Tree based Classification

Dr. Fayyaz Minhas

Department of Computer Science
University of Warwick
https://warwick.ac.uk/fac/sci/dcs/teaching/material/cs909/

Decision Trees

- Task: Predict survival of a passenger on RMS Titanic
- Given features
  - Gender
  - Class
  - Adult or not
- Predict
  - Survived
  - Not survived

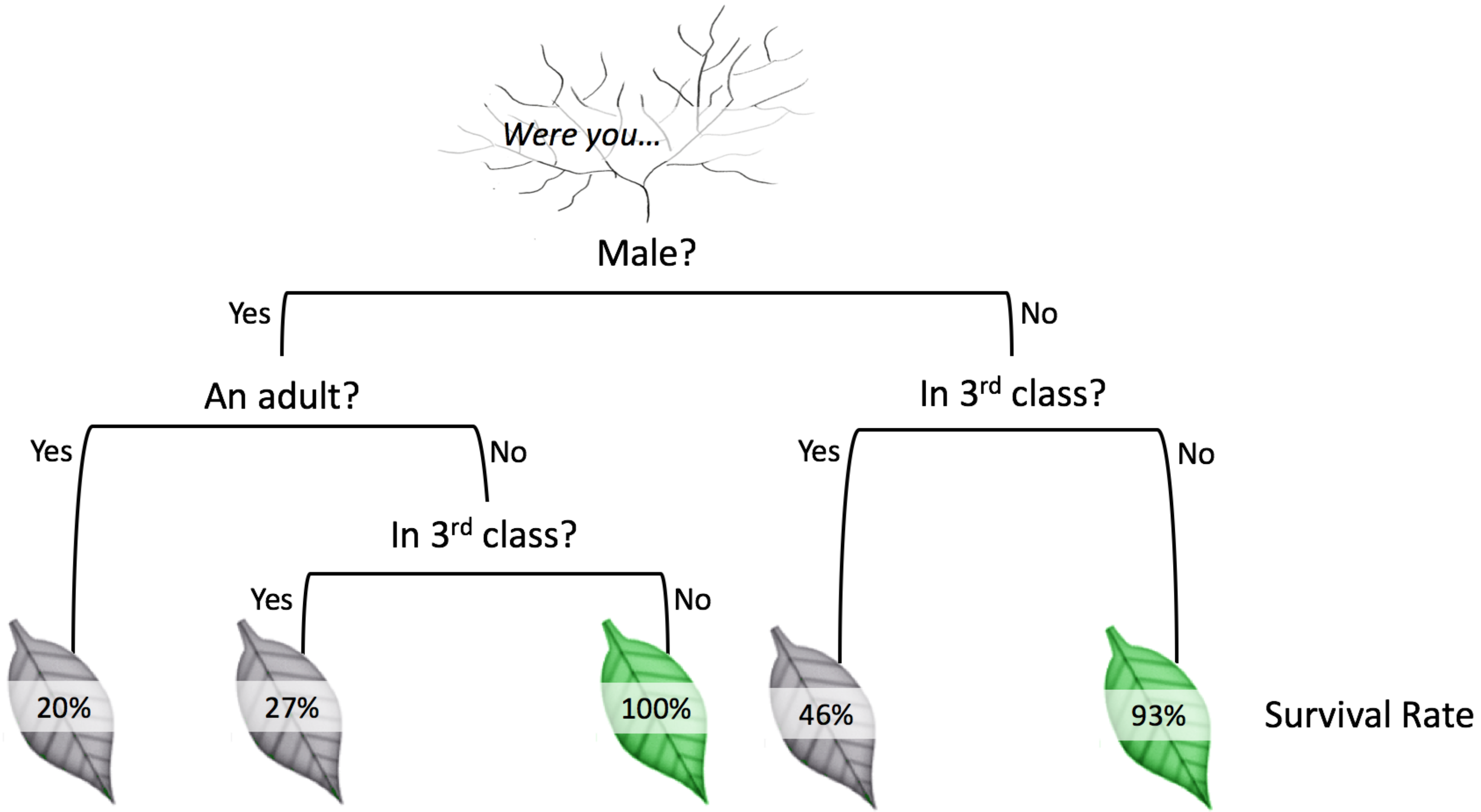
Looking at the data

	A	B	C	D	E	F	G	H	I	J	K	L
1	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
2	1	0	3	Braund, Mr. Owen Harris	male	22	1	0	A/5 21171	7.25		S
3	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Thayer)	female	38	1	0	PC 17599	71.2833	C85	C
4	3	1	3	Heikkinen, Miss. Laina	female	26	0	0	STON/O2. 31	7.925		S
5	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35	1	0	113803	53.1	C123	S
6	5	0	3	Allen, Mr. William Henry	male	35	0	0	373450	8.05		S
7	6	0	3	Moran, Mr. James	male		0	0	330877	8.4583		Q
8	7	0	1	McCarthy, Mr. Timothy J	male	54	0	0	17463	51.8625	E46	S
9	8	0	3	Palsson, Master. Gosta Leonard	male	2	3	1	349909	21.075		S
10	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27	0	2	347742	11.1333		S
11	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14	1	0	237736	30.0708		C
12	11	1	3	Sandstrom, Miss. Marguerite Rut	female	4	1	1	PP 9549	16.7	G6	S
13	12	1	1	Bonnell, Miss. Elizabeth	female	58	0	0	113783	26.55	C103	S
14	13	0	3	Saundercock, Mr. William Henry	male	20	0	0	A/5. 2151	8.05		S
15	14	0	3	Andersson, Mr. Anders Johan	male	39	1	5	347082	31.275		S
16	15	0	3	Vestrom, Miss. Hulda Amanda Adolfina	female	14	0	0	350406	7.8542		S
17	16	1	2	Hewlett, Mrs. (Mary D Kingcome)	female	55	0	0	248706	16		S
18	17	0	3	Rice, Master. Eugene	male	2	4	1	382652	29.125		Q
19	18	1	2	Williams, Mr. Charles Eugene	male		0	0	244373	13		S
20	19	0	3	Vander Planke, Mrs. Julius (Emelia Maria Vandemoortele)	female	31	1	0	345763	18		S
21	20	1	3	Masselmani, Mrs. Fatima	female		0	0	2649	7.225		C
22	21	0	2	Fynney, Mr. Joseph J	male	35	0	0	239865	26		S
23	22	1	2	Beesley, Mr. Lawrence	male	34	0	0	248698	13	D56	S
24	23	1	3	McGowan, Miss. Anna "Annie"	female	15	0	0	330923	8.0292		Q
25	24	1	1	Sloper, Mr. William Thompson	male	28	0	0	113788	35.5	A6	S
26	25	0	3	Palsson, Miss. Torborg Danira	female	8	3	1	349909	21.075		S
27	26	1	3	Asplund, Mrs. Carl Oscar (Selma Augusta Emilia Johansson)	female	38	1	5	347077	31.3875		S
28	27	0	3	Emir, Mr. Farred Chehab	male		0	0	2631	7.225		C

Converting data to numbers

- Numerical
  - Example: Age
- Categorical
  - Nominal: No intrinsic ordering (e.g., Gender)
    - If more than two values, you may want to use a single column for each type
  - Ordinal: Clear Ordering (e.g., Class)
- Sometimes, it may be useful to convert numerical variables to categorical ones or ordinal to nominal ones
  - Age to IsAdult
  - Class to In 3^{rd} Class

Decision Trees



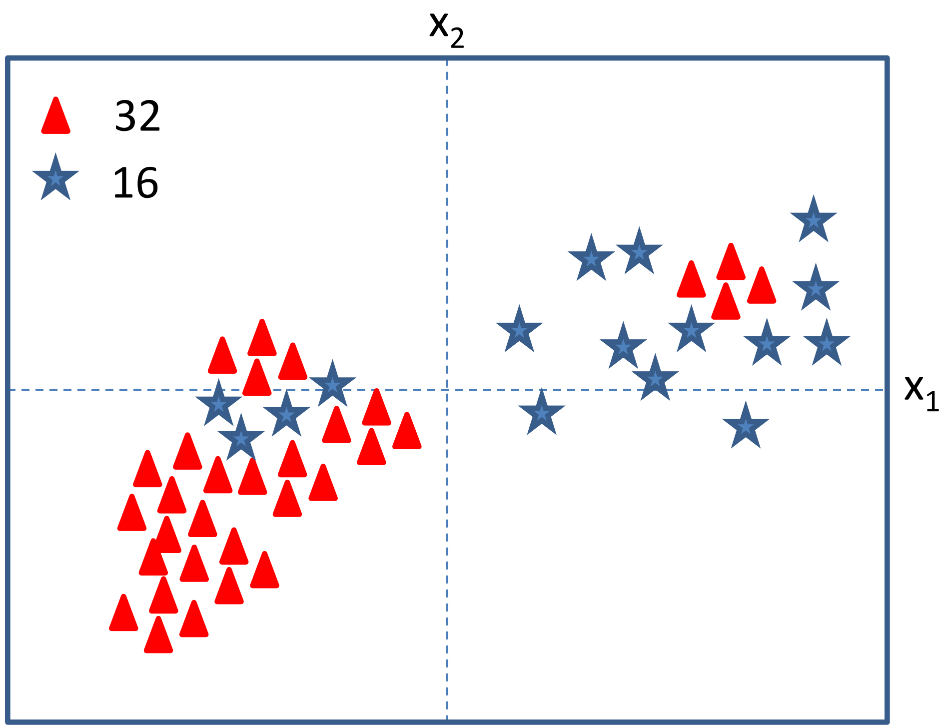
- A tree predicting survival rate for titanic passengers
- A decision tree, in essence, “explains” a dataset by partitioning the space with respect to a single feature at a time

How to build decision trees?

- We need to decide what feature (or attribute) to pick for splitting and what value
  - Simplest case: A single feature divides the data into two groups which correspond to class labels (Done!)
  - Practically: We pick an attribute and its value such that the division into groups based on this attribute leads to “pure” groups
  - Recursively do this to get the tree
  - We use a metric that tells us how purer will be the groups if we use a certain attribute/value for splitting
    - Called **Information Gain**
      - **How much we gain by splitting based on a certain attribute?**

$$IG(T,a) = I(T) - \sum_{k=1}^{|a|} \frac{|a_k|}{N} I(a_k)$$

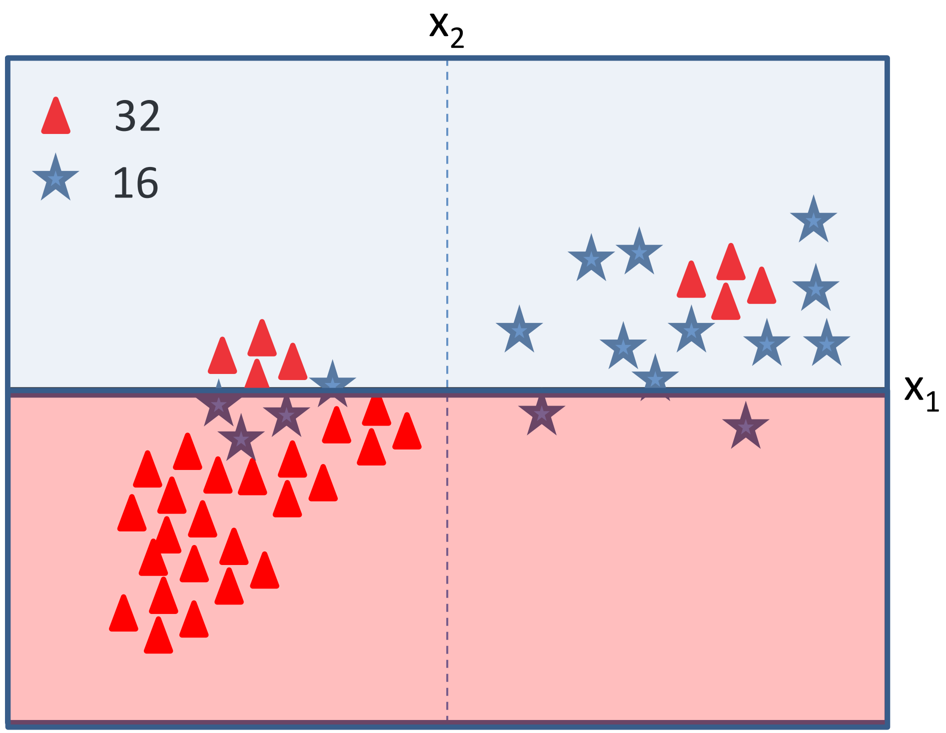
Which feature should we pick first?



Current Error Rate: 16/48 = 1/3

At each step pick the feature that gives the most “information gain” or the most reduction in error

Check x_2



Total points: 48
Current Error Rate: 16/48 = 1/3

For a split along $x_2 = 0$
Total points in the top half = 19 out of 48
Error in the top half: 8/19

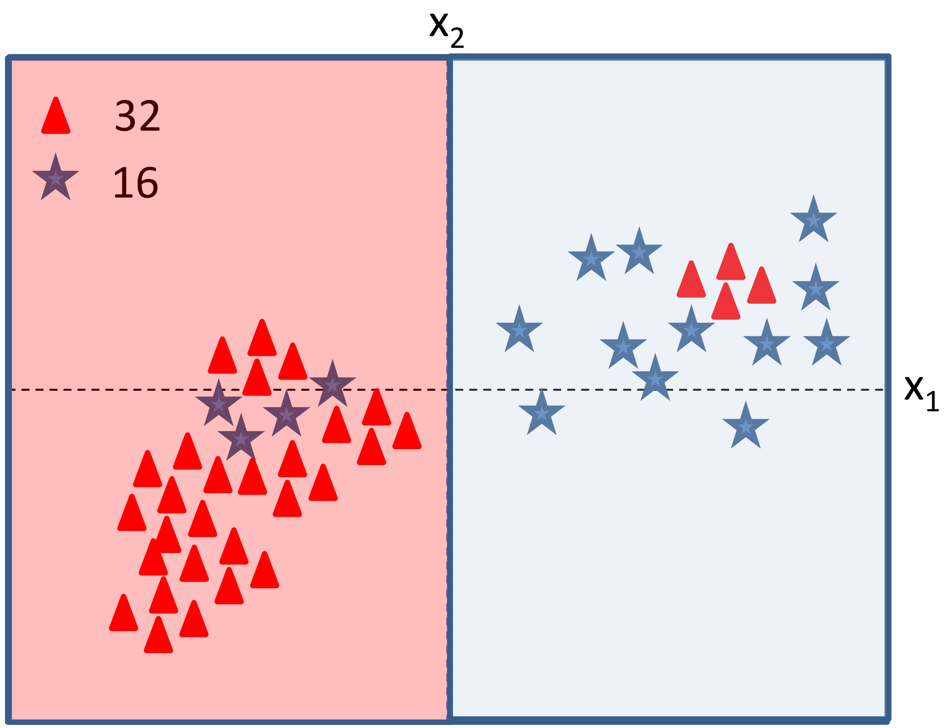
Total points in the bottom half: 29
Error in the bottom half = 5/29

Total error: $\frac{8}{19}\frac{19}{48} + \frac{5}{29}\frac{29}{48} = \ 13/48$

Reduction in error = 16/48-13/48 = 3/48

At each step pick the feature that gives the most “information gain”

Check x_1



x_1 Gives the most improvement in error rate

Total points: 48
Current Error Rate: 16/48 = 1/3

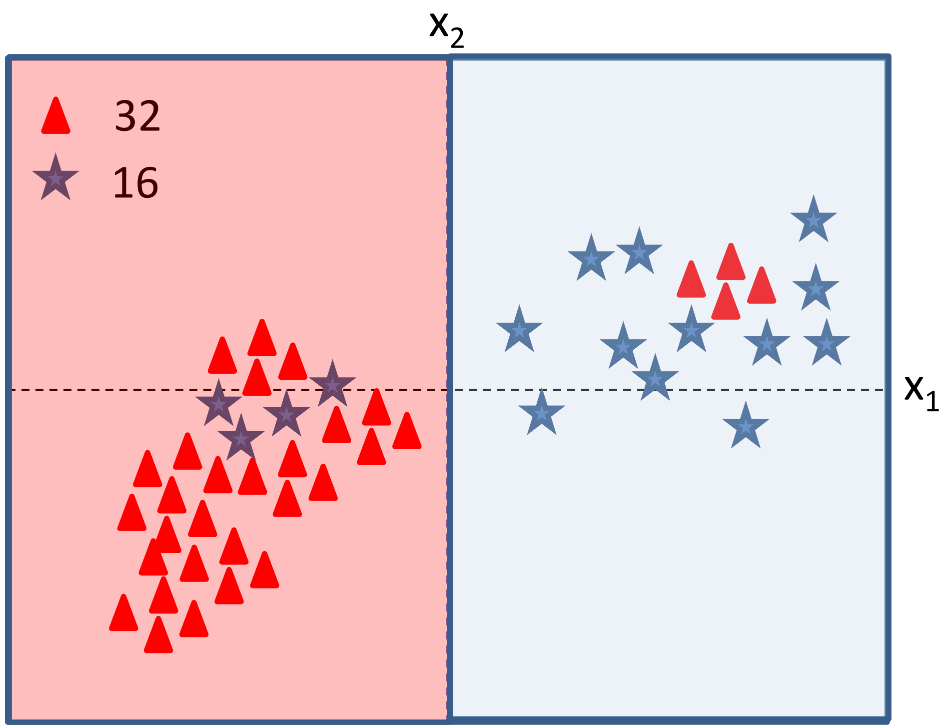
For a split along x_1 = 0
Total points in the L half = 32 out of 48
Error in the L half: 4/32

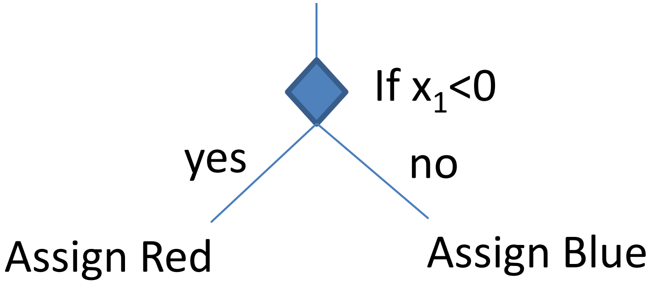
Total points in the R half: 16
Error in the bottom half = 4/16

Total error: $\frac{4}{32}\frac{32}{48} + \frac{4}{16}\frac{16}{48}$ = 8/48

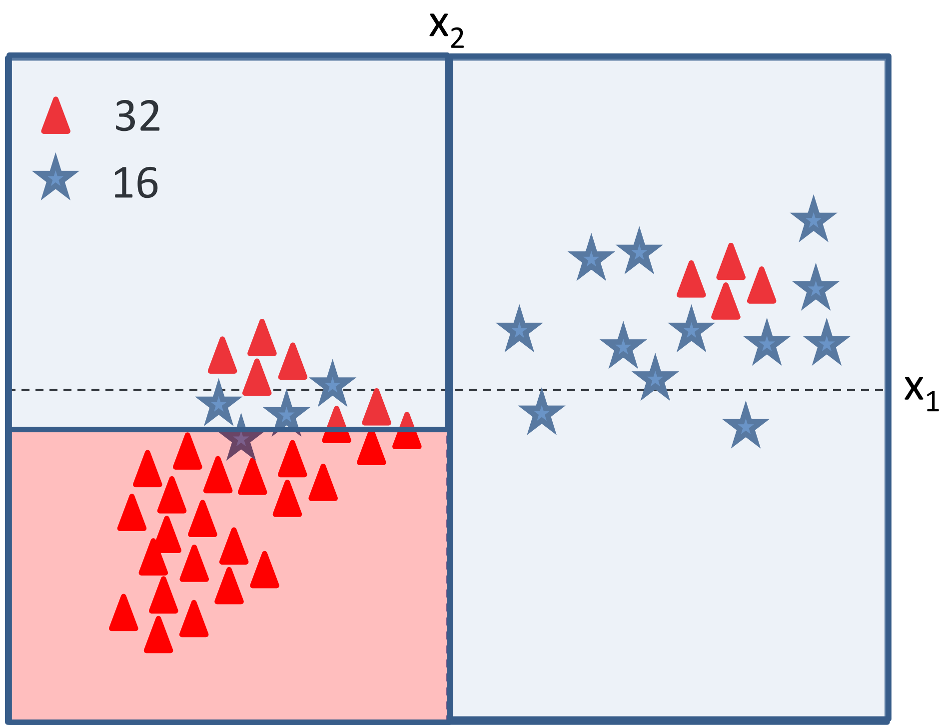
Reduction in error = 16/48-8/48 = 8/48

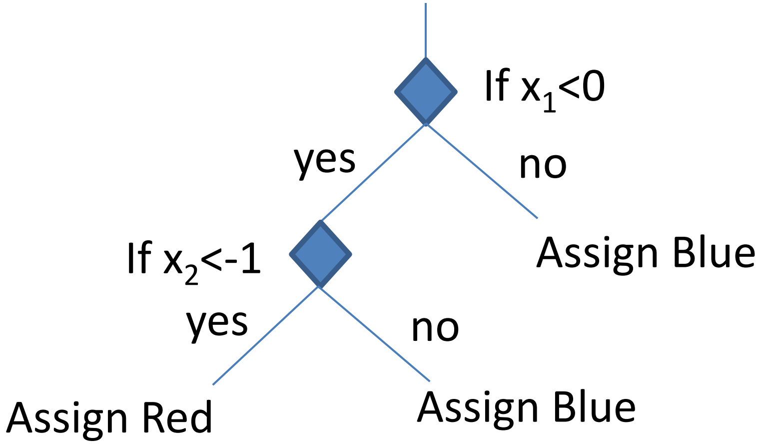
Continuing: Depth = 1



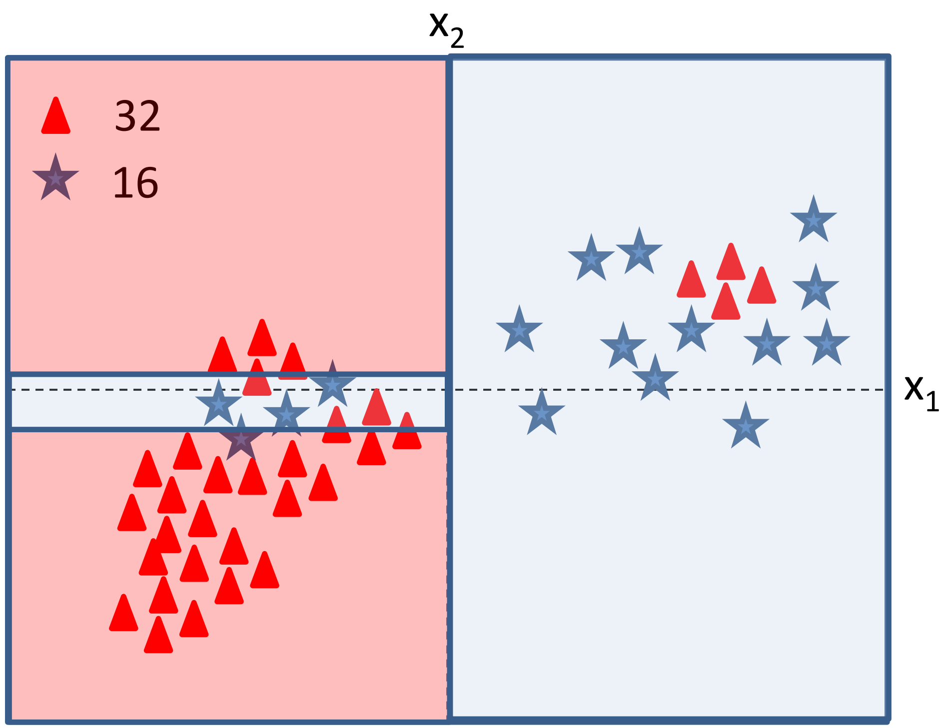


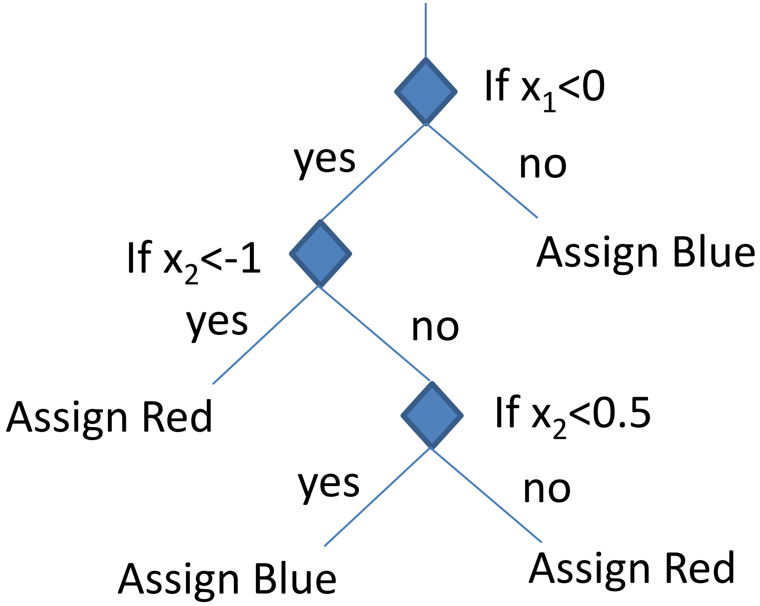
Continuing: Depth = 2



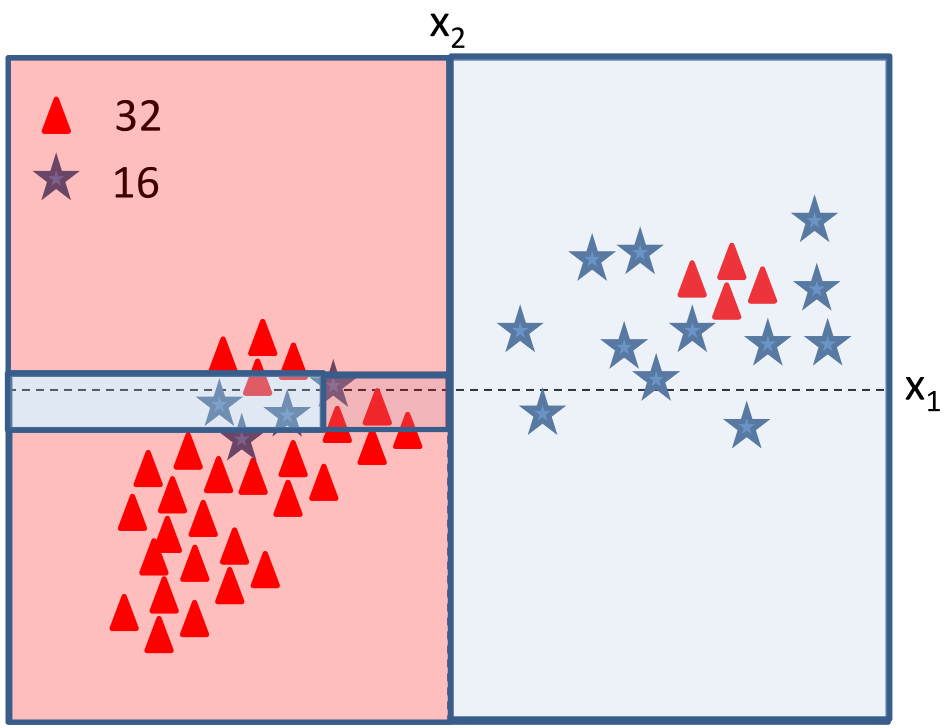


Continuing: Depth = 3

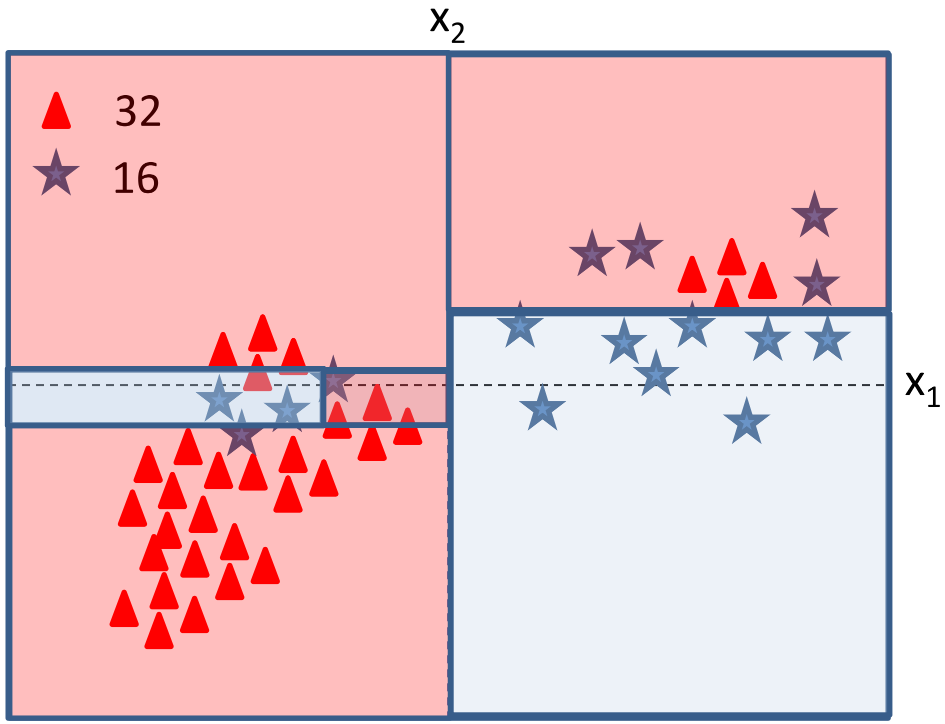




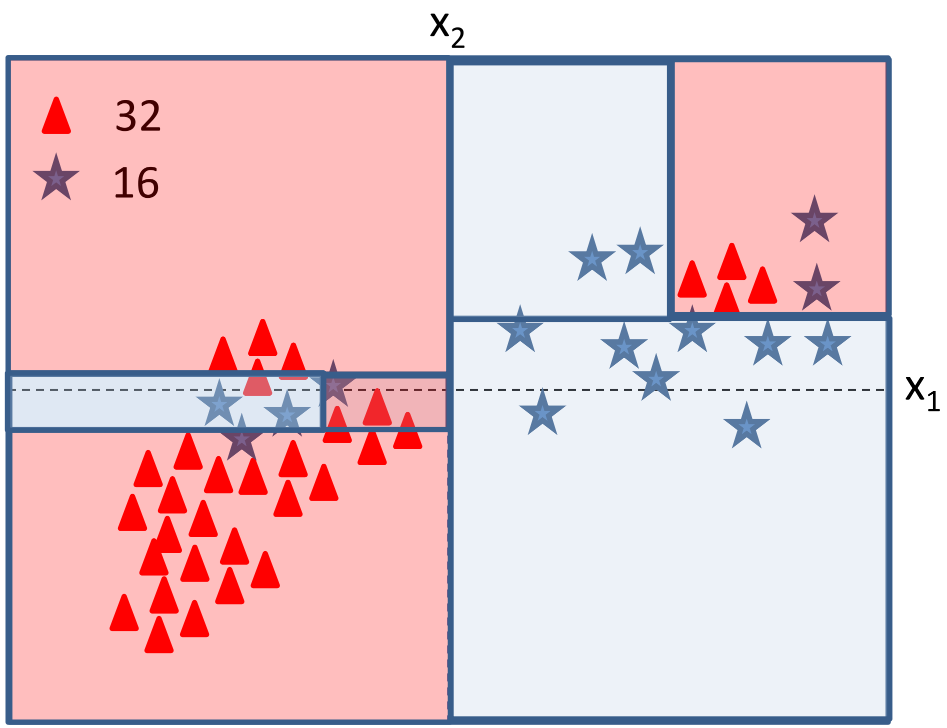
Continuing



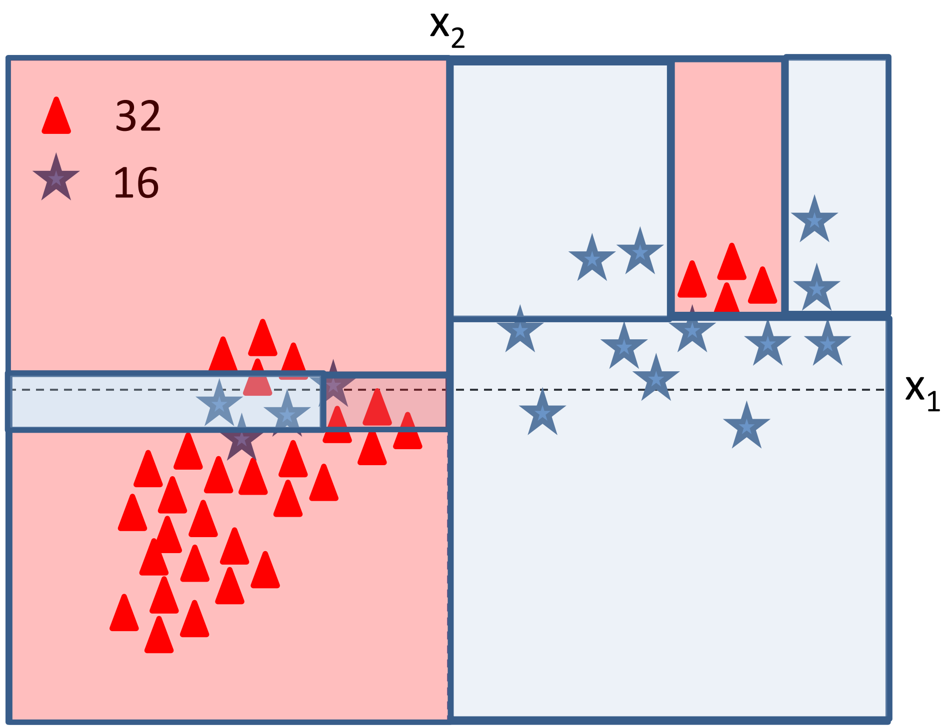
Continuing

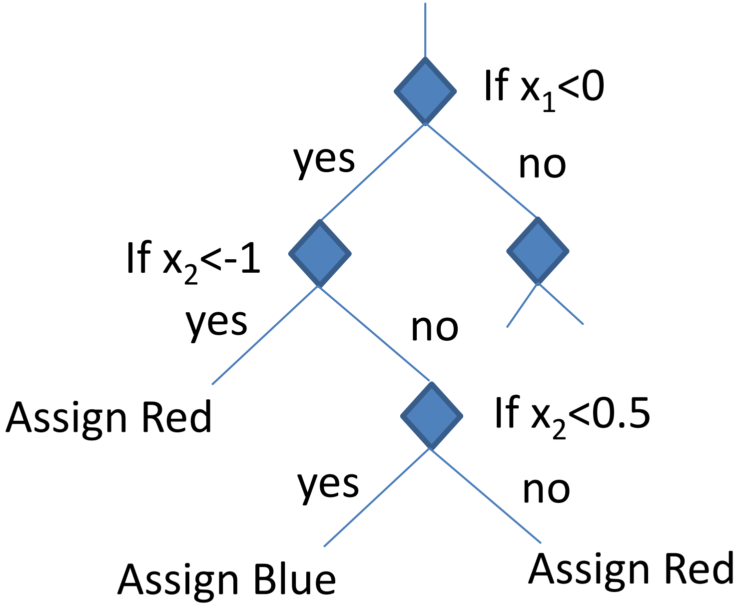


Which feature should we pick first?



Final


x2
32
16
x1

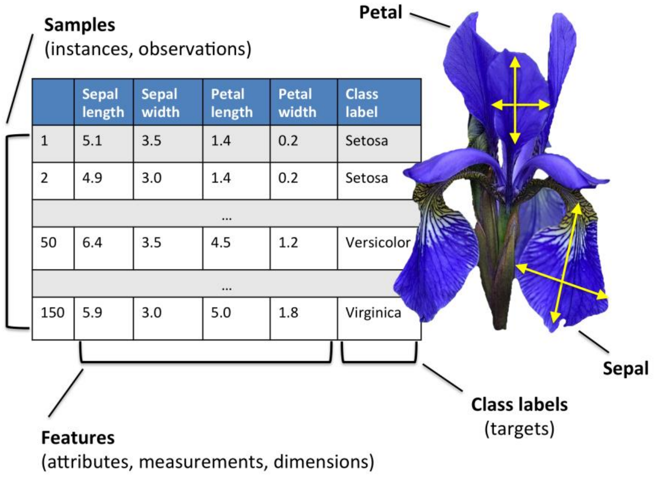

If x1<0
yes
no
If x2<-1
yes
no
Assign Red
If x2<0.5
yes
no
Assign Blue
Assign Red

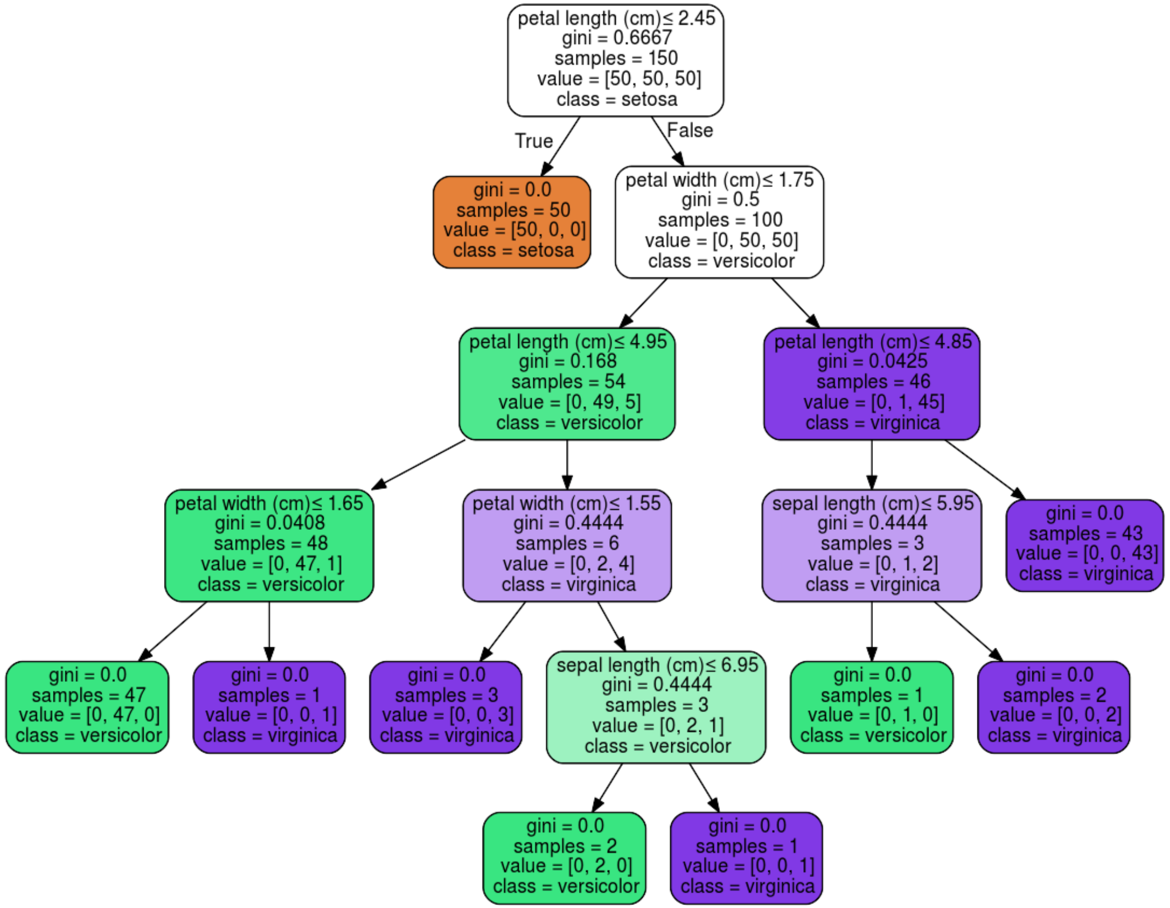
Other ways of defining Information Gain

- We can define information gain based on
  - Error
  - Weighted error
  - Entropy (Measures how much “disordered” each branch is)
  - Gini (Measure how much “pure” each branch is)

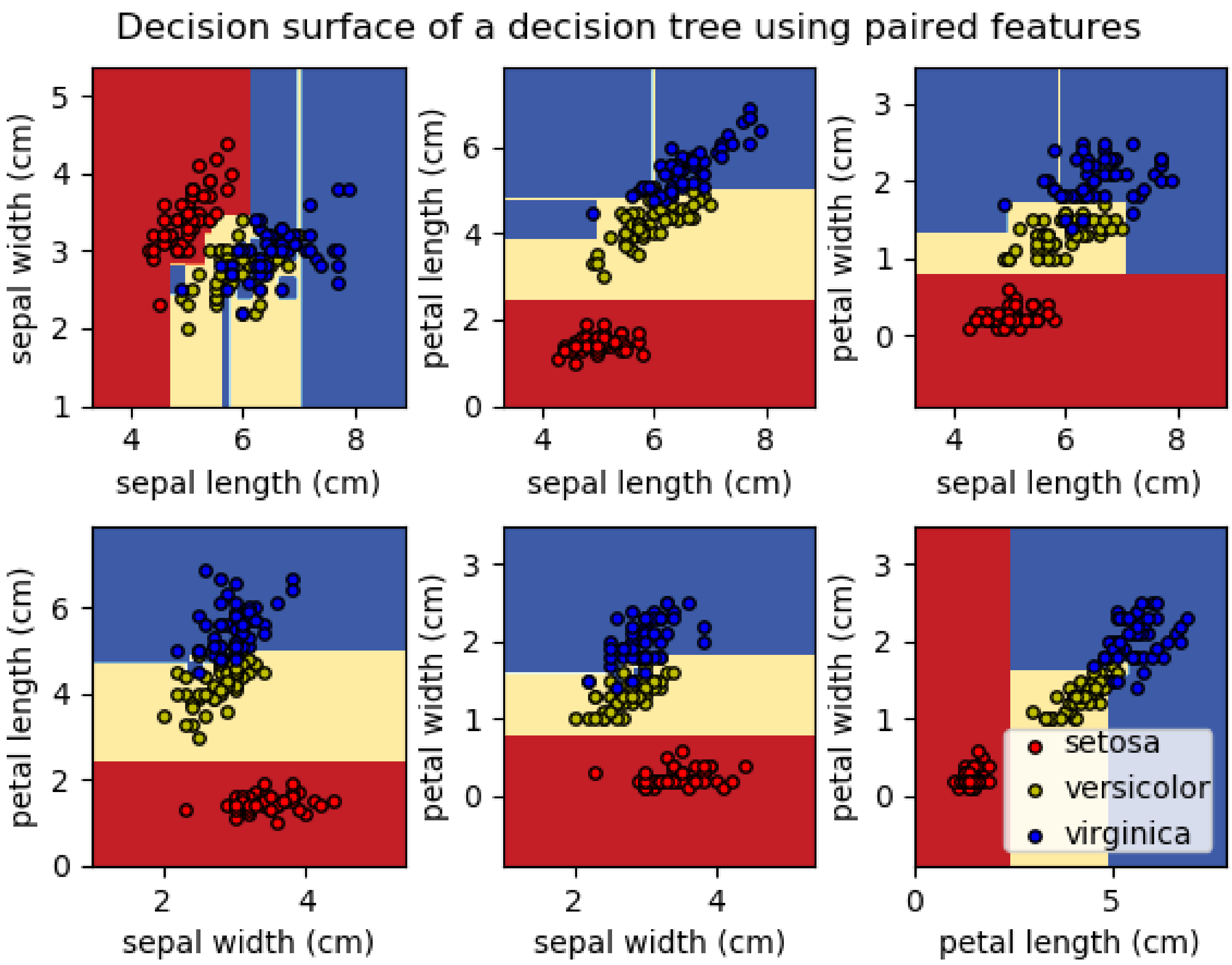
Using sklearn

```
from sklearn import tree
clf = tree.DecisionTreeClassifier()
clf = clf.fit(X, Y)
```





https://scikit-learn.org/stable/modules/tree.html


Decision surface of a decision tree using paired features
sepal width (cm)
sepal length (cm)
petal length (cm)
sepal length (cm)
petal width (cm)
sepal length (cm)
petal length (cm)
sepal width (cm)
petal width (cm)
sepal width (cm)
petal width (cm)
petal length (cm)
setosa
versicolor
virginica

A detailed look

```
DecisionTreeClassifier(criterion='gini', splitter='best’,
            max_depth=None,
            min_samples_split=2,
            min_samples_leaf=1,
            min_weight_fraction_leaf=0.0,
            max_features=None,
            max_leaf_nodes=None,
            min_impurity_decrease=0.0,
            min_impurity_split=None,
            class_weight=None)
```

- A large number of hyperparameters
  - Require careful selection
  - You need to understand the role of each one of them

Choosing depth

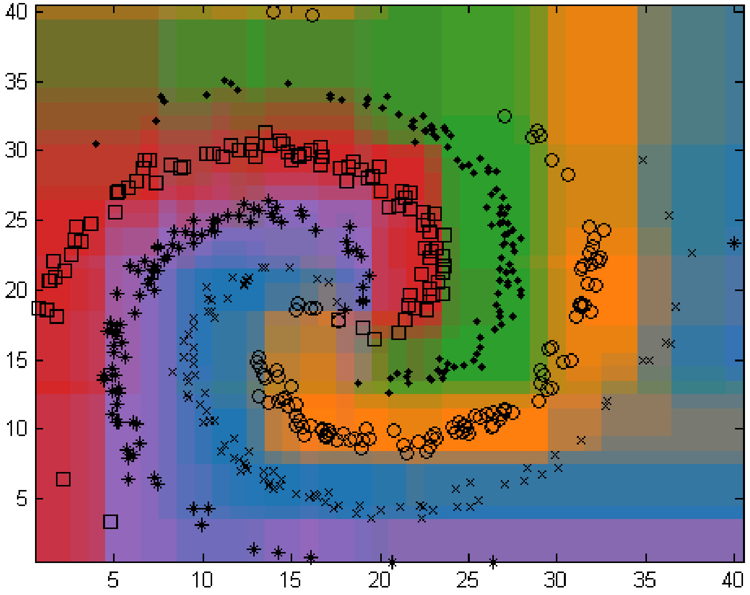
- How deep to go?
  - The shallow we stay, the higher empirical error but the simpler the boundary
  - The deeper we go, the lower the empirical error but the more complex the boundary
    - May lead to poorer generalization

Advantages and Disadvantages

Advantages	Disadvantages
Simple to understand and interpret	Less Accurate
Able to handle both numerical and categorical data	Optimal Decision Tree learning is NP-complete
Requires little data preparation	Sensitive to data changes
Uses a white box model	Can create overly complex boundaries
Possible to validate a model using statistical tests	Impurity metrics can bias results to more levels
Non-statistical approach that makes no assumptions of the training data or prediction residuals	Complexity control through tree depth parameter
Built-in feature selection and interpretation	Practical implementation needs some “tricks”

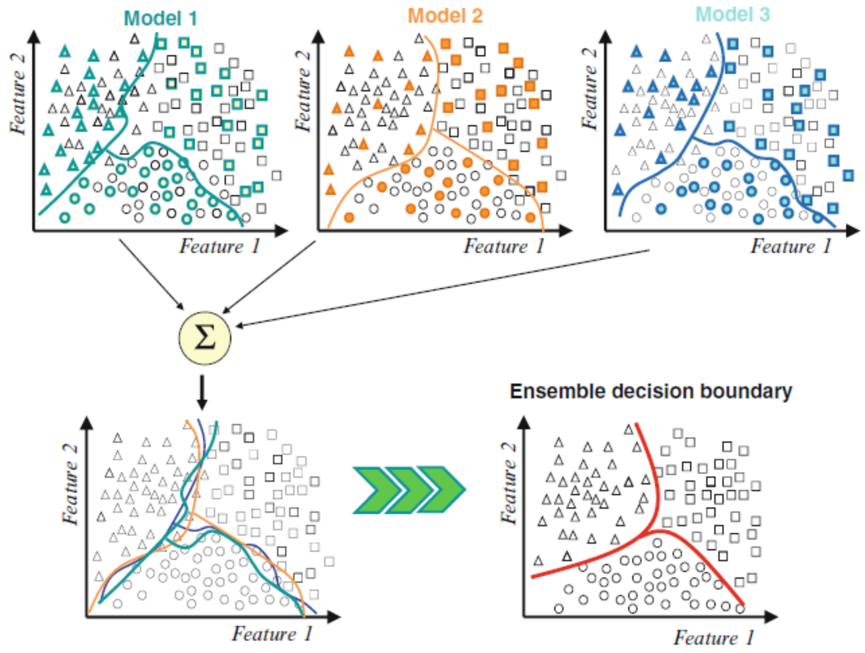
The curious case of weak learners

- Shallow trees give weak classification
  - However, if we combine the outputs of many weak learners, we can get a strong learner
  - Change data given to each tree learner
  - Change features given



Ensemble Methods

- Combine the predictions from multiple “weak” learners
  - Uncorrelated errors in predictions
    - Each learner makes errors on different examples
  - If errors are correlated, little advantage in combining the classifiers
- How to make different classifiers
  - Different Data set partitioning
  - Different Features
  - Different parameters
  - Learning errors from previously trained methods



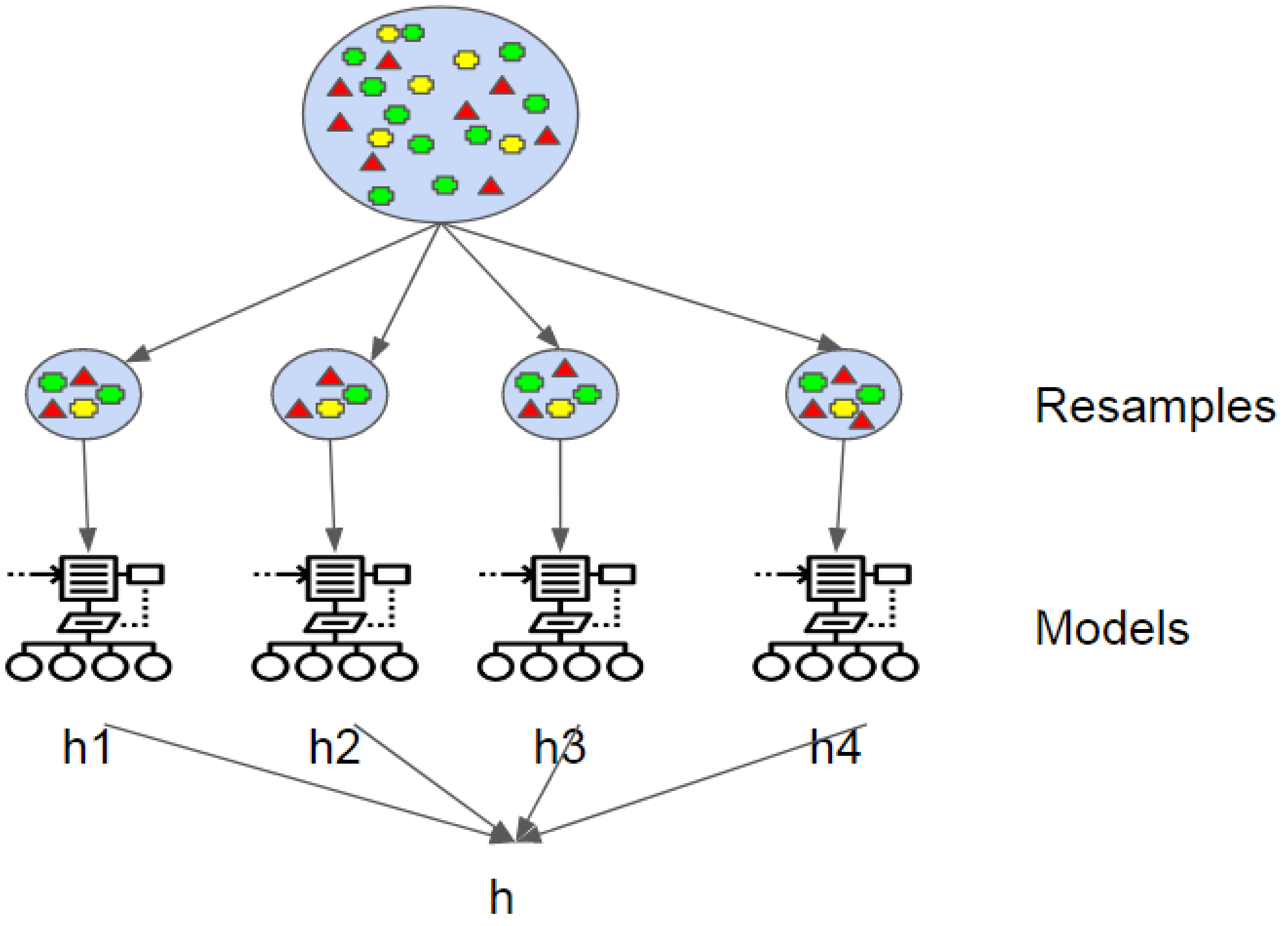
Polikar 2006: http://users.rowan.edu/~polikar/RESEARCH/PUBLICATIONS/csm06.pdf
Ensemble Machine Learning Methods and Applications (chapter 1), 2012
https://doc.lagout.org/science/Artificial%20Intelligence/Machine%20learning/Ensemble%20Machine%20Learning_%20Methods%20and%20Applications%20%5BZhang%20%26%20Ma%202012-02-17%5D.pdf

Ensemble Methods

- Bootstrap Aggregation (Bagging)
  - Involves having each model in the ensemble vote with equal weight.
  - Trains each model in the ensemble using a randomly drawn subset of the training set.
  - **Random Forest algorithm**

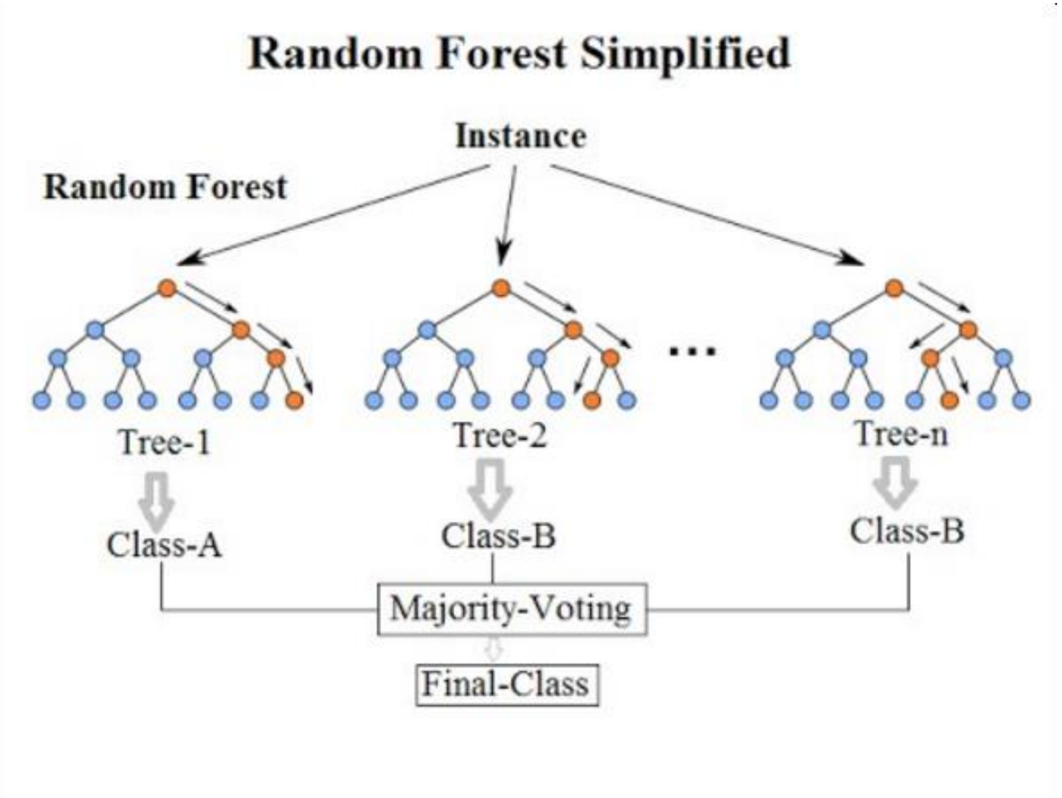
https://en.wikipedia.org/wiki/Ensemble_learning

Bagging

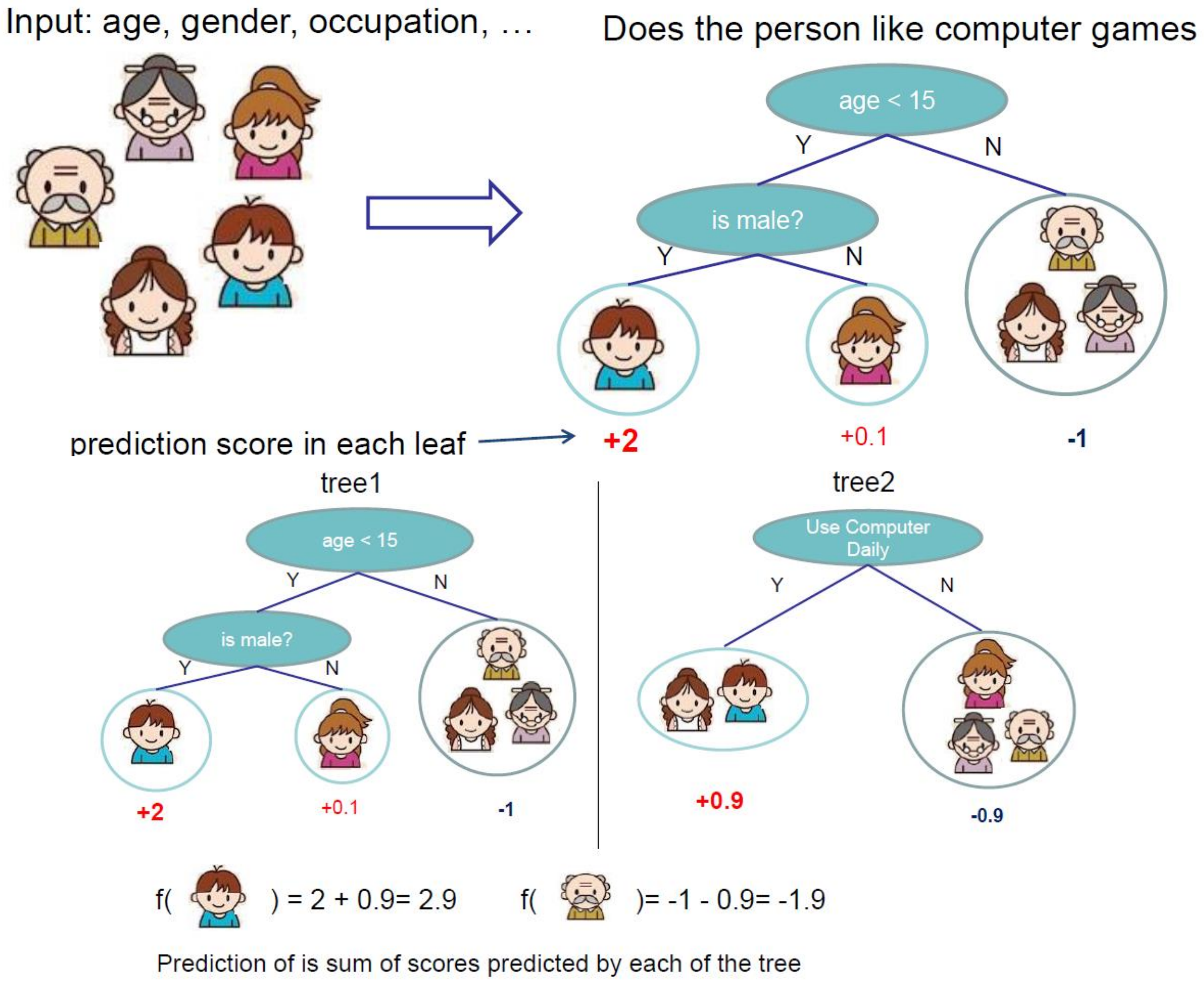


Random Forest Classification

sklearn.ensemble.RandomForestClassifier(n_estimators=100, max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, bootstrap=True, class_weight=None, max_samples=None)

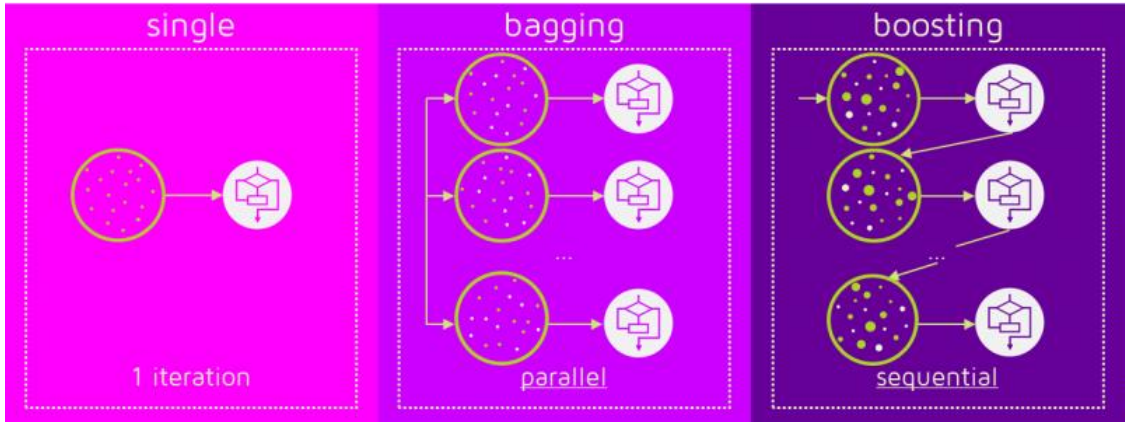


Each leaf can produce a weighted output as well



Ensemble Methods

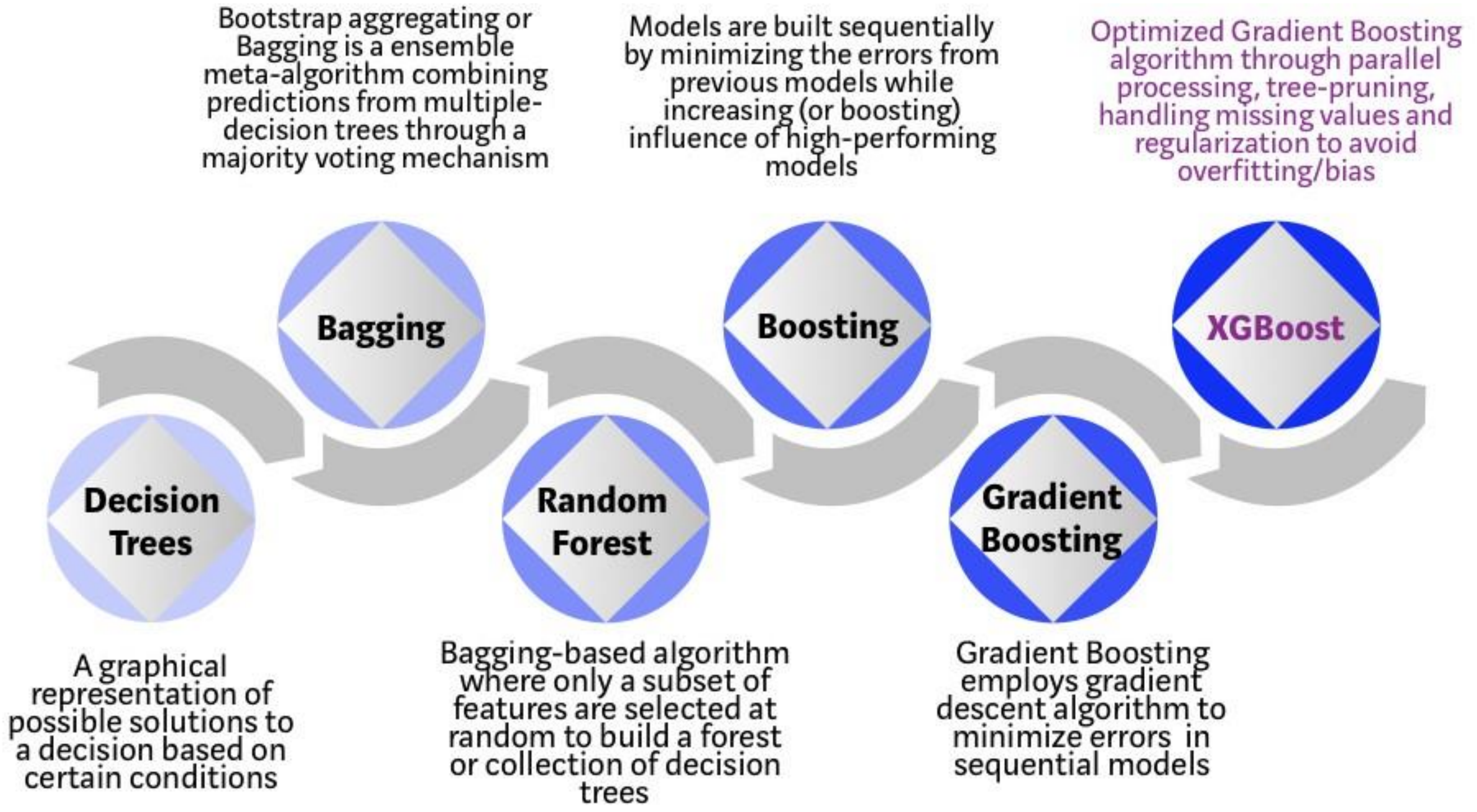
- Boosting
  - Boosting involves incrementally building an ensemble by training each new model instance to emphasize the training instances that previous models mis-classified.
  - Adaboost
  - **Gradient Boosted Trees (XGBoost)**



XGBoost: A Scalable Tree Boosting System

- An implementation of gradient-boosted trees
- Uses structural risk minimization
- Incrementally builds a machine learning model by combining simple trees
- Very successful in different Kaggle Competitions
- Easy to use

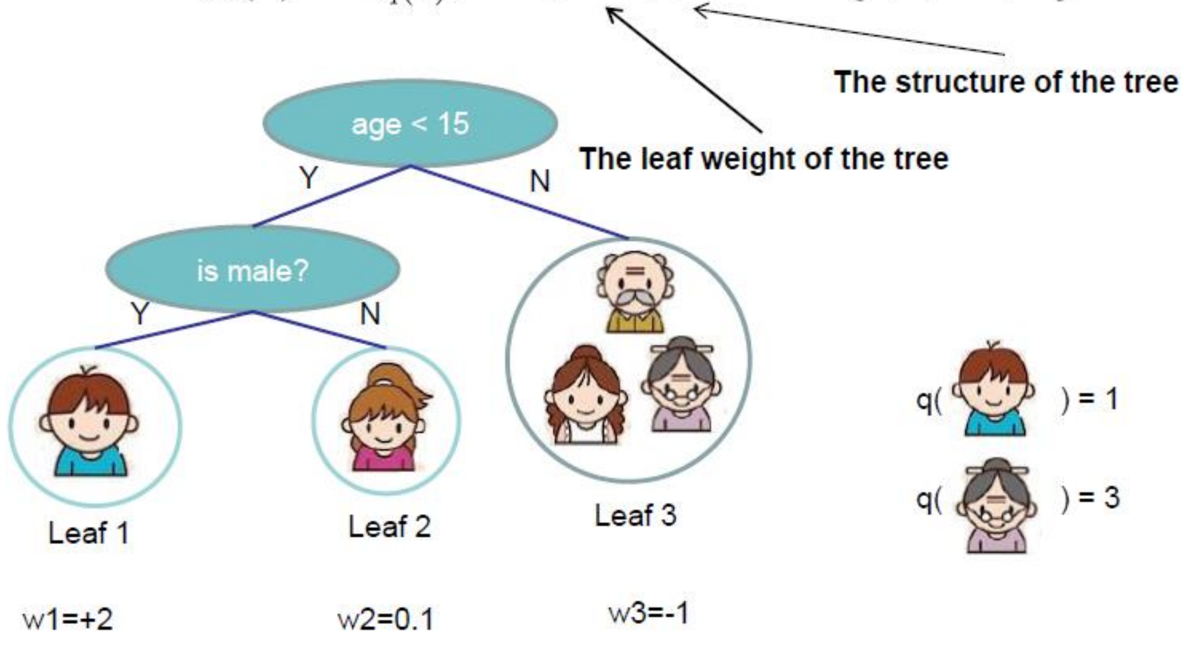
XGBoost


Bootstrap aggregating or Bagging is a ensemble meta-algorithm combining predictions from multiple-decision trees through a majority voting mechanism
Models are built sequentially by minimizing the errors from previous models while increasing (or boosting) influence of high-performing models
Optimized Gradient Boosting algorithm through parallel processing, tree-pruning, handling missing values and regularization to avoid overfitting/bias
Bagging
Boosting
XGBoost
Decision Trees
Random Forest
Gradient Boosting
A graphical representation of possible solutions to a decision based on certain conditions
Bagging-based algorithm where only a subset of features are selected at random to build a forest or collection of decision trees
Gradient Boosting employs gradient descent algorithm to minimize errors in sequential models

Algorithm Description

- Representation
  - A collection of trees
    - For a given example, the outputs of all trees is added to produce the final output

$$f_t(x) = w_{q(x)}, \quad w \in \mathbf{R}^T, q : \mathbf{R}^d \to \{1, 2, \cdots, T\}$$



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

Algorithm Description

- Evaluation
  - Using Structural Risk Minimization
    - Regularization
      - Reduce the number of tree leaves
      - Reduce the weight of the leaves
    - Reduce the empirical error

- Optimization
  - At t^{th} step, learn a single tree using gradient descent that to reduce the error

XGBoost Parameters

subsample
- ratio of instances to take
- example values: 0.6, 0.9

colsample_by_tree
- ratio of columns used for whole tree creation
- example values: 0.1, …, 0.9

colsample_by_level
- ratio of columns sampled for each split
- example values: 0.1, …, 0.9

eta
- how fast algorithm will learn (shrinkage)
- example values: 0.01, 0.05, 0.1

max_depth
- maximum numer of consecutive splits
- example values: 1, …, 15 (for dozen or so or more, needs to be set with regularization parametrs)

min_child_weight
- minimum weight of children in leaf, needs to be adopted for each measure
- example values: 1 (for linear regression it would be one example, for classification it is gradient of pseudo-residual)

alpha
- L1 norm (simple average) of weights for whole objective function

lambda
- L2 norm (root from average of squares) of weights, added as penalty to objective function

gamma
- L0 norm, multiplied by numer of leafs in a tree is used to decide whether to make a split

https://machinelearningmastery.com/develop-first-xgboost-model-python-scikit-learn/

SHAP: A unified approach to explain the output of any machine learning model

```
# https://github.com/slundberg/shap
import xgboost
import shap
# load JS visualization code to notebook
shap.initjs()

# train XGBoost model
X,y = shap.datasets.boston()
model = xgboost.train({"learning_rate": 0.01},
xgboost.DMatrix(X, label=y), 100)

# explain the model's predictions using SHAP values
# (same syntax works for LightGBM, CatBoost, and scikit-
learn models)
shap_values = shap.TreeExplainer(model).shap_values(X)
# visualize the first prediction's explanation
shap.force_plot(shap_values[0,:], X.iloc[0,:])
```

References

- **XGBoost: A Scalable Tree Boosting System**
- https://www.slideshare.net/JaroslawSzymczak1/xgboost-the-algorithm-that-wins-every-competition
  - Especially the feature importance
- https://homes.cs.washington.edu/~tqchen/pdf/BoostedTree.pdf

Class Notebook

- https://github.com/foxtrotmike/CS909/blob/master/trees.ipynb

- Hyperparameter guide
  - https://xgboost.readthedocs.io/en/latest/tutorials/param_tuning.html
- Missing values: built-in handling
  - https://datascience.stackexchange.com/questions/15305/how-does-xgboost-learn-what-are-the-inputs-for-missing-values
  - https://towardsdatascience.com/xgboost-is-not-black-magic-56ca013144b4

LightGBM and catboost

- Faster and Lighter GBM with early stopping
  - https://lightgbm.readthedocs.io/en/latest/Python-Intro.html
  - However, xgboost also supports a similar model now
    - https://github.com/dmlc/xgboost/issues/1950
- https://catboost.ai/

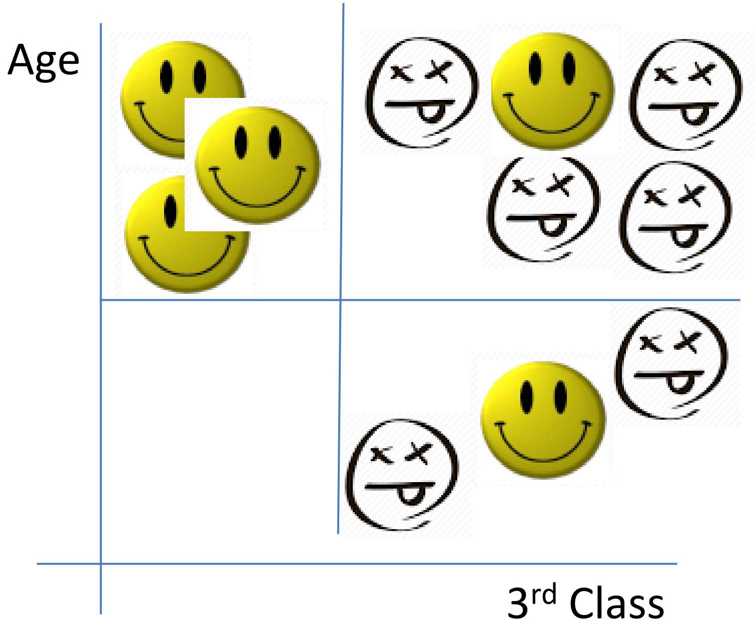
Function	XGBoost	CatBoost	Light GBM
Important parameters which control overfitting	1. **learning_rate or eta** – optimal values lie between 0.01-0.2 2. **max_depth** 3. **min_child_weight:** similar to min_child leaf; default is 1	1. **Learning_rate** 2. **Depth** - value can be any integer up to 16. Recommended - [1 to 10] 3. No such feature like min_child_weight 4. **l2-leaf-reg**: L2 regularization coefficient. Used for leaf value calculation (any positive integer allowed)	1. **learning_rate** 2. **max_depth**: default is 20. Important to note that tree still grows leaf-wise. Hence it is important to tune **num_leaves** (number of leaves in a tree) which should be smaller than 2^(max_depth). It is a very important parameter for LGBM 3. **min_data_in_leaf**: default=20, alias= min_data, min_child_samples
Parameters for categorical values	Not Available	1. **cat_features**: It denotes the index of categorical features 2. **one_hot_max_size**: Use one-hot encoding for all features with number of different values less than or equal to the given parameter value (max – 255)	1. **categorical_feature**: specify the categorical features we want to use for training our model
Parameters for controlling speed	1. **colsample_bytree**: subsample ratio of columns 2. **subsample**: subsample ratio of the training instance 3. **n_estimators**: maximum number of decision trees; high value can lead to overfitting	1. **rsm:** Random subspace method. The percentage of features to use at each split selection 2. No such parameter to subset data 3. **iterations**: maximum number of trees that can be built; high value can lead to overfitting	1. **feature_fraction:** fraction of features to be taken for each iteration 2. **bagging_fraction:** data to be used for each iteration and is generally used to speed up the training and avoid overfitting 3. **num_iterations:** number of boosting iterations to be performed; default=100

End of Lecture

We want to make a machine that will be proud of us.

- Danny Hillis

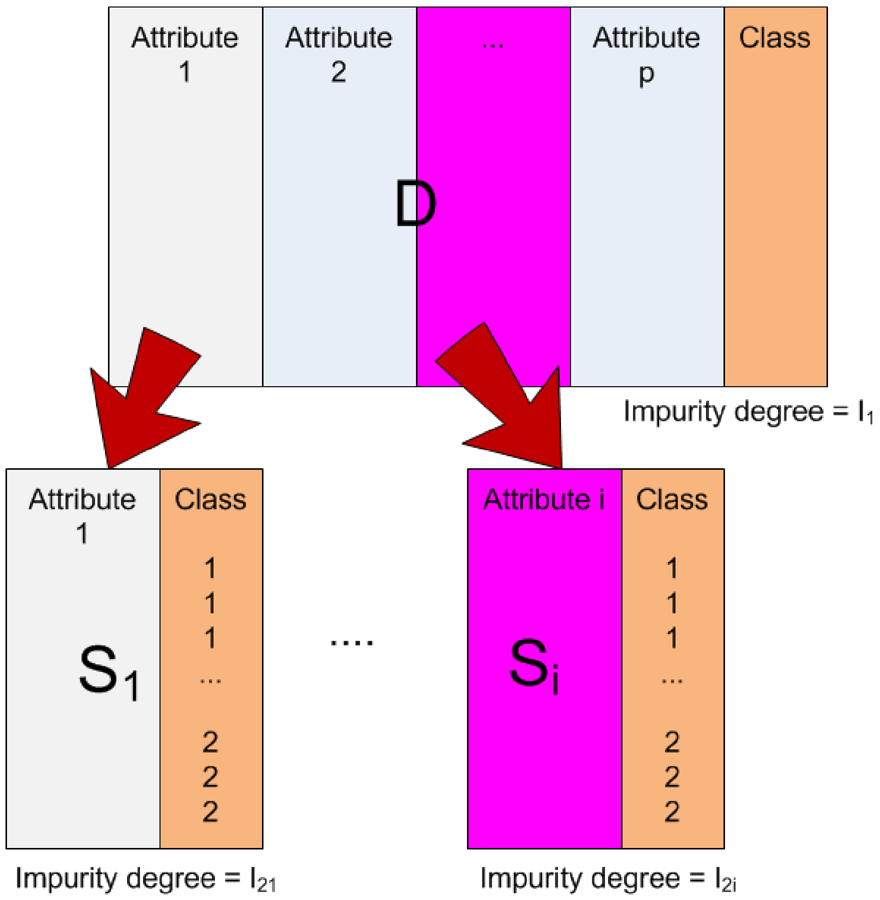
Decision Tree Learning



- Decision tree learning is the construction of a decision tree from class-labeled training tuples. A decision tree is a flow-chart-like structure, where each internal (non-leaf) node denotes a test on an attribute, each branch represents the outcome of a test, and each leaf (or terminal) node holds a class label. The topmost node in a tree is the root node.
- Algorithms for constructing decision trees usually work top-down, by choosing a variable at each step that best splits the set of items.

Top-down induction of decision trees (TDIDT)

- A tree can be “learned” by splitting the source set into subsets based on an attribute value test
- Pick the attribute that creates “purer” subsets (greedy approach!)
- This process is repeated on each derived subset in a recursive manner called recursive partitioning.
- The recursion is completed when the subset at a node has all the same value of the target variable, or when splitting no longer adds value to the predictions



Teknomo Kardi (2009): http://people.revoledu.com/kardi/tutorial/DecisionTree/

Measures of Impurity

- Gini impurity (CART)

To compute Gini impurity for a set of items with J classes, suppose $i \in \{1, 2, \ldots, J\}$, and let p_i be the fraction of items labeled with class i in the set.

$$I_G(p) = \sum_{i=1}^{J} p_i \sum_{k \neq i} p_k = \sum_{i=1}^{J} p_i (1 - p_i) = \sum_{i=1}^{J} (p_i - {p_i}^2) = \sum_{i=1}^{J} p_i - \sum_{i=1}^{J} {p_i}^2 = 1 - \sum_{i=1}^{J} {p_i}^2$$

- $GI(T, a) = I(T) - \sum_{k=1}^{|a|} \frac{|a_k|}{N} I(a_k)$

- Entropy-Based Information Gain (ID, C4.5, C5.0)

$$H(T) = I_E(p_1, p_2, \ldots, p_J) = -\sum_{i=1}^{J} p_i \log_2 p_i$$

$$\overbrace{IG(T, a)}^{\text{Information Gain}} = \overbrace{H(T)}^{\text{Entropy(parent)}} - \overbrace{H(T|a)}^{\text{Weighted Sum of Entropy(Children)}}$$

- $IG(T, a) = H(T) - \sum_{k=1}^{|a|} \frac{|a_k|}{N} H(a_k)$ $\quad$ a_k is the k[th] subset partition

Measuring Impurity

- Variance Reduction (Regression)

The variance reduction of a node N is defined as the total reduction of the variance of the target variable x due to the split at this node:

$$I_V(N) = \frac{1}{|S|^2} \sum_{i \in S} \sum_{j \in S} \frac{1}{2}(x_i - x_j)^2 - \left(\frac{1}{|S_t|^2} \sum_{i \in S_t} \sum_{j \in S_t} \frac{1}{2}(x_i - x_j)^2 + \frac{o1}{|S_f|^2} \sum_{i \in S_f} \sum_{j \in S_f} \frac{1}{2}(x_i - x_j)^2 \right)$$

where S, S_t, and S_f are the set of presplit sample indices, set of sample indices for which the split test is true, and set of sample indices for which the split test is false, respectively.

Data

Attributes				Classes
Gender	Car ownership	Travel Cost ($)/km	Income Level	Transportation mode
Male	0	Cheap	Low	Bus
Male	1	Cheap	Medium	Bus
Female	1	Cheap	Medium	Train
Female	0	Cheap	Low	Bus
Male	1	Cheap	Medium	Bus
Male	0	Standard	Medium	Train
Female	1	Standard	Medium	Train
Female	1	Expensive	High	Car
Male	2	Expensive	Medium	Car
Female	2	Expensive	High	Car

Travel Cost ($)/km	Transportation mode
Cheap	Bus
Cheap	Bus
Cheap	Bus
Cheap	Bus
Cheap	Train
Expensive	Car
Expensive	Car
Expensive	Car
Standard	Train
Standard	Train

4B, 3C, 3T

Entropy	1.571
Gini index	0.660
Classification error	0.600

Travel Cost ($)/km	Classes
Cheap	Bus
Cheap	Bus
Cheap	Bus
Cheap	Bus
Cheap	Train

4B, 1T

Entropy	0.722
Gini index	0.320
classification error	0.200

Travel Cost ($)/km	Classes
Expensive	Car
Expensive	Car
Expensive	Car

3C

Entropy	0.000
Gini index	0.000
classification error	0.000

Travel Cost ($)/km	Classes
Standard	Train
Standard	Train

2T

Entropy	0.000
Gini index	0.000
classification error	0.000

IG = 1.571- {(5/10)0.722+(2/10)0+(3/10)0 = 1.210

Teknomo Kardi (2009): http://people.revoledu.com/kardi/tutorial/DecisionTree/

Choosing the feature

Results of first Iteration

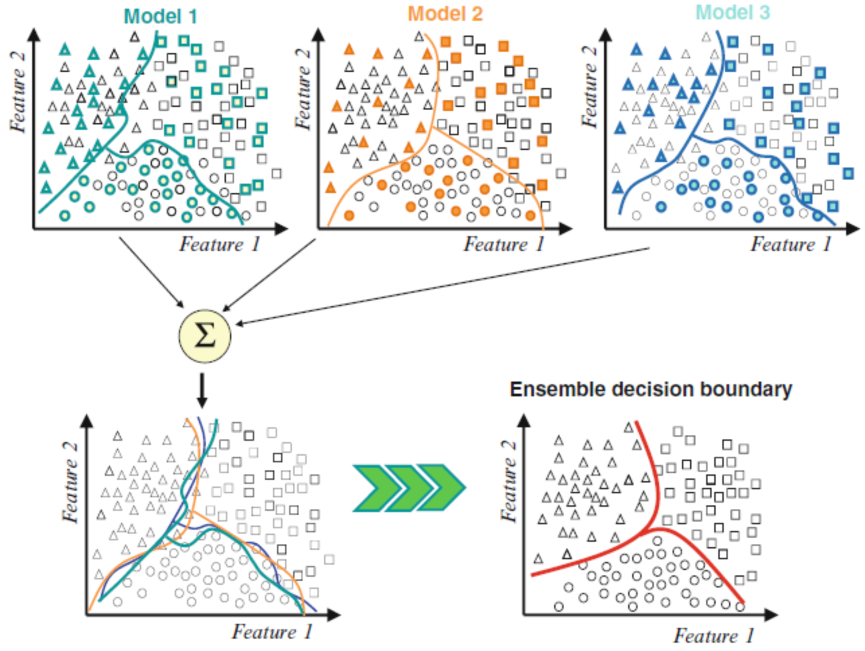
Gain	Gender	Car ownership	Travel Cost/KM	Income Level
Entropy	0.125	0.534	1.210	0.695
Gini index	0.060	0.207	0.500	0.293
Classification error	0.100	0.200	0.500	0.300

- Advantages and Disadvantages

Advantages	Disadvantages
Simple to understand and interpret	Less Accurate
Able to handle both numerical and categorical data	Optimal Decision Tree learning is NP-complete
Requires little data preparation	Sensitive to data changes
Uses a white box model	Can create overly complex boundaries
Possible to validate a model using statistical tests	Impurity metrics can bias results to more levels
Non-statistical approach that makes no assumptions of the training data or prediction residuals	Complexity control through tree depth parameter
Built-in feature selection and interpretation	Practical imlementation needs some “tricks”

Ensemble Methods

- Combine the predictions from multiple “weak” learners
  - Uncorrelated errors in predictions
    - Each learner makes errors on different examples
  - If errors are correlated, little advantage in combining the classifiers
- How to make different classifiers
  - Different Data set partitioning
  - Different Features
  - Different parameters
  - Learning errors from previously trained methods

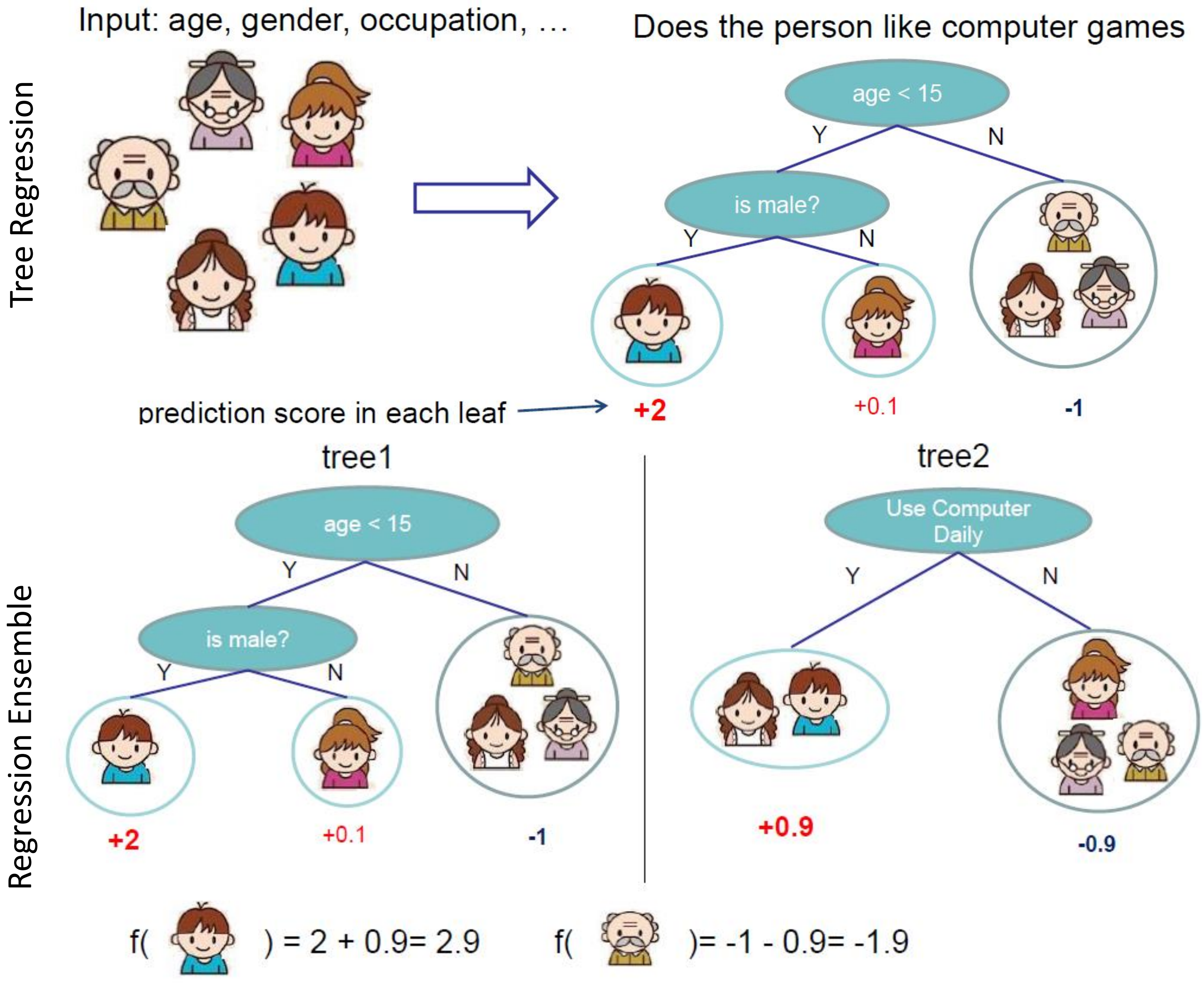


[Polikar 2006: http://users.rowan.edu/~polikar/RESEARCH/PUBLICATIONS/csm06.pdf](http://users.rowan.edu/~polikar/RESEARCH/PUBLICATIONS/csm06.pdf)
Ensemble Machine Learning Methods and Applications (chapter 1), 2012
https://doc.lagout.org/science/Artificial%20Intelligence/Machine%20learning/Ensemble%20Machine%20Learning_%20Methods%20and%20Applications%20%5BZhang%20%26%20Ma%202012-02-17%5D.pdf

Ensemble methods

- Bootstrap Aggregation (Bagging)
  - Involves having each model in the ensemble vote with equal weight.
  - Trains each model in the ensemble using a randomly drawn subset of the training set.
  - Random Forest algorithm
- Boosting
  - Boosting involves incrementally building an ensemble by training each new model instance to emphasize the training instances that previous models mis-classified.
  - Adaboost
  - Gradient Boosted Trees
- Stacking (Stacked Generalization)
  - Build models and then build a model that predicts the output based on the prediction of individual models
- Bayesian Parameter Modeling, Bayesian Model Combination

https://en.wikipedia.org/wiki/Ensemble_learning



Prediction of is sum of scores predicted by each of the tree

XGBoost: A Scalable Tree Boosting System

- An implementation of gradient-boosted trees
- Uses structural risk minimization
- Incrementally builds a machine learning model by combining simple trees
- Very successful in different Kaggle Competitions
- Easy to use

SRM

$$f^* = argmin_f \{L(X,Y;f) + \lambda g(\|f\|)\}$$

X, Y is the training data
f is the learning function

Regularization Classifier Complexity (smoothing) term

Empirical Loss (or risk) term

Structural Risk

- Representation: Output score for a given example is the sum of K tree scores

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), \quad f_k \in \mathcal{F}$$

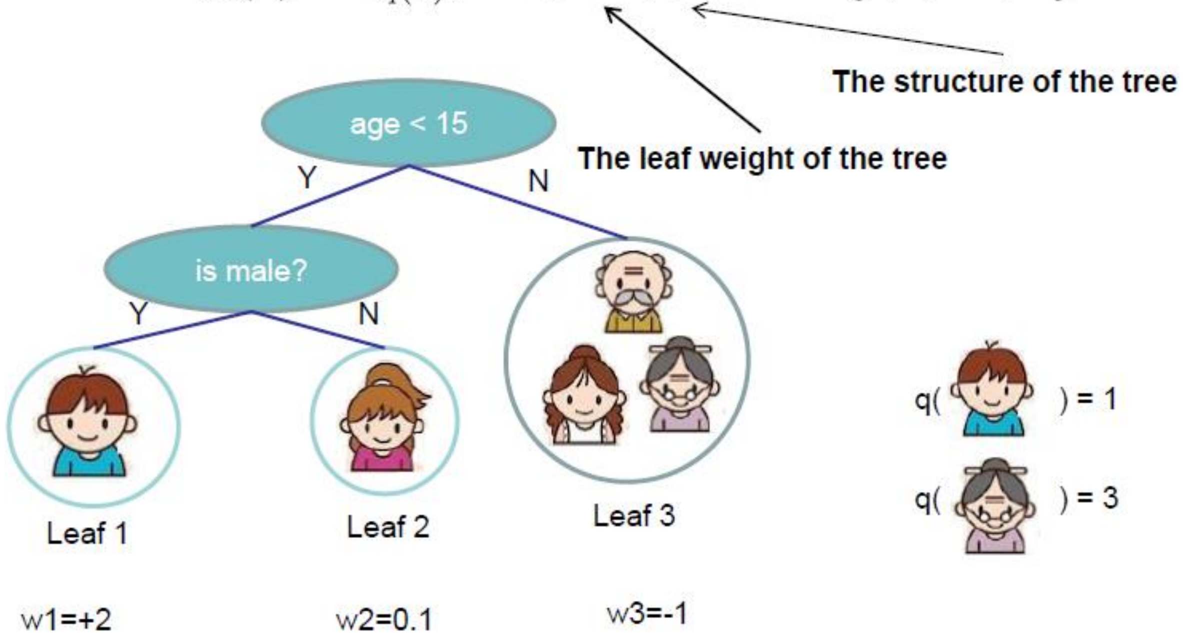
- Loss: Sum of losses over individual examples (regression loss, classification loss, etc.)
- Model Complexity: Number of trees, norm of leaf weights, etc.

$$Obj = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

Structural Risk in Trees: Model Complexity

- Assume a regression tree with T leaves
- We define tree by a vector of scores in leafs, and a leaf index mapping function that maps an instance to a leaf

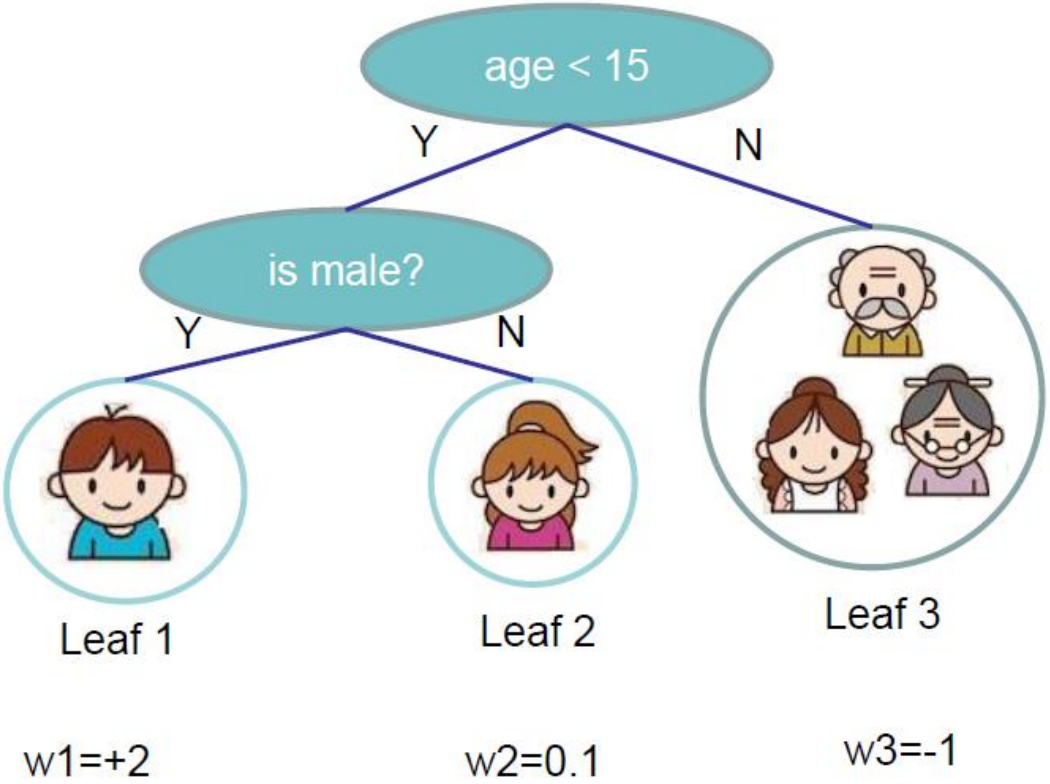
$$f_t(x) = w_{q(x)}, \quad w \in \mathbf{R}^T, q : \mathbf{R}^d \to \{1, 2, \cdots, T\}$$



Structural Risk in Trees : Model Complexity

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

Number of leaves **L2 norm of leaf scores**



$$\Omega = \gamma 3 + \frac{1}{2}\lambda(4 + 0.01 + 1)$$

Representation: Using Additive Boosting

- Start off with a simple predictor
- The next step predictor tries to reduce the error between the prediction of the previous stage and the target by addition

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

Evaluation: Additive Training

- How do we decide which f to add?
  - Optimize the objective!!
- The prediction at round t is $\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$

This is what we need to decide in round t

$$
\begin{aligned}
Obj^{(t)} &= \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t)}) + \sum_{i=1}^{t} \Omega(f_i) \\
&= \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t) + constant
\end{aligned}
$$

Goal: find f_t to minimize this

- Consider square loss

$$
\begin{aligned}
Obj^{(t)} &= \sum_{i=1}^{n} \left(y_i - (\hat{y}_i^{(t-1)} + f_t(x_i))\right)^2 + \Omega(f_t) + const \\
&= \sum_{i=1}^{n} \left[2(\hat{y}_i^{(t-1)} - y_i) f_t(x_i) + f_t(x_i)^2\right] + \Omega(f_t) + const
\end{aligned}
$$

This is usually called residual from previous round

Optimization: Taylor Expansion

- Goal $Obj^{(t)} = \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)\right) + \Omega(f_t) + constant$
  - Seems still complicated except for the case of square loss
- Take Taylor expansion of the objective
  - Recall $f(x + \Delta x) \simeq f(x) + f'(x)\Delta x + \frac{1}{2}f''(x)\Delta x^2$
  - Define $g_i = \partial_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)}), \quad h_i = \partial^2_{\hat{y}^{(t-1)}} l(y_i, \hat{y}^{(t-1)})$

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

- *If you are not comfortable with this, think of square loss*

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

- Compare what we get to previous slide

Optimization

- This gives (notice, g_i, h_i depend only on loss)

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

  - 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 the instance set in leaf j as $I_j = \{i | q(x_i) = j\}$

Regroup the objective by each leaf

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

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

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

Optimization

- We know

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

- Therefore, for our objective function

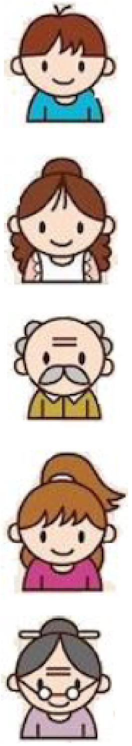
$$\begin{aligned} Obj^{(t)} &= \sum_{j=1}^{T} \left[(\sum_{i \in I_j} g_i) w_j + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) w_j^2 \right] + \gamma T \\ &= \sum_{j=1}^{T} \left[G_j w_j + \frac{1}{2} (H_j + \lambda) w_j^2 \right] + \gamma T \end{aligned}$$

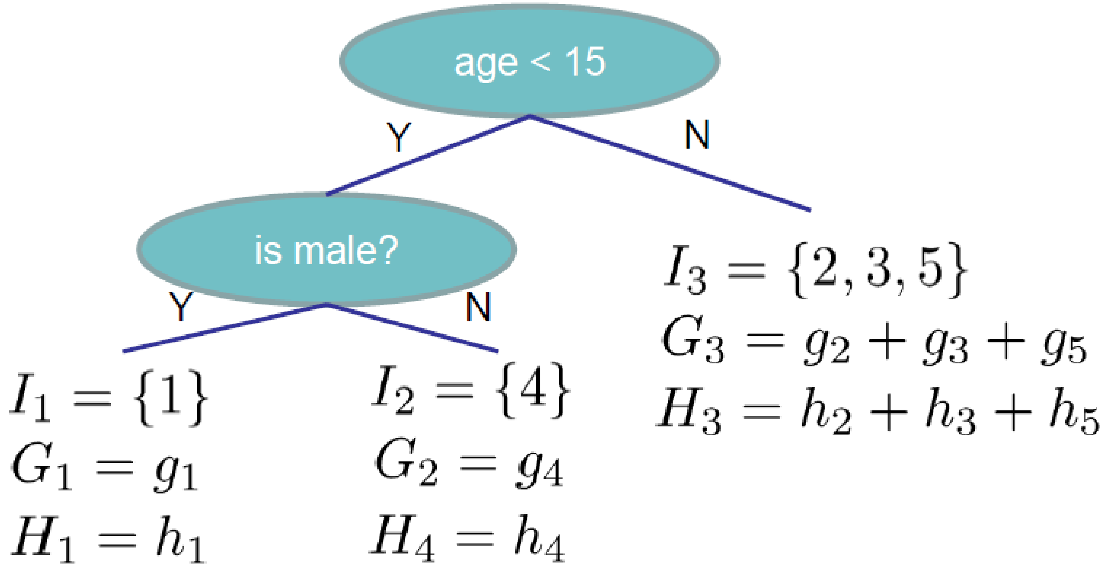
- We get

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

Structural Risk of Trees

Instance index		gradient statistics
1		g1, h1
2		g2, h2
3		g3, h3
4		g4, h4
5		g5, h5





$$Obj = -\sum_j \frac{G_j^2}{H_j+\lambda} + 3\gamma$$

The smaller the score is, the better the structure is

Searching Algorithm for Single Tree

- Enumerate the possible tree structures q
- Calculate the structure score for the q, using the scoring eq.

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

- Find the best tree structure, and use the optimal leaf weight

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

- But... there can be infinite possible tree structures..

Greedy Split: Information Gain

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

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

The complexity cost by introducing additional leaf

the score of left child

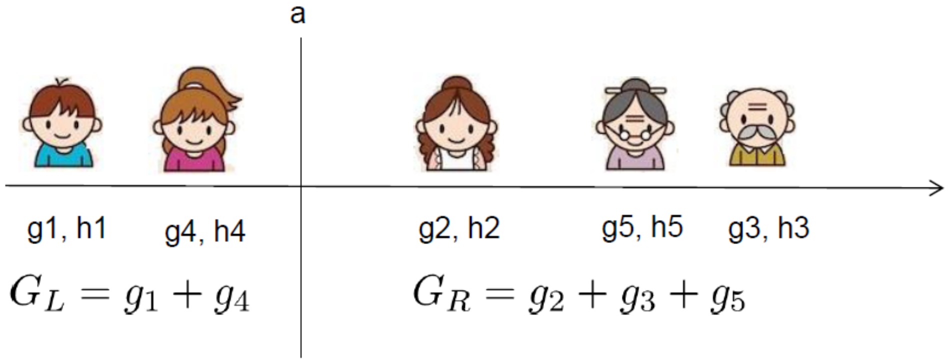
the score of right child

the score of if we do not split

  - Remaining question: how do we find the best split?

Greedy Splitting

- What is the gain of a split rule $x_j < a$? Say x_j is age



- All we need is sum of g and h in each side, and calculate

$$Gain = \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{(G_L+G_R)^2}{H_L+H_R+\lambda} - \gamma$$

- Left to right linear scan over sorted instance is enough to decide the best split along the feature

- **Algorithm**
- For each node, enumerate over all features
  - For each feature, sorted 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

Boosted Tree Algorithm

- Add a new tree in each iteration
- Beginning of each iteration, calculate

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

- Use the statistics to greedily grow a tree $f_t(x)$

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

- Add $f_t(x)$ to the model $\hat{y}_i^{(t)} = \hat{y}_i^{(t-1)} + f_t(x_i)$
  - Usually, instead we do $y^{(t)} = y^{(t-1)} + \epsilon f_t(x_i)$
  - ϵ is called step-size or shrinkage, usually set around 0.1
  - This means we do not do full optimization in each step and reserve chance for future rounds, it helps prevent overfitting

Code

```
import pandas as pd
import numpy as np
import xgboost as xgb

train = pd.read_csv("../input/train.csv")
test = pd.read_csv("../input/test.csv")
submission = pd.read_csv("../input/sampleSubmission.csv")
#target is class_1, ..., class_9 - needs to be converted to 0, ..., 8
train['target'] = train['target'].apply(lambda val: np.int64(val[-1:]))-1

Xy_train = train.as_matrix()
X_train = Xy_train[:,1:-1]
y_train = Xy_train[:,-1:].ravel()

X_test = test.as_matrix()[:,1:]

dtrain = xgb.DMatrix(X_train, y_train, missing=np.NaN)
dtest = xgb.DMatrix(X_test, missing=np.NaN)

params = {"objective": "multi:softprob", "eval_metric": "mlogloss", "booster" : "gbtree",
          "eta": 0.05, "max_depth": 3, "subsample": 0.6, "colsample_bytree": 0.7, "num_class": 9}

num_boost_round = 100

gbm = xgb.train(params, dtrain, num_boost_round)
pred = gbm.predict(dtest)

print(gbm.eval(dtrain))
```

to account for examples importance we can assign weights to them in DMatrix (not done here)

Feature Importance

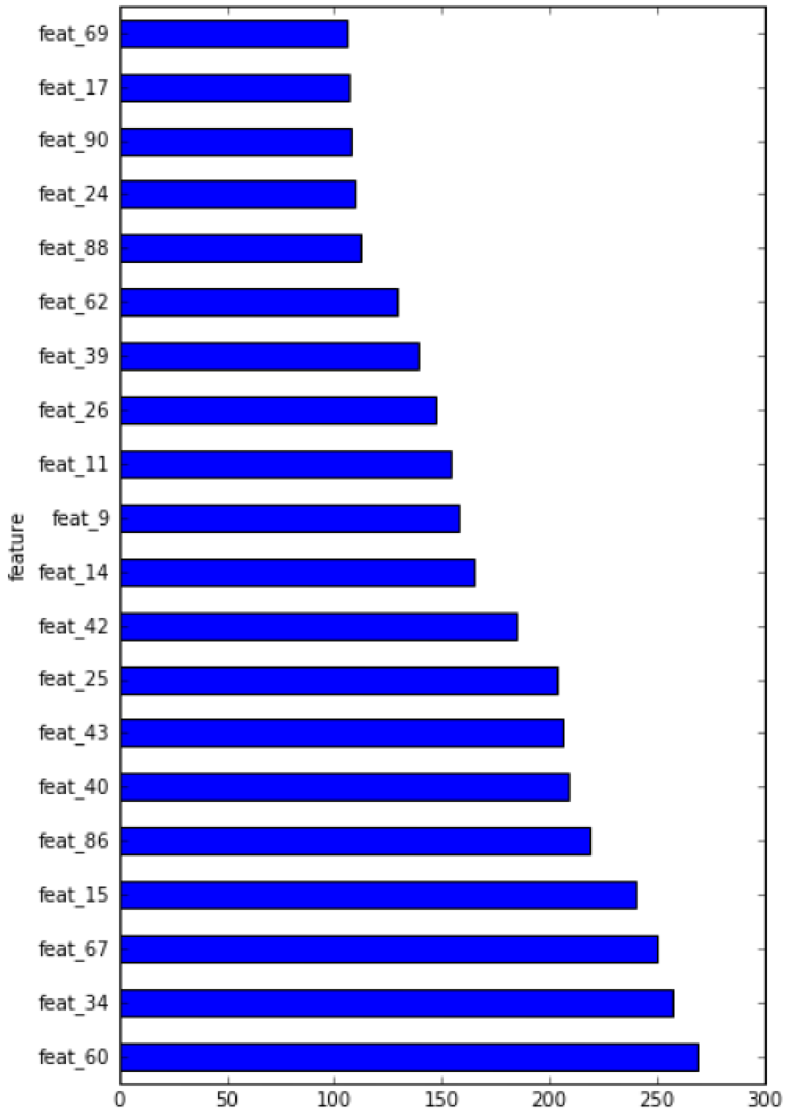
```
importance = gbm.get_fscore()

fdict = {}
for key, name in enumerate(train.columns[1:-1]):
    fdict['f{0}'.format(key)] = name

importance_with_names = []

for key, value in importance.items():
    importance_with_names.append((fdict[key], value))

pd.DataFrame(importance_with_names, columns=['feature',
'fscore']).\
set_index('feature').sort_values(['fscore'],
ascending=[0])[:20].\
plot(kind="barh", legend=False, figsize=(6, 10))
```



XGBoost via Scikit

```
import pandas as pd
import numpy as np
import xgboost as xgb
from sklearn.metrics import log_loss

train = pd.read_csv("../input/train.csv")
test = pd.read_csv("../input/test.csv")
submission = pd.read_csv("../input/sampleSubmission.csv")
#target is class_1, ..., class_9 - needs to be converted to 0, ..., 8
train['target'] = train['target'].apply(lambda val: np.int64(val[-1:]))-1

Xy_train = train.as_matrix()
X_train = Xy_train[:,1:-1]
y_train = Xy_train[:,-1:].ravel()

X_test = test.as_matrix()[:,1:]

num_boost_round = 100

gbm = xgb.XGBClassifier(max_depth=3, learning_rate=0.05, objective="multi:softprob", subsample=0.6,
                  colsample_bytree=0.7, n_estimators=num_boost_round)

gbm = gbm.fit(X_train, y_train)

pred = gbm.predict_proba(X_test)

y_hat_train = gbm.predict_proba(X_train)
print(log_loss(y_train, y_hat_train))
```

XGBoost Parameters

subsample

- ratio of instances to take
- example values: 0.6, 0.9

colsample_by_tree

- ratio of columns used for whole tree creation
- example values: 0.1, …, 0.9

colsample_by_level

- ratio of columns sampled for each split
- example values: 0.1, …, 0.9

eta

- how fast algorithm will learn (shrinkage)
- example values: 0.01, 0.05, 0.1

max_depth

- maximum numer of consecutive splits
- example values: 1, …, 15 (for dozen or so or more, needs to be set with regularization parametrs)

min_child_weight

- minimum weight of children in leaf, needs to be adopted for each measure
- example values: 1 (for linear regression it would be one example, for classification it is gradient of pseudo-residual)

alpha

- L1 norm (simple average) of weights for whole objective function

lambda

- L2 norm (root from average of squares) of weights, added as penalty to objective function

gamma

- L0 norm, multiplied by numer of leafs in a tree is used to decide whether to make a split

SHAP: A unified approach to explain the output of any machine learning model

```
# https://github.com/slundberg/shap
import xgboost
import shap
# load JS visualization code to notebook
shap.initjs()

# train XGBoost model
X,y = shap.datasets.boston()
model = xgboost.train({"learning_rate": 0.01},
xgboost.DMatrix(X, label=y), 100)

# explain the model's predictions using SHAP values
# (same syntax works for LightGBM, CatBoost, and scikit-
learn models)
shap_values = shap.TreeExplainer(model).shap_values(X)
# visualize the first prediction's explanation
shap.force_plot(shap_values[0,:], X.iloc[0,:])
```

References

- **XGBoost: A Scalable Tree Boosting System**
- https://www.slideshare.net/JaroslawSzymczak1/xgboost-the-algorithm-that-wins-every-competition
  - Especially the feature importance
- https://homes.cs.washington.edu/~tqchen/pdf/BoostedTree.pdf

End of Lecture

We want to make a machine that will be proud of us.

- Danny Hillis