Flipped Class Group 1

Ann Strange, Hongyu Du, Rifei (Faye) Liang, Amy Dye-Robinson, Kirk Hohsfield

colorado school of public health
UNIVERSITY OF COLORADO
COLORADO STATE UNIVERSITY
UNIVERSITY OF NORTHERN COLORADO

Outline

- K-nearest neighbors
- Decision trees
- Random forests
- Gradient boosting
- Summary

Non Parametric Models

- Make no a priori assumptions about input density (e.g. do not assume data was pulled from a distribution).
- Do Assume: Similar inputs have similar outputs

- Different methods/models vary in the way they define similarity, or interpolate from the similar training instances.

KNN Algorithm

colorado school of public health
UNIVERSITY OF COLORADO
COLORADO STATE UNIVERSITY
UNIVERSITY OF NORTHERN COLORADO

K-Nearest Neighbor Algorithm

“**Lazy**!” No decisions are made until new data is received.

For each record to be classified or predicted:

1. Find k records that have similar features
2. For classification, find out what the majority class is among those similar records and assign that class to the new record
3. For prediction (aka KNN Regression), find the average among those similar records and predict that average for the new record

Assumptions:

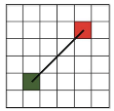
1. (usually): Independence of features
2. Features are on the same scale
3. We have labeled data (Supervised)

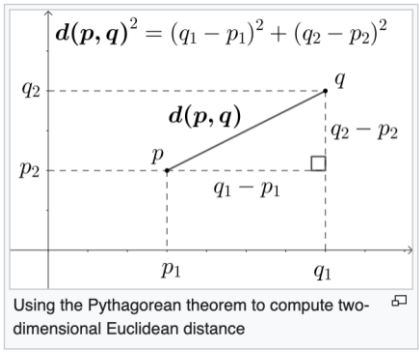
Distance Measures

There are many:

Euclidean (most common): Pythagorean distance between two points.

$$d(\mathbf{a}, \mathbf{b}) = ||\mathbf{a} - \mathbf{b}|| = \sqrt{\sum (a_i - b_i)^2}$$

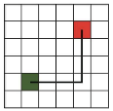




Using the Pythagorean theorem to compute two-dimensional Euclidean distance

Manhattan: (“taxi”) good for higher dimensional data

$$\sum_i |a_i - b_i|$$



Cosine: when directionality (of vectors) matters, used often to compare documents for similar text

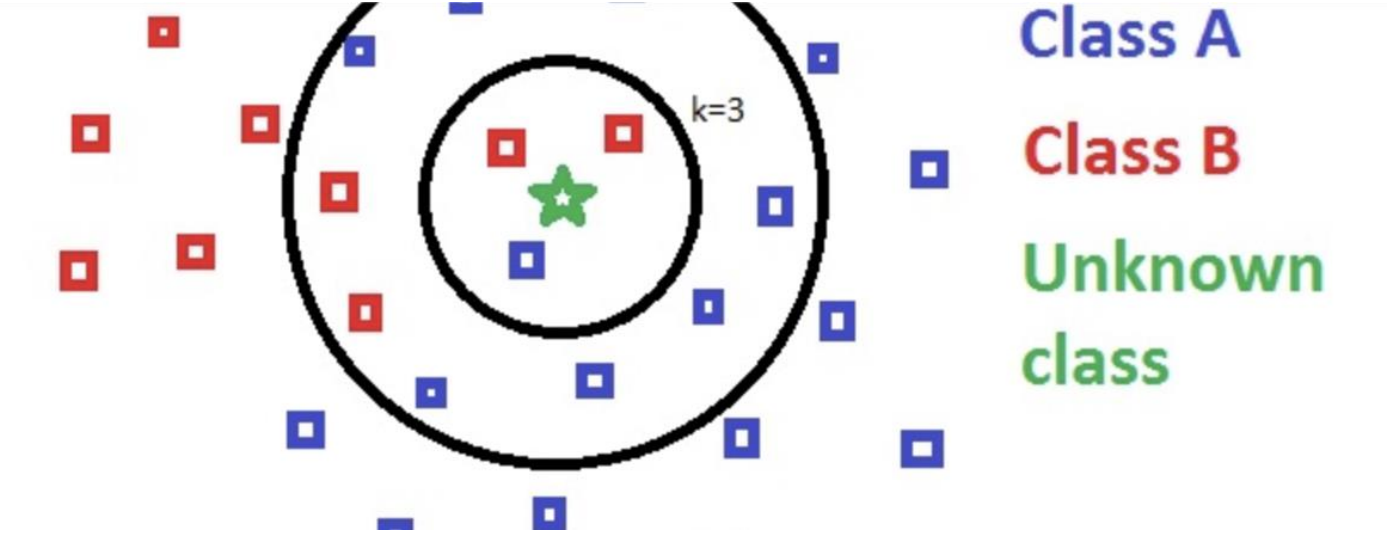
$$\text{cosine similarity} = S_C(A, B) := \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \cdot \sqrt{\sum_{i=1}^{n} B_i^2}},$$

Hamming Distance: compares two binary strings for bits in common: good for comparing two sets of OneHotEncoded categorical data.

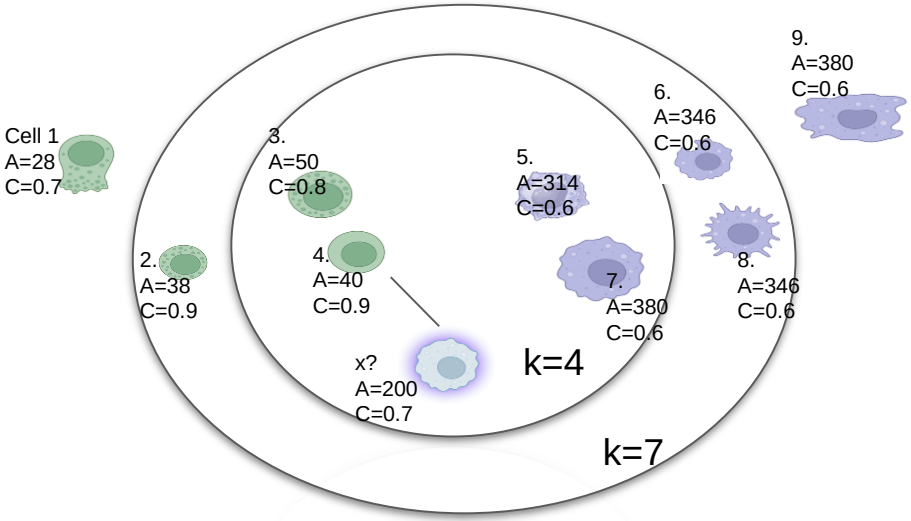
Mahalanobis: takes covariance into account (also normalizes for you): $D(x, m_i) = (x-m_i)^T S_i^{-1} (x-m_i)$

Types of Aggregations:

1. Regression: For a new input, predict the mean of the nearest k neighbors
2. Classification: For a new input, predict the most common class of the nearest k
3. Weighted: (e.g. by distance)
4. Outlier Detection: Are there no neighbors within a threshold distance?



Classification Example: T-cell or Macrophage?



New cell: closest k clustered cells, majority class will be assigned to the new cell.
Features: Cell Area(um), Circularity (0-1.0)

Euclidean Distance: d(x,y) = $\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$

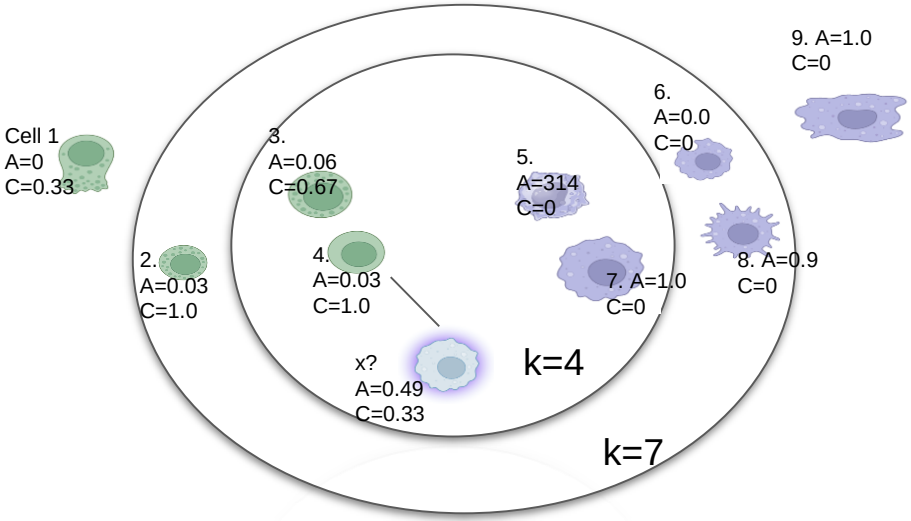
New cell x: Area = 200, Circularity = 0.7

d(x, cell4) = $\sqrt{}$ ((x.Area - cell4.Area)^2 - (x.Circularity - cell4.Circularity)^2)

d(x, cell4) = $\sqrt{}$ ((200-40)^2 + (0.7-0.7)^2)

= 260

Classification Example: T-cell or Macrophage?



SCALED

New cell: closest k clustered cells, majority class will be assigned to the new cell.
Features: Cell Area(um), Circularity (0-1.0)

Euclidean Distance: d(x,y) = $\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$

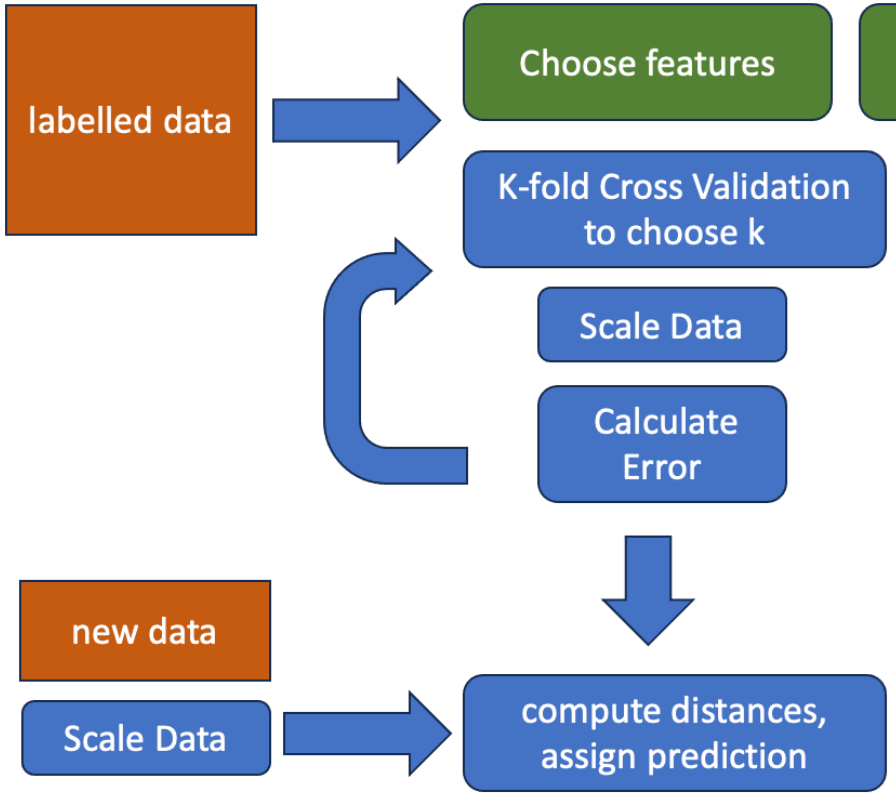
New cell x: Area = 200-> 0.49, Circularity = 0.7->0.33

d(x, cell4) = $\sqrt{}$ ((x.Area - cell4.Area)^2 - (x.Circularity - cell4.Circularity)^2)

d(x, cell4) = $\sqrt{}$ ((0.49-0.03)^2 + (0.33-0.33)^2)

= 0.46

KNN - How to Implement

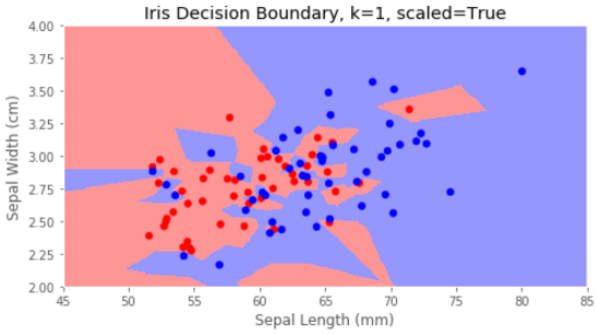


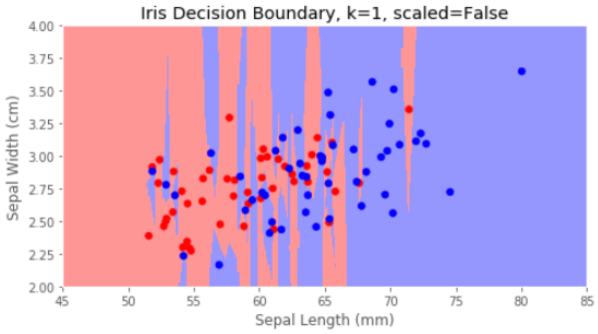
test k from 1 to something less than N
tip: $\sqrt{N}$ is often a good guess (go higher to test).
GridSearchCV or RandomSearchCV

Scale [0,1], [-1,1] with MinMaxScaler or Normalizer.

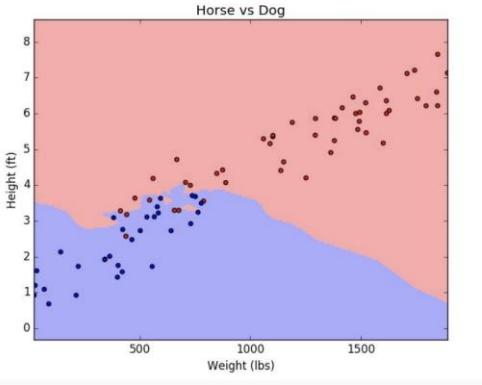
Error function appropriate for job: e.g. Classification, Regression, or Outlier detection

How’s the fit: What problems do you see?

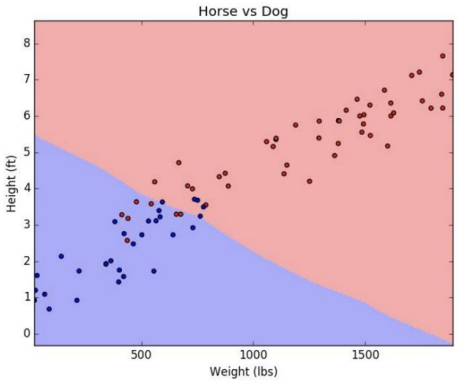




k = 5

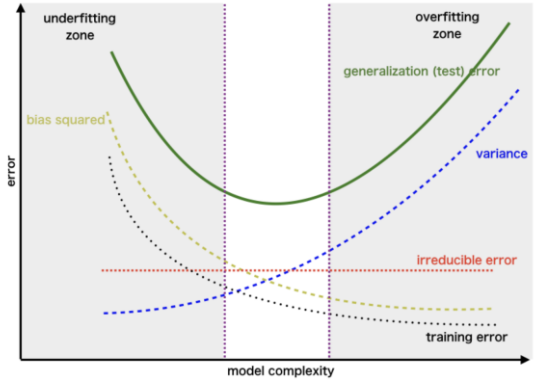


k = 50



Total Error = Bias^2 + Variance + Irreducible Error

The best fit will be given by the hypothesis on the tradeoff point. The error to complexity graph to show trade-off is given as –



Region for the Least Value of Total Error

When to select KNN

- Supervised – When you have good labelled data.
- Somewhat simple/straightforward: explainable. Good for real world applications
- Efficient for small datasets. “lazy learning”: makes it faster than SVMs and linear regression.
- Since it’s non-parametric (not assuming structure of the underlying data), it works without needing to follow a distribution.

When Not to:

- Noisy data - can result in over-fitting
- Computationally expensive (especially for large N): all the training data lives in memory
- Prediction stage may struggle if too many features. E.g. how close is one genetic seq to another? (more features = very large distances) Curse of dimensionality!
- Ensemble methods often perform better

KNN - Applications

- Classification
- Regression
- Impute missing data
- Outlier Detection

Medical Diagnosis

Pattern recognition

Data mining

Intrusion or fraud detection

Text mining

Churn

Facial recognition

Recommendation systems

KNN - Code Outline

```
#Import the necessary libraries
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

iris = load_iris()


data = iris.data
target = iris.target

#Use train_test_split to split data into training and test sets
 X_train, X_test, y_train, y_test = train_test_split(data, target, test_size = 0.25, random_state = 0)

#Instantiate the KNN class and fit to X_train data and labels in y_train

knn = KNN()

knn.fit(X_train, y_train)

#Generate data predictions for data in X_test:
preds = knn.predict(X_test)

print("Testing Accuracy: {}":format(accuracy_score(y_test, preds)))
```

Decision Trees

colorado school of public health
UNIVERSITY OF COLORADO
COLORADO STATE UNIVERSITY
UNIVERSITY OF NORTHERN COLORADO

Contents

(1) Binary Classification

Watermelon: good or bad?

(2) Multi-label Classification

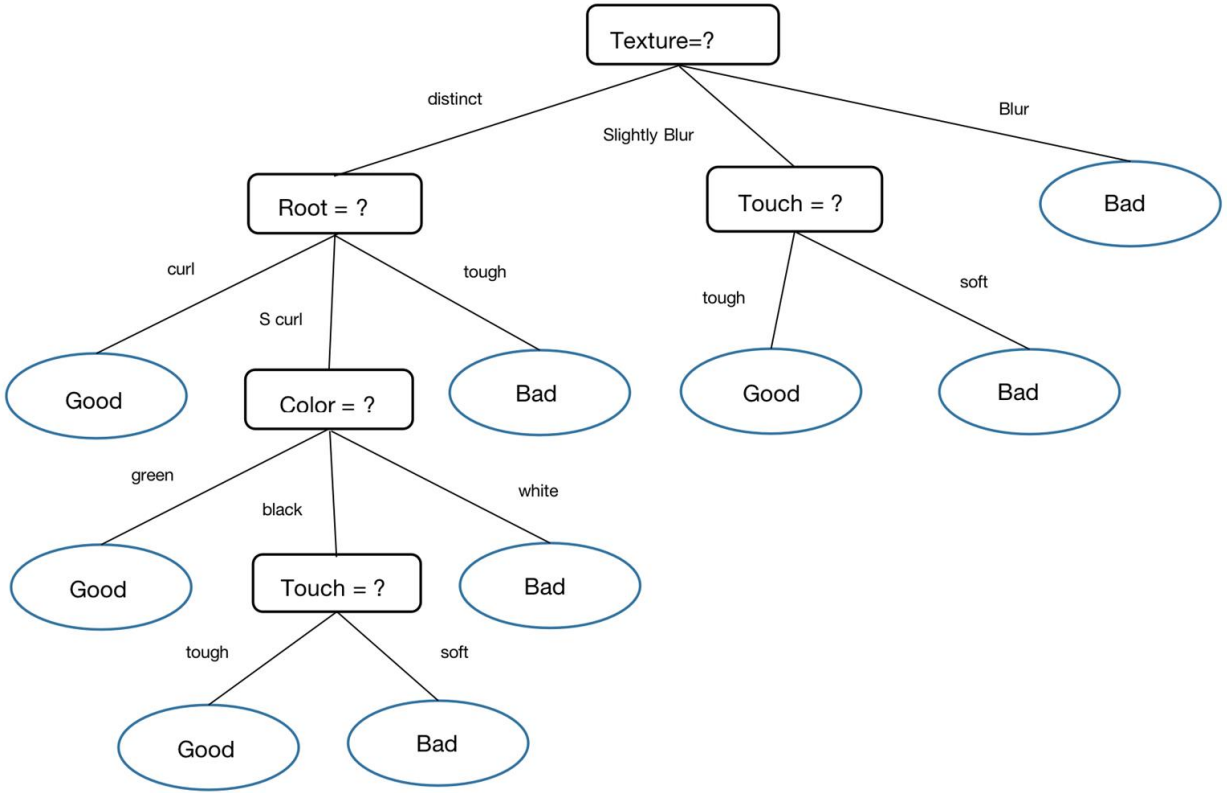
Characteristics: Color, Root, Knock, Texture, Umbilical, Touch

(3) Information Entropy and Information Gain

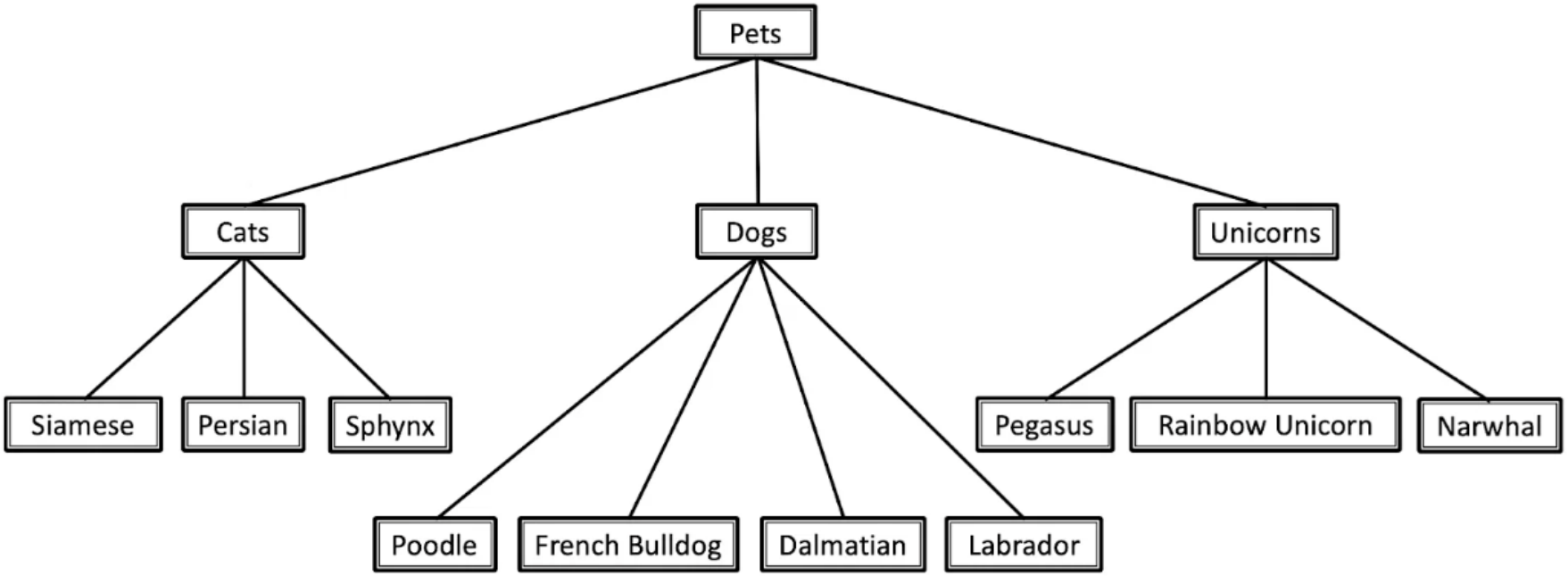
(4) Gini Impurity and Gini Gain

(5) Regression Using Decision Tree

Example of Binary Classification

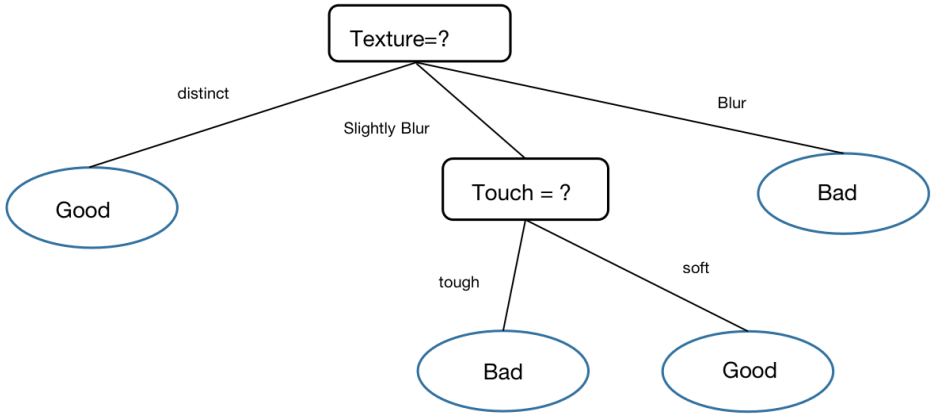


Simple Example of Multi-label Classification



Simple Example of Binary Classification

Number	Root	Texture	Touch	Outcome
1	curl	distinct	tough	Good
2	curl	distinct	tough	Good
7	S curl	S blur	soft	Good
9	S curl	S blur	tough	Bad
11	tough	blur	tough	Bad
12	curl	blur	soft	Bad



Information Entropy and Information Gain

Number	Root	Outcome
1	curl	Good
2	curl	Good
7	S curl	Good
9	S curl	Bad
11	tough	Bad
12	curl	Bad

Information Entropy

$$\mathrm{Ent}(D) = -\sum_{k=1}^{|\mathcal{Y}|} p_k \log_2 p_k \ .$$

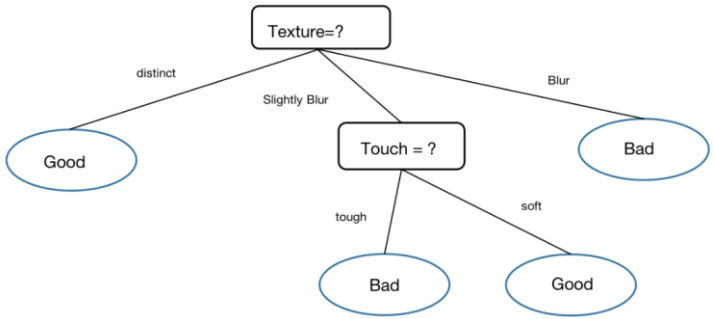
Information Gain

$$Gain(D,a) = Ent(D) - \sum_{v=1}^{V_a} \frac{|D_a^v|}{|D|} Ent(D_a^v)$$

D is the whole dataset; p_k is the ratio of k outcomes; a is the specific characteristic; V_a is the number of levels for characteristic a. D_a^v means the dataset with the v level for a.

Calculation

Number	Root	Texture	Touch	Outcome
1	curl	distinct	tough	Good
2	curl	distinct	tough	Good
7	S curl	S blur	soft	Good
9	S curl	S blur	tough	Bad
11	tough	blur	tough	Bad
12	curl	blur	soft	Bad



$$Ent(D) = -\sum_{k=1}^{2} p_k log_2 p_k = -\left(\frac{3}{6}log_2\frac{3}{6} + \frac{3}{6}log_2\frac{3}{6}\right) = 0.693$$

$$Ent(D_{Scurl}) = -\left(\frac{1}{2}log_2\frac{1}{2} + \frac{1}{2}log_2\frac{1}{2}\right) = 0.693$$

$$Ent(D_{curl}) = -\left(\frac{2}{3}log_2\frac{2}{3} + \frac{1}{3}log_2\frac{1}{3}\right) = 0.637$$

$$Ent(D_{tough}) = 0$$

$$Gain(D, Root) = Ent(D) - \sum_{v=1}^{3} \frac{|D^v|}{D} Ent(D^v)$$

$$= 0.693 - \left(\frac{3}{6} \times 0.637 + \frac{2}{6} \times 0.693 + 0 \times \frac{1}{6}\right) = 0.1435$$

$$Gain(D, Touch) = 0$$

$$Gain(D, Texture) = 0.462$$

$$Gain(D, Touch) = 0.693$$

$$Gain(D, Root) = 0$$

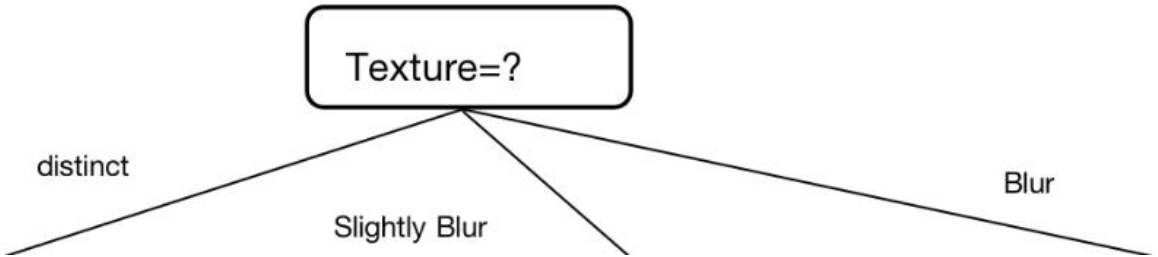
Training Dataset

Number	Color	Root	Knock	Texture	Umbilical	Touch	Outcome
1	green	curl	dull	distinct	concave	tough	Good
2	black	curl	dreary	distinct	concave	tough	Good
3	black	curl	dull	distinct	concave	tough	Good
4	green	curl	dreary	distinct	concave	tough	Good
5	white	curl	dull	distinct	concave	tough	Good
6	green	S curl	dull	distinct	S concave	soft	Good
7	black	S curl	dull	S blur	S concave	soft	Good
8	black	S curl	dull	distinct	S concave	tough	Good
9	black	S curl	clear	S blur	S concave	tough	Bad
10	green	tough	clear	distinct	flat	soft	Bad
11	white	tough	dreary	blur	flat	tough	Bad
12	white	curl	dull	blur	flat	soft	Bad
13	green	S curl	dull	S blur	concave	tough	Bad
14	white	S curl	dreary	S blur	concave	tough	Bad
15	black	S curl	dull	distinct	S concave	soft	Bad
16	white	curl	dull	blur	flat	tough	Bad
17	green	curl	dreary	S blur	S concave	tough	Bad

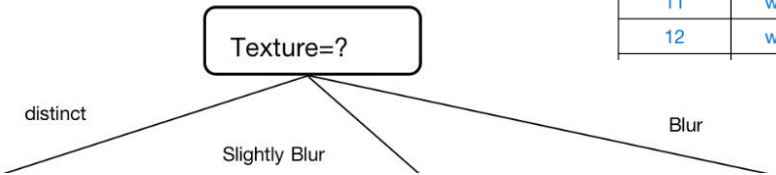
Calculation

Gain(D,Texture)=0.381 Gain(D,Root)=0.143 Gain(D,Color)=0.109

Gain(D,Touch)=0.006 Gain(D,Knock)=0.141 Gain(D,Umbilical)=0.289

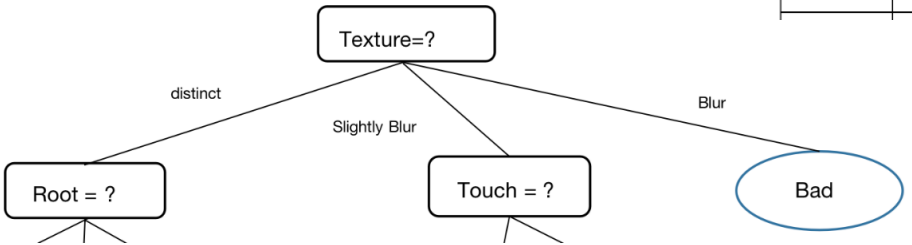


Stop Adding Branches



11	white	tough	dreary	blur	flat	tough	Bad
12	white	curl	dull	blur	flat	soft	Bad

{1,2,3,4,5,6,7,8,10,15} {7,9,13,14,17} {11,12,16}



16	white	curl	dull	blur	flat	tough	Bad

More Calculation

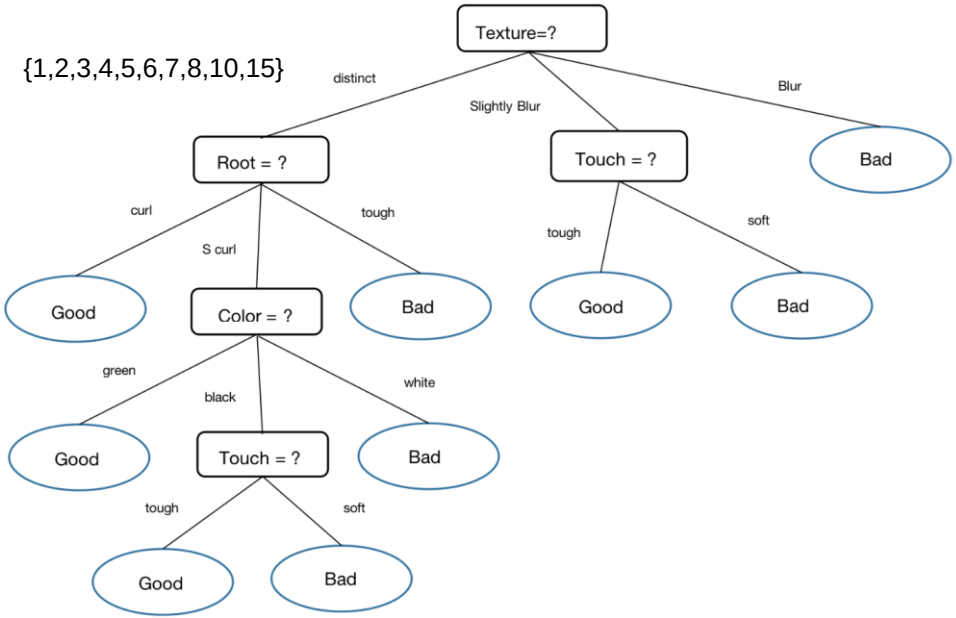
Gain(D,Root)=0.458

Gain(D,Color)=0.043

Gain(D,Touch)=0.458

Gain(D,Knock)=0.331

Gain(D,Umbilical)=0.458



Gini Impurity and Gini Gain

Gini impurity

$$\begin{aligned}\mathrm{Gini}(D) &= \sum_{k=1}^{|\mathcal{Y}|} \sum_{k' \neq k} p_k p_{k'} \\ &= 1 - \sum_{k=1}^{|\mathcal{Y}|} p_k^2 \ .\end{aligned}$$

Information Entropy

$$\mathrm{Ent}(D) = -\sum_{k=1}^{|\mathcal{Y}|} p_k \log_2 p_k \ .$$

Gini Index

$$\mathrm{Gini\_index}(D, a) = \sum_{v=1}^{V} \frac{|D^v|}{|D|} \mathrm{Gini}(D^v) \ .$$

Information Gain

$$Gain(D, a) = Ent(D) - \sum_{v=1}^{V_a} \frac{|D_a^v|}{|D|} Ent(D_a^v)$$

Choice

$$a_* = \arg\min_{a \in A} \mathrm{Gini\_index}(D, a).$$

Difference Between Gini and Information

1. Gini impurity and information entropy are both metrics used in decision tree algorithms for splitting nodes and measuring the “impurity” or uncertainty in a dataset.

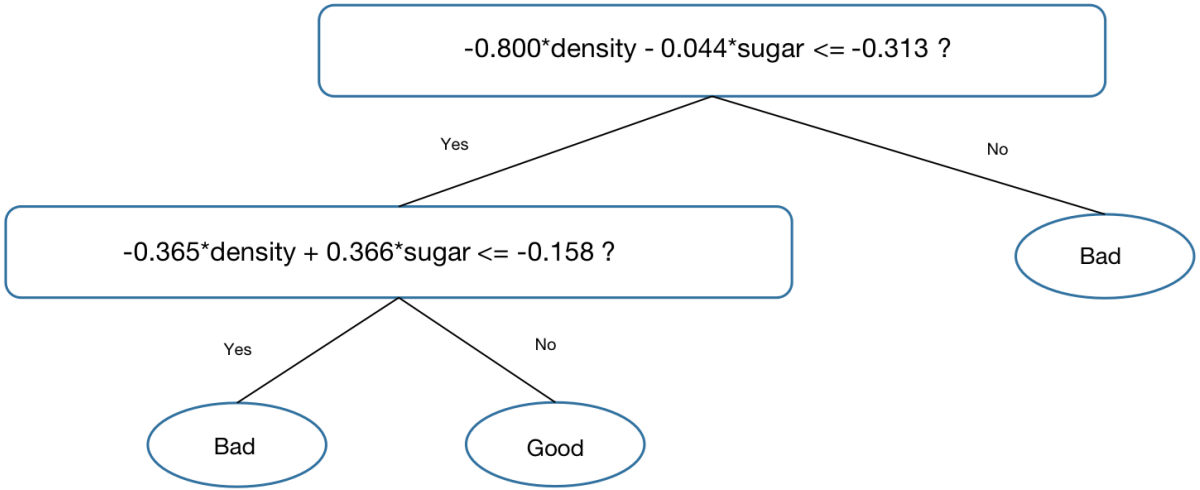
1. Gini impurity tends to create more equally sized partitions and it is slightly faster to compute than entropy. Information entropy can create biased partitions with a preference for smaller partitions with high purity.

1. The Gini index ranges from 0 to 0.5, but entropy ranges from 0 to log(number of classes).

1. Gini impurity is less sensitive to class imbalance compared to entropy. Entropy can be more sensitive to class imbalance, making it tend to overfit in the presence of imbalance datasets.

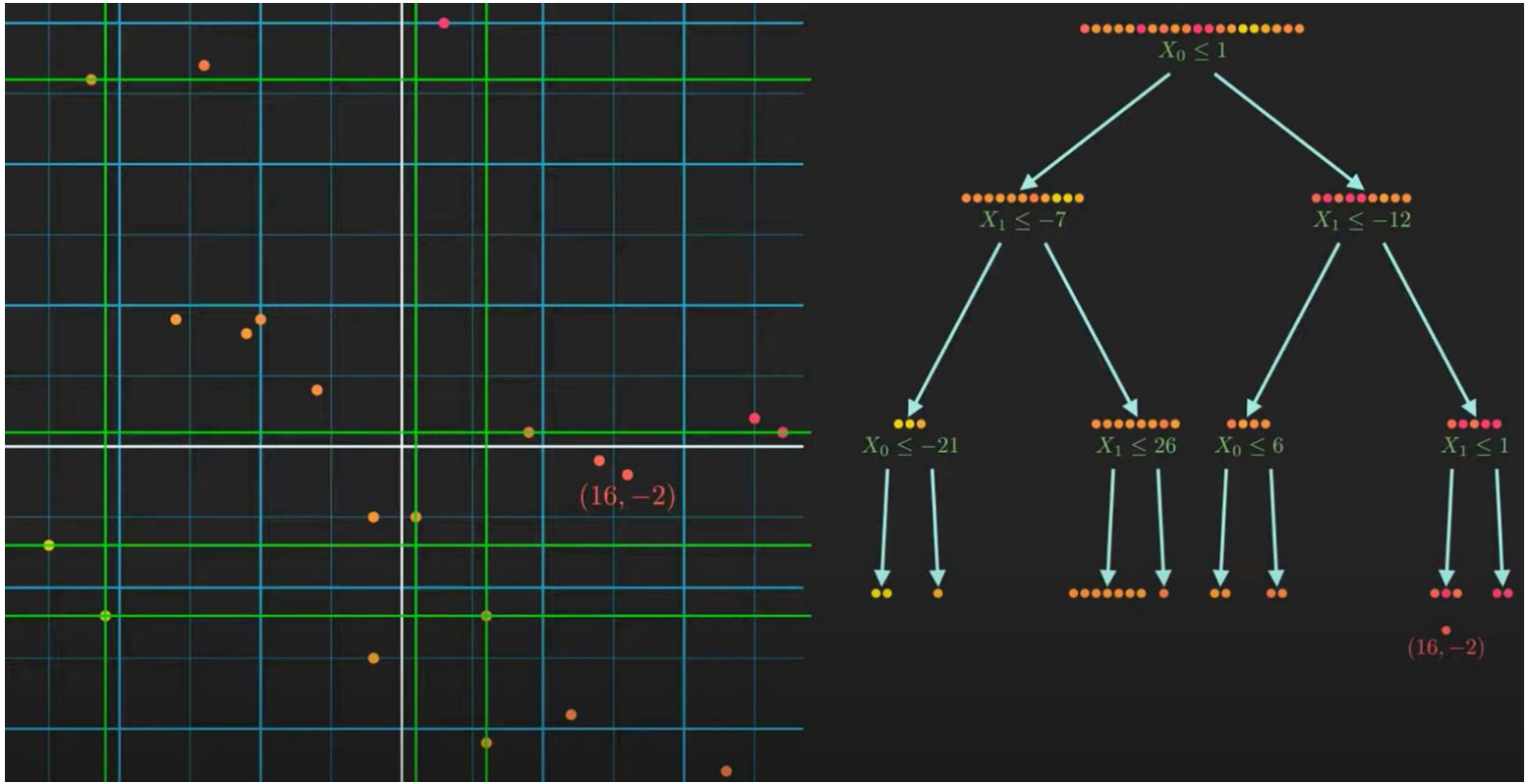
1. In decision tree algorithms, we could often choose between Gini impurity and entropy as the splitting criteria. While some decision tree algorithms use entropy, others link CART.

Continuous Variable

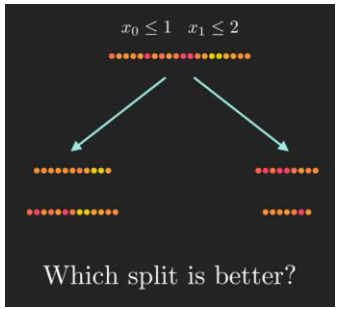


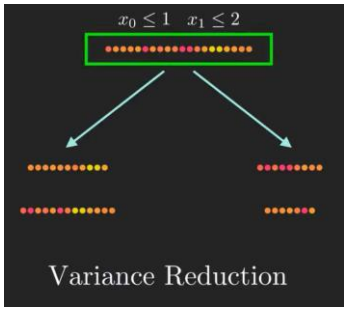
Density	Sugar
0.697	0.460
0.774	0.376
0.634	0.264
0.608	0.318
0.556	0.215
0.403	0.237
0.481	0.149
0.437	0.211
0.666	0.091
0.243	0.267
0.245	0.057
0.343	0.099
0.639	0.161
0.657	0.198
0.360	0.370
0.593	0.042
0.719	0.103

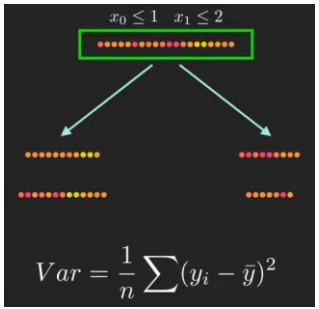
Regression Using Decision Tree

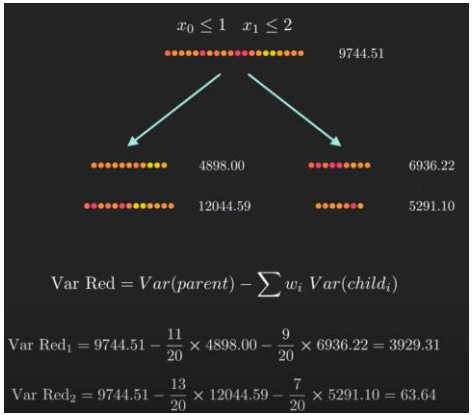


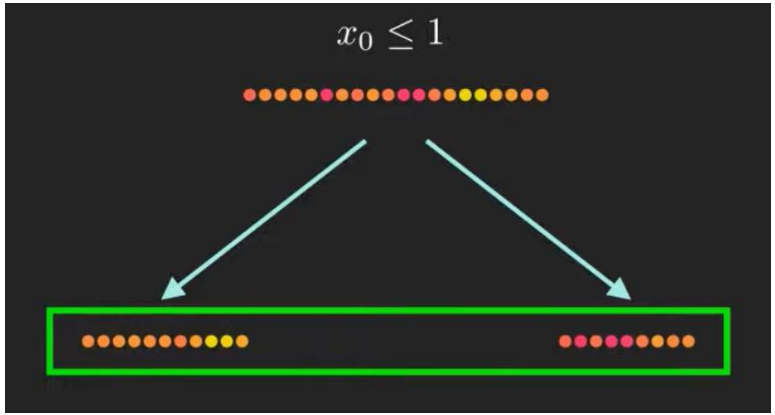
Regression Using Decision Tree









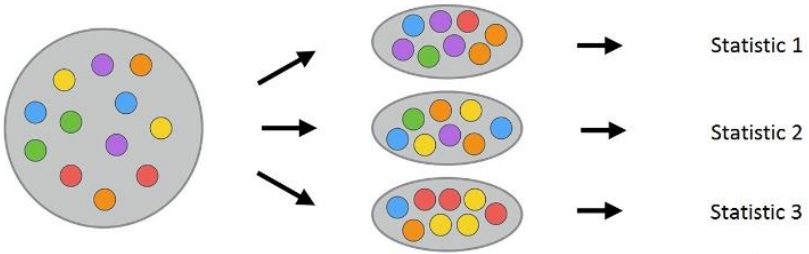


Weak vs. Strong Learners

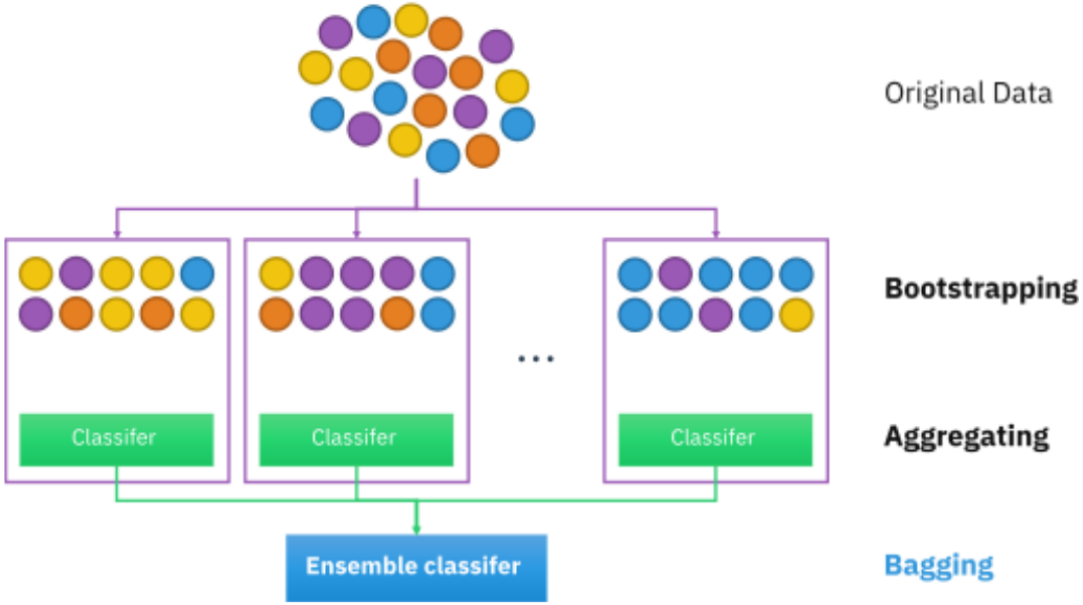
- A weak learner is a model that performs slightly better than random guessing.
  - E.g., in a binary classification problem, a weak learner would have an accuracy slightly better than 50%
- A strong learner is a model that is well-correlated with the true classification.
  - E.g., In classification, the model has a high accuracy rate

Bootstrap

- A resampling technique: creating multiple random subsamples from the original dataset by **sampling with replacement**
- Generates diverse training datasets



Bagging: Bootstrap aggregating



Bagging: Bootstrap aggregating

- Steps:
  1. Create multiple bootstrapped samples from the original training data
  2. Train a separate model on each bootstrapped samples
  3. Ensemble the outputs
     a. Regression: Average of all model predictions
     b. Classification: Majority vote
- Reduces variance and helps in avoiding overfitting

Random Forest

colorado school of public health

UNIVERSITY OF COLORADO
COLORADO STATE UNIVERSITY
UNIVERSITY OF NORTHERN COLORADO

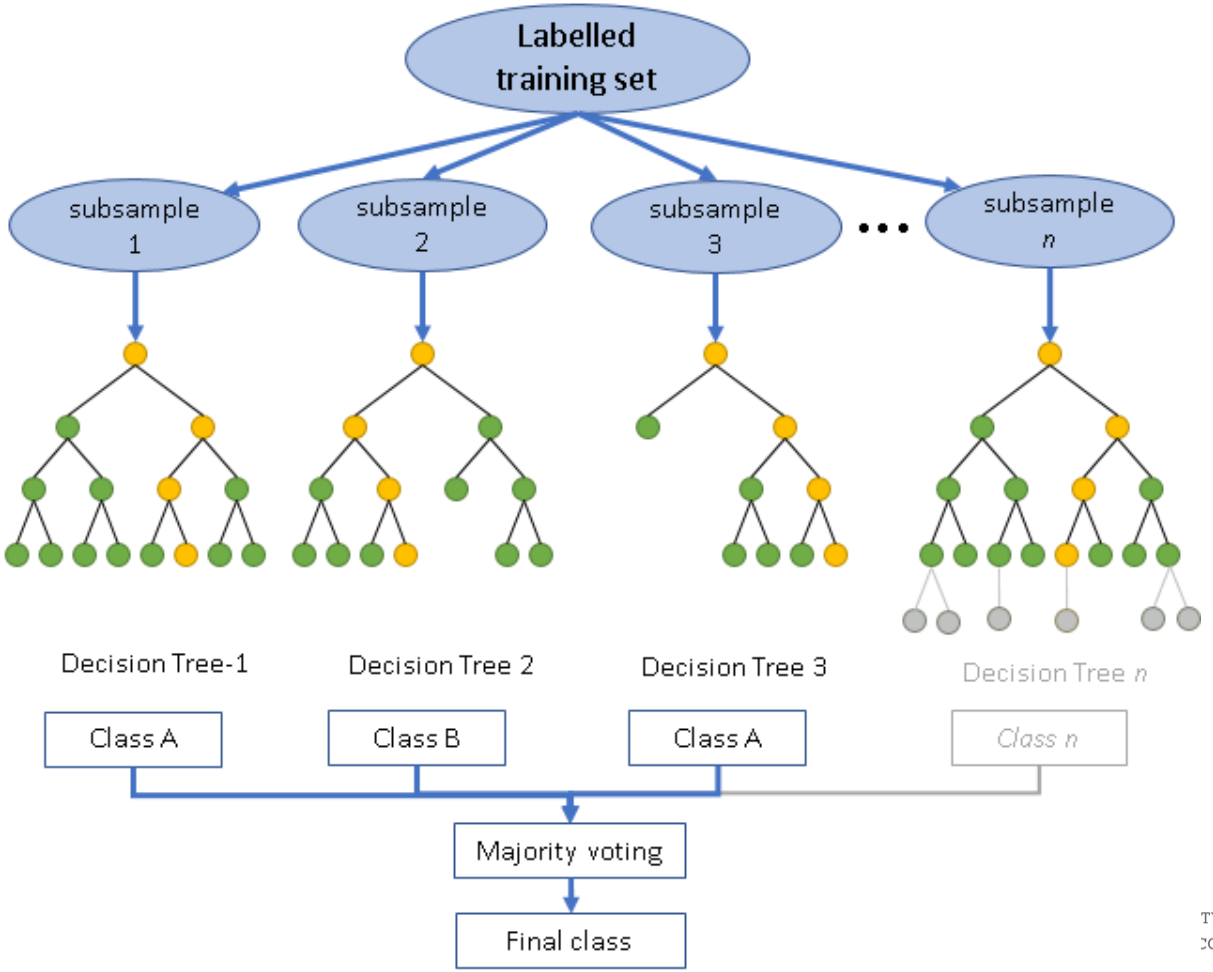
Random Forest

- An **ensemble** of Decision Trees
- Aggregates the individual outcomes of different decision trees trained on **the bootstrapped samples and randomly selected features** (Bagging)

Bootstrap

Building the trees on a random set of features

Ensemble of trees (aggregation)



Algorithm procedures

Step 1: For $b = 1$ to B:

a. Draw a **bootstrapped sample** of size N from the training data
b. Grow a random forest tree T_b to the bootstrapped sample, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached
    i. Select m variables at **random** from the p variables
    ii. Pick the best variable/split-point among the m
    iii. Split the node into two daughter nodes

Step 2: Output the ensemble of trees $\{T_b\}$ from 1 to B

Hyperparameters

- Number of decision trees, B
- Number of samples, N
- Number of features, m

Numbers of decision trees and samples

- Number of decision tree, B: Increasing the number of trees can improve the model's performance up to a point, but it also increases computational complexity
- Number of sample, N: A smaller value might increase the variance of the model since each tree will be trained on less data. On the other hand, a larger value might reduce variance but can increase bias if the samples are not diverse enough.

Determine the number of features

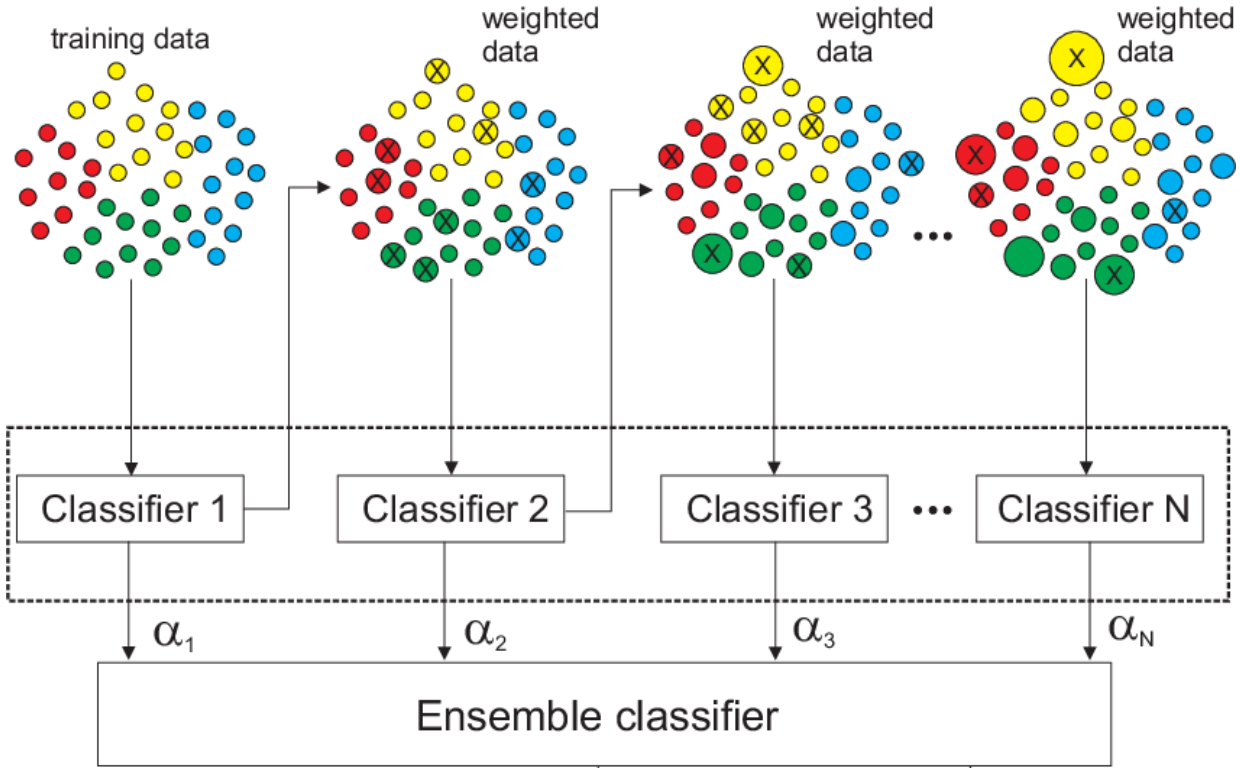
- The number of features selected at each split is m. And the total number of features is p.
  - For classification: $m = \sqrt{p}$
  - For regression: $m = p/3$
- Smaller values of m can improve generalization and reduce overfitting

Approaches for data aggregation

- Classification: predicts a class label
  - Selects the class that received the majority of votes
- Regression: predicts a continuous value
  - Average of these predicted values
- AdaBoost
  - Aka adaptive boosting = weighted addition

AdaBoost

- Each tree is not independent because each new tree is fit sequentially on modified version of the training dataset based on results from the previous tree
- The datapoints associated with the largest residuals (error/misclassification) are weighted the most in the new training set.


training data
weighted data
weighted data
weighted data
Classifier 1
Classifier 2
Classifier 3
Classifier N
α_1
α_2
α_3
α_N
Ensemble classifier

AdaBoost procedures

1. Initialize the observation weights $w_i = 1/N, i = 1, 2, \ldots, N$.
2. For $m = 1$ to M (each iteration):
   a) Fit a classifier $G_m(x)$ to the training data using weights w_i.
   b) Compute $err_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}$.
   c) Compute $\alpha_m = \log((1 - err_m)/err_m)$.
   d) Set $w_i \leftarrow w_i \cdot exp[\alpha_m \cdot I(y_i \neq G_m(x_i))], i = 1, 2, \ldots, N$
3. Output $G(x) = \text{sign}[\sum_{m=1}^{M} \alpha_m G_m(x)]$.

Limitations

- Challenging in interpretation
- Subjectivity in the decision of hyperparameters B, N and m → hyperparameters tunning

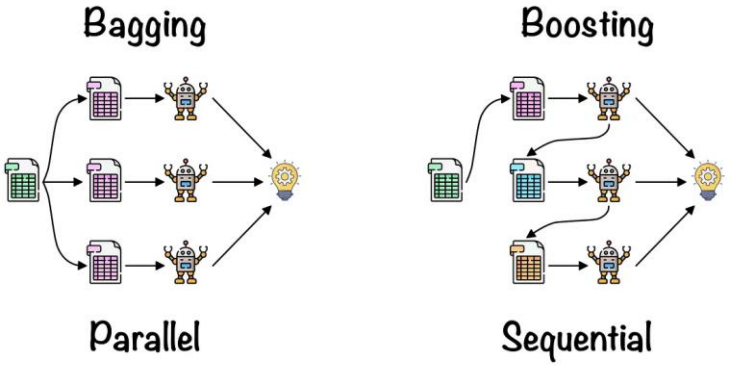
Gradient Boosted Trees

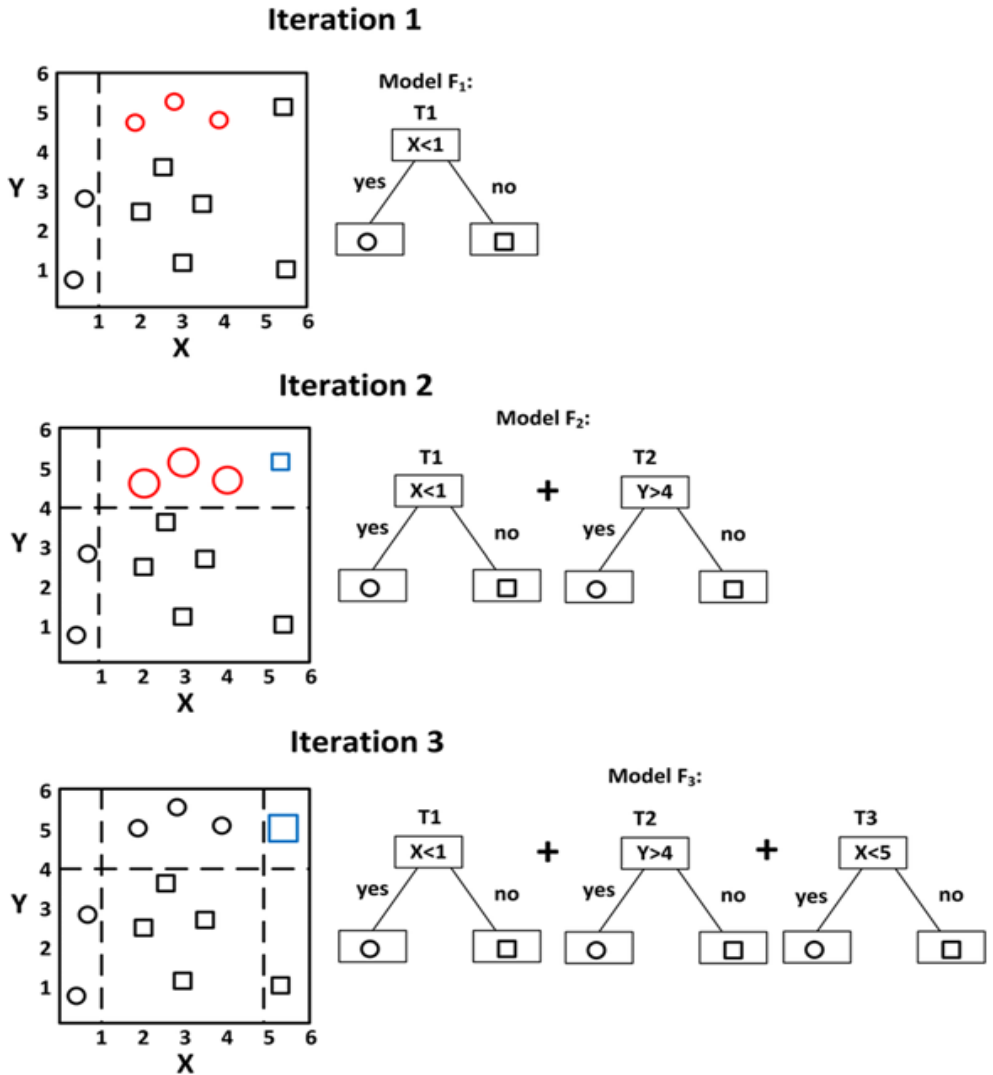
colorado school of public health
UNIVERSITY OF COLORADO
COLORADO STATE UNIVERSITY
UNIVERSITY OF NORTHERN COLORADO

Like random forests, gradient boosted trees are an ensemble method

Combines the output of individual decision trees with boosting

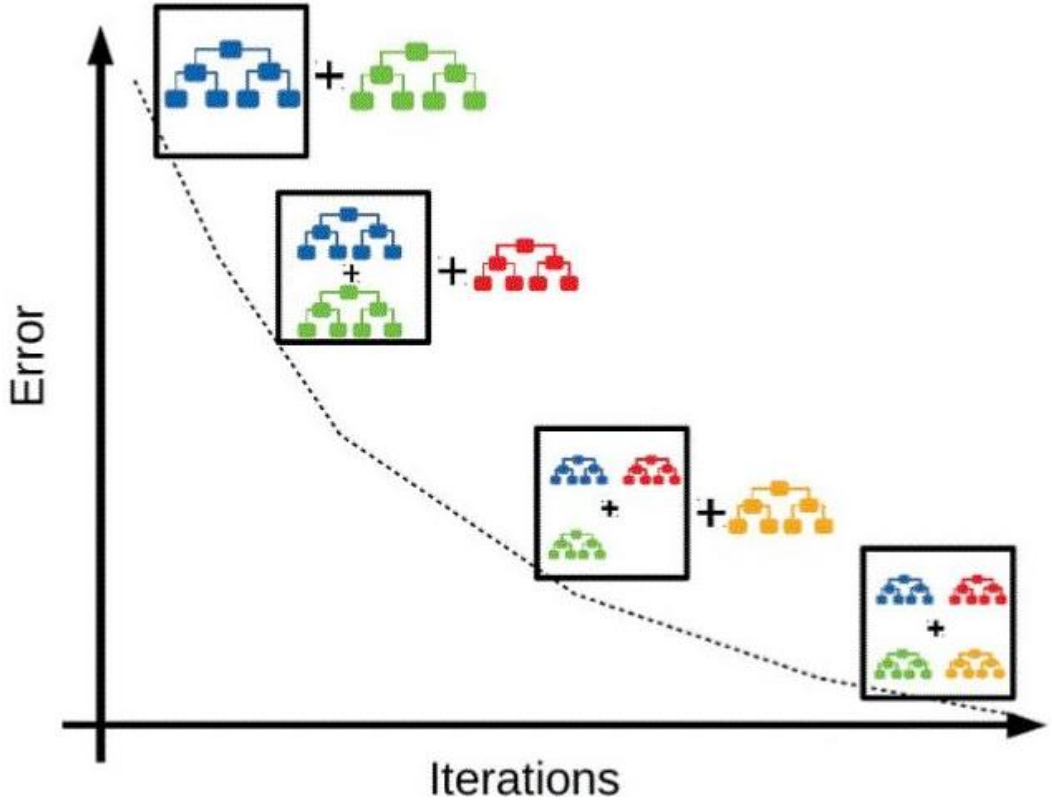
Boosting is a sequential process





Boosting combines trees so that each tree corrects the error of the previous tree

Overall, the loss of the ensemble goes down as trees are added


Error
Iterations

Gradient Boosting

Step 1: initialize with a single decision tree and evaluating the loss

Step 2: Add a second tree such that the loss of the ensemble is lower than the loss of the first tree alone

Ensemble = First tree + η* Second tree

Where η is the learning rate

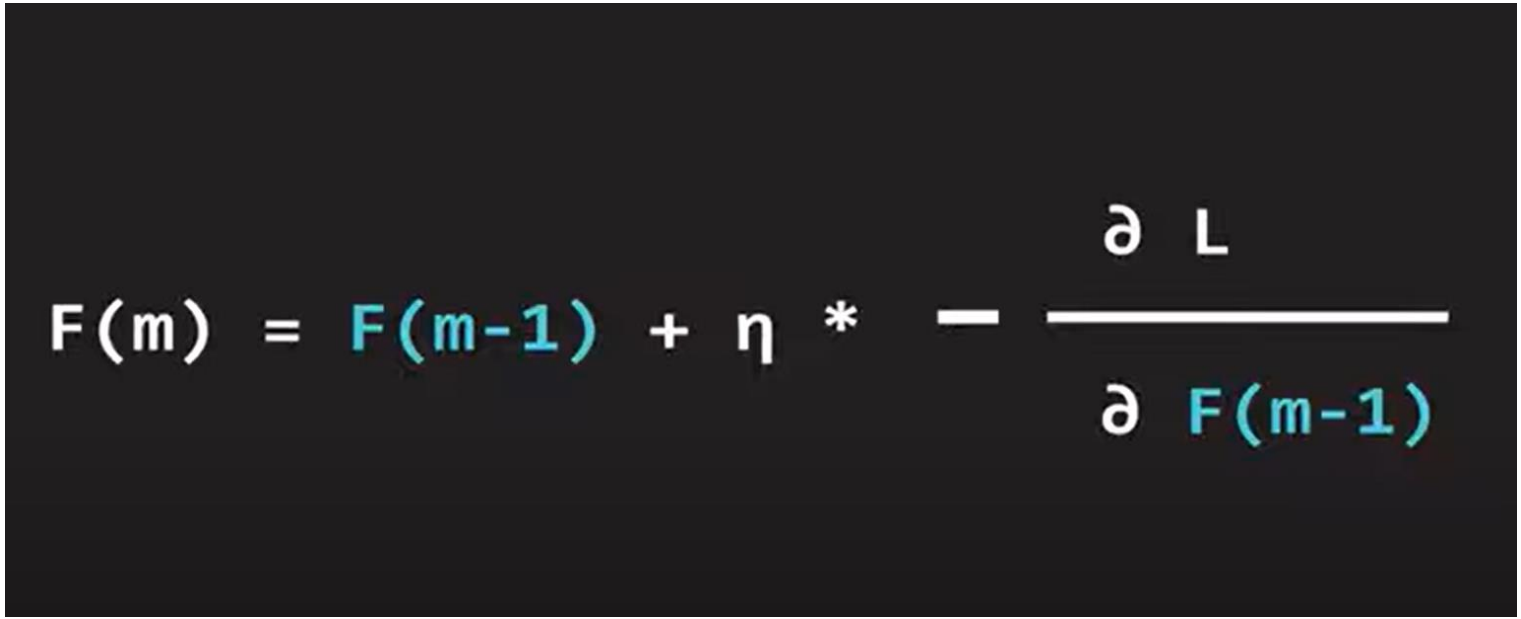
Loss(Ensemble) < Loss (First Tree)

We want to find the direction where the loss decreases the fastest

Therefore, we fit the second tree on the gradient of the loss function with respect to previous model's output

$$\text{Second Tree} = -\frac{\partial L}{\partial F(1)} = -\frac{\partial \text{ Loss Function}}{\partial \text{ Previous Model's Output}}$$

The ensemble at m is equal to the ensemble at m-1 plus the learning rate times the weak learner

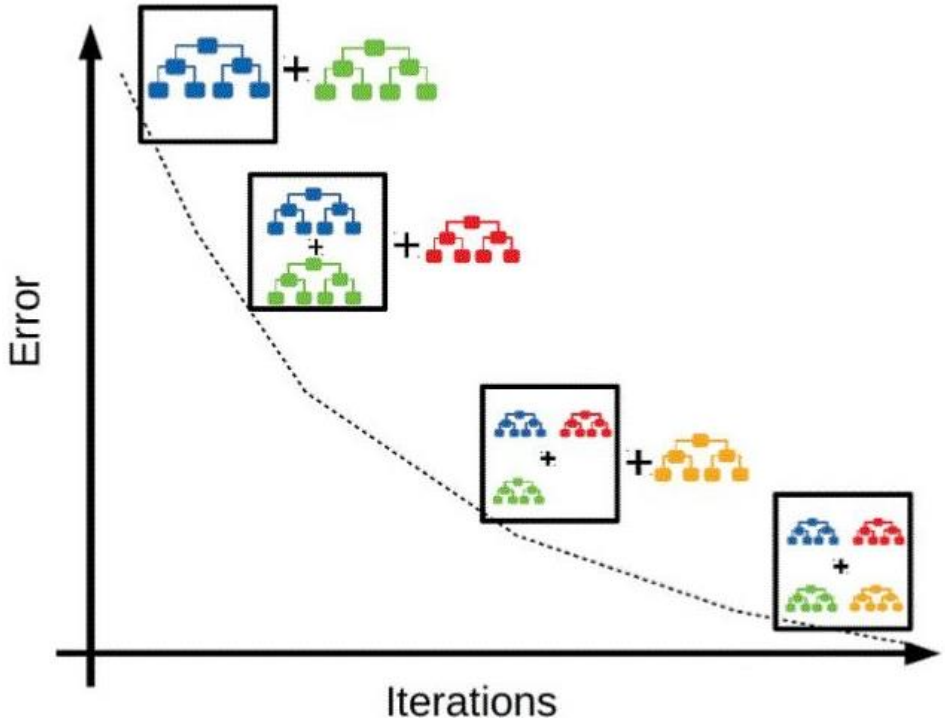


Setting	Loss Function	$-\partial L(y_i, f(x_i))/\partial f(x_i)$
Regression	$\frac{1}{2}[y_i - f(x_i)]^2$	$y_i - f(x_i)$
Regression	$\lvert y_i - f(x_i)\rvert$	$\text{sign}[y_i - f(x_i)]$
Regression	Huber	$y_i - f(x_i)$ for $\lvert y_i - f(x_i)\rvert \leq \delta_m$ $\delta_m \text{sign}[y_i - f(x_i)]$ for $\lvert y_i - f(x_i)\rvert > \delta_m$ where $\delta_m = \alpha$th-quantile$\{\lvert y_i - f(x_i)\rvert\}$
Classification	Deviance	kth component: $I(y_i = \mathcal{G}_k) - p_k(x_i)$

Newton’s Method

The minimum loss of an ensemble often does not have a closed form solution

We can use Newton’s methods which uses a first- order Taylor expansion around F(m-1) of the loss function to numerically approximate the minimum


Error
Iterations

We can think of each new addition as a weak learner

The final prediction is a combination of these increasingly specific weak learners

With each addition both the bias and the variance is decreasing

Hyperparameters

Gradient Boosted Trees have the same hyperparameters as random forests:

Number of boosting stages to perform

Number of Samples

Number of features

Additional Hyperparameters

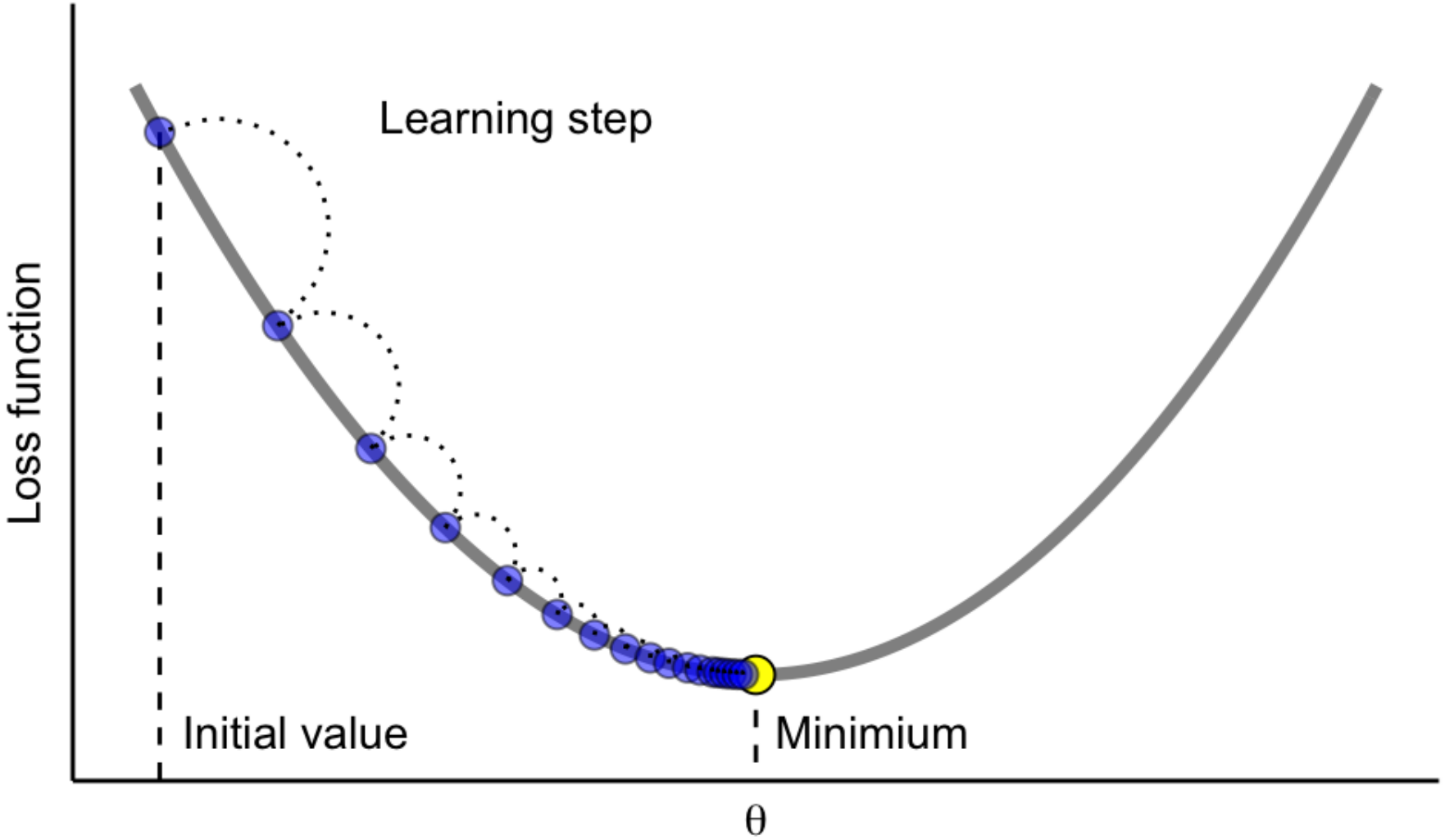
Learning Rate: shrinks the contribution of each tree

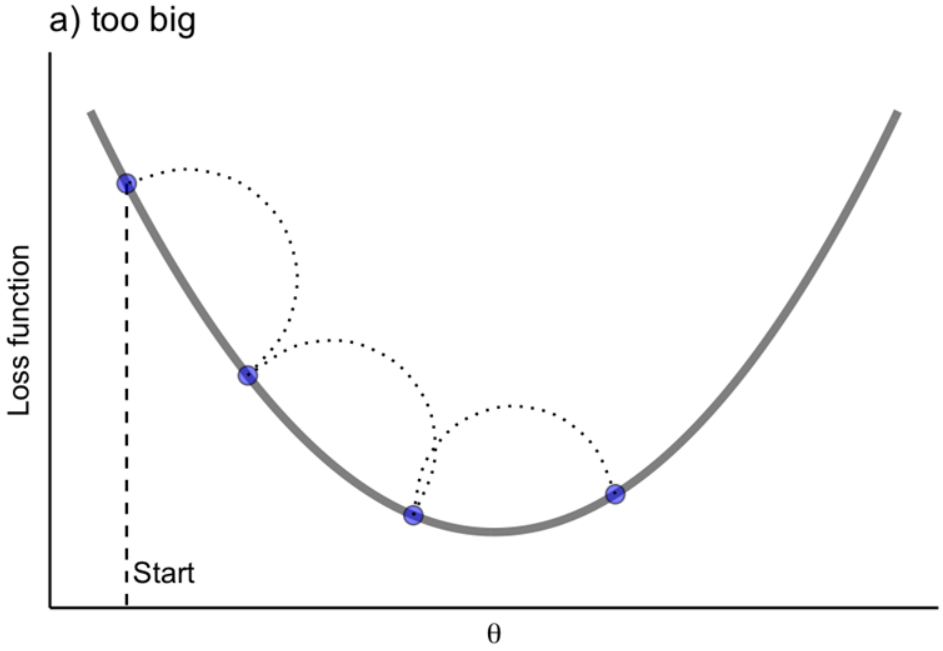
- Rule of thumb ~0.2

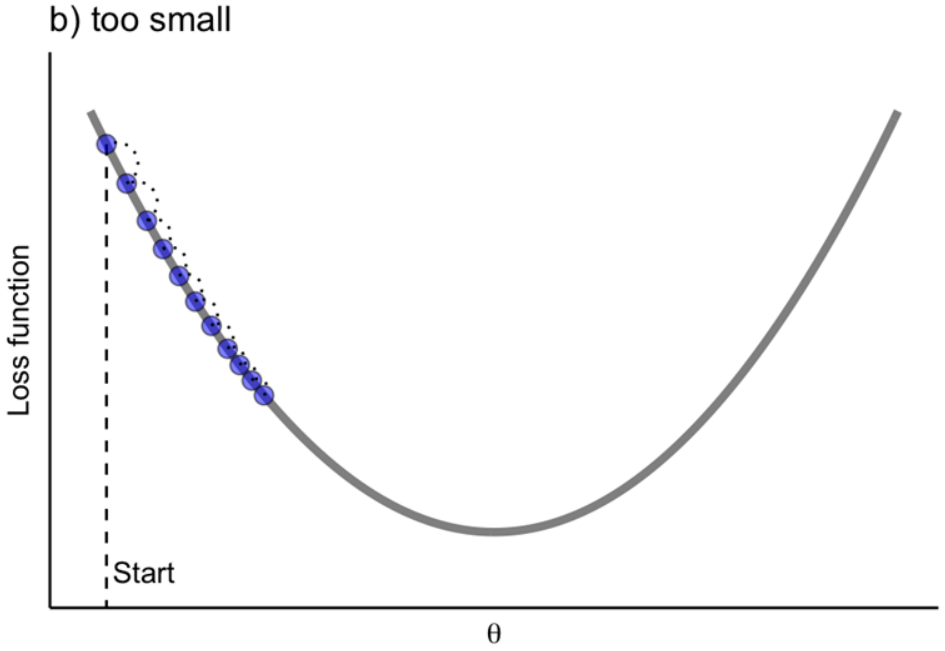
Subsample: Fraction of sample used to fit base learner

Max Features: Subsets the feature space

Using subsample and max feature parameters is a way to introduce randomness into our model which can improve speed (Stochastic Gradient Boosting)


Learning step
Loss function
Initial value
Minimium
θ


a) too big
Loss function
Start
θ


b) too small
Loss function
Start
θ

Gradient Boosted Trees: Medical Applications

Cardiovascular Events:
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6345960/

Development of sepsis: https://pubmed.ncbi.nlm.nih.gov/30661855/

Delirium Risk: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6324291/

Hospital readmissions after lower lumbar laminectomy:
https://pubmed.ncbi.nlm.nih.gov/30544346/

Predicting Hospital Readmission

Objective: Identify patients who are at risk for postoperative hospital readmission after lumbar laminectomy

Researchers used both a basic decision tree and a gradient boosted tree for prediction

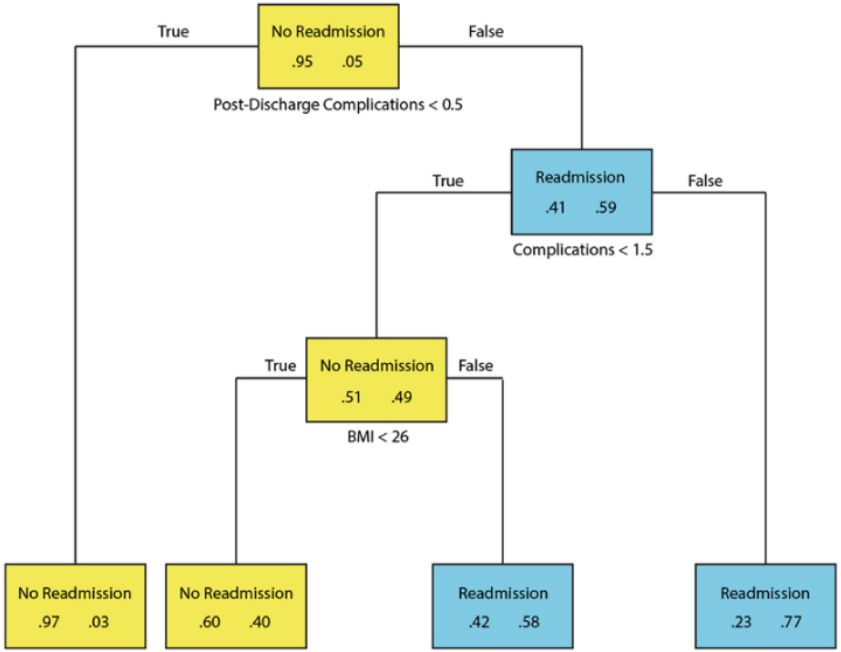


FIG. 1. Decision tree to predict readmission based on patient clinical and demographic factors. Figure is available in color online only.

59% of patients with post-discharge complications were re-admitted

Only 3% of patients without post-discharge complications were readmitted

In reality patients without post-discharge complications made up 59.2% of all patients readmitted

What is the decision tree missing?

Missed the impact of comorbidities on readmission

The gradient boosted model was able to account for comorbidities and had a accuracy of 95.33%

AUC 0.806

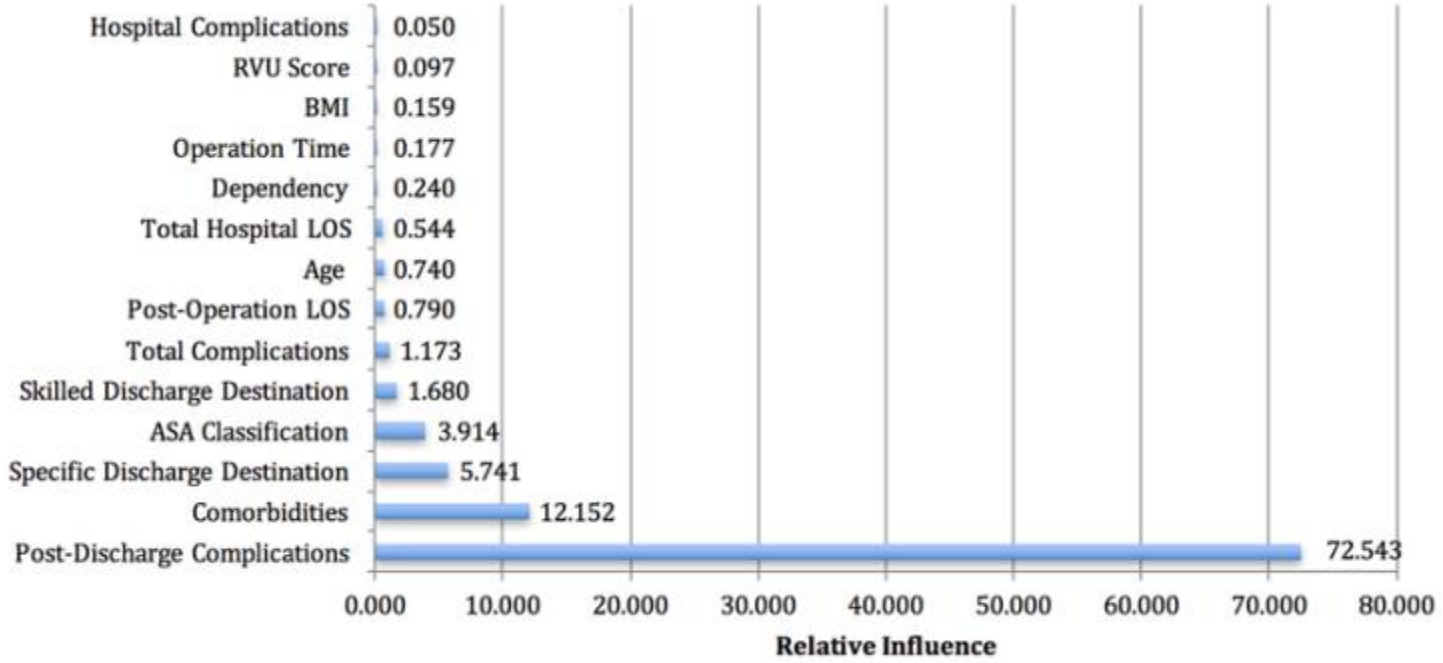


FIG. 2. Variable influence scores in full GBM readmission predictive model. Figure is available in color online only.

```
Package XGBOOST

model2 = XGBClassifier(objective='multiclass:softmax',
learning_rate = 0.1,
                max_depth = 1, n_estimators = 330)

Sklearn.ensemble.GradientBoostingClassifier()
```

Summary

Classification support

- K-Nearest Neighbors
  - Binary or multi-label
- Decision Tree
  - Binary or multi-label
- Random Forest
  - Binary or multi-label
- Gradient Boosting
  - Binary or multi-label

Algorithm complexity p.1

- K-Nearest Neighbors
  - There is no training phase but prediction time can be high when number of training samples and/or features is large
  - Performance is highly dependent of the number of neighbors (k) and distance metric used

- Decision Tree
  - Training time is dependent on the number of samples and number of features
  - Prediction time is proportional to depth of the tree

Algorithm complexity p.2

- Random Forest
  - Training time is dependent on the number of trees, samples, and features
  - Prediction time is dependent on the number of trees and depth of each tree

- Gradient Boosting
  - Training time is dependent on the number of trees, samples, and features
  - Prediction time is dependent to the number of trees and depth of each tree

Data normalization

- K-Nearest Neighbors
  - necessary → distance-based algorithm, sensitive to scale of input features
  - ensures that each feature contributes equally to distance metric used
  - features with larger scales can dominate distance metric and lead to lower performance
- Decision Tree
  - not necessary → not sensitive to scale of input features
- Random Forest
  - not necessary
- Gradient Boosting
  - not necessary

Sensitivity to changes in data structure p.1

- K-Nearest Neighbors
  - sensitive to changes since it relies on distance between data points to make predictions
  - if changed significantly, then distance metric used may not be appropriate → lower performance

- Decision Tree
  - sensitive to changes since it relies on recursive partitioning of feature space
  - if changed significantly, then optimal partitioning may also change → lower performance

Sensitivity to changes in data structure p.2

- Random Forest
  - less sensitive with ensemble of trees
  - if changed significantly, then some trees may be less accurate → lower performance
- Gradient Boosting
  - similar to random forest

Data imbalance p.1

- K-Nearest Neighbors
  - sensitive: relies on distance between data points to make predictions → imb. training data may bias towards majority
  - one way to address → weighted KNN

- Decision Tree
  - sensitive: can become biased towards majority class if tree is not pruned properly
  - one way to address → weighting samples during training to evaluate optimal partitioning

Data imbalance p.2

- Random Forest
  - less sensitive with ensemble of trees → imb. training data may bias towards majority class and reduce performance
  - one way to address → balanced random forests = use balanced random sampling at each tree.

- Gradient Boosting
  - Similar to random forest

Outliers p.1

- K-Nearest Neighbors
  - sensitive: instance-based algorithm using distance criteria, and could create a biased outcome

- Decision Tree
  - less sensitive: Sensitivity is dependent on the percentage of outliers and the depth of the tree (deeper trees are more likely to be sensitive to outliers).

Outliers p.2

- Random Forest
  - less sensitive: weak trees (not deep, less sensitive to outliers) are aggregated

- Gradient Boosting
  - Each tree is relatively weak/shallow so it is less sensitive than a normal decision tree.
  - But outliers have larger residuals than non-outliers → focus attention on those points

Overfitting p.1

- K-Nearest Neighbors
  - Since there is no model, it cannot overfit.
  - Performance is highly dependet on the value of k.

- Decision Tree
  - can overfit since it can create complex models that fit training data too closely
  - one way to prevent overfitting is to use pruning, which involves removing branches from tree that do not improve its performance on the validation set

Overfitting p.2

- Random Forest
  - less likely to overfit than decision trees with ensemble of trees
  - each tree in ensemble is trained on a different subset of data, which reduces chance of overfitting and improves generalization (at the expense of explainability)

- Gradient Boosting
  - can overfit if number of trees in ensemble is too large or if learning rate is too high since model would fit training data too closely and perform poorly on new, unseen data
  - one way to prevent overfitting in gradient boosting is to use regularization techniques such as shrinkage (lower learning rate) or early stopping

Algorithm	Function in scikit-learn	Hyperparameters	Binary/Multi-Label Classification	Algorithm Complexity	Normalization	Changes in Data Structure	Data Imbalance	Outlier Sensitivity	Overfitting	Notes
K-Nearest Neighbors	KNeighborsClassifier	n_neighbors, weights, algorithm, leaf_size, p, metric, etc.	Binary and multi-label supported.	- training time low, prediction time high with high number of features - sensitive to k, distance metric	?	Sensitive	Sensitive	Sensitive	Sensitive, but depends on k	- Requires distance metric to calculate similarity between instances - Sensitive to irrelevant features and scale of data - Computationally expensive for large datasets
Decision Tree	DecisionTreeClassifier	criterion, splitter, max_depth, min_samples_split, min_samples_leaf, etc.	Binary and multi-label supported.	- training time proportional to number of samples, features - prediction time proportional to depth of tree	Not necessary	Sensitive	Sensitive	Not sensitive	??	- Prone to overfitting, especially when tree is deep - Sensitive to small changes in data, which can lead to instability
Random Forest	RandomForestClassifier	n_estimators, criterion, max_depth, min_samples_split, min_samples_leaf, etc.	Binary and multi-label supported.	- training time proportional to number of trees, samples, features - prediction time proportional to number, depth of trees	Not necessary	Less sensitive	Less sensitive	???	Less sensitive	- Can be computationally expensive for large datasets and complex models - Can use feature bagging for high-dimensional data or data with strong correlations between features
Gradient		loss, learning_rate,	Binary and multi-	- training time proportional to number of trees, samples, features					Sensitive, but depends on	- Can be computationally expensive for large datasets and complex

Algorithm	Function in scikit-learn	Hyperparameters	Binary/Multi-Label Classification	Algorithm Complexity	Normalization	Changes in Data Structure	Data Imbalance	Outlier Sensitivity	Overfitting	Notes
K-Nearest Neighbors	KNeighborsClassifier	n_neighbors, weights, algorithm, leaf_size, p, metric, etc.	Binary and multi-label supported.	- training time low, prediction time high with high number of features - sensitive to k, distance metric	Necessary	Sensitive	Sensitive	Sensitive	Sensitive, but depends on k	- Requires distance metric to calculate similarity between instances - Sensitive to irrelevant features and scale of data - Computationally expensive for large datasets
Decision Tree	DecisionTreeClassifier	criterion, splitter, max_depth, min_samples_split, min_samples_leaf, etc.	Binary and multi-label supported.	- training time proportional to number of samples, features - prediction time proportional to depth of tree	Not necessary	Sensitive	Sensitive	Not sensitive	??	- Prone to overfitting, especially when tree is deep - Sensitive to small changes in data, which can lead to instability
Random Forest	RandomForestClassifier	n_estimators, criterion, max_depth, min_samples_split, min_samples_leaf, etc.	Binary and multi-label supported.	- training time proportional to number of trees, samples, features - prediction time proportional to number, depth of trees	Not necessary	Less sensitive	Less sensitive	???	Less sensitive	- Can be computationally expensive for large datasets and complex models - Can use feature bagging for high-dimensional data or data with strong correlations between features
Gradient		loss, learning_rate,	Binary and multi-	- training time proportional to number of trees, samples, features					Sensitive, but depends on	- Can be computationally expensive for large datasets and complex

Algorithm	Function in scikit-learn	Hyperparameters	Binary/Multi-Label Classification	Algorithm Complexity	Normalization	Changes in Data Structure	Data Imbalance	Outlier Sensitivity	Overfitting	Notes
K-Nearest Neighbors	KNeighborsClassifier	n_neighbors, weights, algorithm, leaf_size, p, metric, etc.	Binary and multi-label supported.	- training time low, prediction time high with high number of features - sensitive to k, distance metric	Necessary	Sensitive	Sensitive	Sensitive	Sensitive, but depends on k	- Requires distance metric to calculate similarity between instances - Sensitive to irrelevant features and scale of data - Computationally expensive for large datasets
Decision Tree	DecisionTreeClassifier	criterion, splitter, max_depth, min_samples_split, min_samples_leaf, etc.	Binary and multi-label supported.	- training time proportional to number of samples, features - prediction time proportional to depth of tree	Not necessary	Sensitive	Sensitive	Not sensitive	Sensitive	- Prone to overfitting, especially when tree is deep - Sensitive to small changes in data, which can lead to instability
Random Forest	RandomForestClassifier	n_estimators, criterion, max_depth, min_samples_split, min_samples_leaf, etc.	Binary and multi-label supported.	- training time proportional to number of trees, samples, features - prediction time proportional to number, depth of trees	Not necessary	Less sensitive	Less sensitive	???	Less sensitive	- Can be computationally expensive for large datasets and complex models - Can use feature bagging for high-dimensional data or data with strong correlations between features
Gradient	GradientBoostingClassifier	loss, learning_rate,	Binary and multi-	- training time proportional to number of trees, samples, features	Not necessary	Less sensitive	????	Sensitive	Sensitive, but depends on	- Can be computationally expensive for large datasets and complex

Algorithm	Function in scikit-learn	Hyperparameters	Binary/Multi-Label Classification	Algorithm Complexity	Normalization	Changes in Data Structure	Data Imbalance	Outlier Sensitivity	Overfitting	Notes
K-Nearest Neighbors	KNeighborsClassifier	n_neighbors, weights, algorithm, leaf_size, p, metric, etc.	Binary and multi-label supported.	- training time low, prediction time high with high number of features - sensitive to k, distance metric	Necessary	Sensitive	Sensitive	Sensitive	Sensitive, but depends on k	- Requires distance metric to calculate similarity between instances - Sensitive to irrelevant features and scale of data - Computationally expensive for large datasets
Decision Tree	DecisionTreeClassifier	criterion, splitter, max_depth, min_samples_split, min_samples_leaf, etc.	Binary and multi-label supported.	- training time proportional to number of samples, features - prediction time proportional to depth of tree	Not necessary	Sensitive	Sensitive	Not sensitive	Sensitive	- Prone to overfitting, especially when tree is deep - Sensitive to small changes in data, which can lead to instability
Random Forest	RandomForestClassifier	n_estimators, criterion, max_depth, min_samples_split, min_samples_leaf, etc.	Binary and multi-label supported.	- training time proportional to number of trees, samples, features - prediction time proportional to number, depth of trees	Not necessary	Less sensitive	Less sensitive	Not sensitive	Less sensitive	- Can be computationally expensive for large datasets and complex models - Can use feature bagging for high-dimensional data or data with strong correlations between features
Gradient	GradientBoostingClassifier	loss, learning_rate,	Binary and multi-	- training time proportional to number of trees, samples, features	Not necessary	Less sensitive	????	Sensitive	Sensitive, but depends on	- Can be computationally expensive for large datasets and complex

Algorithm	Function in scikit-learn	Hyperparameters	Binary/Multi-Label Classification	Algorithm Complexity	Normalization	Changes in Data Structure	Data Imbalance	Outlier Sensitivity	Overfitting	Notes
K-Nearest Neighbors	KNeighborsClassifier	n_neighbors, weights, algorithm, leaf_size, p, metric, etc.	Binary and multi-label supported.	- training time low, prediction time high with high number of features - sensitive to k, distance metric	Necessary	Sensitive	Sensitive	Sensitive	Sensitive, but depends on k	- Requires distance metric to calculate similarity between instances - Sensitive to irrelevant features and scale of data - Computationally expensive for large datasets
Decision Tree	DecisionTreeClassifier	criterion, splitter, max_depth, min_samples_split, min_samples_leaf, etc.	Binary and multi-label supported.	- training time proportional to number of samples, features - prediction time proportional to depth of tree	Not necessary	Sensitive	Sensitive	Not sensitive	Sensitive	- Prone to overfitting, especially when tree is deep - Sensitive to small changes in data, which can lead to instability
Random Forest	RandomForestClassifier	n_estimators, criterion, max_depth, min_samples_split, min_samples_leaf, etc.	Binary and multi-label supported.	- training time proportional to number of trees, samples, features - prediction time proportional to number, depth of trees	Not necessary	Less sensitive	Less sensitive	Not sensitive	Less sensitive	- Can be computationally expensive for large datasets and complex models - Can use feature bagging for high-dimensional data or data with strong correlations between features
Gradient	GradientBoostingClassifier	loss, learning_rate,	Binary and multi-	- training time proportional to number of trees, samples, features	Not necessary	Less sensitive	Sensitive	Sensitive	Sensitive, but depends on	- Can be computationally expensive for large datasets and complex

Thank you!

Questions?