multi-level decision tree



- **Decision "stumps"**:
  - 1-level decision tree
  - A decision boundary based on one feature
    - E.g.: If someone is not a smoker, then predict them to live past 80 years old
  - Building blocks of AdaBoost algorithm
  - **Decision stump** is a **weak learner**



Boosting theory:
if weak learners have >50% accuracy then we can learn a perfect classifier.

Suppose we have training data

$$\{(x_i, y_i)\}_{i=1}^{N}, \quad y_i \in \{1, -1\}$$

- Initialize equal weights for all observations: $w_i = \frac{1}{N}$
- At each iteration t:
  1. Train a stump G_t using data weighted by w_i
  2. Compute the **misclassification error** adjusted by w_i
  3. Compute the weight of the current tree α_t
  4. Reweight each observation based on prediction accuracy

Update Step

- Calculate the weighted misclassification error:

$$\text{err}_t = \frac{\sum_{i=1}^{N} w_i \, I(y_i \neq G_t(x_i))}{\sum_{i=1}^{N} w_i}$$

- Use the error score to weight the current tree in the final classifier:

$$\alpha_t = \log\left(\frac{1 - \text{err}_t}{\text{err}_t}\right)$$

  A classifier with 50% accuracy is given a weight of zero;

- Use misclassification error and tree weight to reweight the training data:

$$w_i \leftarrow w_i \exp\left[\alpha_t \, I(y_i \neq G_t(x_i))\right]$$

  Training instances that are harder to classify get higher weight

Final Prediction

- Final prediction is a weighted sum of the predictions from each stump:

$$G(x) = \text{sign}\left[\sum_{t=1}^{T} \alpha_t G_t(x)\right]$$

- More accurate trees are weighted higher in the final model

AdaBoost: Summary

1. Initialize the observation weights $w_i = \frac{1}{N}$, $i = 1, 2, \ldots, N$.
2. For $m = 1$ to M:
   - 2.1 Fit a classifier $G_m(x)$ to the training data using weights w_i.
   - 2.2 Compute
   $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i \mathbb{I}(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}$$
   - 2.3 Compute $\alpha_m = \log\left(\frac{1-\text{err}_m}{\text{err}_m}\right)$.
   - 2.4 Set $w_i \leftarrow w_i \cdot \exp\left[\alpha_m \cdot \mathbb{I}(y_i \neq G_m(x_i))\right], \quad i = 1, 2, \ldots, N$.
3. Output
$$G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$$

AdaBoost Conclusion

- **Iteratively train weak learners (decision stumps) to form a strong model:**
  - Trees with high accuracy are given more weights in the final model
  - Misclassified data get higher weights in the next iteration
- AdaBoost is the equivalent to additive training with the exponential loss (Friedman et al. 2000)
- We will talk about additive training in more general scenarios next!

Gradient Boosted Decision Trees

- Idea: Optimize an Additive model
- Additive prediction model:

$$\hat{y}_i = \sum_{t=1}^{T} f_t(x_i)$$

  - Here f_t can be multi-level!

- Objective (cost) function:

$$\text{obj}(\theta) = \sum_{i=1}^{N} l(y_i, \hat{y}_i) + \sum_{t=1}^{T} \omega(f_t)$$

  - $\omega(f_t)$ is a regularization term that models the complexity of the tree.

- Use Additive model to train sequentially:
  - Start from constant prediction, add a new decision tree f_i each time:

$$\hat{y}_i^{(0)} = 0$$

$$\hat{y}_i^{(1)} = f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i)$$

$$\hat{y}_i^{(2)} = f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i)$$

$$\dots$$

$$\hat{y}_i^{(t)} = \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)$$

How to decide which f to add?

- Prediction at round t is:

$$\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$$

  - Where we need to decide what $f_t()$ to add

- Goal: Find tree $f_t(\cdot)$ that minimizes loss $l()$:

$$\text{obj}^{(t)} = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t)}\right) + \omega(f_t)$$

- y_i: The ground-truth label
- $\hat{y}_i^{(t-1)} + f_t(x_i)$: The prediction made at round t
- $\omega(f_t)$: The model complexity

How to decide which f to add?

$$\text{obj}^{(t)} = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \omega(f_t)$$

- Take Taylor expansion of the objective:

$$g(x+\Delta) \approx g(x) + g'(x)\Delta + \frac{1}{2}g''(x)\Delta^2$$

- So, we get the approximate objective:

$$\text{obj}^{(t)} = \sum_{i=1}^{n} \left[l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \omega(f_t)$$

- where:

$$g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)}), \quad h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$$

Our New Goal

- Our new goal: Find tree f_t that:

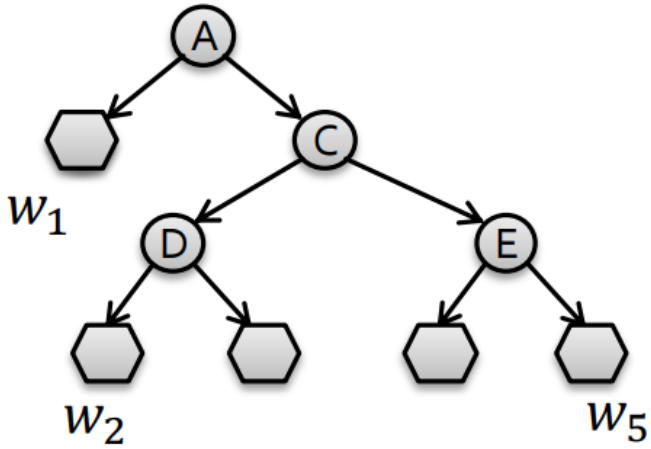
$$\sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \omega(f_t)$$

- Why spend so much effort to derive the objective, why not just grow trees...
  - **Theoretical benefit**: Know what we are learning
  - **Engineering benefit**:
    - g and h come from definition of loss function
    - Learning f_t only depends on the objective via g and h
    - We can now directly learn trees that optimize the loss (rather than using some heuristic procedure)

- Every leaf j has a weight w_j
  - We will predict w_j for any data belonging to leaf j

$$f_t(x) = w_{q(x)}$$

where $q(x)$ indicates the leaf node that data point x belongs to.



Define a Tree (cont.)

- Define complexity of tree f as:

$$\Omega(f) = \gamma \cdot T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

where:

- T: number of leaves of tree f
- γ: cost of adding a leaf to the tree f

- **Define:**
  - The set of examples in the leaf j:

$$I_j = \{i \mid q(x_i) = j\}$$

  where $q(x)$ denotes the leaf that data point x belongs to.

- The parameters that depend on the loss:

$$G_j = \sum_{i \in I_j} g_i \qquad H_j = \sum_{i \in I_j} h_i$$

- Then the objective function becomes:

$$\text{obj}^{(t)} = \sum_{j=1}^{T} \left[G_j w_j + \frac{1}{2}(H_j + \lambda) w_j^2 \right] + \gamma T$$

How to find a single tree f_t

- Given a tree f_t, we know how to
  - Calculate the score for f:

$$\text{Obj} = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j+\lambda} + \gamma T$$

  - And then set optimal weights for the chosen f:

$$w_j^* = -\frac{G_j}{H_j+\lambda}$$

- In principle we could:
  - Enumerate possible tree structures f and take the one that minimizes Obj

How to find a single tree f_t

- In practice we grow tree greedily:
  - Start with tree with depth 0
  - For each leaf node in the tree, try to add a split
  - The change of the objective after adding a split is:

$$\text{Gain} = \frac{1}{2}\left[\frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda}\right] - \gamma$$

  - Take the split that gives best gain
- Next: **How to find the best split?**

How to Find the Best Split?

- For each node, enumerate over all features:
  - For each feature, sort the instances by feature value
  - Use a linear scan to decide the best split along that feature
  - Take the best split solution along all the features
- Pre-stopping:
  - Stop split if the best split has negative gain
  - But maybe a split can benefit future splits
- Post-pruning:
  - Grow a tree to maximum depth, recursively prune all the leaf splits with negative gain

Summary: GBDT Algorithm

- Add a new tree $f_t(x)$ in each iteration
  - Compute necessary statistics for our objective

$$g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}), \quad h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$$

  - Greedily grow the tree that minimizes the objective:

$$Obj = -\frac{1}{2} \sum_{j=1}^{T} \frac{G_j^2}{H_j + \lambda} + \gamma T$$

- Add $f_t(x)$ to our ensemble model

$$y^{(t)} = y^{(t-1)} + \epsilon f_t(x_i)$$

  ϵ is called step-size or shrinkage, usually set around 0.1. Goal: prevent overfitting.

- Repeat until we use M ensemble of trees

- **XGBoost**: **eXtreme Gradient Boosting**
  - A highly scalable implementation of gradient boosted decision trees with regularization
- Widely used by data scientists and provides state-of-the-art results on many problems!
- System optimizations:
  - Parallel tree constructions using column block structure
  - Distributed Computing for training very large models using a cluster of machines.
  - Out-of-Core Computing for very large datasets that don't fit into memory.

Summary of the Lecture

- Basics of supervised learning
- **Decision Trees**
  - **Key idea**: split data at each internal node, make prediction at each leaf node
  - **How to construct a tree**: Information Gain
- **Ensemble of decision trees:**
  - **Bagging**: Random forests
  - **Boosting**: Boosted decision trees

References

[1] Jure Leskovec, Anand Rajaraman, Jeff Ullman. *Mining of Massive Datasets.* MMDS 2020. http://www.mmds.org/.