05.03.2019

Statistical Methods in AI (CSE/ECE 471)

Lecture-16: Ensemble Methods (Bagging, Boosting, Stacking)

Ravi Kiran

Center for Visual Information Technology (CVIT), IIIT Hyderabad

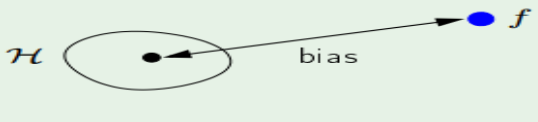
$$\mathbb{E}_{\mathcal{D}}\left[\left(g^{(\mathcal{D})}(\mathbf{x}) - f(\mathbf{x})\right)^2\right] = \underbrace{\mathbb{E}_{\mathcal{D}}\left[\left(g^{(\mathcal{D})}(\mathbf{x}) - \bar{g}(\mathbf{x})\right)^2\right]}_{\text{var}(\mathbf{x})} + \underbrace{\left(\bar{g}(\mathbf{x}) - f(\mathbf{x})\right)^2}_{\text{bias}(\mathbf{x})}$$

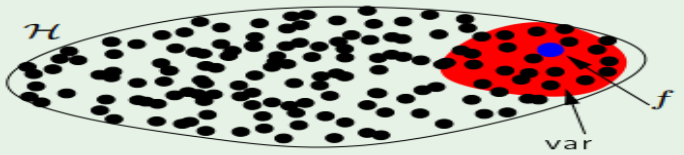
$$\mathbb{E}_{\mathbf{x}}[\text{bias}(\mathbf{x}) + \text{var}(\mathbf{x})]$$

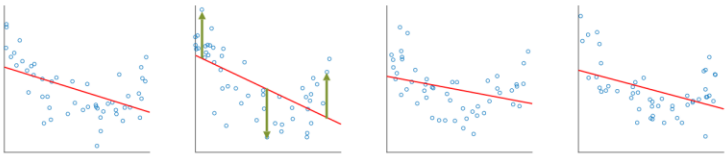
The tradeoff

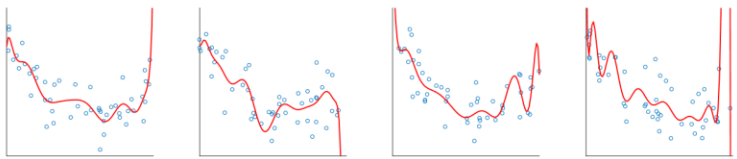
$$\text{bias} = \mathbb{E}_{\mathbf{x}}\left[\left(\bar{g}(\mathbf{x}) - f(\mathbf{x})\right)^2\right]$$

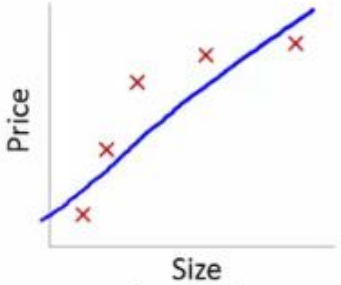
$$\text{var} = \mathbb{E}_{\mathbf{x}}\left[\mathbb{E}_{\mathcal{D}}\left[\left(g^{(\mathcal{D})}(\mathbf{x}) - \bar{g}(\mathbf{x})\right)^2\right]\right]$$





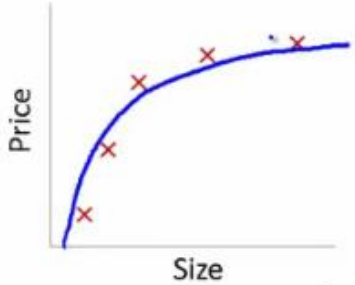






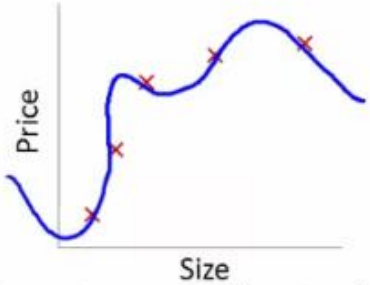
$\theta_0 + \theta_1 x$

High bias
(underfit)



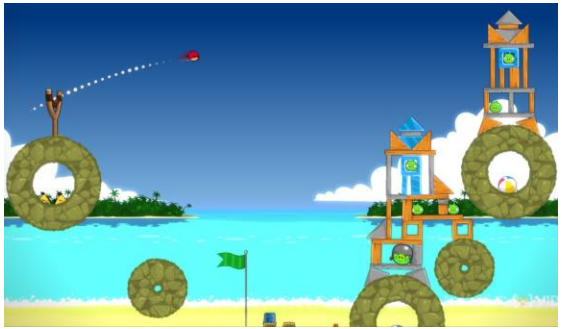
$\theta_0 + \theta_1 x + \theta_2 x^2$

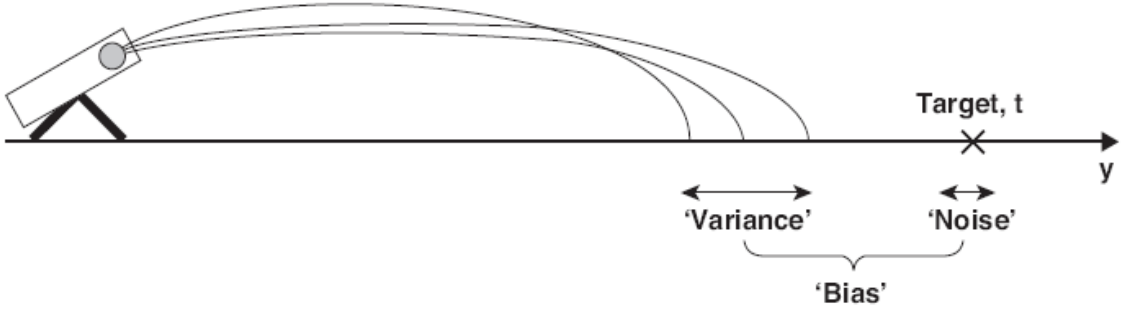
"Just right"

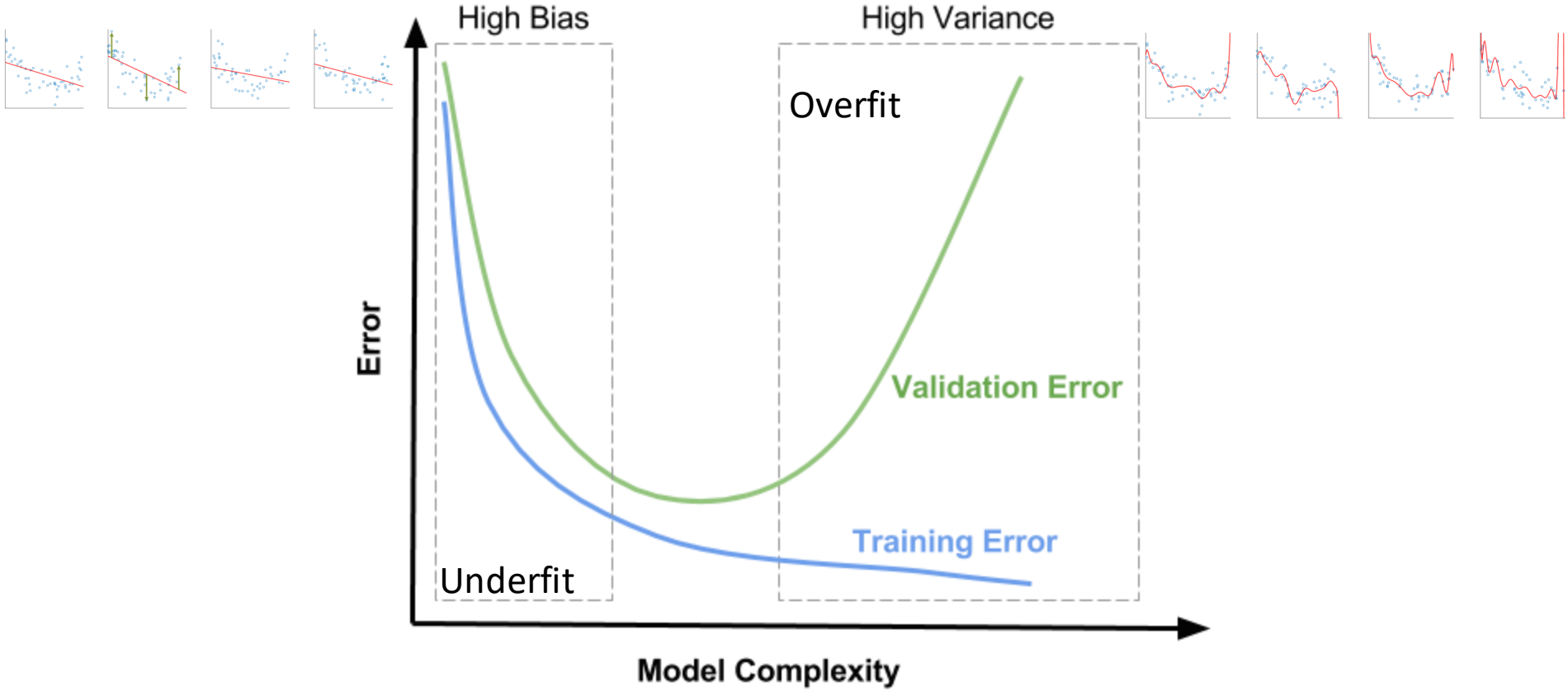


$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

High variance
(overfit)






High Bias
High Variance
Overfit
Validation Error
Training Error
Underfit
Error
Model Complexity

Regularization

- Remember the intuition: complicated hypotheses lead to overfitting
- Idea: change the error function to *penalize hypothesis complexity*:

$$J(\mathbf{w}) = J_D(\mathbf{w}) + \lambda J_{pen}(\mathbf{w})$$

  This is called *regularization* in machine learning and *shrinkage* in statistics
- λ is called *regularization coefficient* and controls how much we value fitting the data well, vs. a simple hypothesis

Regularization for linear models

- A squared penalty on the weights would make the math work nicely in our case:

$$\frac{1}{2}(\mathbf{\Phi w} - \mathbf{y})^T(\mathbf{\Phi w} - \mathbf{y}) + \frac{\lambda}{2}\mathbf{w}^T\mathbf{w}$$

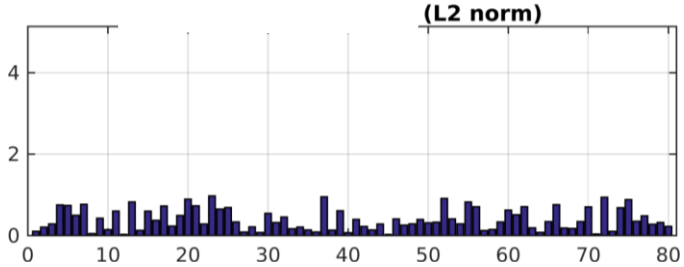
- This is also known as *L_2 regularization*, or *weight decay* in neural networks
- By re-grouping terms, we get:

$$J_D(\mathbf{w}) = \frac{1}{2}(\mathbf{w}^T(\mathbf{\Phi}^T\mathbf{\Phi} + \lambda\mathbf{I})\mathbf{w} - \mathbf{w}^T\mathbf{\Phi}^T\mathbf{y} - \mathbf{y}^T\mathbf{\Phi w} + \mathbf{y}^T\mathbf{y})$$

- Optimal solution (obtained by solving $\nabla_{\mathbf{w}} J_D(\mathbf{w}) = 0$)

$$\mathbf{w} = (\mathbf{\Phi}^T\mathbf{\Phi} + \lambda I)^{-1}\mathbf{\Phi}^T\mathbf{y}$$

- If there are irrelevant features in the input (i.e. features that do not affect the output), L_2 will give them small, but non-zero weights.
- Ideally, irrelevant input should have weights exactly equal to 0.



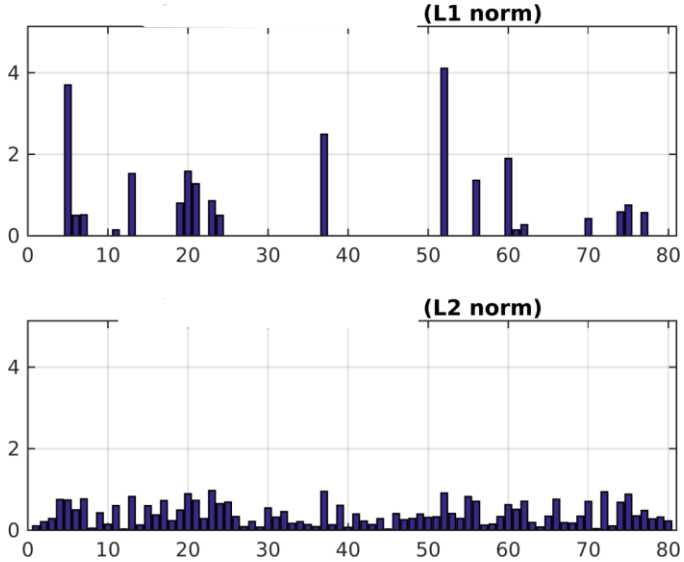
L_1 Regularization for linear models

- Instead of requiring the L_2 norm of the weight vector to be bounded, make the requirement on the L_1 norm:

$$\begin{aligned} \min_{\mathbf{w}} J_D(\mathbf{w}) &= \min_{\mathbf{w}} (\mathbf{\Phi w} - \mathbf{y})^T (\mathbf{\Phi w} - \mathbf{y}) \\ \text{such that } \sum_{i=1}^{n} |w_i| &\leq \eta \end{aligned}$$

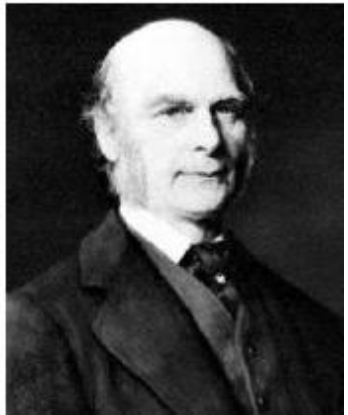
- This yields an algorithm called Lasso (Tibshirani, 1996)

- If there are irrelevant features in the input (i.e. features that do not affect the output), L_2 will give them small, but non-zero weights.
- Ideally, irrelevant input should have weights exactly equal to 0.



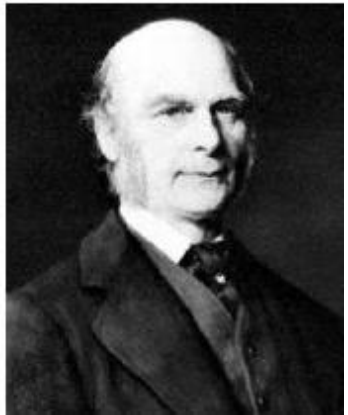
Francis Galton

- Galton promoted statistics and invented the concept of correlation.
- In 1906 Galton visited a livestock fair and stumbled upon an intriguing contest.
- An ox was on display, and the villagers were invited to guess the animal's weight.
- Nearly 800 gave it a go and, not surprisingly, not one hit the exact mark: 1,198 pounds.



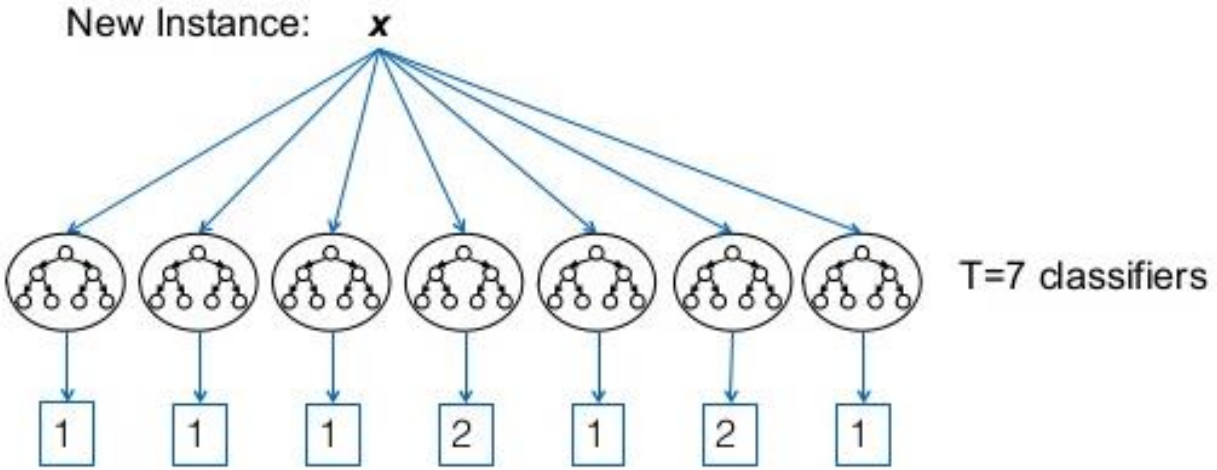
Francis Galton

- Galton promoted statistics and invented the concept of correlation.
- In 1906 Galton visited a livestock fair and stumbled upon an intriguing contest.
- An ox was on display, and the villagers were invited to guess the animal's weight.
- Nearly 800 gave it a go and, not surprisingly, not one hit the exact mark: 1,198 pounds.
- Astonishingly, however, the average of those 800 guesses came close - very close indeed. It was 1,197 pounds.



Ensemble Learning

- An ensemble is a combination of classifiers that output a final classification.



General idea

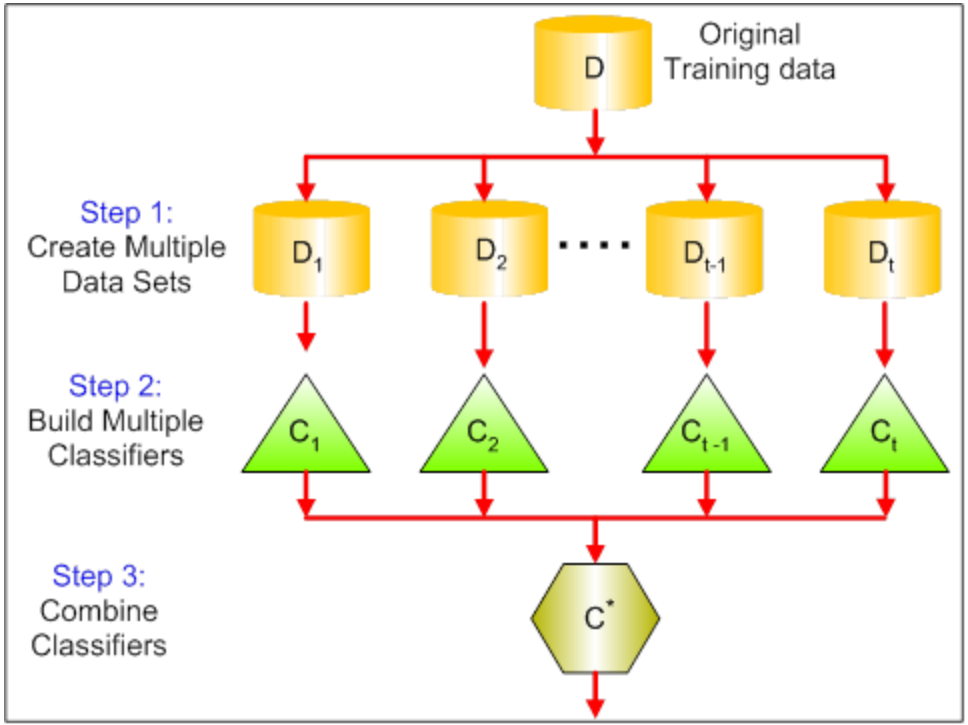
- Generate many classifiers and combine them to get a final classification
- They perform very good. In general better than any of the single learners they are composed of
- The classifiers should be different from one another
- It is important to generate diverse classifiers from the available data

How to build them?

- There are several techniques to build diverse base learners in an ensemble:
  - Use modified versions of the training set to train the base learners

- Modifications of the training set can be generated by
  - Resampling the dataset. By bootstrap sampling (e.g. bagging), weighted sampling (e.g. boosting).

Bootstrap Aggregating (Bagging)



"Bagging Predictors" [Leo Breiman, 1994]
http://statistics.berkeley.edu/sites/default/files/tech-reports/421.pdf

Why does it work?

- Suppose there are 25 base classifiers
  - Each classifier has error rate, $\varepsilon = 0.35$
  - Assume classifiers are independent
  - Probability that the ensemble classifier makes a wrong prediction:

Why does it work?

- Suppose there are 25 base classifiers
  - Each classifier has error rate, $\varepsilon = 0.35$
  - Assume classifiers are independent
  - Probability that the ensemble classifier makes a wrong prediction: $\sum_{i=13}^{25} \binom{25}{i} \varepsilon^i (1-\varepsilon)^{25-i} = 0.06$

Input:

Dataset L
Ensemble size T

```
1.for t=1 to T:
2.    sample = BootstrapSample(L)
3.    h_t = TrainClassifier(sample)
```

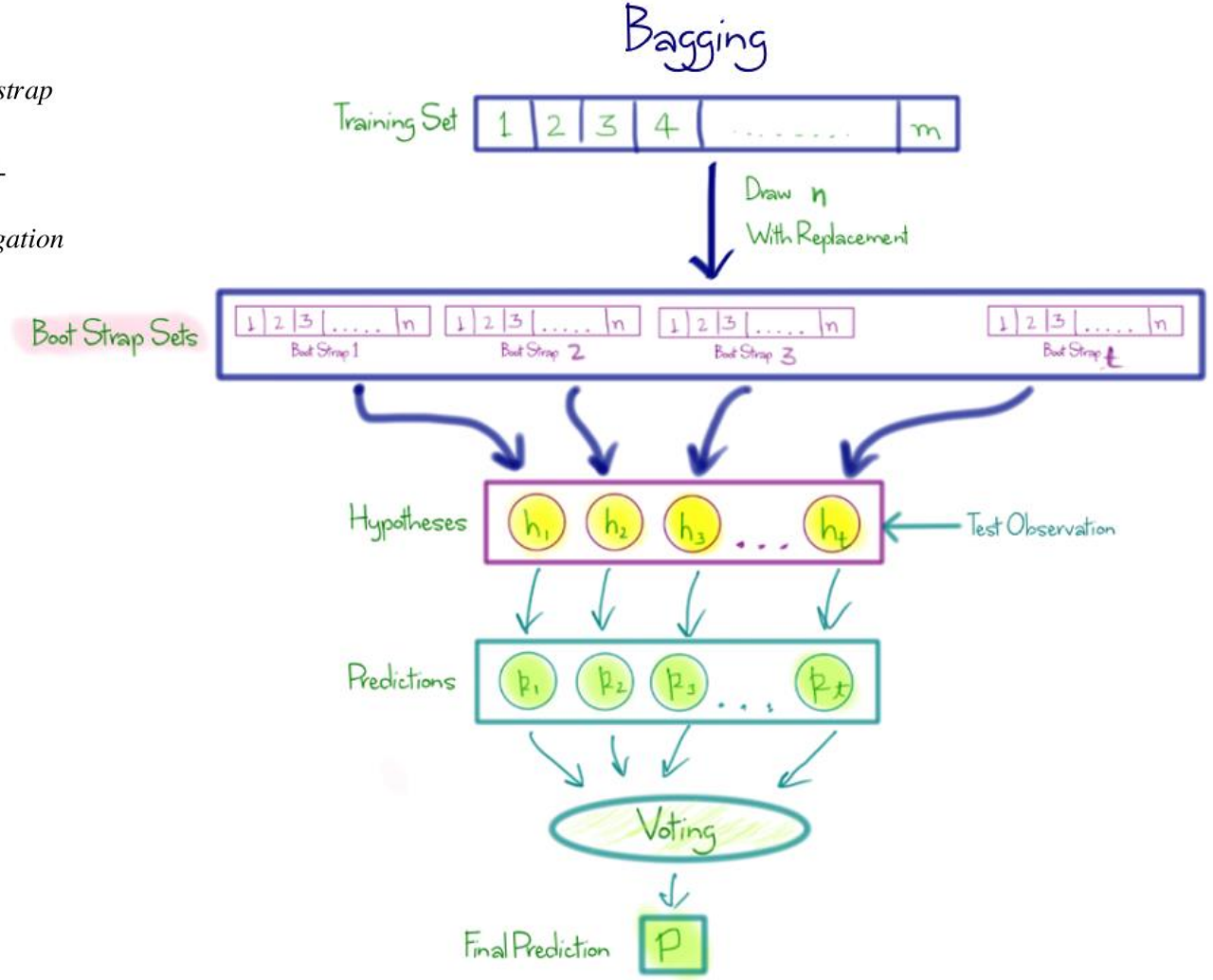
Bootstrap

+

Output:

$$H(\mathbf{x}) = \underset{j}{\operatorname{argmax}} \left(\sum_{t=1}^{T} I(h_t(\mathbf{x}) = j) \right)$$

Aggregation



Input:
Dataset L
Ensemble size T

```
1.for t=1 to T:
2.    sample = BootstrapSample(L)
3.    h_t = TrainClassifier(sample)
```

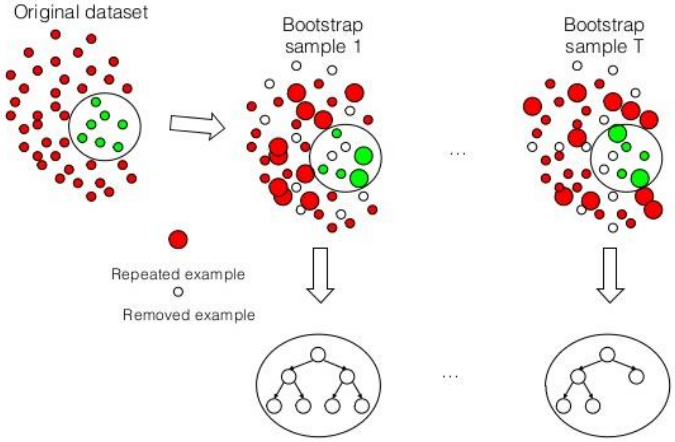
Bootstrap

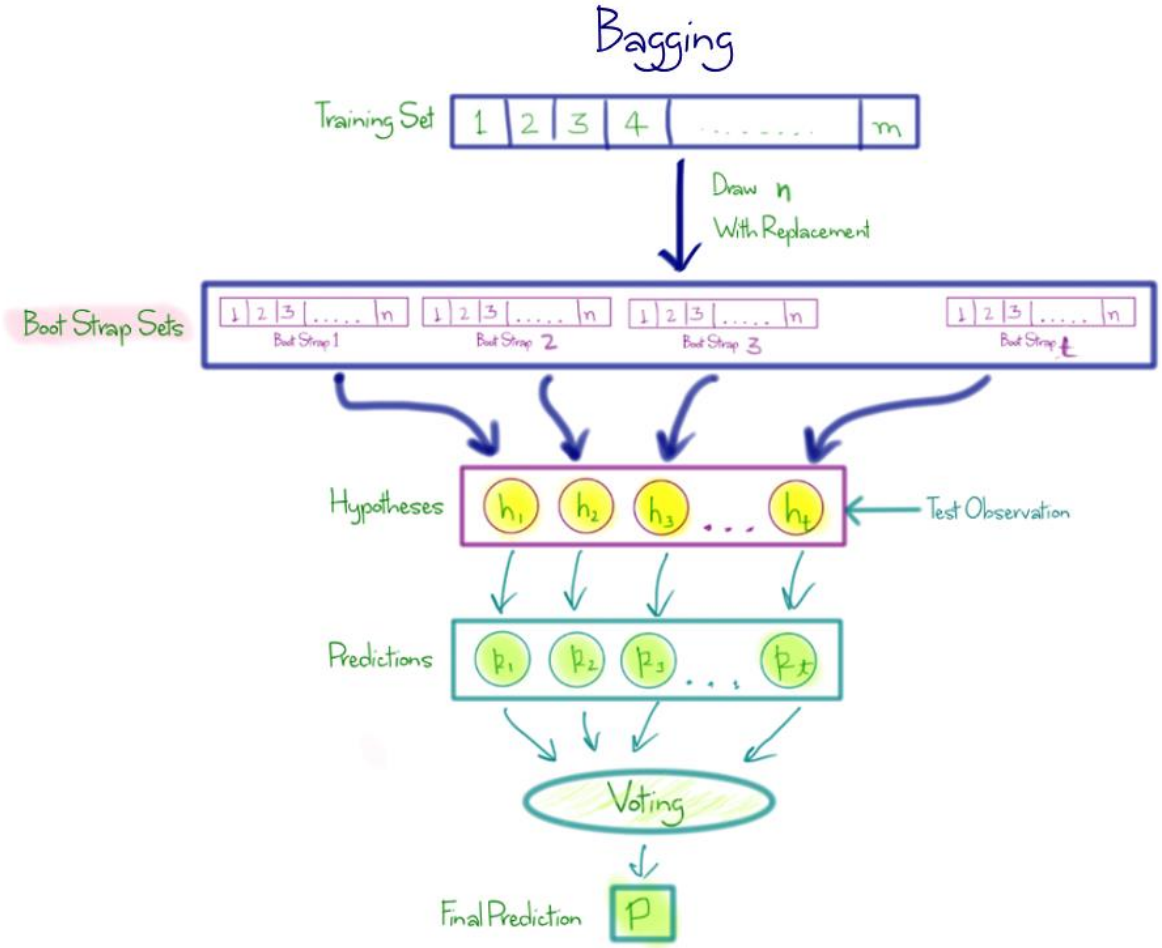
+

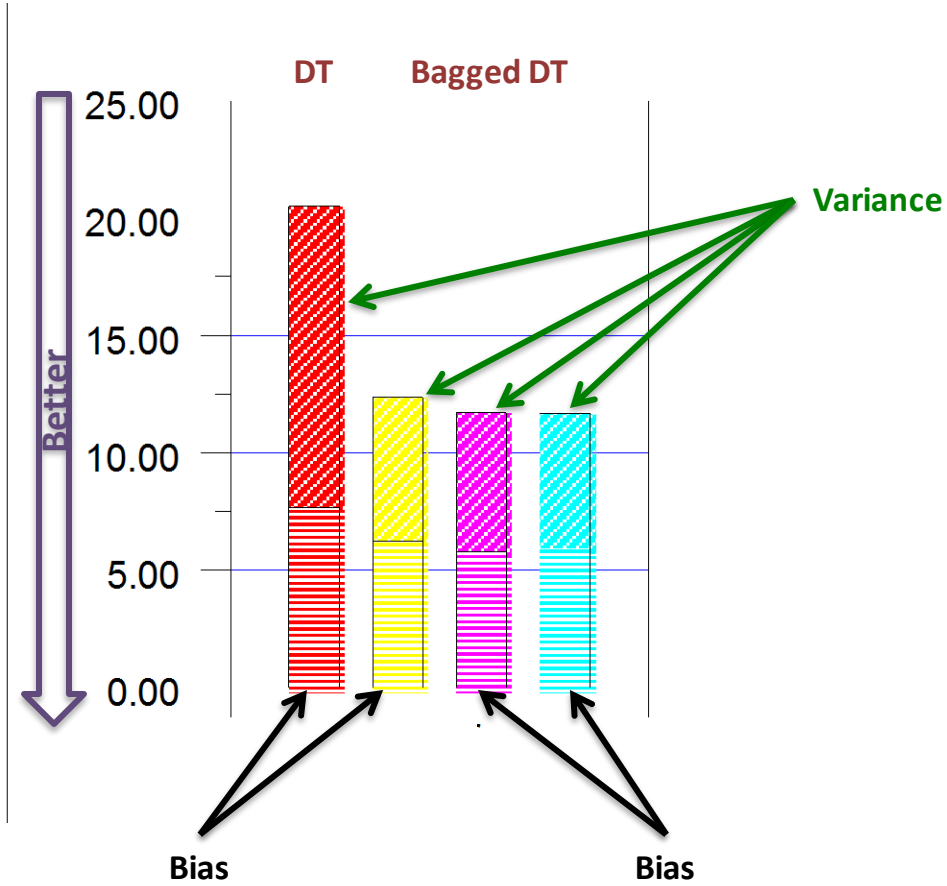
Output:

$$H(\mathbf{x}) = \underset{j}{\operatorname{argmax}} \left(\sum_{t=1}^{T} I(h_t(\mathbf{x}) = j) \right)$$

Aggregation



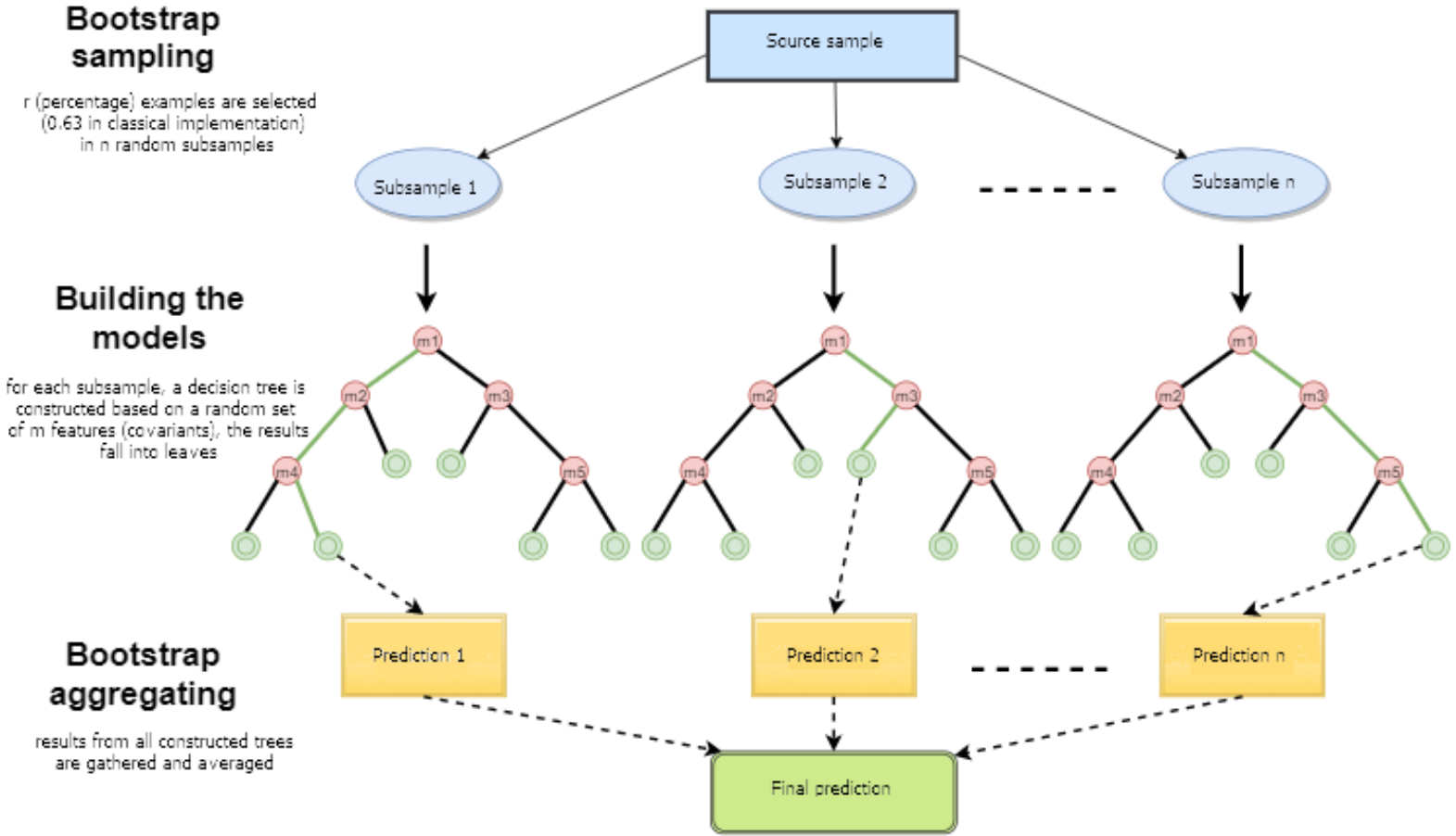


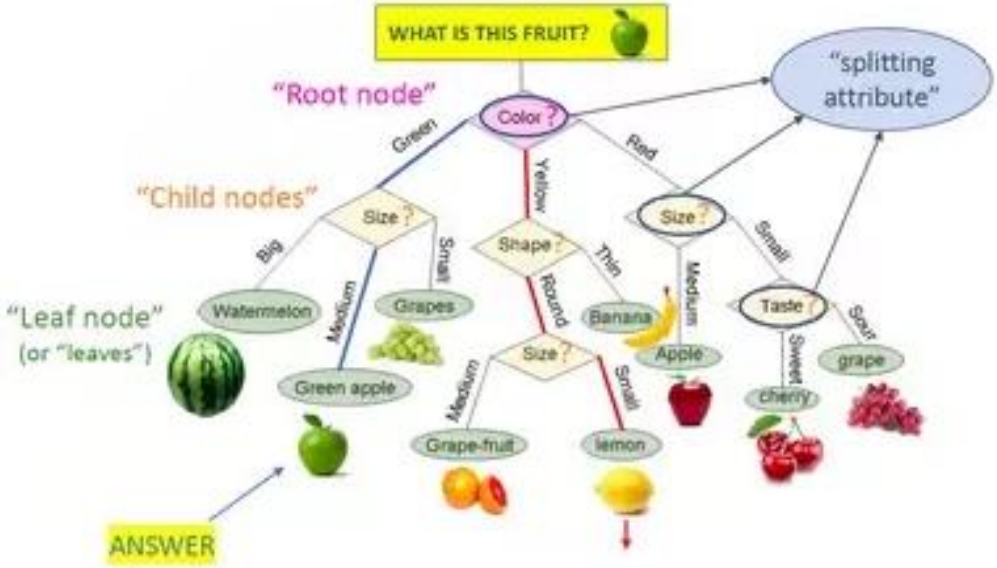


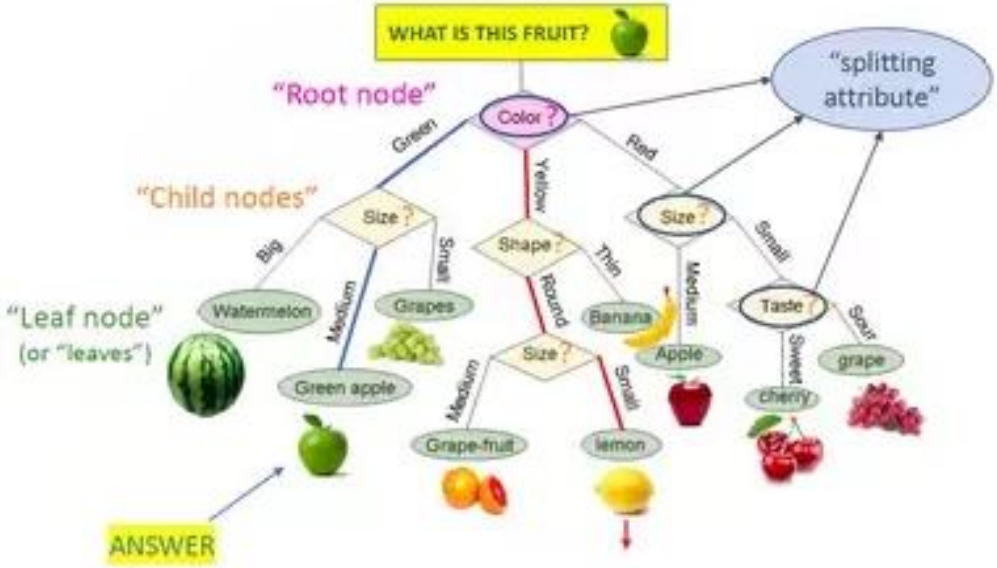
"An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants"
Eric Bauer & Ron Kohavi, Machine Learning 36, 105–139 (1999)

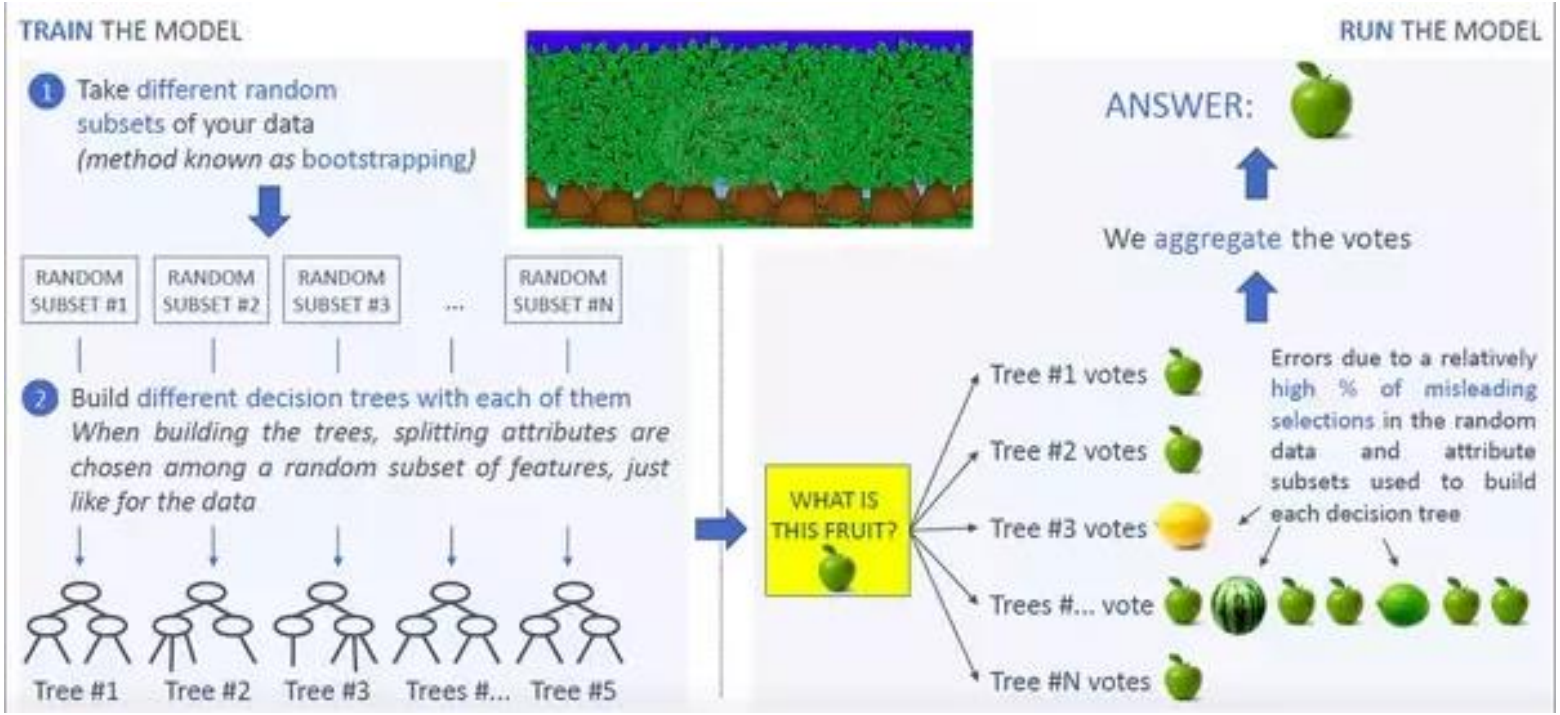
- Modifications of the training set can be generated by
  - Resampling the dataset. By bootstrap sampling (e.g. bagging), weighted sampling (e.g. boosting).
  - Altering the attributes: The base learners are trained using different feature subsets (e.g Random subspaces)

Random Forests



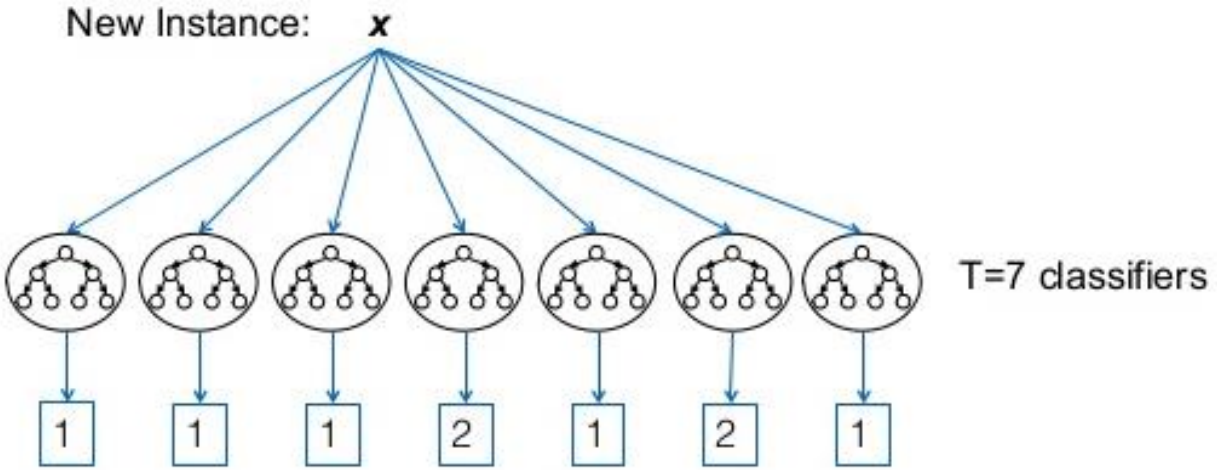

WHAT IS THIS FRUIT?
"Root node"
"splitting attribute"
Color ?
Green
Yellow
Red
"Child nodes"
Size ?
Shape ?
Size ?
Big
Medium
Small
Round
Thin
Medium
Small
"Leaf node"
(or "leaves")
Watermelon
Grapes
Banana
Apple
Taste ?
Green apple
Size ?
Sweet
Sour
grape
cherry
Medium
Small
Grape-fruit
lemon
ANSWER


WHAT IS THIS FRUIT?
"Root node"
"splitting attribute"
Color ?
Green
Yellow
Red
"Child nodes"
Size ?
Shape ?
Size ?
Big
Medium
Small
Round
Thin
Medium
Small
"Leaf node"
(or "leaves")
Watermelon
Grapes
Banana
Taste ?
Apple
Green apple
Grape-fruit
lemon
Sweet
Sour
cherry
grape
ANSWER


TRAIN THE MODEL
RUN THE MODEL
1 Take different random subsets of your data (method known as bootstrapping)
RANDOM SUBSET #1
RANDOM SUBSET #2
RANDOM SUBSET #3
...
RANDOM SUBSET #N
2 Build different decision trees with each of them
When building the trees, splitting attributes are chosen among a random subset of features, just like for the data
Tree #1
Tree #2
Tree #3
Trees #...
Tree #5
WHAT IS THIS FRUIT?
Tree #1 votes
Tree #2 votes
Tree #3 votes
Trees #... vote
Tree #N votes
Errors due to a relatively high % of misleading selections in the random data and attribute subsets used to build each decision tree
We aggregate the votes
ANSWER:

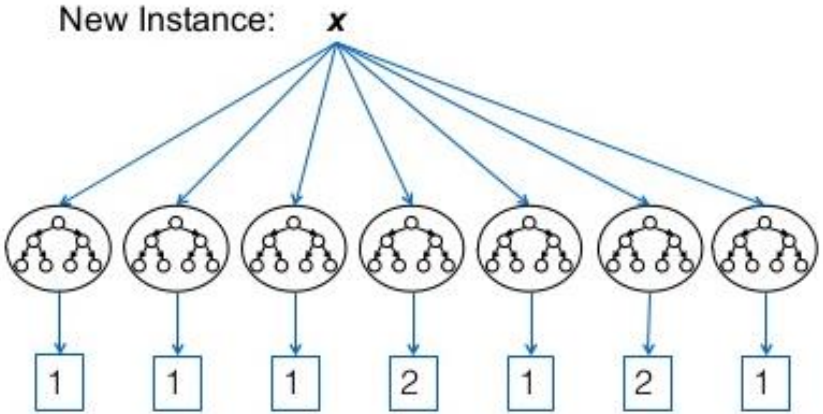
Ensemble Learning

- An ensemble is a combination of classifiers that output a final classification.

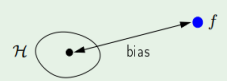


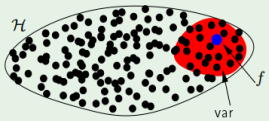
- **Ensemble methods that minimize variance**
  - **Classifier Bagging**
  - **Classifier + Feature Bagging (e.g. Random Forests)**

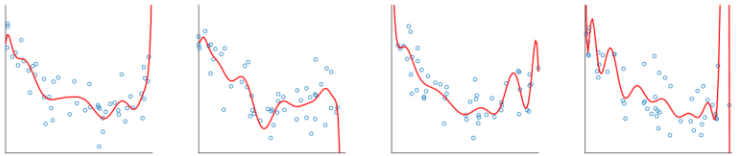
$$\mathbb{E}_{\mathcal{D}}\left[\left(g^{(\mathcal{D})}(\mathbf{x}) - f(\mathbf{x})\right)^2\right] = \underbrace{\mathbb{E}_{\mathcal{D}}\left[\left(g^{(\mathcal{D})}(\mathbf{x}) - \bar{g}(\mathbf{x})\right)^2\right]}_{\text{var}(\mathbf{x})} + \underbrace{\left(\bar{g}(\mathbf{x}) - f(\mathbf{x})\right)^2}_{\text{bias}(\mathbf{x})}$$



$$\text{bias} = \mathbb{E}_{\mathbf{x}}\left[\left(\bar{g}(\mathbf{x}) - f(\mathbf{x})\right)^2\right] \qquad \text{var} = \mathbb{E}_{\mathbf{x}}\left[\mathbb{E}_{\mathcal{D}}\left[\left(g^{(\mathcal{D})}(\mathbf{x}) - \bar{g}(\mathbf{x})\right)^2\right]\right]$$



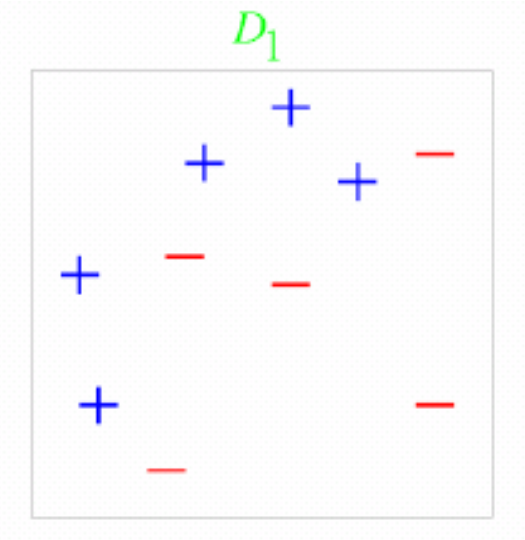




How to build them?

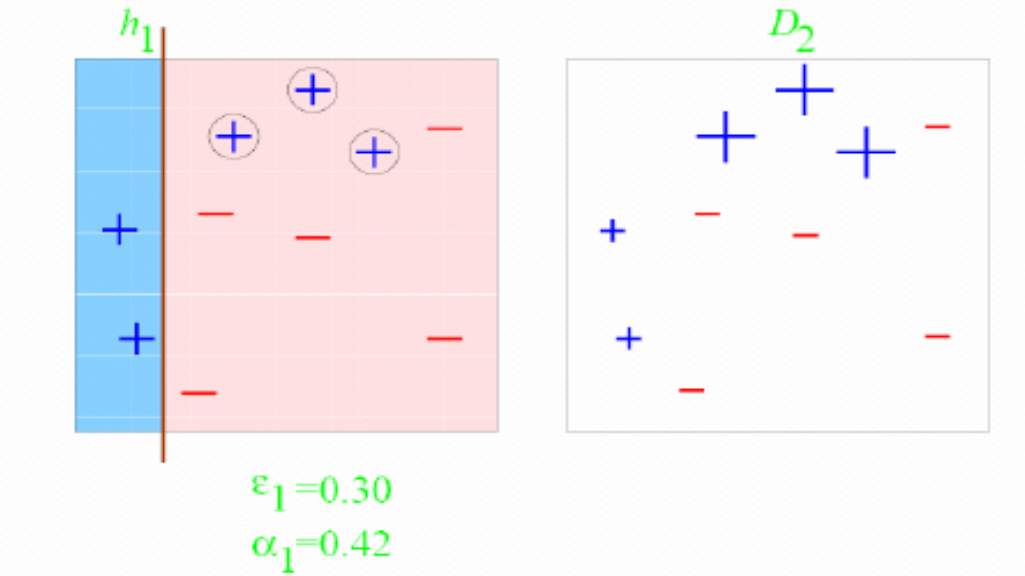
- There are several techniques to build diverse base learners in an ensemble:
  - Use modified versions of the training set to train the base learners
  - Introduce changes in the learning algorithms

A toy example[2]



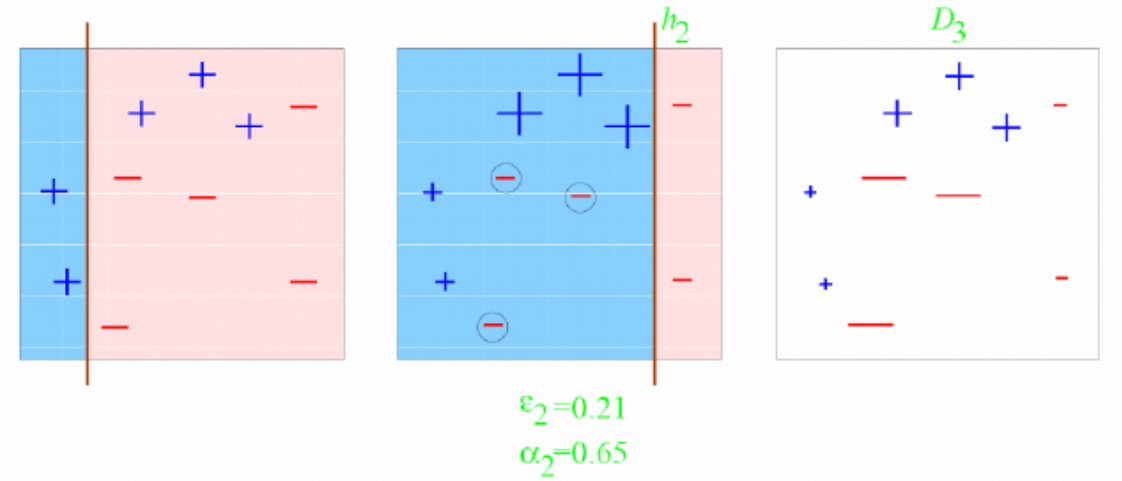
Training set: 10 points
(represented by plus or minus)
Original Status: Equal Weights for all training samples

A toy example(cont'd)



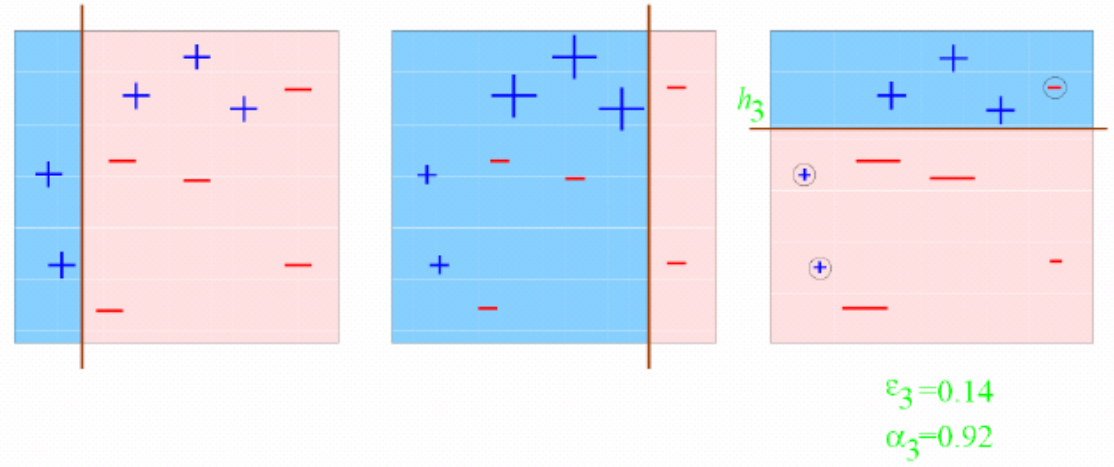
Round 1: Three "plus" points are not correctly classified;
They are given higher weights.

A toy example(cont'd)



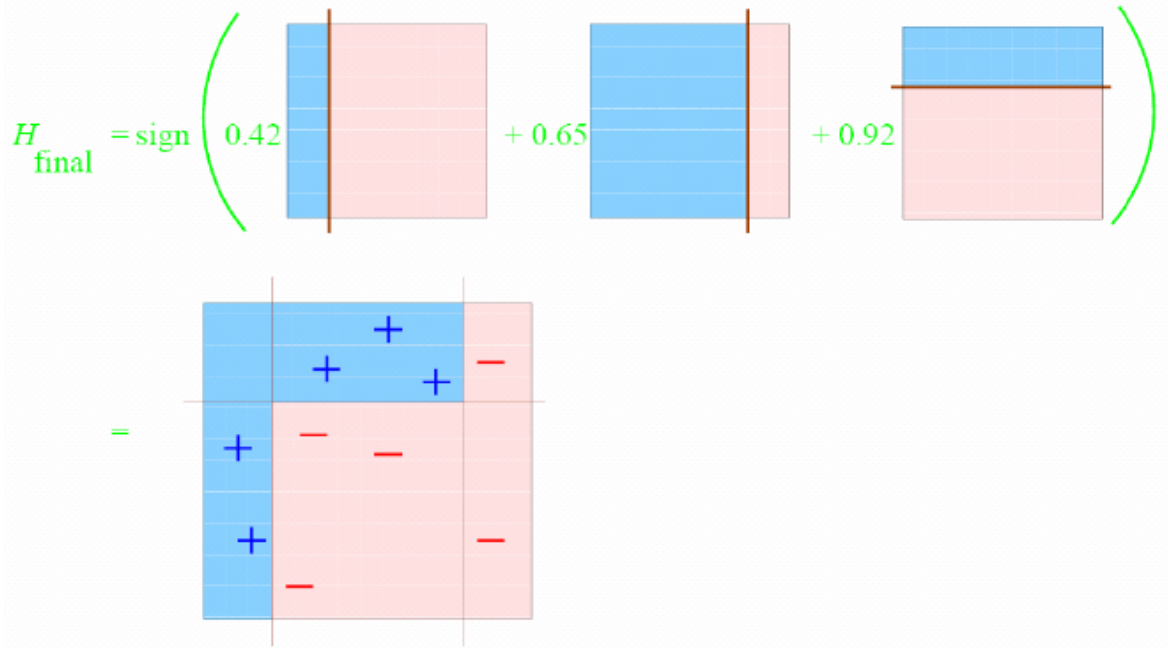
Round 2: Three "minus" points are not correctly classified;
They are given higher weights.

A toy example(cont'd)



Round 3: One "minus" and two "plus" points are not correctly classified; They are given higher weights.

A toy example(cont'd)



Final Classifier: integrate the three "weak" classifiers and obtain a final strong classifier.

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in \mathcal{X}$, $y_i \in \{-1, +1\}$.

Initialize $D_1(i) = 1/m$ for $i = 1, \ldots, m$. ← Initial Distribution of Data

For $t = 1, \ldots, T$:

- Train weak learner using distribution D_t.
- Get weak hypothesis $h_t : \mathcal{X} \to \{-1, +1\}$. ← Train model
- Aim: select h_t with low weighted error:

$$\varepsilon_t = \Pr_{i \sim D_t}\left[h_t(x_i) \neq y_i\right].$$ ← Error of model

- Choose $\alpha_t = \frac{1}{2} \ln\left(\frac{1 - \varepsilon_t}{\varepsilon_t}\right)$. ← Coefficient of model
- Update, for $i = 1, \ldots, m$:

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$ ← Update Distribution

where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

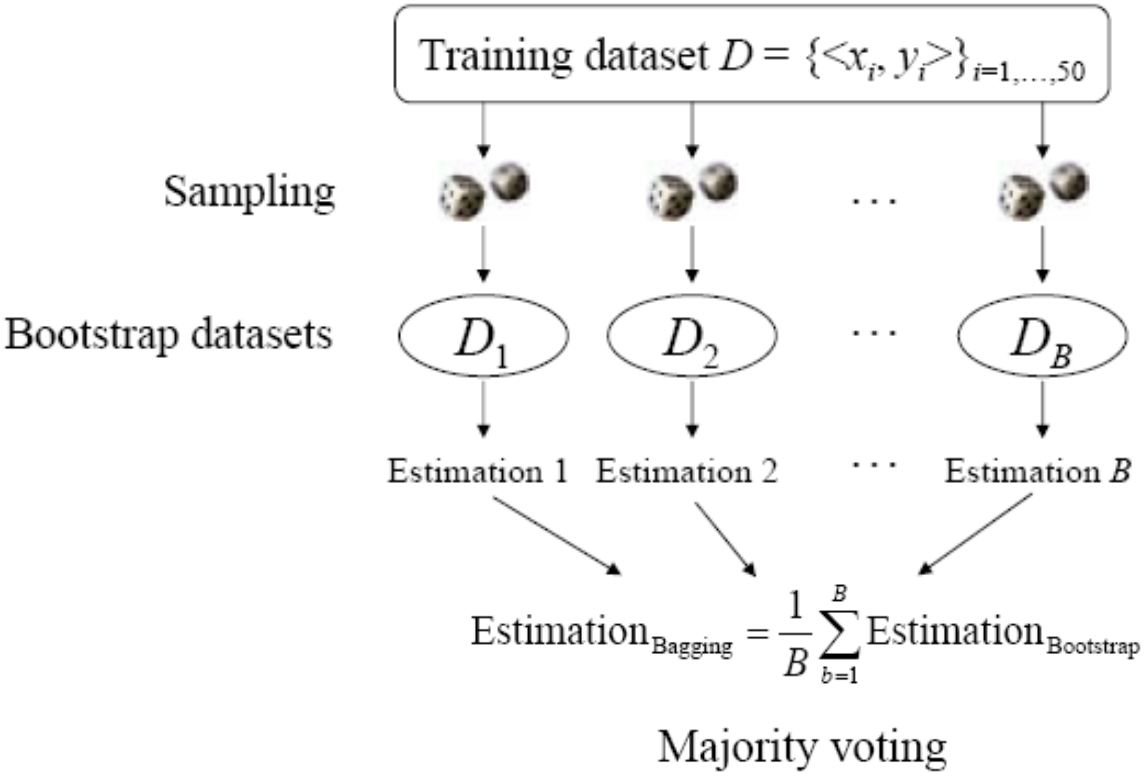
Output the final hypothesis:

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right).$$ ← Final average

Theorem: training error drops exponentially fast

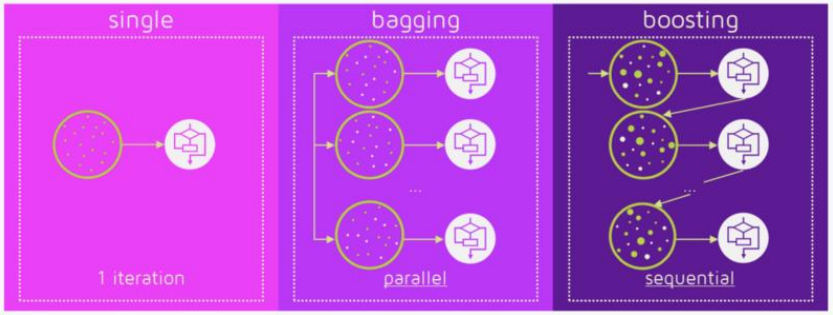
https://www.cs.princeton.edu/~schapire/papers/explaining-adaboost.pdf

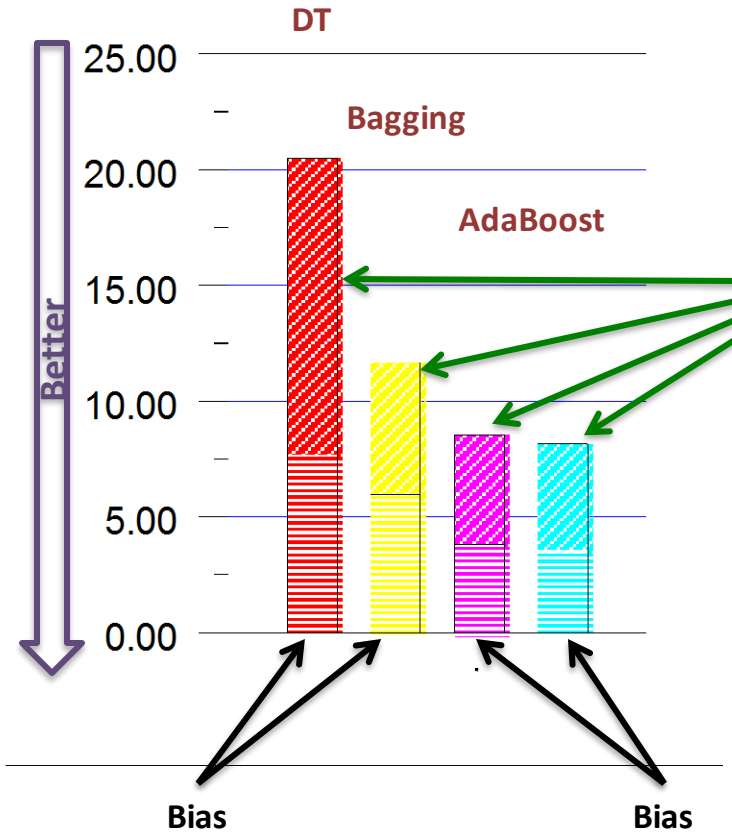
Revisit Bagging



Bagging *vs* Boosting

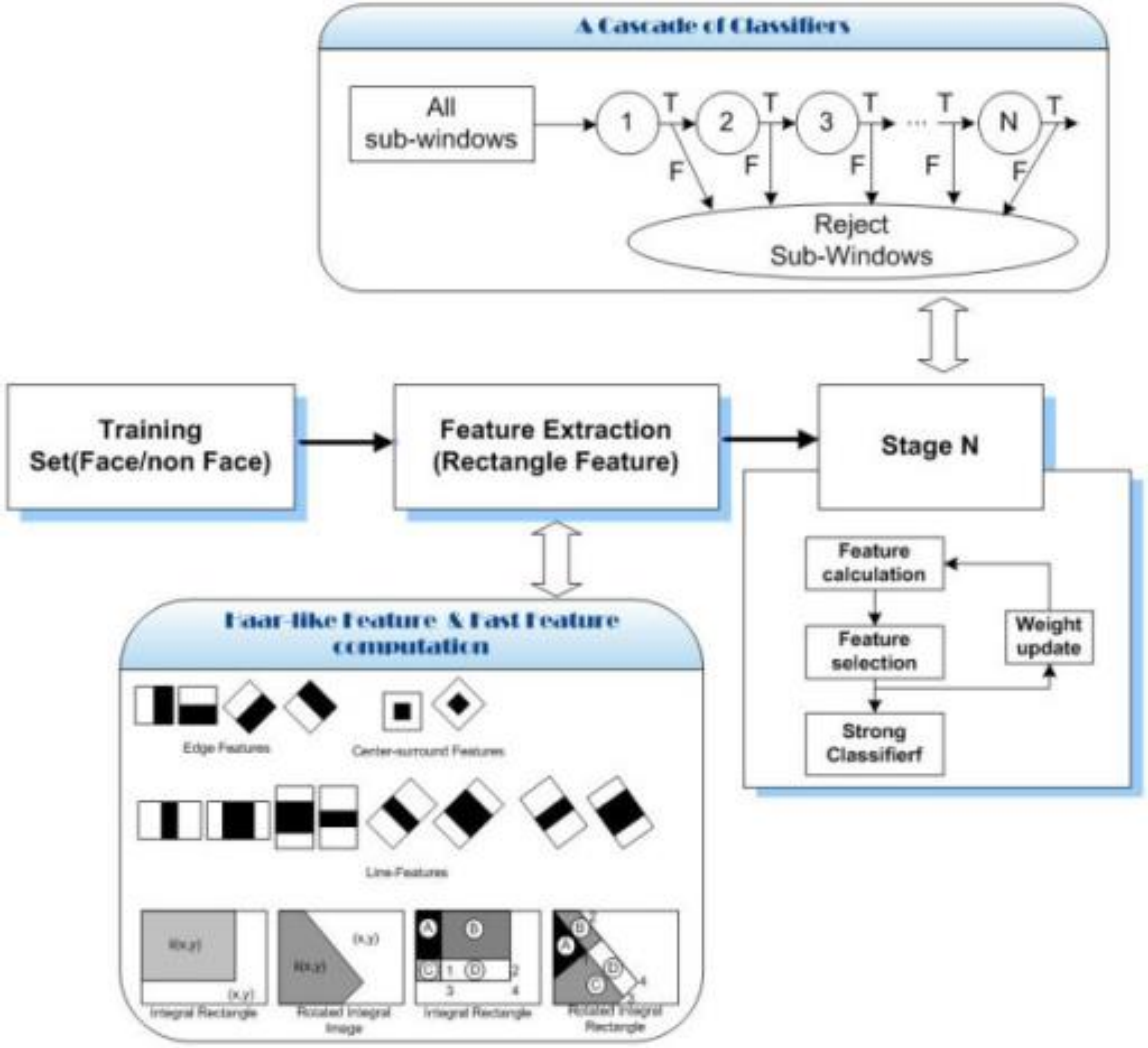
- Bagging
  - Construction of complementary base-learners is left to chance
  - .. and to the unstability of the learning methods.
- Boosting
  - Actively seek to generate complementary base-learner
  - Training the next base-learner based on the mistakes of the previous learners.

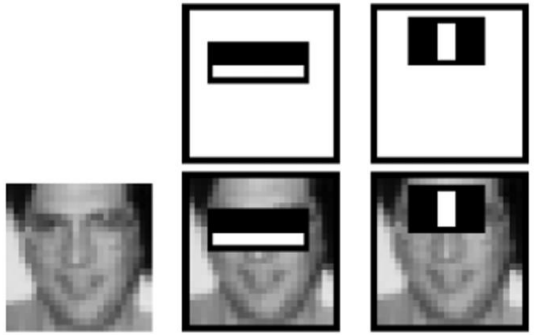


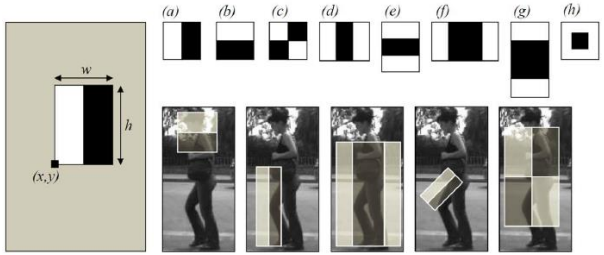


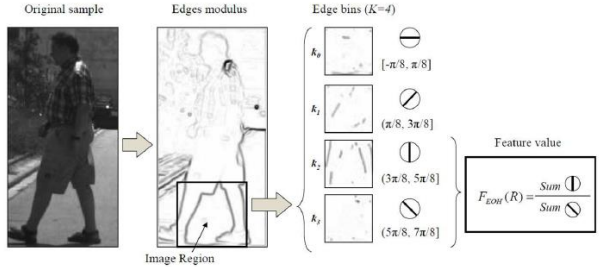
Boosting often uses weak models
E.g, "shallow" decision trees
Weak models have lower variance

"An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants"
Eric Bauer & Ron Kohavi, Machine Learning 36, 105–139 (1999)


A Cascade of Classifiers
All sub-windows
1
2
3
N
T
F
Reject Sub-Windows
Training Set(Face/non Face)
Feature Extraction (Rectangle Feature)
Stage N
Feature calculation
Feature selection
Weight update
Strong Classifierf
Haar-like Feature & Fast Feature computation
Edge Features
Center-surround Features
Line-Features
Integral Rectangle
Rotated Integral Image
Integral Rectangle
Rotated Integral Rectangle




(a)
(b)
(c)
(d)
(e)
(f)
(g)
(h)
w
h
(x,y)


Original sample
Edges modulus
Edge bins (K=4)
[-π/8, π/8]
(π/8, 3π/8]
(3π/8, 5π/8]
(5π/8, 7π/8]
Feature value
Image Region

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in \mathcal{X}$, $y_i \in \{-1, +1\}$.

Initialize $D_1(i) = 1/m$ for $i = 1, \ldots, m$. ← Initial Distribution of Data

For $t = 1, \ldots, T$:

- Train weak learner using distribution D_t.
- Get weak hypothesis $h_t : \mathcal{X} \to \{-1, +1\}$. ← Train model
- Aim: select h_t with low weighted error:

$$\varepsilon_t = \Pr_{i \sim D_t}\left[h_t(x_i) \neq y_i\right].$$ ← Error of model

- Choose $\alpha_t = \frac{1}{2} \ln\left(\frac{1 - \varepsilon_t}{\varepsilon_t}\right)$. ← Coefficient of model
- Update, for $i = 1, \ldots, m$:

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$ ← Update Distribution

where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis:

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right).$$ ← Final average

Adaboost: Additive model with exponential loss function

$$\min_{\alpha_{n=1:N}, \beta_{n=1:N}} L\left(y, \sum_{n=1}^{N} \alpha_n f(x, \beta_n)\right)$$

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in \mathcal{X}$, $y_i \in \{-1, +1\}$.

Initialize $D_1(i) = 1/m$ for $i = 1, \ldots, m$. ← Initial Distribution of Data

For $t = 1, \ldots, T$:

- Train weak learner using distribution D_t.
- Get weak hypothesis $h_t : \mathcal{X} \to \{-1, +1\}$. ← Train model
- Aim: select h_t with low weighted error:

$$\varepsilon_t = \Pr_{i \sim D_t}\left[h_t(x_i) \neq y_i\right].$$ ← Error of model

- Choose $\alpha_t = \frac{1}{2} \ln\left(\frac{1 - \varepsilon_t}{\varepsilon_t}\right)$. ← Coefficient of model
- Update, for $i = 1, \ldots, m$:

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$ ← Update Distribution

where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution).

Output the final hypothesis:

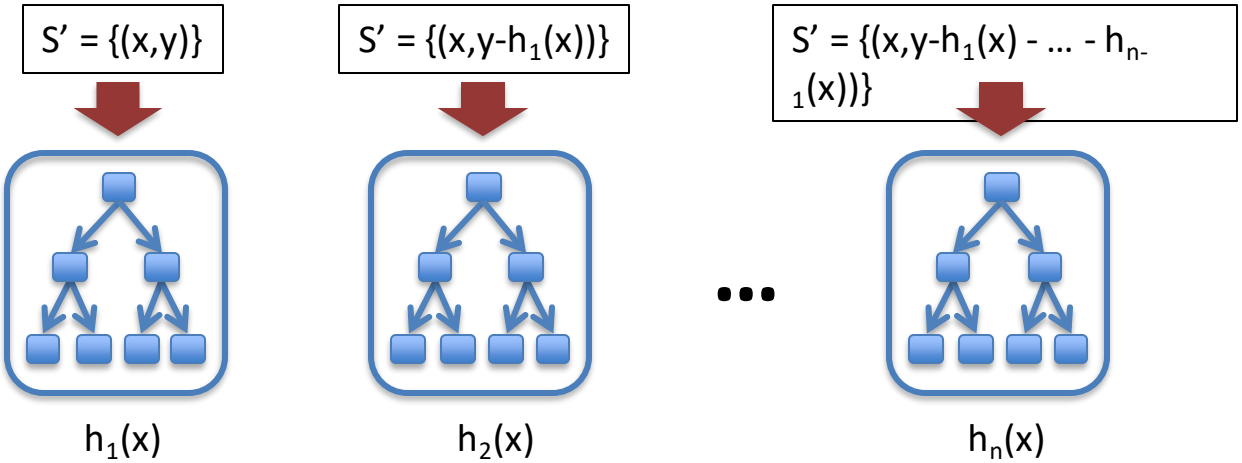
$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right).$$ ← Final average

Adaboost: Additive model with exponential loss function

$$\min_{\alpha_{n=1:N}, \beta_{n=1:N}} L\left(y, \sum_{n=1}^{N} \alpha_n f(x, \beta_n)\right) \Rightarrow \min_{\alpha_n, \beta_n} L\left(y, f_{n-1}((x) + \alpha_n f_n(x, \beta_n)\right)$$

Functional Gradient Descent

$$h(x) = h_1(x) + h_2(x) + \ldots + h_n(x)$$



Gradient Boosting

1. Start with a constant model f_0
2. Fit a weak learner h_n to the negative gradient of the loss function w.r.t. f_{n-1}
3. Take a step γ s.t. $f_n = f_{n-1} + \gamma h_n$ minimizes the loss $L\left(y, f_n(x)\right)$

Adaboost: Additive model with exponential loss function

Gradient Boost: Adaboost w/ Generic loss function

- Ensemble methods that minimize variance
  - Bagging
  - Random Forests
- **Ensemble methods that minimize bias**
  - **Boosting**

Combine bagging and boosting ?

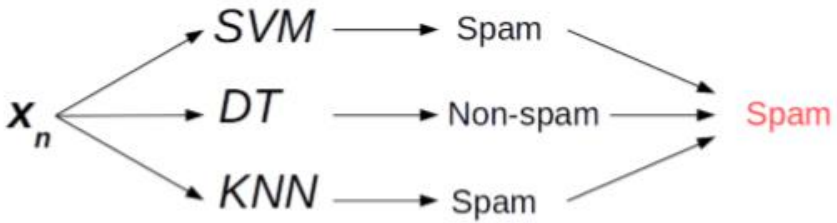
- Yes ! Infinite Boosting
- .. But time-consumsing
- https://arxiv.org/abs/1706.01109
- Code: https://github.com/arogozhnikov/infiniteboost

Some Practical Advice

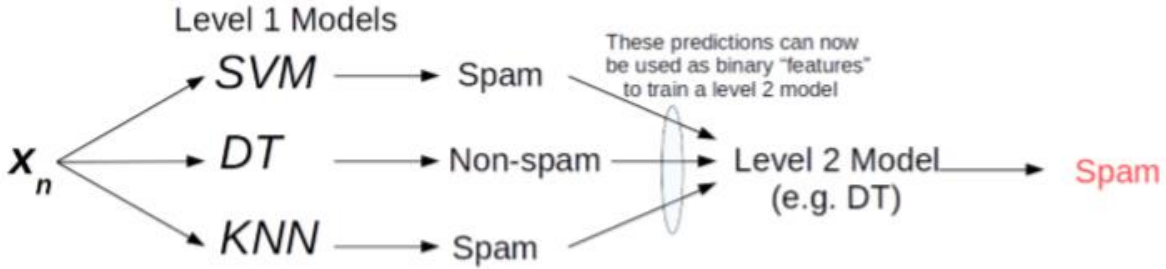
- If the classifier is unstable (high variance), then apply bagging!
- If the classifier is stable and simple (high bias), then apply boosting!
- If the classifier is stable and complex then apply randomization injection!

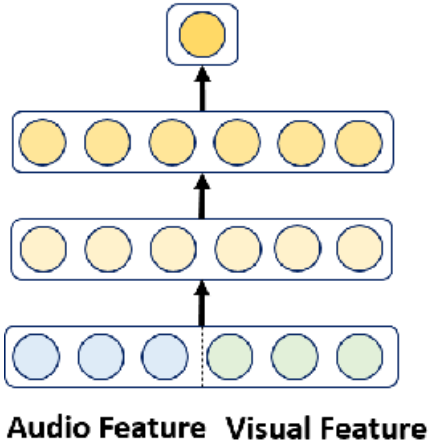
Train different models on same data

- Voting or Averaging of predictions of multiple pre-trained models

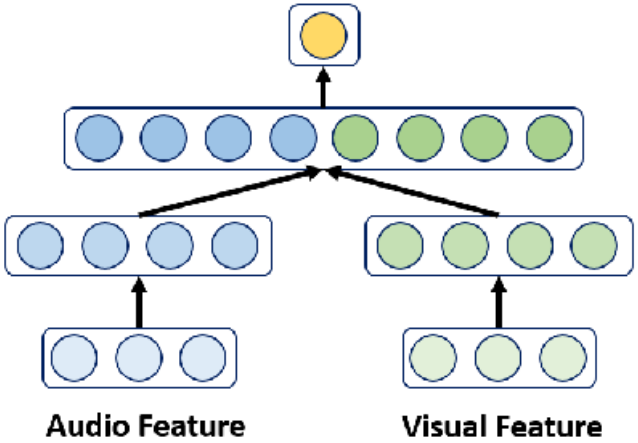


- "Stacking": Use predictions of multiple models as "features" to train a new model and use the new model to make predictions on test data

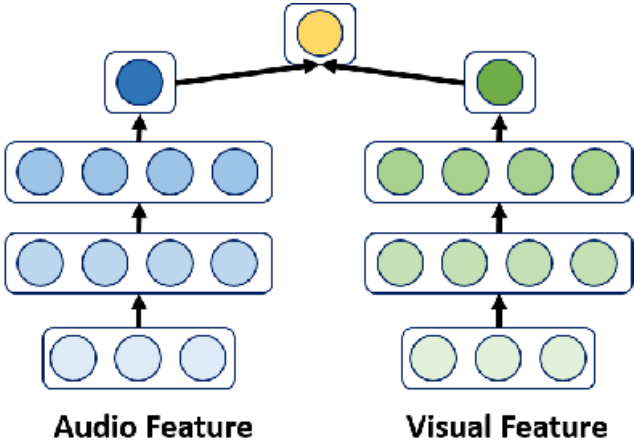




(a) Early Fusion



(b) Model-level Fusion



(c) Late Fusion

Eyes detector

Mouth detector

Ears detector

Skin detector

Meta Classifier

High accuracy!

Low individual accuracy
Computationally efficient

References

- https://cedar.buffalo.edu/~srihari/CSE555/Boosting.pdf