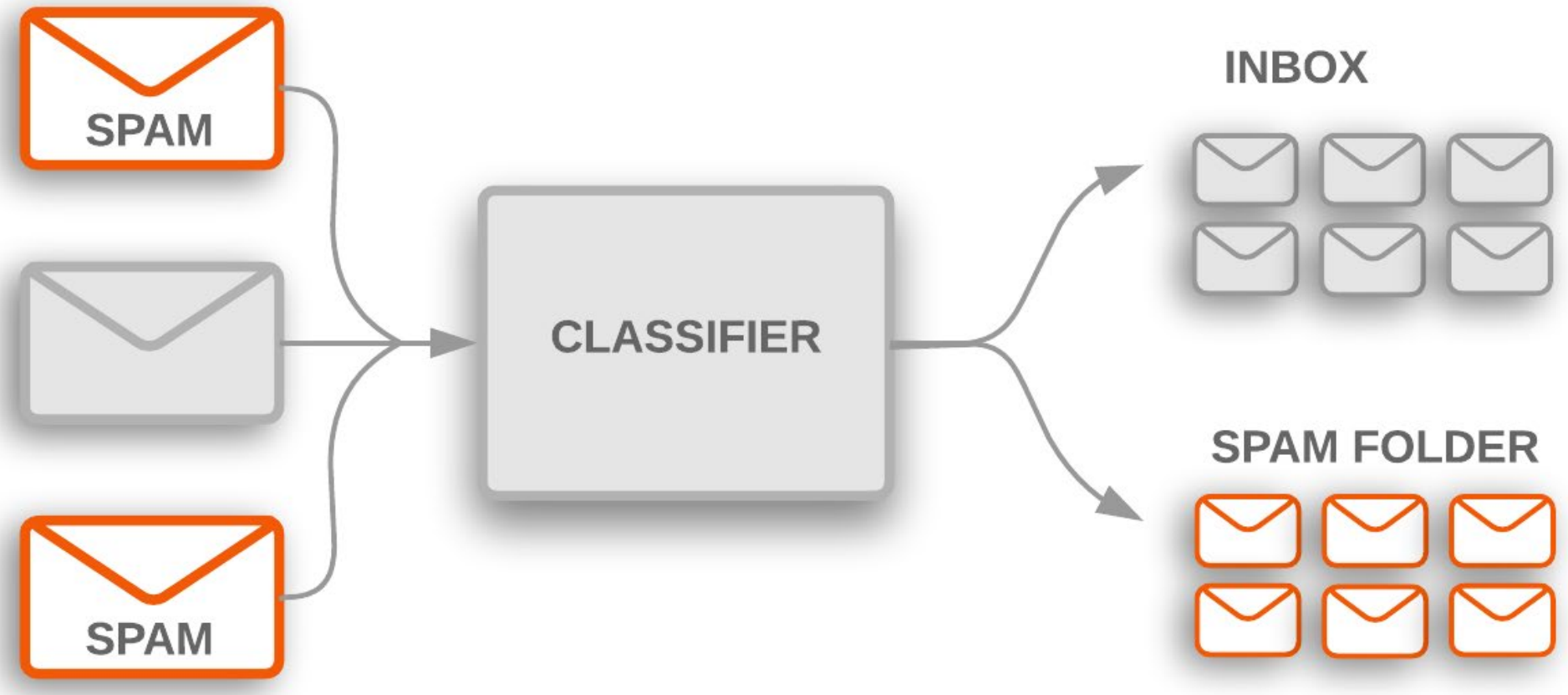
Machine Learning and its Applications

Supervised Learning: Classification & KNN

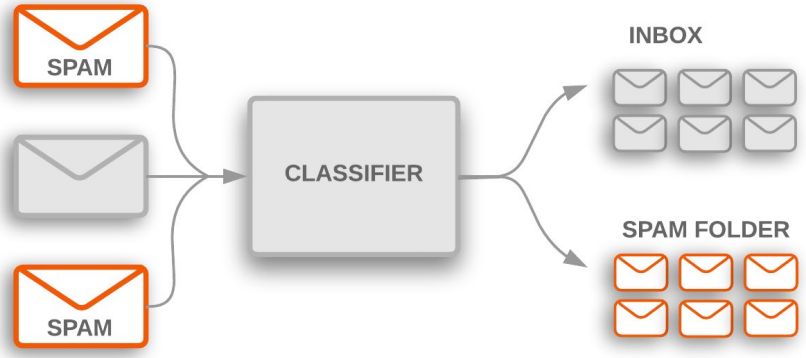
Urban Information Lab

What is Classification?



Binary Classification

1. Email Spam Detection (Spam or not)
2. Conversion Prediction (Buy or not)
3. Churn Prediction (Churn or not)



Multi-class Classification

1. Face Classification
2. Plant Species Classification
3. Optical character recognition

Binary Classification

1. Email Spam Detection (Spam or not)
2. Conversion Prediction (Buy or not)
3. Churn Prediction (Churn or not)

Methods

1. Logistic Regression
2. K-Nearest Neighbors
3. Decision Trees
4. Support Vector Machine
5. Naive Bayes

Multi-class Classification

1. Face Classification
2. Plant Species Classification
3. Optical character recognition

Methods

1. K-Nearest Neighbors
2. Decision Trees
3. Naïve Bayes
4. Random Forest
5. Gradient Boosting

Multi-class Classification

1. Face Classification
2. Plant Species Classification
3. Optical character recognition

Methods

1. K-Nearest Neighbors
2. Decision Trees
3. Naïve Bayes
4. Random Forest
5. Gradient Boosting

1. One vs. Rest

(Binary algorithm that can do multi-class classification)

- Fit only binary classification model for each class v. all other classes

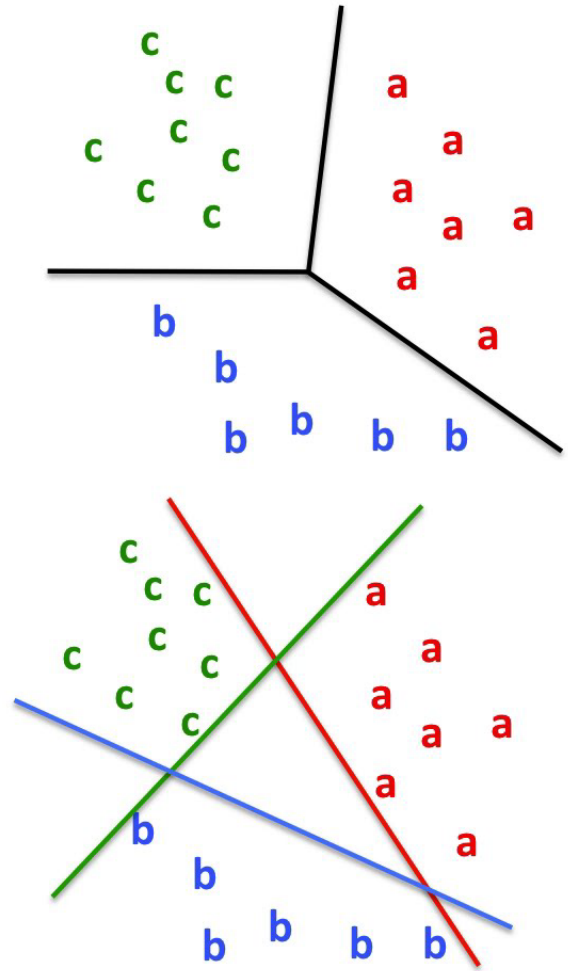
-> Logistic Regression & Support Vector Machine

2. One vs. One

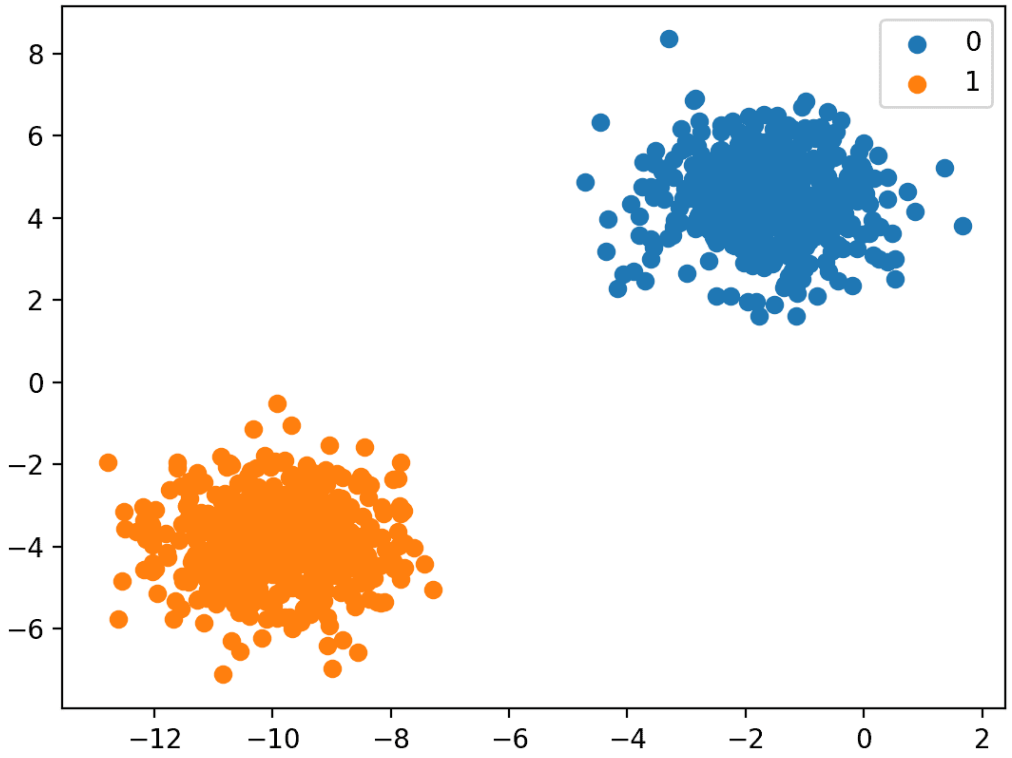
- Fit only binary classification model for each pair of classes

Multi-class vs. Binary classification

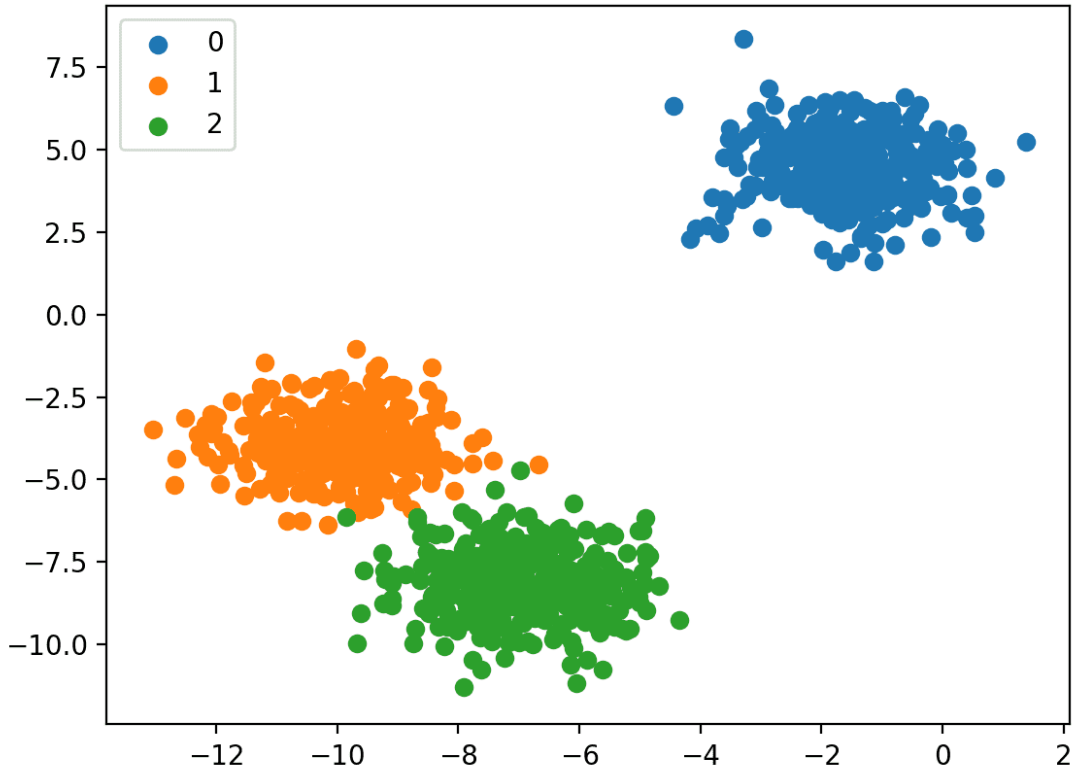
- Multi-class:
  - classes mutually exclusive:
    - instance is either a or b or c
    - even if it's an outlier
  - NB, kNN, DT, logistic
- Binary:
  - one-vs-rest:
    - {a} vs {not a}, {b} vs {not b}
  - classes may overlap
    - instance can be both a and b
    - can be in none of the classes
  - SVM, logistic, perceptron



Binary Classification

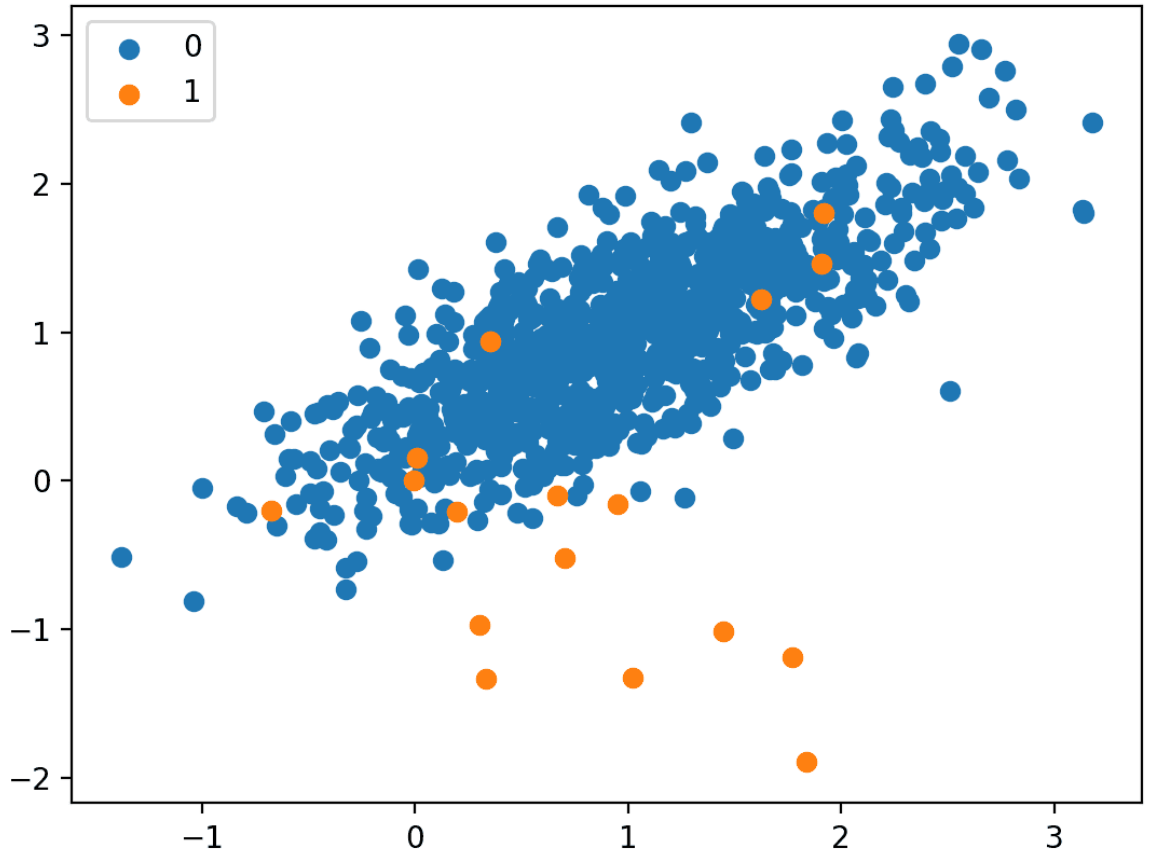


Multi-class Classification



Imbalanced Classification

1. Fraud detection
2. Outlier detection
3. Medical diagnostic tests



KNN

- Nearest neighbors sensitive to noise or mis-labeled data ("class noise"). Solution?
- Smooth by having k nearest neighbors vote

Algorithm (kNN):

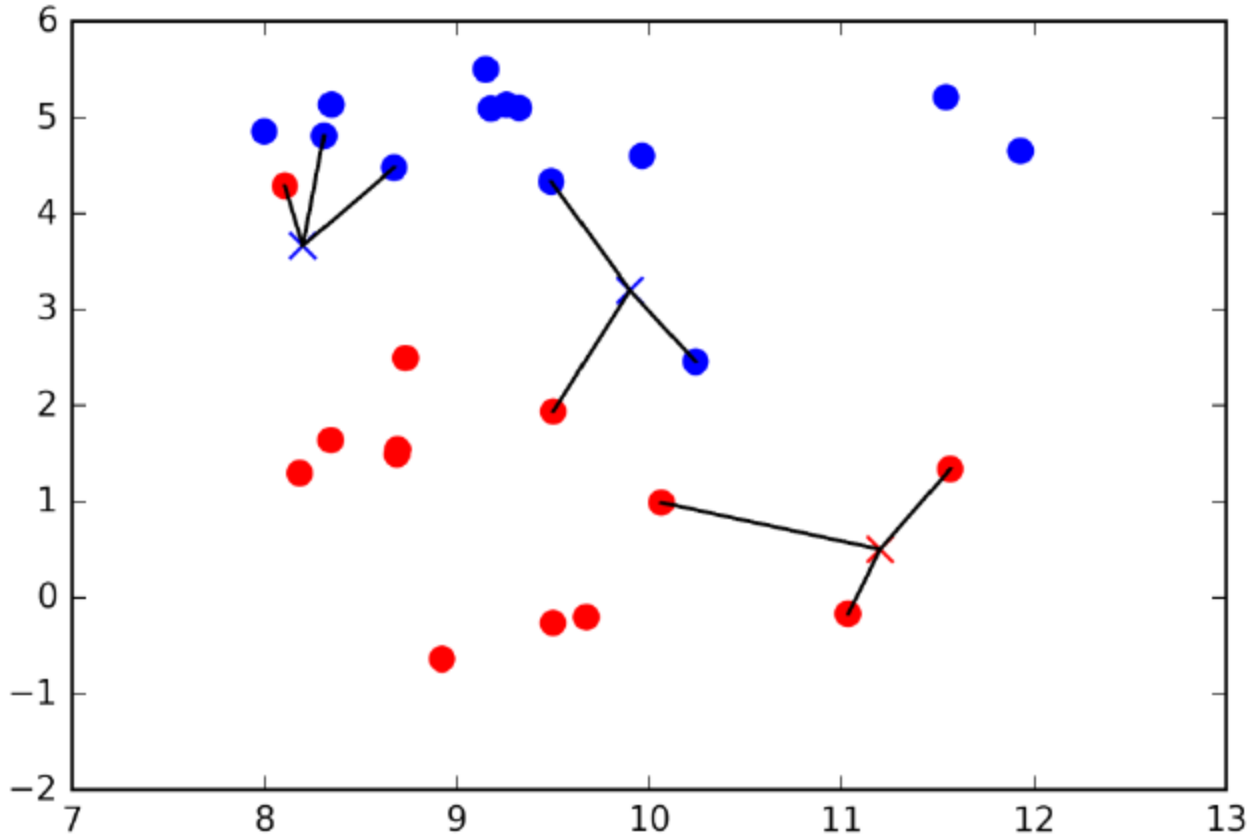
1. Find k examples $\{\mathbf{x}^{(i)}, t^{(i)}\}$ closest to the test instance $\mathbf{x}$
2. Classification output is majority class

$$y = arg\max_{t^{(z)}} \sum_{r=1}^{k} \delta(t^{(z)}, t^{(r)})$$

https://www.cs.toronto.edu/~jlucas/teaching/csc411/lectures/lec5_handout.pdf

KNN

- For more classes, we count how many neighbors belong to each class, and again predict the most common class.



source: Muller and Guido:Binary Classification

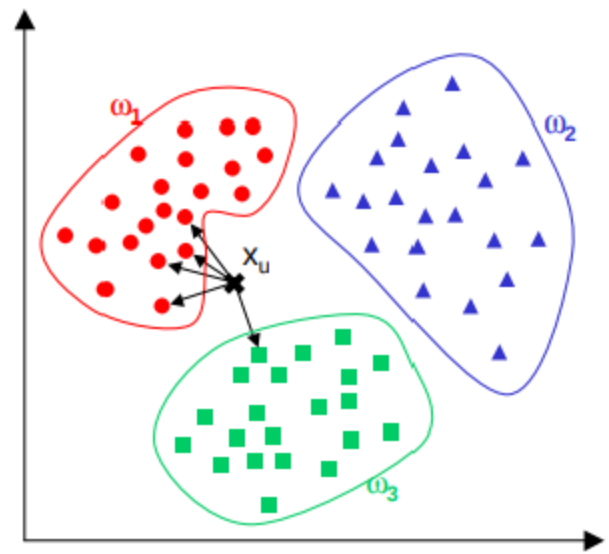
- We do not make any assumption about the functional form of the kNN algorithm, a kNN model is also considered a **non-parametric** model.
- kNN does not have an explicit training step and defers all of the computation until prediction,
- Since the prediction is based on a comparison of a query point with data points in the training set (rather than a global model), kNN is also categorized as **instance-based** (or "memory-based") method.
- While kNN is a lazy instance-based learning algorithm, an example of an eager instance-based learning algorithm would be the **support vector machine**.

`https://github.com/rasbt/stat479-machine-learning-fs19/blob/master/02_knn/02-knn__notes.pdf`

Parametric v. Non-parametric

- **Parametric**
  - Fixed Number of Parameters.
- **Non-parameteric**
  - No fixed number of parameters
  - parameters may grow with the number of training data points. Non-parametric does not mean no parameters!

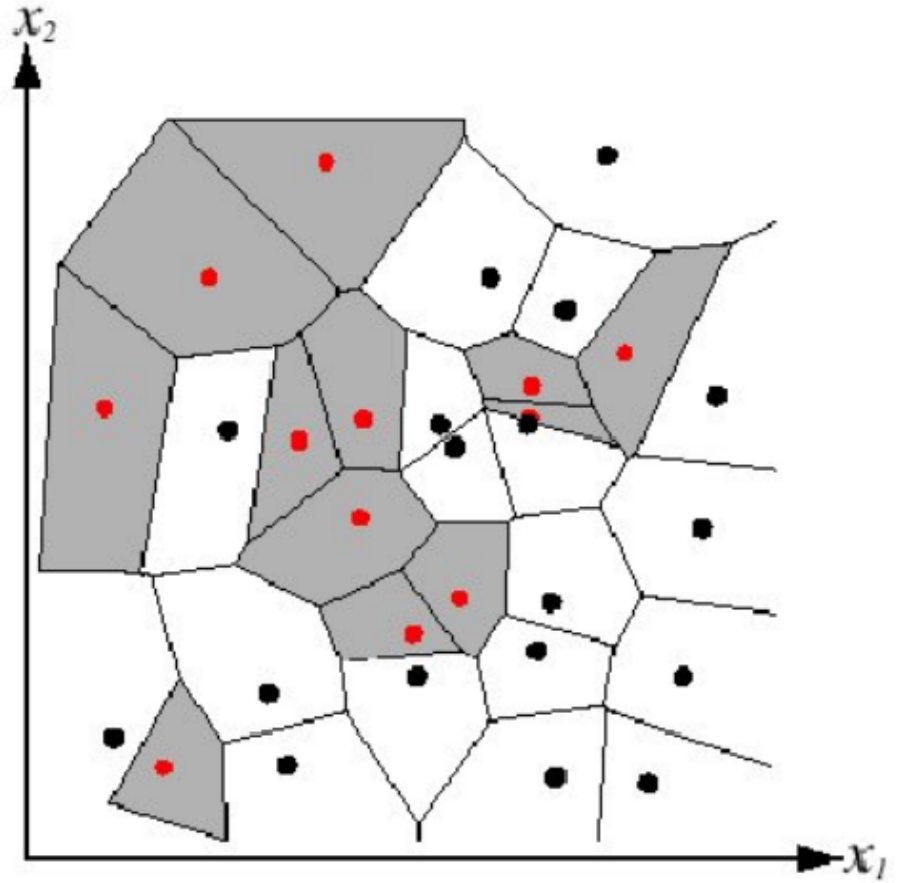
- The k-NN only requires
  - An integer k
  - A set of labeled examples (training data)
  - A metric to measure "closeness"
- 3 classes and goal: find a class label for the unknown X_u.
- The example uses k=5 neighbors and the Euclidean distance.
- Of the 5 closest neighbors, 4 belong to W1 and 1 belongs to W3, so the test point is assigned to W1, the predominant class.



source: Ricardo Gutierrez-Osuna, Wright State University

KNN: Decision Boundary

- Nearest neighbor algorithm does not explicitly compute decision boundaries, but these can be inferred.
- **Voronoi diagram** visualization



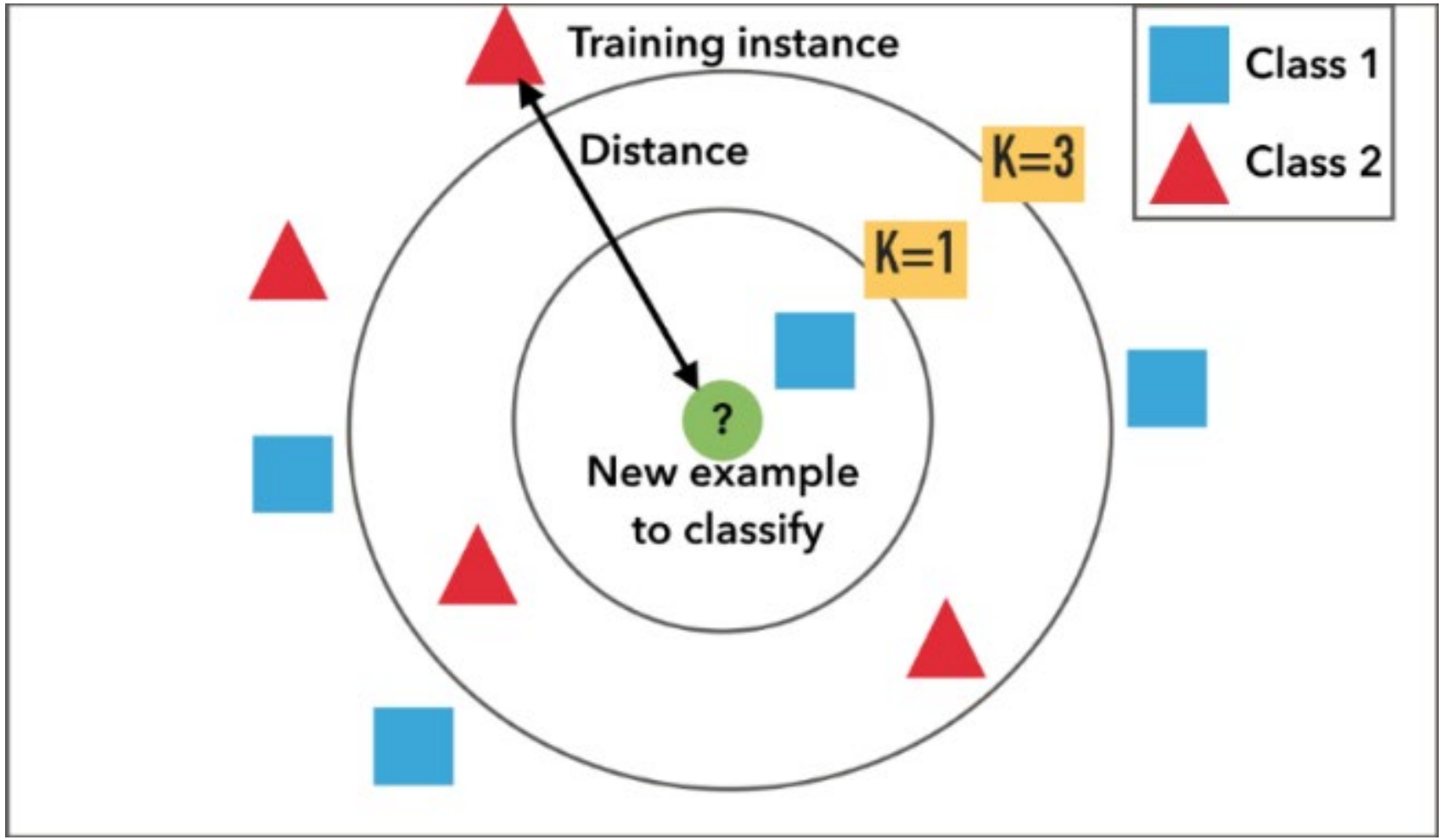
https://www.cs.toronto.edu/~jlucas/teaching/csc411/lectures/lec5_handout.pdf

How do we choose k?

- Larger k may lead to better performance.
- But if we set k too large we may end up looking at samples that are not neighbors (are far away from the query)
- We can use validation set / cross-validation to find k.
- Rule of thumb is $k < \sqrt{n}$, where n is the number of training examples.

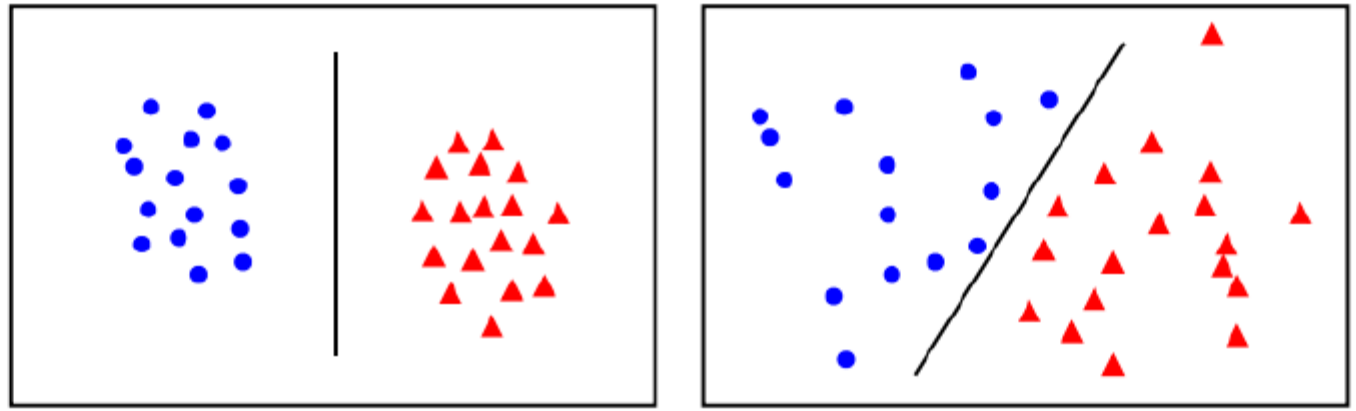
Step 1: Calculate Euclidean Distance
Step 2: Get Nearest Neighbors
Step 3: Make Predictions

$$Euclidean\ Disctance = \sqrt{\sum_{i}^{N} (x1_i - x2_i)^2}$$

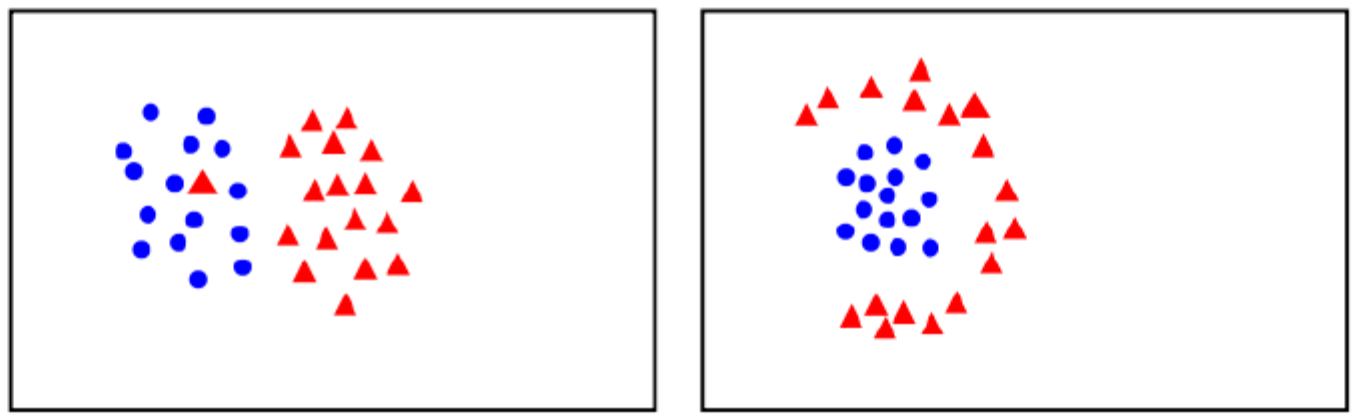


Linear separability

linearly separable



not linearly separable

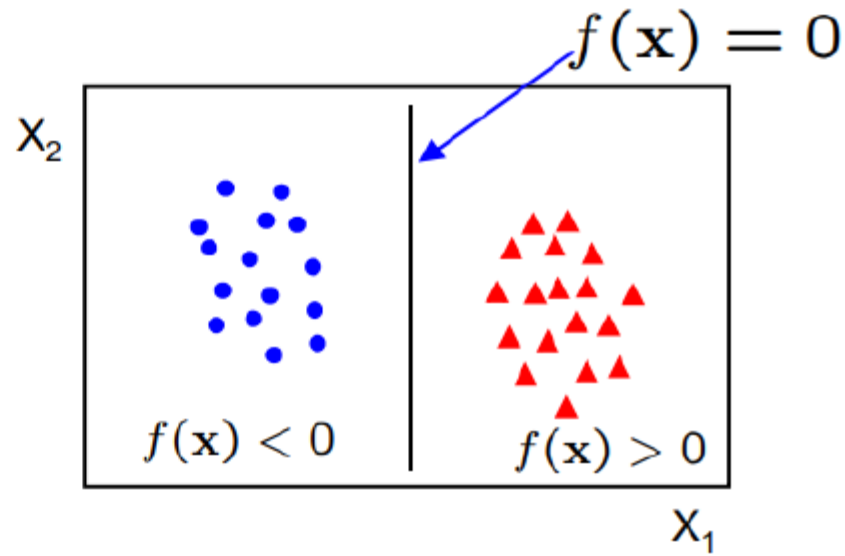


https://www.robots.ox.ac.uk/~az/lectures/ml/

Linear classifiers

A linear classifier has the form

$$f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$$

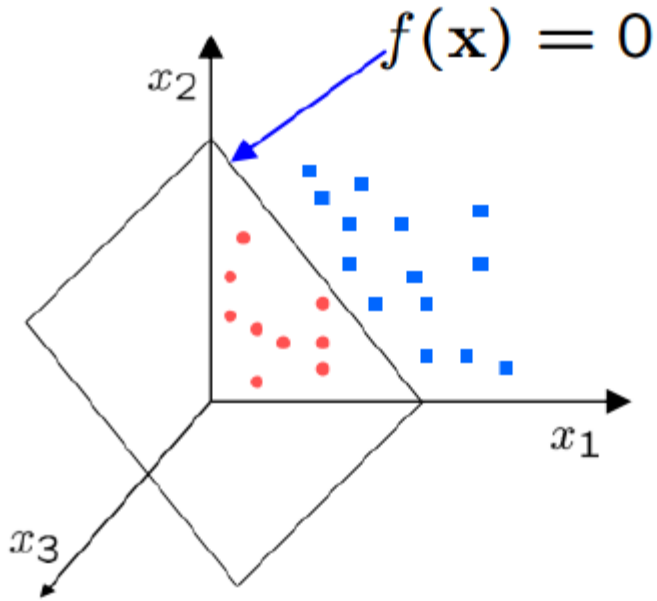


- in 2D the discriminant is a line
- $\mathbf{w}$ is the normal to the line, and b the bias
- $\mathbf{w}$ is known as the weight vector

Linear classifiers

A linear classifier has the form

$$f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$$



- in 3D the discriminant is a plane, and in nD it is a hyperplane

For a K-NN classifier it was necessary to `carry' the training data

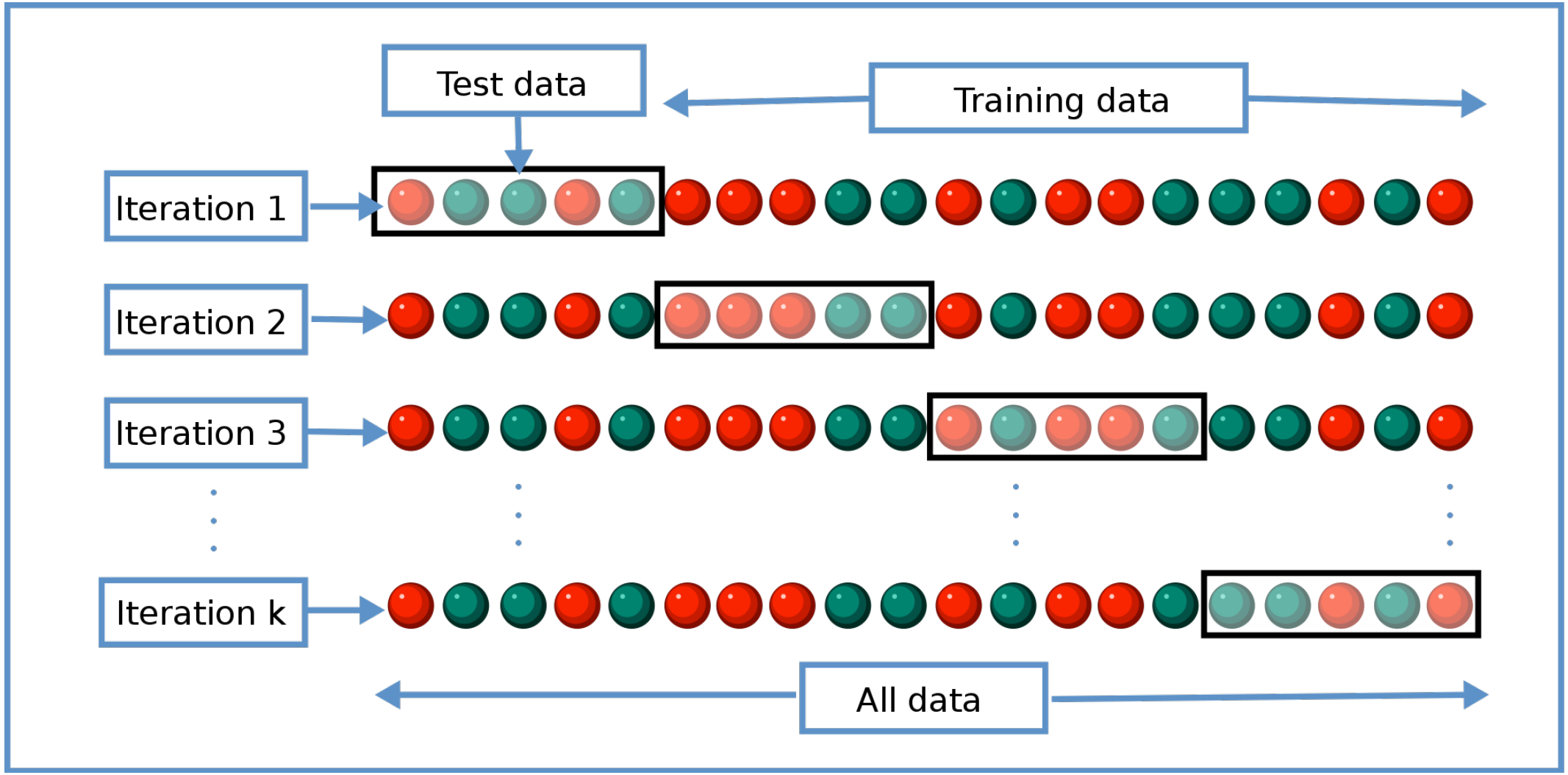
For a linear classifier, the training data is used to learn **w** and then discarded

Only **w** is needed for classifying new data

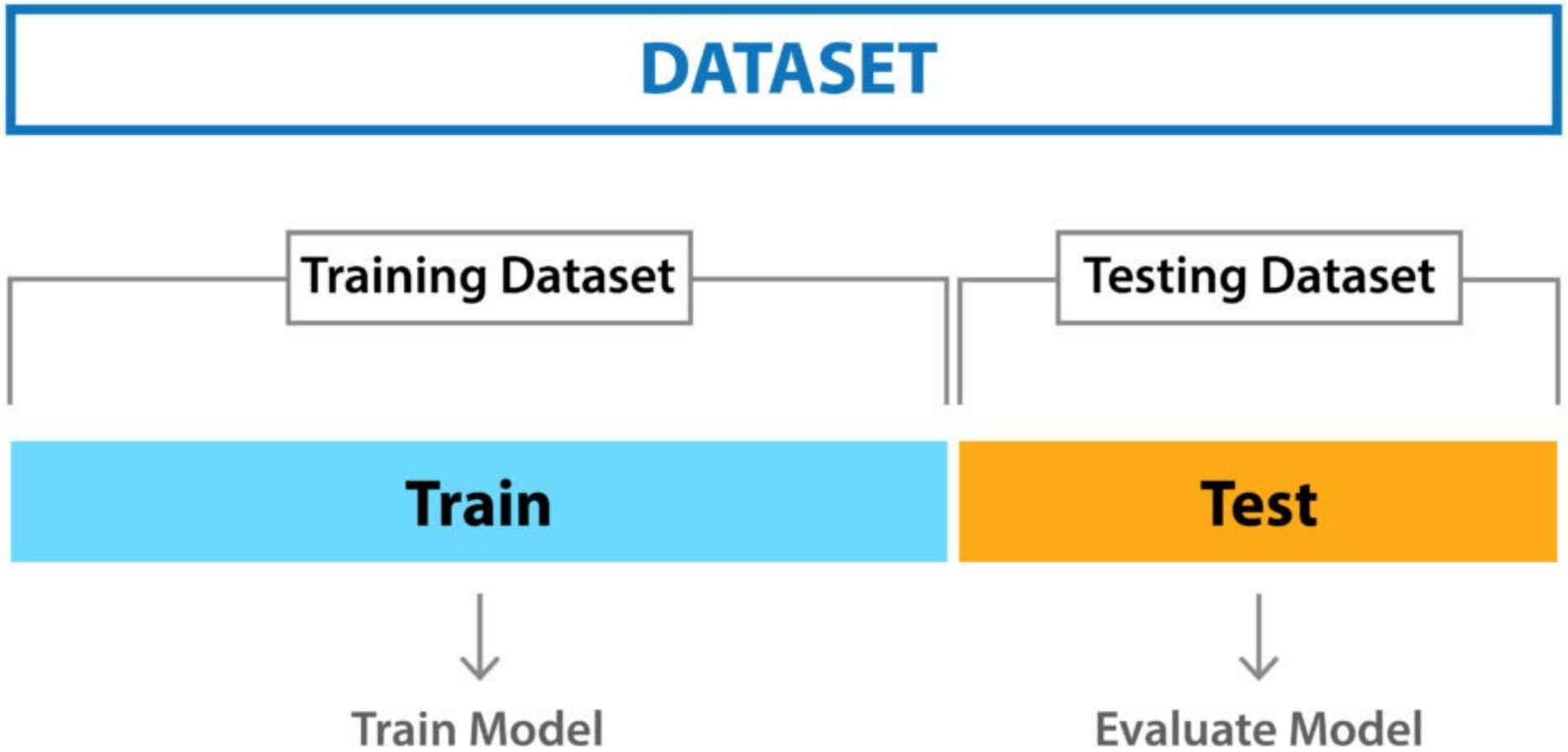
KNN Shortcoming

- If some attributes (coordinates of x) have larger ranges, they are treated as more important. Normalize scale.
- Irrelevant, correlated attributes add noise to distance measure. eliminate some attributes or vary and possibly adapt weight of attributes
- High Dimensional Data: "Curse of Dimensionality" - Computational cost also increases!
- Expensive at test time. Use subset of dimensions; Remove redundant data.
- Storage Requirements!

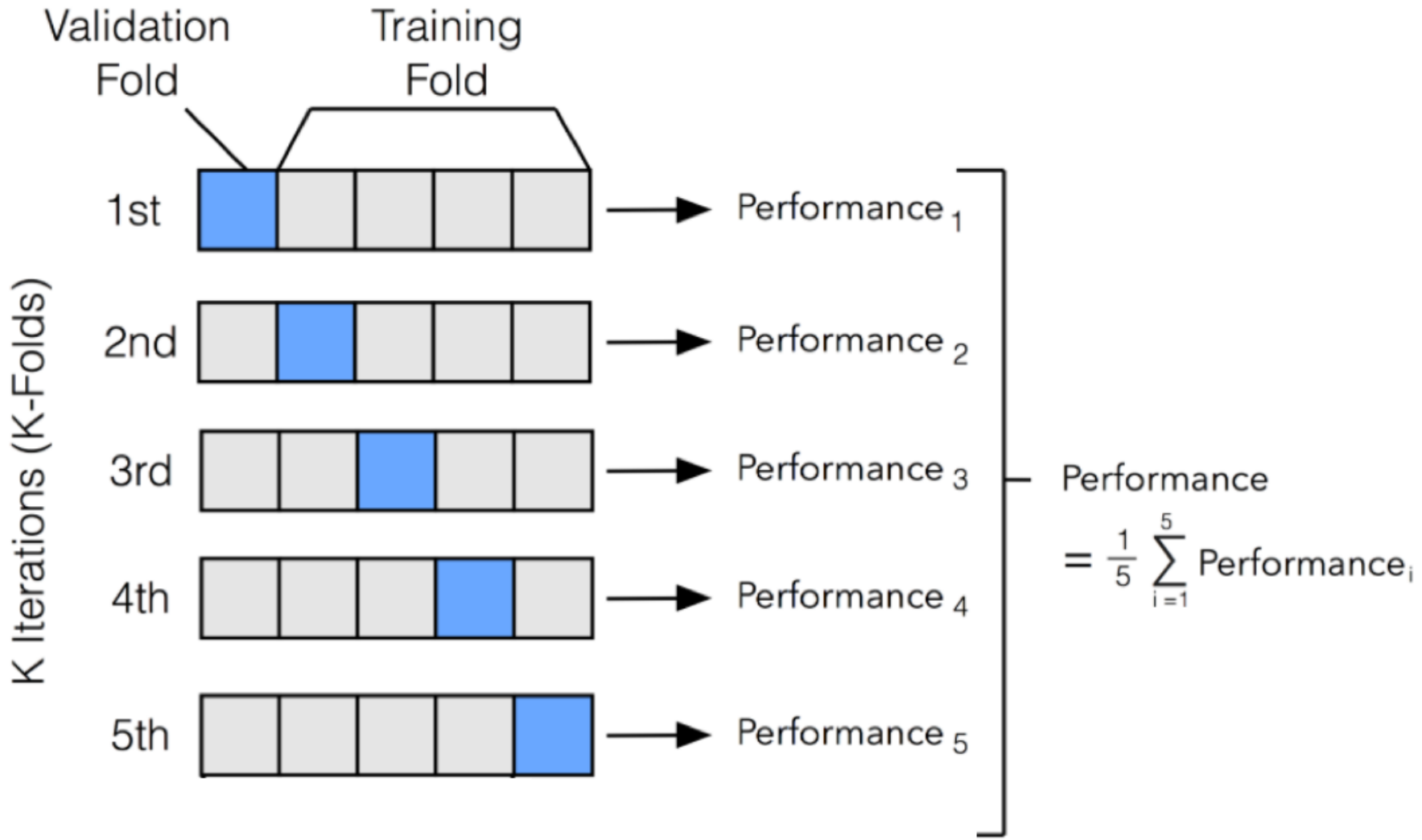
How to evaluate your model?: Cross Validation



Originally…

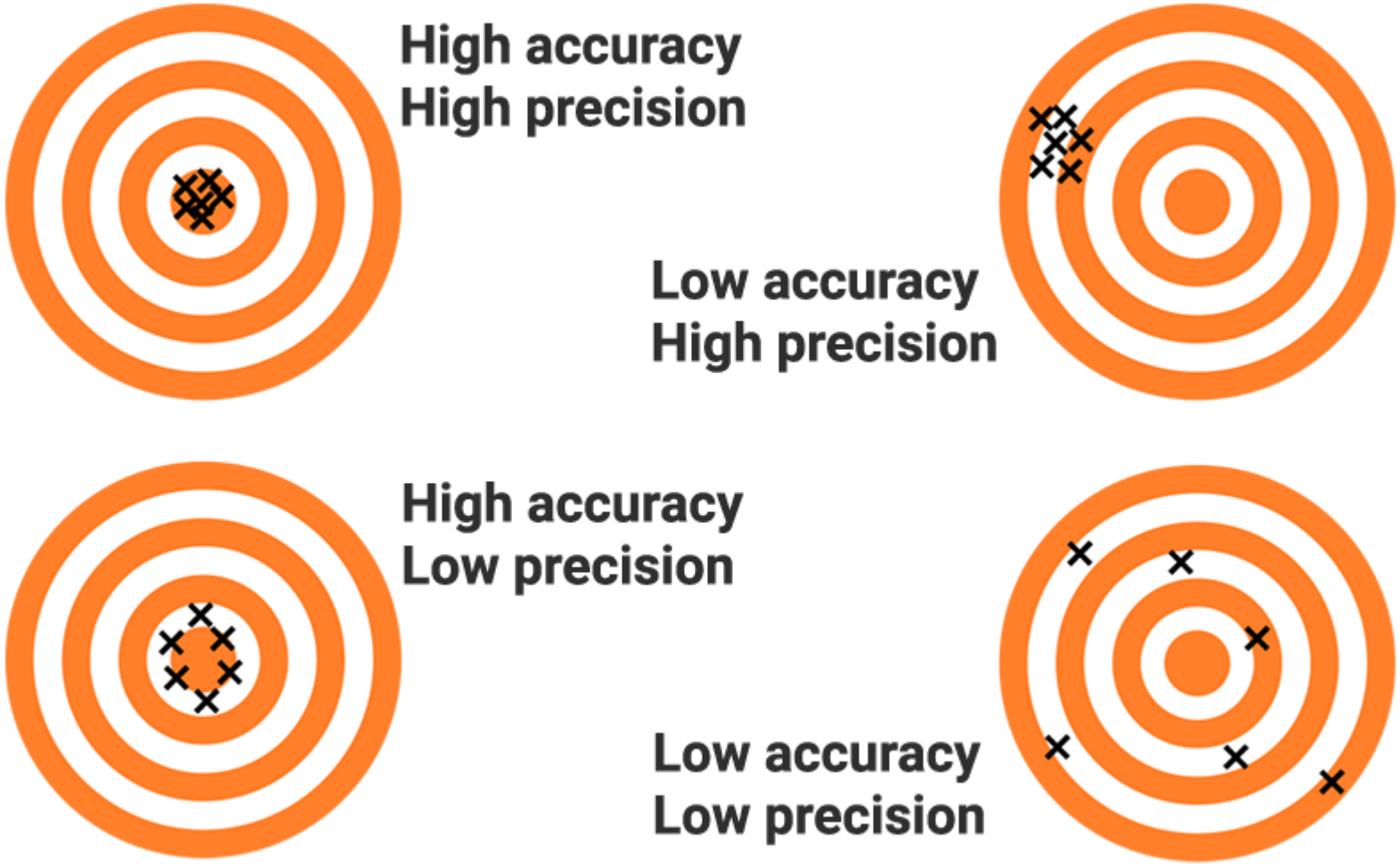

DATASET
Training Dataset
Testing Dataset
Train
Test
Train Model
Evaluate Model

K-Folds: Divide the datasets into k number



Other important scores in Classification

Accuracy / Precision v. Recall / F1 Score / ROC Curve / AUC

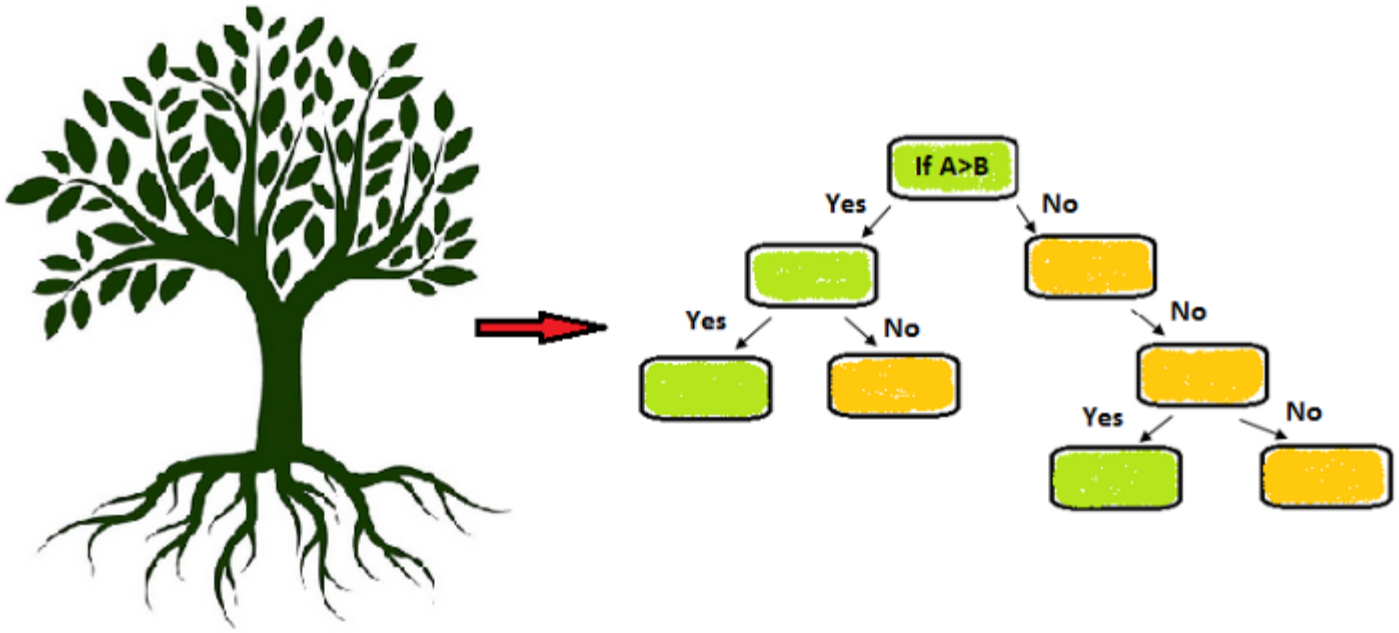


Machine Learning and its Applications

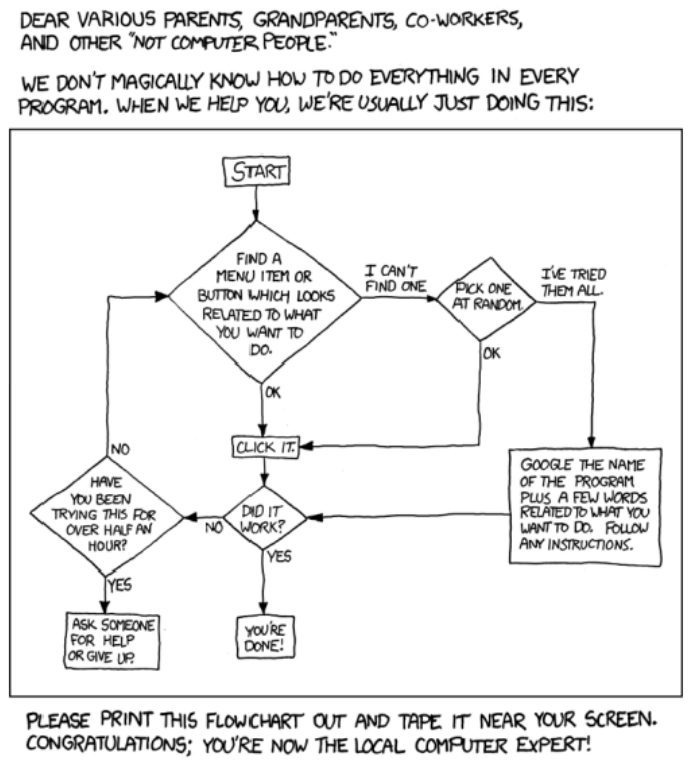
Supervised Learning: Decision Tree in Classification

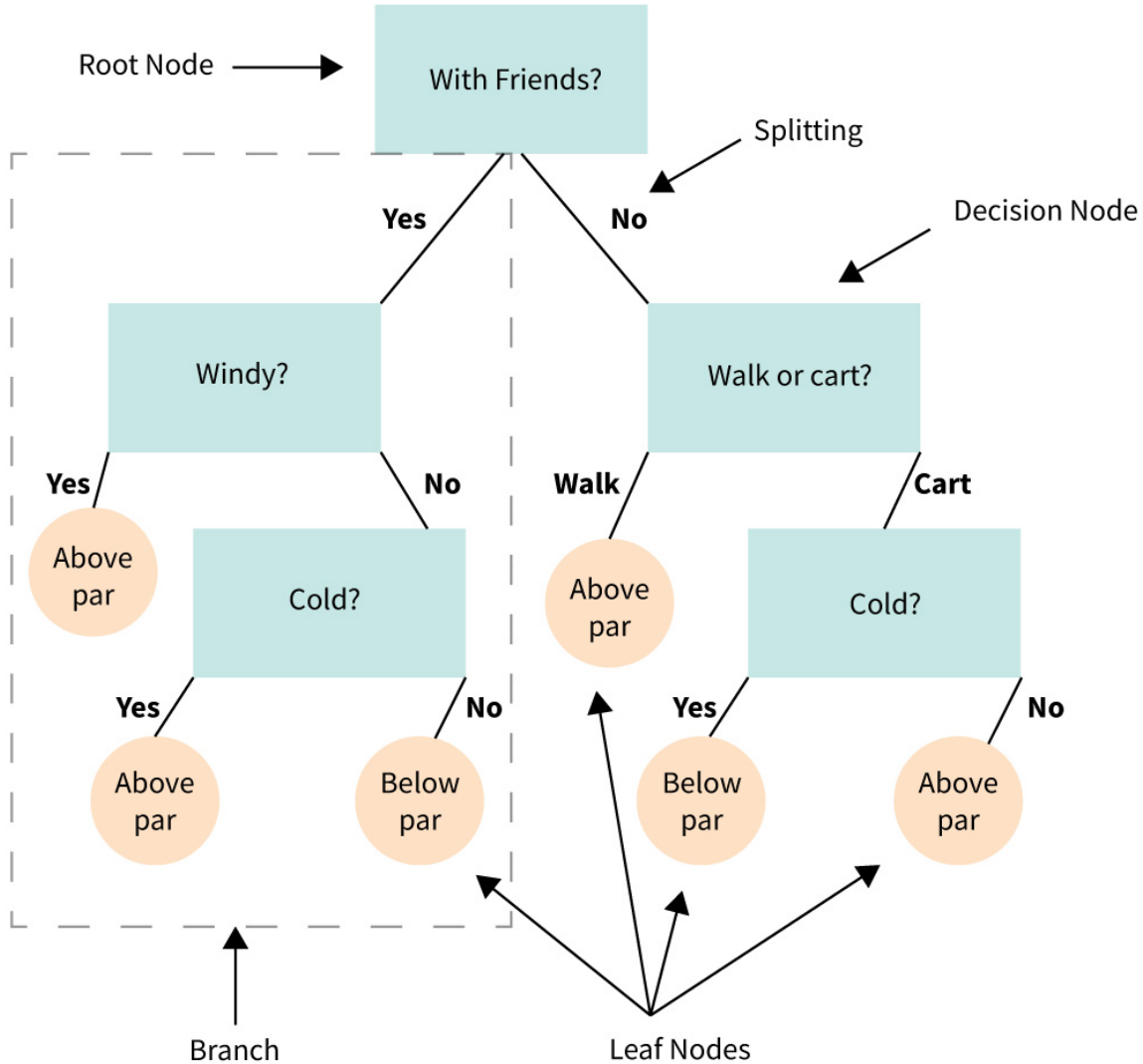
Urban Information Lab

Introduction to Decision Tree

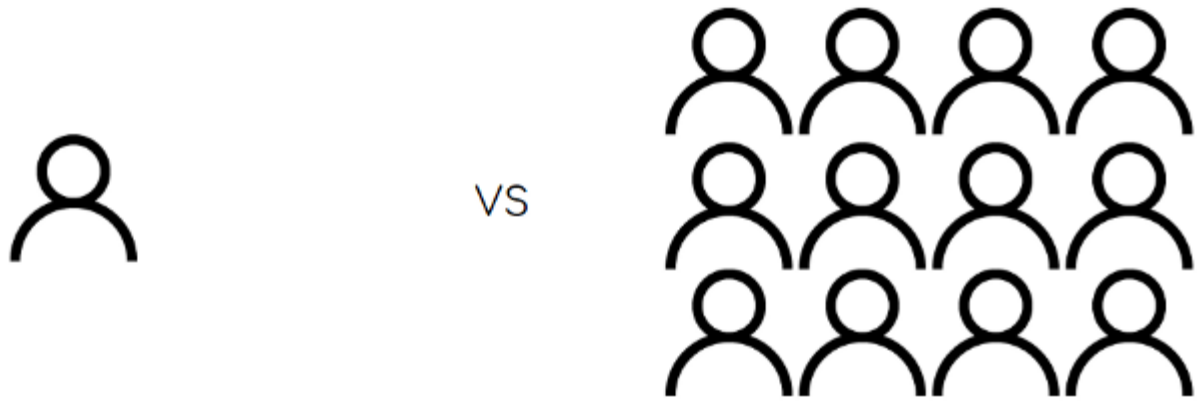


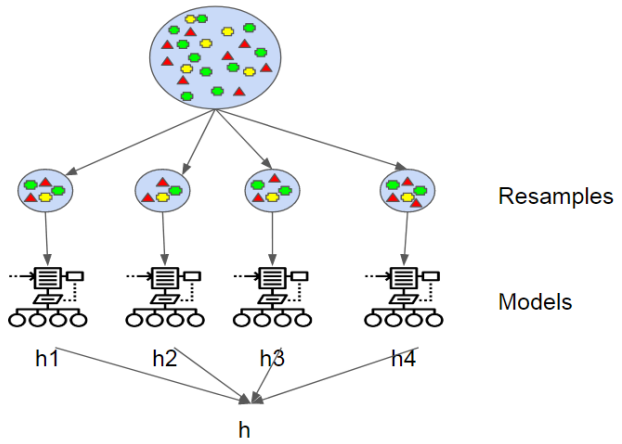
Decision Flowchart





Two Heads are better than one





Decision Tree

Ensemble Learning

Voting

Bagging

Boosting

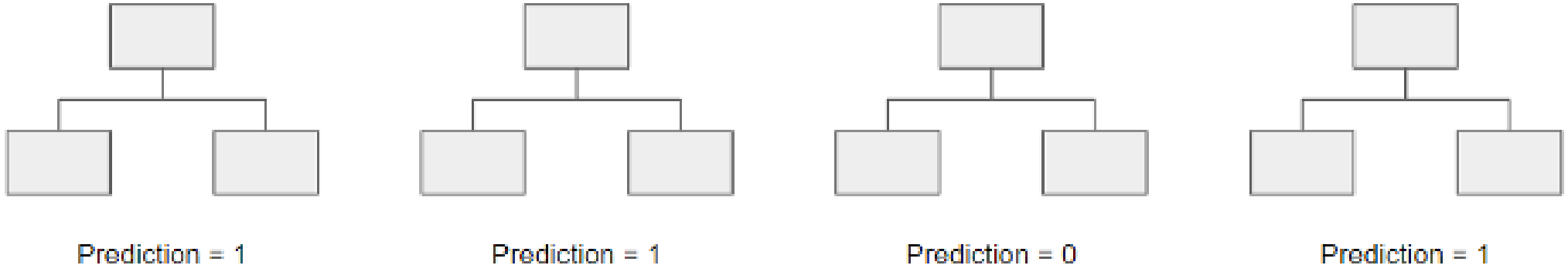
Staking

Regression

Classification

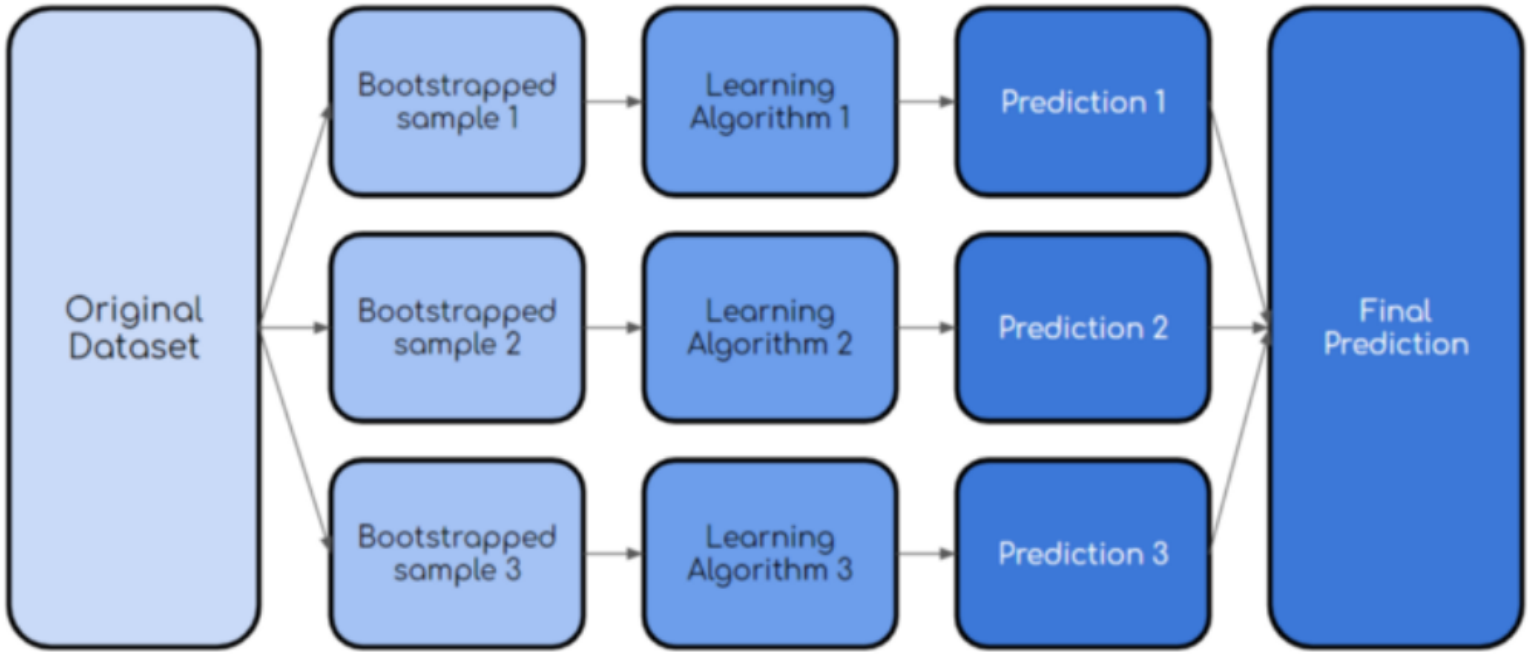
Random Forest (Bagging)

Random forests are an ensemble learning technique that builds off of decision trees. Random forests involve creating multiple decision trees using bootstrapped datasets of the original data. The model then selects the mode (the majority) of all of the predictions of each decision tree. What's the point of this? By relying on a "majority wins" model, it reduces the risk of error from an individual tree.



What is Bagging? = Bootstrap aggregating

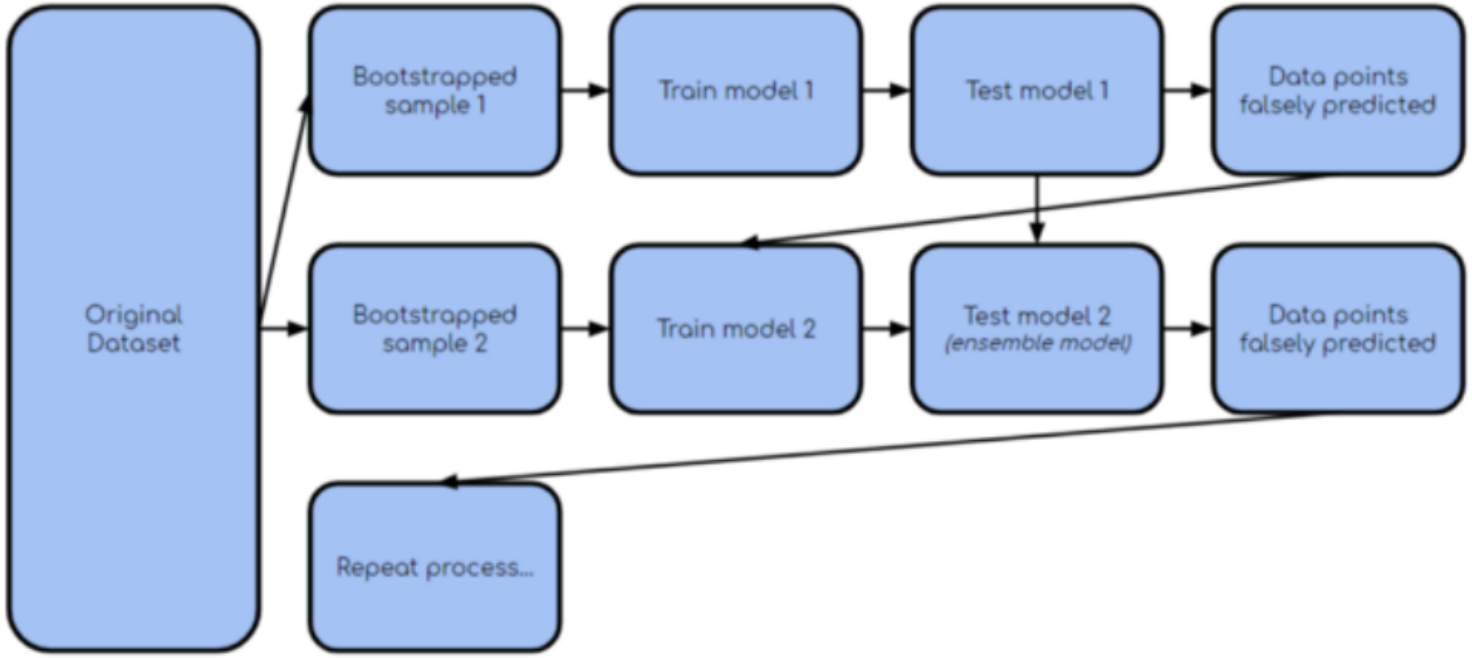
Bagging, also known as **bootstrap aggregating**, is the process in which multiple models of the same learning algorithm are trained with bootstrapped samples of the original dataset. Then, like the random forest example above, a vote is taken on all of the models' outputs.



Bagging Process

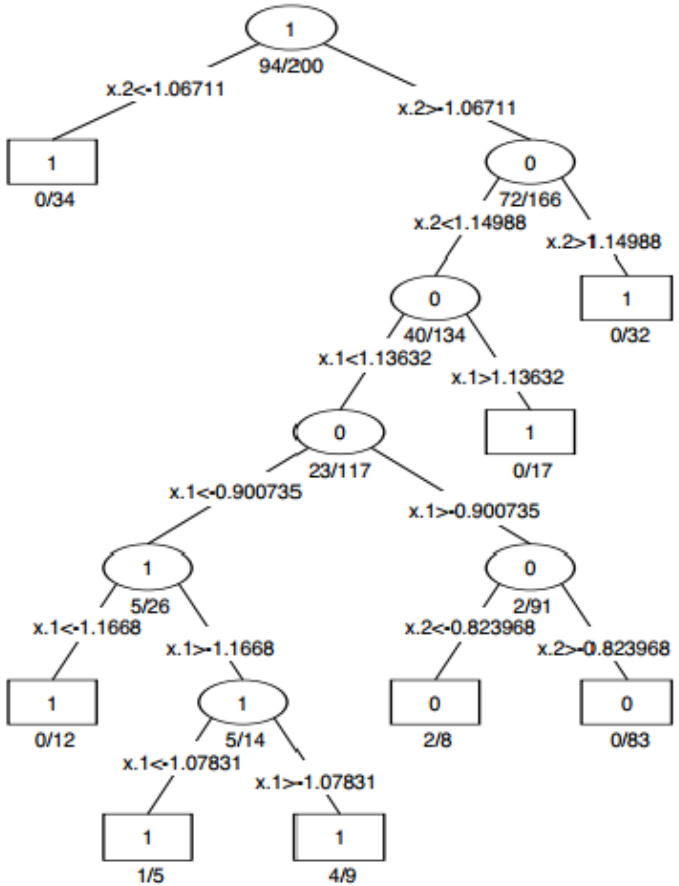
What is Boosting?

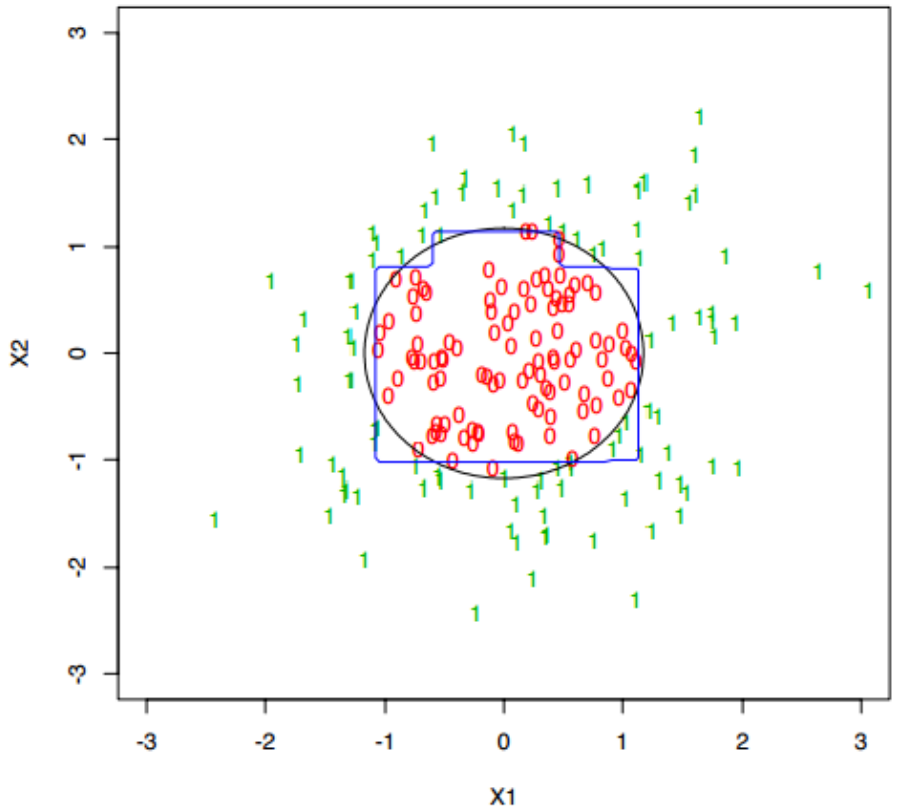
Boosting is a variation of bagging where each individual model is built sequentially, iterating over the previous one. Specifically, any data points that are falsely classified by the previous model is emphasized in the following model. This is done to improve the overall accuracy of the model. Here's a diagram to make more sense of the process:



- **Bagging** (Bootstrap Aggregation) is used when our goal is to reduce the variance of a decision tree. Here idea is to create several subsets of data from training sample chosen randomly with replacement. Now, each collection of subset data is used to train their decision trees. As a result, we end up with an ensemble of different models. Average of all the predictions from different trees are used which is more robust than a single decision tree.
- **Random Forest** is an extension over bagging. It takes one extra step where in addition to taking the random subset of data, it also takes the random selection of features rather than using all features to grow trees. When you have many random trees. It's called Random Forest

Toy Example: Decision Tree





- Sample size: 200
- 7 branching nodes; 6 layers.
- Classification error: 7.3% when $d = 2$; $> 30\%$ when $d = 10$.

Model Averaging

Decision trees can be simple, but often produce noisy (bushy) or weak (stunted) classifiers.

- Bagging (Breiman, 1996): Fit many large trees to bootstrap-resampled versions of the training data, and classify by majority vote.
- Boosting (Freund & Shapire, 1996): Fit many large or small trees to reweighted versions of the training data. Classify by weighted majority vote.
- Random Forests (Breiman 1999): Fancier version of bagging.

In general Boosting>Random Forests>Bagging>Single Tree.

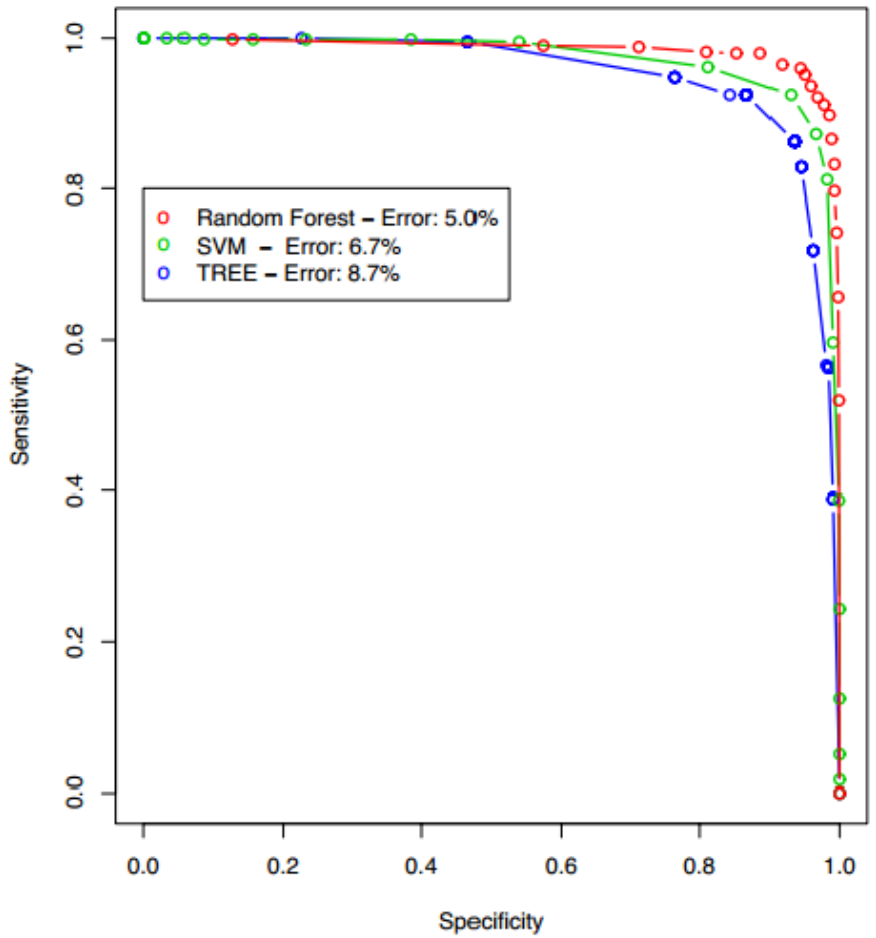
Advantages

- Handles higher dimensionality data very well.
- Handles missing values and maintains accuracy for missing data.

Disadvantages

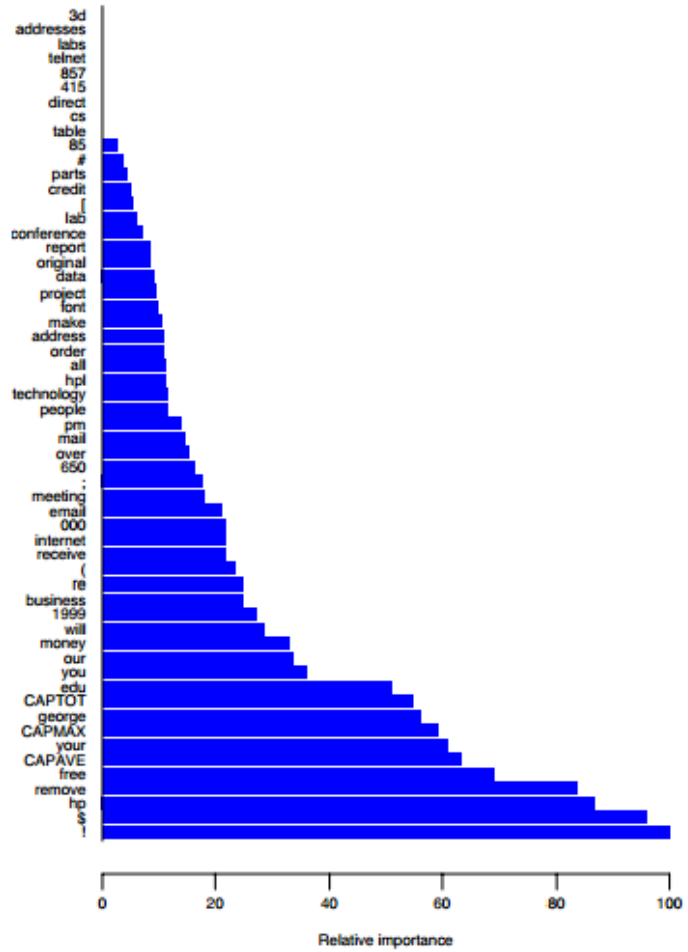
- Since final prediction is based on the mean predictions from subset trees, it won't give precise values for the regression model.

Random Forest for Spam Classification



- RF outperforms SVM.
- 500 Trees.

Random Forest: Variable Importance Scores



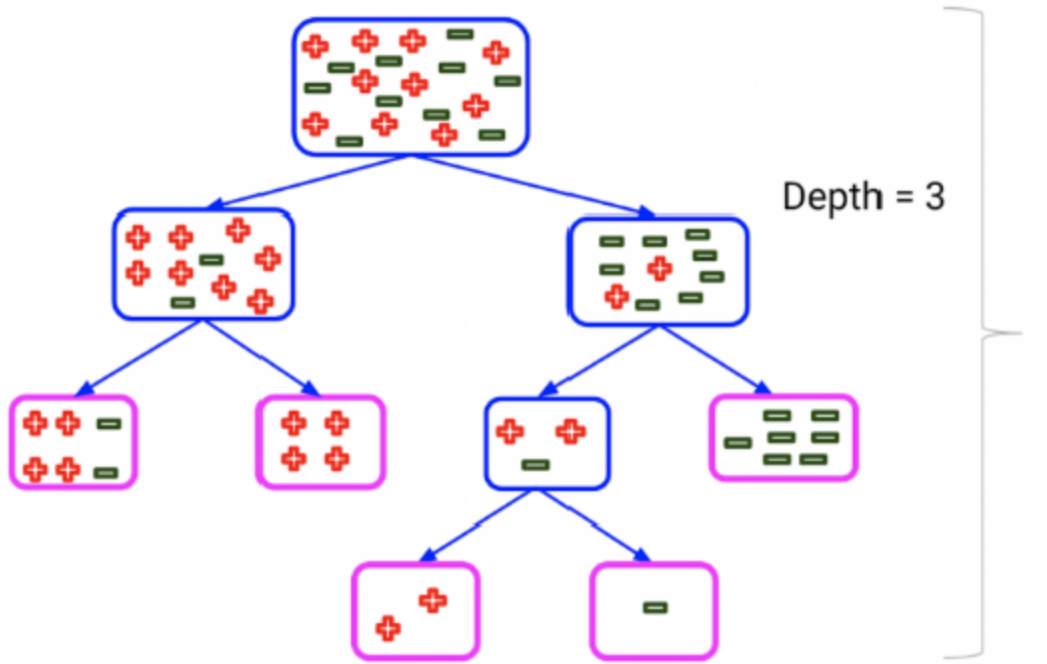
- The bootstrap roughly covers $1/3$ samples for each time.
- Do permutations over variables to check how important they are.

Hyperparameters

- max_depth
- min_sample_split
- max_leaf_nodes
- min_samples_leaf
- n_estimators
- max_sample (bootstrap sample)
- max_features

Random Forest Hyperparameter #1: max_depth

Let's discuss the critical *max_depth* hyperparameter first. The *max_depth* of a tree in Random Forest is defined as the longest path between the root node and the leaf node:



Using the *max_depth* parameter, I can limit up to what depth I want every tree in my random forest to grow.

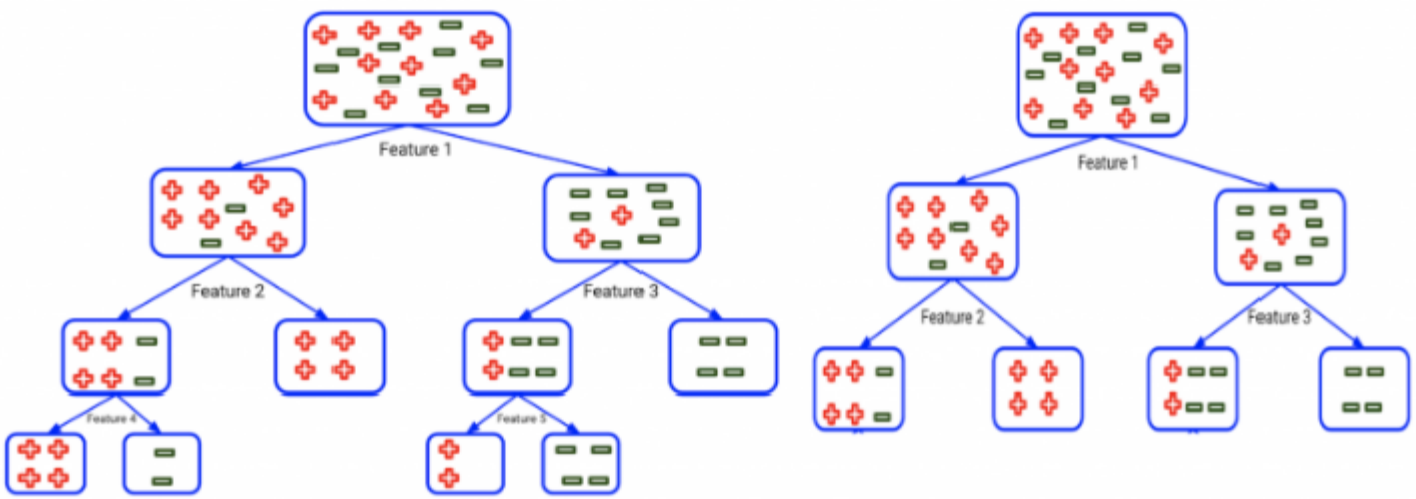
https://www.analyticsvidhya.com/blog/2020/03/beginners-guide-random-forest-hyperparameter-tuning/

Random Forest Hyperparameter #2: min_sample_split

> *min_sample_split – a parameter that tells the decision tree in a random forest the minimum required number of observations in any given node in order to split it.*

The default value of the minimum_sample_split is assigned to 2. This means that if any terminal node has more than two observations and is not a pure node, we can split it further into subnodes.

Having a default value as 2 poses the issue that a tree often keeps on splitting until the nodes are completely pure. As a result, the tree grows in size and therefore overfits the data.

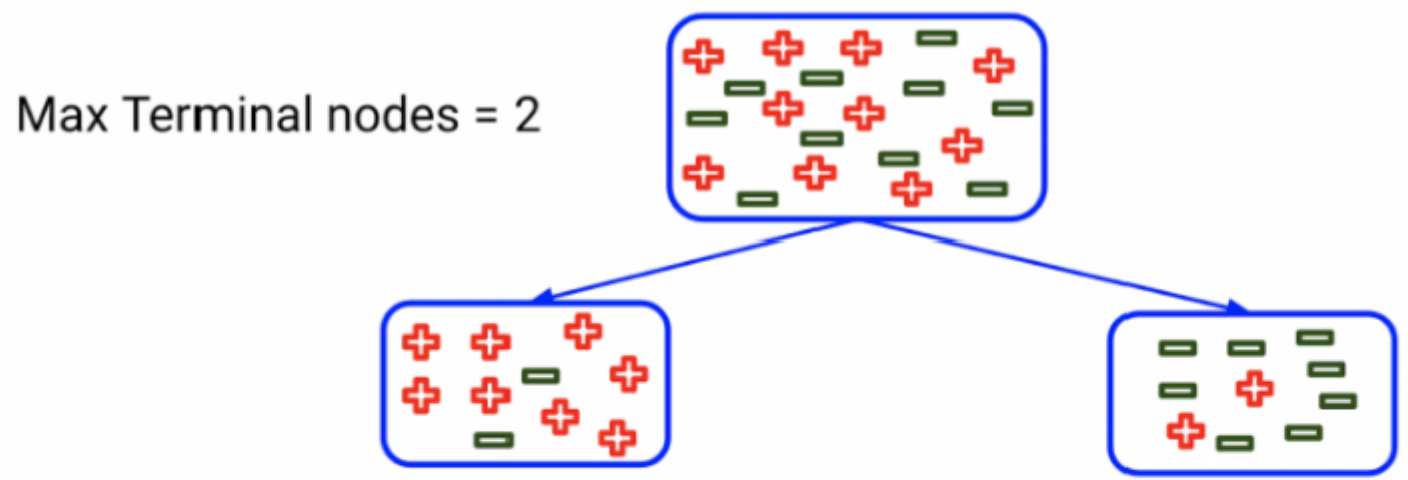


By increasing the value of the *min_sample_split*, we can reduce the number of splits that happen in the decision tree and therefore prevent the model from overfitting. In the above example, if we increase the *min_sample_split* value from 2 to 6, the tree on the left would then look like the tree on the right.

Random Forest Hyperparameter #3: max_terminal_nodes

Next, let's move on to another Random Forest hyperparameter called *max_leaf_nodes*. **This hyperparameter sets a condition on the splitting of the nodes in the tree and hence restricts the growth of the tree.** If after splitting we have more terminal nodes than the specified number of terminal nodes, it will stop the splitting and the tree will not grow further.

Let's say we set the maximum terminal nodes as 2 in this case. As there is only one node, it will allow the tree to grow further:

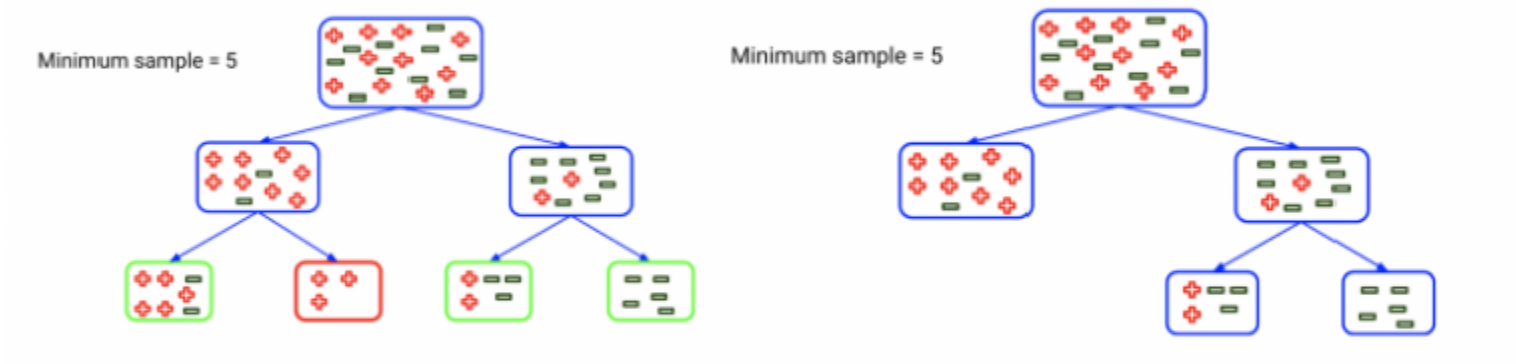


Now, after the first split, you can see that there are 2 nodes here and we have set the maximum terminal nodes as 2. Hence, the tree will terminate here and will not grow further. This is how setting the maximum terminal nodes or *max_leaf_nodes* can help us in preventing overfitting.

Random Forest Hyperparameter #4: min_samples_leaf

Time to shift our focus to *min_sample_leaf*. This Random Forest hyperparameter specifies the minimum number of samples that should be present in the leaf node **after splitting** a node.

Let's understand *min_sample_leaf* using an example. Let's say we have set the minimum samples for a terminal node as 5:



The tree on the left represents an unconstrained tree. Here, the nodes marked with green color satisfy the condition as they have a minimum of 5 samples. Hence, they will be treated as the leaf or terminal nodes.

However, the red node has only 3 samples and hence it will not be considered as the leaf node. Its parent node will become the leaf node. That's why the tree on the right represents the results when we set the minimum samples for the terminal node as 5.

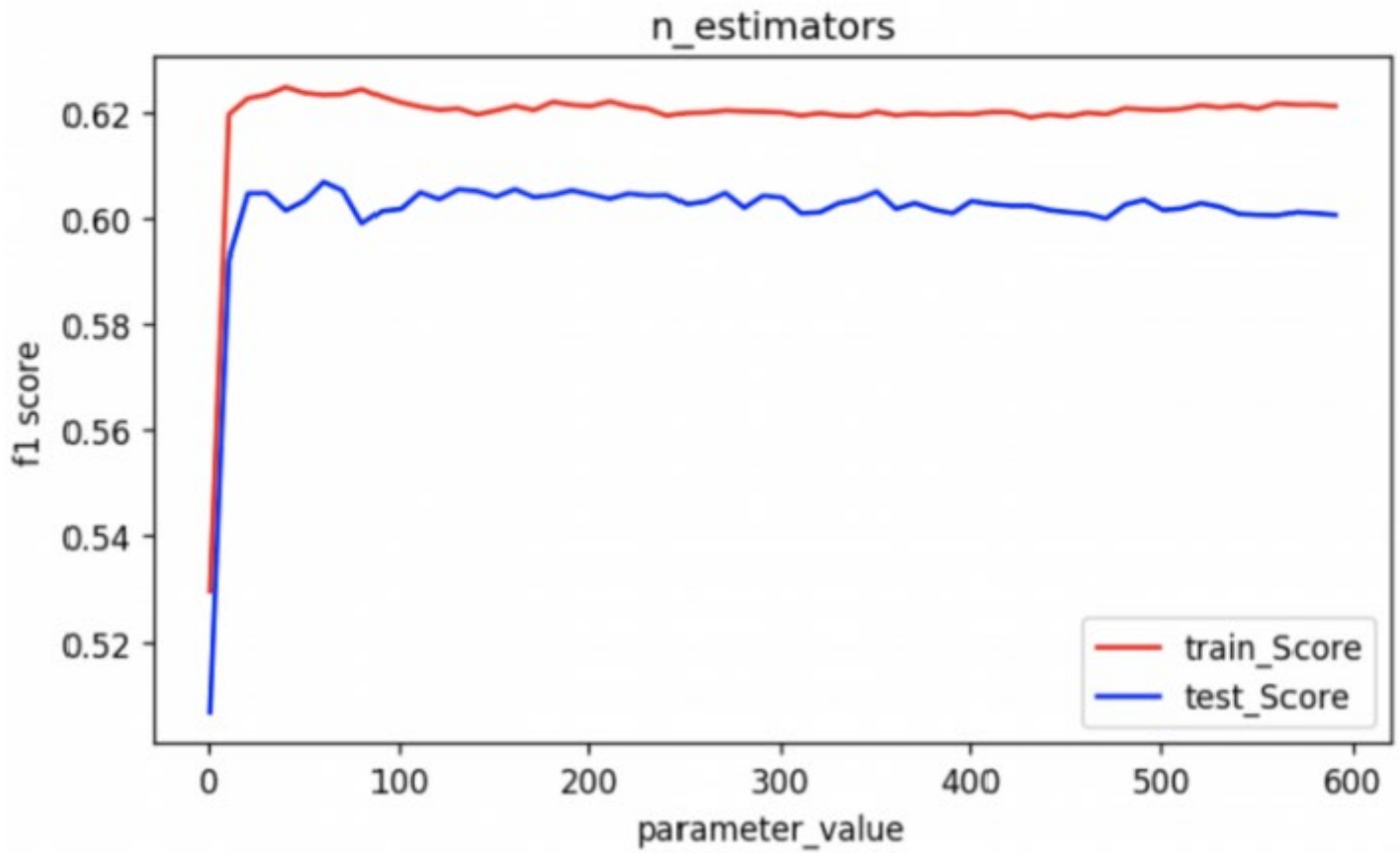
So, we have controlled the growth of the tree by setting a minimum sample criterion for terminal nodes. As you would have guessed, similar to the two hyperparameters mentioned above, this hyperparameter also helps prevent overfitting as the parameter value increases.

Random Forest Hyperparameter #5: n_estimators

We know that a Random Forest algorithm is nothing but a grouping of trees. But how many trees should we consider? That's a common question fresher data scientists ask. And it's a valid one!

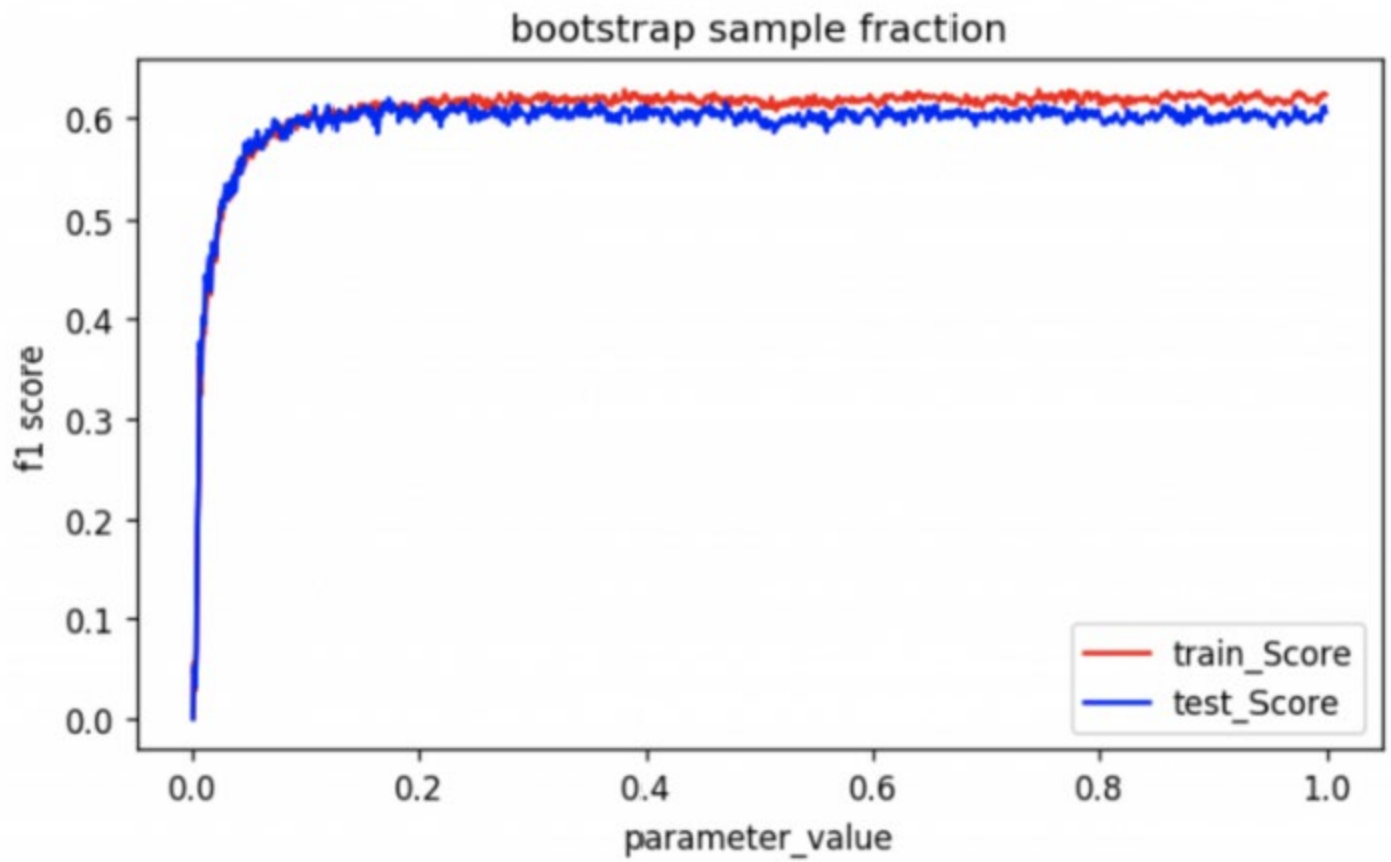
We might say that more trees should be able to produce a more generalized result, right? But by choosing more number of trees, the time complexity of the Random Forest model also increases.

In this graph, we can clearly see that the performance of the model sharply increases and then stagnates at a certain level:



Random Forest Hyperparameter #6: max_samples

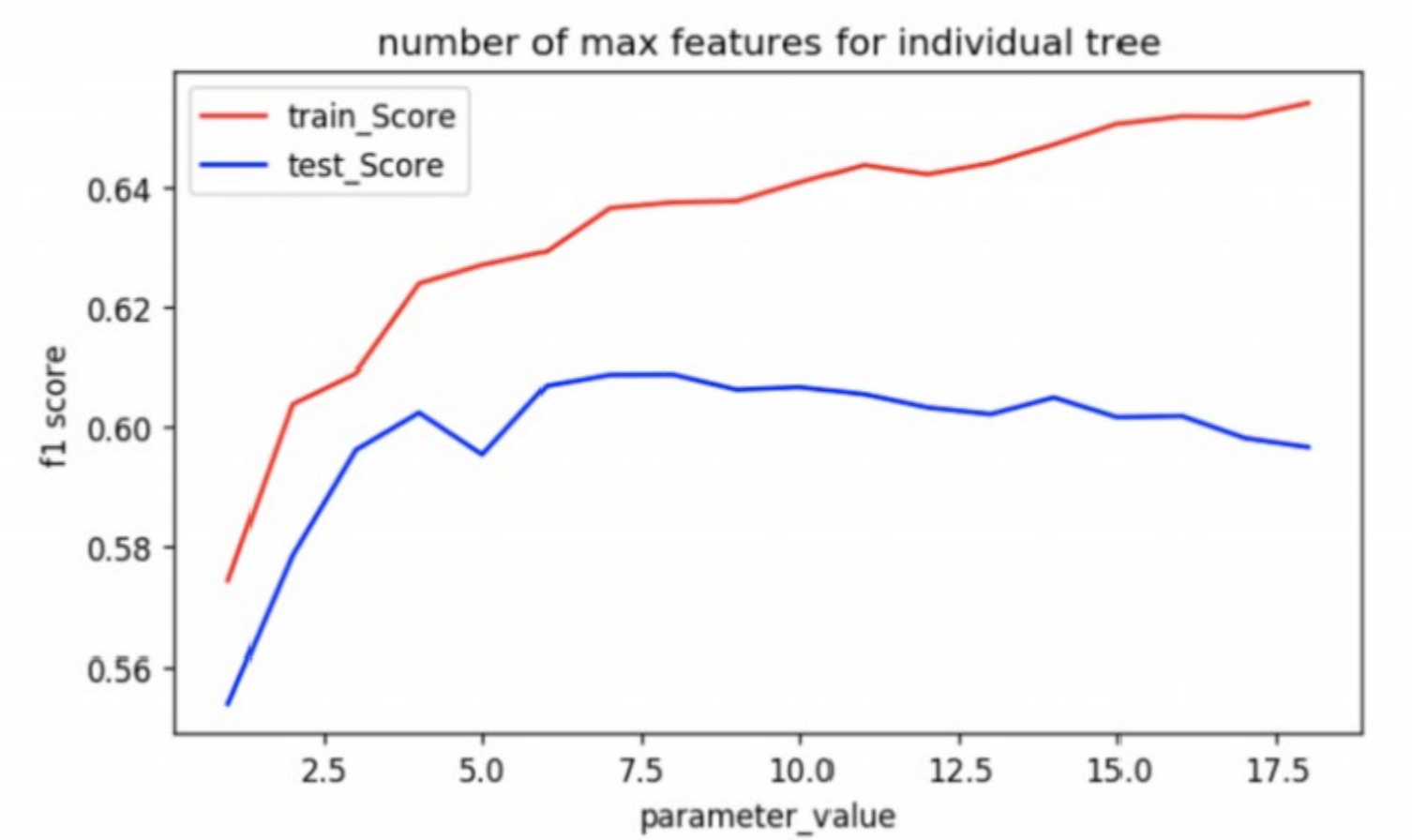
The *max_samples* hyperparameter determines what fraction of the original dataset is given to any individual tree. You might be thinking that more data is always better. Let's try to see if that makes sense.



Random Forest Hyperparameter #7: max_features

Finally, we will observe the effect of the *max_features* hyperparameter. This resembles the number of maximum features provided to each tree in a random forest.

We know that random forest chooses some random samples from the features to find the best split. Let's see how varying this parameter can affect our random forest model's performance.



- Gradient Boosting = Gradient Descent + Boosting
- It uses gradient descent algorithm which can optimize any differentiable loss function. An ensemble of trees are built one by one and individual trees are summed sequentially. Next tree tries to recover the loss (difference between actual and predicted values).

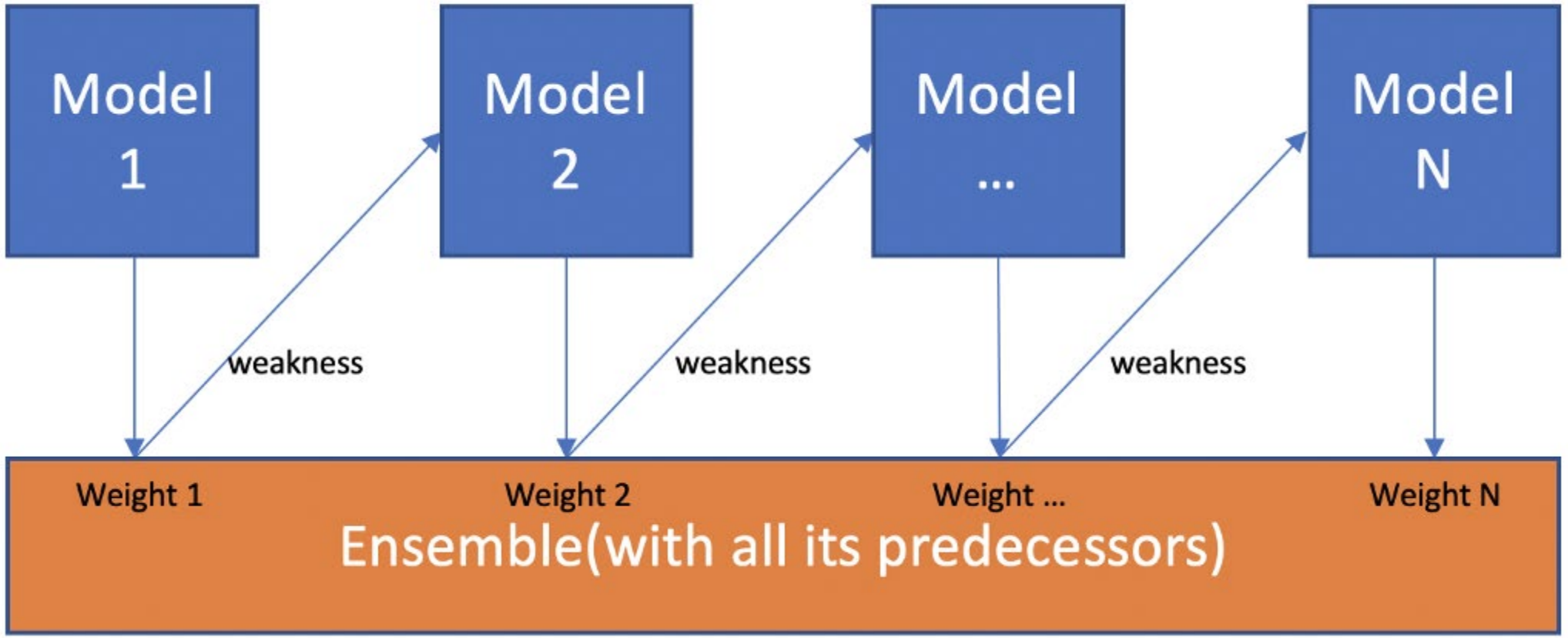
Boosting

Unlike many ML models which focus on high quality prediction done by a single model, boosting algorithms seek to improve the prediction power by training a sequence of weak models, each compensating the weaknesses of its predecessors.

Ex. AdaBoost

Boosting

Model 1,2,…, N are individual models (e.g. decision tree)



Boosting

Strength and Weakness

Strength	Weakness
Easy to interpret	sensitive to outliers
implicit feature selection	hard to scale up (sequential)
resilient to overfitting (in a degree)	slower to train (compared to bagging)
Strong prediction power	

Thank You!