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

Major 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

Given some data:

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

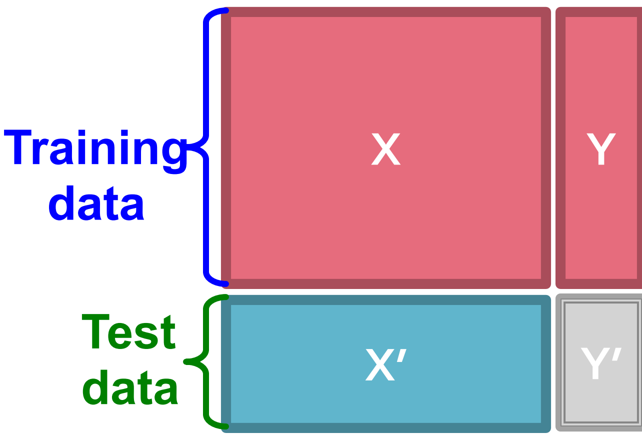
- **Given:**
  Training examples $\left(\boldsymbol{x_i}, \boldsymbol{y_i} = \boldsymbol{f}(\boldsymbol{x_i})\right)$ for some unknown function $\boldsymbol{f}$
- **Find:**
  A good approximation to $\boldsymbol{f}$

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 label, or a real number

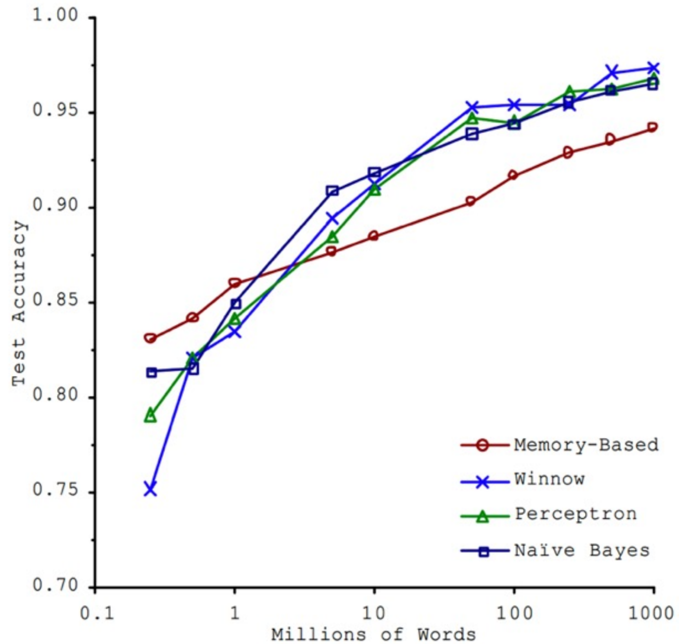
Supervised Learning

- **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 unable to predict Y'**
  - **We want to build a model that <u>generalizes</u> well to unseen data**



Why Large-Scale ML?

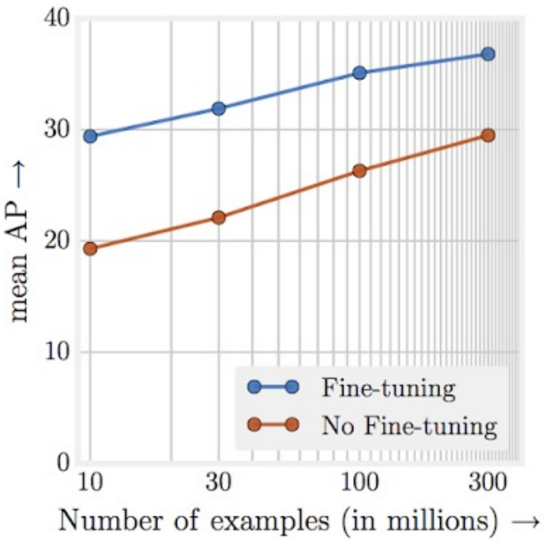
- **Brawn or Brains?**
  - In 2001, Microsoft researchers ran a test to evaluate 4 of different approaches to ML-based language translation
- **Findings:**
  - **Size of the dataset** used to train the model **mattered more** than the model itself
  - As the dataset grew large, performance difference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

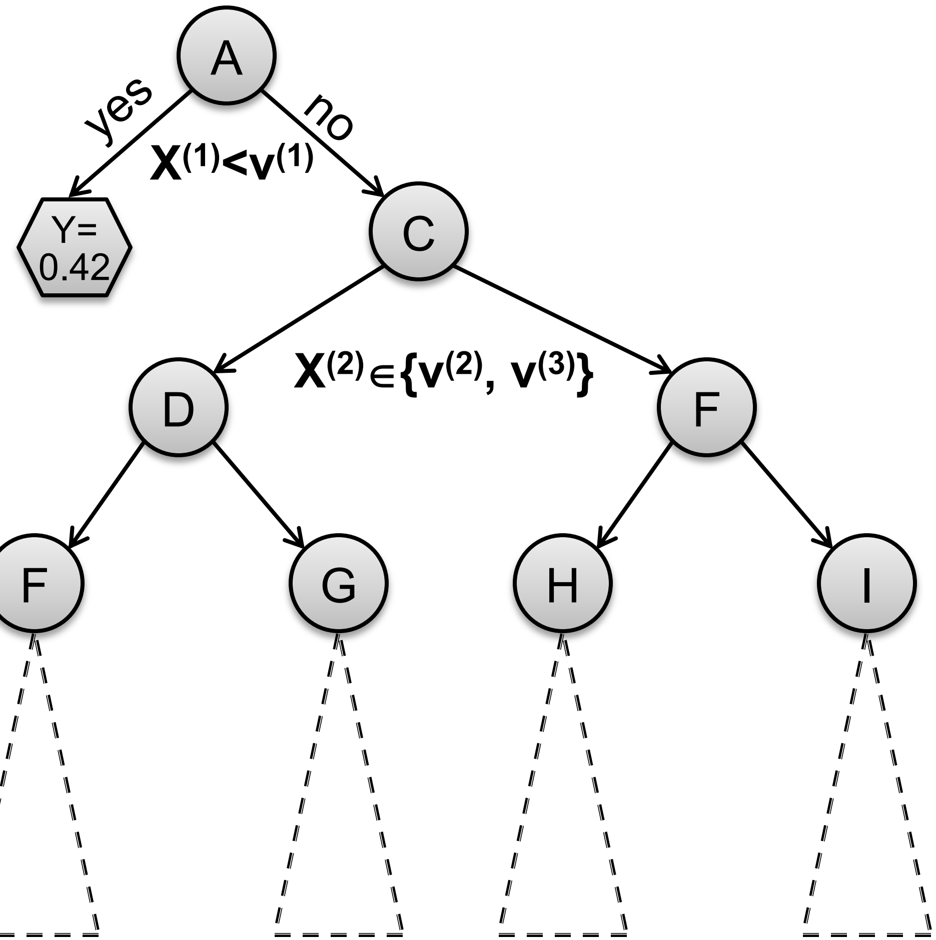
- Decision trees are part of ML since 1980s
  - Introduced by Leo Breiman in 1984
  - Notable algorithms: ID3, C4.5
- **More recent innovations include:**
  - Boosted decision trees (gradient boosted DT)
  - Random forest
- Even though DTs are old, hand-engineered and heuristic, they are a method of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

Decision Tree Learning

- **Given one target 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 $\boldsymbol{y}$

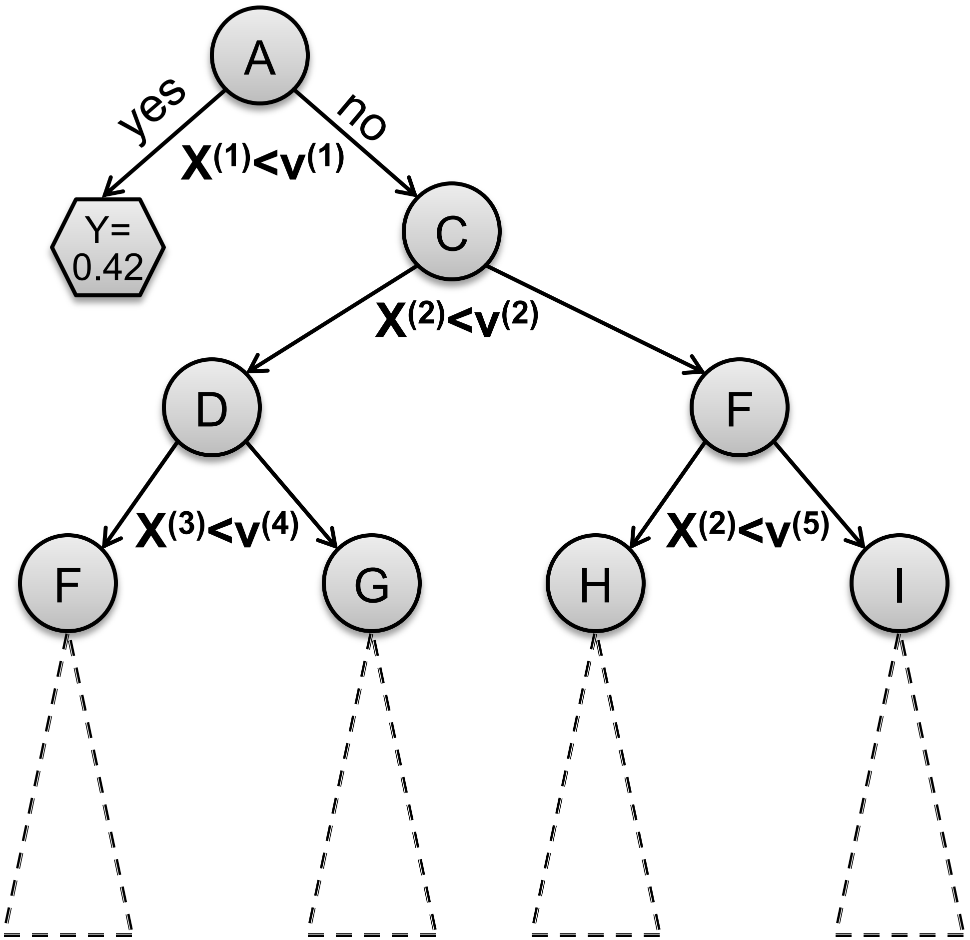
Decision Trees

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



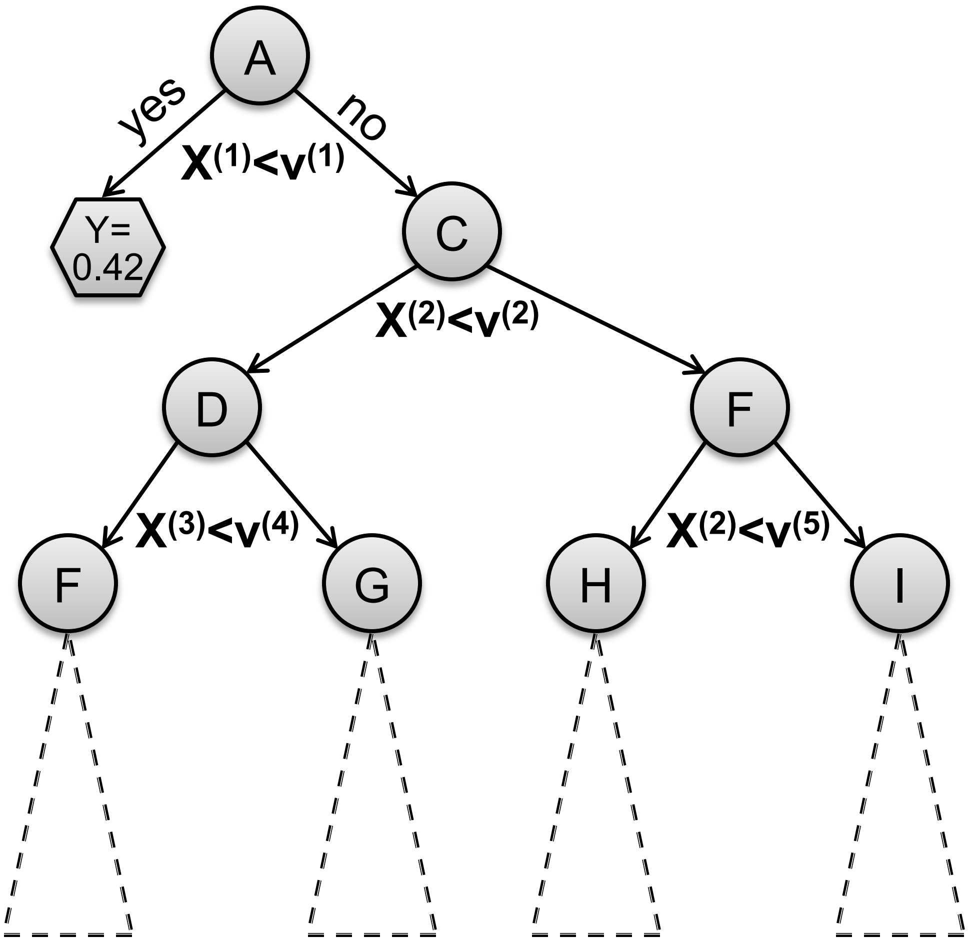
Decision Trees

- **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



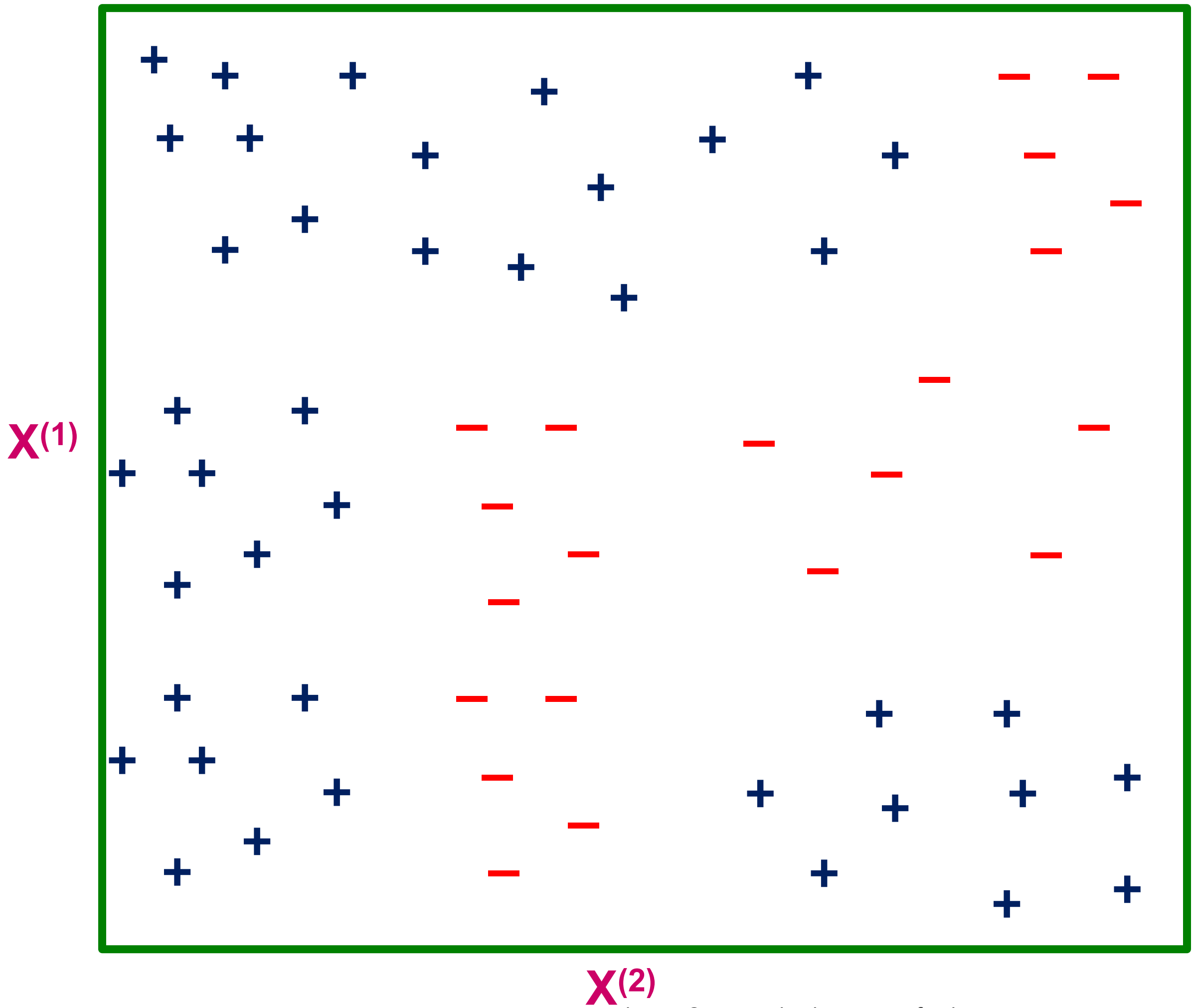
How to make predictions?

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



Decision Trees: feature space

- **Alternative view:**

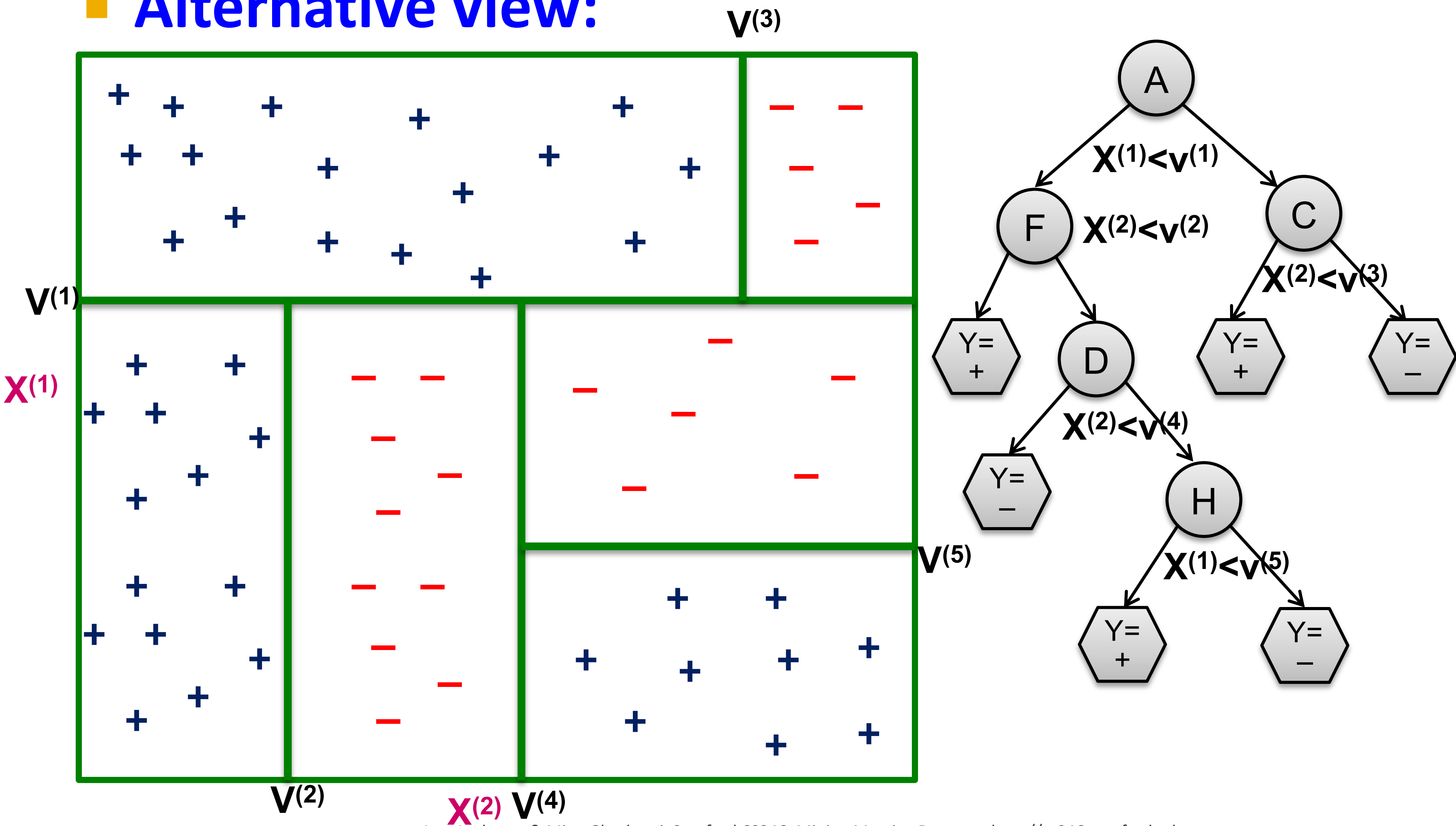


Data is 2-dim: $x = x^{(1)}, x^{(2)}$

Class label: $y = \{+, -\}$

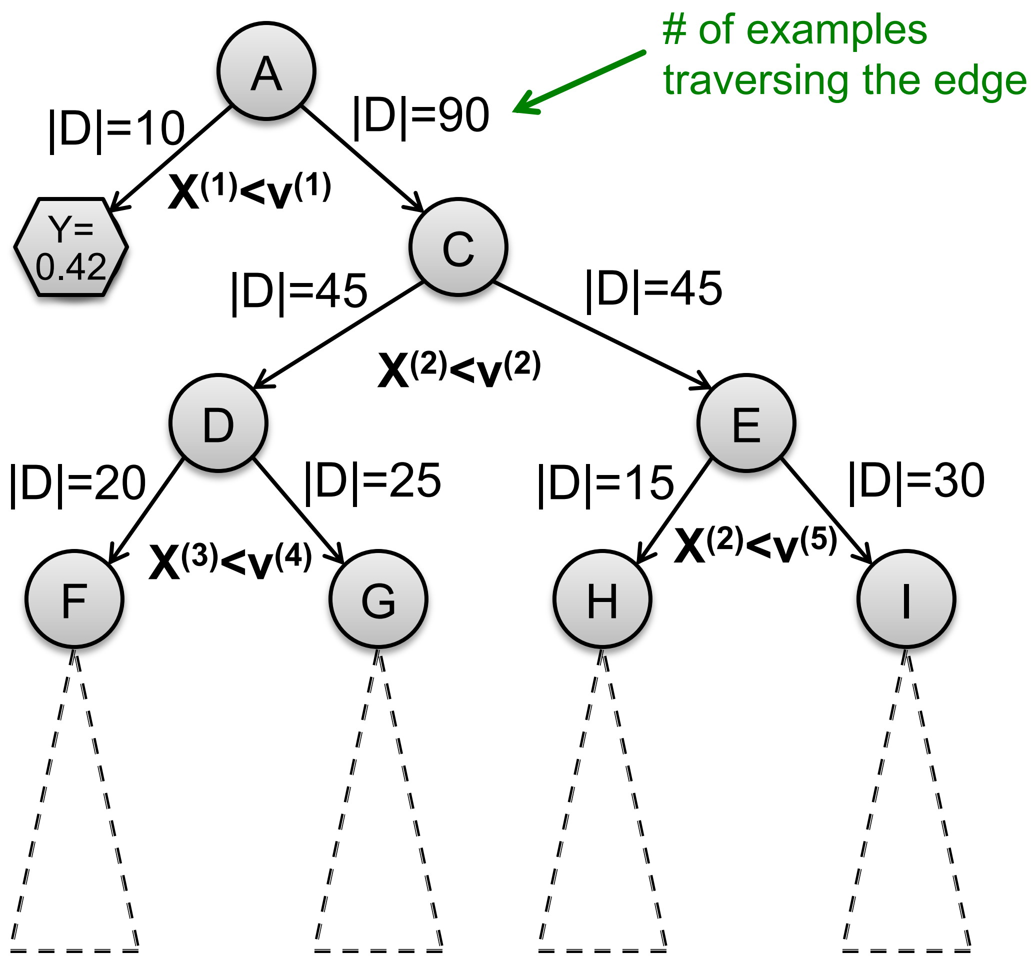
Decision Trees: feature space

- **Alternative view:**

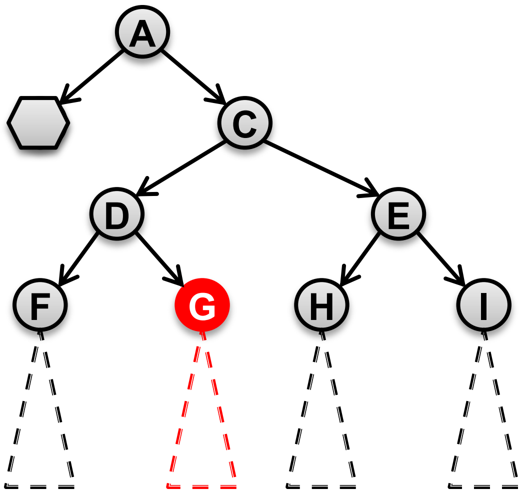


How to construct a tree?

- **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 $D \subseteq D^*$

$(n \rightarrow \text{split}, D_L, D_R) = \text{FindBestSplit}(D)$ **(1)**

if $\text{StoppingCriteria}(D_L)$ **then** **(2)**

  $n \rightarrow \text{left\_prediction} = \text{FindPrediction}(D_L)$ **(3)**

else

  BuildSubtree $(n \rightarrow \text{left}, D_L)$

if $\text{StoppingCriteria}(D_R)$ **then**

  $n \rightarrow \text{right\_prediction} = \text{FindPrediction}(D_R)$

else

  BuildSubtree $(n \rightarrow \text{right}, D_R)$

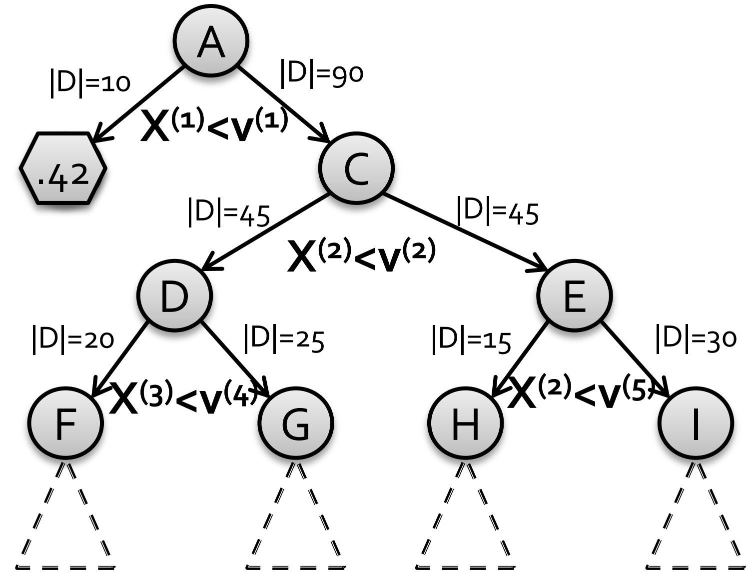
- **Requires at least a single pass over the data!**

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**:

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

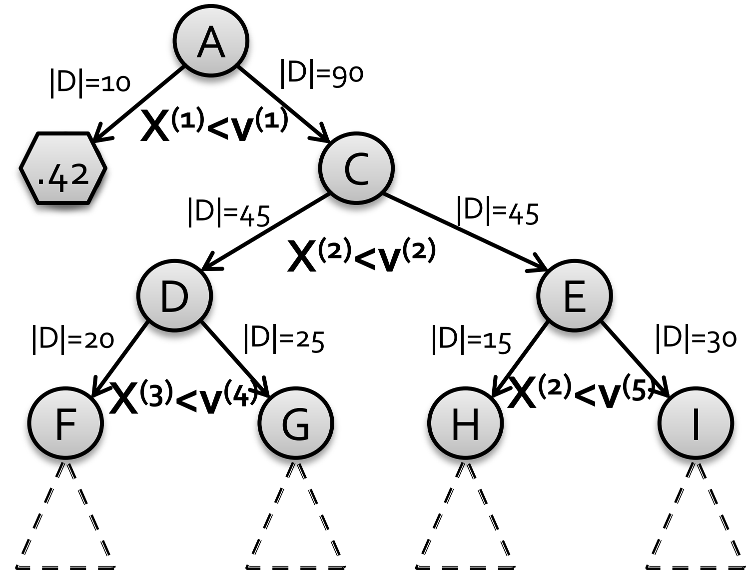
      - $Var(D) = \frac{1}{|D|} \sum_{i \in D} (y_i - \bar{y})^2$ … variance of y_i in D



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?

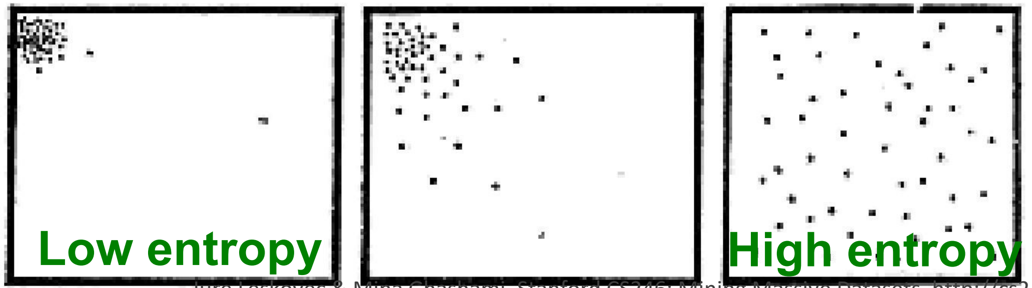


$$IG(Y|X) = H(Y) - H(Y \mid X)$$

Why Information Gain? Entropy

Entropy:
Consider random variable $X = \{X_1, \dots, X_m\}$
What's the smallest possible number of bits, on average, per symbol, that we need 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**
  - $P(Y = Yes) = 0.5$
  - $H(Y) = -½\log_2(½) - ½\log_2(½) = \mathbf{1}$
  - $P(X = CS) = 0.25$
  - $P(X = History) = 0.25$
  - $P(X = Math) = 0.5$
  - $H(X) = \sum_{j=1}^{m} p(X_j)\, log\, p(X_j) = 1.5$

Why Information Gain? Entropy

- **Suppose I want to predict $\boldsymbol{Y}$ and I have input $\boldsymbol{X}$**
  - $\boldsymbol{X}$ = College Major
  - $\boldsymbol{Y}$ = Likes “Casablanca”

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

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

Why Information Gain?

- Suppose I want to predict $\boldsymbol{Y}$ and I have input $\boldsymbol{X}$
  - $\boldsymbol{X}$ = College Major
  - $\boldsymbol{Y}$ = Likes "Casablanca"

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

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

Why Information Gain?

- **Suppose I want to predict Y and I have input X**

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

- $H(Y \mid X)$ $=$ The average specific conditional entropy of Y

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

- **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 $\boldsymbol{Y}$ and I have input $\boldsymbol{X}$**

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

- **Def: Information Gain**
  - $\boldsymbol{IG(Y|X)}$ = I must transmit $\boldsymbol{Y}$. **How many bits on average would it save me if both ends of the line knew X?**

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

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

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 $\boxed{\textbf{BuildSubtree}}$

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

$(n \rightarrow \text{split}, D_L, D_R) = \boxed{\text{FindBestSplit}(D)}$ **(1)**

if $\boxed{\text{StoppingCriteria}(D_L)}$ **then** **(2)**

  $n \rightarrow \text{left\_prediction} = \boxed{\text{FindPrediction}(D_L)}$ **(3)**

else

  BuildSubtree $(n \rightarrow \text{left}, D_L)$

if StoppingCriteria(D_R) **then**

  $n \rightarrow \text{right\_prediction} = \text{FindPrediction}(D_R)$

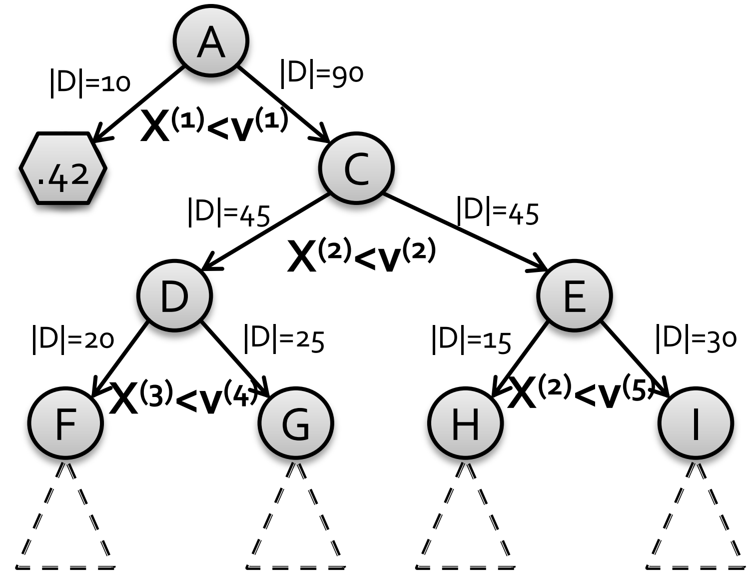
else

  BuildSubtree $(n \rightarrow \text{right}, D_R)$

When to stop?

(2) When to stop?

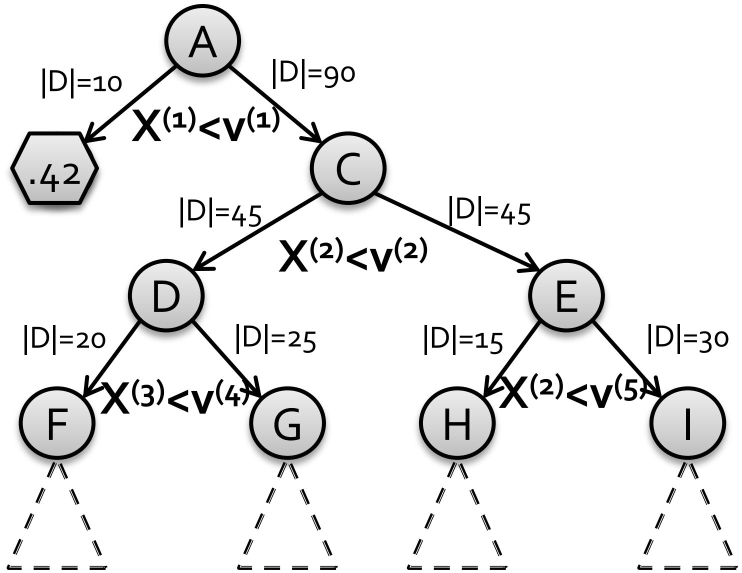
- Many different heuristic options
- **Two ideas:**
  - **(1) When the leaf is "pure"**
    - The target variable does not vary too much: $\boldsymbol{Var(y)} < \varepsilon$
  - **(2) When # of examples in the leaf is too small**
    - For example, $|\boldsymbol{D}| \leq \mathbf{100}$



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!**

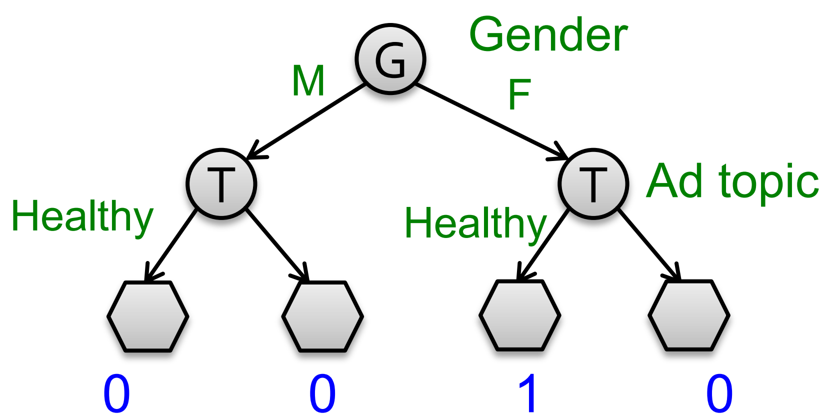
Benefit: Feature Transforms

- **Problem:** Many times we want to predict association between a user u and an item x
  - For example, how likely user u is to interact with item x, e.g. how likely is she/he to click on a specific ad
- **Issue:** Many sparse features:
    - User: Demographics, interests, prior activity, …
    - Ad: Keywords, topic, provider, …
- **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:
  - Decision tree picks the best cross-features

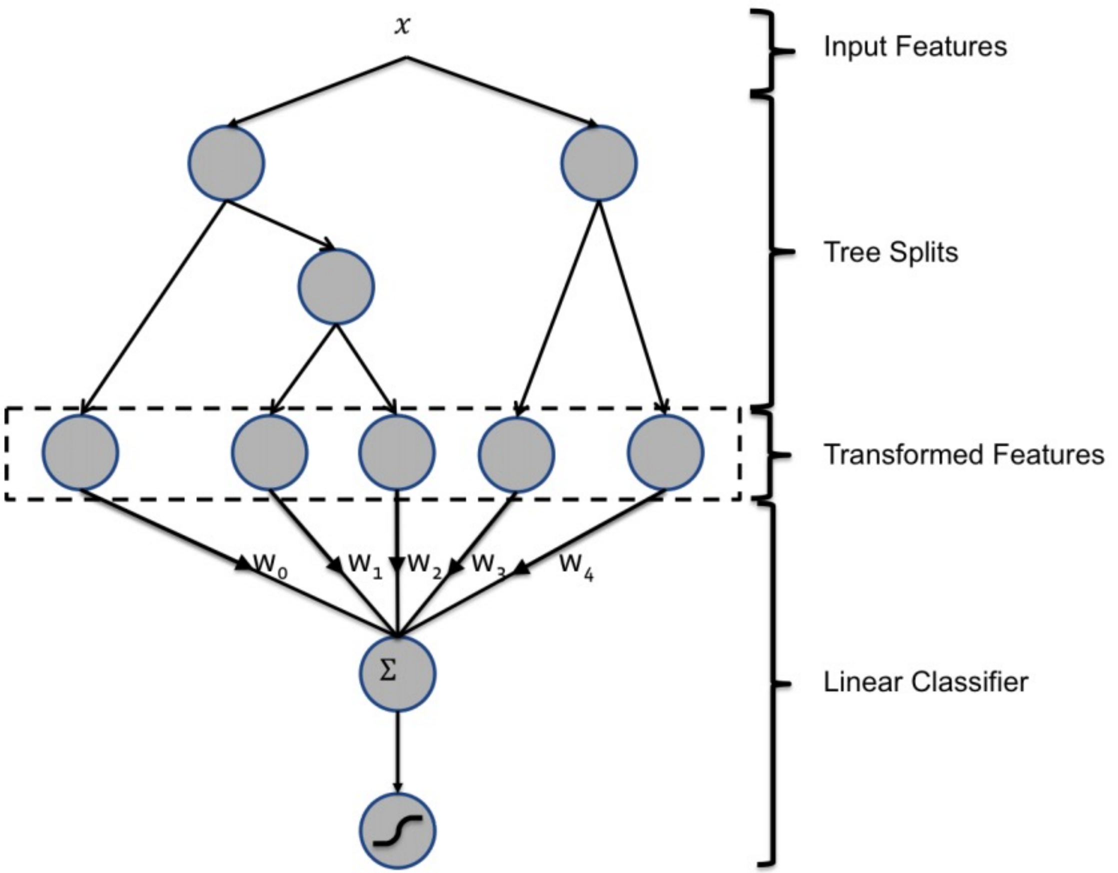
(Female, Healthy) is encoded as:



  - Drop the example into the tree and use 1-hot encoding to denote the leaf it ends at.
  - "gender-adTopic" is a new feature; it takes 4 values
  - Use these 1-hot vectors as inputs to a linear classifier

Feature Transforms

- **Overall architecture:**



Decision Trees: Learning Ensembles

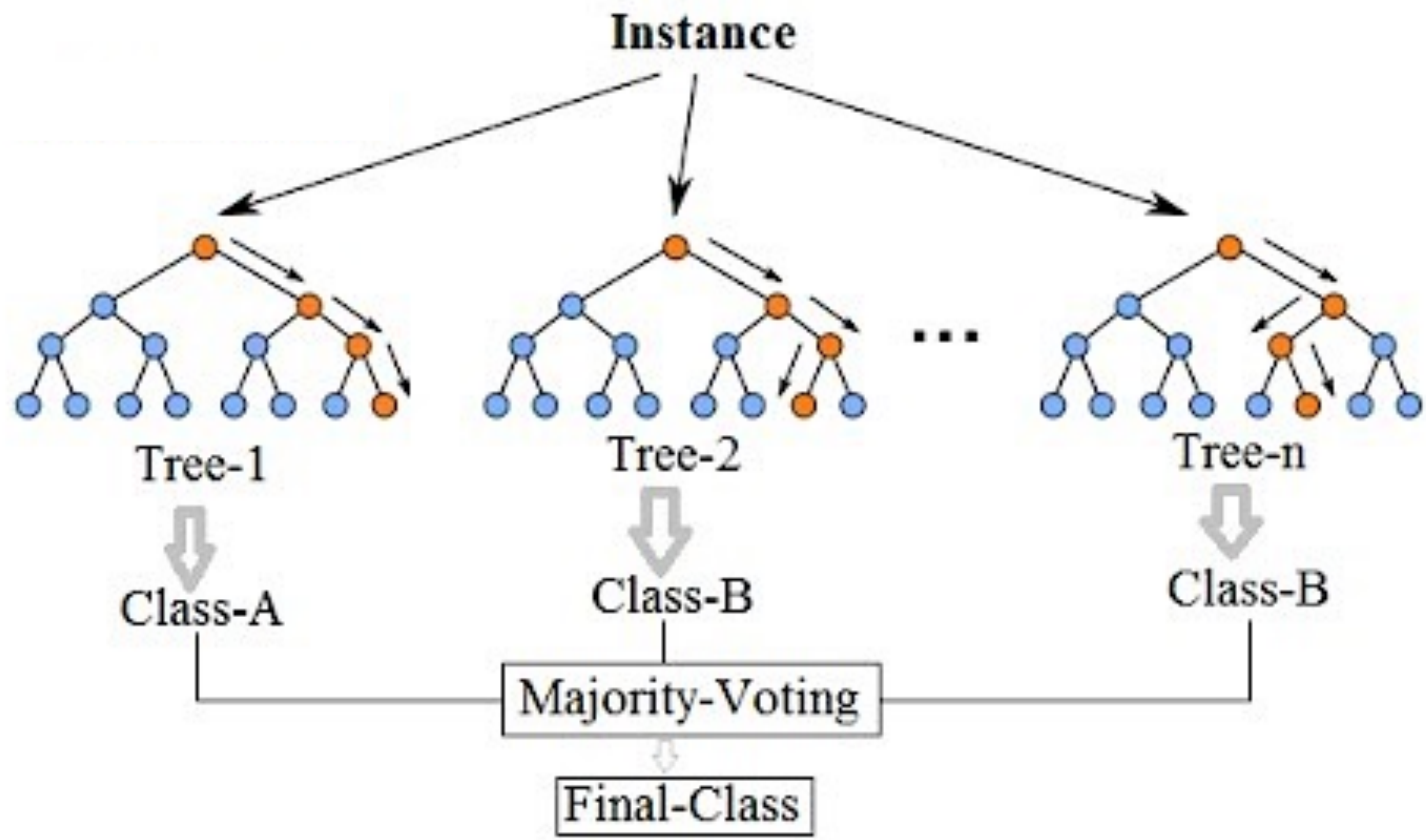
Learning Ensembles

- **Learn multiple trees and combine their predictions**
  - The "wisdom of crowds"
  - A group/ ensemble of base learners that collectively achieve a better final prediction.
  - Decision trees are prone to
    - Overfitting (high variance and low bias) when it hasn't been pruned
    - Underfitting (low variance and high bias) when it's very small, i.e. a decision stump.
  - Ensemble reduces bias or variance, yielding better model performance.

Learning Ensembles

- **There are two main types of ensemble learning:**
  - Bagging and Boosting
- **Bagging (bootstrap aggregation):**
  - Learns multiple trees in parallel over independent samples of the training data
  - **1) Bootstrapping:** Given a dataset, create multiple datasets by sampling data points randomly and with replacement.
  - **2) Parallel training:** Train decision trees on samples independently and in parallel with each other
  - **3) Aggregation:** Depending on the task (i.e. regression or classification), an average or a majority of the predictions are computed for a more accurate estimate.
    - Regression, an average is taken of all the outputs predicted by the individual classifiers; this is known as "soft voting".
    - Classification, the class with the highest majority of votes is accepted; this is "hard voting" or majority voting.

(1): Bagging Decision Trees



(2) Improvement: Random Forests

- So far we did *instance bagging*.
  - **Decision trees are greedy**
  - They choose which variable to split on using a greedy algorithm that minimizes error.
  - Even with Bagging, the decision trees can have a lot of structural similarities and in turn have <u>high correlation</u> in their predictions.

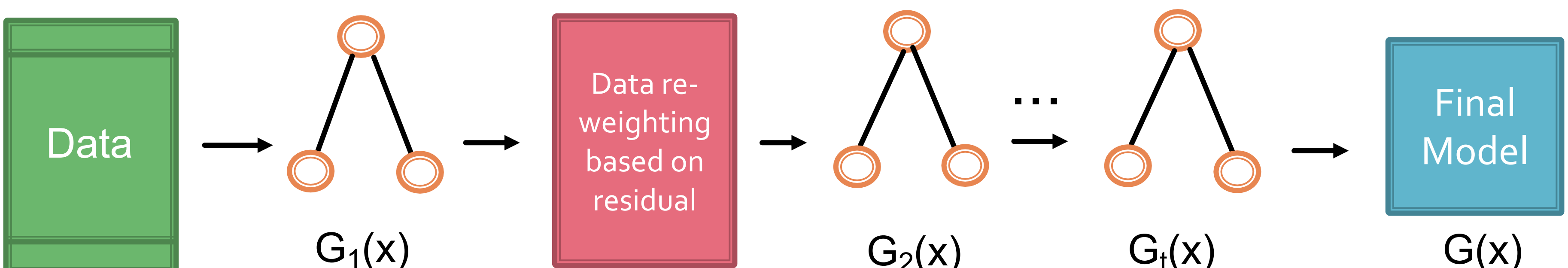
- **Feature Bagging**
  - Pick a random sample of features at each split
  - less correlation among trees
  - **Random forest**

(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

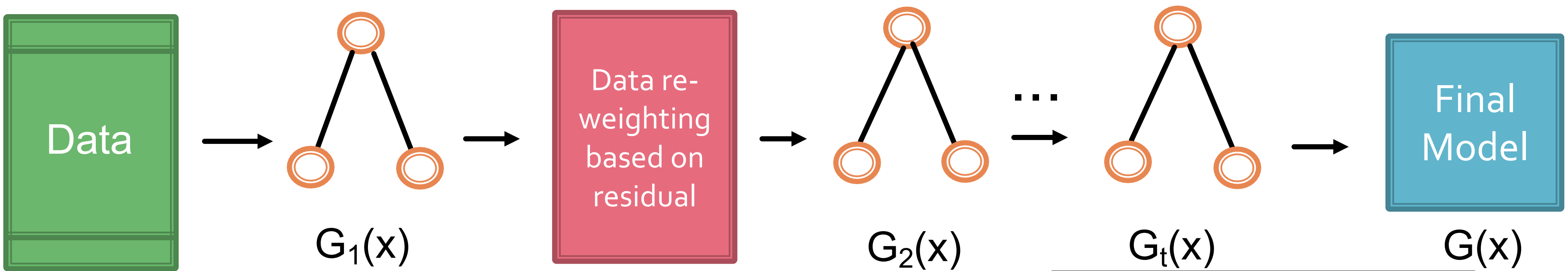


Residual: Difference between prediction and ground truth

$$G(x) = \sum_{i=1}^{t} G_i(x)$$

(3): Boosting

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



x

Residual: difference between prediction and ground truth

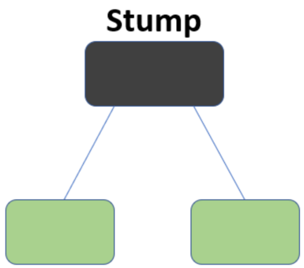
$$G(x) = \sum_{i=1}^{t} G_i(x)$$

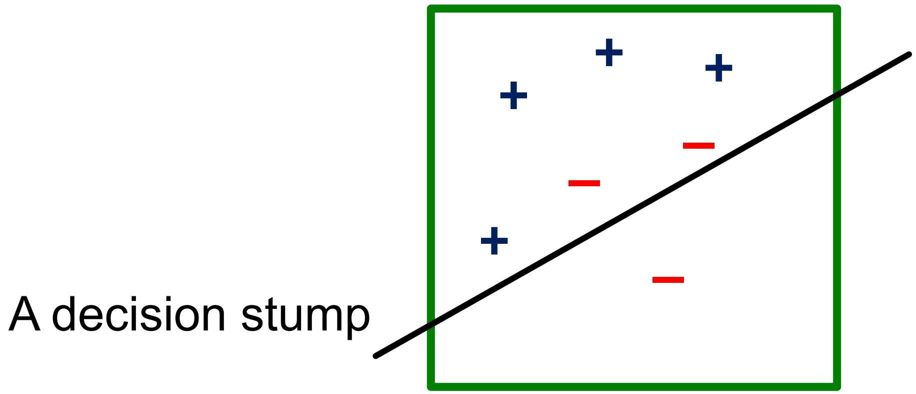
AdaBoost

- AdaBoost = Adaptive Boosting
- It builds many weak learners and ensembles their predictions
- The individual learners can be weak, but as long as the performance of each one is slightly better than random guessing, the final model converge to a strong learner.
- Every weak learner used in AdaBoost is tweaked in favor of instances misclassified by previous weak learners.

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**





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

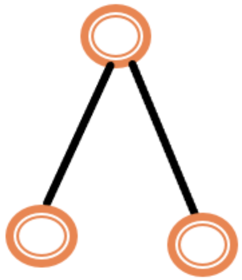
Build Decision Trees with AdaBoost

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 $\mathrm{w_i}$
  2. **Compute the misclassification error** adjusted by $\mathrm{w_i}$
  3. **Compute the weight of the current tree** α_t
  4. **Reweight each observation** based on prediction accuracy

Training One Decision Tree

- **Step1- Train a stump. 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 $IG(Y|X) = H(Y) - H(Y|X)$
  - where

$$H(X) = -\sum_{i=1}^{N} p(X_i) log(P(X_i))$$

- After weighting:

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

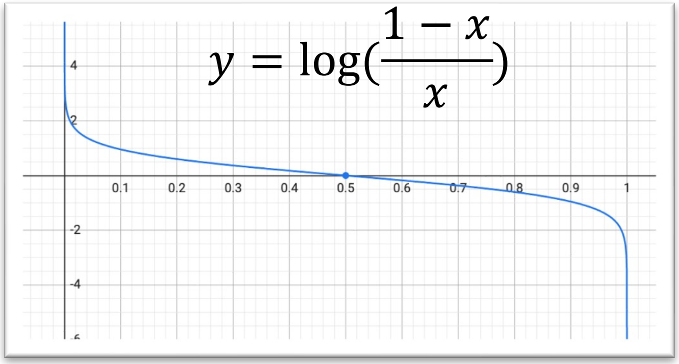
Update Step

- **Step2-** Calculate the weighted misclassification error

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

- **Step3-** 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;

- **Step4-** Use misclassification error and tree weight to reweight the training data:

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

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) $\boxed{\text{Fit a classifier } G_m(x) \text{ to the training data using weights } w_i.}$

   (1) Train a stump

   (b) Compute

   $$\boxed{\mathrm{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}}.$$

   (2) Compute error

   (c) $\boxed{\text{Compute } \alpha_m = \log((1 - \mathrm{err}_m)/\mathrm{err}_m).}$

   (3) Compute tree weight

   (d) $\boxed{\text{Set } w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))],}\ i = 1, 2, \ldots, N.$

   (4) Reweight data
3. Output $G(x) = \mathrm{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

AdaBoost Conclusion

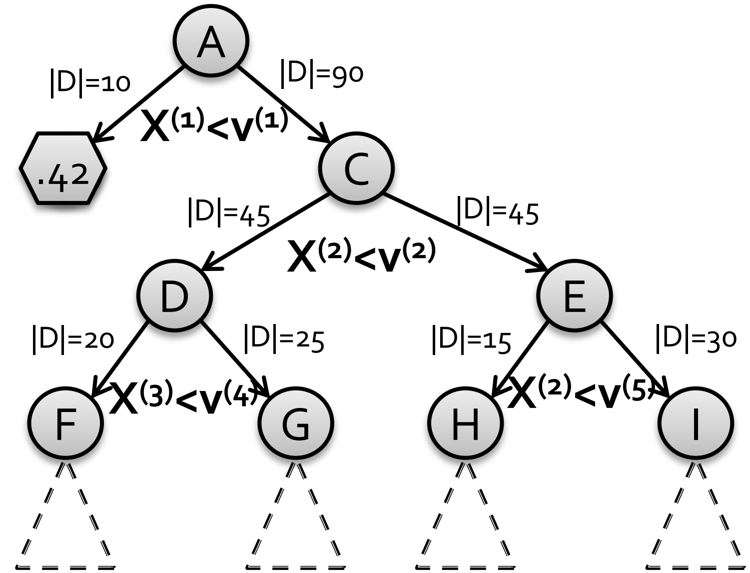
- **Iteratively train weak learners (decision stumps) to form a strong model:**
  - Trees with high accuracy are given more weights in the final model
  - Misclassified data get higher weights in the next iteration

- 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!

Large Scale Machine Learning: Decision Trees (2)

CS246: Mining Massive Datasets
Jure Leskovec, Stanford University
Mina Ghashami, Amazon
http://cs246.stanford.edu

Decision Tree



(1) How to construct?

- **Regression**
  - Purity
- **Classification**
  - Information Gain : IG(Y|X)

(2) When to stop?

- **When the leaf is "pure"**
- **When # examples in the leaf is too small**

(3) How to predict?

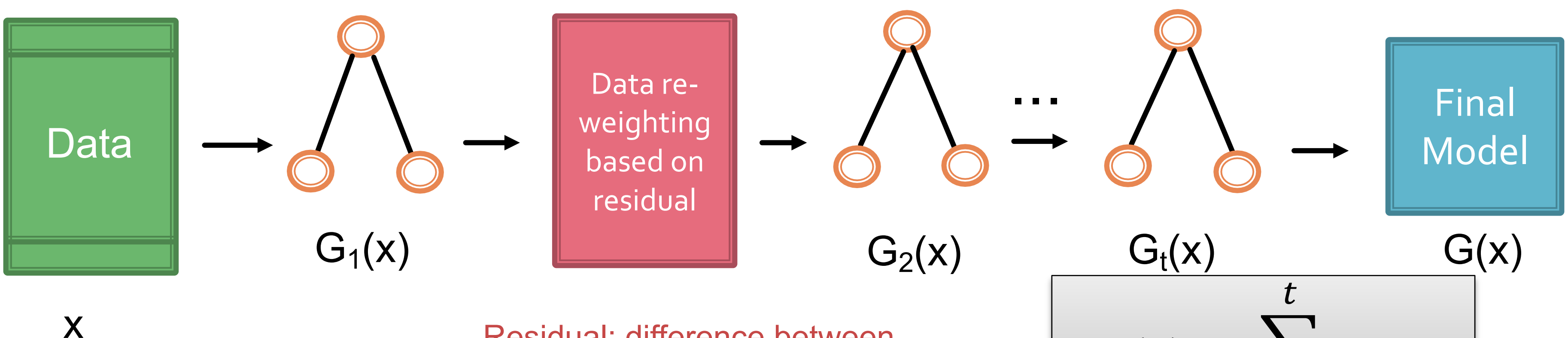
- **Regression:**
  - Predict average y_i of the examples in the leaf,
- **Classification:**
  - Predict most common y_i of the examples in the leaf

Learning Ensembles

- **Learn multiple trees, combine their predictions**
  - Decision trees are prone to overfitting
- **Two Ensemble approaches:**
  - Bagging (bootstrap aggregation)
    - Train multiple trees in parallel
    - Instance bagging:
      - sample dataset with replacement, train a tree on each sample set
    - Feature bagging => random forest
      - Sample a subset of features at each split point
  - Boosting
    - Train multiple trees sequentially

Boosting

- **Two boosting algorithms:**
  - **AdaBoost**
    - Where each $G_t(x)$ is a one–level decision tree
  - **Gradient Boosted Decision Trees (GBDT)**
    - Where each $G_t(x)$ is a multi–level decision tree



Residual: difference between prediction and ground truth

$$G(x) = \sum_{i=1}^{t} G_i(x)$$

AdaBoost

Equal weight to all data points

1. Initialize the observation weights $\boxed{w_i = 1/N,\ i = 1, 2, \ldots, N.}$

2. For $m = 1$ to M:

   (a) $\boxed{\text{Fit a classifier } G_m(x) \text{ to the training data using weights } w_i.}$

   (1) Train a stump

   (b) Compute

   $$\boxed{\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}}.$$

   (2) Compute error

   (c) $\boxed{\text{Compute } \alpha_m = \log((1 - \text{err}_m)/\text{err}_m).}$

   (3) Compute tree weight

   (d) $\boxed{\text{Set } w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))],}$ $i = 1, 2, \ldots, N.$

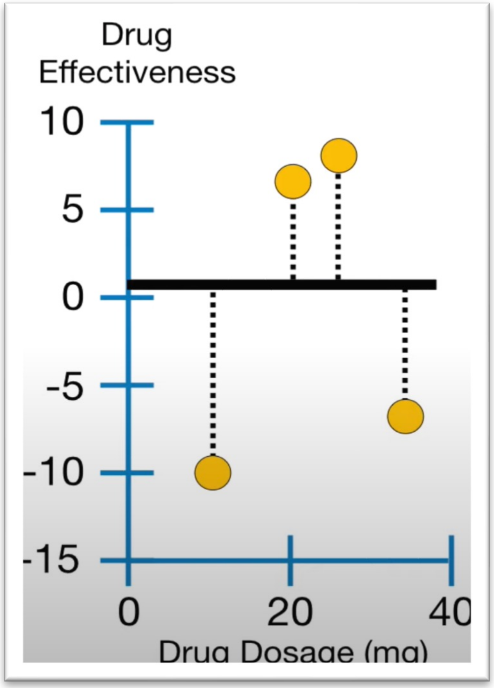
   (4) Reweight data

3. Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

Gradient Boosted Decision Trees

Gradient Boosted Decision Trees

- **Idea: Additive training**
  - Start with a constant prediction, **add a new decision tree each time**. It can be multi-level!
  - Let's see it in an example for regression.



drug dosage (x)	drug effectiveness (y)
10	-10
20	7
25	8
35	-7

Since problem is regression, the loss function is

$$L = \sum (yi - \hat{y}_i)^2$$

Gradient Boosted Decision Trees

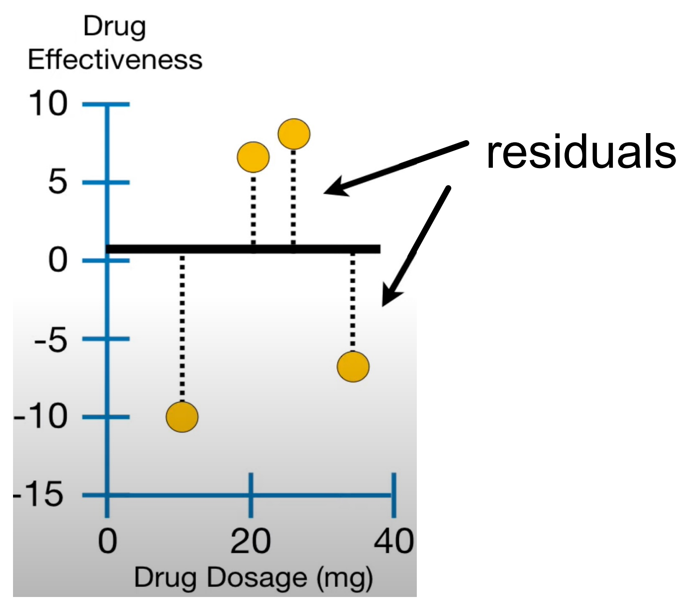
- Start with a constant prediction: $\hat{y}_i^{(0)} = 0.5$

0.5

- Compute *residuals = observed – predicted*

- Build next tree by putting All residuals into a leaf

-10.5, 6.5, 7.5, -7.5



- Compute similarity score

$$\text{similarity score} = \frac{\sum(residuals)^2}{number\ of\ residuals + \lambda}$$

λ is a regularization hyperparameter

Gradient Boosted Decision Trees

- Let's set $\lambda = 0$

-10.5, 6.5, 7.5, -7.5

similarity score = $\frac{(-10.5+6.5+7.5-7.5)^2}{4+0} = 4$

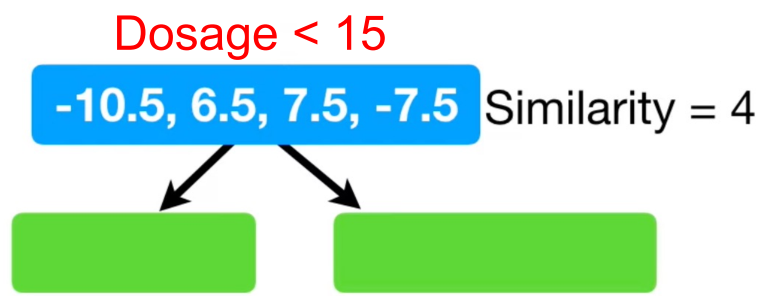
Gradient Boosted Decision Trees

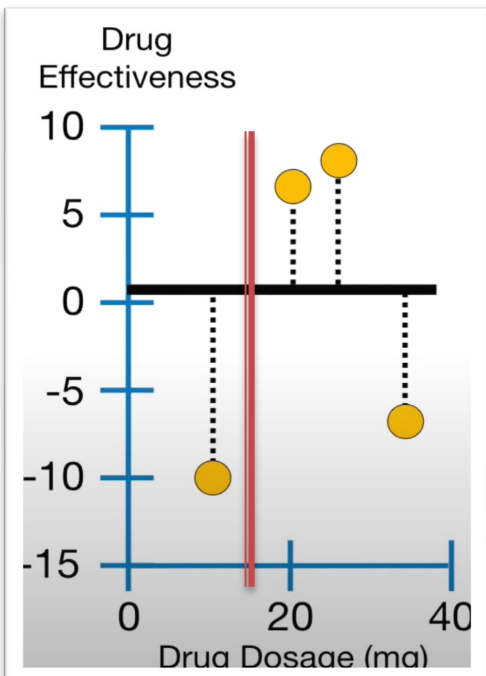
- Let's set $\lambda = 0$

-10.5, 6.5, 7.5, -7.5

$$\text{similarity score} = \frac{(-10.5+6.5+7.5-7.5)^2}{4+0} = 4$$

- Let's grow the tree:





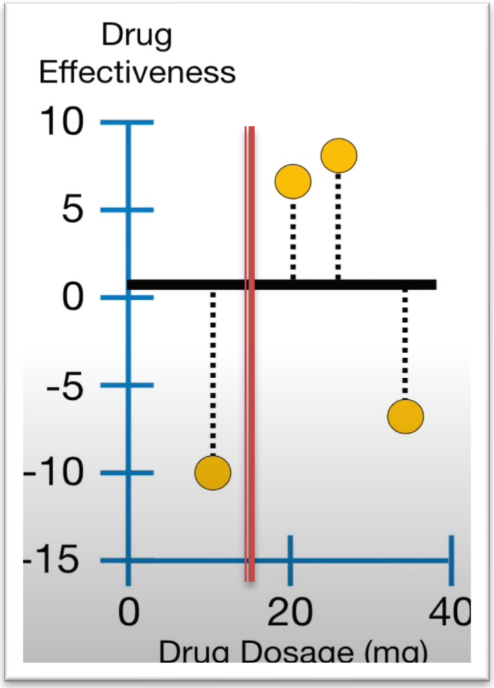
Gradient Boosted Decision Trees

- Let's set $\lambda = 0$

-10.5, 6.5, 7.5, -7.5

$$\text{similarity score} = \frac{(-10.5+6.5+7.5-7.5)^2}{4+0} = 4$$

- Let's grow the tree:



Dosage < 15

-10.5, 6.5, 7.5, -7.5 Similarity = 4

-10.5
Similarity = 110.25

6.5, 7.5, -7.5
Similarity = 14.08

Gain = $\text{Left}_{\text{Similarity}} + \text{Right}_{\text{Similarity}} - \text{Root}_{\text{Similarity}}$

Gain = 110.25 + 14.08 - 4 = 120.33

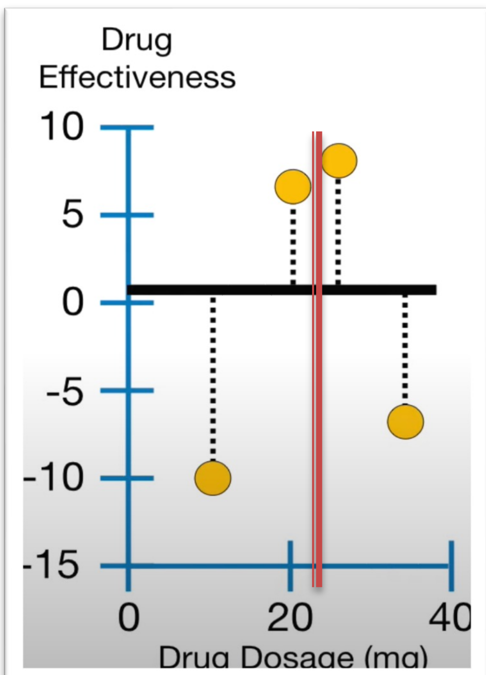
Gradient Boosted Decision Trees

- Let's set $\lambda = 0$

-10.5, 6.5, 7.5, -7.5

similarity score = $\frac{(-10.5+6.5+7.5-7.5)^2}{4+0} = 4$

- Let's grow the tree:



Dosage < 22.5

-10.5, 6.5, 7.5, -7.5

Similarity = 4

-10.5, 6.5

Similarity = 8

7.5, -7.5

Similarity = 0

Gain = $\text{Left}_{\text{Similarity}} + \text{Right}_{\text{Similarity}} - \text{Root}_{\text{Similarity}}$

Gain = 8 + 0 - 4 = 4

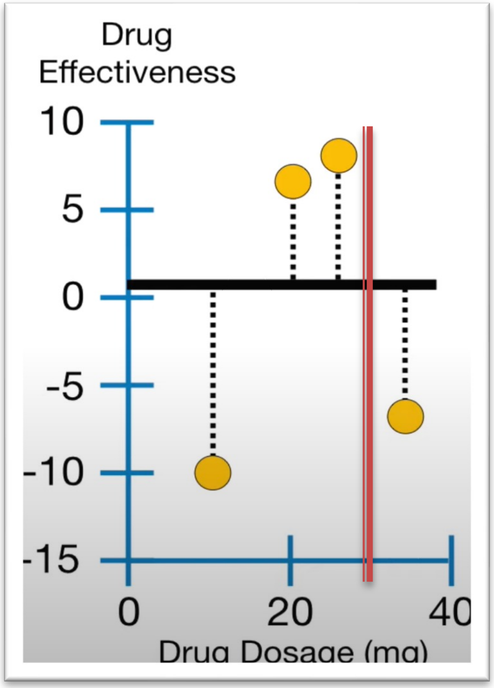
Gradient Boosted Decision Trees

- Let's set $\lambda = 0$

-10.5, 6.5, 7.5, -7.5

similarity score = $\frac{(-10.5+6.5+7.5-7.5)^2}{4+0} = 4$

- Let's grow the tree:



Dosage < 30

-10.5, 6.5, 7.5, -7.5 Similarity = 4

-10.5, 6.5, 7.5 — Similarity = 4.08

-7.5 — Similarity = 56.25

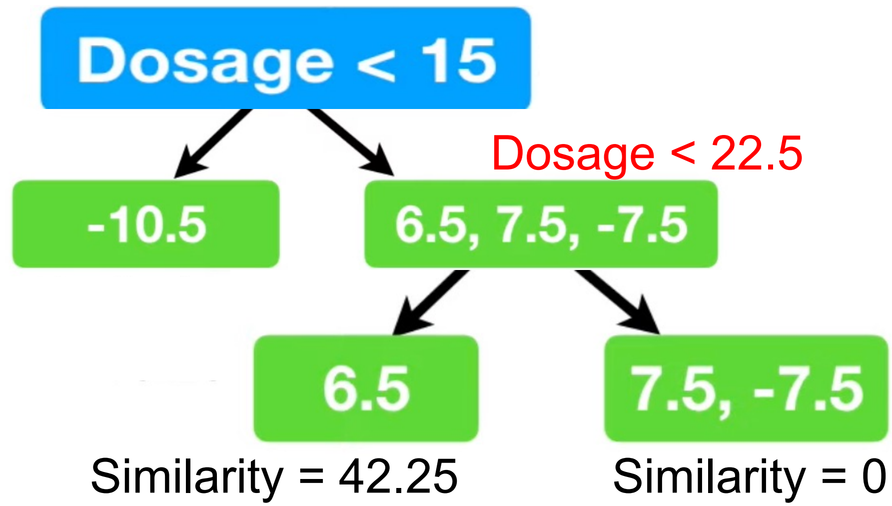
Gain = $\text{Left}_{\text{Similarity}} + \text{Right}_{\text{Similarity}} - \text{Root}_{\text{Similarity}}$

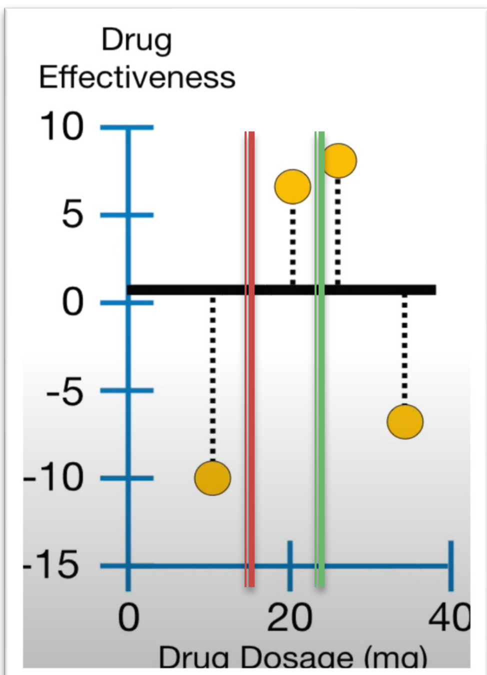
Gain = 4.08 + 56.25 - 4 = 56.33

Gradient Boosted Decision Trees

- So far:

0.5

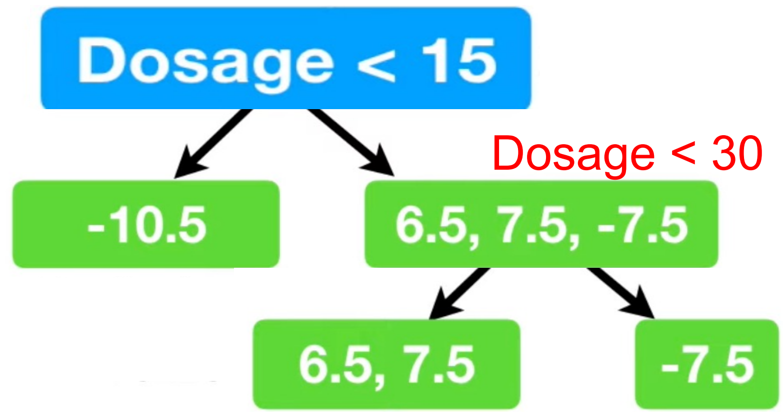




Gain = 42.25 + 0 - 14.08 = 28.17

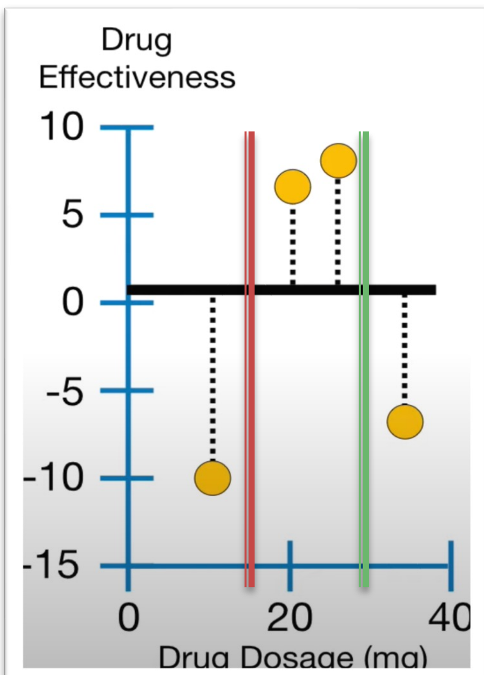
Gradient Boosted Decision Trees

- So far:



Similarity = 98

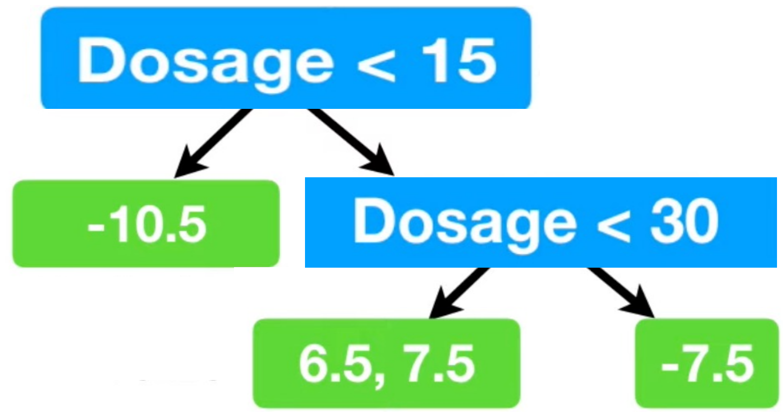
Similarity = 56.25



Gain = 98 + 56.25 - 14.08 = 140.17

Gradient Boosted Decision Trees

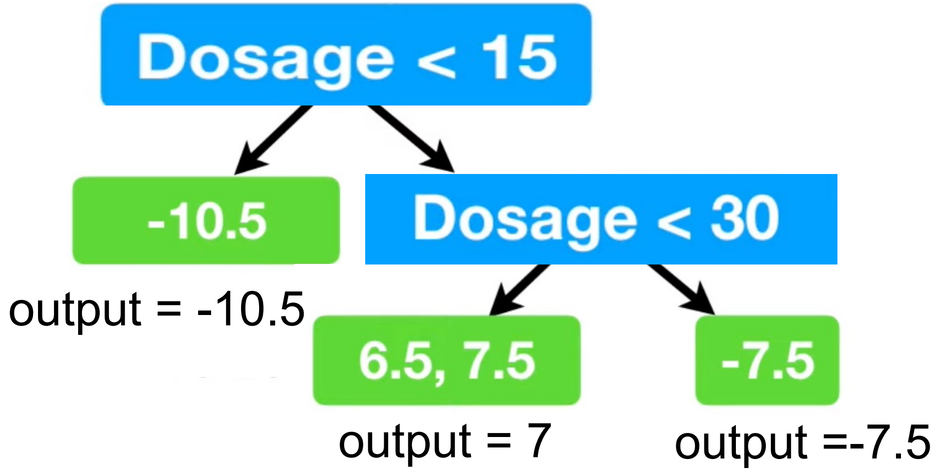
- So far we have:



- Let’s assume we are done growing this tree!

Gradient Boosted Decision Trees

- So far we have:



- To make predictions:
  - Compute an *output value* for each leaf

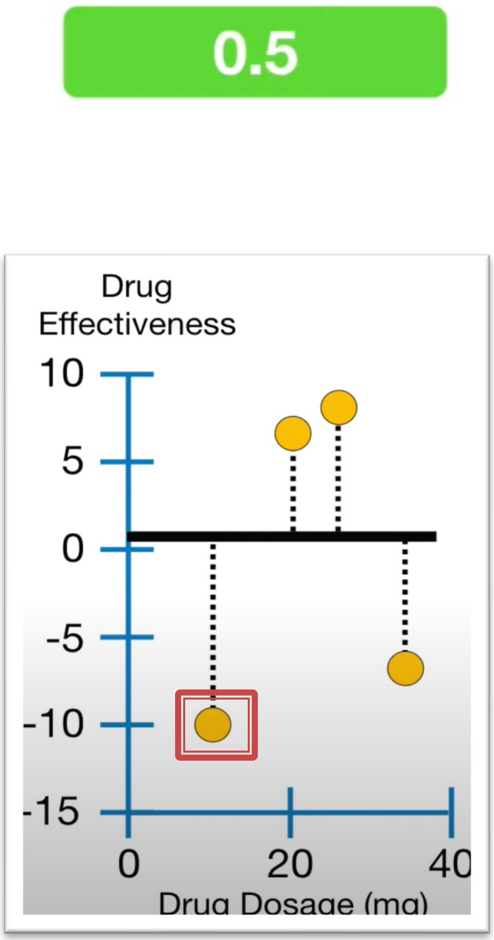
$$\text{Output value}= \frac{\sum residual}{number\ of\ residuals+\lambda}$$

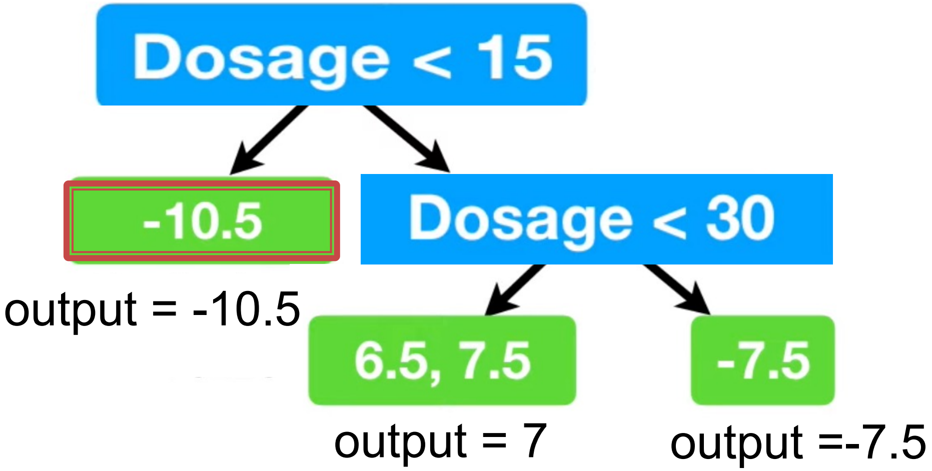
$$\text{New prediction} = \hat{y}^{(0)} + \eta\, \hat{y}^{(1)}$$

η is learning rate, by default 0.3

Gradient Boosted Decision Trees

- So far we have:



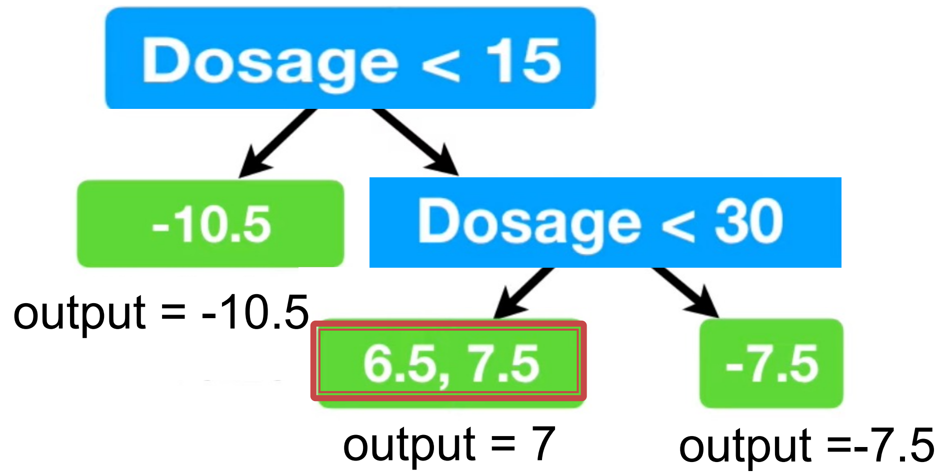


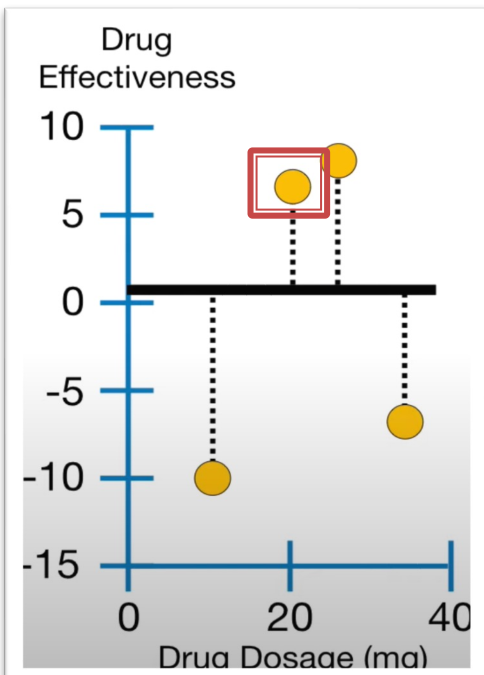
New prediction=0.5 + 0.3 x(-10.5)=-2.65

Gradient Boosted Decision Trees

- So far we have:

0.5





New prediction=0.5 + 0.3 x7=2.6

Gradient Boosted Decision Trees

- To build the next tree:
  - Compute residuals = $\boldsymbol{y} - \widehat{\boldsymbol{y}}$

drug dosage (x)	drug effectiveness (y)	predictions $\widehat{\boldsymbol{y}}$	residual $\boldsymbol{y} - \widehat{\boldsymbol{y}}$
10	-10	-2.65	-7.35
20	7	2.6	4.4
25	8	2.6	5.4
35	-7	-1.75	-5.25

  - Put all residuals into a leaf -7.35, 4.4, 5.4, -5.25
  - Compute its similarity score and split.....

Gradient Boosted Decision Trees

- Now let's look at the math

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

Prediction at training round t

Keep predictions from previous rounds

New model

How to decide which f to add?

- **Prediction at round t is:** $\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$
  - we need to decide what $f_t()$ to add
- **Goal: Find tree $f_t(\cdot)$ that minimizes loss $l()$:**

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

  - y_i: The ground-truth label
  - $\widehat{y_i}^{(t-1)} + f_t(x_i)$: 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:**

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

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?

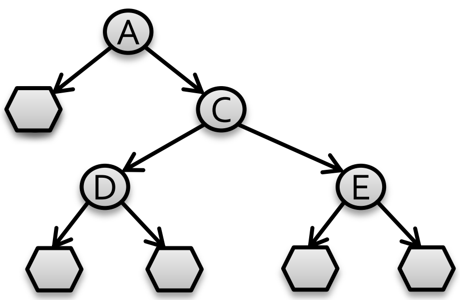
- **The approximate objective:**

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

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

- **Define model 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

w_j is output value of j-th leaf

Revisiting the Objective

- **So our objective is:**

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

- **We can re-write it by leaf**

I_j contains index of data points that are in leaf j

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

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

$$= \sum_{j=1}^{T}\underbrace{\left[\left(\sum_{i\in I_j} g_i\right) w_j + \frac{1}{2}\left(\sum_{i\in I_j} h_i + \lambda\right) w_j^2\right]}_{\text{Associated with leaf node } j} + \gamma T$$

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

Finding the Optimal w_j^*

Each quadratic function associated with leaf j:

$$\underbrace{(\sum_{i\in I_j} g_i)}_{G_j} w_j + \frac{1}{2}(\underbrace{\sum_{i\in I_j} h_i}_{H_j} + \lambda) w_j^2$$

The minimizer is:

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

The minimum value of objective is:

The Obj function measures the quality of the set of T trees. This score is like the impurity score for evaluating decision trees, except that it is derived for a wider range of objective functions

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

For Regression Loss

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

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

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

- **And**

$$w_j^* = -\frac{G_j}{H_j + \lambda} = -\frac{\sum_{i \in Ij} g_i}{\sum_{i \in Ij} h_i + \lambda}$$

For Regression Example

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

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

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

- **And**

$$w_j^* = -\frac{G_j}{H_j + \lambda} = -\frac{\sum_{i \in Ij} g_i}{\sum_{i \in Ij} h_i + \lambda}$$

$$= -\frac{2 \sum_{i \in Ij} (\hat{y}^{(t-1)} - yi)}{\sum_{i \in Ij} 2 + \lambda}$$

This is the output value we saw before

$$\boxed{= \frac{\sum_{i \in Ij} (residual)}{\#\text{examples in leaf} + \lambda/2}}$$

For Regression Example

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

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

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

- **And**

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

$$= -\frac{1}{2}\sum_{j=1}^{T} \frac{4\sum_{i\in Ij}(\hat{y}^{(t-1)} - yi)^2}{\sum_{i\in Ij} 2 + \lambda}$$

$$= -\sum_{j=1}^{T} \boxed{\frac{\sum_{i\in Ij}(residual)^2}{\#\text{examples in leaf} + \lambda/2}}$$

we saw this in similarity score

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:

$$Gain = \frac{1}{2}\left[\underbrace{\frac{G_L^2}{H_L+\lambda}}_{\text{similarity score of left child}} + \underbrace{\frac{G_R^2}{H_R+\lambda}}_{\text{Similarity score of right child}} - \underbrace{\frac{(G_L+G_R)^2}{H_L+H_R+\lambda}}_{\text{Similarity score of parent}}\right] - \gamma$$

  - 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 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**

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

η is called step-size or shrinkage, usually set around 0.1 to 0.3 to prevent overfitting

- **Repeat until we use 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.