Machine Learning 520
Advanced Machine Learning

Lesson 3: Random Forest and Gradient Boosted Trees

UNIVERSITY of WASHINGTON

Today's Agenda

- Decision/Regression tree bagging
- Random forest regression/classification
- Gradient boosting regression/classification

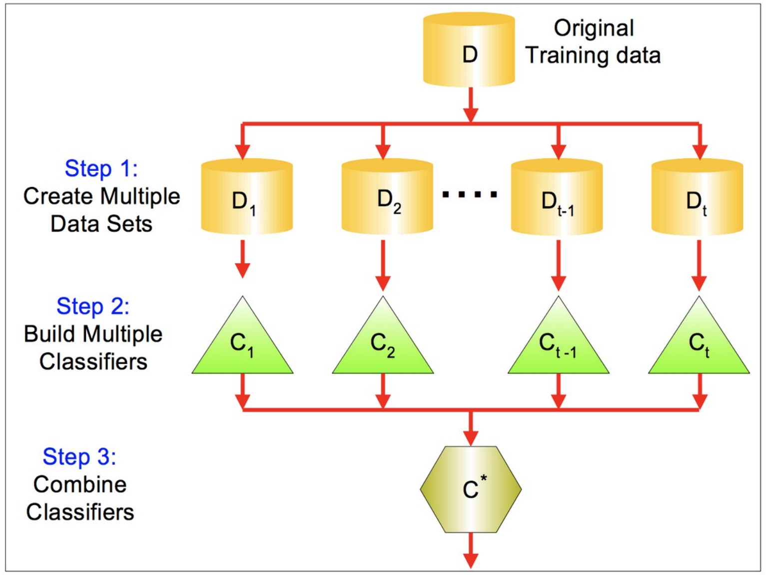
Learning Objectives

By the end of this session, you should be able to:

- Describe bootstrap and why bootstrap helps.
- Demonstrate how bootstrap aggregation works for classification/regression trees.
- Apply random forest and gradient boosting trees to data sets and evaluate performance.

Bagging

- **Bootstrap aggregation**, or **bagging**, is a general-purpose procedure for **reducing the variance** of a machine learning method by combining the result of multiple classifiers trained on different sub-samples of the same data set.
  - E.g. random forest.
- **Bagging:**
  - Step 1: Create bootstrap samples (sample with replacement).
  - Step 2: Train separate classifier on each sample.
  - Step 3: Classify new data point by majority vote or average.



Random Forest (Decision Forests)

Ensemble of multiple independently trained decision trees

- Each tree is trained using a sample of observations and a sample of independent variables
  - Think about three doctors diagnosing heart disease. One doctor is trained by just looking at ECG, one doctor is a Chinese medicine doctor who is trained only by only touching the pulse, and one doctor is trained by looking at the ultrasound image
  - Each doctor is trained on data of different patients (there might be overlapping among the sets of patients)

Random Forest (Decision Forests)

Advantages of Random Forest:

- Significantly better performance than individual trees
- Automatic Feature Selection
- Less risk of overfitting
- Can be parallelized easily (training of multiple doctors can happen at the same time independently)

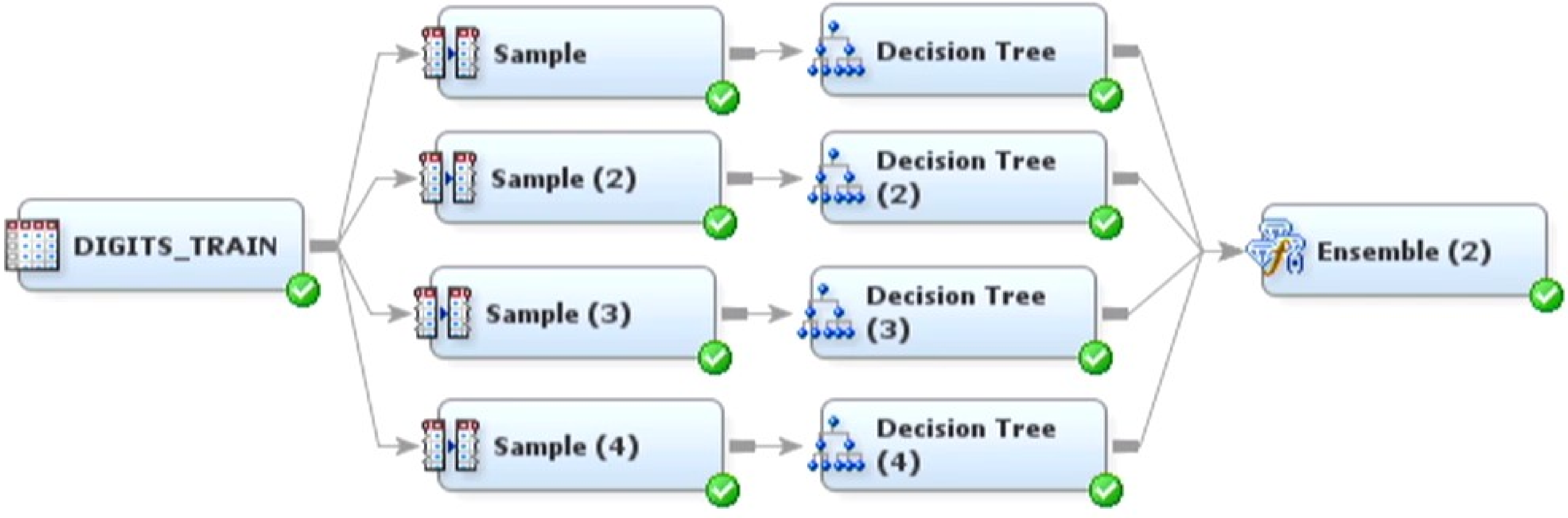
Disadvantages:

- Less interpretability than decision trees
- In some algorithms, data is copied in order to train each tree. Has higher requirement in memory space than individual trees.

Random Forest (Decision Forests)

- Combination of decision trees and bagging concepts
- A large number of decision trees is trained, each on a different bagging sample
- At each split, only a random number of the original variables is available (i.e. small selection of columns)
- Data points are classified by majority voting of the individual trees

Random Forest (Decision Forests)

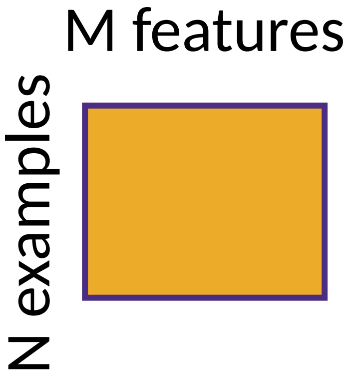


Random Forest (Decision Forests)

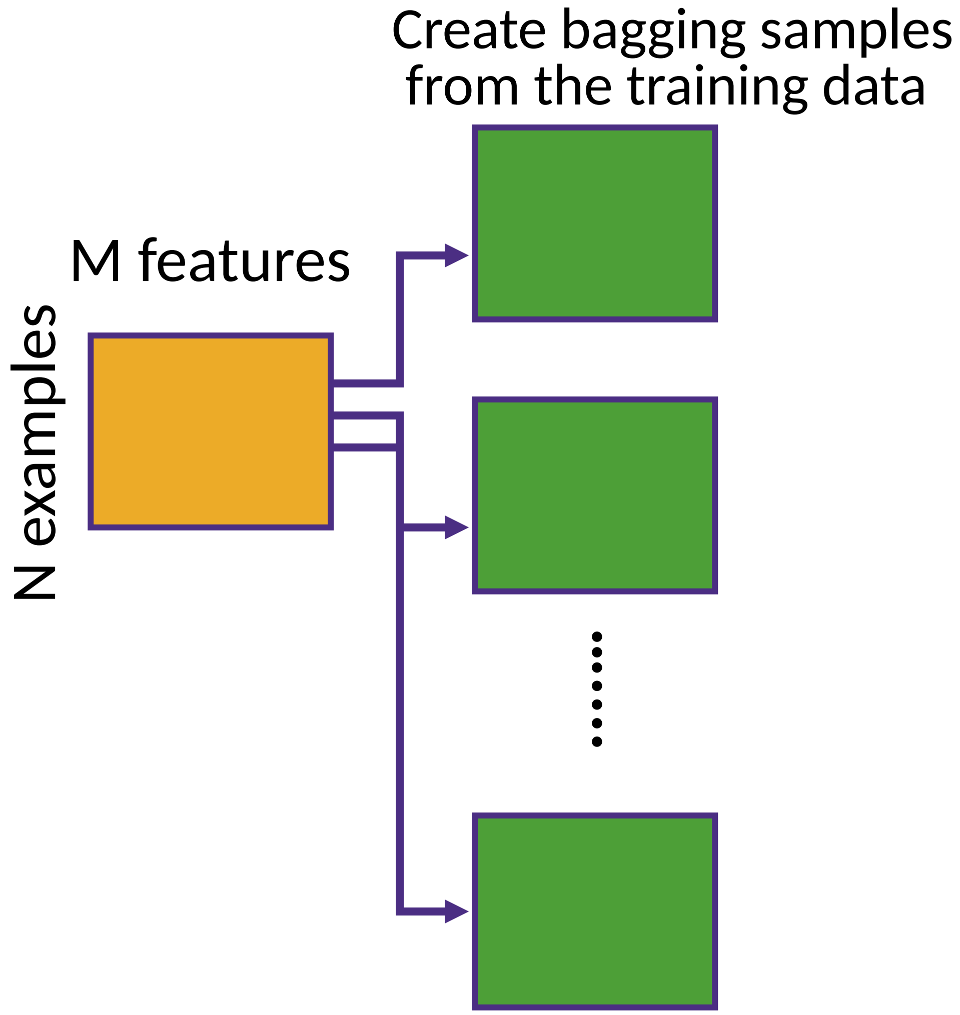
```
D = training set                          F = set of tests
k = nb of trees in forest                 n = nb of tests

for i = 1 to k do:
   build data set Di by sampling with replacement from D
   learn tree Ti (Tilde) from Di:
      at each node:
         choose best split from random subset of F of size n
      allow aggregates and refinement of aggregates in tests

make predictions according to majority vote of the set of k trees.
```

Random Forest Classifier

Training Data

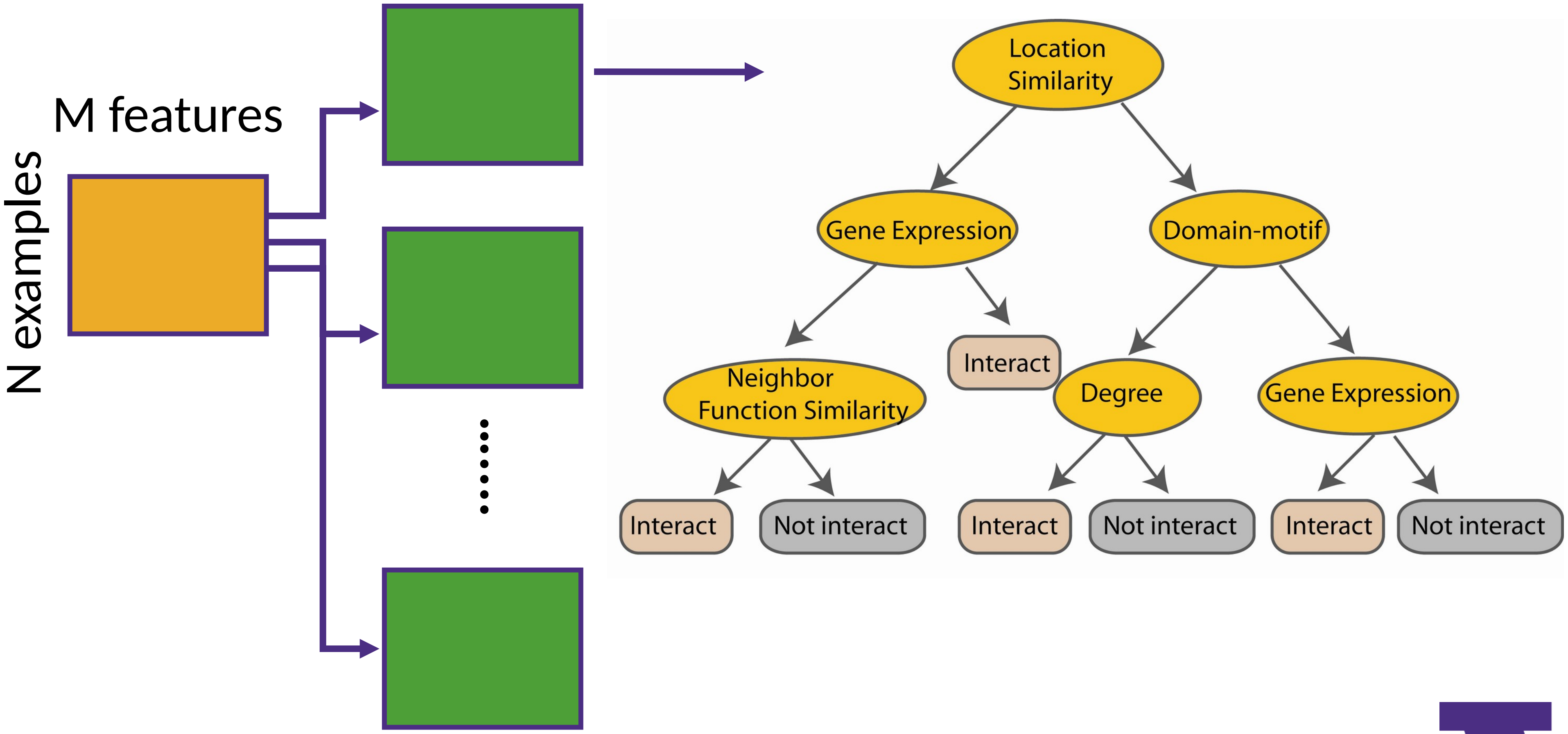


Random Forest Classifier



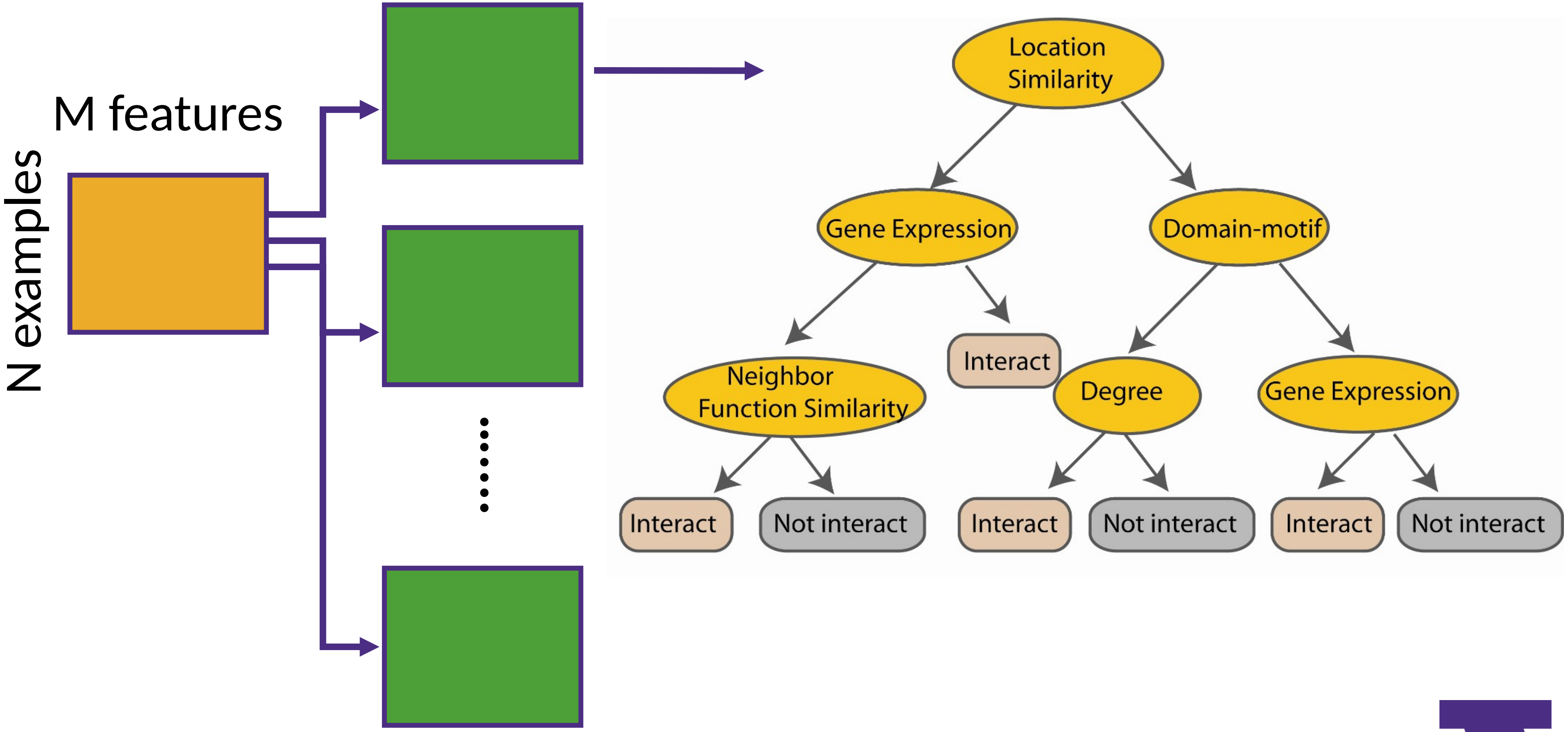
Random Forest Classifier

Construct a decision tree



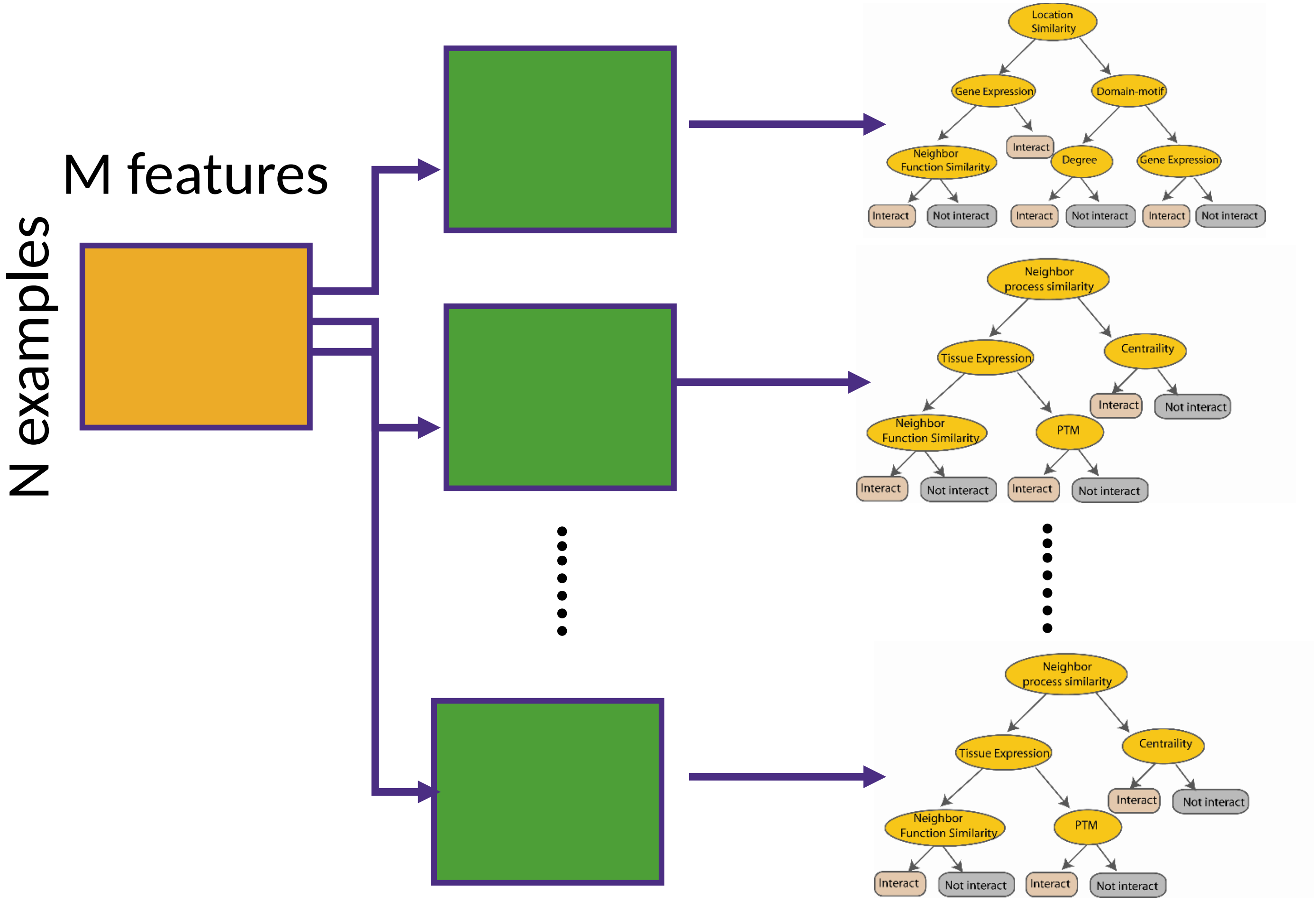
Random Forest Classifier

At each node in choosing the split feature
choose only among *m*<*M* features

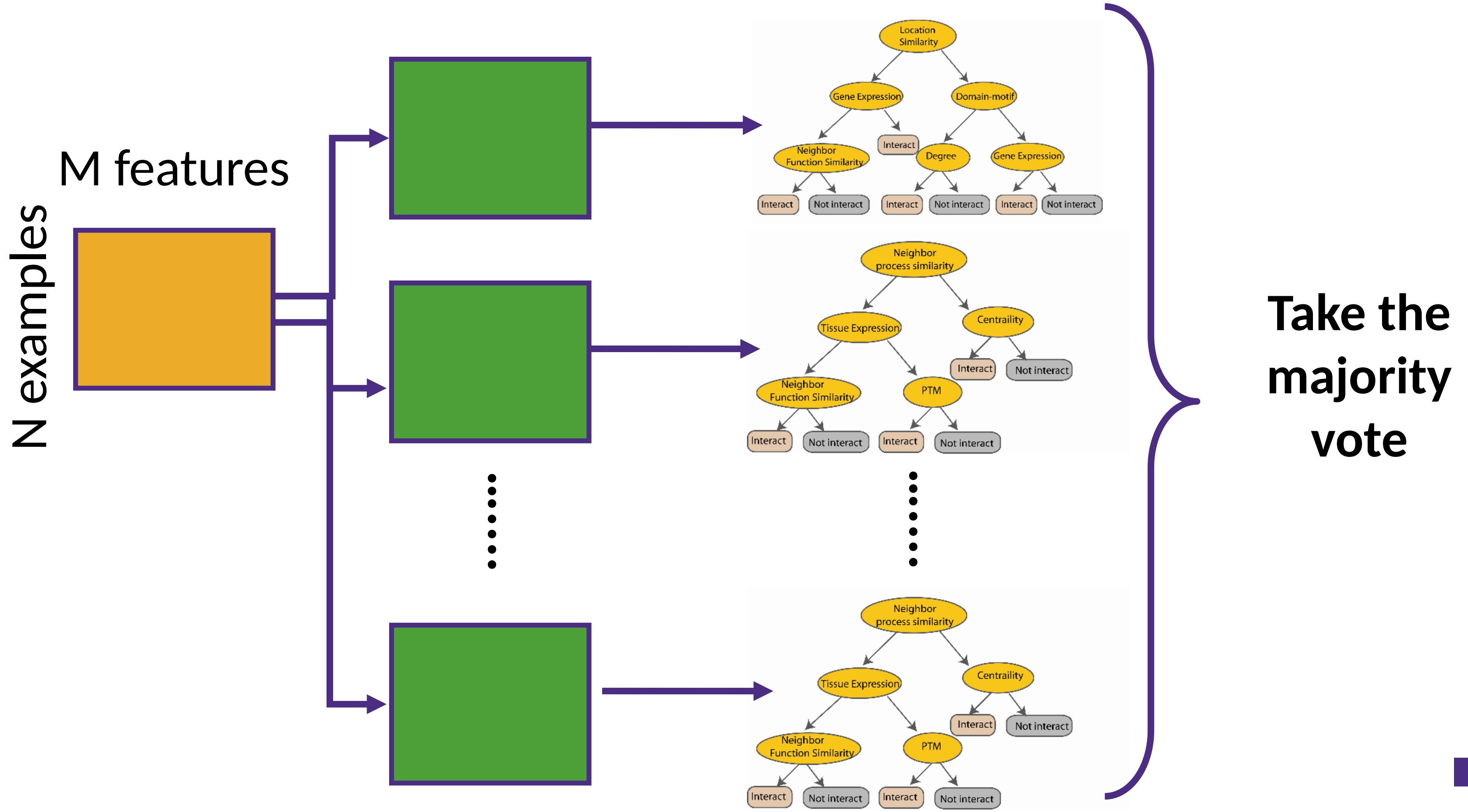


Random Forest Classifier

Create decision tree from each bootstrap sample



Random Forest Classifier



Random Forest options in Scikit-learn

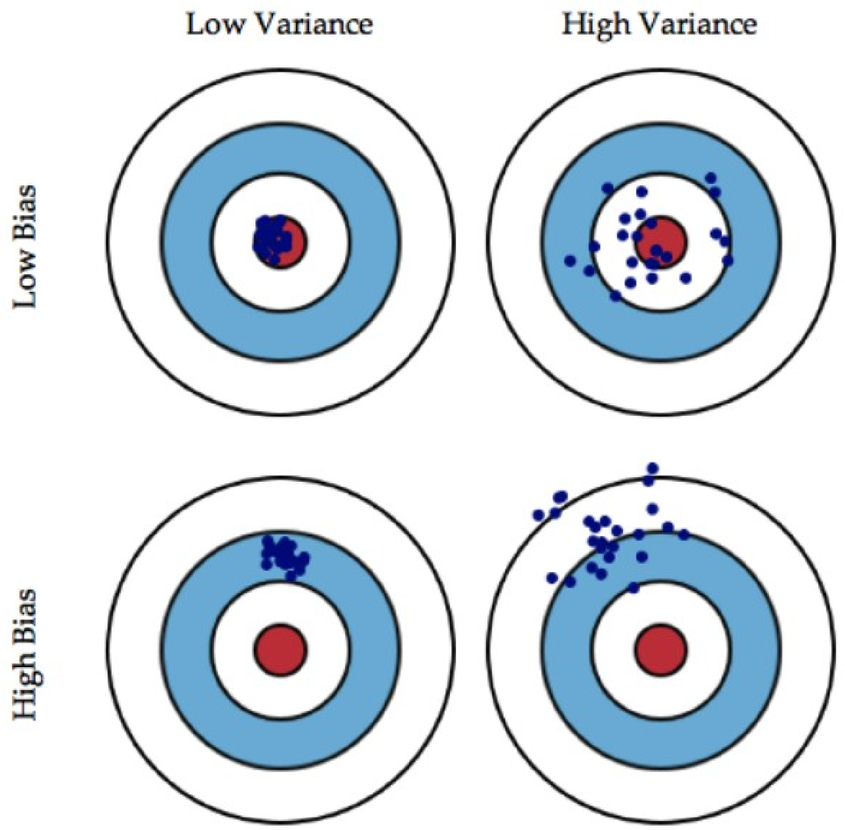
Parameter	Description
n_estimators	number of tree
criterion	"gini" or "entropy"
max_features	The number of features to consider when looking for the best split
max_depth	The maximum depth of the tree
min_samples_split	The minimum number of samples required to split an internal node
min_samples_leaf	The minimum number of samples required to be at a leaf node
min_weight_fraction_leaf	The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node.
max_leaf_nodes	Grow trees with max_leaf_nodes in best-first fashion.
min_impurity_split	Threshold for early stopping in tree growth.
bootstrap	Whether bootstrap samples are used when building trees.
oob_score	Whether to use out-of-bag samples to estimate the generalization accuracy.
warm_start	When set to True, reuse the solution of the previous call to fit and add more estimators to the ensemble, otherwise, just fit a whole new forest.

Bias/Variance Tradeoff

ERORR = BIAS + VARIANCE + NOISE

- **Bias**: the difference between the predicted value and the actual value.
  - Model underfits the training data and fails to capture the underlying pattern within the data.
  - e.g. Linear model

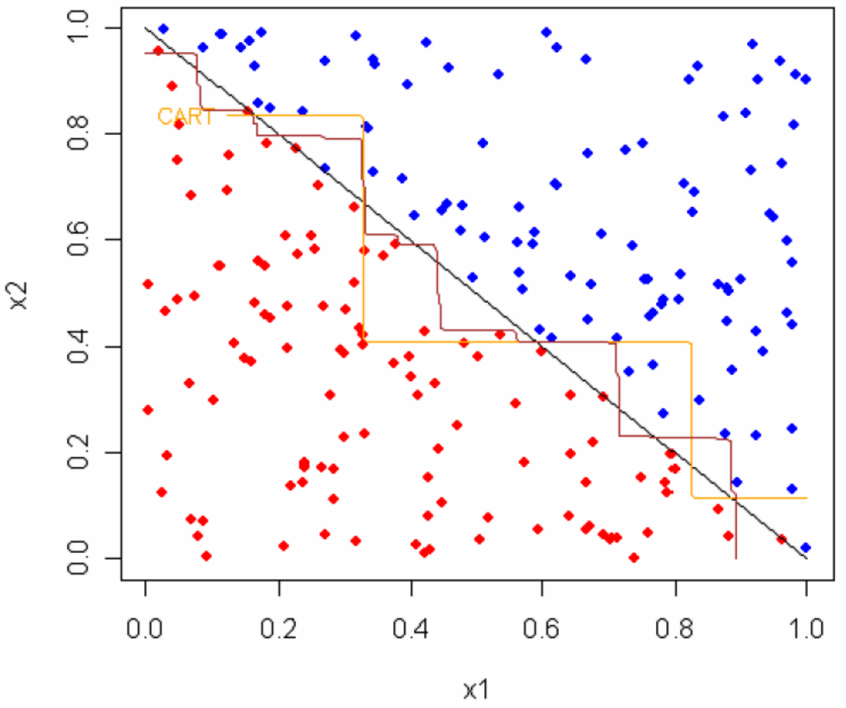
- **Variance**: the variability of a model prediction for a given data point.
  - Learning is not stable. A small change in the training data or hyper-parameter can lead to a very different model/predictoin.
  - E.g. decision tree



Bagging: reduces variance – Example 1

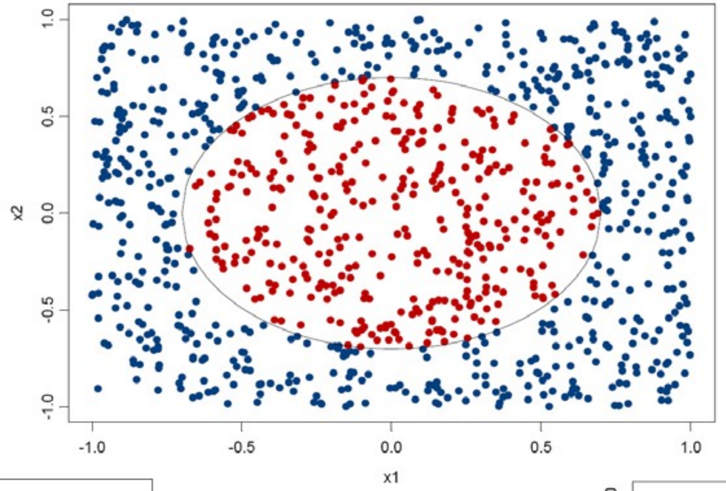
- Two categories of samples: blue, red
- Two predictors: x1 and x2

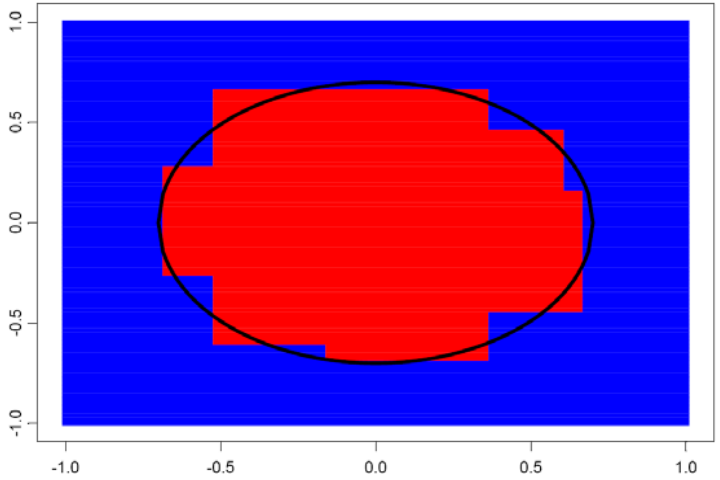
  Diagonal separation…hardest case for tree-based cla
  - Single tree decision boundary in orange.
  - Bagged predictor decision boundary in red.



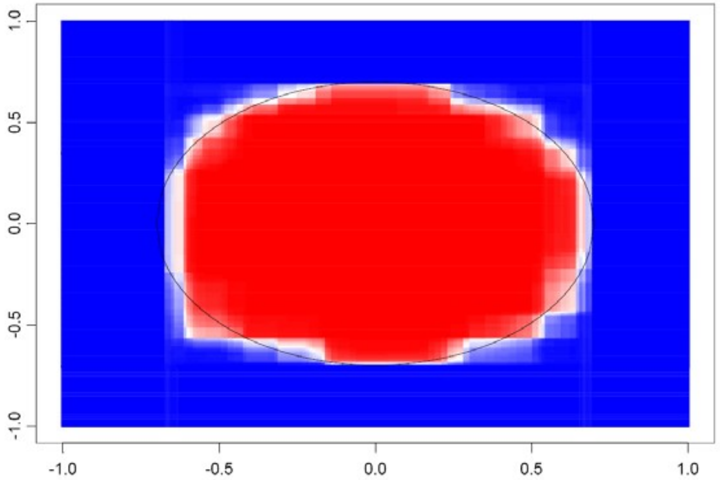
Bagging: reduces variance – Example 2

Ellipsoid separation
Two categories,
Two predictors





Single tree decision boundary



100 bagged trees..

Why does bagging work?

> Bagging reduces **variance** by averaging the predictions from multiple classifiers.

$$Var(\overline{X}) = \frac{Var(X)}{N}$$

(when prediction are **independent**)

> Bagging has little effect on **bias**.

> Can we average and reduce both **bias** and **variance**?

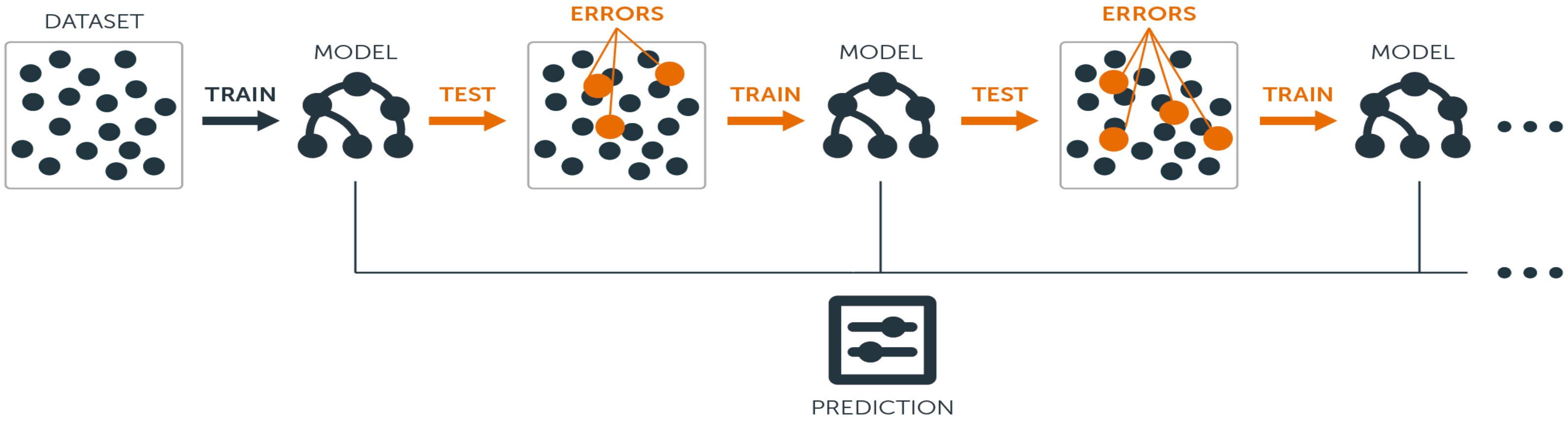
Yes: Boosting

Boosting

Main Idea of Boosting

> **Learn classifiers in sequence**: later classifiers focus on examples that were misclassified by earlier classifiers.

> **Weighted voting**: weight the predictions of the classifiers according their prediction accuracy.



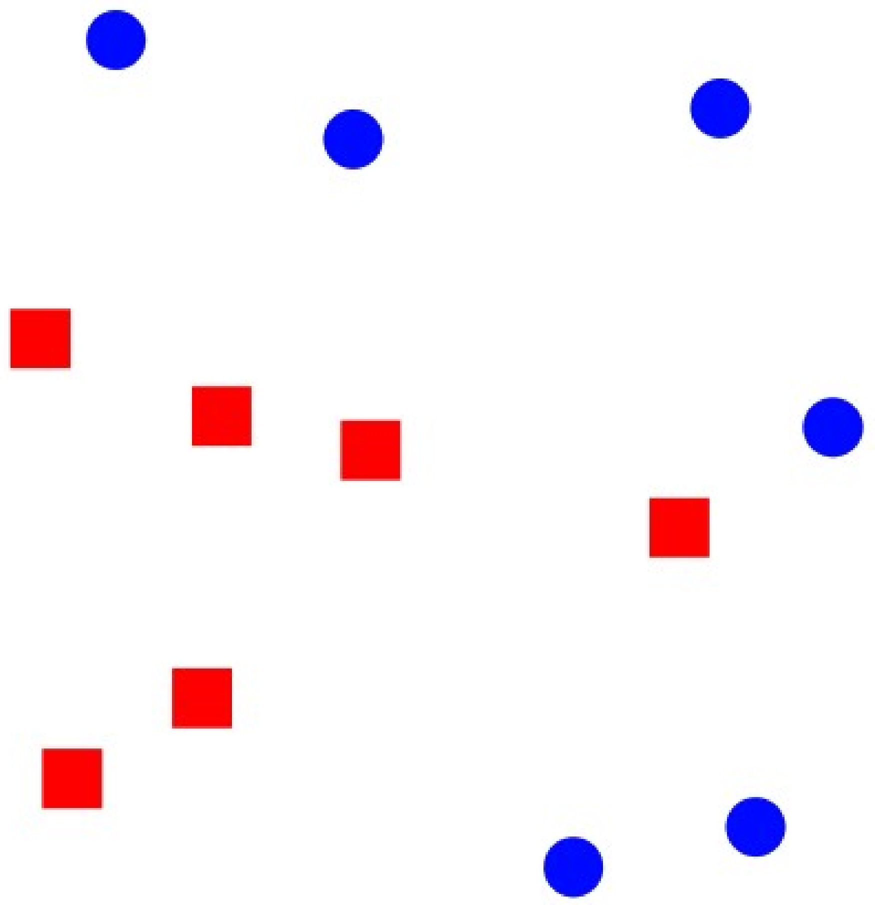
Boosting Realization

- On each iteration t:
  - Weight each training example by how incorrectly it was classified by previous classifiers.
    - increasing the weight of incorrectly classified examples ensures that they will become more important in the next iteration.
  - Learn a classifier $h_t(x)$ based on the weighted training data.
  - Calculate a strength factor α_t for $h_t(x)$ based on its accuracy.

- Final classifier:
  - Weighted voting of different classifiers where weight is their strength factor.

Boosting Intuition

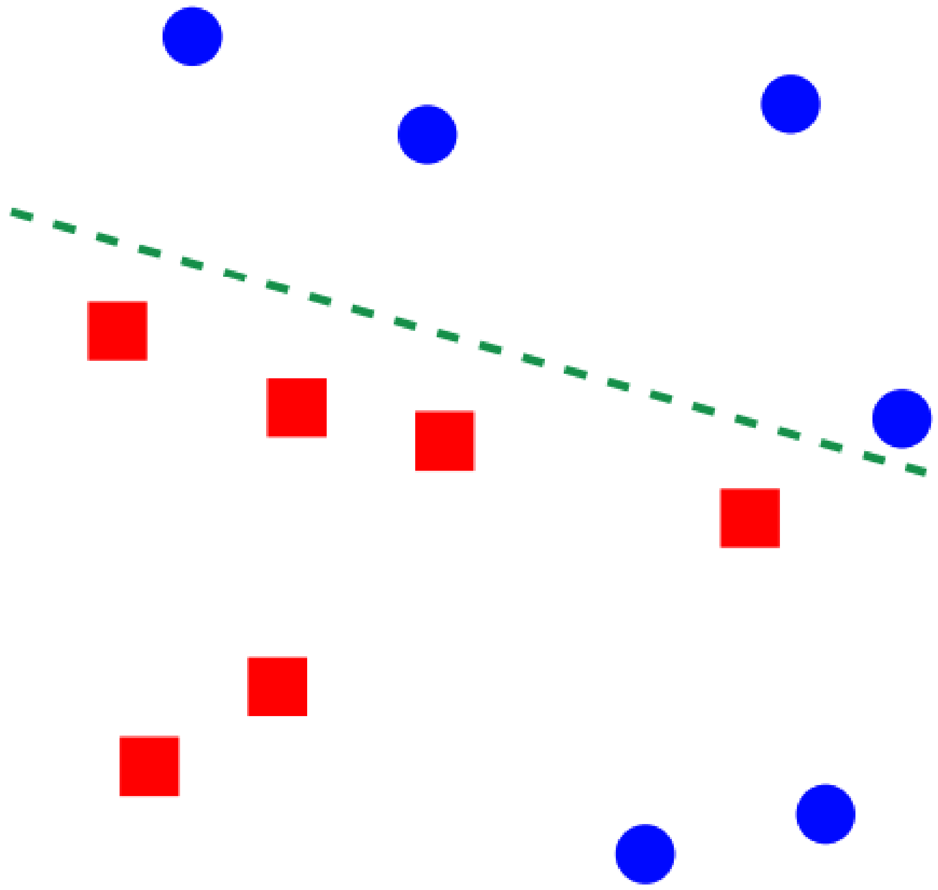
Goal: turn weak learners (e.g. linear model) into a strong learner.



- Pick a weak linear classifier $h_t(x)$.
- Adjust weights: misclassified examples get heavier weight.
- Calculate strength α_t according to the weighted error of $h_t(x)$.

Boosting Intuition

Goal: turn weak learners (e.g. linear model) into a strong learner.



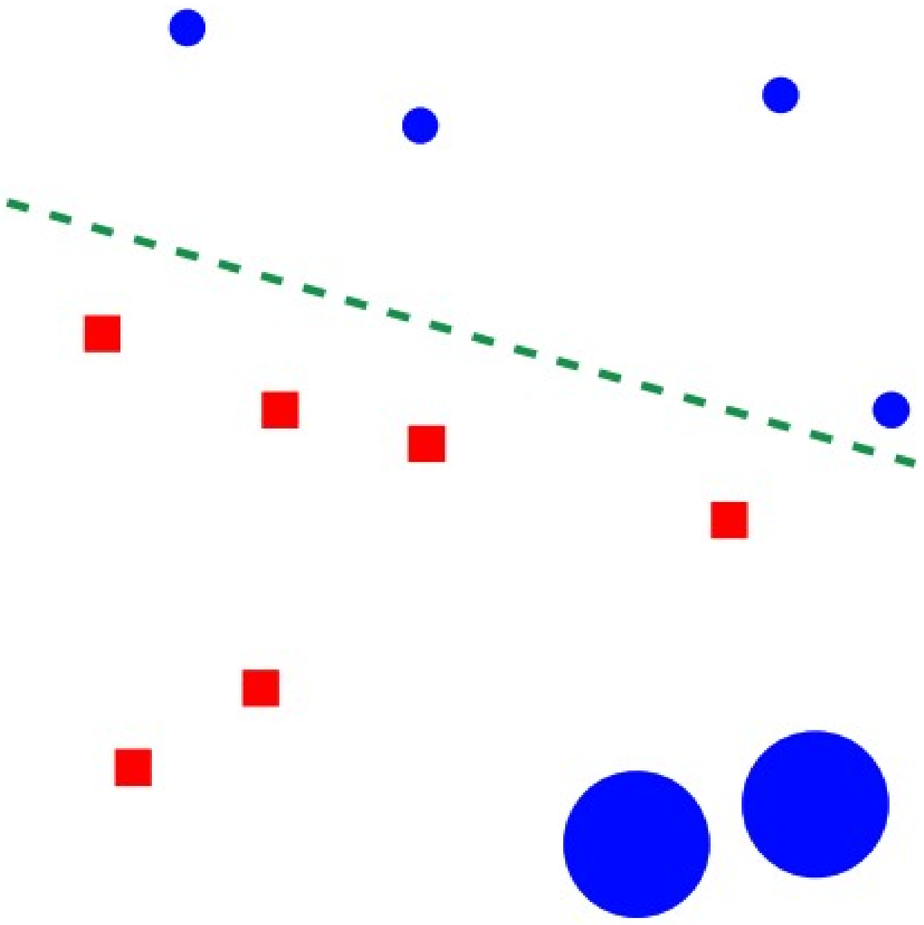
> Pick a weak linear classifier $h_t(x)$.

> Adjust weights: misclassified examples get heavier weight.

> Calculate strength α_t according to the weighted error of $h_t(x)$.

Boosting Intuition

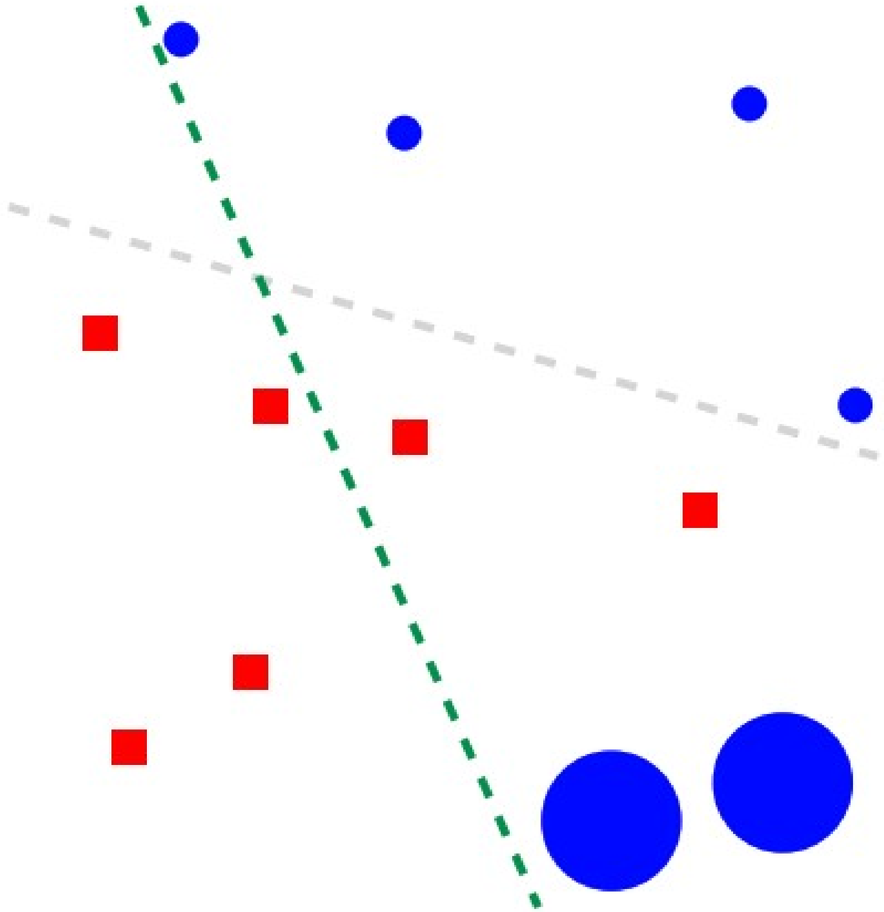
Goal: turn weak learners (e.g. linear model) into a strong learner.



- Pick a weak linear classifier $h_t(x)$.
- Adjust weights: misclassified examples get heavier weight.
- Calculate strength α_t according to the weighted error of $h_t(x)$.

Boosting Intuition

Goal: turn weak learners (e.g. linear model) into a strong learner.



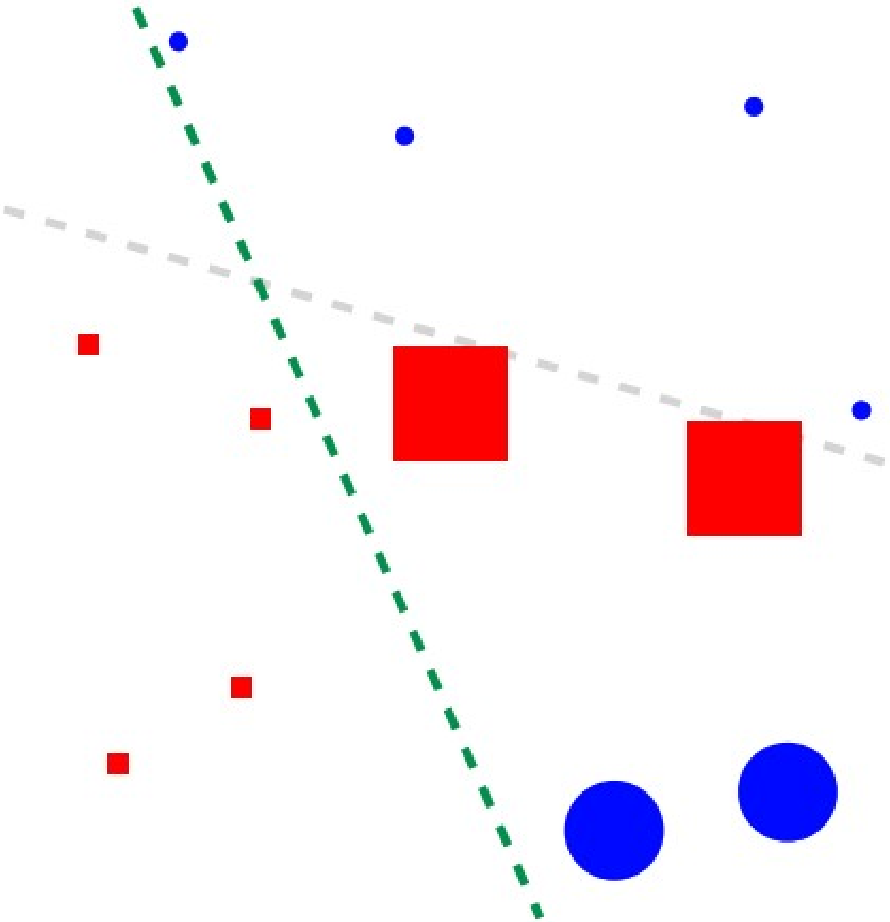
> Pick a weak linear classifier $h_t(x)$.

> Adjust weights: misclassified examples get heavier weight.

> Calculate strength α_t according to the weighted error of $h_t(x)$.

Boosting Intuition

Goal: turn weak learners (e.g. linear model) into a strong learner.



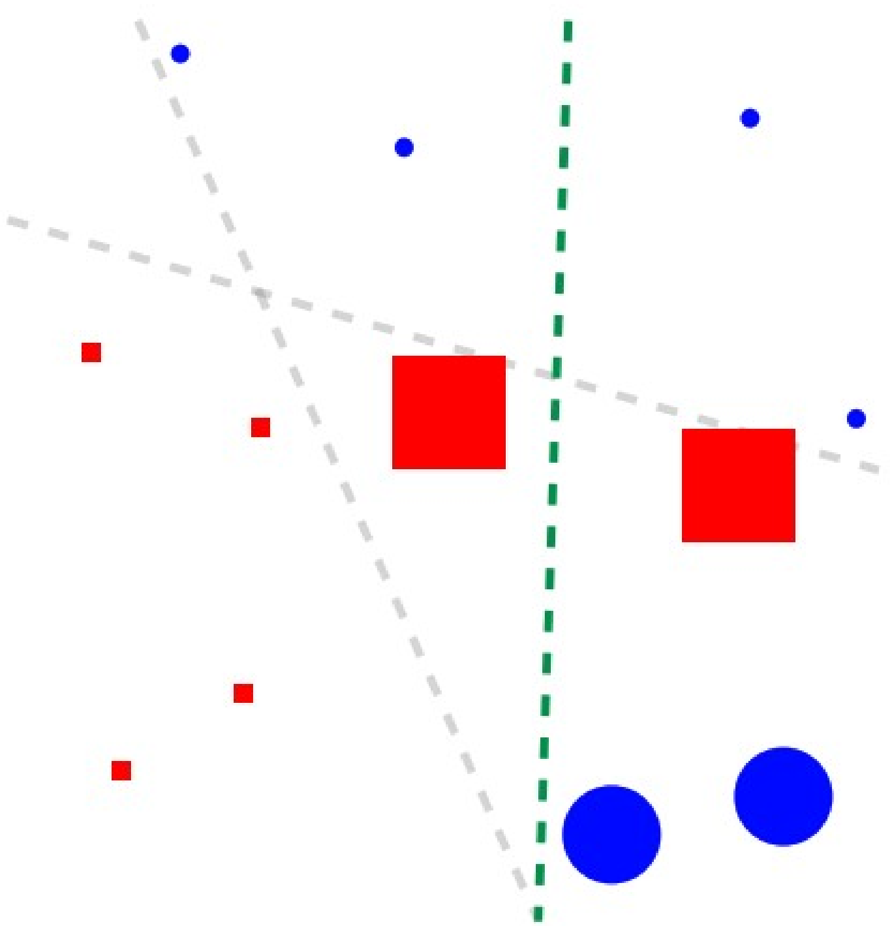
> Pick a weak linear classifier $h_t(x)$.

> Adjust weights: misclassified examples get heavier weight.

> Calculate strength α_t according to the weighted error of $h_t(x)$.

Boosting Intuition

Goal: turn weak learners (e.g. linear model) into a strong learner.



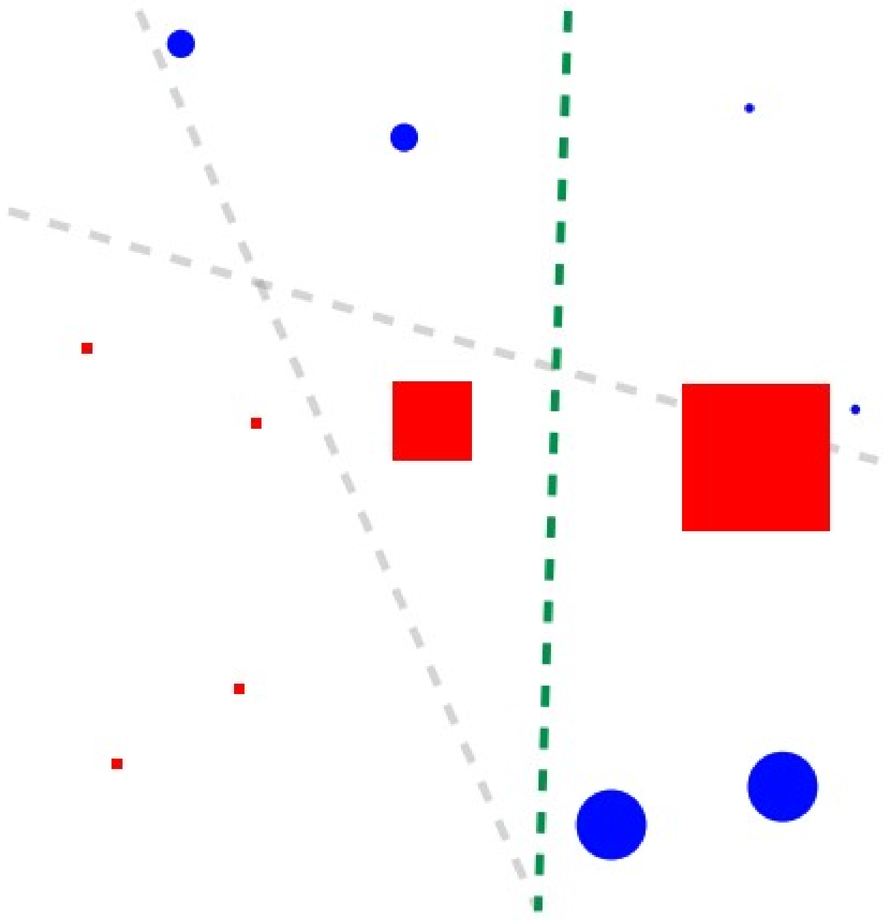
> Pick a weak linear classifier $h_t(x)$.

> Adjust weights: misclassified examples get heavier weight.

> Calculate strength α_t according to the weighted error of $h_t(x)$.

Boosting Intuition

Goal: turn weak learners (e.g. linear model) into a strong learner.



- Pick a weak linear classifier $h_t(x)$.
- Adjust weights: misclassified examples get heavier weight.
- Calculate strength α_t according to the weighted error of $h_t(x)$.

Boosting history

[Schapire '89]:

- first provable boosting algorithm

[Freund '90]:

- "optimal" algorithm that "boosts by majority"

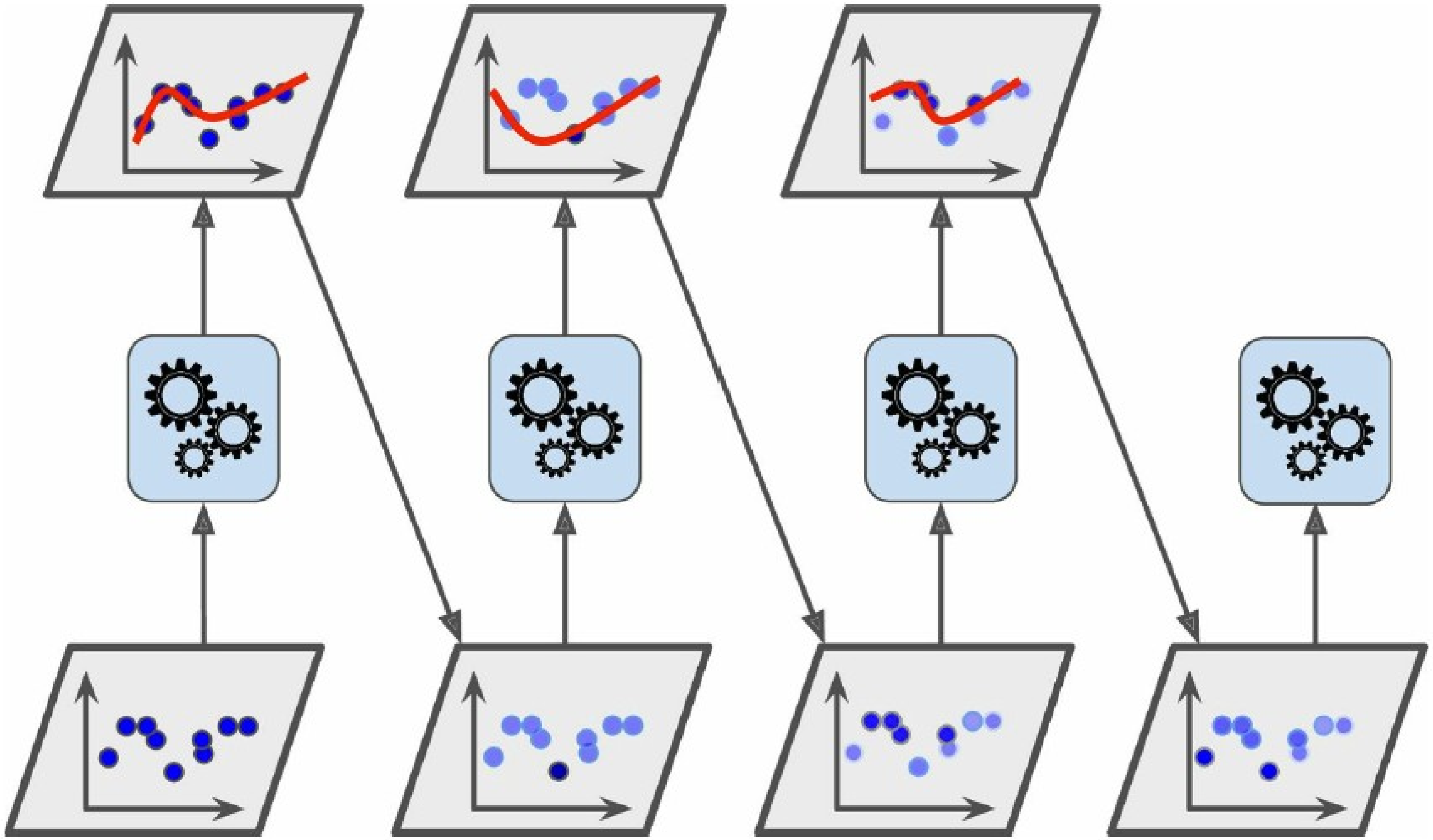
[Drucker, Schapire & Simard '92]:

- first experiments using boosting
- limited by practical drawbacks

[Freund & Schapire '95]:

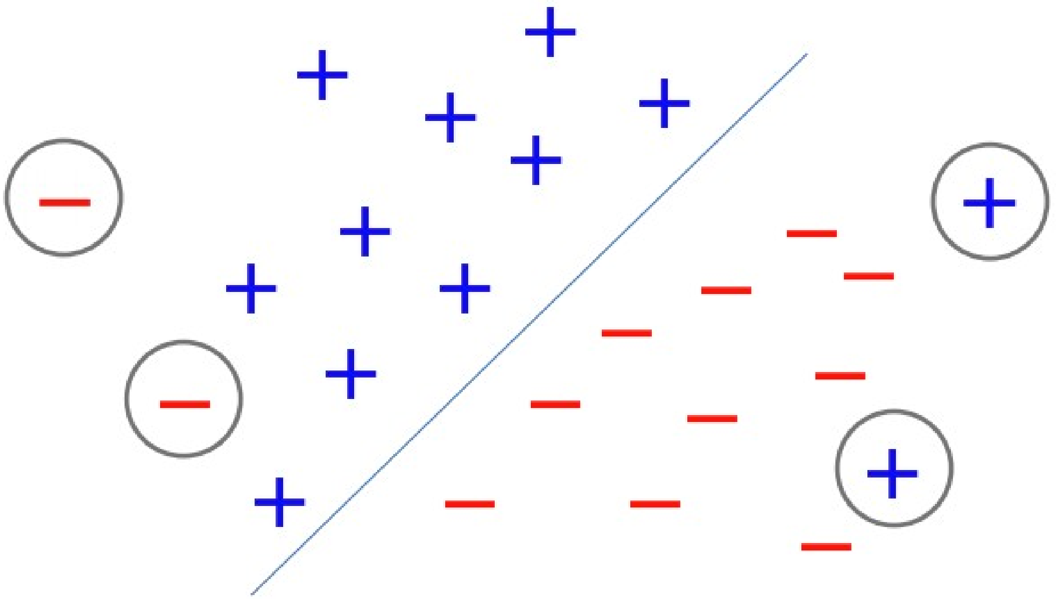
- introduced "**AdaBoost**" algorithm
- strong practical advantages over previous boosting algorithms

AdaBoost



Effect of Outliers

> Too many outliers can degrade classification performance dramatically increase time to convergence.

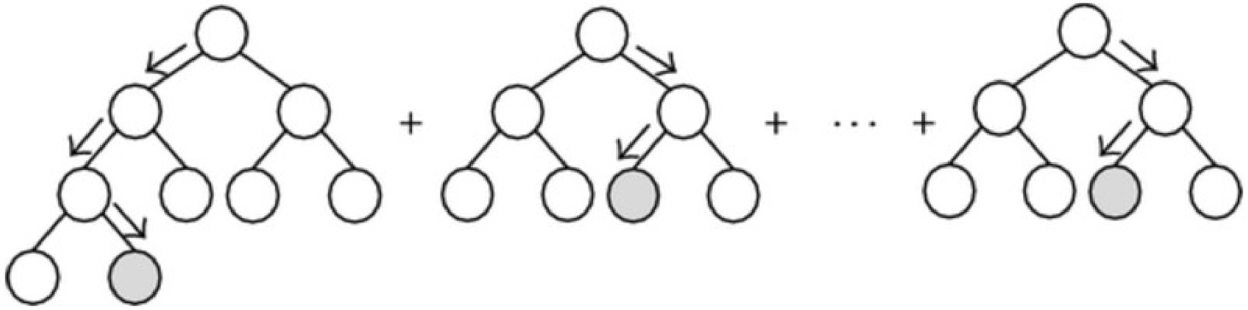


Gradient Boost Decision Trees

> **Adaboost**: redistribute the weights in the data so that later classifier focuses more on the misclassified examples by previous classifiers.

> **Gradient boosting**:

- Step 1: Calculate **the residual** for all examples in the training data (the difference between the outcome of the first learner and the real value)**.**
- Step 2: Build learner to predict/fit **the residual** left from the previous classifiers.
- These two steps continues until certain threshold is met.



XGBoost

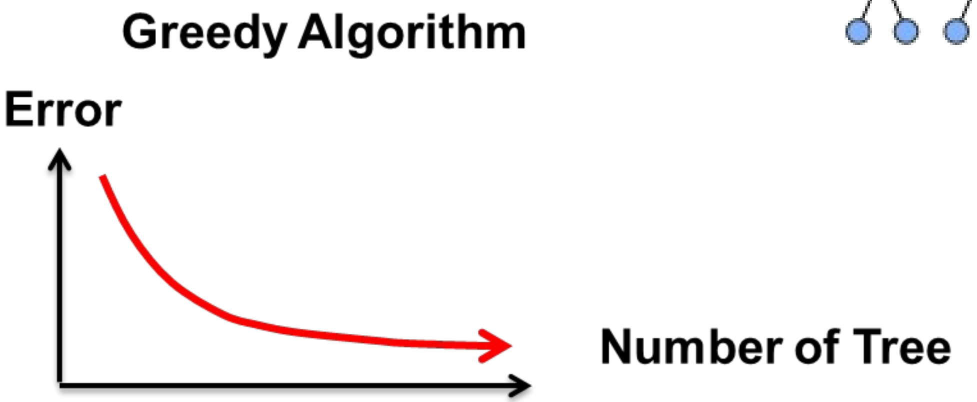
- XGBoost stands for e**X**treme **G**radient **B**oosting.
- XGBoost is an implementation of gradient boosted decision trees, created by Tianqi Chen (UW).

> *The name xgboost, though, actually refers to the engineering goal to push the limit of computations resources for boosted tree algorithms. Which is the reason why many people use xgboost.*

- The two advantages of XGBoost:
  - Execution Speed (written in C++).
  - Model Performance (a more regularized model formalization to control over-fitting).

XGBoost

- Additive tree model: add new trees that complement the already-built ones
- Response is the optimal linear combination of all decision trees



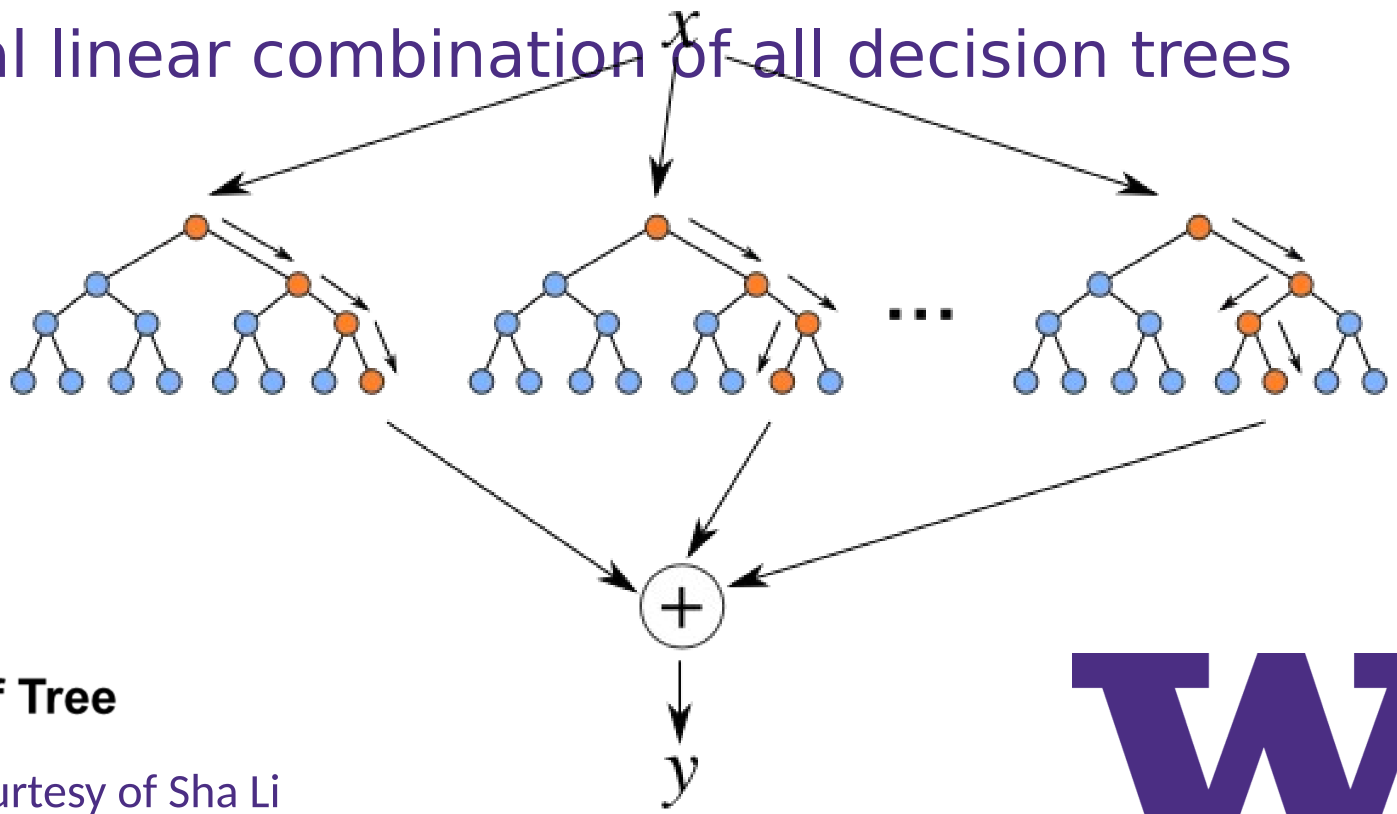
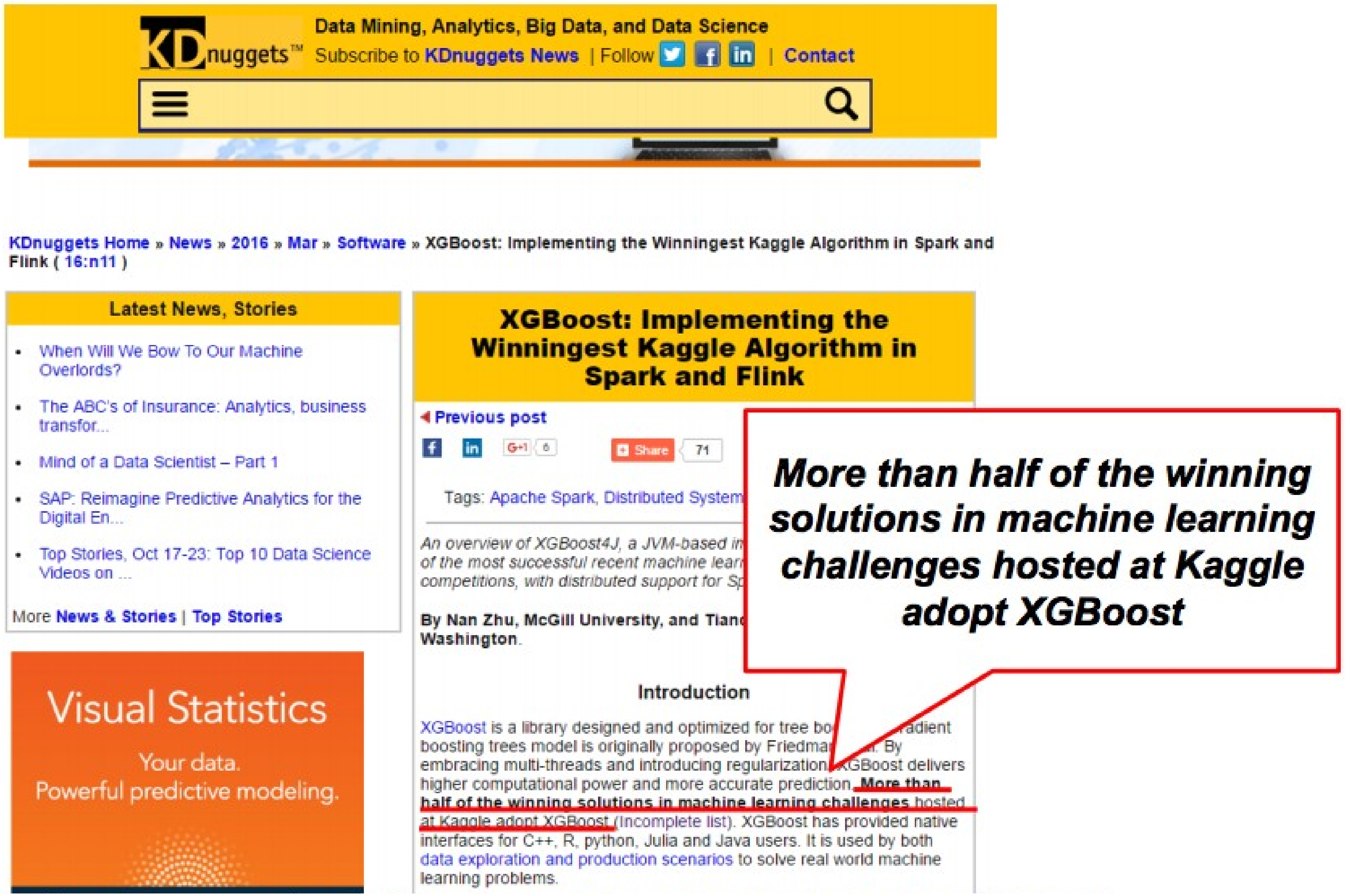


Image courtesy of Sha Li

XGBoost Occupied Kaggle



XGBoost Resources

- The original paper: https://arxiv.org/abs/1603.02754
- Github repo: https://github.com/dmlc/xgboost

Boosting: Pros and Cons

- Pros:
  - Often best off-the-shelf accuracy on many problems.
  - Using model for prediction requires only modest memory and is fast.
  - Does not require careful normalization of features to perform well.
  - Like decision trees, handles a mixture of feature types.
- Cons:
  - The models are often difficult for humans to interpret.
  - Requires careful tuning of the learning rate and other parameters.
  - Not easy to parallel (unlike random forest) since each classifier can only be trained after the previous one has been trained.

Bagging vs. Boosting

- Bagging:
  - Resamples data points
  - Weight of each classifier is the same
  - Only variance reduction
- Boosting:
  - Reweights data points
  - Weight is dependent on classifier's accuracy.
  - Both bias and variance reduced.