To cite this presentation: Pendyala, V.S. (2022) "Meta-algorithms in Machine Learning". PPT Presentation. IEEE Computer Society, Santa Clara Valley Chapter Webinar.

Video Recording: https://ieeetv.ieee.org/video/meta-algorithms-in-machine-learning

Meta-algorithms in Machine Learning

Vishnu S. Pendyala, PhD

How do we make the best of both?

PARAMETRIC MODELS LIKE LOGISTIC REGRESSION ARE IMPACTED BY BIAS

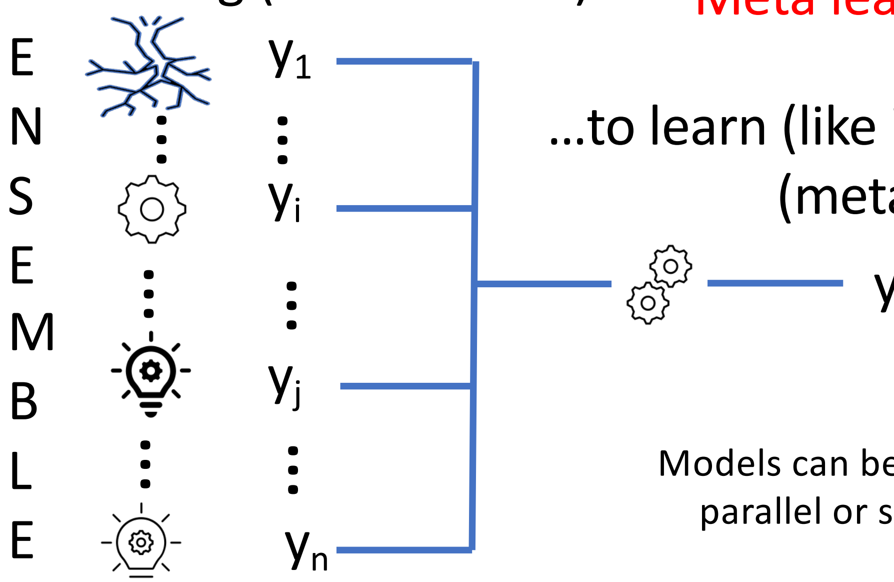
SOME OTHERS LIKE DECISION TREES ARE IMPACTED BY VARIANCE

How do you reduce the risk arising from fluctuations (variance) in a stock's price?

Diversify!

Learning (like in school)…

Meta learning



…to learn (like in industry)
(meta)

Models can be generated in parallel or sequentially

Base (often, weak) learners

Meta Learning

- Learning is never perfect – incurs a loss
- Meta learning minimizes the losses from the previous round of learning
- Meta learning is a big deal for deep learning – few shot learning and more
- Deep learning itself is a kind of meta learning

The rationale

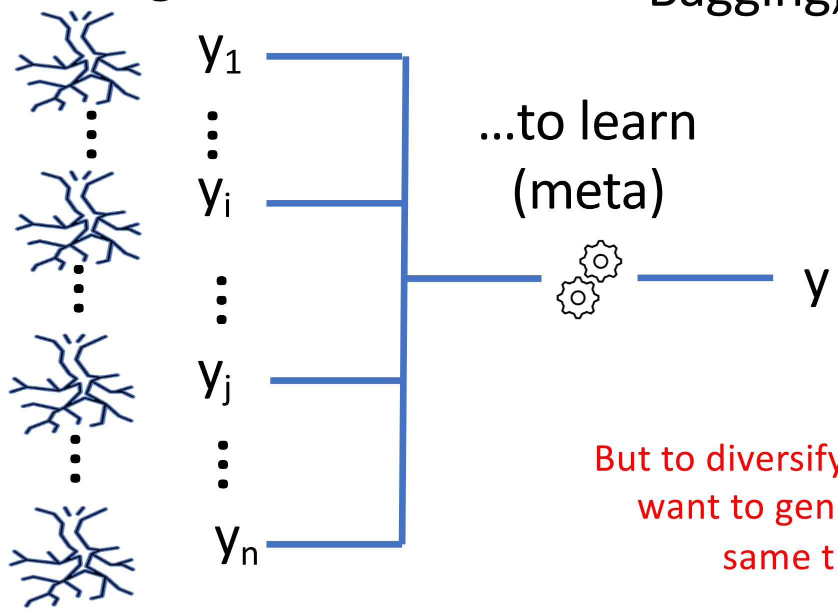
- **Consider 256 weak learners for binary classification, $y_i \in \{-1, +1\}$**
- **Each uncorrelated classifier is a weak learner with error rate, say 0.45**
- **For the ensemble to make a misclassification, majority (in this case 129 or more) base learners must misclassify**
- **This is like the coin-toss experiment, so we use binomial distribution for the probability of the ensemble making a misclassification**

$P = P_{129} + P_{130} + \ldots + P_{256}$ where $P_i = \binom{256}{i} p^i q^{256-i}$ p = 0.45 and q = 0.55

Cumulative probability, $P = \sum_{i=129}^{256} \binom{256}{i} p^i q^{256-i}$ = 0.04

Learning…

Bagging, Adaboost



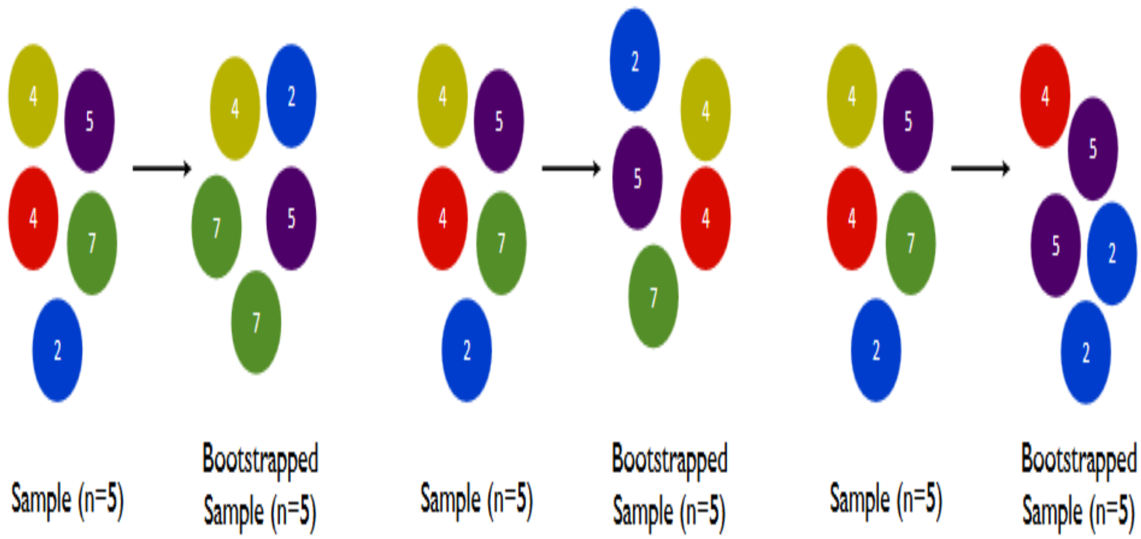
But to diversify, we do not want to generate the same tree!

How can we generate different decision trees from the same training dataset?

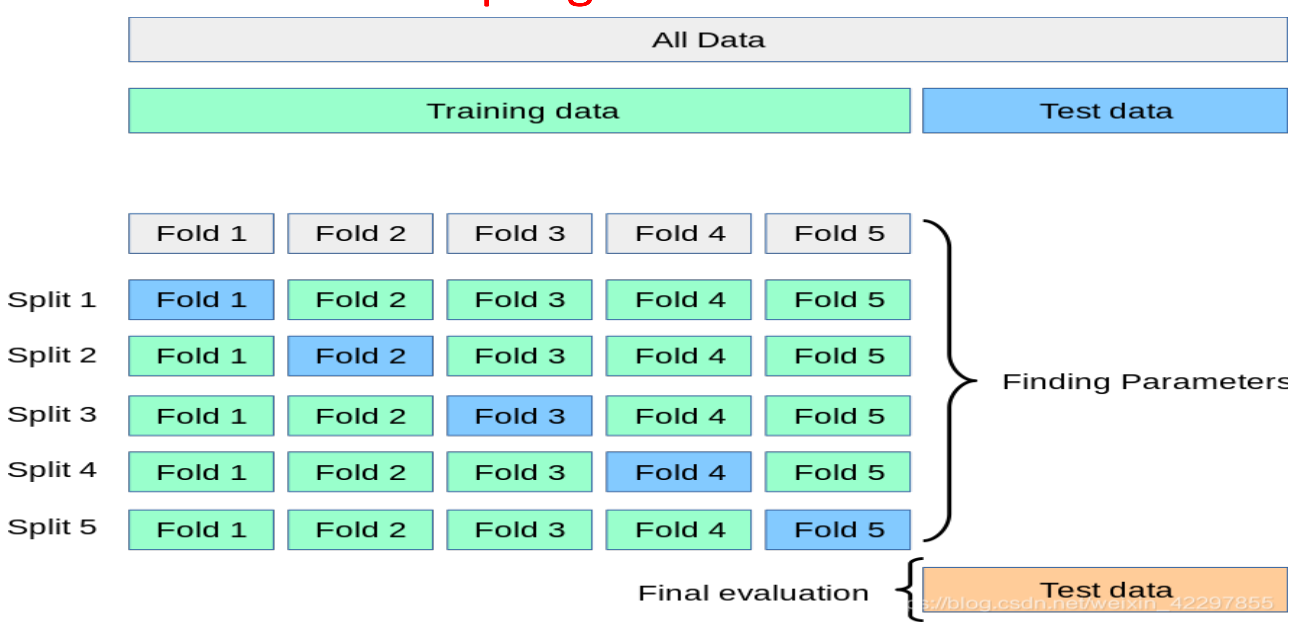
- Perturb X or Perturb Y
- Perturb X in two ways: row-wise or column-wise – randomly, no bias!
- Perturbation of X can be via
  - bootstrap sampling
  - k-fold sampling
  - weighted sampling
  - random subspaces

x_1	x_2	…	x_n	Y

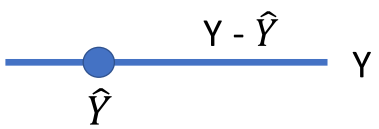
Bootstrap Sampling



K-fold Sampling for Cross Validation

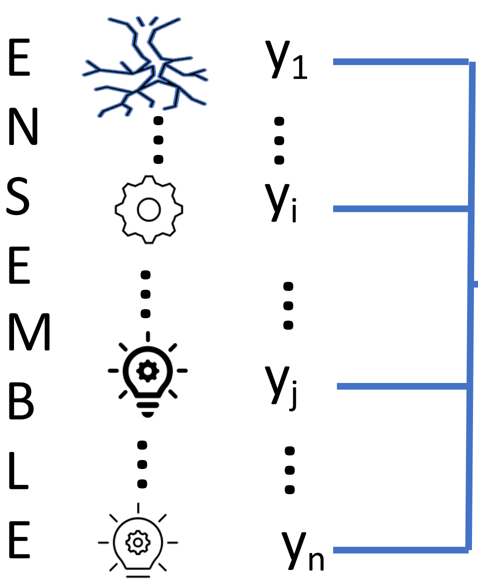


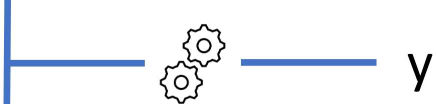
How can we perturb Y?



- What aspect of Y typically changes with each iteration on a model in an ensemble?
  - The predicted value $\hat{Y}$
- Suppose we need to reach a target Y taking several steps (iterations).
- We can know where we are, as an absolute value $\hat{Y}$ or as a measure for how far (Y - $\hat{Y}$) we are from the target Y
- What if we use (Y - $\hat{Y}$) as the target variable instead of Y?
- Each iteration, we get new values for the target variable and Y is perturbed => we get a different model each time.
- Instead of trying to predict Y, we predict how far we are from Y; i.e. we fit the residuals instead of the target value.

Next Question: How can we aggregate the models?





- Majority vote (mode)
- Average (mean)
- Weighted response
- Metamodel

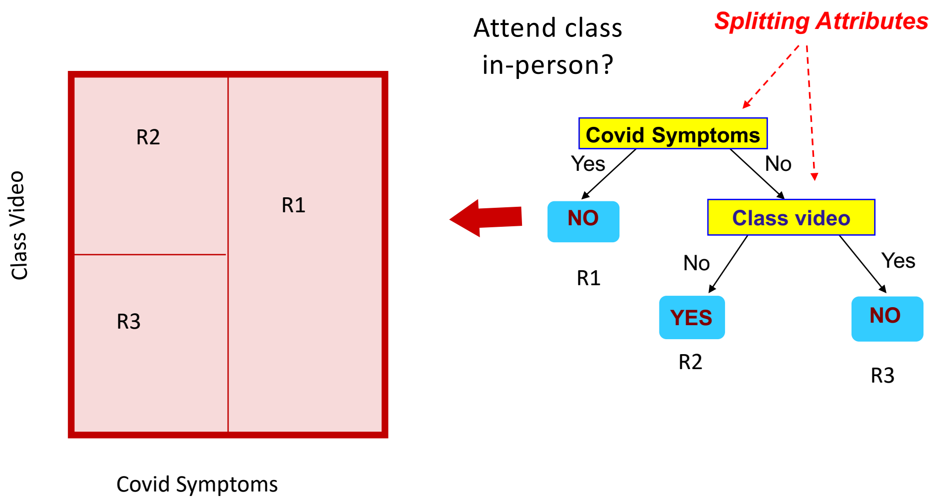
Dataset Perturbations and response aggregation for model ensembling

Ensemble Method	Perturb X row-wise using	Perturb X Column-wise using	Perturb Y using	Model generation	Aggregation strategy
Bagging	Bootstrap Sampling	None	None	In parallel	Mean or mode
Random Forest	Bootstrap Sampling	Random subspaces (at each tree/node)	None	In parallel	Mean or mode
Adaboost	Bootstrap Sampling with weighting	None	None	Sequential	Weighted response
Gradient Boosting	None	None	Pseudo-residuals	Sequential	Weighted response
Stacking	None (or) K-fold sampling	None	None	In parallel	Metamodel

Next question: Where do we start?

High bias or high variance?

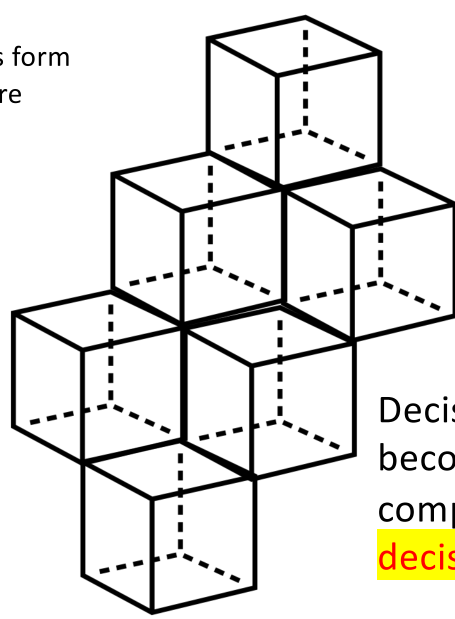
Decision tree divides the instance space into regions
High Variance, Complex Decision boundary



Complex Decision Boundary

Decision Trees form regions that are ==**hypercubes**==

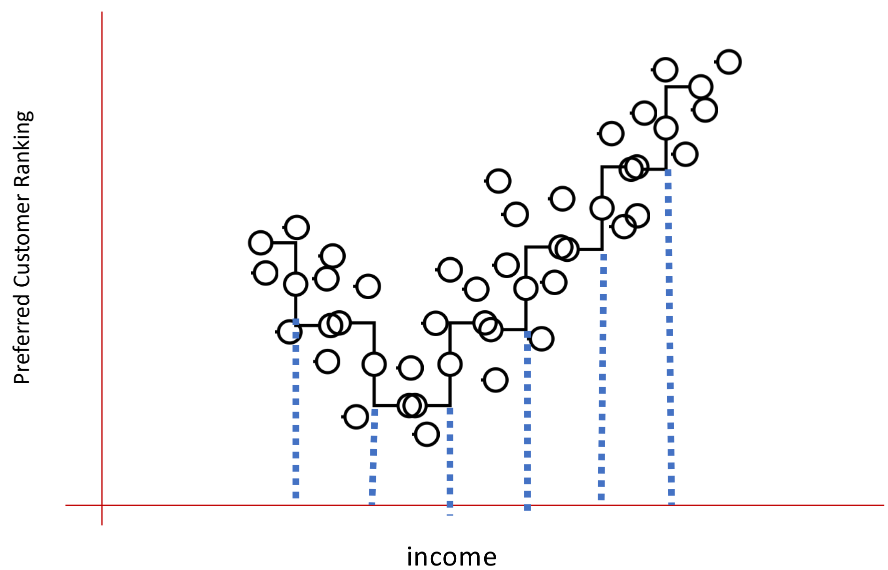
Hypercubes are formed based on ==**decision splits**==



We can start with high variance as in decision trees

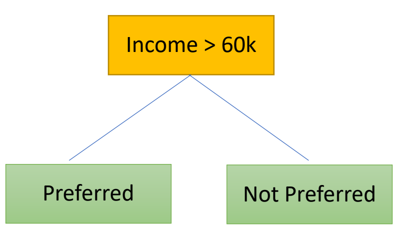
Decision boundary becomes increasingly complex as we add ==decision stumps==

High Complexity of the Decision Boundary

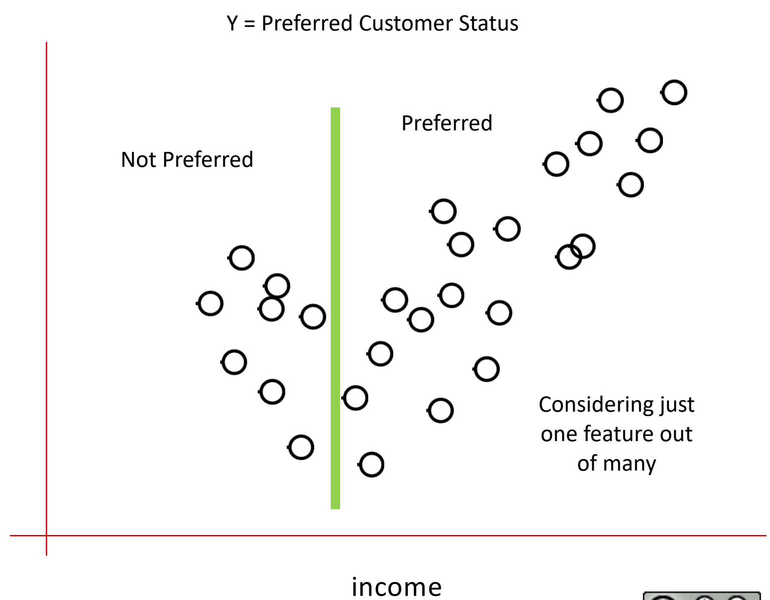


Low Complexity of the Decision Boundary

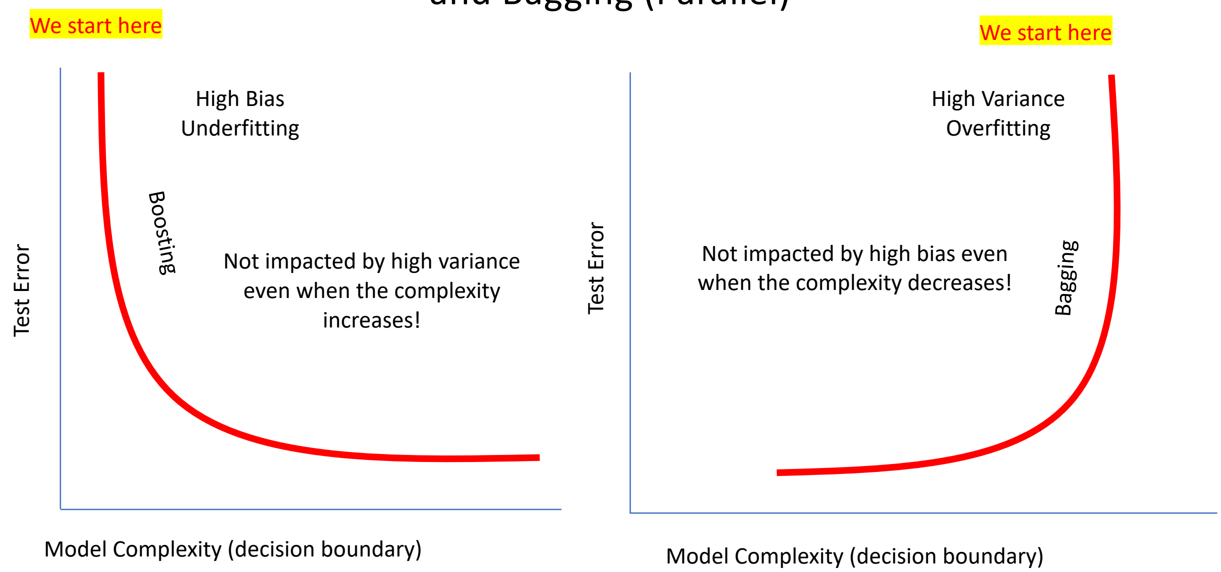
A Decision Stump



OR we can start with high bias as in a decision stump

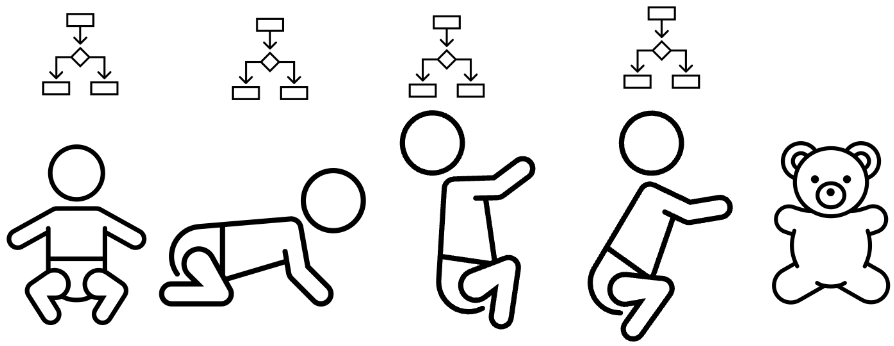


Two Approaches to Ensembles: Boosting (Sequential) and Bagging (Parallel)



Boosting

Aggregating the model responses using weights



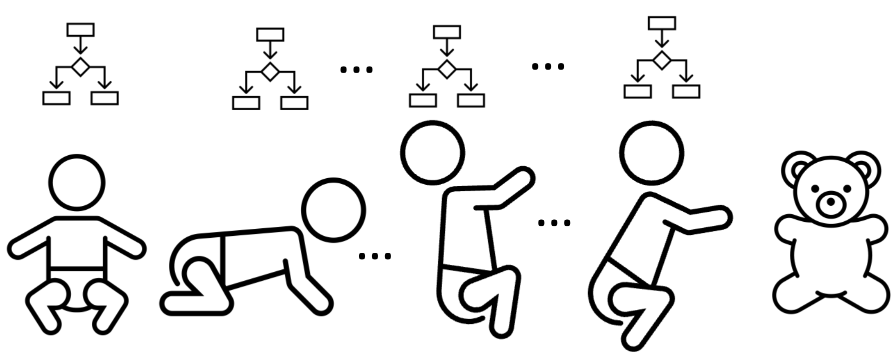
Boosting: Baby steps to loss minimization

Error, loss function, and the initial prediction

- Baby steps => errors at each step. How do we model the error or for the entire dataset, loss?
- The simplest error is the residual, r = (Y - $\acute{Y}$) called the residual
- This is same as the negative gradient of the popular loss function, the squared error, L = (Y - $\acute{Y}$)² ; r = - $\frac{1}{2}$ $(\frac{\partial L}{\partial \acute{Y}})$
- To be agnostic to the loss function, $(\frac{\partial L}{\partial \acute{Y}})$, which may not always be in the form of a residual (Y - $\acute{Y}$) , is called pseudo-residual.
- For the loss to be minimum, $(\frac{\partial L}{\partial \acute{Y}}) = 0$
- For squared error, $\sum_{i=1}^{N}$ (Yi – $\acute{Y}_i$) = 0 => if we have to start with a good estimate for $\acute{Y}_i$ for all data items, N* $\acute{Y}_i$ = $\sum_{i=1}^{N}$ Y_i
- First baby step is a single leaf with the mean of target values – not even a stump!

Baby steps summing up to the target

- Residuals tell how bad the base learners are and how far the base learner is from the target variable.
- How can we bridge the gap (residual) left by the base learner?
- Why not generate a series of models $h_b(x)$ that try to bridge the gap, by predicting not the final target, but the subsequent residuals?
- The training dataset for $h_b(x)$ at each step is not $\{(x_i, y_i)\}$ but $\{(x_i, y_i - \hat{y}_i)\}$
- Then, $\hat{Y} = H(\mathbf{x}) = \sum_{b=1}^{B} \lambda h_b(x)$ where λ is the regularization parameter to slow the learning process and avoid overfitting.



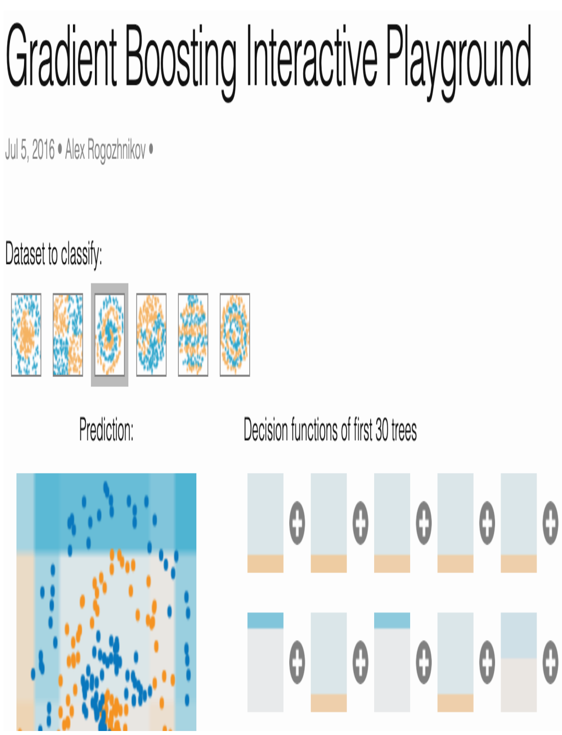
$h_0(x) + \lambda h_1(x) + \lambda h_2(x) + \cdots + \lambda h_i(x) \cdots + \lambda h_B(x) = H(x)$

$r_0 \quad r_1 = r_0 - \lambda h_1(x) \quad \cdots \quad r_i = r_{i-1} - \lambda h_i(x) \quad \cdots$

Boosting: Baby steps to loss minimization

Works for any differentiable loss function

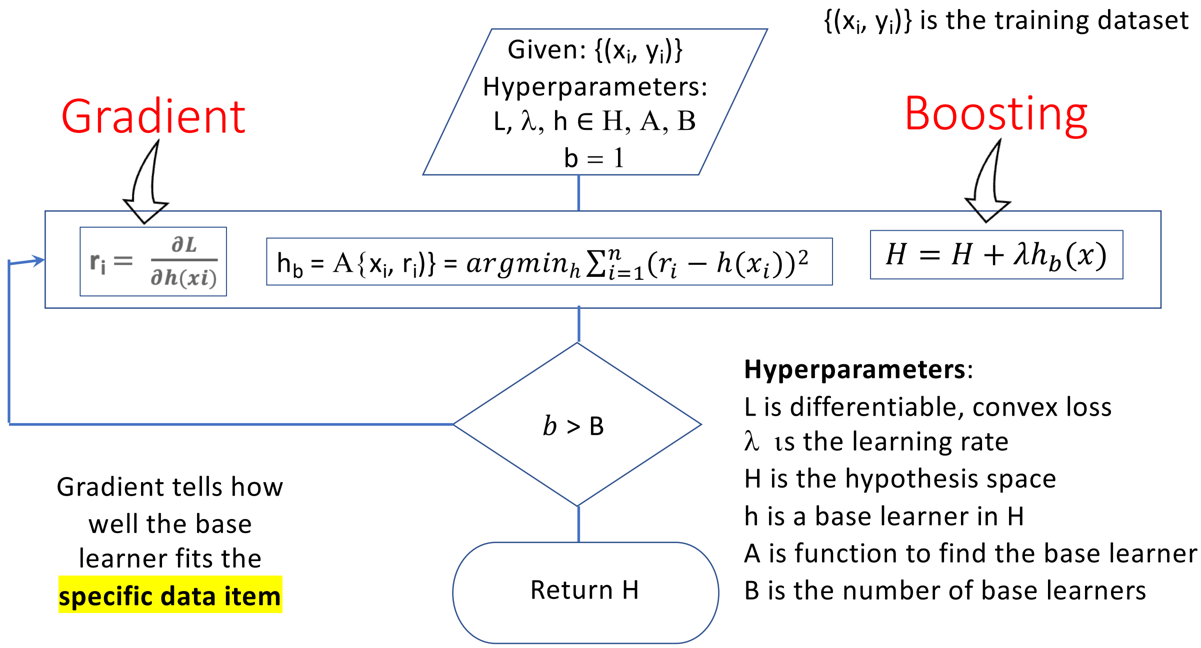
replace r with the pseudo-residual = $(\frac{\partial L}{\partial \hat{Y}})$



Some notes

The base learner model, $h_i(x)$ is typically a decision tree as well and to regularize further, can just be a decision stump (depth=1).

In some sense the pseudo-residual indicates how difficult it is to fit the item using the base learner – can be thought of as a relative weight of the data item



Boosting is yet another Gradient Descent!

Source: Mason, L., Baxter, J., Bartlett, P., & Frean, M. (1999). Boosting algorithms as gradient descent. *Advances in neural information processing systems, 12*.

w → Slope is positive

Hiker must go left on the horizontal axis (decrease w)

w → Slope is negative

Hiker must go right on the horizontal axis (increase w)

How do we express the last two slides in one line in math?

$$w_i = w_i - \lambda \frac{dL}{dw_i}$$

- Derivative is the slope of the loss function
- λ is the length of the stride in the direction of the slope
- If slope is positive, weights decrease and vice versa
- If we extend it to multidimensions, we use gradient instead of derivative

Adaboost

Adaptive boosting

Adaboost vs Gradient boosting

- Both take baby steps; base learners are typically decision stumps
- Instead of shrinking the trees using a λ that is constant for all the decision trees, in Adaboost, we use a varying α that is proportional to how well the base learner performs.

$$\widehat{\boldsymbol{Y}} = \boldsymbol{H}(\mathbf{x}) = \mathbf{sign}(\sum_{\boldsymbol{b}=\mathbf{1}}^{\boldsymbol{B}} \alpha_b h_b(x))$$

- The gradient, which indicated the relative importance of a data item is now used to perturb X row-wise (Adaboost) instead of Y (GB)

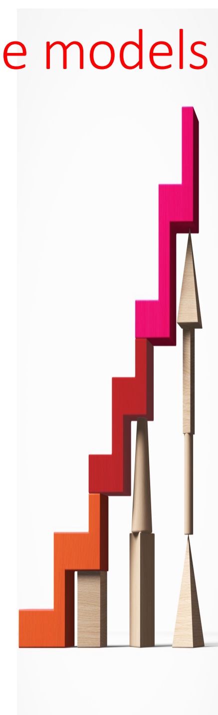
Boosting for Classification: Weighting the base models

- What is the odds ratio of a model classifying correctly with probability p?

$$\frac{p}{1-p} = \frac{1-\epsilon}{\epsilon}$$ => better the model, higher the odds ratio

- Taking logarithm of ratios simplifies operations – multiplications convert to additions, divisions to subtractions
- The logit function, $\ln(\frac{1-\epsilon}{\epsilon})$ converts a probability $p \in (0,1)$ to a number $r \in (-\infty, +\infty)$ and = 2*arctanh(1-2ϵ)
- $\ln(\frac{1-\epsilon}{\epsilon})$ is an indication of how well the base learner can classify, so can be used as a weight, α_b for the base learner

$$\alpha_b = \frac{1}{2}\ln\left(\frac{1-\epsilon}{\epsilon}\right) = \text{arctanh}(1\text{-}2\epsilon)$$



Boosting for Classification: Weighting the data items

- Initially, all samples have the same weight: $\frac{1}{N}$
- We factor in α_b into the subsequent weight updates; α_b is a logarithm, so we use exponentiation:

$$w_i = w_i \ * e^{-\alpha_b y_i h(x_i)}$$

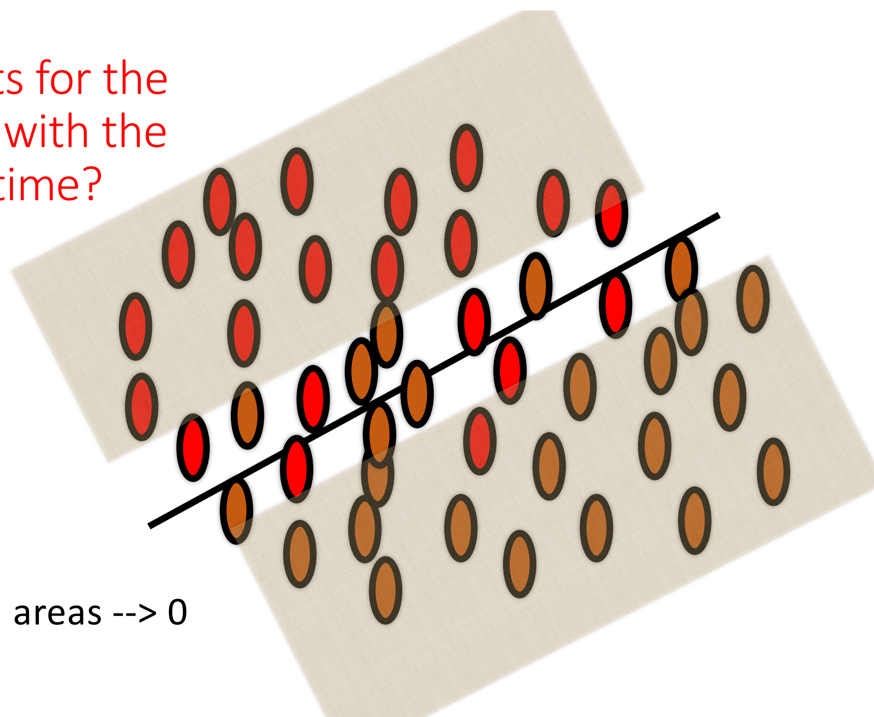
$\Rightarrow$The weight for correctly classified data items, $e^{-\alpha_b}$ is exponentially low and for difficult ones, e^{α_b} is exponentially high

- Weights need to be normalized so that they sum up to 1 and become a probability distribution.

$$\text{Error } \epsilon = \frac{w_i * I(y_i \neq h_b(x_i))}{\sum_{i=1}^{N} w_i}$$

How do the weights for the data items change with the progression of time?

For simplicity, here we are assuming a linear decision boundary



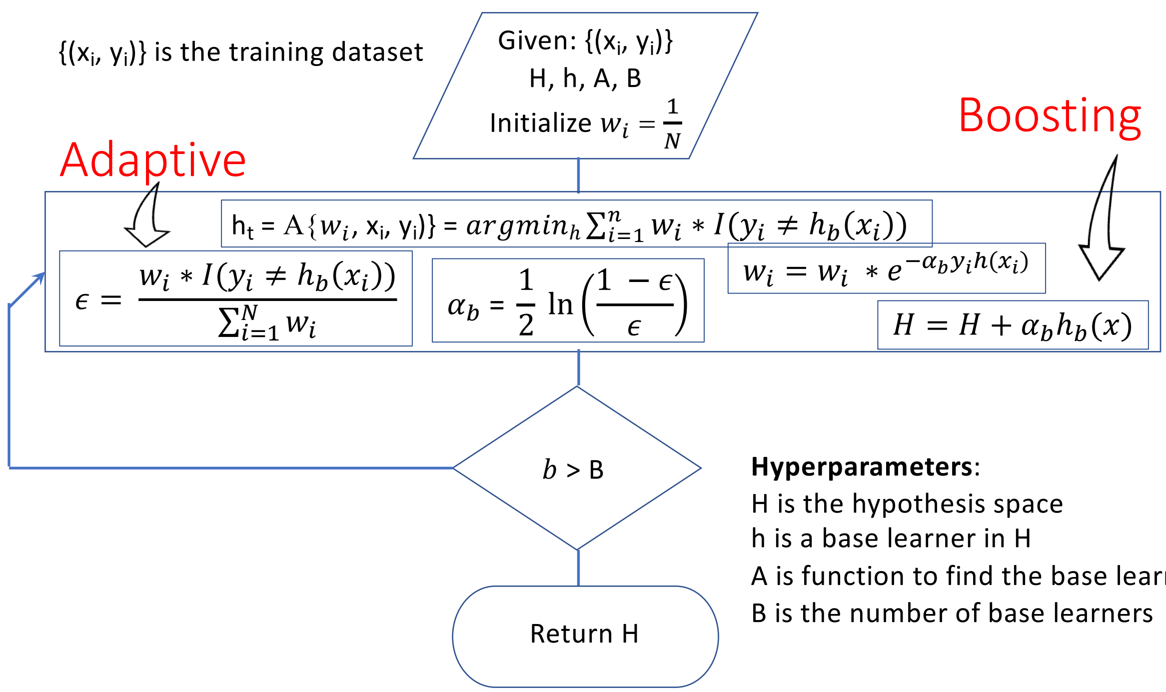
Weights in the shaded areas --> 0

Loss function

- The loss function we are minimizing corresponding to $\alpha_b = \frac{1}{2}\ln\left(\frac{1-\epsilon}{\epsilon}\right)$ and $w_i = w_i * e^{-\alpha_b y_i h(x_i)}$ is exponential:

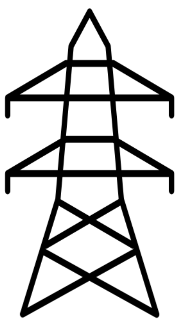
$$\mathcal{L} = \sum_{i=1}^{N} e^{-y_i H(x_i)} \Rightarrow \text{Gradient}, \left(\frac{\partial L}{\partial H(x_i)}\right) = -y_i\, e^{-y_i H(x_i)}$$

- This time, we use the gradient to perturb X instead of Y and incorporate it into the weight update of the data items



Hyperparameters:
H is the hypothesis space
h is a base learner in H
A is function to find the base learner
B is the number of base learners

Many more variants of boosting

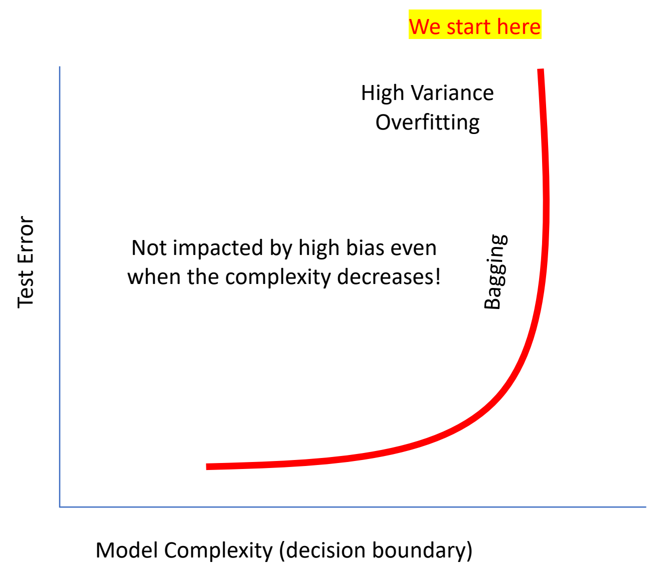


- XGBoost, LightGBM, CatBoost are the popular ones
- Combine several techniques for scaling to big data, faster processing, handling categorical variables, missing values, textual data, etc
- Differ in tree generation, tree types, community support, hyperparameters, and naturally, performance

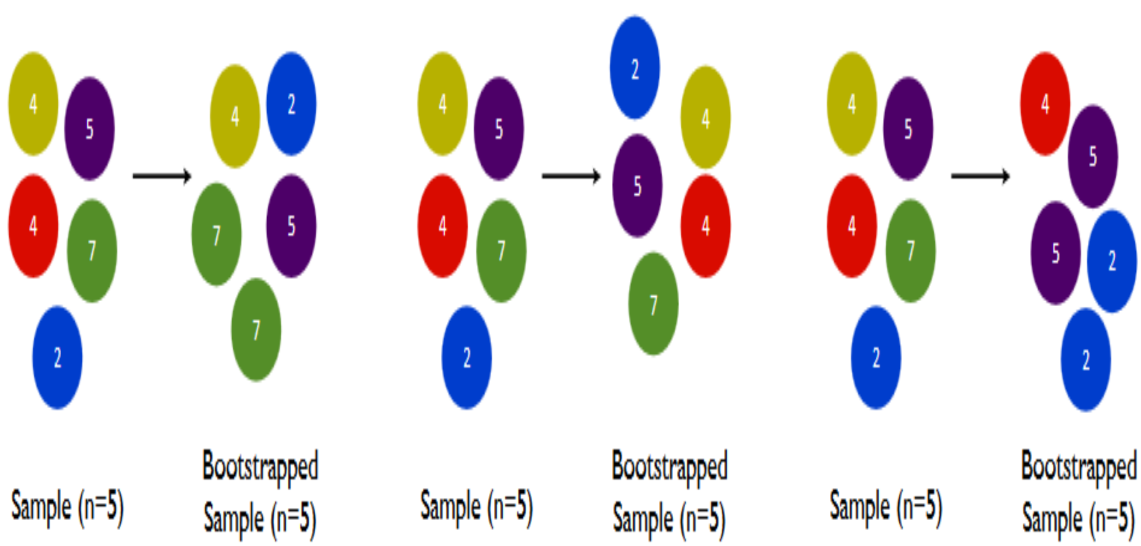
Bagging – starting with high variance

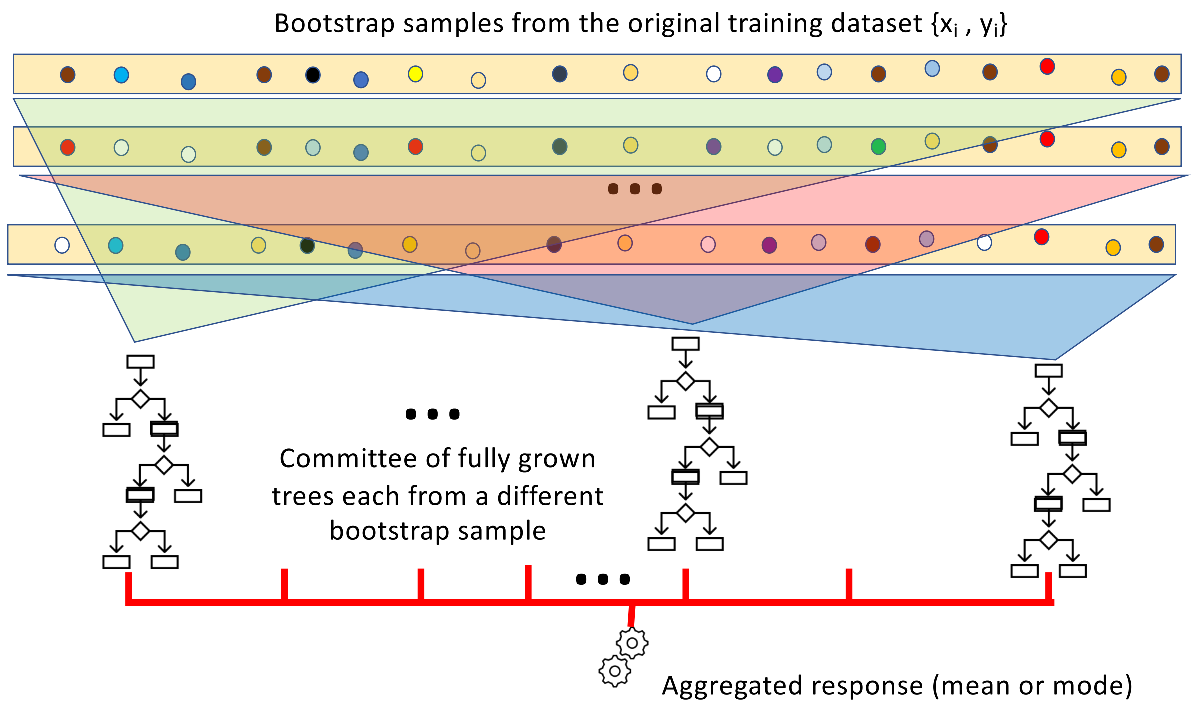
Bagging - Bootstrap Aggregation

- Perturb X by bootstrap sampling
- Uses homogeneous base learners, typically decision trees
- No pruning – each tree is grown fully
- Tree growth can be easily parallelized
- Aggregation is by taking mean (regression) or mode (classification)
- Bootstrap samples generally leave out 1/e ($=\lim_{n\to\infty}(1-1/n)^n$) of the population
- Such samples are considered OOB
- OOB samples automatically provide the validation dataset – no need for train-test split



Bagging uses Bootstrap Sampling

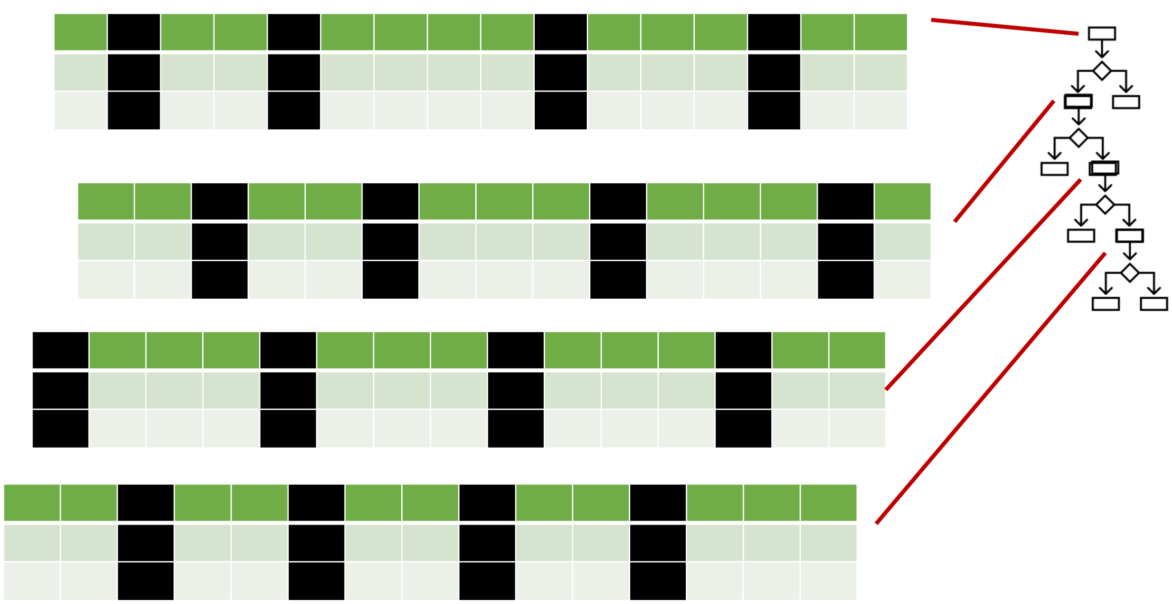


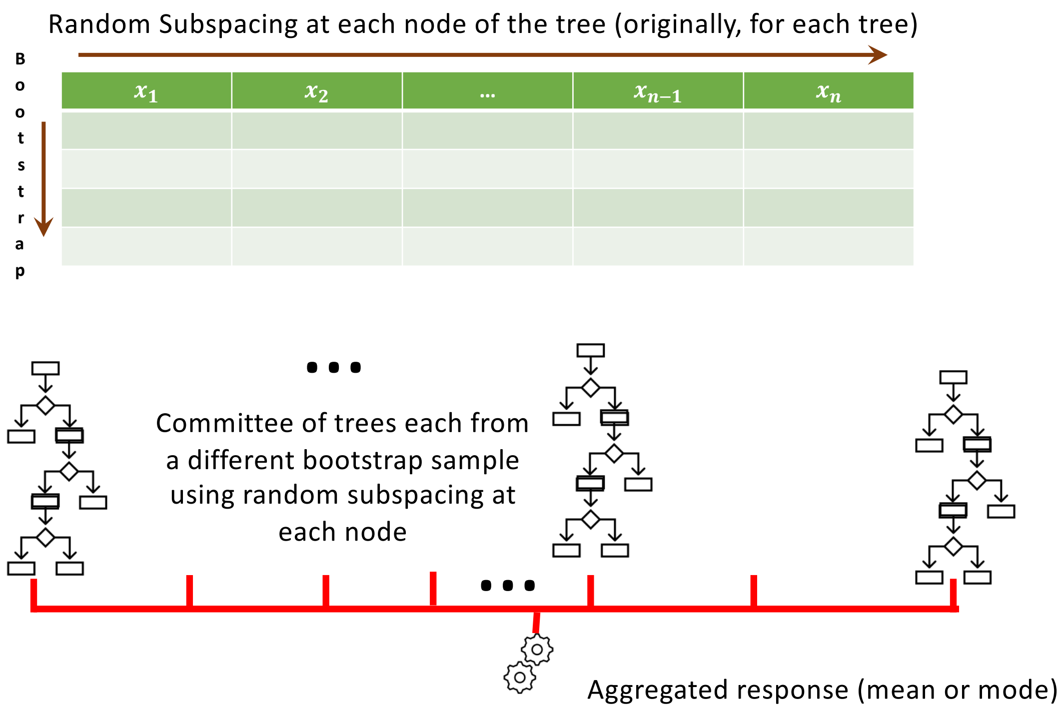


Bagging: Problem

- Bootstrap sampling => Generated trees are identically distributed
- The relative importance of the features are preserved across the trees => trees are correlated => trees are i.d but not i.i.d
- How do decorrelate trees and produce i.i.d trees?

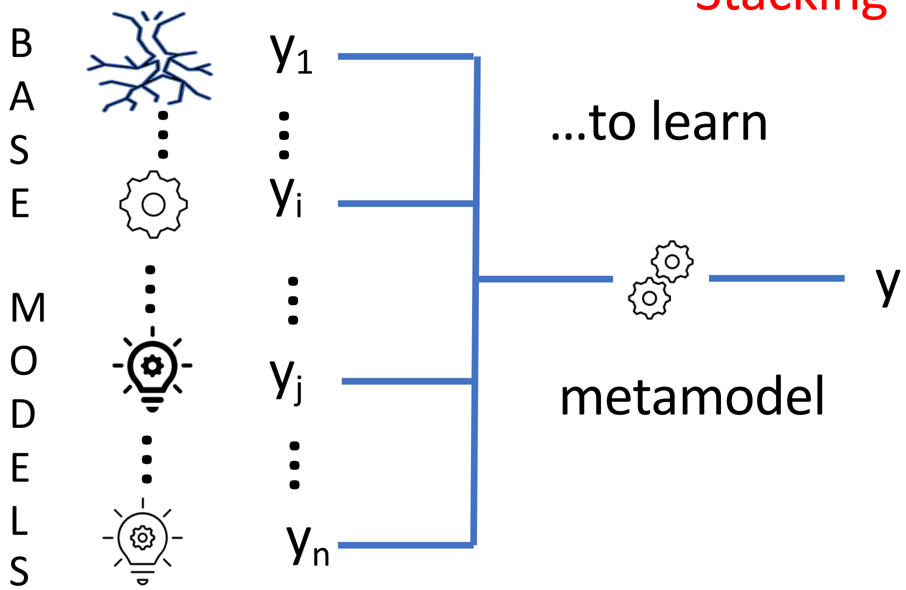
One more perturbation: Random subspacing



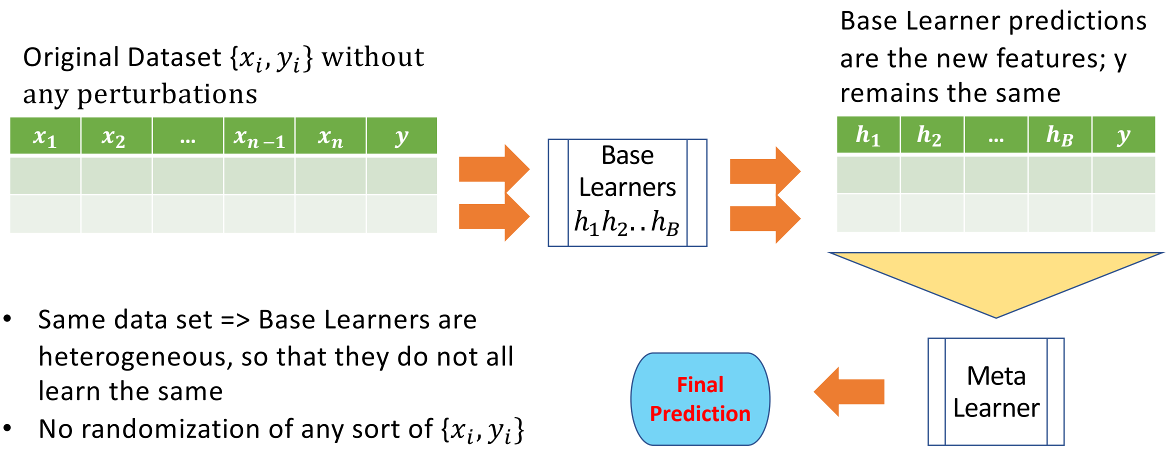

Random Subspacing at each node of the tree (originally, for each tree)
B o o t s t r a p
x_1 | x_2 | … | x_{n-1} | x_n
Committee of trees each from a different bootstrap sample using random subspacing at each node
Aggregated response (mean or mode)


Stacking

Heterogeneous Learning… Stacking

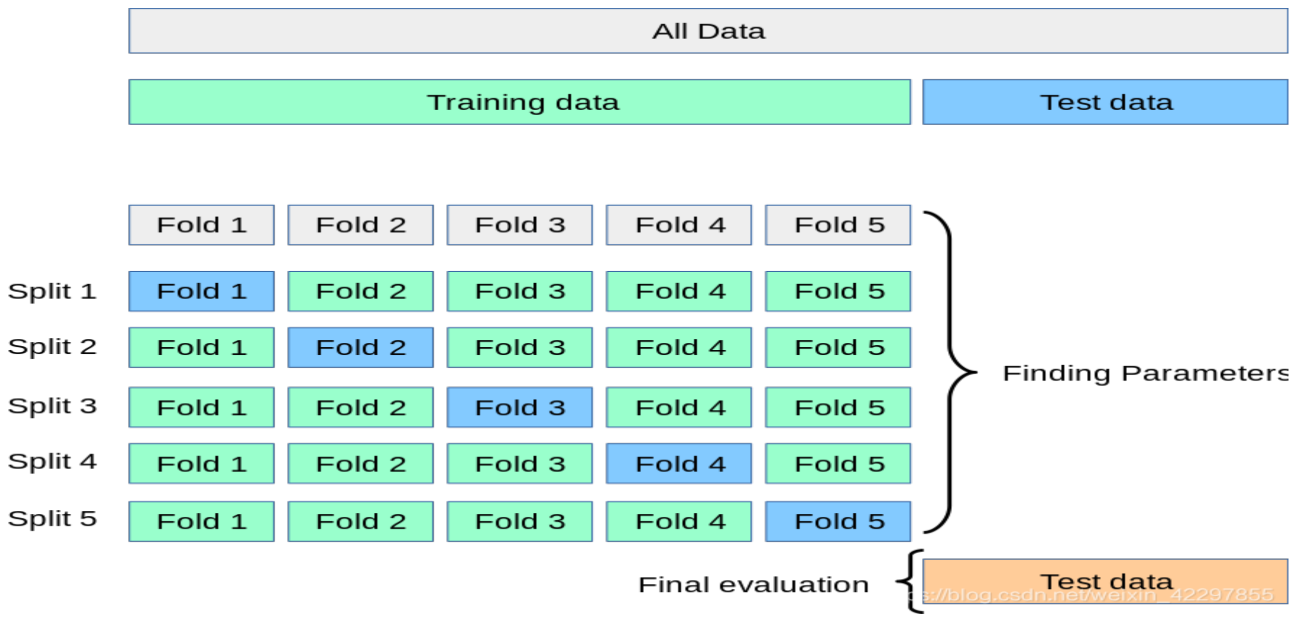


Stacking in a Nutshell

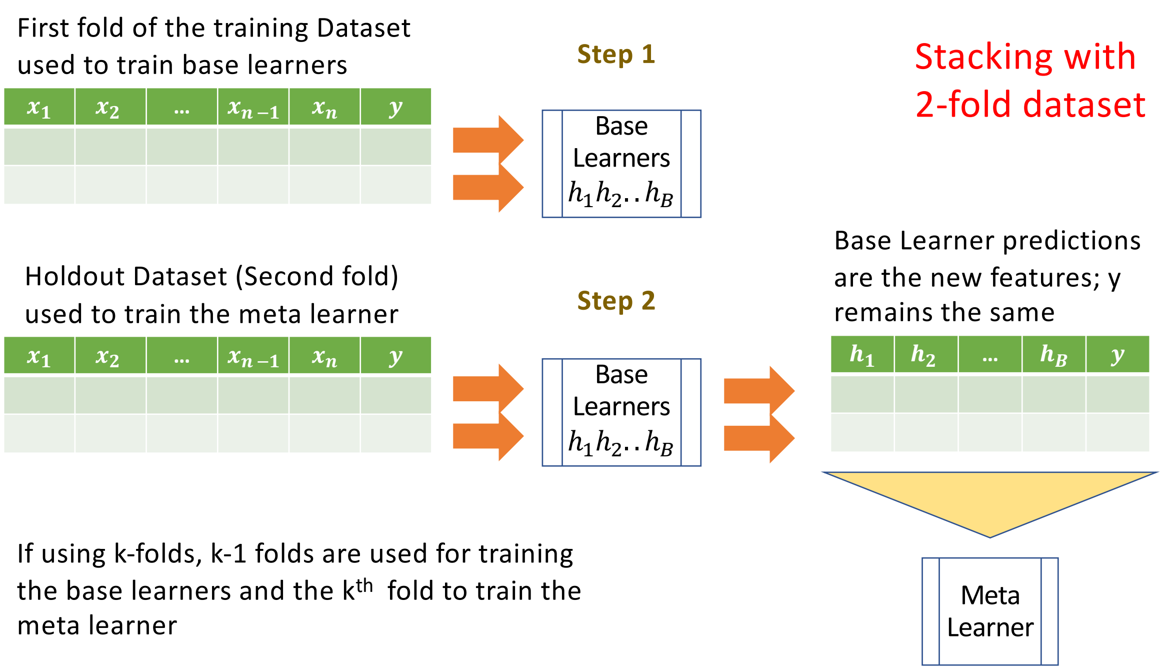


- Same data set => Base Learners are heterogeneous, so that they do not all learn the same
- No randomization of any sort of $\{x_i, y_i\}$ => Chances of overfitting are high
- Solution: Use different data sets for the base learners using k-fold sampling

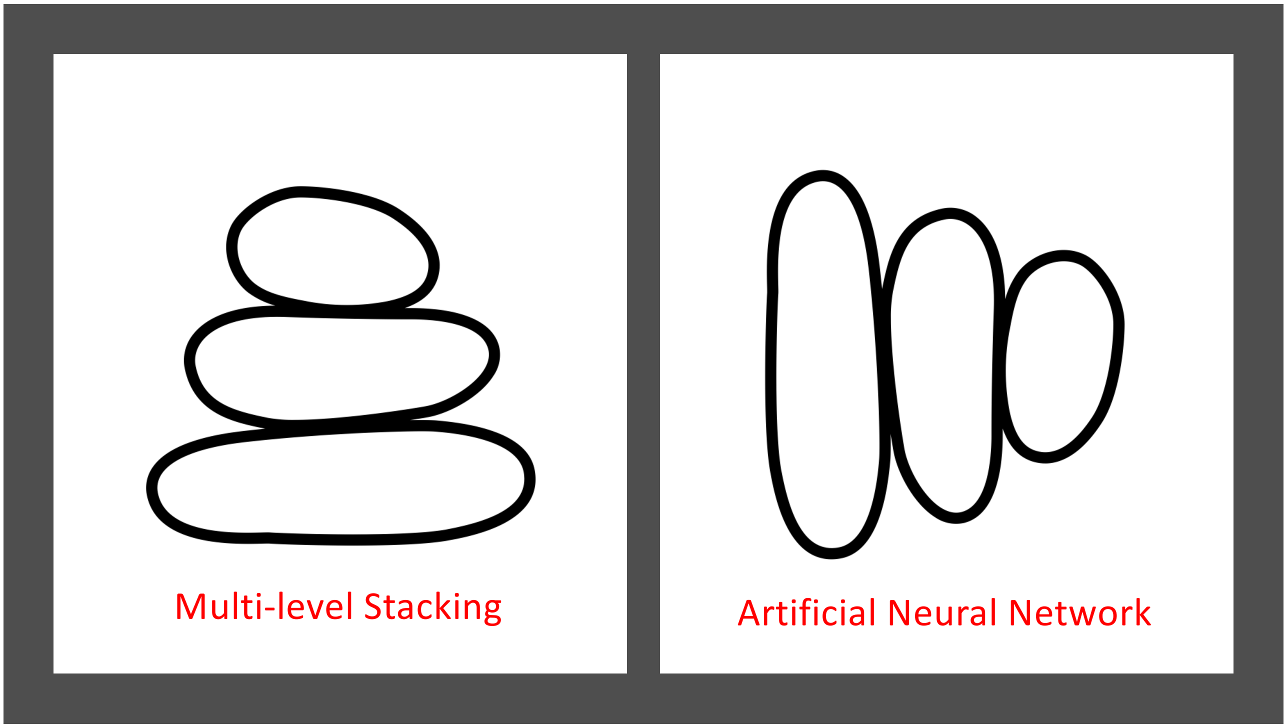
K-fold Sampling for Cross Validation



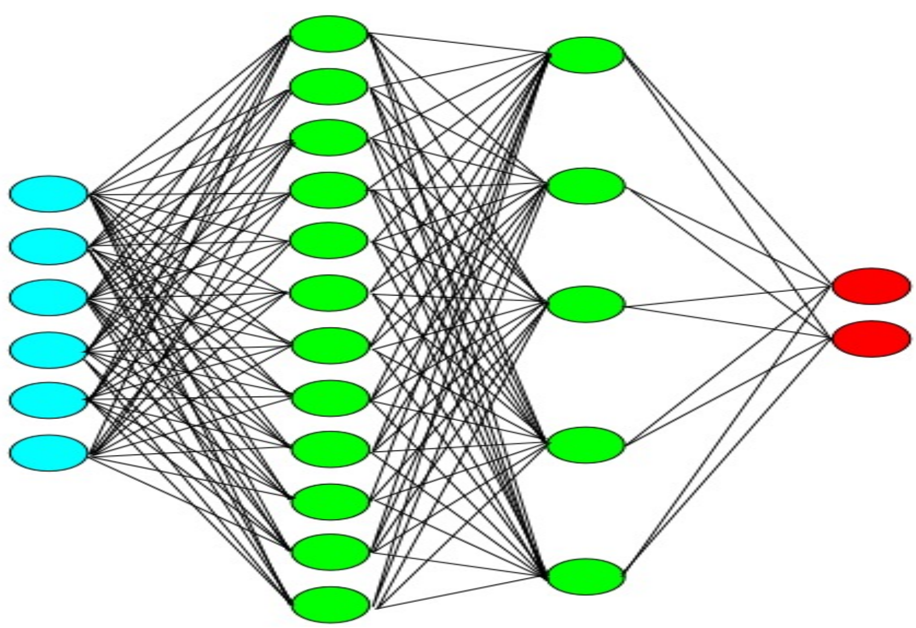
Stacking with 2-fold dataset

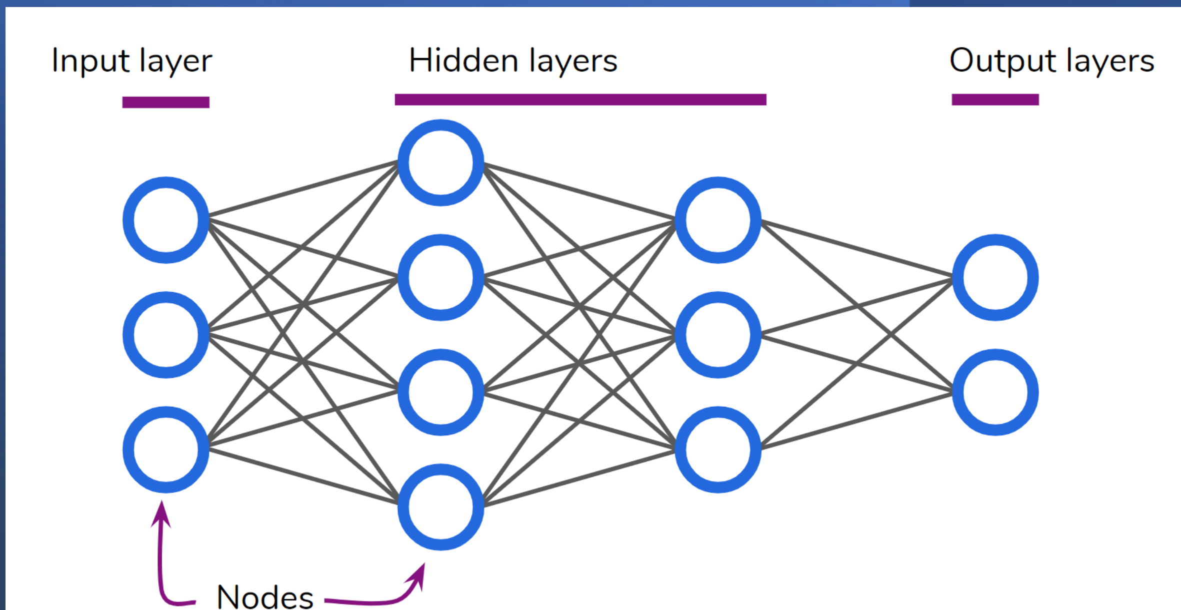


If using k-folds, k-1 folds are used for training the base learners and the k^{th} fold to train the meta learner



Multi-level Stacking





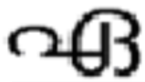
Extending Meta-learning to Deep Learning...

Tell the language of the letter

Malayalam

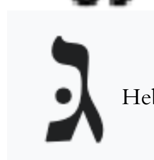
Malayalam

Hebrew



Malayalam

Gujarati

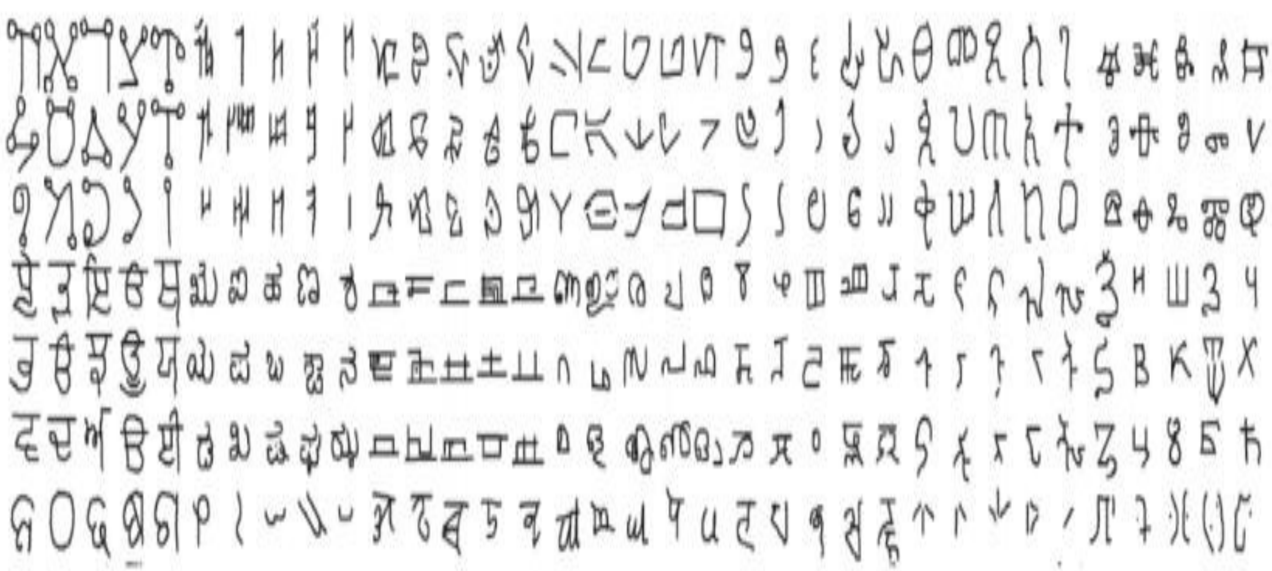


Hebrew

What's the ML algorithm that comes closest to doing this?

Can we make the machine learn in one-shot like you did?

Suppose you have this omniglot labeled dataset…



How do you produce an ML model to recognize a new letter not in the dataset?

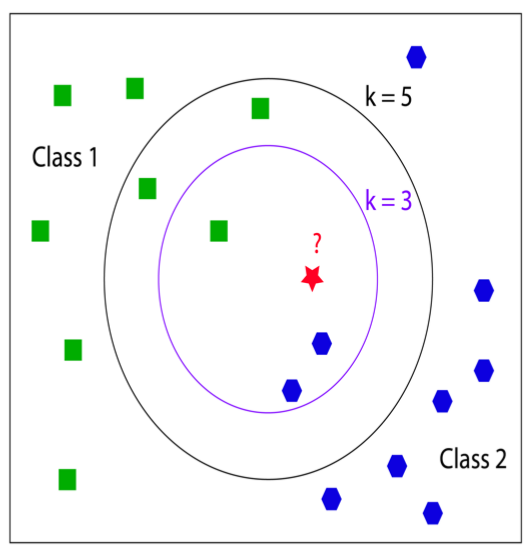
One-shot Learning!

K Nearest Neighbors: Classification

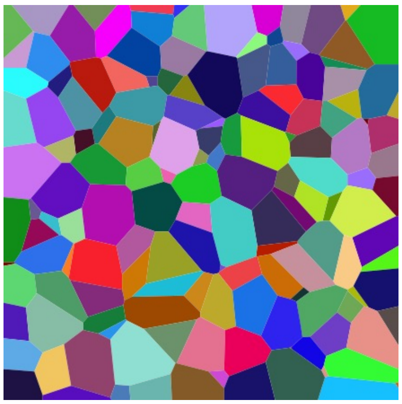
- In binary classification where y = +1 or -1, a test item is classified as

$y_t = \text{sgn}(\sum_{i=1}^{k} y_i)$, where (xi, yi) are the k nearest neighbors of the test data item in the feature space

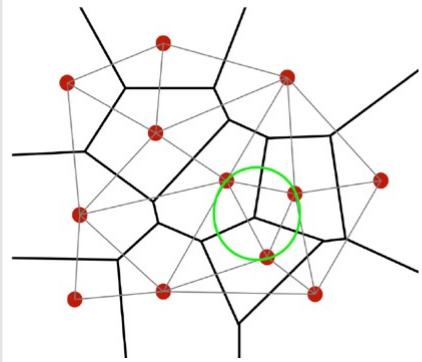
- The k nearest neighbors are determined based on a distance metric, ***usually Euclidean***



Visualization of the Induced Decision Boundary



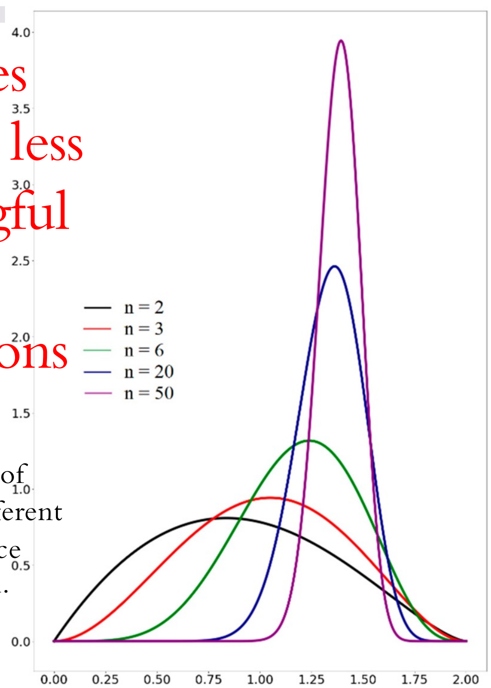
1-NN: VORONOI DIAGRAM



What's the problem with 1-NN?

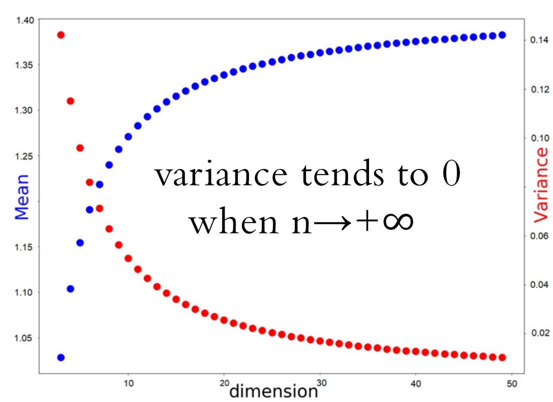
Distances become less meaningful in high dimensions

Distribution of distances for different values of space dimension n.



(a)

Lellouche, S., & Souris, M. (2019). Distribution of distances between elements in a compact set. *Stats*, *3*(1), 1-15.



(b)

Can we build a neural network to learn to classify based on the distance in the feature space?

What if we compute the distances in the extracted feature space instead of in the original pixel space or learn a "metric space"?


Q&A