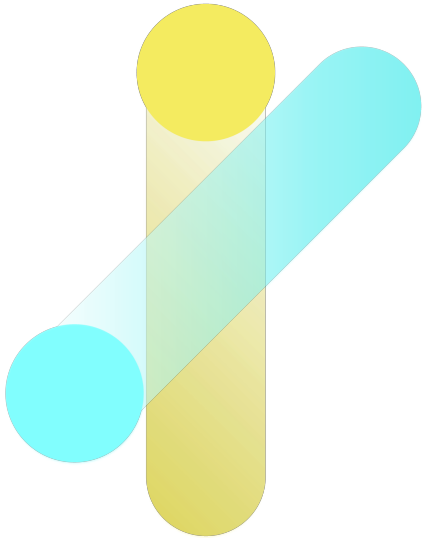
the shortcut × moz://a

DATA SCIENCE
TRAINING PROGRAM



Ensemble models -
wisdom of the crowd

Sirong Huang @ selko.io

Ensemble Models — wisdom of the crowd

What is ensemble models:

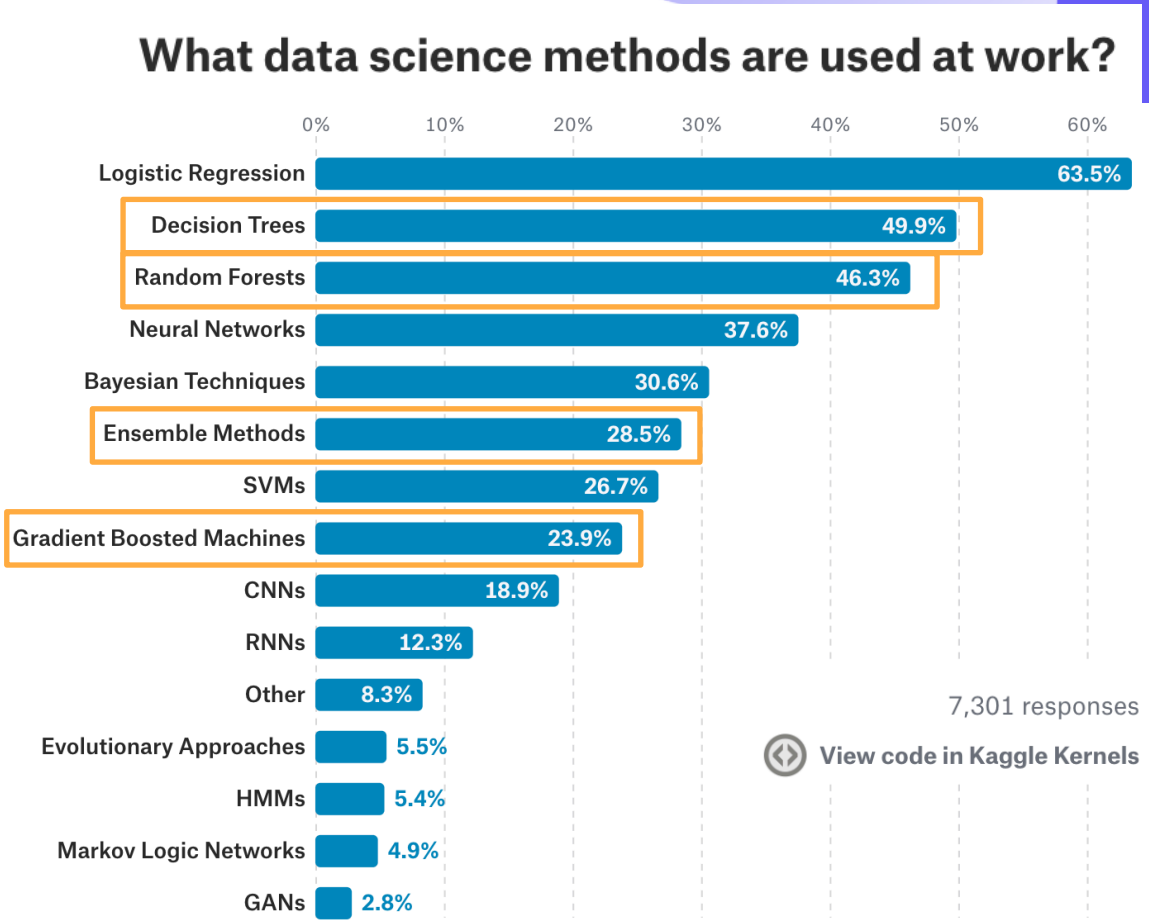
— Combining '**weak learners**' to build a '**strong learner**'

Ensemble Models — wisdom of the crowd

1. **Ensemble methods with tree models**
    1. Decision Tree
    2. Random Forest (bagging)
    3. XGBoost (boosting)
2. **Ensemble general models**
3. **Experiment tracking**

Motivation

- Powerful performance
- Widely used in practice
- Top winning solutions for many data science competitions



Source: https://www.kaggle.com/surveys/2017

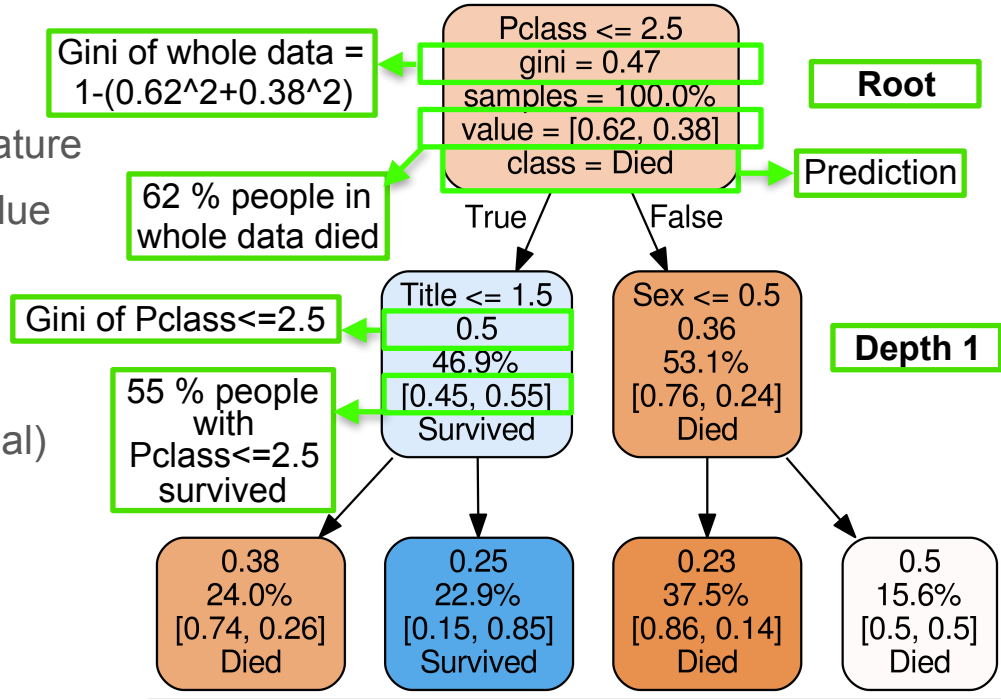
Decision Trees — Illustrated

- **How to split the tree?**
  - Calculate every possible split for every feature
  - Select the one that separate the target value the best
- **Measure of feature split**:
  - Gini: purity of split (0-purest, 0.5 equal)
  - Entropy: information gain (0-purest, 1 equal)
  - Both works with minimal difference

$$Entropy = \sum_{i=1}^{C} -p_i * \log_2(p_i)$$

$$Gini = 1 - \sum_{i=1}^{C} (p_i)^2$$

Fig 1. Decision tree for Titanic dataset

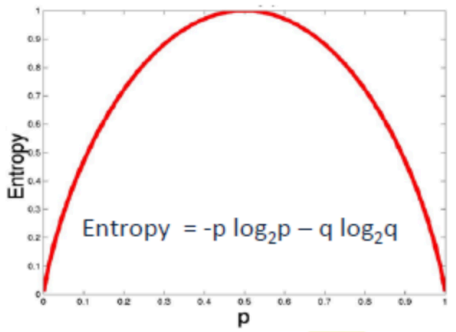


Decision Trees — Illustrated

- **How to split the tree?**
  - Calculate every possible split for every feature
  - Select the one that separate the target value the best
- **Measure of feature split**:
  - Gini: purity of split (0-purest, 0.5 equal)
  - Entropy: information gain (0-purest, 1 equal)
  - Both works with minimal difference

$$Entropy = \sum_{i=1}^{C} -p_i * \log_2(p_i)$$

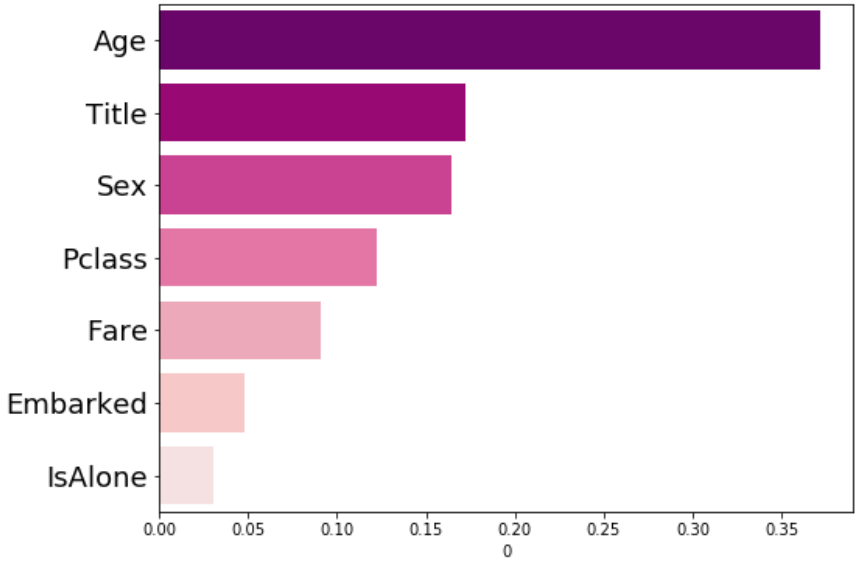
$$Gini = 1 - \sum_{i=1}^{C} (p_i)^2$$



Entropy = -0.5 $\log_2 0.5$ – 0.5 $\log_2 0.5$ = 1

Decision Trees — Feature importance

Fig 2. Feature importance of decision tree

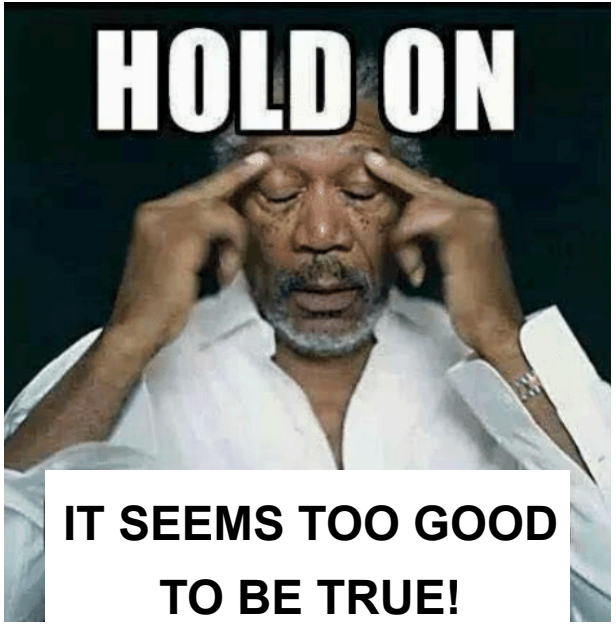


Feature importance =

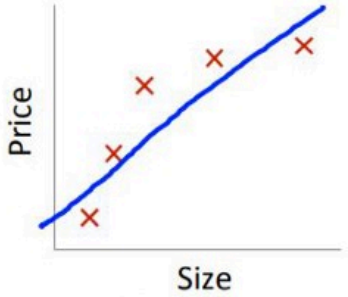
decrease in node purity * probability of reaching the node

Decision Trees — Advantages

- **Easy to understand** and explain
- Built-in feature selection with **feature importance** score
- Handle **mixed datatypes** (both x and numerical, including missing data!)
- No assumption about the **shape of data**
  - No need to be normal distributed
  - No need to scale or normalize features(to similar numerical range)
  - No assumption on linearity and no need to specify interaction terms

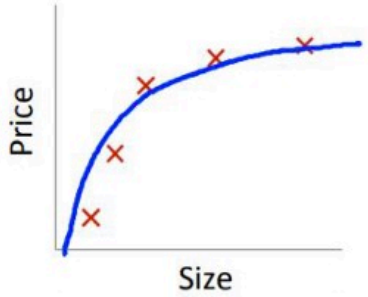


Bias & Variance VS Underfitting & Overfitting



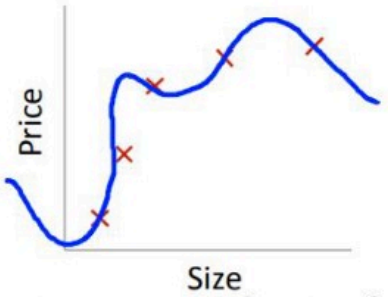
$\theta_0 + \theta_1 x$

High bias



$\theta_0 + \theta_1 x + \theta_2 x^2$

"Just right"



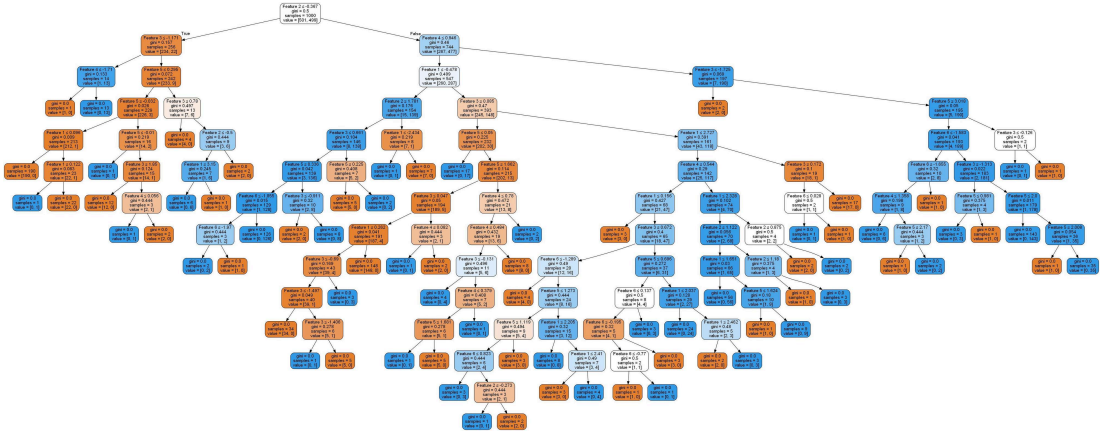
$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

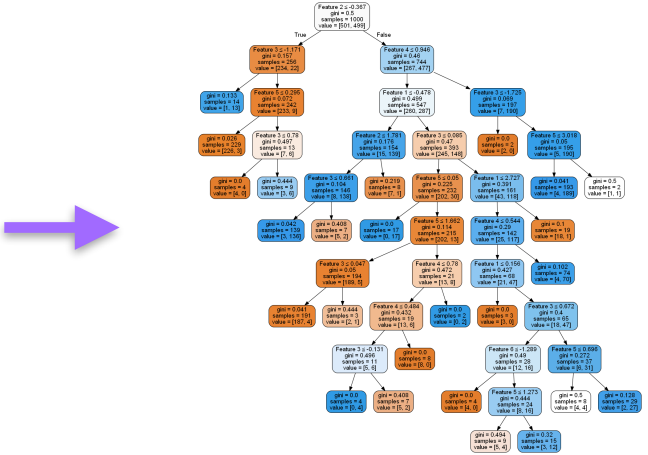
High variance

- Bias = training error (not validation/test error!!!)
- Variance = sensitivity to noise in data

Decision Tree — Pruning

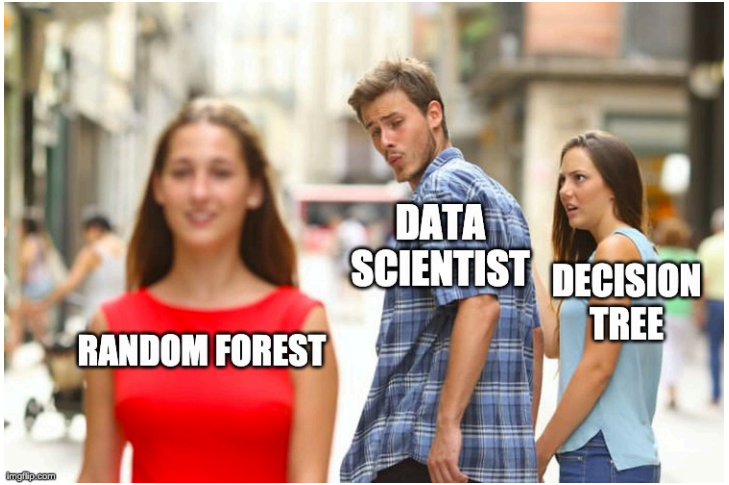
- Removing leaf with number of samples <= certain threshold





Decision Trees — Limitations

- **Overfitting** to training data => Not generalizing well to new data
- **Sensitive** to noise => small change in data can result in large change in tree structure
- Can be **slow** and expensive to compute, compare to efficient parametric models such as logistic regression



Bagging (Bootstrap Aggregating)

1. **Bootstrapping**: draw a random sample of size N with replacement
2. **Train** the same model (eg. decision tree) on these data subsets independently
3. **Aggregate** the predictions (eg. averaging, voting)

- **Reduce variance**, reduce overfitting and improve performance
- Can be used to improve any model!

Fig 3. Distribution of whole data

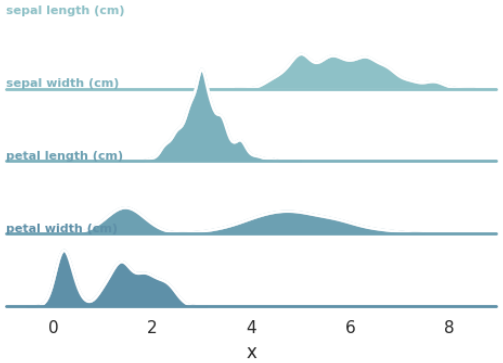
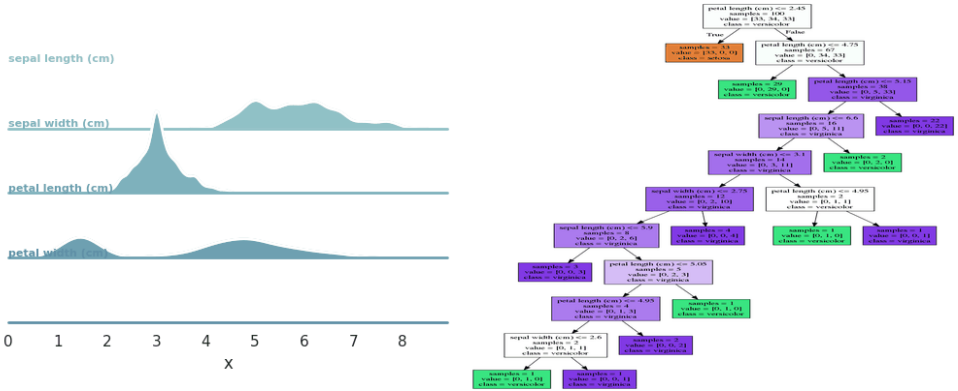


Fig 4. Bootstrapping and Random Forest



Why Bagging Reduces Variance?

Fig 5. Models with High Variance

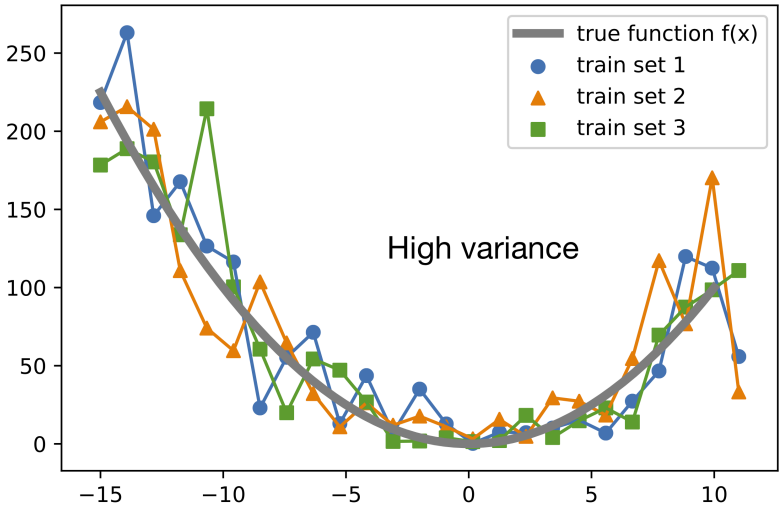
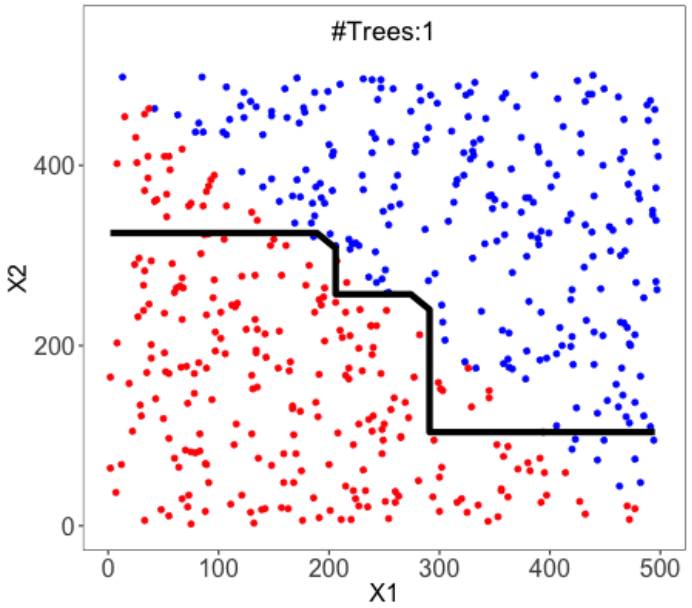


Fig 6. Decision Boundary of Bagging models



http://rasbt.github.io/mlxtend/user_guide/evaluate/bias_variance_decomp/

https://towardsdatascience.com/why-random-forests-outperform-decision-trees-1b0f175a0b5

Random Forest — wisdom of many decision trees

- **Data bagging:** randomly select subset of data with replacement **=> reduce variance**
- **Feature bagging:** random select subset of feature **=> reduce correlation between trees**

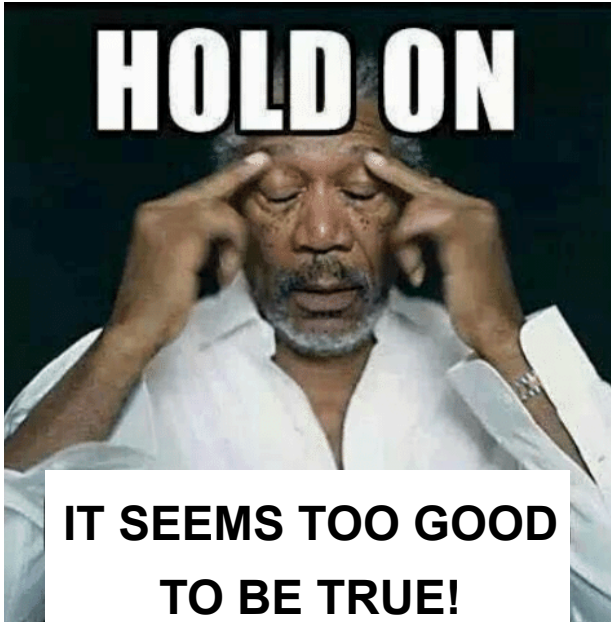
```
from sklearn.ensemble import RandomForestClassifier

RF = RandomForestClassifier(n_estimators=100,
                            criterion='gini',
                            max_depth=5,
                            min_samples_leaf=10,
                            max_features='sqrt',
                            random_state=0)
```

Paper on feature bagging: https://arxiv.org/pdf/1805.02587.pdf

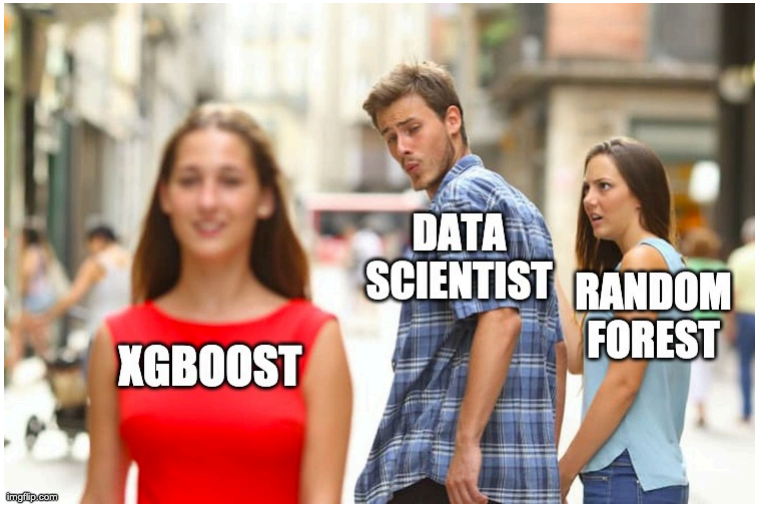
Random Forest — Advantages

- **Better performance**, more **robust** than decision tree
- Built-in feature selection with **feature importance** score (same as DT)
- Handle **mixed datatypes** (same as DT)
- No assumption about the **shape of data** (same as DT)
  - No need to be normal distributed
  - No need to scale or normalize features(to similar numerical range)
  - No assumption on linearity and no need to specify interaction terms



Random Forest — Limitations

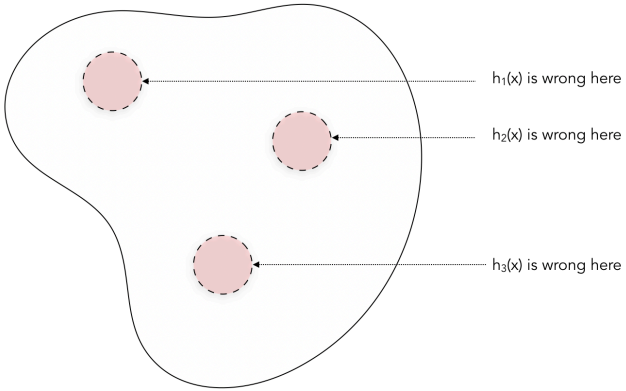
- **Not** easily **interpretable** anymore
- **Sensitive** to noise => small change in data can result in large change in tree structure (same as DT)
- Not very slow due to simplicity of subtrees, and highly parallelizable



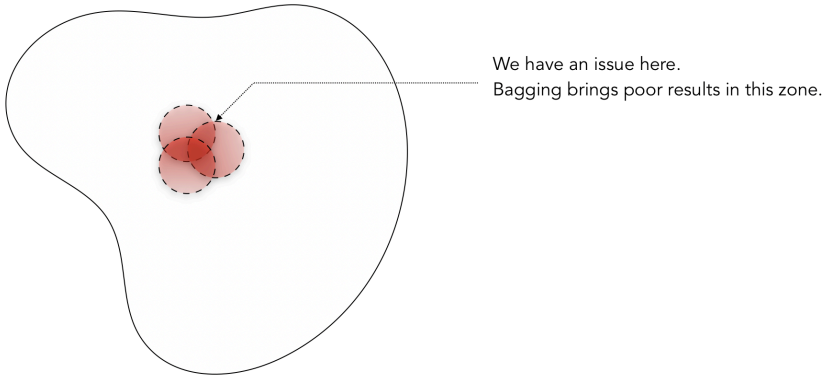
Bagging — Limitations

Success of Bagging relies on **most** sub-models predicts correctly

Bagging - Classification Process



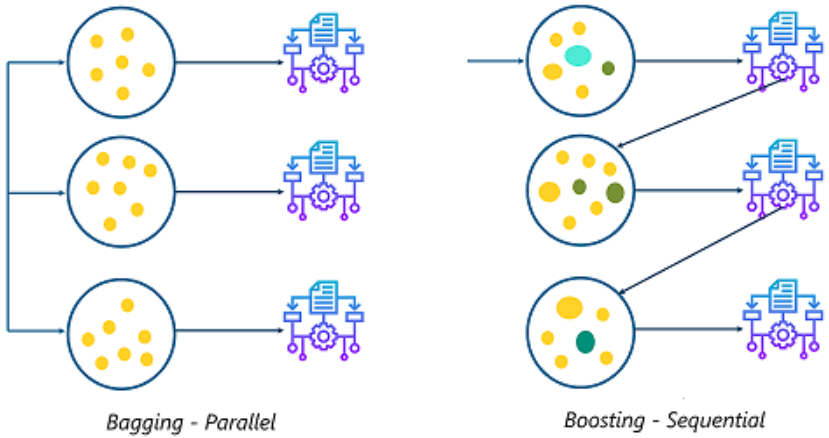
Bagging - Limitations



Boosting — Sequential Ensemble

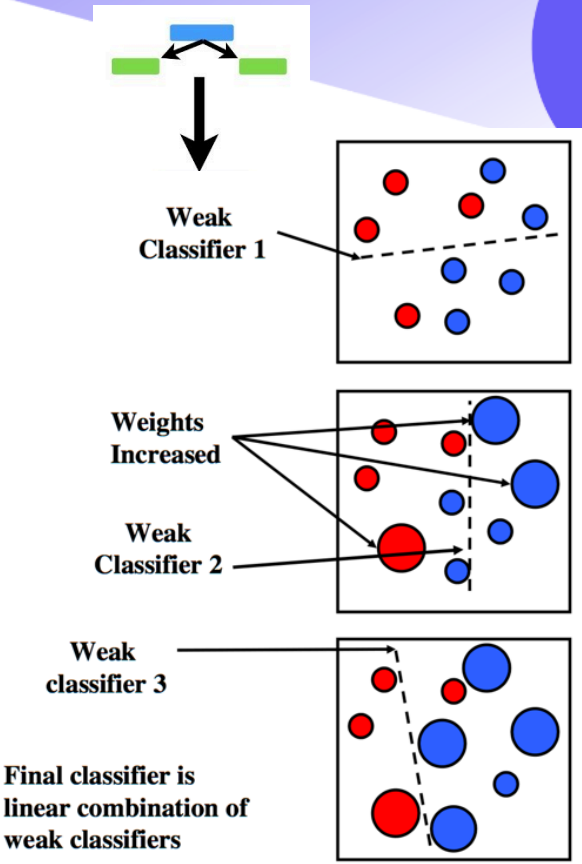
1. Pick a base model: usually high bias low variance (decision tree with depth=1)
2. Fit models **sequentially** by giving **more weight** to the most **wrongly predicted** data
3. Weighted average of predictions from previous models

- **Reduce bias** (mainly, but it can also reduce variance)



Adaboost — Adaptive Boosting

1. All data samples given **equal weight** = 1/N
2. Build the model on data
3. Calculate and update **weights** for the model and each data point
   - **Model** weight $\theta_m = \frac{1}{2} ln(\frac{1-\epsilon_m}{\epsilon_m}).$ accuracy>50% positive weight, <50% negative weight, =50% 0
   - More weight on wrongly predicted **data points** in the error rate calculation => penalize more on mistakes made on these data => model performs better on these data points
4. Fit a new model(same type) on data to optimise for weighted error rate
5. Repeat K times and **weighted averaging** the predictions with model weight



Gradient Boosting

Tree 2 tries to predict the residual of tree 1

Final prediction = Tree 1 prediction + learning rate * Tree 2 prediction

PersonId	Salary	Tree 1 Pred	Tree 1 Residual	Tree 2 Pred	Combined Pred	Final Residual
1	100,000	120,000	-20,000	-15,000	105,000	-5,000
2	90,000	85,000	5,000	0	85,000	5,000
3	120,000	120,000	0	-5,000	115,000	5,000
4	140,000	120,000	20,000	15,000	135,000	5,000
5	180,000	130,000	50,000	25,000	155,000	25,000
6	200,000	210,000	-10,000	10,000	220,000	-20,000
7	70,000	85,000	-15,000	-15,000	70,000	0
8	150,000	120,000	30,000	10,000	130,000	20,000
9	105,000	120,000	-15,000	-15,000	105,000	0

1. Fit model to data and calculate **residual = error**
2. Fit model to residuals
3. Aggregate prediction from models in step 1 & 2
4. Repeat until error stops decreasing / reaches maximum number of runs

Model choice: medium size model eg. 8-32 depth trees

- Learning rate => regularization

https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d

Gradient Descent

Step 2 = Step 1 — learning rate * gradient (slope)

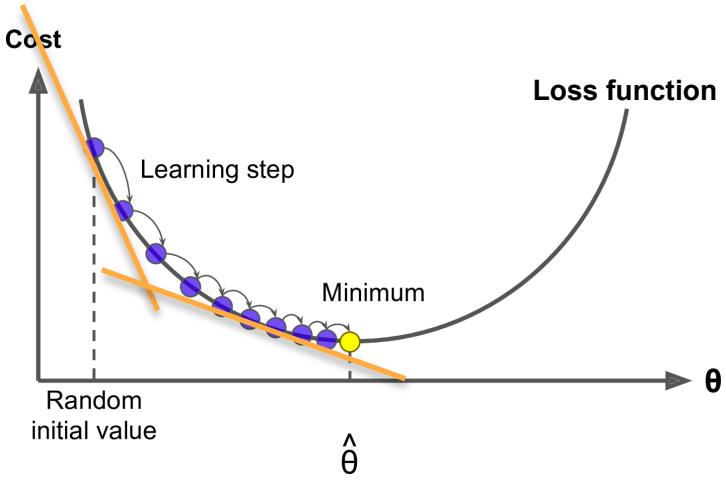


Fig 7. Gradient descent (Geron, 2017)

Fig 8. Tangent (slope) / gradient graph

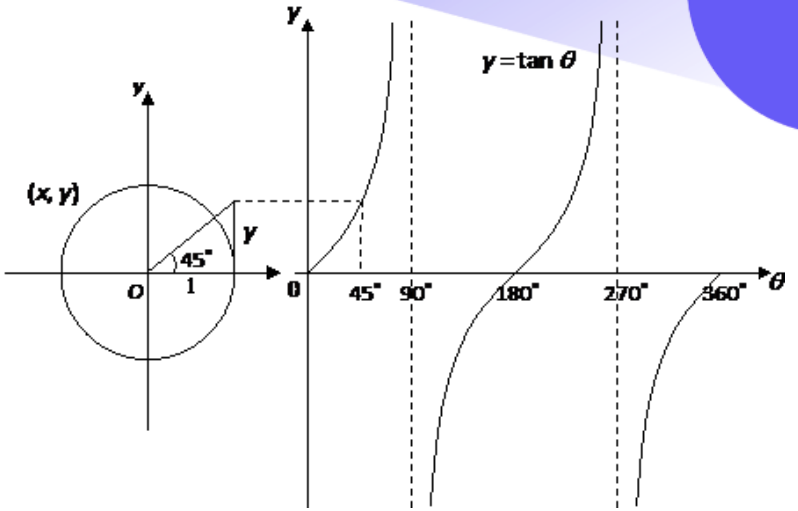
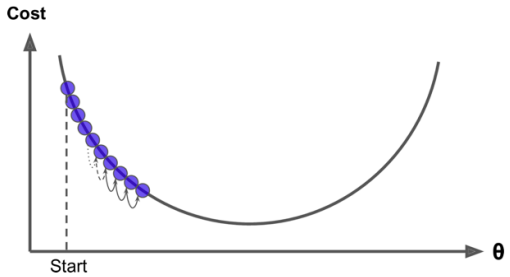
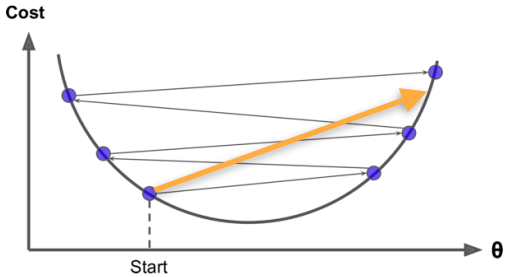


Fig 9. Effect of too small/large learning rate



a) too small



a) too big

Advance Gradient Boosting Models — Best-in-Class

Fig 10. Model experiment with generated data

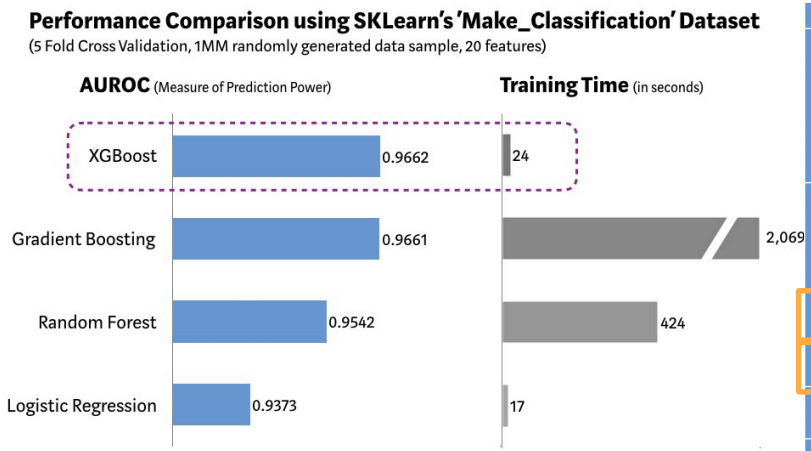


Table 1. Benchmark of flight delay Kaggle dataset

	XGBoost	Light BGM		CatBoost	
Parameters Used	max_depth: 50 learning_rate: 0.16 min_child_weight: 1 n_estimators: 200	max_depth: 50 learning_rate: 0.1 num_leaves: 900 n_estimators: 300		depth: 10 learning_rate: 0.15 l2_leaf_reg= 9 iterations: 500 one_hot_max_size = 50	
Training AUC Score	0.999	Without passing indices of categorical features	Passing indices of categorical features	Without passing indices of categorical features	Passing indices of categorical features
		0.992	0.999	0.842	0.887
Test AUC Score	0.789	0.785	0.772	0.752	0.816
Training Time	970 secs	153 secs	326 secs	180 secs	390 secs
Prediction Time	184 secs	40 secs	156 secs	2 secs	14 secs
Parameter Tuning Time (for 81 fits, 200 iteration)	500 minutes	200 minutes		120 minutes	

https://www.kaggle.com/usdot/flight-delays/data

https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9199be63d

Stacking — Use predictions as input features

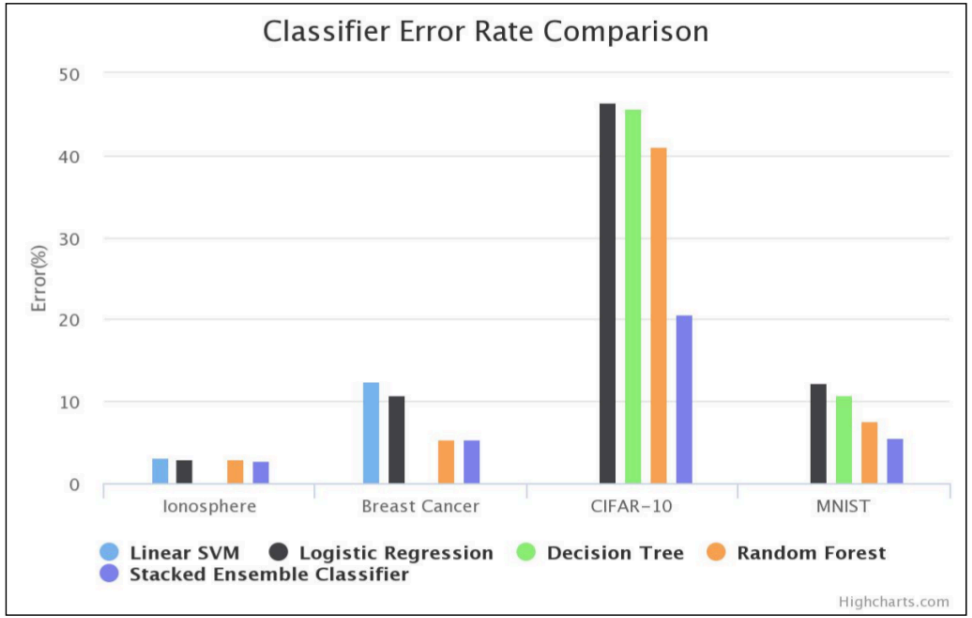
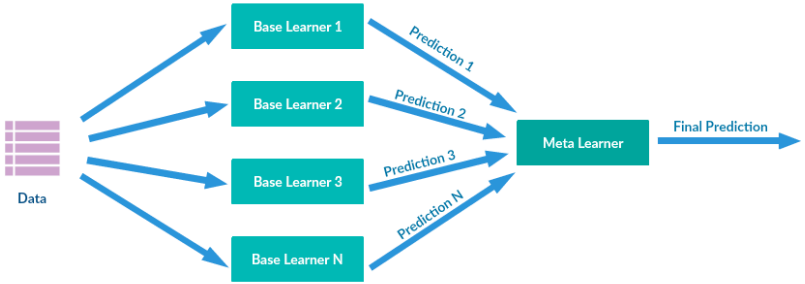


Figure: Comparison of the error rate (%) for different classifiers evaluated on the Ionosphere, Breast Cancer, MNIST and CIFAR-10 datasets.



Improve predictive performance

https://web.njit.edu/~avp38/projects/multi_projects/ensemble.html

Source of image: http://supunsetunga.blogspot.com/

Stacking — Implementation

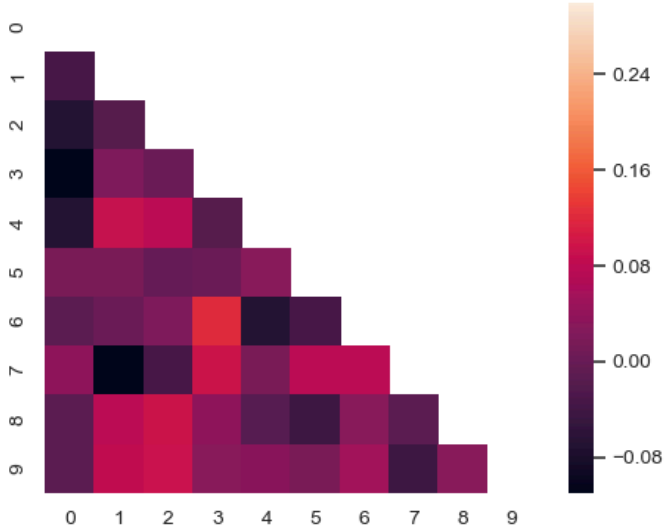
Model selection criteria:

1. Less correlated => diversity
2. Perform well (sometimes low priority compared to criteria 1.)

How to stack:

1. Simple averaging / voting
2. Machine learning models to take predictions as input features

Fig 10. Model prediction correlation heatmap



Tips for stacking

1. Important to design validation set properly so no information is leaked!!!
2. Selection of stacked models/submissions

Ensemble Methods — Flavors

What to ensemble:

- different algorithms
- different hyperparameters
- different feature subsets
- different data subsets

How to ensemble:

- bagging, boosting, stacking…

•Jeong-Yoon Lee, Winning Data Science Competitions

Ensemble Methods — Pros and Cons

Pros:

- Stablized model (generalize better, resilient to noise in training data, low variance…)
- Better performance (reduced bias, while keeping variance in check)

Cons:

- Additional computation time and cost

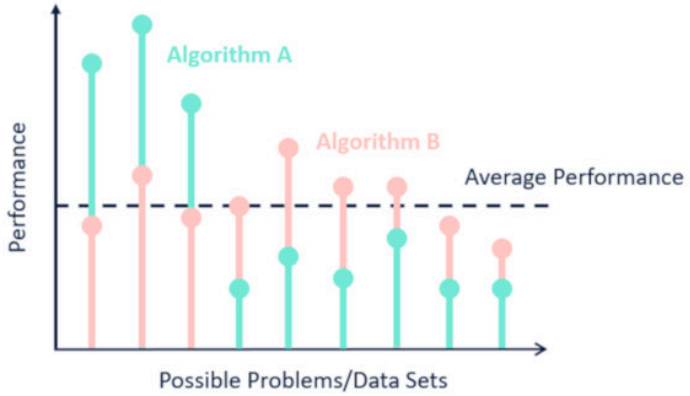
Ensemble Methods — No Free Lunch?

Doesn't apply to real world cases:

- There is no point to average over all possible problems.

- We are solving a specific problem(there might be variations but not as extreme as averaging over all possible problems)

The search landscape is then limited, therefore bias-variance trade-off is not 1:1, or not absolute.



https://www.kdnuggets.com/2019/09/no-free-lunch-data-science.html

Summary of Ensemble Methods

- **Bagging => reduce variance**
  - Fit independent models on subset of data/ feature
- **Boosting => reduce bias**
  - Fit models sequentially
- **Stacking**
  - Fit 2nd stage model using 1st stage models' predictions as features

Experiment Tracking

1. MLflow, Neptune, Modelchimp… …
2. Code snippets
3. Mlbot

Exercise:

- Define a problem to solve with the Airbnb dataset
- Build a tree-based model that solves the problem
- Build an ensemble of several models (tree-based and/or other model)
- Use an experiment tracking tool to track your model performances
- Write an consultancy report of the model (target audience: Airbnb)
  - Performance
  - Assumptions
  - Reasoning the decisions made during modelling process
  - Strength & weakness (pitfalls)
  - …