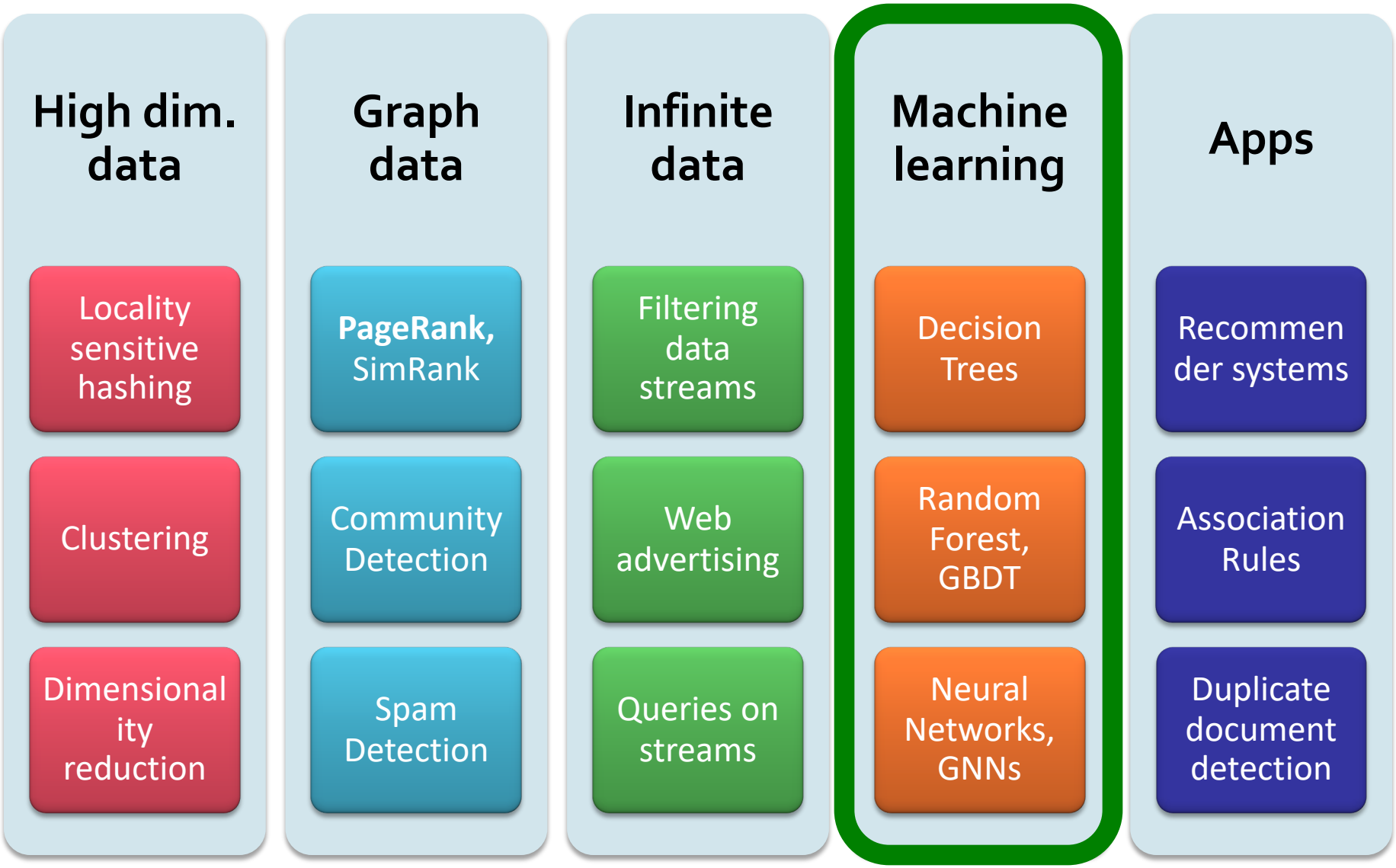
Large Scale Machine Learning: Decision Trees

CS246: Mining Massive Datasets
Jure Leskovec, Stanford University
Mina Ghashami, Amazon

http://cs246.stanford.edu

New Topic: ML!


High dim. data
Locality sensitive hashing
Clustering
Dimensionality reduction
Graph data
PageRank, SimRank
Community Detection
Spam Detection
Infinite data
Filtering data streams
Web advertising
Queries on streams
Machine learning
Decision Trees
Random Forest, GBDT
Neural Networks, GNNs
Apps
Recommender systems
Association Rules
Duplicate document detection

Supervised Learning

Given some data:

- "Learn" a function to map from the **input** to the **output**

- **Given:**

  Training examples $\left(\boldsymbol{x_i}, \boldsymbol{y_i} = \boldsymbol{f}(\boldsymbol{x_i})\right)$ for some unknown function $\boldsymbol{f}$
- **Find:**

  A good approximation to $\boldsymbol{f}$

Many Other ML Paradigms

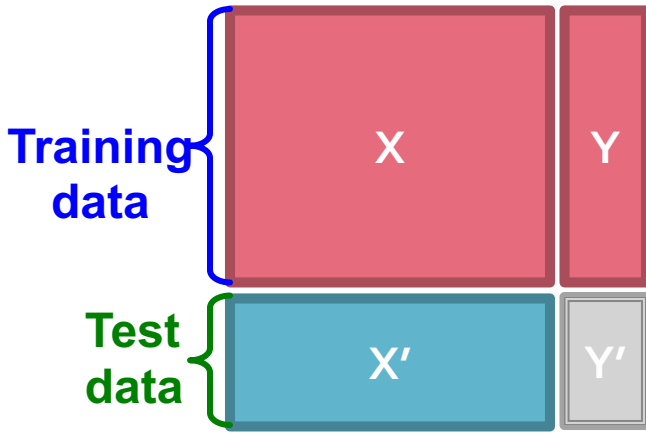
- **Supervised:**
  - Given "labeled data" $\{x, y\}$, learn $f(x) = y$
- **Unsupervised:**
  - Given only "unlabeled data" $\{x\}$, learn $f(x)$
- **Semi-supervised:**
  - Given some labeled $\{x, y\}$ and some unlabeled data $\{x\}$, learn $f(x) = y$
- **Active learning:**
  - When we predict $f(x) = y$, we then receive true y^*
- **Transfer learning:**
  - Learn $f(x)$ so that it works well on new domain $f(z)$

Supervised Learning

- **Would like to do prediction:**
  estimate a function $f(x)$ so that $y = f(x)$
- **Where *y* can be:**
  - **Real number:** Regression
  - **Categorical: Classification**
  - **Complex object:**
    - Ranking of items, Parse tree, etc.
- **Data is labeled:**
  - Have many pairs $\{(x, y)\}$
    - **x** … vector of binary, categorical, real valued features
    - **y** … class, or a real number

Supervised Learning

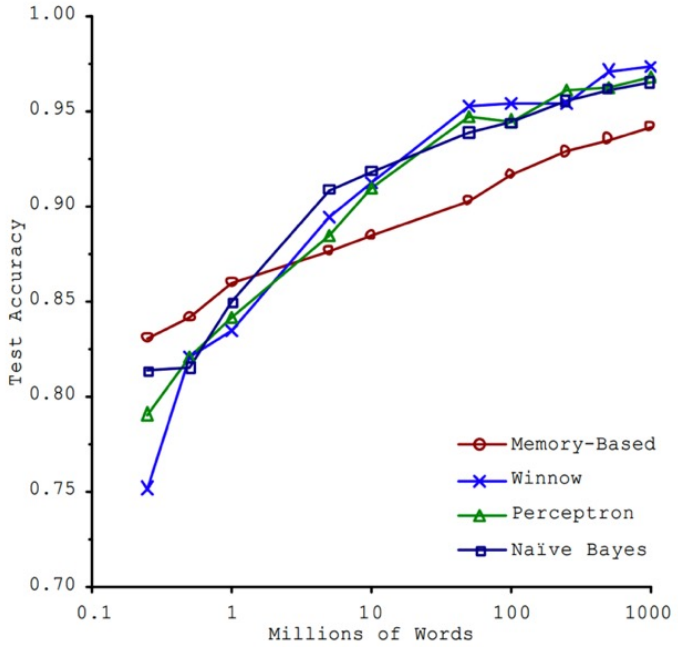
- **Task:** Given data (X, Y) build a model $f()$ to predict Y' based on X'
- **Strategy:** **Estimate $y = f(x)$ on (X, Y)**
  Hope that the same $f(x)$ also works to predict unknown Y'



  - The **"hope"** is called **generalization**
    - **Overfitting:** **If $f(x)$ predicts well Y but is unable to predict Y'**
  - **We want to build a model that generalizes well to unseen data**

Why Large-Scale ML?

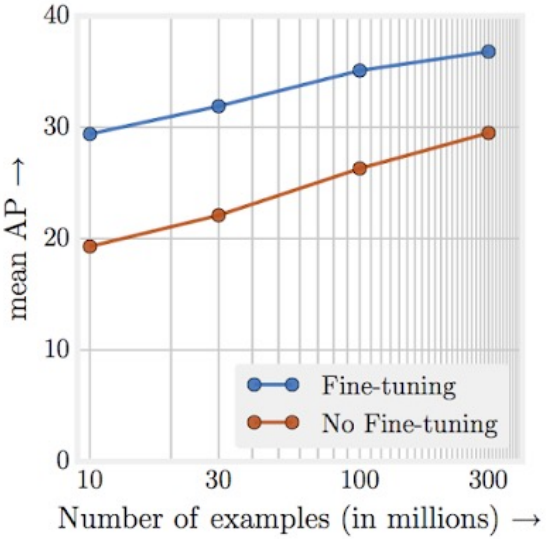
- **Brawn or Brains?**
  - In 2001, Microsoft researchers ran a test to evaluate 4 of different approaches to ML-based language translation
- **Findings:**
  - **Size of the dataset** used to train the model **mattered more** than the model itself
  - As the dataset grew large, performance difference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



Banko, M. and Brill, E. (2001) , "Scaling to Very 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:

- Classical tasks in NLP and Vision are getting commoditized (you take pretrained model and fine tune it), but there are many other unique ML tasks.
- Deep models are often hard to scale and require lots and lots of data. Traditional models allow you to encode prior knowledge better and give you more control.
- Personally, if I am working on a well understood problem I'd use deep learning, but if I am the first person to work on a new problem/classifier I'd use techniques we'll discuss here.

Decision Trees, Random Forests and GBDTs

Preface: Decision Trees

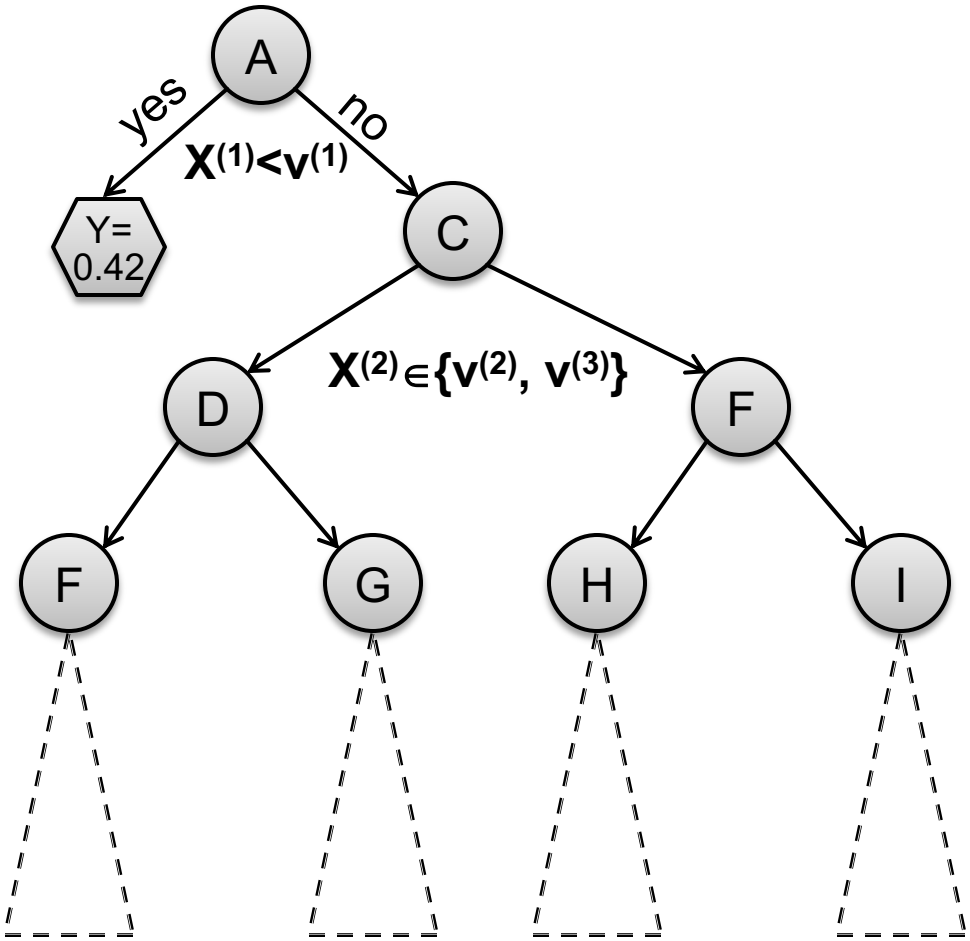
- **Decision trees are part of ML since 1980s**
  - Introduced by Leo Breiman in 1984
  - Notable algorithms: ID3, C4.5
- **More recent innovations include:**
  - Boosted decision trees (gradient boosted DT)
  - Random forest
- Even though DTs are old, hand-engineered and heuristic, they are a method 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 means of some of the other available attribute**
- **Input attributes:**
  - **d** features/attributes: $\boldsymbol{x^{(1)}}, \boldsymbol{x^{(2)}}, \ldots\ \boldsymbol{x^{(d)}}$
  - Each $\boldsymbol{x^{(j)}}$ has **domain** $\boldsymbol{O_j}$
    - **Categorical:** $\boldsymbol{O_j} = \{male, female\}$
    - **Numerical:** $\boldsymbol{H_j} = (1, 200)$
  - $\boldsymbol{Y}$ is output variable with domain $\boldsymbol{O_Y}$:
    - **Categorical:** Classification e.g. ***Y* = eye color**
    - **Numerical:** Regression e.g. ***Y* = lifespan**
- **Data D:**
  - $\boldsymbol{n}$ examples $(\boldsymbol{x_i}, \boldsymbol{y_i})$ where $\boldsymbol{x_i}$ is a $\boldsymbol{d}$-dim feature vector, $\boldsymbol{y_i} \in \boldsymbol{O_Y}$ is output variable
- **Task:**
  - Given an input data vector $\boldsymbol{x}$ predict output label $\boldsymbol{y}$

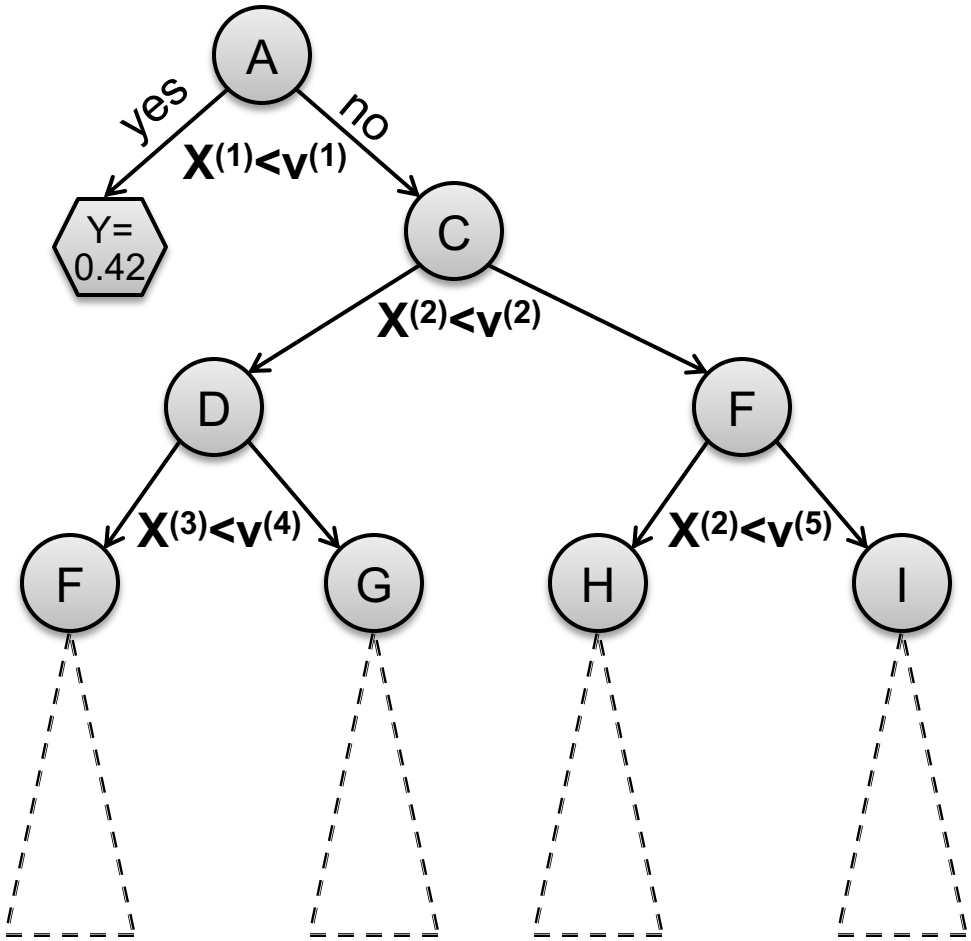
Decision Trees

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



Decision Trees

- **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

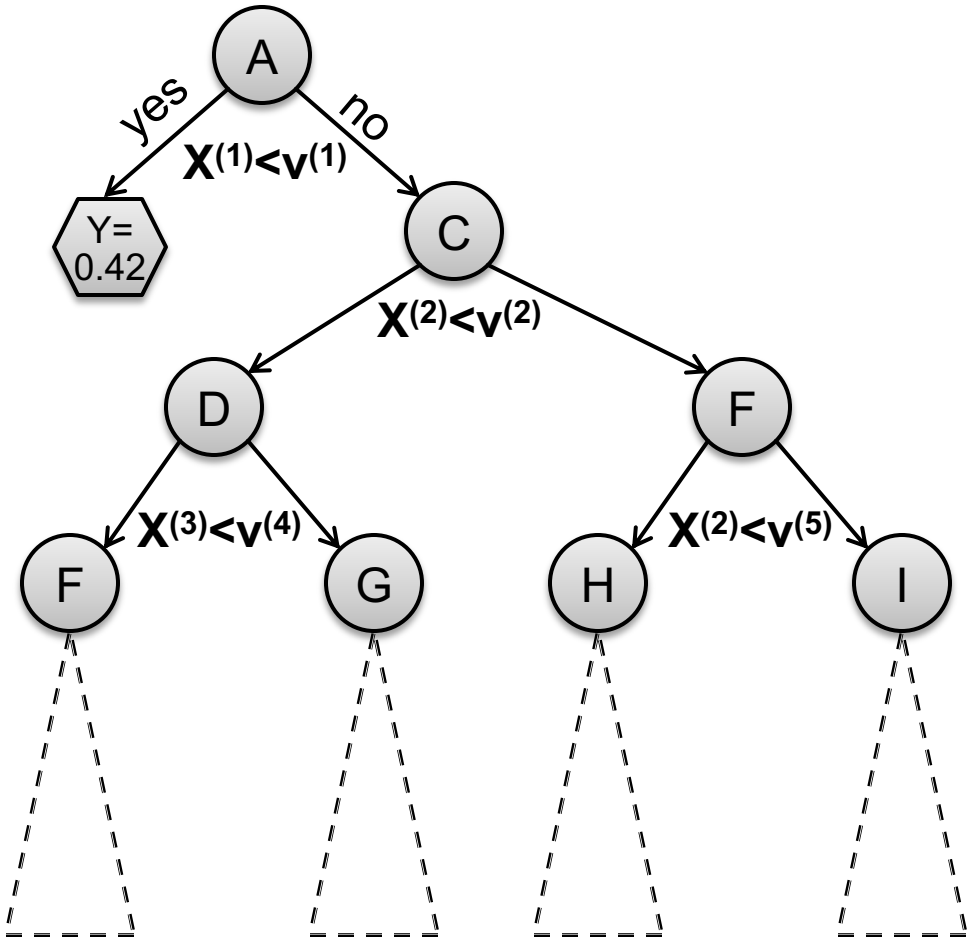


How to make predictions?

- **Input:** Example x_i
- **Output:** Predicted $\hat{y}_i$

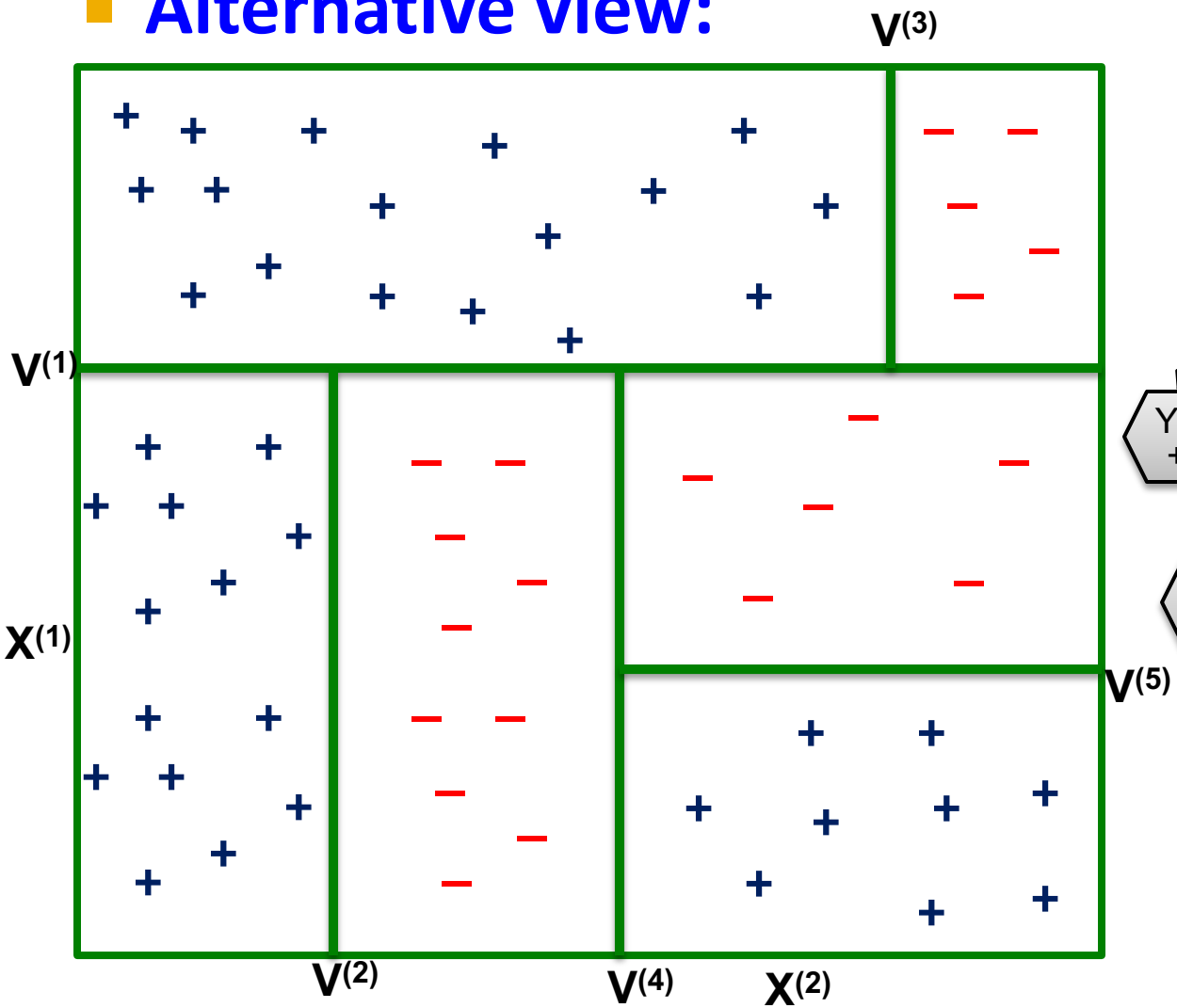
- "Drop" x_i down the tree until it hits a leaf node

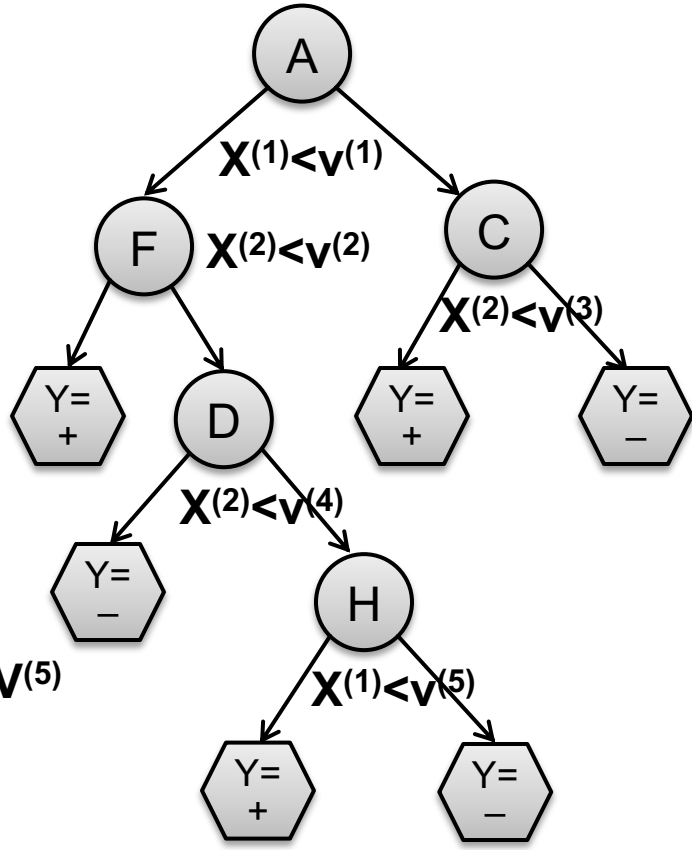
- Predict the value stored in the leaf that x_i hits



Decision Trees: feature space

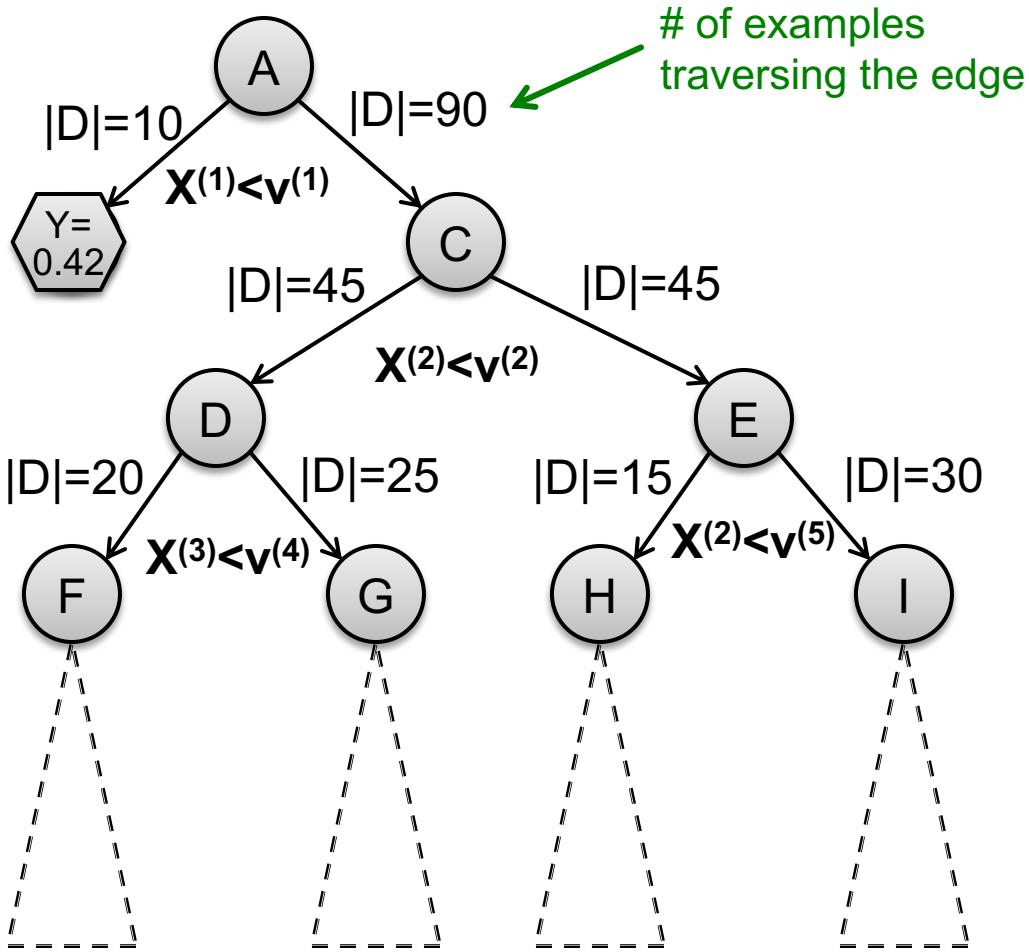
- **Alternative view:**





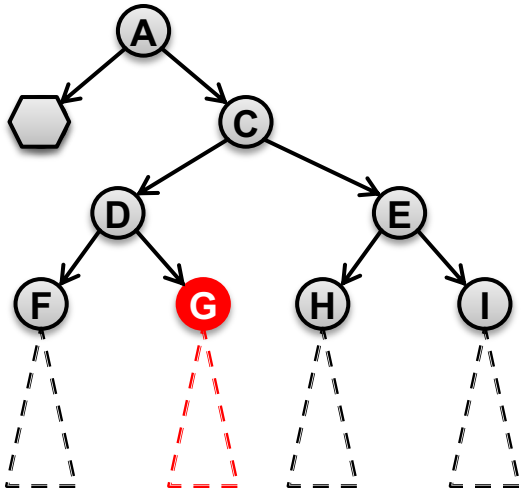
How to construct a tree?

- **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!**

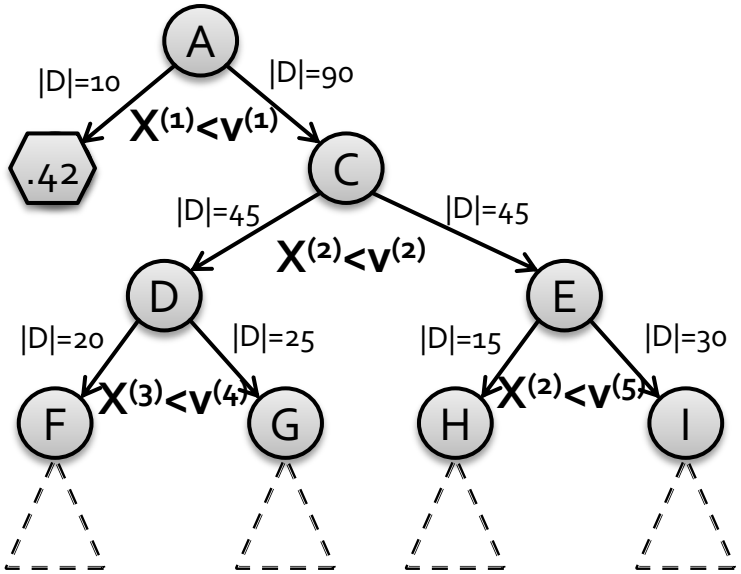
How to construct a tree?

(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**:

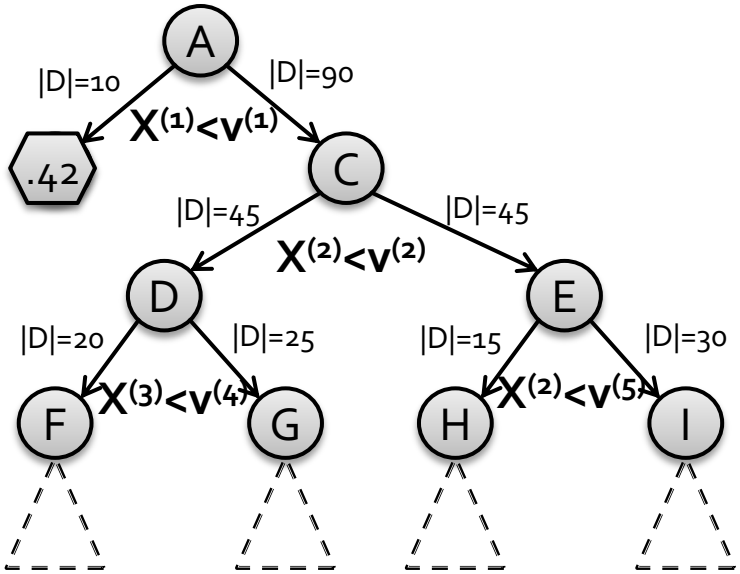
$$|D| \cdot Var(D) - \left(|D_L| \cdot Var(D_L) + |D_R| \cdot Var(D_R)\right)$$

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



How to construct a tree?

(1) How to split? Pick attribute & value that optimizes some criterion



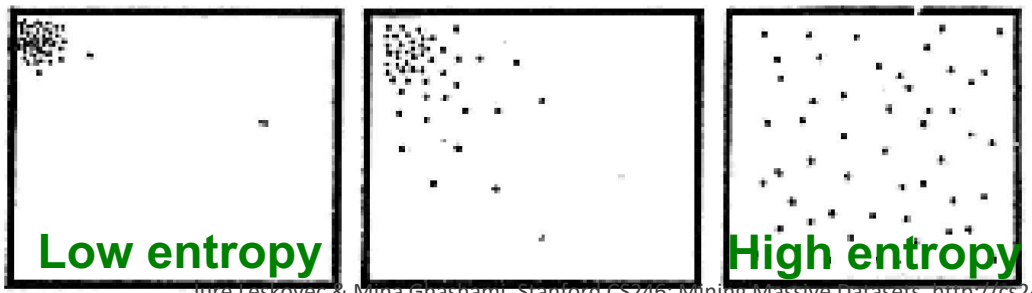
- **Classification:**
  Information Gain
  - **Measures how much a given attribute X tells us about the class Y**
  - $IG(Y \mid X)$: We must transmit Y over a binary link. How many bits on average would it save us if both ends of the line knew X?

Why Information Gain? Entropy

Entropy: What's the smallest possible number of bits, on average, per symbol, needed to transmit a stream of symbols drawn from X's distribution?

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) distrib.
  - A histogram of the frequency distribution of values of X would have many lows and one or two highs



Why Information Gain? Entropy

- **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**
  - $P(Y = Yes) = 0.5$
  - $P(X = Math \,\&\, Y = No) = 0.25$
  - $P(X = Math) = 0.5$
  - $P(Y = Yes \mid X = History) = 0$
- **Note:**
  - $H(Y) = -½\log_2(½) - ½\log_2(½) = \mathbf{1}$
  - $H(X) = 1.5$

Why Information Gain? Entropy

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

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

- **Def: Specific Conditional Entropy**
  - $H(Y \mid X = v)$ = The entropy of Y among only those records in which X has value v
  - **Example:**
    - $H(Y|X = Math) = 1$
    - $H(Y|X = History) = 0$
    - $H(Y|X = CS) = 0$

Why Information Gain?

- **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

- **Def: Conditional Entropy**
  - $H(Y \mid X)$ = The average specific conditional entropy of Y
    - = if you choose a record at random what will be the conditional entropy of Y, conditioned on that row's value of X
    - = Expected number of bits to transmit Y if both sides knew the value of X
    - = $\sum_j P(X = v_j) H(Y|X = v_j)$

Why Information Gain?

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

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

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

  - **Example:**

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

  - **So:** H(Y|X)=0.5*1+0.25*0+0.25*0 = **0.5**

Why Information Gain?

- **Suppose I want to predict Y and I have input X**
  - **Def: Information Gain**
    - $IG(Y|X)$ = I must transmit Y. **How many bits on average would it save me if both ends of the line knew X?**

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

  - **Example:**
    - H(Y) = 1
    - H(Y|X) = 0.5
    - Thus **IG(Y|X) = 1 – 0.5 = 0.5**

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

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 $IG(Y|X)$ is a good split!**

3 steps in constructing a tree

Algorithm 1 **BuildSubtree**

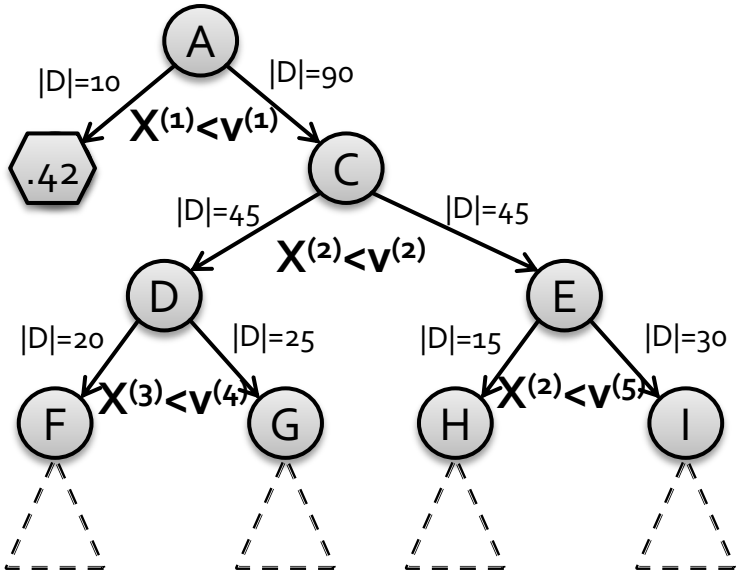
Require: Node n, Data $D \subseteq D^*$

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?

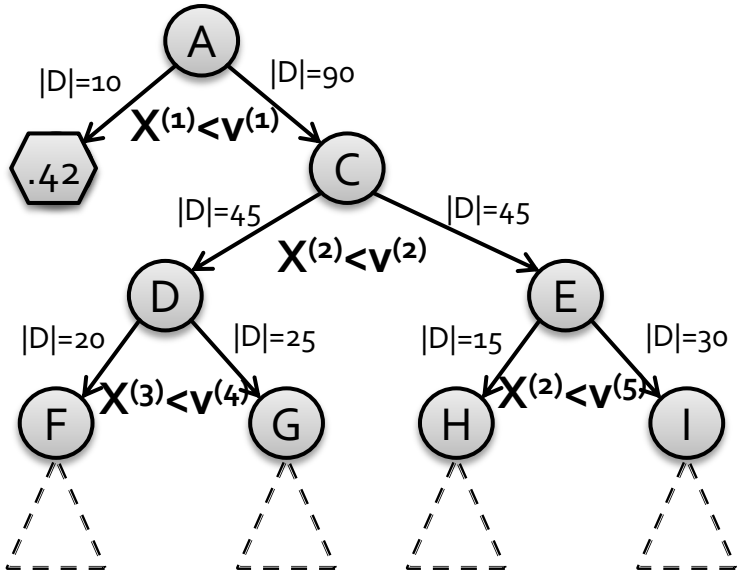
- Many different heuristic options
- **Two ideas:**
  - **(1) When the leaf is "pure"**
    - The target variable does not vary too much: $\boldsymbol{Var(y) < \varepsilon}$
  - **(2) When # of examples in the leaf is too small**
    - For example, $\boldsymbol{|D| \leq 100}$



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



Decision Trees

- **Characteristics**
  - **Classification & Regression**
    - Multiple (~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

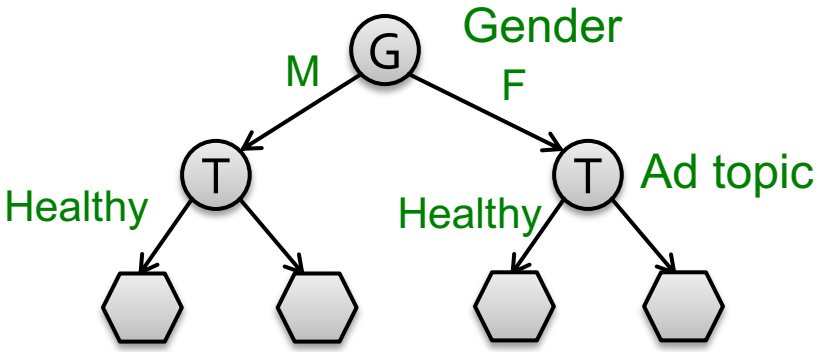
- **Decision trees are the single most popular data mining tool:**
  - Easy to understand
  - Easy to implement
  - Easy to use
  - Computationally cheap
  - It's possible to mitigate overfitting (i.e., with ensemble methods)
  - **They do classification as well as regression!**

Benefit: Feature Transforms

- **Problem:** Many times we want to predict association between a user u and an item x
  - For example, how much will user u like movie x; how likely is u to click on an ad x.
- **Issue:** Many sparse features:
  - User: Demographics, interests, prior activity, …
  - Movie/ad: Keywords, actors, director, …
- **Goal:** Build $f(u, x)$
- **Notice:**
  - Linear model that concatenates features ($w \cdot [u, x]$) is not able to learn that women like healthy food ads.
  - We need to "**cross**" features: $u \times x$
    - Create new feature: (gender, ad topic)
    - E.g., (man, healthy food), (woman, healthy food)
  - **Issue:** Number of features explodes!

Feature Transforms

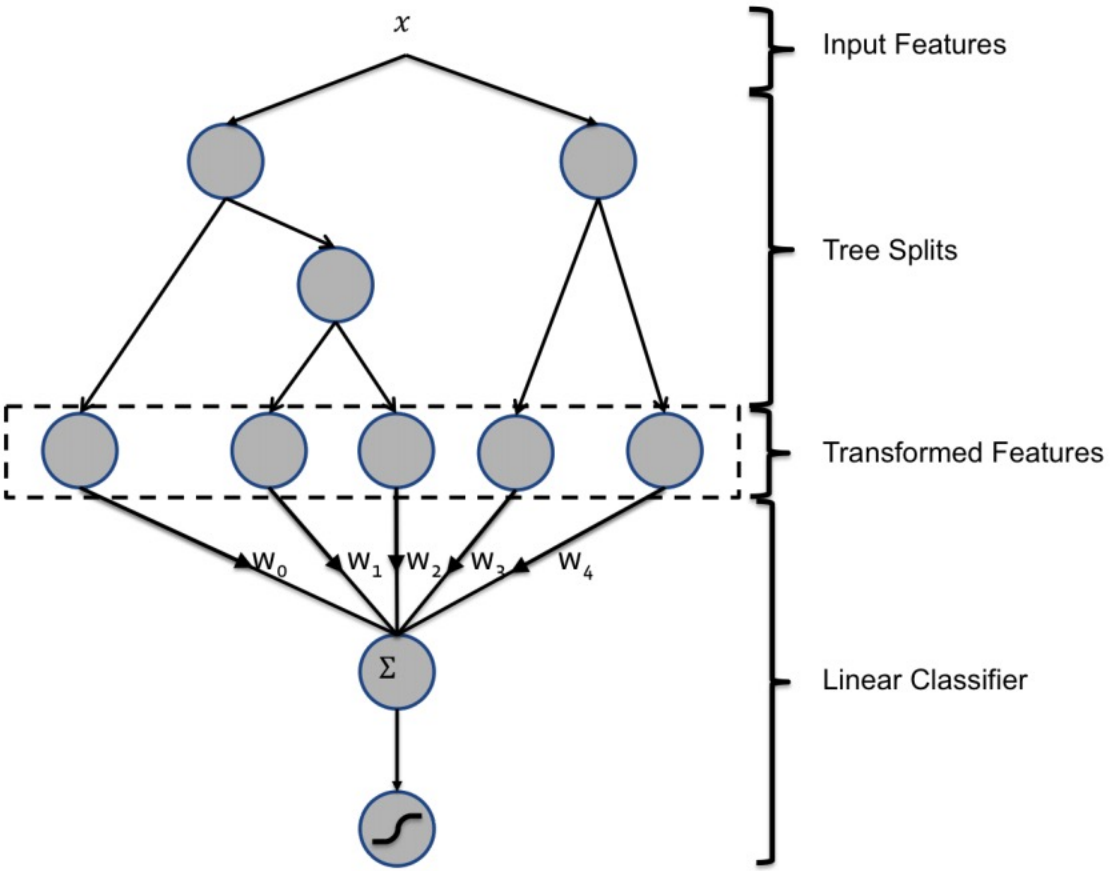
- **Solution:** Build Feature Transforms using decision trees:



- Drop the example into the tree and use 1-hot encoding to denote the leaf it ends at.
- Use these 1-hot vectors as inputs to a linear classifier

Feature Transforms

- **Overall architecture:**

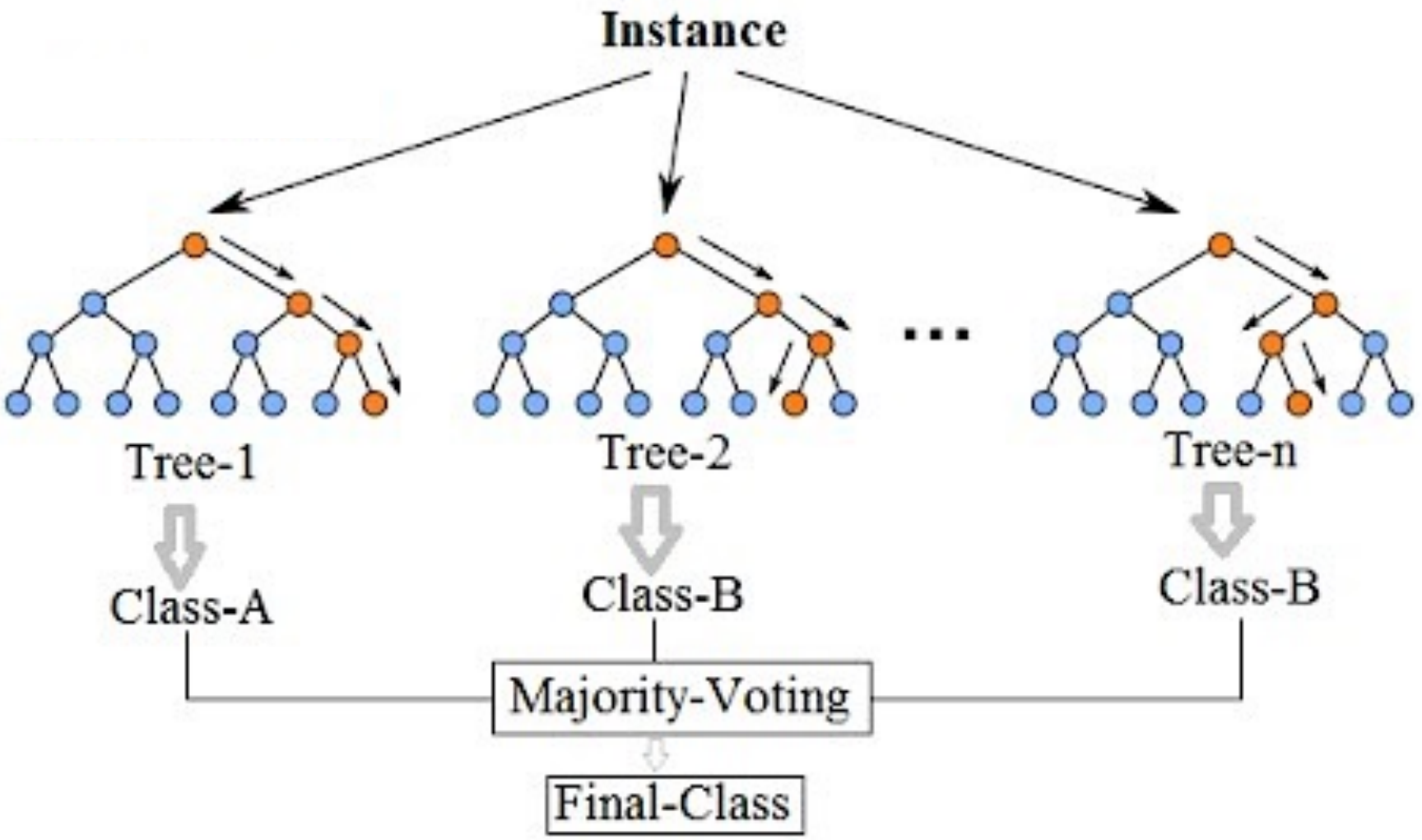


Decision Trees: Learning Ensembles

Learning Ensembles

- **Learn multiple trees and combine their predictions**
  - Gives better performance in practice
- **Bagging:**
  - Learns multiple trees over independent samples of the training data
    - For a dataset **D** on **n** data points: Create dataset **D'** of **n** points but sample from **D** with replacement:
      - 33% points in D' will be duplicates, 66% will be unique
  - Predictions from each tree are averaged to compute the final model prediction

(1): Bagging Decision Trees



(1): Bagged Decision Trees

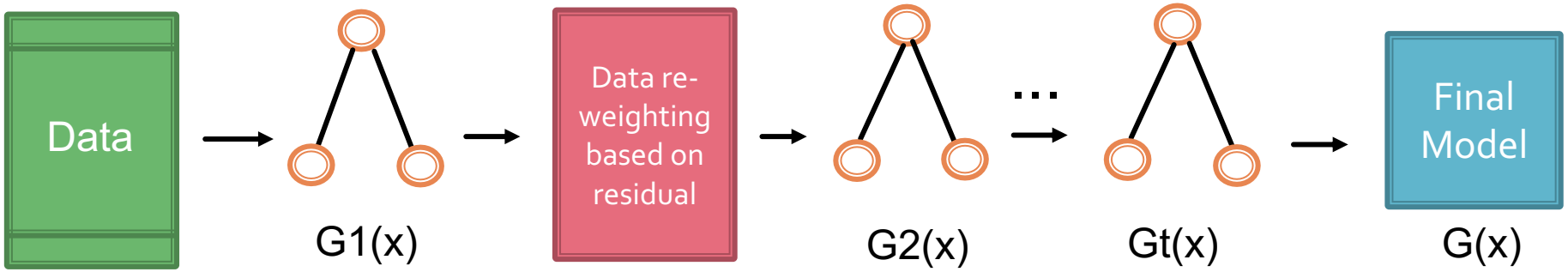
- **How to create random samples of D?**
  - Compute a hash of a training record's id and tree id
  - Use records that hash into a particular range to learn a tree
  - This way the same sample is used for all nodes in a tree
  - **Note:** This is sampling **D** without replacement (but samples of **D** should be created with replacement)

(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
    - If one feature is very strong predictor, then every tree will select it, causing trees to be correlated.
- **Random Forests achieve state-of-the-art results in many classification problems!**

(3): Boosting

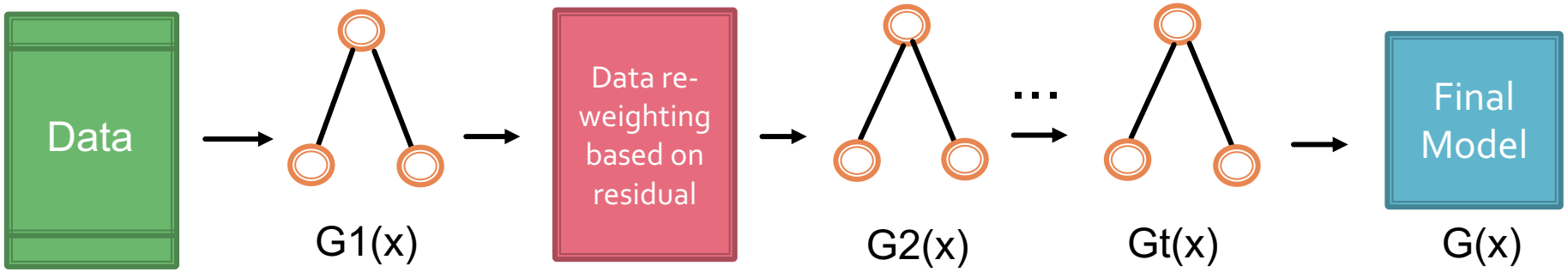
- **Boosting:** Another ensemble learning algorithm
  - Combines the outputs of many "weak" classifiers to produce a powerful "committee"
  - Learns multiple trees sequentially, each trying to improve upon its predecessor
  - Final classifier is weighted sum of the individual classifiers



Residual: Difference between prediction and ground truth

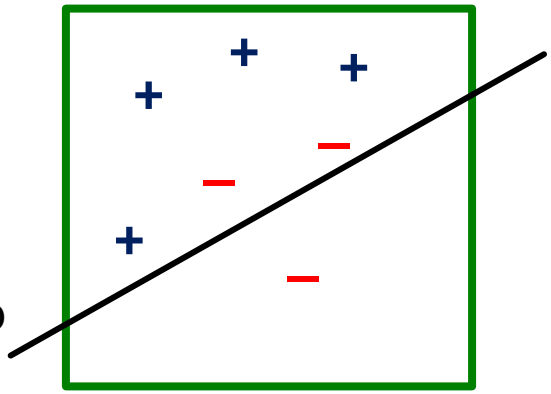
(3): Boosting

- **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



AdaBoost: Weak learner

- **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**



A decision stump

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

Build Decision Trees with AdaBoost

Suppose we have training data $\{(x_i, y_i)\}_{i=1}^{N}$, $y_i \in \{1, -1\}$

- Initialize equal weights for all observations $w_i = 1/N$
- 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

Training One Decision Tree

- **How to split?**
  - Apply weighting to the splitting criterion function and optimize the function to find the best split
  - We'll use information gain as an example
- Recall :
  - Information gain $\mathrm{IG(Y|X)} = \mathrm{H(Y)} - \mathrm{H(Y|X)}$
  - where $$H(X) = -\sum_{i=1}^{N} p(X_i) log\big(P(X_i)\big)$$
- After weighting:

$$H_w(X) = \frac{-\sum_{i=1}^{N} w_i \, p(X_i) log\big(P(X_i)\big)}{\sum_{i=1}^{N} w_i}$$

Update Step

- Calculate the weighted misclassification error

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

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

$$\alpha_t = log\left(\frac{1 - err_t}{err_t}\right)$$

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

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

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

Harder to classify training instances get higher weight

Final Prediction

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

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

- **More accurate trees are weighted higher in the final model**

AdaBoost: Summary

1. Initialize the observation weights $w_i = 1/N,\ i = 1, 2, \ldots, N$.
2. For $m = 1$ to M:
   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.
   (b) Compute (1) Train a stump
   $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$ (2) Compute error
   (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$. (3) Compute tree weight
   (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))],\ i = 1, 2, \ldots, N$. (4) Reweight data
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: Additive training:**
  - Start from constant prediction, **add a new decision tree f_i each time**:
    - Here f_i can be multi-level!

$$
\begin{aligned}
\hat{y}_i^{(0)} &= 0 \\
\hat{y}_i^{(1)} &= f_1(x_i) = \hat{y}_i^{(0)} + f_1(x_i) \\
\hat{y}_i^{(2)} &= f_1(x_i) + f_2(x_i) = \hat{y}_i^{(1)} + f_2(x_i) \\
&\dots \\
\hat{y}_i^{(t)} &= \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{(t-1)} + f_t(x_i)
\end{aligned}
$$

Prediction at training round t

Keep predictions from previous rounds

New model

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 $l()$:**

$$\sum_i l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t)$$

  - 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?

- $Obj = \sum_i l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t)$
- **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:**

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

We can ignore this part, since we are optimizing over f_t

  - **where:**

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

How to decide which f to add?

- **Example: The square loss (L2 loss)**
  - **The approximate objective:**

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

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

  - **Derive g and h for square loss:**

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

Our New Goal

- **Our new goal: Find tree f_t that:**

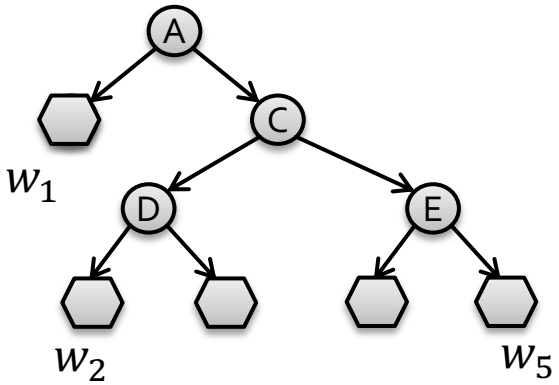
$$\sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t)$$

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

- **Why spending s much efforts to derive the objective, why not just grow trees …**
  - **Theoretical benefit:** Know what we are learning
  - **Engineering benefit:**
    - g and h comes 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)

Define a Tree

- **Every leaf j have a weight w_j**
  - **We will predict w_j for any data belongs to leaf j**



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

$q(x)$ indicate the leaf node that data point x belongs to

- **Define complexity of tree f as:**

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

T… number of leaves of tree f

γ… cost adding a leaf to the tree f

Revisiting the Objective

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

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

q(x) denotes the leaf that data point *x* belongs to

- **Reorder the objective from slide 55 by leaf:**

$$\begin{aligned}
&\simeq \sum_{i=1}^{n} \left[g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i) \right] + \Omega(f_t) \\
&= \sum_{i=1}^{n} \left[g_i w_{q(x_i)} + \frac{1}{2} h_i w_{q(x_i)}^2 \right] + \gamma T + \lambda \frac{1}{2} \sum_{j=1}^{T} w_j^2 \\
&= \sum_{j=1}^{T} \left[(\sum_{i \in I_j} g_i) w_j + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) w_j^2 \right] + \gamma T
\end{aligned}$$

Associated with leaf node *j*

  - Notice this is a sum of T quadratic functions, each function is associated with a leaf node *j*

Finding the Optimal w_j^*

- **Two facts about single var. quadratic function:**

$$argmin_x \; Gx + \tfrac{1}{2}Hx^2 = -\tfrac{G}{H}, \; H > 0 \qquad \min_x \; Gx + \tfrac{1}{2}Hx^2 = -\tfrac{1}{2}\tfrac{G^2}{H}$$

- **We can use the facts to derive optimal w_j:**

Let us define $G_j = \sum_{i \in I_j} g_i \quad H_j = \sum_{i \in I_j} h_i$

$$\begin{aligned} Obj^{(t)} &= \sum_{j=1}^{T} \left[(\sum_{i \in I_j} g_i) w_j + \tfrac{1}{2} (\sum_{i \in I_j} h_i + \lambda) w_j^2 \right] + \gamma T \\ &= \sum_{j=1}^{T} \left[G_j w_j + \tfrac{1}{2} (H_j + \lambda) w_j^2 \right] + \gamma T \end{aligned}$$

- **So, for a fixed tree the optimal w_j and Obj are:**

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

How to find a single tree f_t

Given a tree f_t, we know how to

- Calculate the score for f:

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

- 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:

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

Score of left child | Score of right child | Score if we do not split

  - 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 have negative gain
  - But maybe a split can benefit future splits..
- **Post-Prunning:**
  - 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 user M ensemble of trees**

XGBoost

- XGBoost: e**X**treme **G**radient **B**oosting
  - 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