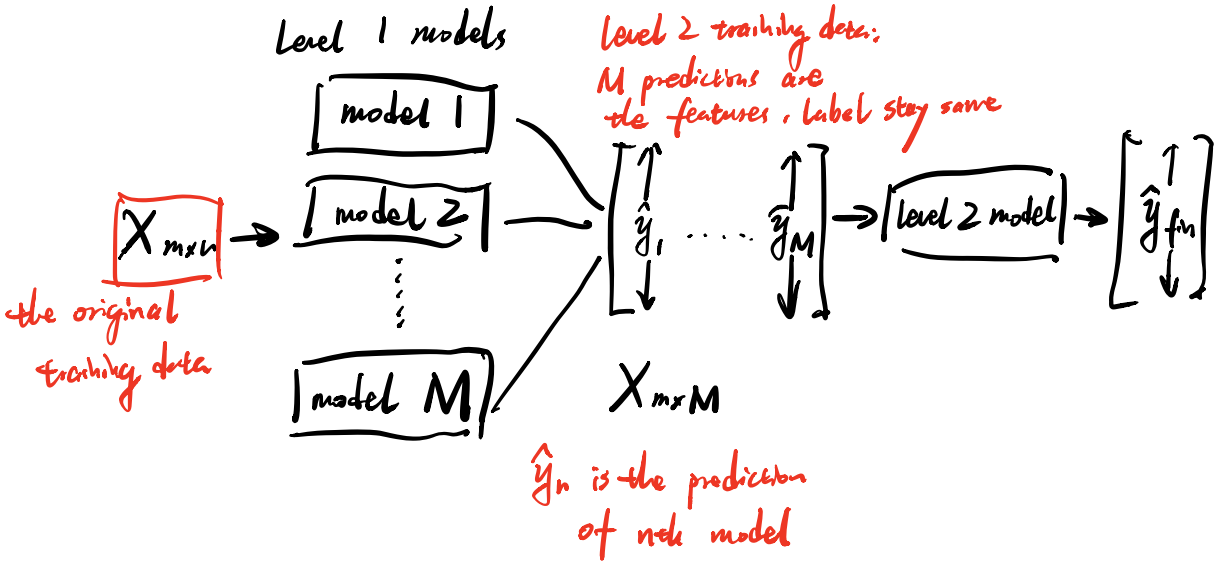
Ensemble Method:

1. Stacking

Taking a weighted combination of the predictions of a total of S different classifiers.

The weights can be a simple average, or learned via a secondary classification/regression process.

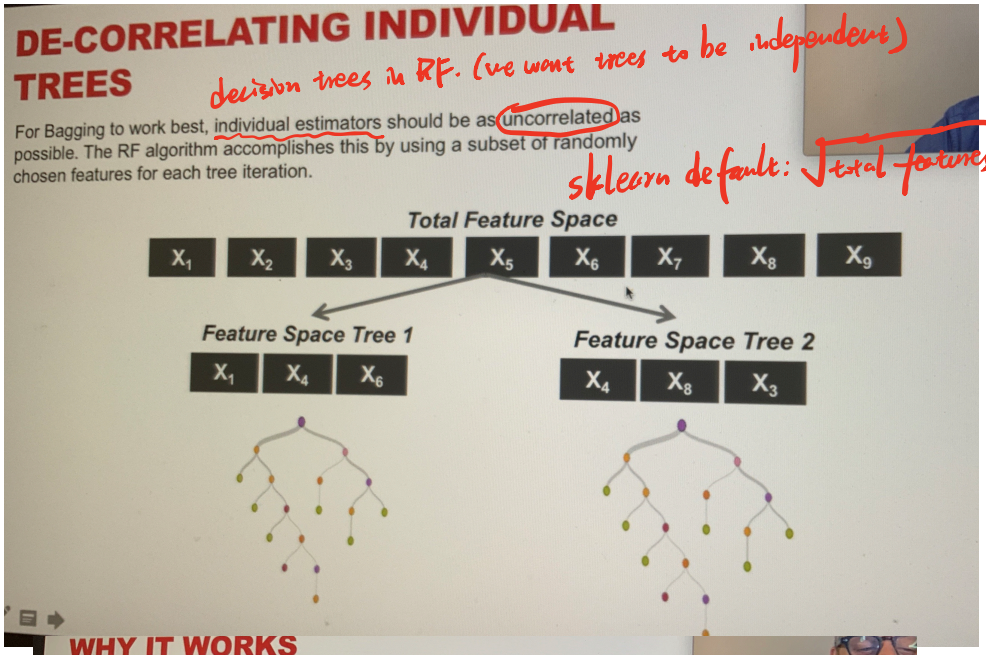
$$f(X_i) = \sum_{s=1}^{S} w^s \times f^s(X_i)$$

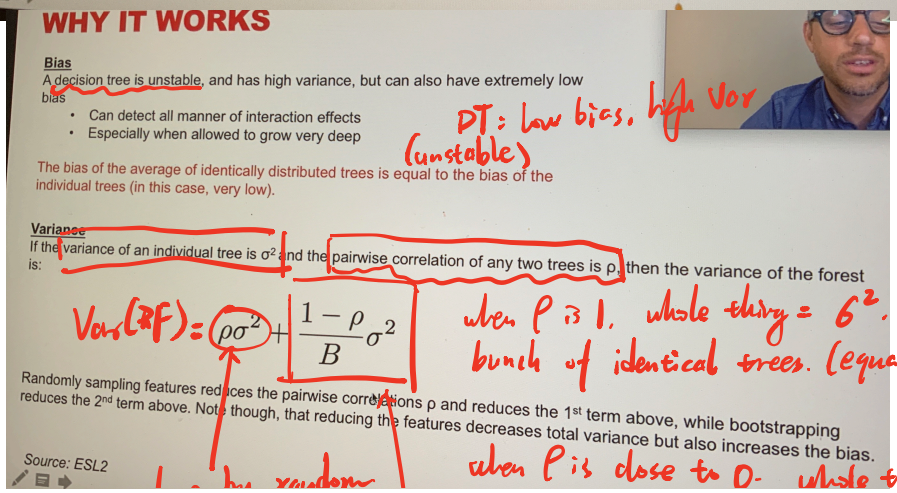


In order for learned stacking to generalize, the stage 2 model needs to be built from data that is seperate from stage 1 models. Otherwise, the system risks overfitting.

Bagging: (Bootstrap + Aggregation)

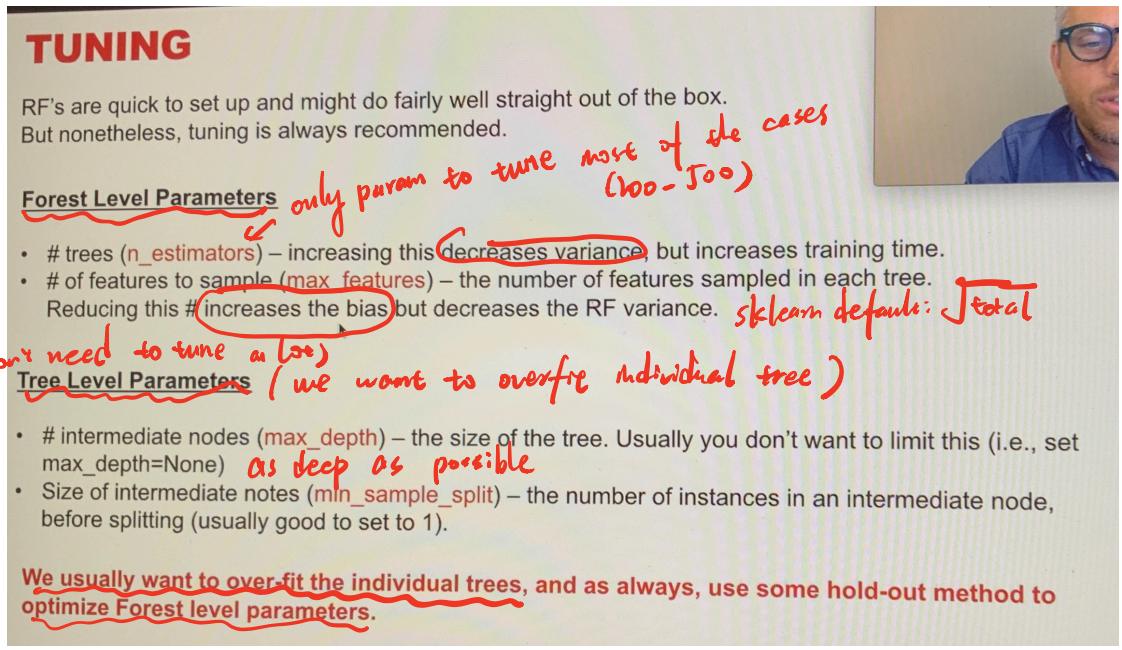
1. create a bootstrap sample of the data
2. fit the model on the bootstrap sample
3. make the prediction on train/test data using the model
4. Repeat step 1.2.3 for n times. storing each prediction
5. Make the final prediction by averaging the bootstrapped predictions.





reduce by selected features | $\frac{1}{B}\sigma^2$, decrease as $B\uparrow$ (more trees, less var)

reduce by bootstrapping



Out-of-bag error (built-in RF validation property)

for each bootstrap sample. $\frac{1}{3}$ of sample were not selected to build the tree. so they serve as the validation set.

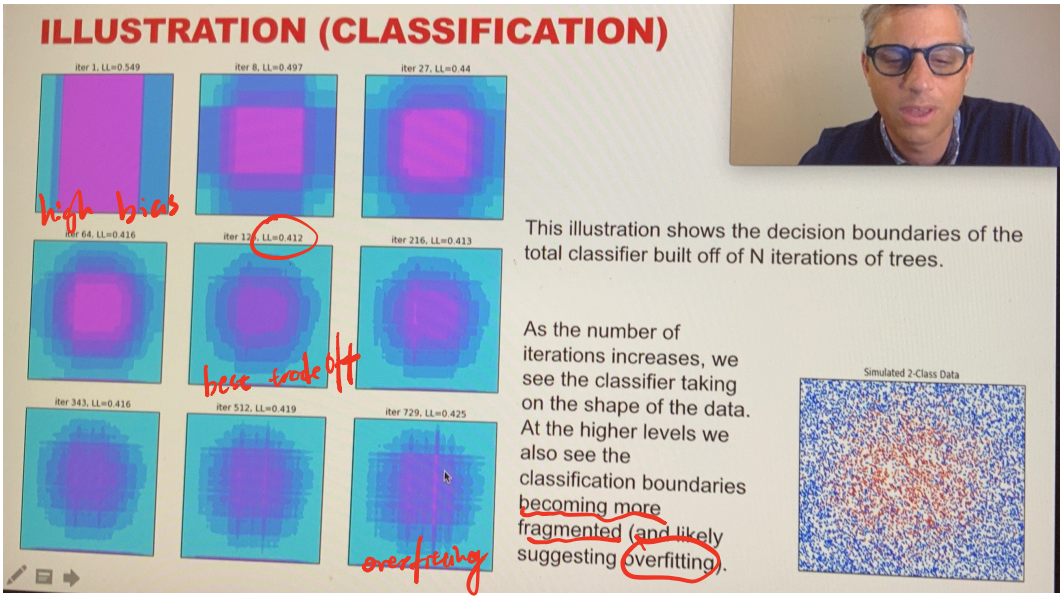
so RF on each OOB sample and average the error can give us the out-of-sample error estimation.

(RF has the built-in validation property called OOB error)

Gradient Boosting

We start with our best guess, then compute the error on each observation, and fit an DT to the residuals. The fitted DT becomes the next f(x) that gets added to the ensemble.

Different from the RF, GBT is an iterative process, where each tree is fit on the residuals of the cumulative prediction from all prior trees.



Modern variants: XGBoost, LightGBM, CatBoost

which one to use is an empirical question

kNN:

$$\hat{y}(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i$$

or we can take the weighted average. weight can be the distance. (closer neighbor can influence more)

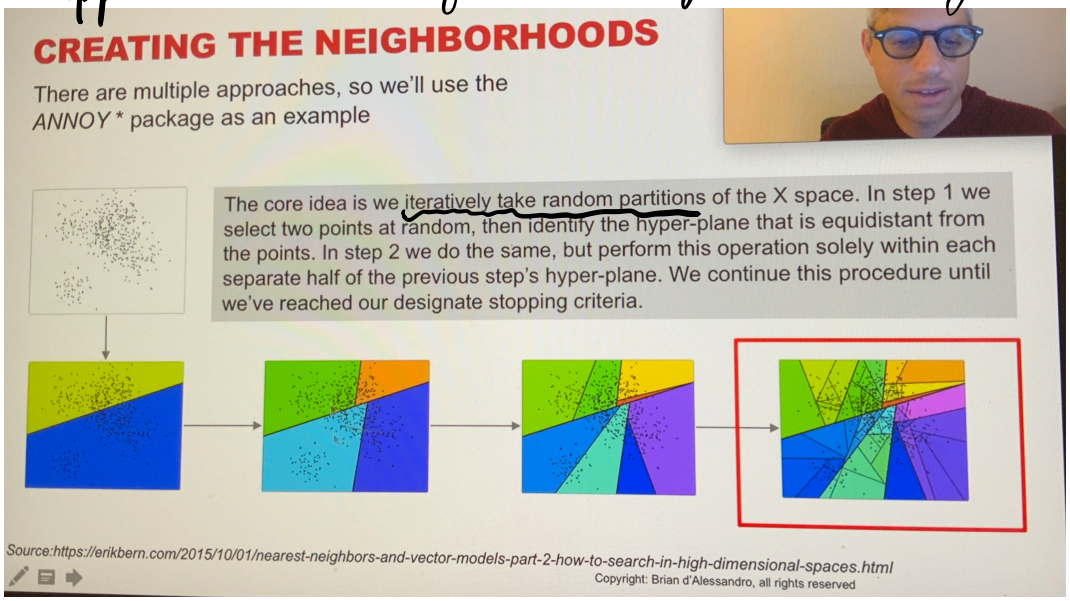
Property:

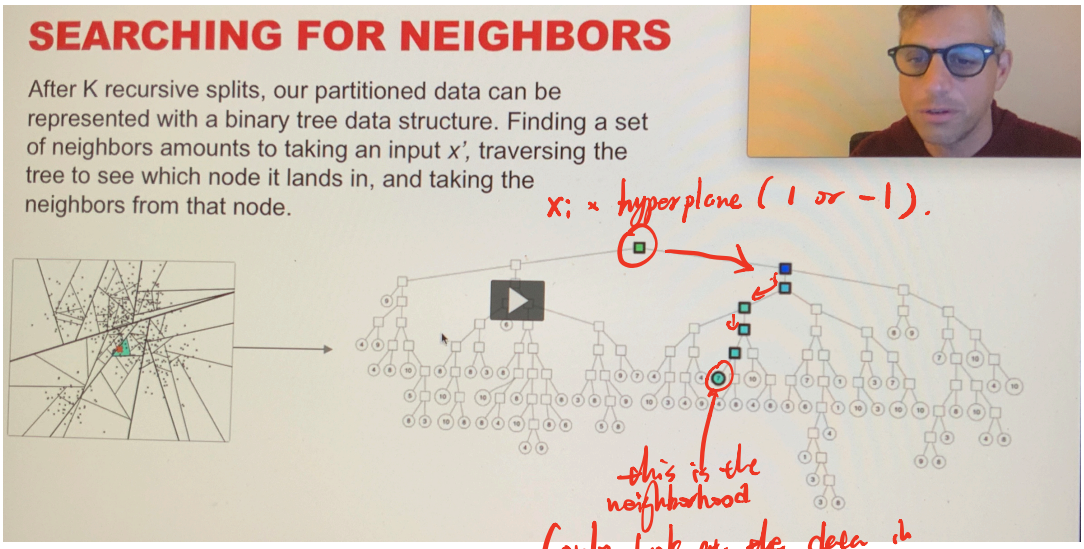
1. Non-parametric (like decision tree)
2. Instance-based learning (don't need to train)

Challenges:

1. Expensive Search (sklearn.fit() allows more efficient search)
2. Curse of dimensionality (when d ↑, all points are equally apart)
3. Scale matters (features with higher magnitude tend to dominate, need to normalize first)

Approximate Nearest Neighbors O(log N) use binary tree



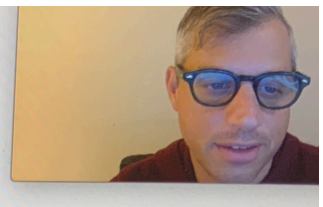


Problem 1: you are limited to just the neighbors in the terminal node of the tree.

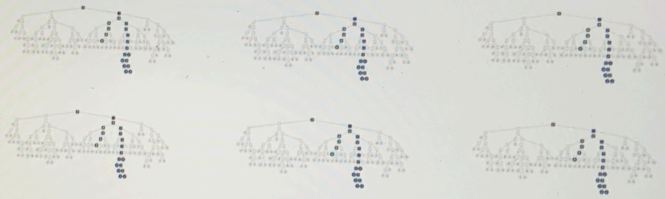
Solution 1. you don't have to use a single terminal node. Nearby splits can also be considered

2. you can build a forest using multiple trees. with different starting points. Run the algorithm D times to get D seperate neighborhoods for a point. When scoring the candidate set becomes the union of all D neighborhoods.

THE TRADEOFFS



Problem: You may miss true nearest neighbors that are on the boundaries of the partitions

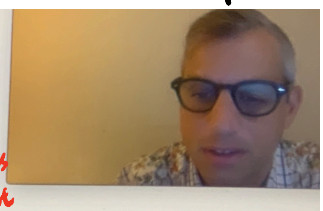


Solution: You can build a forest using multiple trees. Run the full algorithm D times to get D separate neighborhoods for a point. When scoring, the candidate set becomes the union of all D neighborhoods.

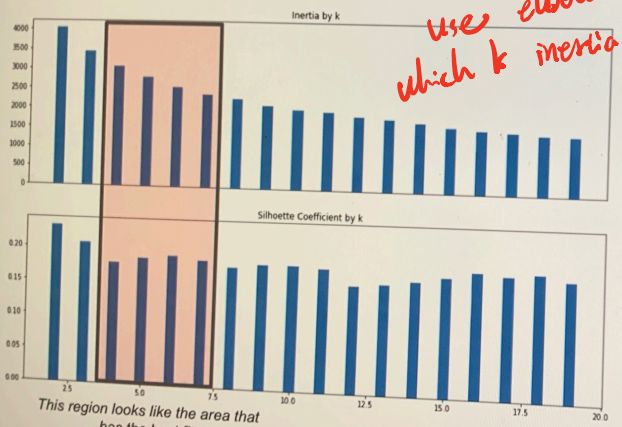
Clustering (do dimension reduction first ⇒ stable clusters)

Quantify Goodness of fit to find K (# of clusters)

WE CAN QUANTIFY GOODNESS OF FIT TO IDENTIFY THE RIGHT K



use elbow method which k inertia decreases the most



This region looks like the area that has the best fit

The *inertia* measures the within cluster sum of squares distance from the mean.

$$inertia = \sum_{j=1}^{k} \sum_{x_i \in C_j} |x_i - \mu_j|^2$$

within cluster distance

a = the mean distance between a sample and all other points in the same cluster

b = the mean distance between a sample and all other points in the nearest cluster

$$s = \frac{b - a}{max(a, b)}$$