Introduction to Causal Inference

CS109A Introduction to Data Science

Pavlos Protopapas, Kevin Rader and Chris Gumb

Lecture is **heavily** motivated by the work of Brady Neal

Photo: Latusek, Walter

Logistic Regression

Logistic Regression I

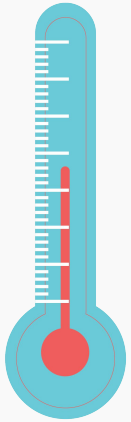
- What is Classification?
- Log Odds and Logistic Model
- Loss Function and Optimization
- Understanding the Coefficients
- Multi-, Poly-, and Regularized Logistic Regression
- Classification Decision Boundaries

What is Classification?

In summary,

Regression

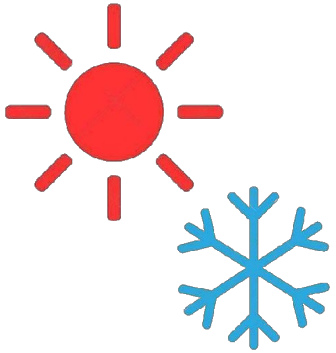
Performs well on tasks that require the prediction of a quantitative response variable.



What is the temperature going to be tomorrow?

Classification

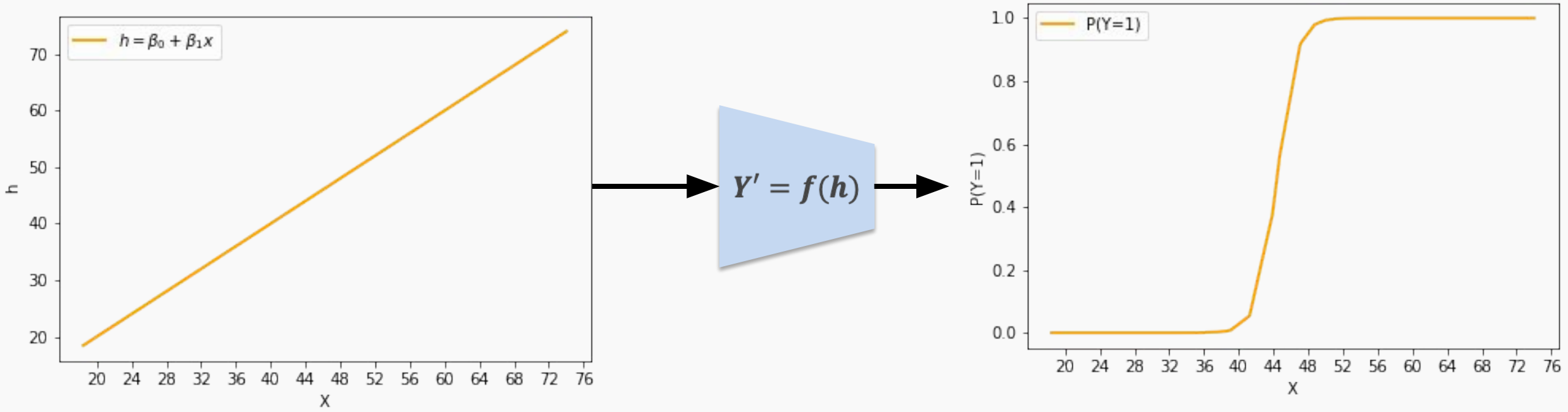
Performs well on tasks that require the prediction of a categorical or qualitative response variable. It classifies an observation into a category or class labeled by Y.



Is it going to be hot or cold tomorrow?

What function should we use?

Now we know that linear regression yields values for probability that are larger than 1 or smaller than 0. So what can we do to fix this?



What function should we use?

We can use the sigmoid function:

$$h = \beta_0 + \beta_1 X \longrightarrow \mathrm{p} = \frac{1}{1 + e^{-h}} \longrightarrow \mathrm{P}(Y = 1) = \frac{1}{1 + e^{-\beta_0 - \beta_1 X}}$$

Logistic Model through Log Odds

With a little bit of algebraic work, the logistic model can be rewritten as:

$$\ln\left(\underbrace{\frac{P(Y=1)}{1-P(Y=1)}}_{odds}\right) = \beta_0 + \beta_1 X$$

Or the other way around!

Start with log-odds, and with some algebra, we end up with the logistic function!

$$\mathrm{P}(Y=1) = \frac{1}{1+e^{-\beta_0-\beta_1 X}}$$

Estimating the Simple Logistic Model

The likelihood of a single observation for *p* given *x and true label y* is:

$$L(p_i|Y_i) = P(Y_i = y_i) = p_i^{y_i}(1-p_i)^{1-y_i}$$

Assuming the observations are independent, the total probability (or likelihood $L(p_i|Y_i)$) of getting a certain outcome will be the product of all individual likelihoods

$$L(p|Y) = \prod_i P(Y_i = y_i) = \prod_i p_i^{y_i}(1-p_i)^{1-y_i}$$

Estimating the Simple Logistic Model

Maximizing a function is equivalent to minimizing the –ve of the function

$$\underset{\beta}{\text{argmax}}\, L(p|y) = \underset{\beta}{\text{argmax}} \log L(p|y) = \underset{\beta}{\text{argmin}}\, (-\log L(p|y)) = \underset{\beta}{\text{argmin}}\, (-l(p|Y))$$

Using the formula for the log-likelihood, we get

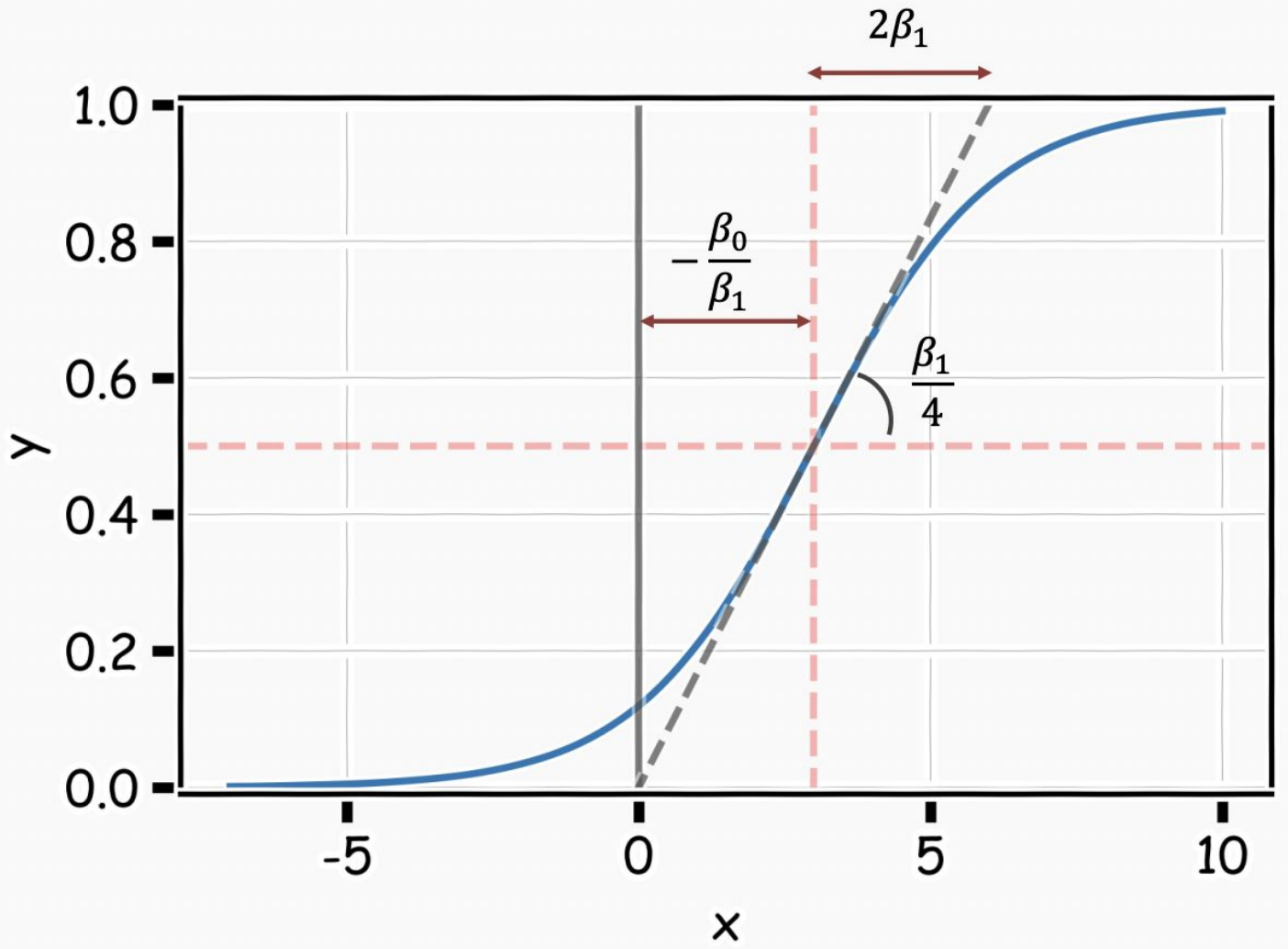
$$\beta^* = \underset{\beta}{\text{argmin}}\, (-l(p|Y)\ ) = -\sum_i y_i \log p_i\ + (1-y_i)\log(1-p_i)$$

The final loss, the Binary Cross Entropy is then:

$$L_{BCE} = -\sum_i \{y_i \log p_i + (1-y_i)\log(1-p_i)\}$$

β's Effect on Sigmoid Curve

The coefficients β_0 and β_1 now control the shape of this s-shape curve.



Interpretation of β's

Logistic regression is said to model the ***log-odds*** with a linear function of the predictors or features, X.

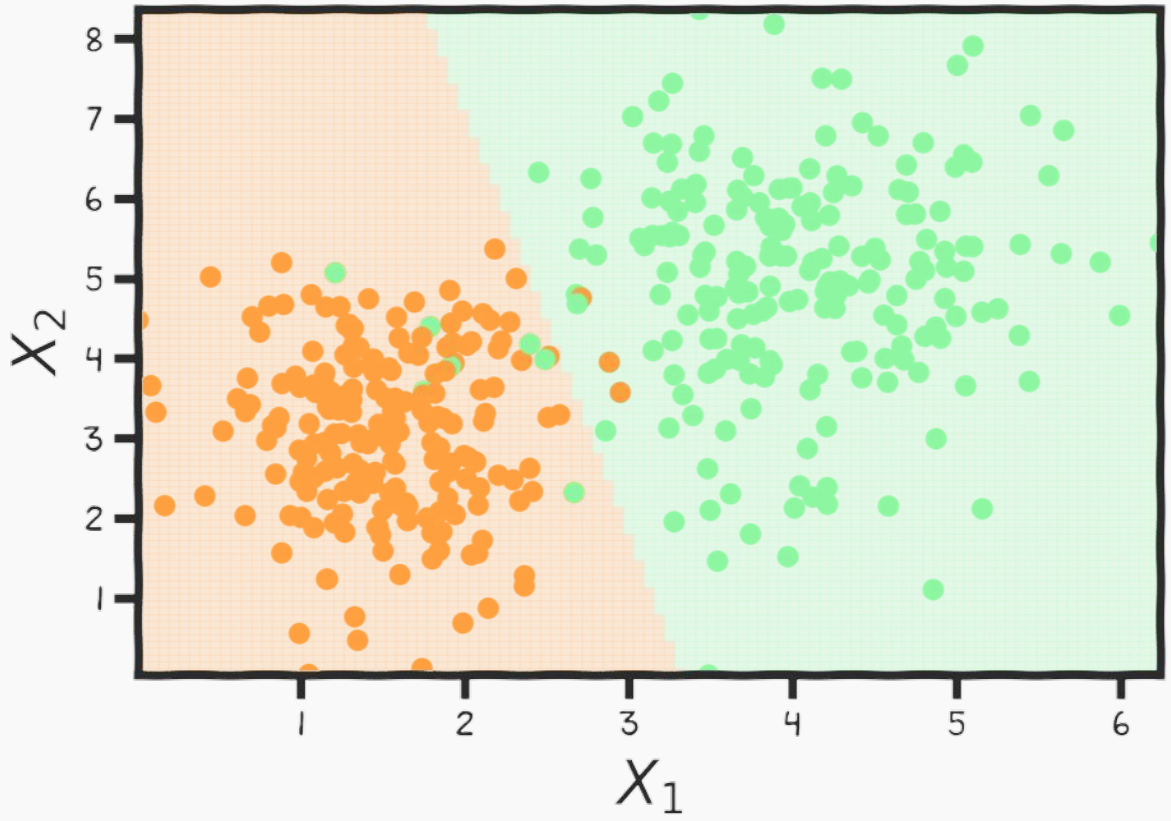
A one unit change in X is associated with a β_1 change in the log-odds of $\mathrm{P}(Y = 1)$; or better yet, a one unit change in X is associated with an e^{β_1} change in the odds that $Y = 1$.

$$\ln\left(\underbrace{\frac{P(Y=1)}{1-P(Y=1)}}_{\text{odds}}\right) = \beta_0 + \beta_1 X$$

Classifier with two predictors

How can we estimate a classifier, based on logistic regression, for the following plot?

The challenge here is that we have



Multiple logistic regression is a generalization to multiple predictors:

$$\log\left(\frac{P(Y=1)}{1-P(Y=1)}\right) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + .. + \beta_p X_p$$

Polynomial Logistic Regression

We saw a 2-D plot last time which had two predictors, X_1, X_2. A similar one is shown here but the decision boundary is not linear.

We can extend multiple Logistic Regression as we did with polynomial regression:

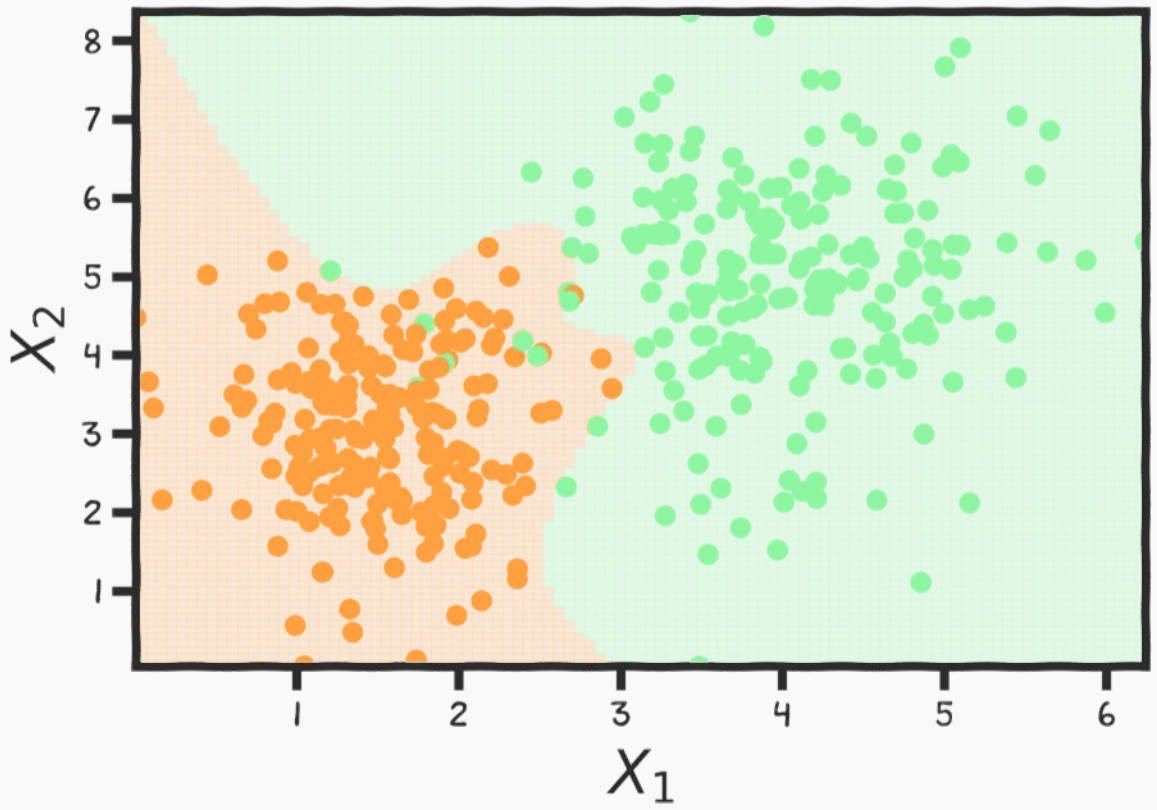
We transform the data by adding new predictors:

$$\tilde{x} = [1, \tilde{x}_1, \tilde{x}_2, \dots, \tilde{x}_M]$$

where $\tilde{x}_k = x^k$

The polynomial Logistic Regression can be expressed as:

$$\log\left(\frac{P(Y=1)}{1-P(Y=1)}\right) = \tilde{X}\beta$$

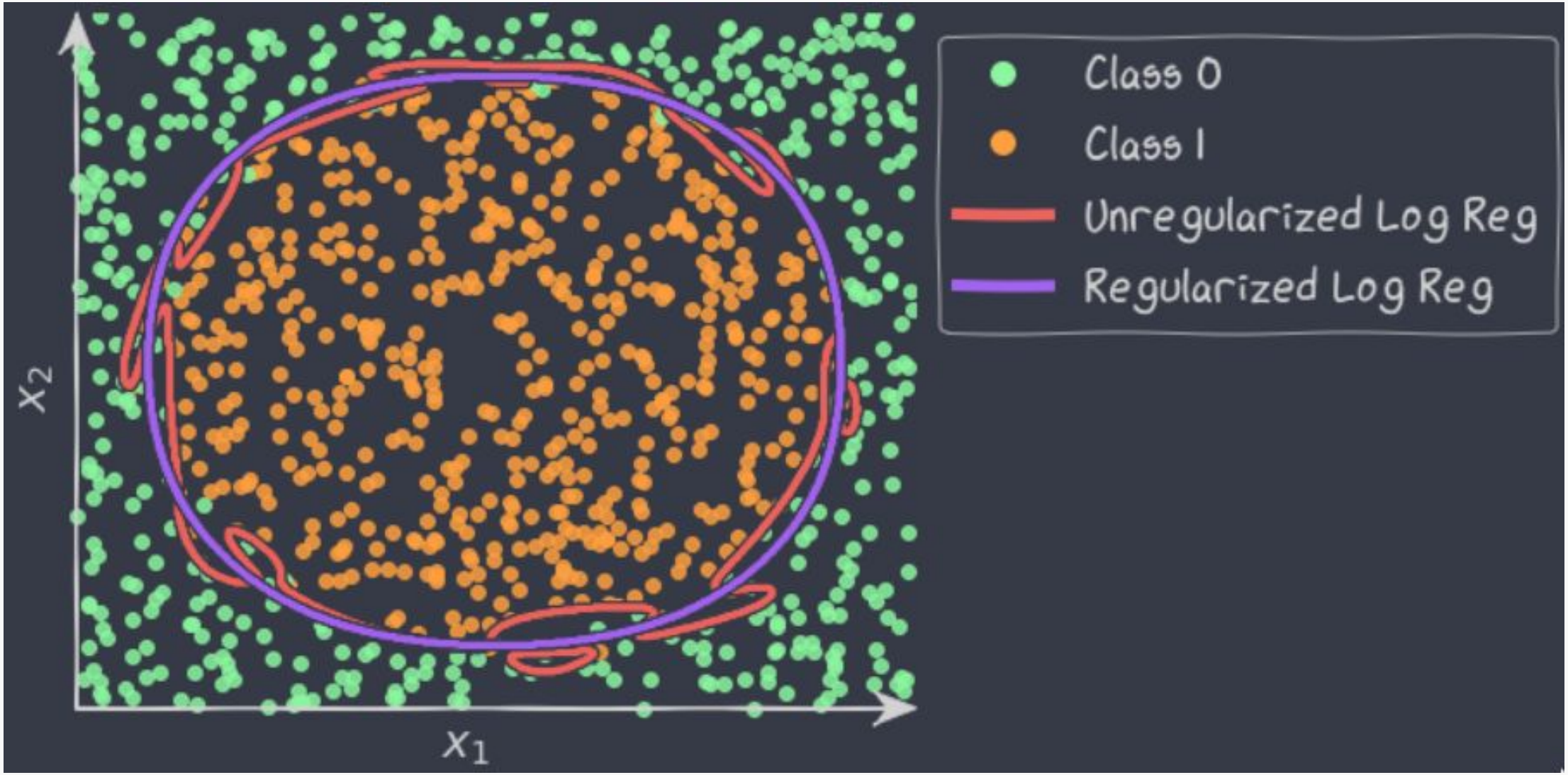


Regularization in Logistic Regression

A penalty factor can then be added to this loss function and results in a new loss function that penalizes large values of

$$\underset{\beta}{\operatorname{argmin}} \left[-\sum \; y_i \log p_i + (1 - y_i) \log(1 - p_i) \; + \lambda \sum_{j=1}^{J} \beta_j^2 \right]$$

Regularized Decision Boundaries



Logistic Regression II

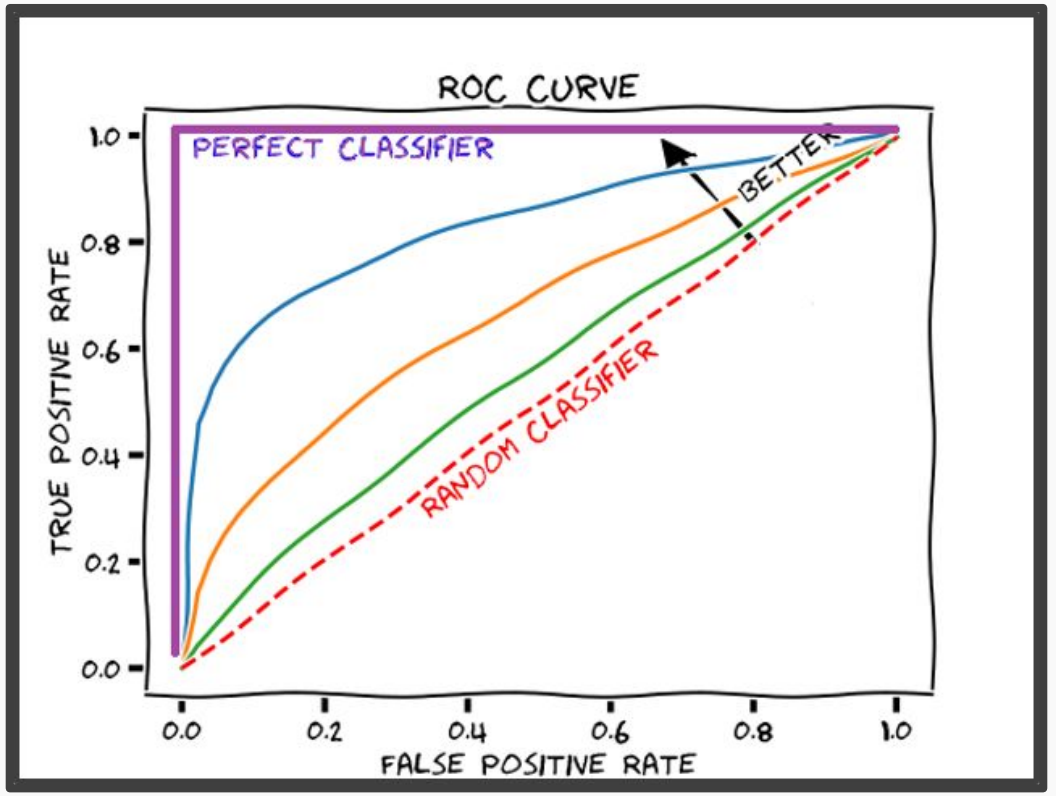
- Classifometrics TM
- ROC Curves
- OVR & Multinomial Logistic Regression

Classifometrics™

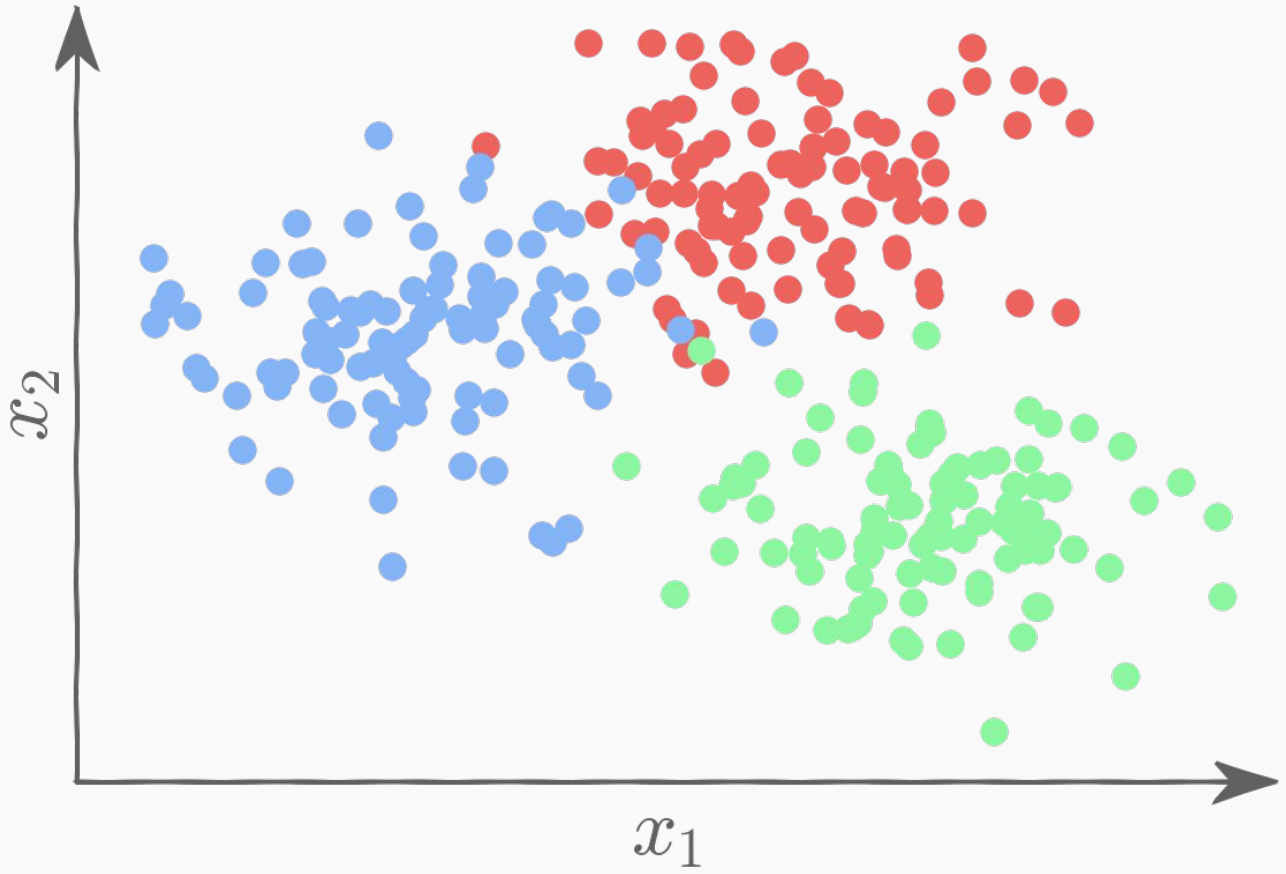
Classification Metric	Formula	Logistic Regression	kNN Classification
Accuracy		62%	62%
Sensitivity (Recall)		72%	75%
Specificity		50%	47%
Precision		62%	62%
F1 score		67%	68%

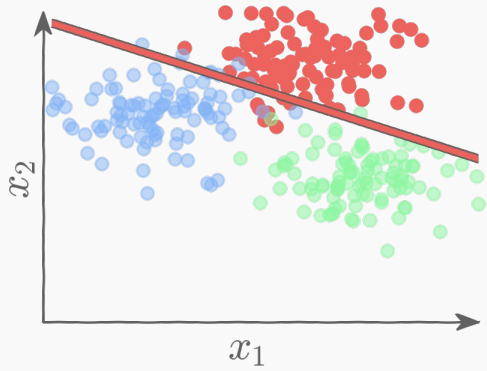
Receiver Operating Characteristic curve (ROC)

- The ROC curve was first developed by radar engineers during World War II for detecting enemy objects in battlefields.
- The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings.



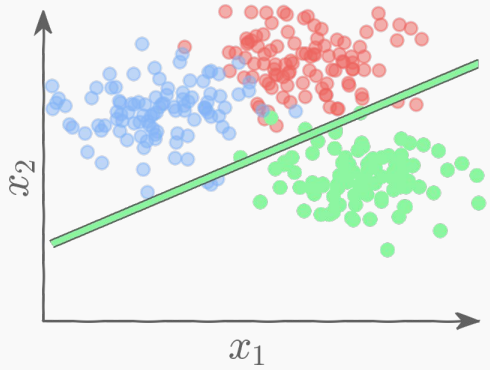
One-vs-Rest Logistic Regression





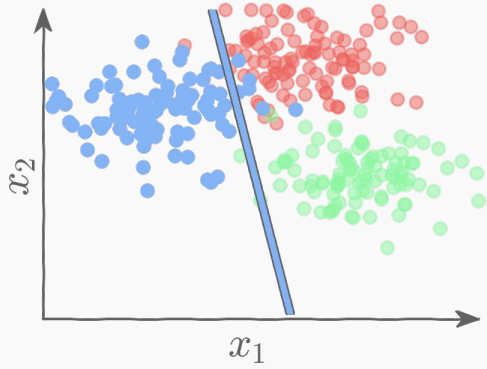
Red vs others

$$log\frac{p(r)}{p(o)} = \beta_r X$$



Green vs others

$$log\frac{p(g)}{p(o)} = \beta_g X$$

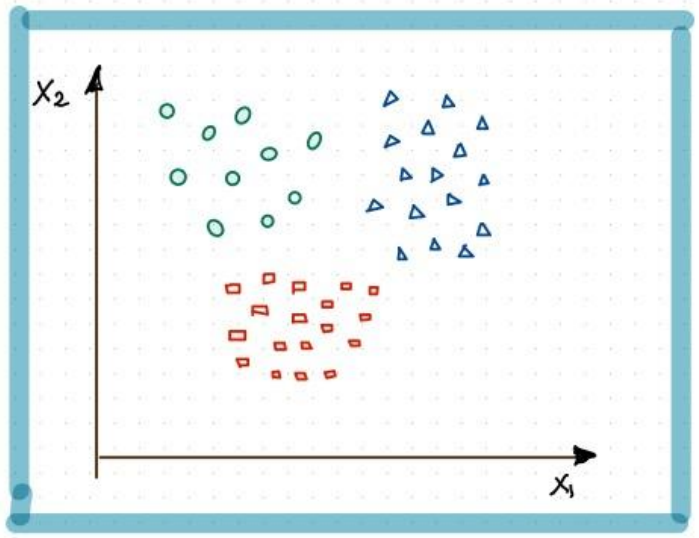


Blue vs others

$$log\frac{p(b)}{p(o)} = \beta_b X$$

One vs. Rest (ovr)

Classifying three classes,
Red, Blue and Green can be turn into three binary Logistic Regressions



Blue vs others

$$\log\frac{P(b)}{P(o)} = \beta_b X$$

Green vs others

$$\log\frac{P(g)}{P(o)} = \beta_g X$$

Red vs others

$$\log\frac{P(r)}{P(o)} = \beta_r X$$

`sklearn` normalizes the output of each of the three models when predicting probabilities:

$$\tilde{P}(b) = \frac{P(b)}{P(g)+P(b)+P(r)} \qquad \tilde{P}(g) = \frac{P(g)}{P(g)+P(b)+P(r)} \qquad \tilde{P}(r) = \frac{P(r)}{P(g)+P(b)+P(r)}$$

True Multinomial Logistic Regression

This is mathematically equivalent to using the softmax function:

$$P(y = k) = \frac{e^{X\beta_k}}{\sum_{\quad} e^{X\beta_{\quad}}}$$

Missing Data

Missing Data

- Types of Missingness
- Dealing with Missingness

Types of Missingness

There are 3 major **types of missingness** to be concerned about:

1. **Missing Completely at Random (MCAR)**
2. **Missing at Random (MAR)**
3. **Missing Not at Random (MNAR)**

Types of Missingness

There are 3 major types of missingness to be concerned about:

1. **Missing Completely at Random (MCAR)** - the probability of missingness in a variable is the same for all units. Like randomly poking holes in a data set.
2. **Missing at Random (MAR)**
3. **Missing Not at Random (MNAR)**

Types of Missingness

There are 3 major types of missingness to be concerned about:

1. **Missing Completely at Random (MCAR)**
2. **Missing at Random (MAR)** - the probability of missingness in a variable depends only on available information (in other predictors).
3. **Missing Not at Random (MNAR)**

Types of Missingness

There are 3 major types of missingness to be concerned about:

1. **Missing Completely at Random (MCAR)**
2. **Missing at Random (MAR)**
3. **Missing Not at Random (MNAR)** - the probability of missingness depends on information that has not been recorded and this information also predicts the missing values.

Naively Handling Missingness (Dropping)

What's the simplest ways to handle missing data?

We can just throw away or '**drop**' the observations that have any missing values.

df.dropna(axis=0) will drop any rows with one or more missing values from your Pandas DataFrame.

Dropping is 'easy,' but when is it advisable?

Simple Imputation Methods

A common method is to impute using some sort of average value.

What we mean by ‘average’ can depend on the type of the predictor variable we want to impute:

Quantitative: Impute the mean or median value

Categorical: Impute the mode, that is, the most common class

Model-Based Imputation Example

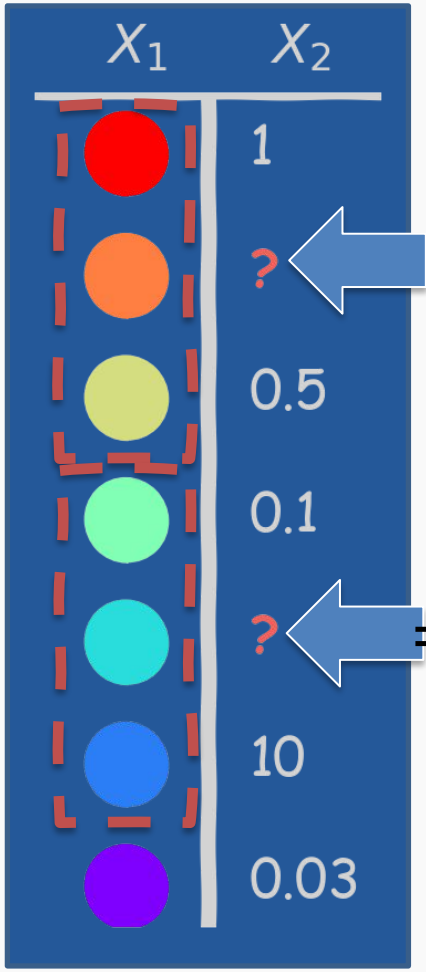
- Here we have two predictors, X_1 and X_2.
- How might we use a model to fill in the missing values in X_2?
- Think of X_1 as the predictor and X_2 as the response!
- An imputation model can then learn a relation between X_1 and X_2 from the complete observations

kNN Imputation

Let's see a kNN imputation model in action with k = 2

- To impute the first missing value…
- Find the offending observation's 2 nearest neighbors in terms of the data we *do* have (that is, X_1)
- Average the neighbor's values for the missing predictor and impute!

  And repeat as needed!



= (1+0.5)/2 = 0.75

= (0.1+10)/2 = 5.05

Imputation with Missingness Indicator Variable

We can capture information about the fact of the original missingness by creating 2 new indicator variables.

For example, the indicator variable

10	.	10	0	0	1
5	1	5	1	0	0
21	0	21	0	0	0
15	0	15	0	0	0
16	.	16	0	0	1
.	.	14.29	0	1	1
21	1	21	1	0	0
12	0	12	0	0	0
.	0	14.29	0	1	0

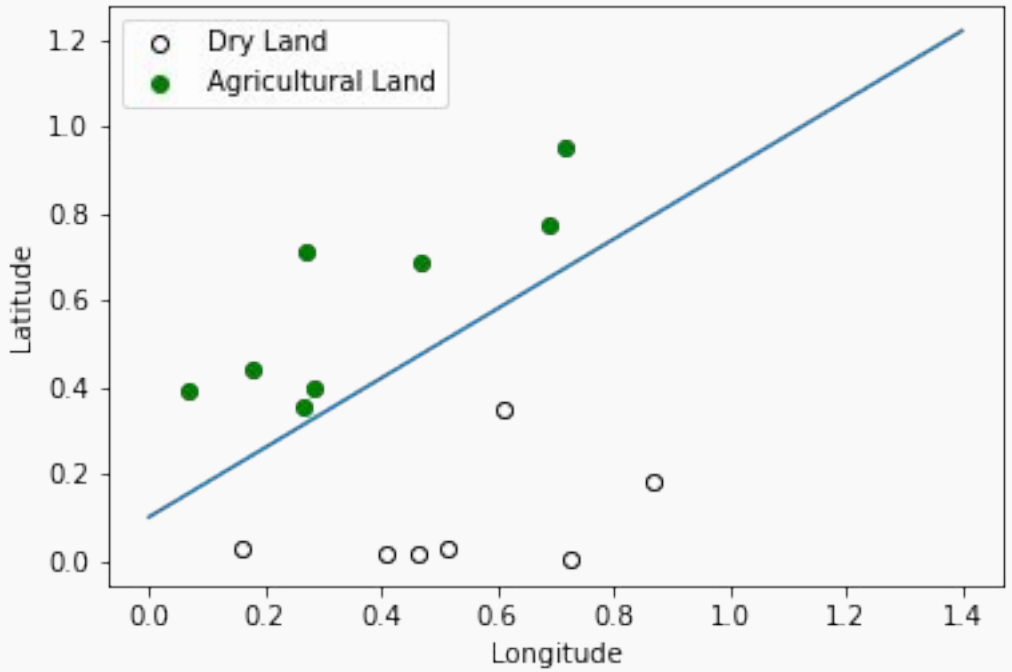
Decision Trees

Decision Trees

- Motivation: Classifcation Boundaries
- Predictions
- Splitting Criteria
- Stopping Conditions

Decision Trees: Motivation

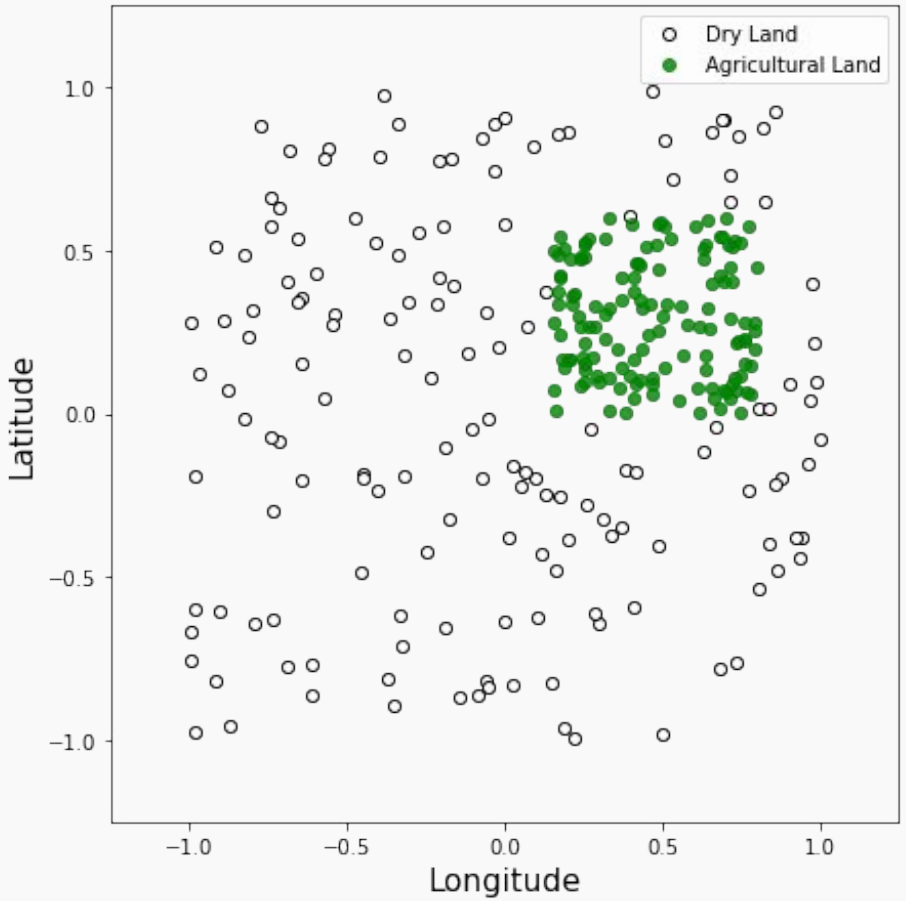
Question: Can you guess the equation that defines the decision boundary below?

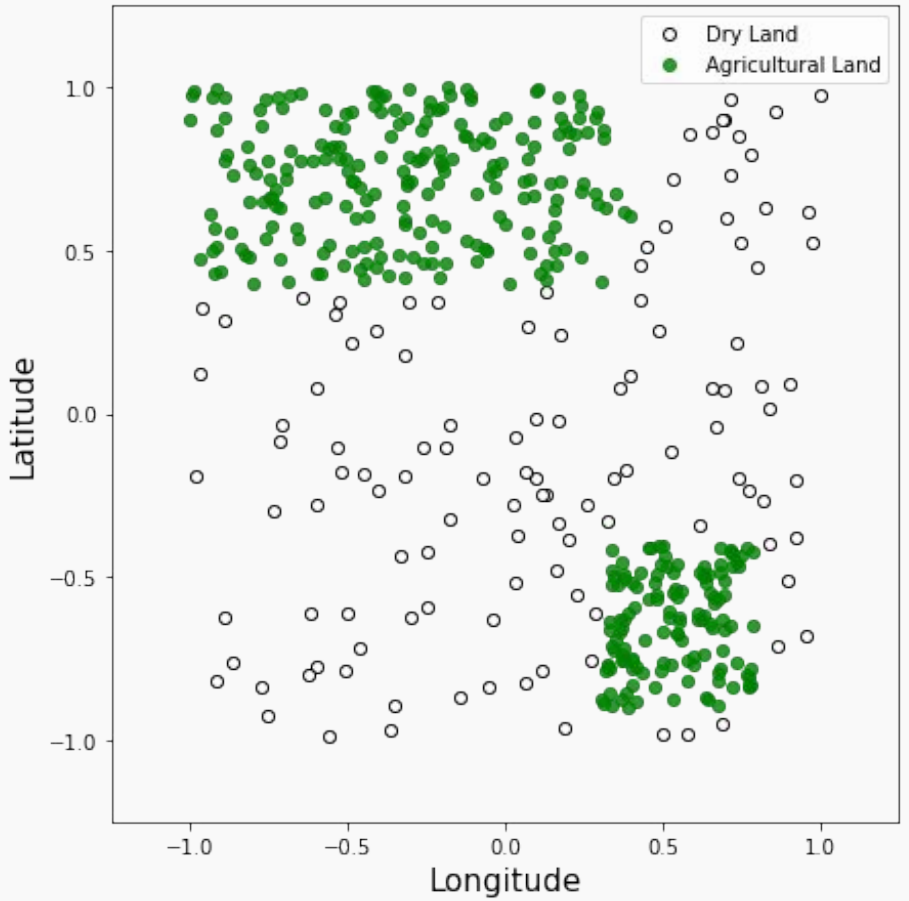


$$Latitude = 0.8\ Longitude + 0.1$$
$$or$$
$$-0.8Longitude + Latitude - \ 0.1 = 0$$

Motivation

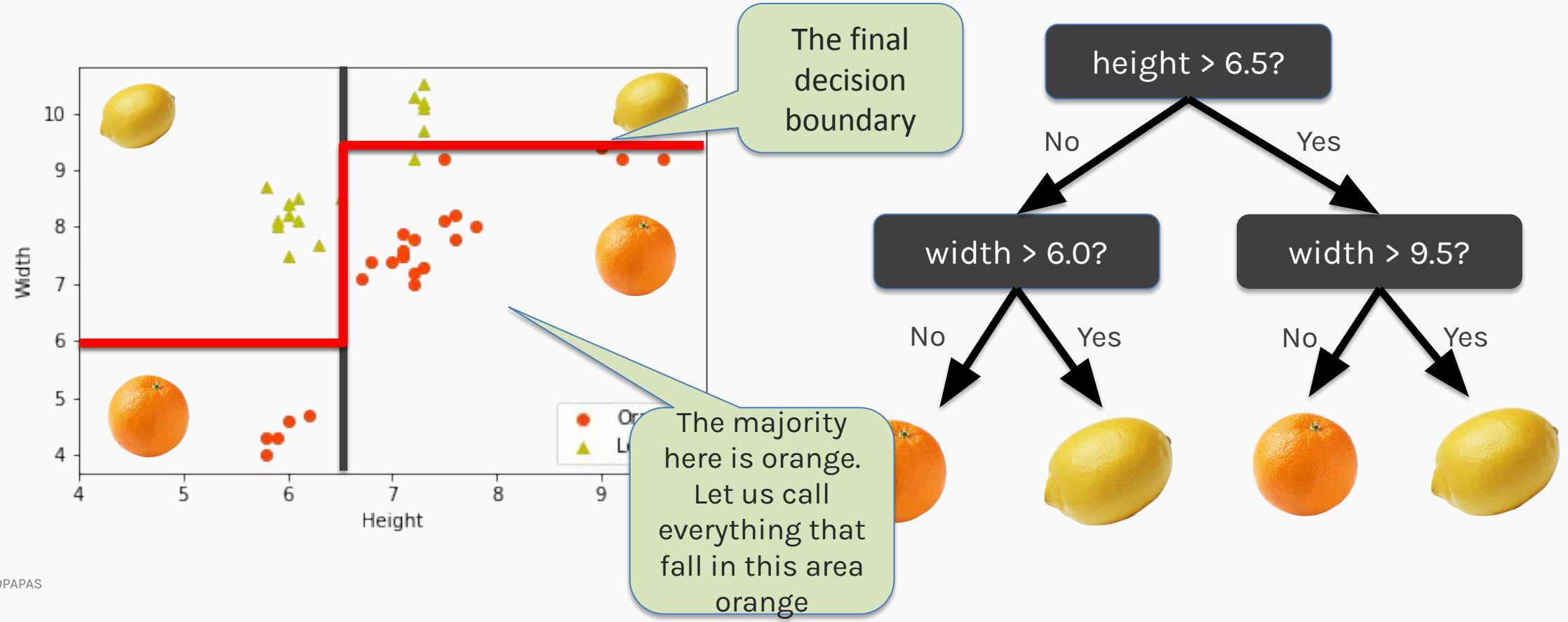
Question: How about these?





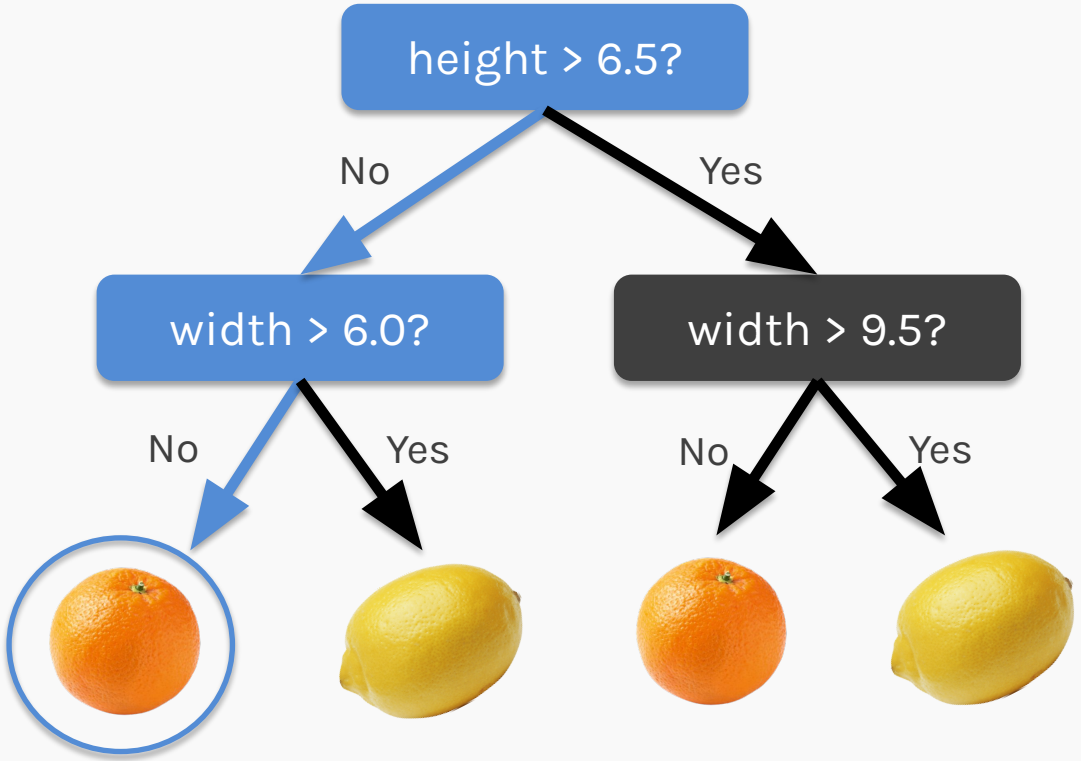
The Geometry of Flow Charts

Every flow chart tree corresponds to a partition of the feature space by **axis aligned lines or (hyper) planes**. Conversely, every such partition can be written as a flow chart tree.

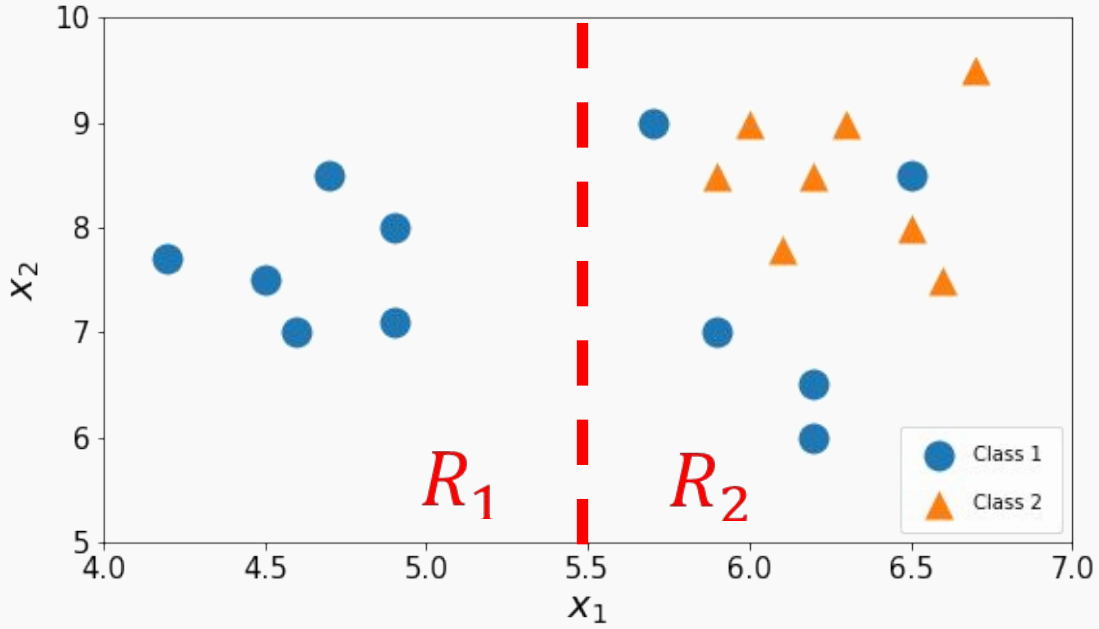


Prediction

height = 5.9
width = 5.8



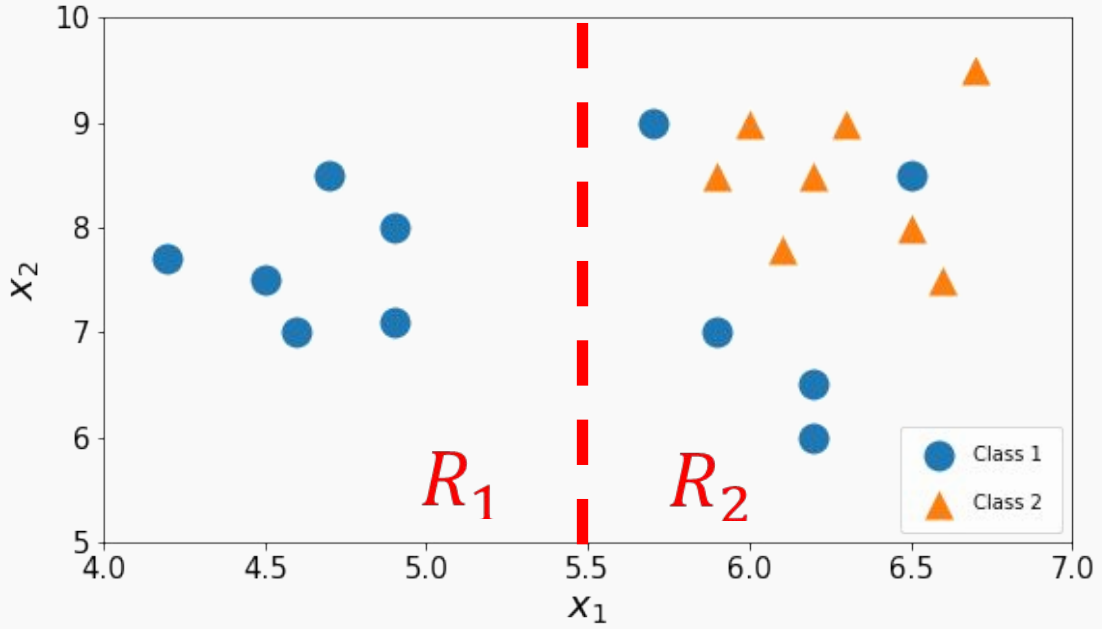
Gini Index



	●	▲	
R_1	6	0	$1 - \left[\left(\frac{6}{6}\right)^2 + \left(\frac{0}{6}\right)^2\right] = 0$
R_2	5	8	$1 - \left[\left(\frac{5}{13}\right)^2 + \left(\frac{8}{13}\right)^2\right] = \frac{80}{169} = 0.47$

Compared to 0.38 when we used mis-classification erro

Entropy - Example



	●	▲	
	6	0	$-\left[\frac{6}{6}\log_2\frac{6}{6}+\frac{0}{6}\log_2\frac{0}{6}\right]=0$
	5	8	$\left[\frac{5}{13}\log_2\frac{5}{13}+\frac{8}{13}\log_2\frac{8}{13}\right]\approx 1.3$

Compared to 0.38 and 0.47 using misclassification rate and the Gini index

Stopping Conditions

Hyperparameters that determine when the splitting terminates

- Max depth
- Minimum samples per leaf
- Max leaf nodes
- Min impurity decrease

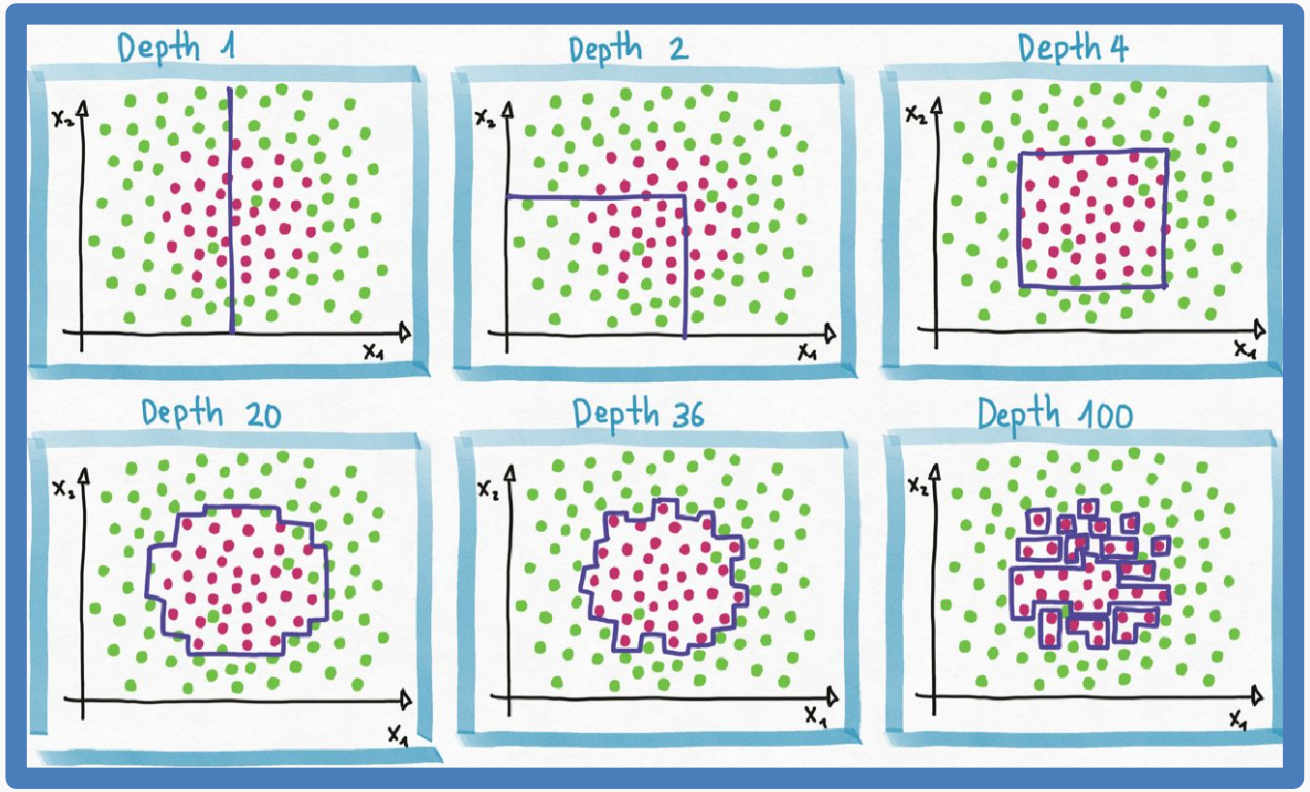
Ensemble Methods:
Bagging & Random Forest

Bagging & Random Forest

- Motivation: Deep Trees Overfit!
- Ensemble Learning
- Out of Bag Error (OOB)
- Random Forest: Improved Variance Reduction
- Variable Importance
- Imbalanced Data

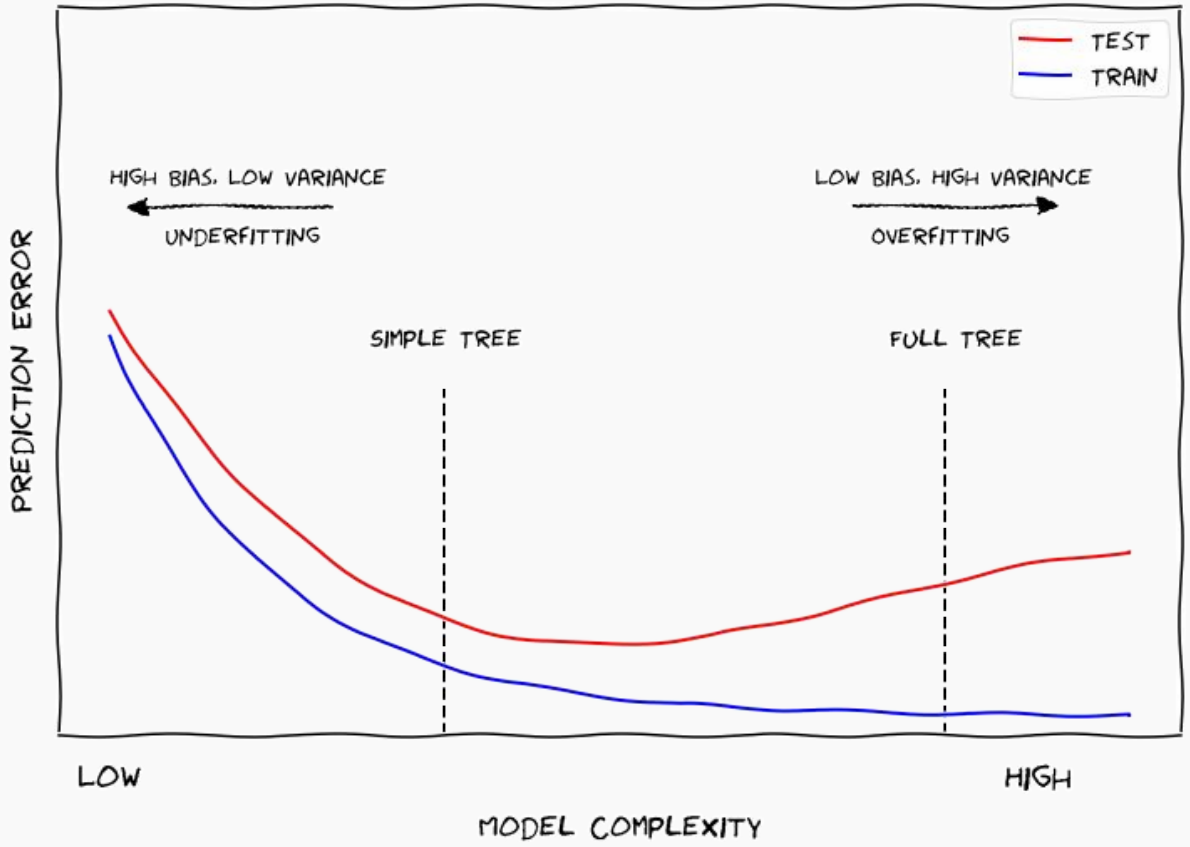
Overfitting

When a tree is too shallow, it cannot divide the input data into enough regions, so the model underfits. When the tree is too deep it cuts the input space into too many regions and fit to the noise of the data so it overfits.



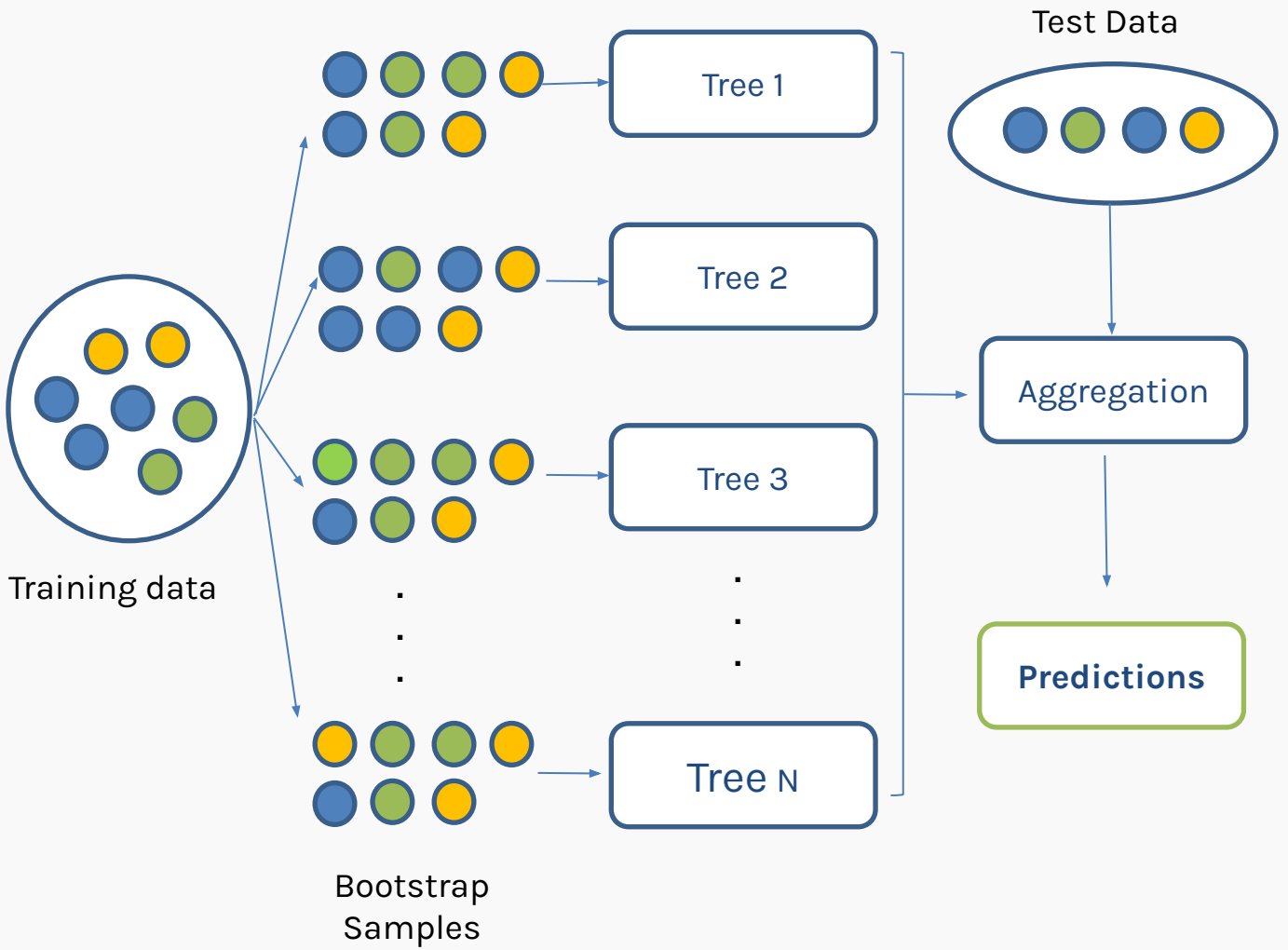
Overfitting

Avoid overfitting by pruning or limiting the depth of the tree and using CV.

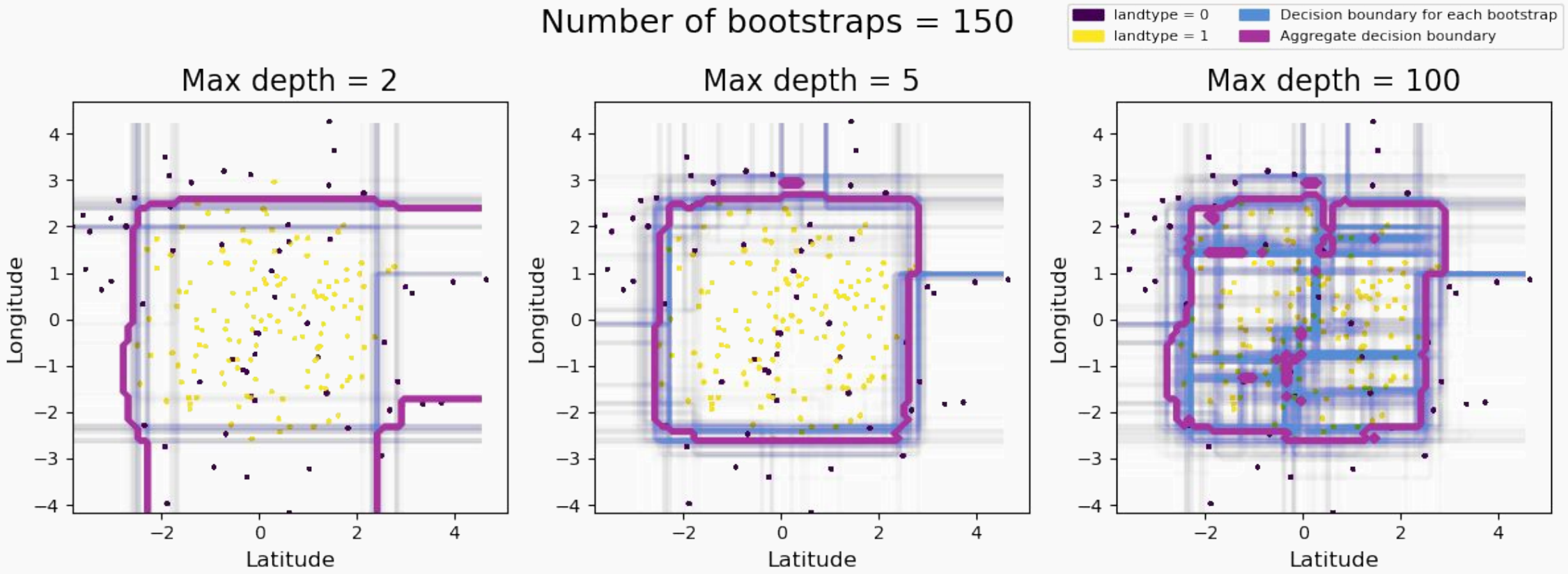


Bagging - Bootstrap + Aggregate

1. **Bootstrap:** we generate multiple samples of training data, via bootstrapping. We train a deeper decision tree on each sample of data.
2. **Aggregate:** for a given input, we output the averaged outputs of all the models for that input.



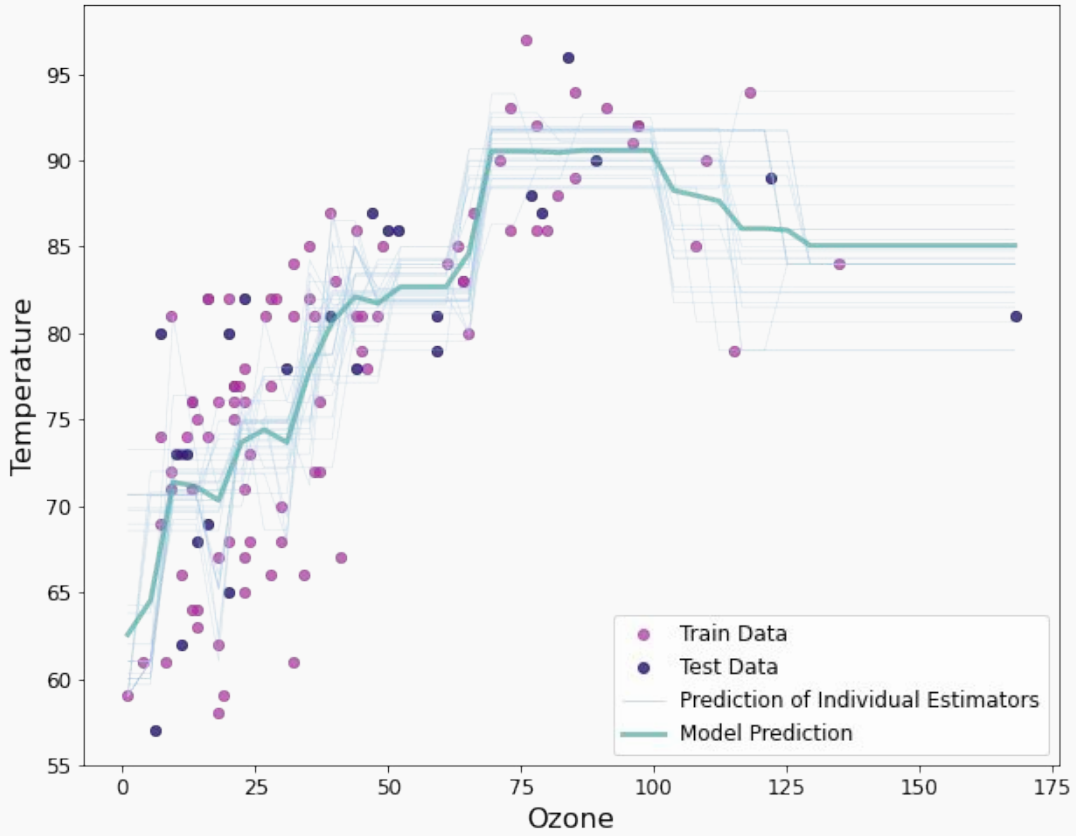
150 magic realisms



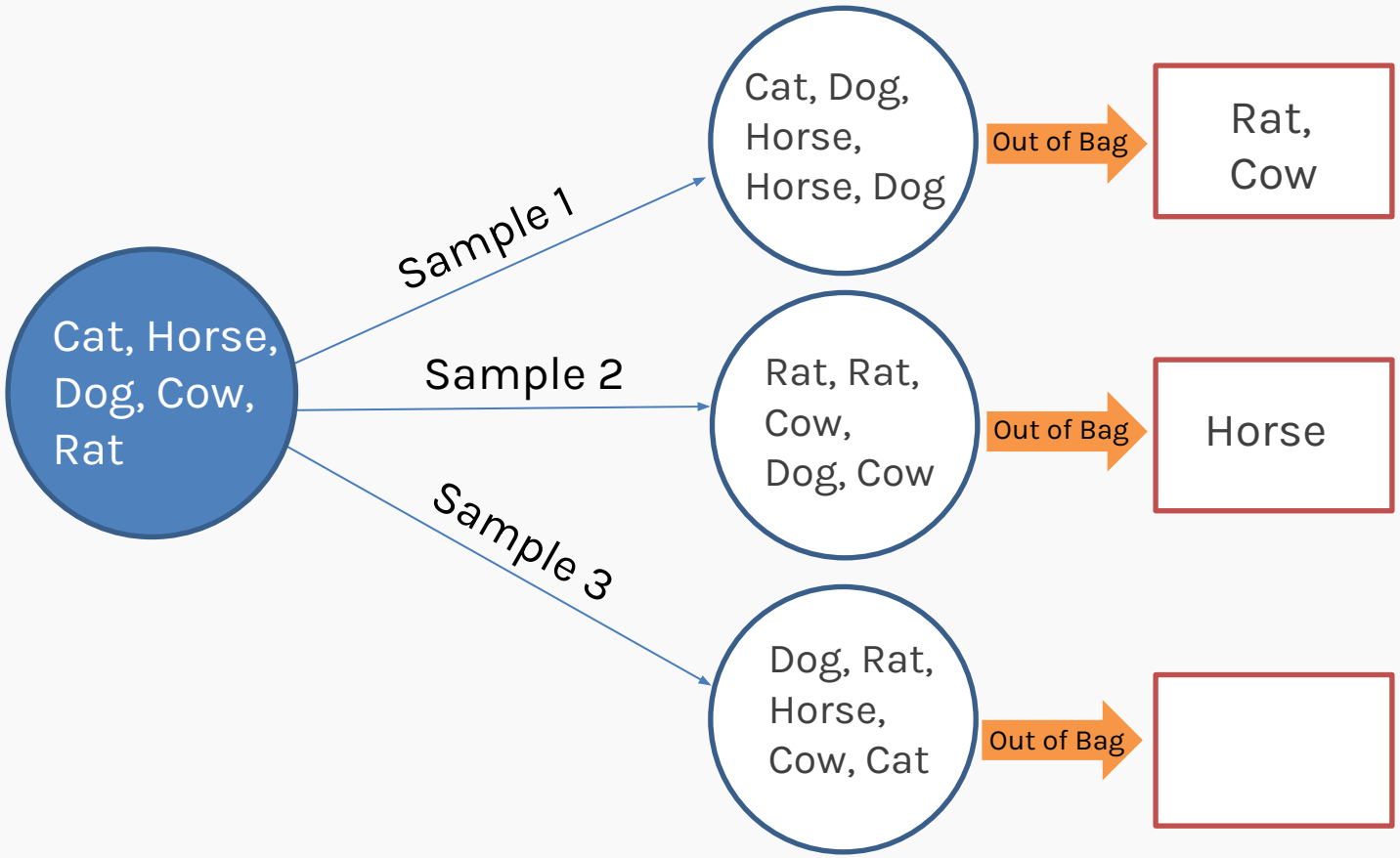
Regression in Bagging

The prediction by the Bagging Regressor model is average of all the individual predictions of the trees or estimators.

Resulting Prediction by the Bagging Regressor



What is OOB?

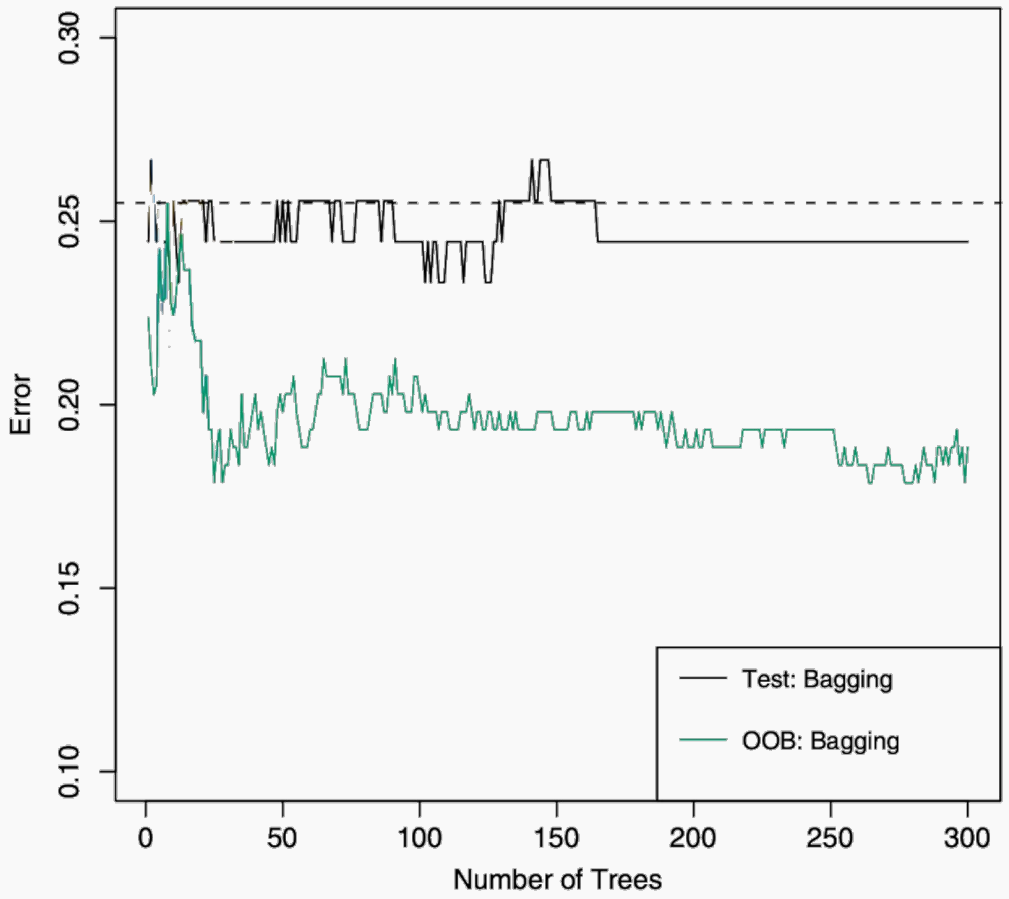


Out-of-bag (OOB) error/Out-of-bag estimate is a method of determining the prediction error that allows the trees to be fit and validated whilst being trained.

Why?

- To measure generalizability.
- Replaces the need for a separate measurement of performance for a validation-set performance.

Why OOB Error?

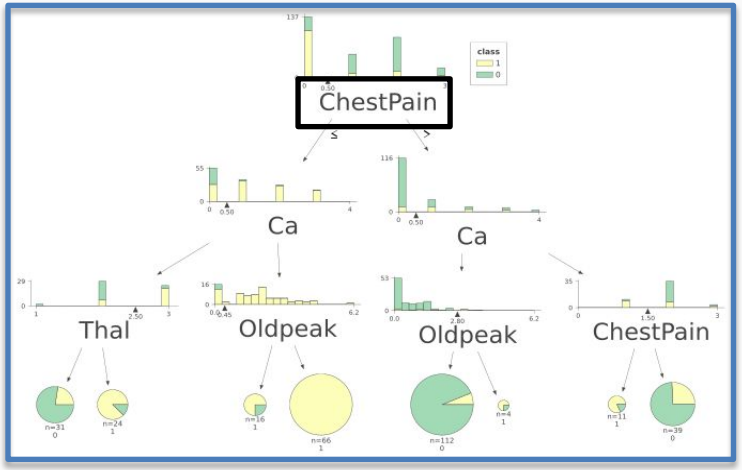


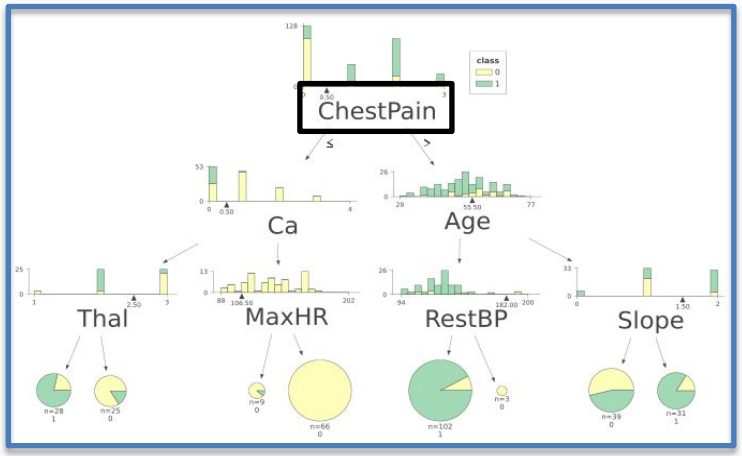
Source: An Introduction to Statistical Learning with Applications in R

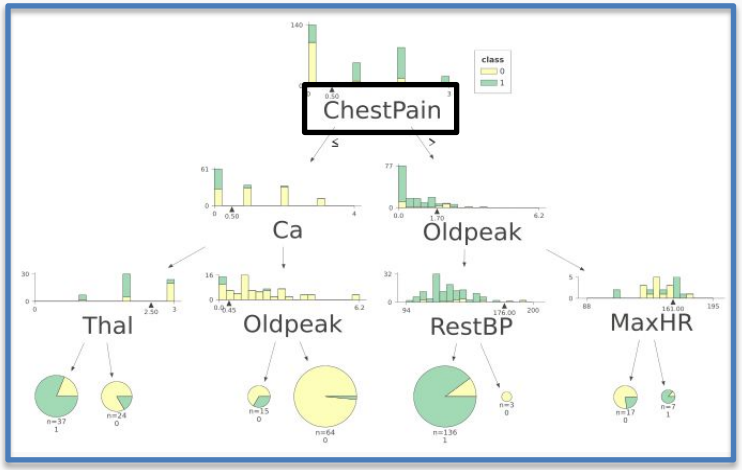
- While using the cross-validation technique, every validation set has already been seen or used in training by a few decision trees and hence there is a leakage of data, therefore more variance. But, OOB Error prevents leakage and gives a better model with low variance, so we use OOB error.
- There is also lesser computational cost for OOB error as compared to CV for bagging.

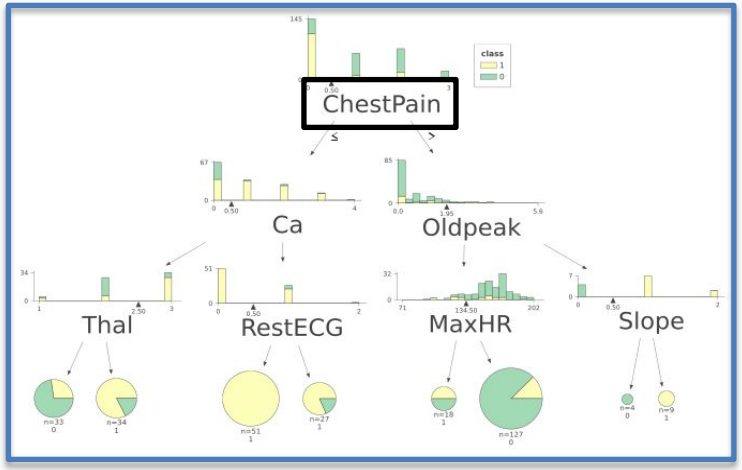
Random Forest Motivation: Decorrelate Trees

Consider the following decision trees in a bagging model that predicts if a person has heart disease:



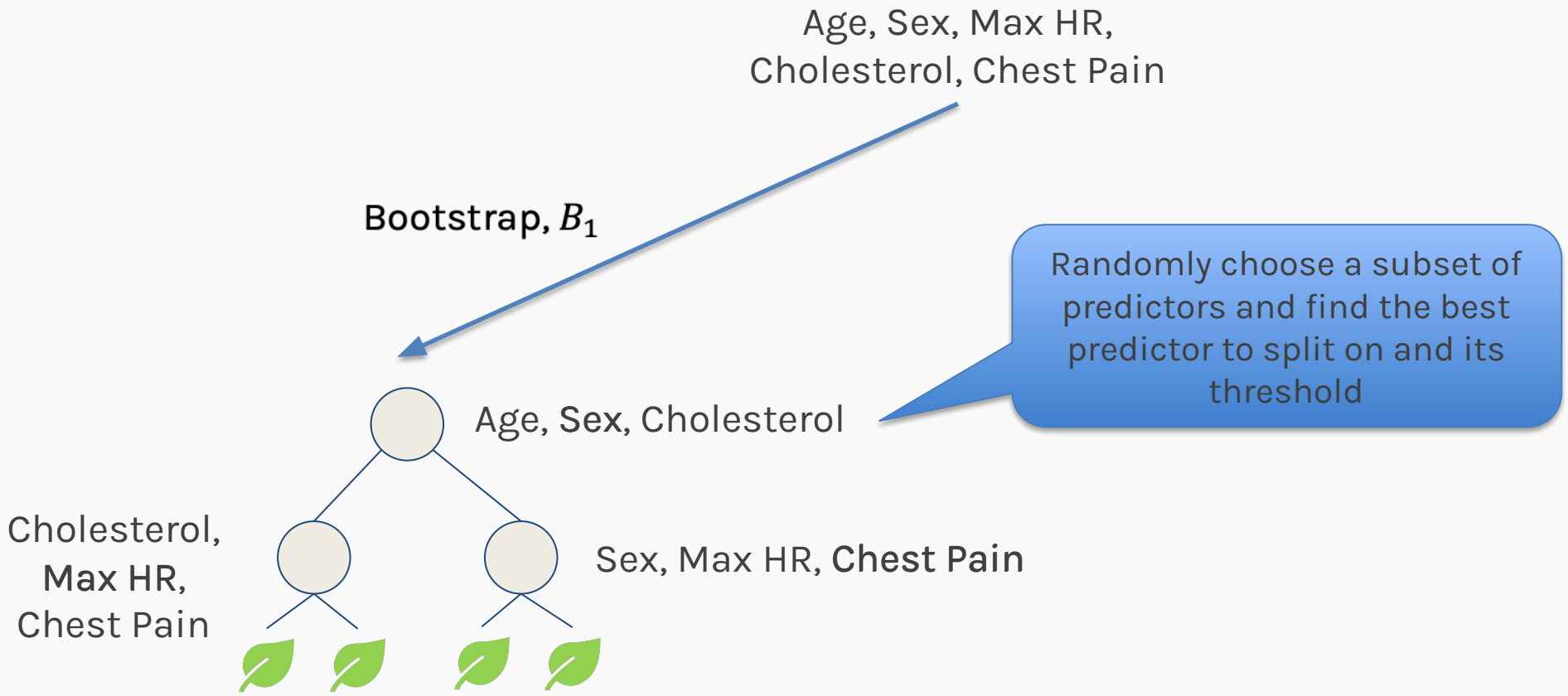




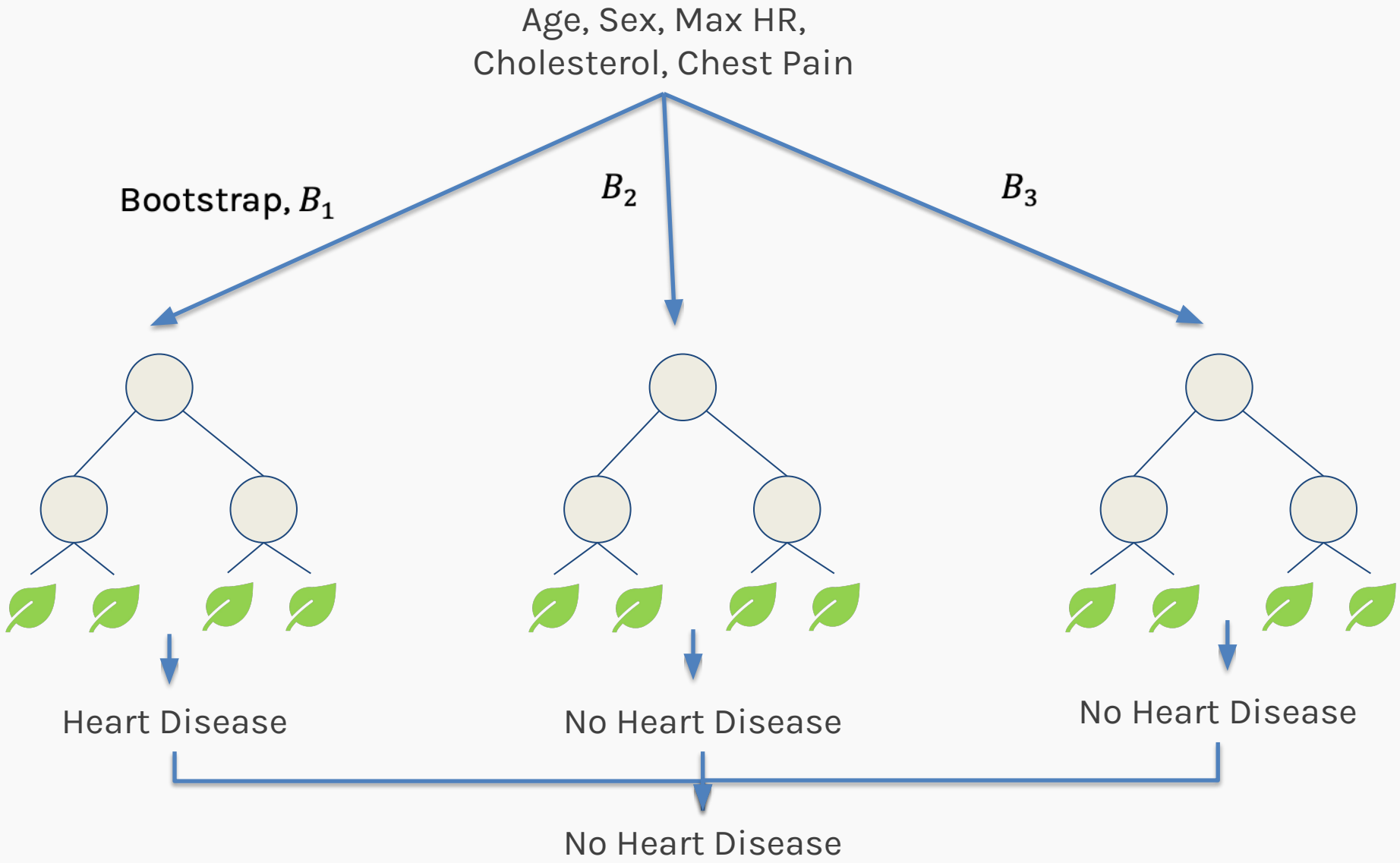


Random Forests: Split on Random Subset of Predictors

Consider a dataset that contains the following predictors:

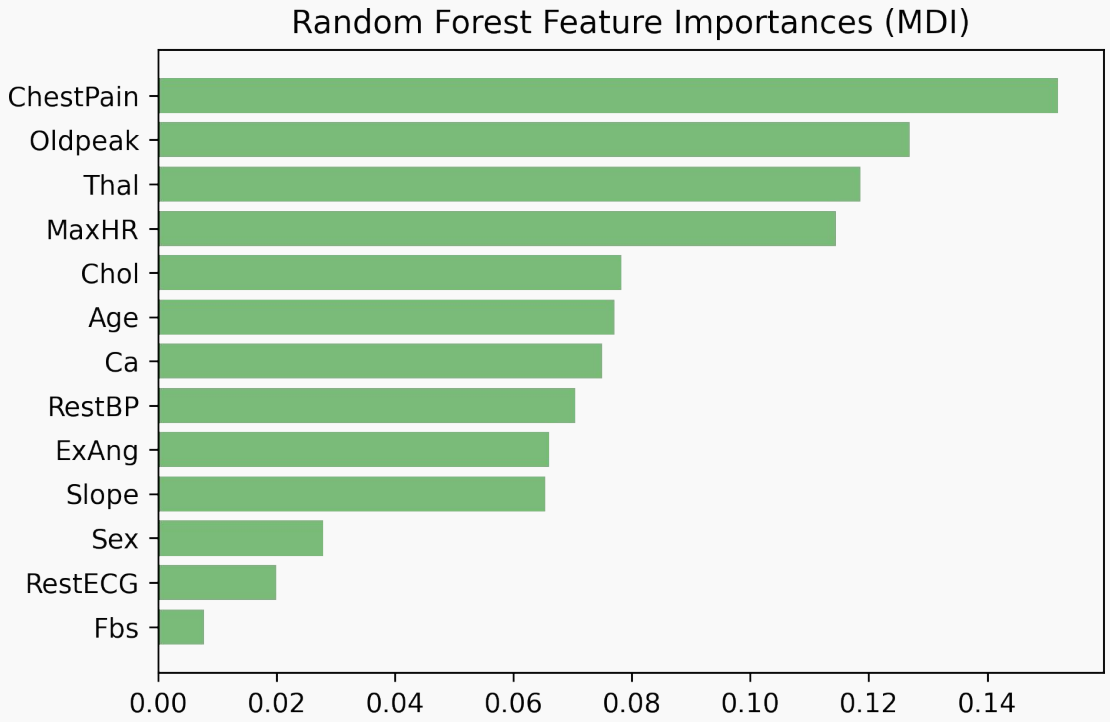


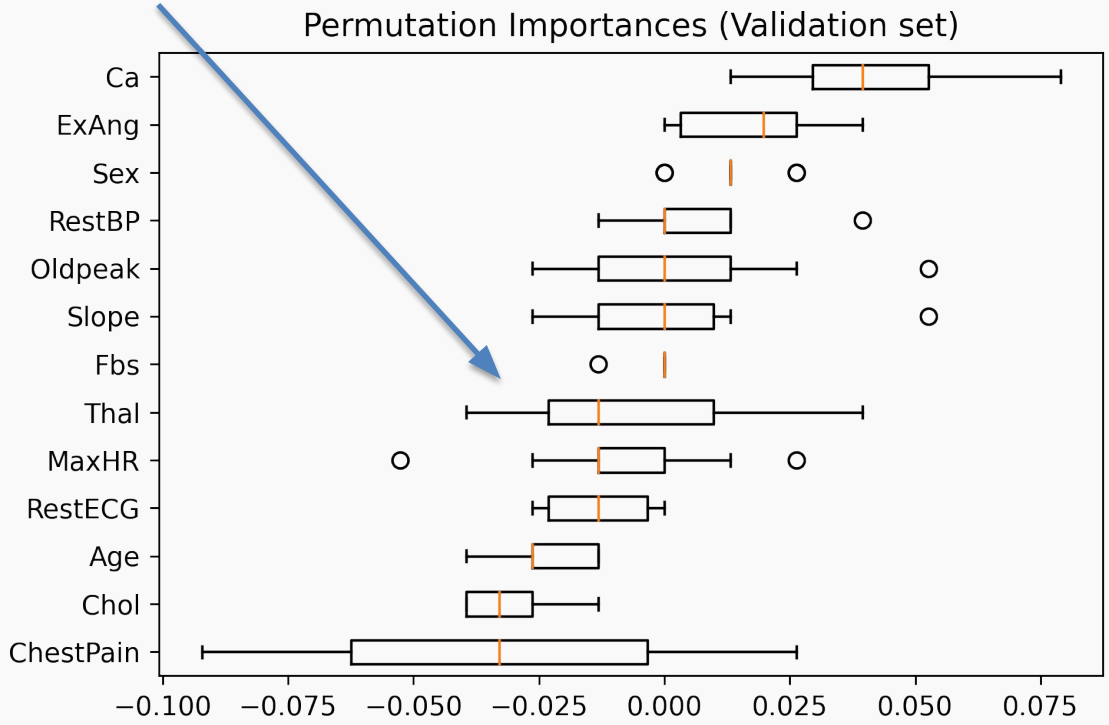
Random Forests: Split on Random Subset of Predictors



Feature Importance: MDI vs Permutation Importance

The experiment is repeated k times.

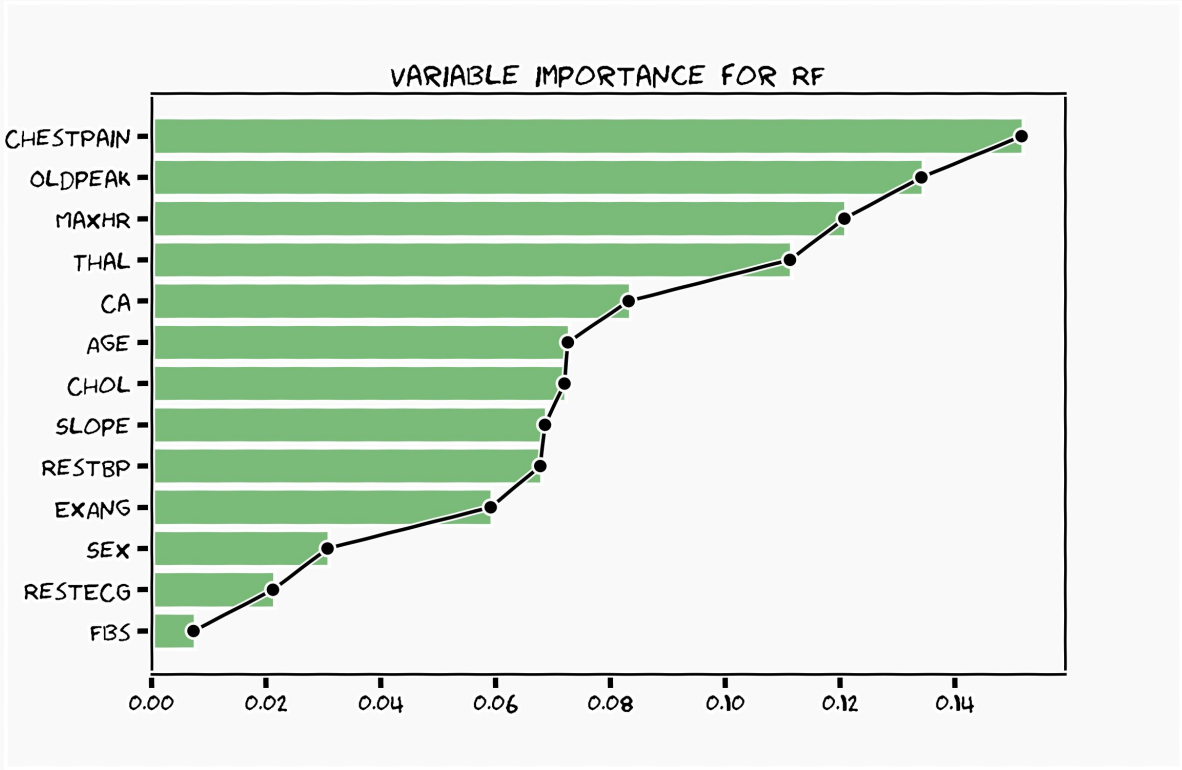


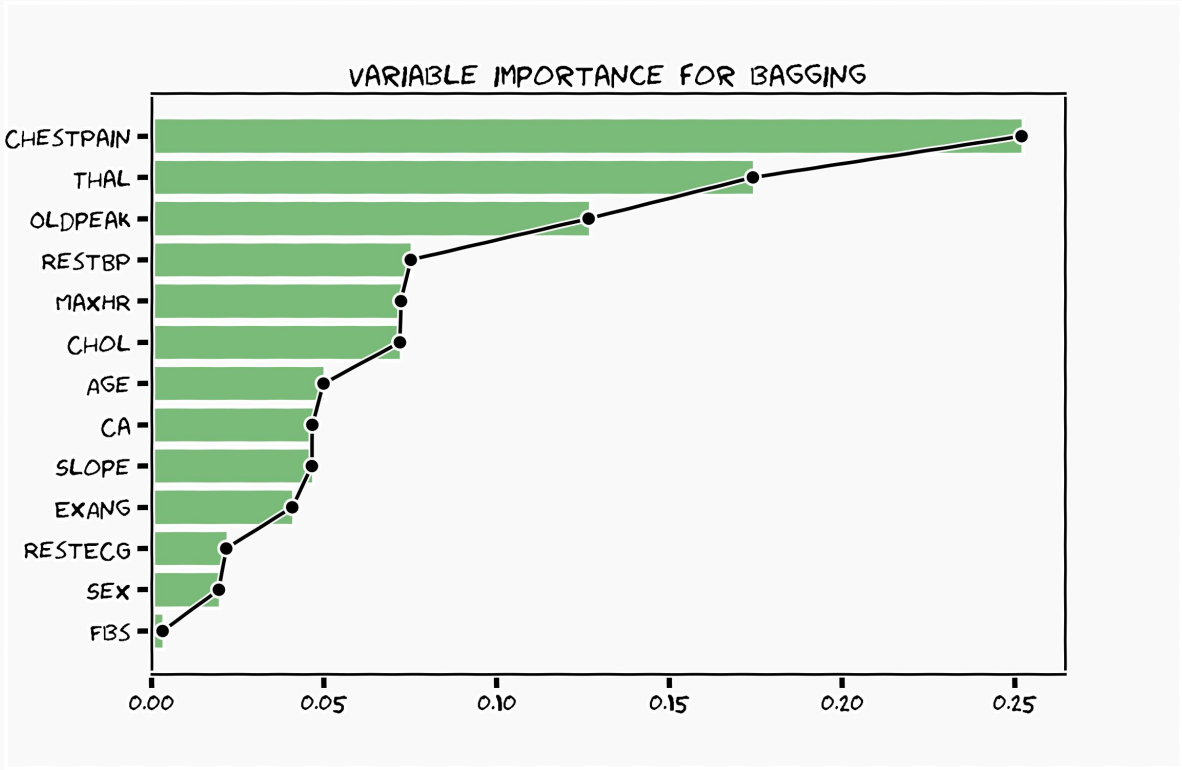


Features like *ChestPain*, *OldPeak*, *Thal* are ranked most important in MDI importance plot but they are ranked low in permutation importance plot.

Variable Importance for bagging vs RF

Variable importance for RF is smoother than that for bagging due to randomness introduced by selecting a subset of predictors to choose from.



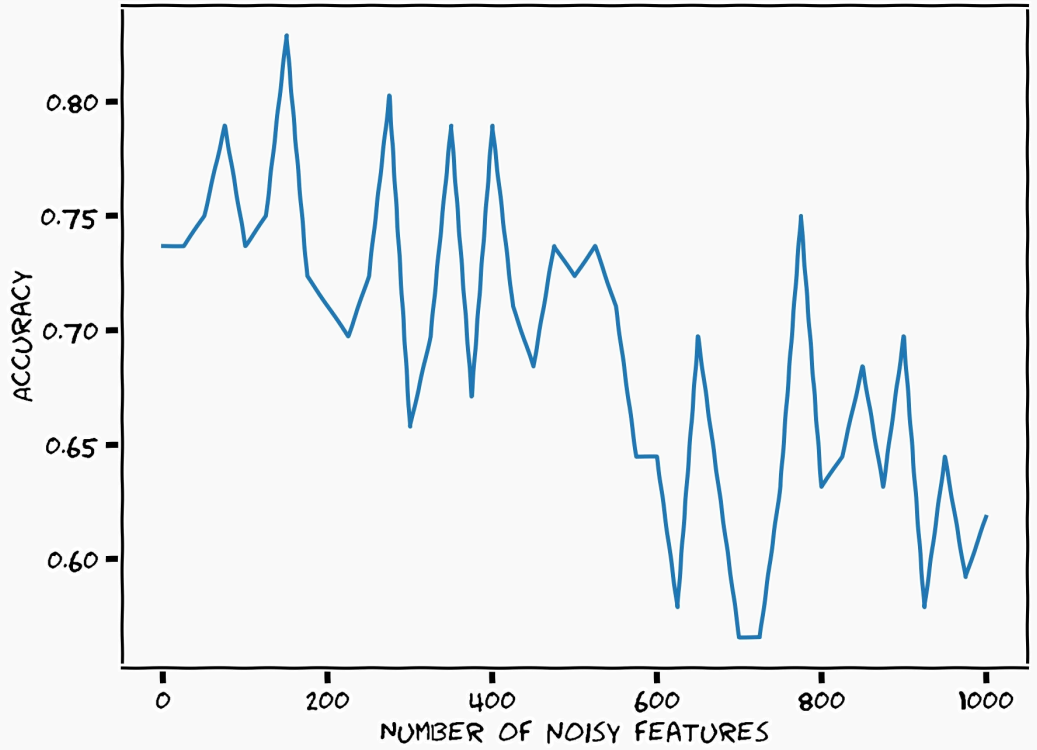


Final Thoughts on Random Forests

When the number of predictors is large, but the number of relevant predictors is small, random forests can perform poorly.

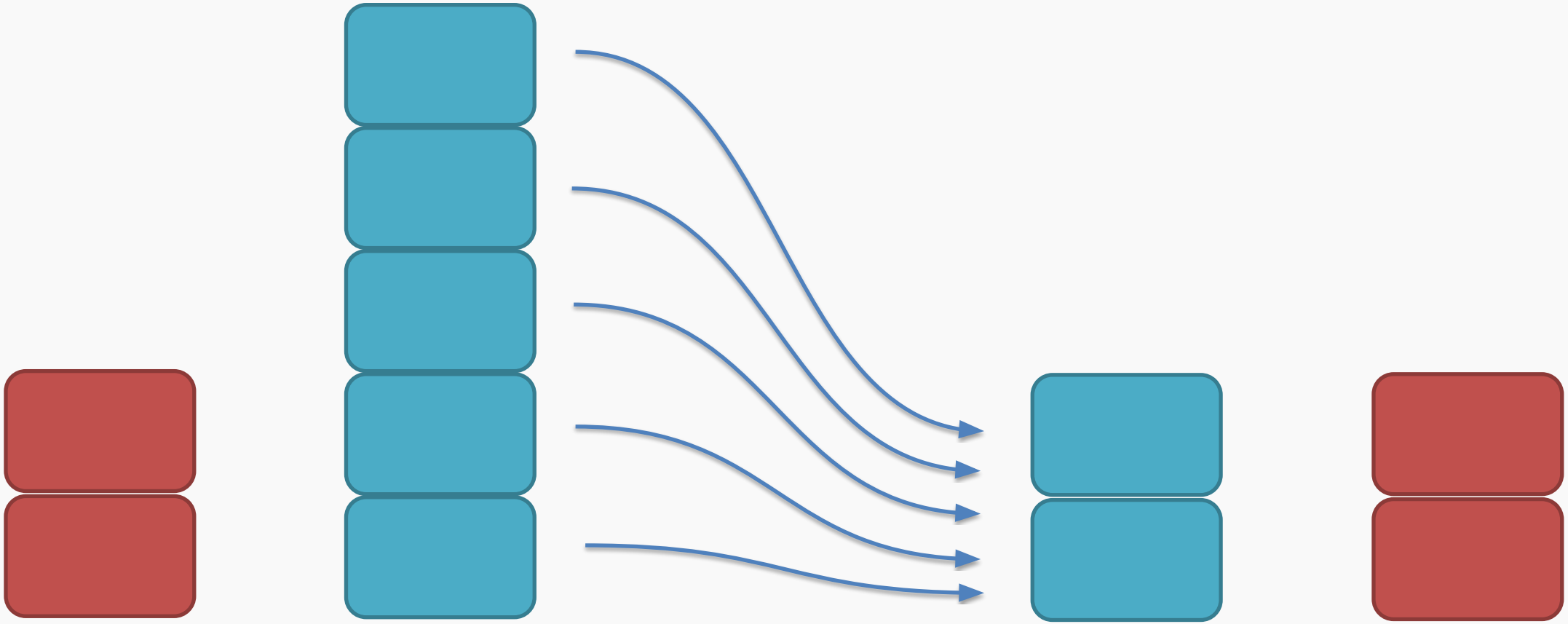
Question: Why?

In each split, the chances of selecting a relevant predictor will be low and hence most trees in the ensemble will be weak models.



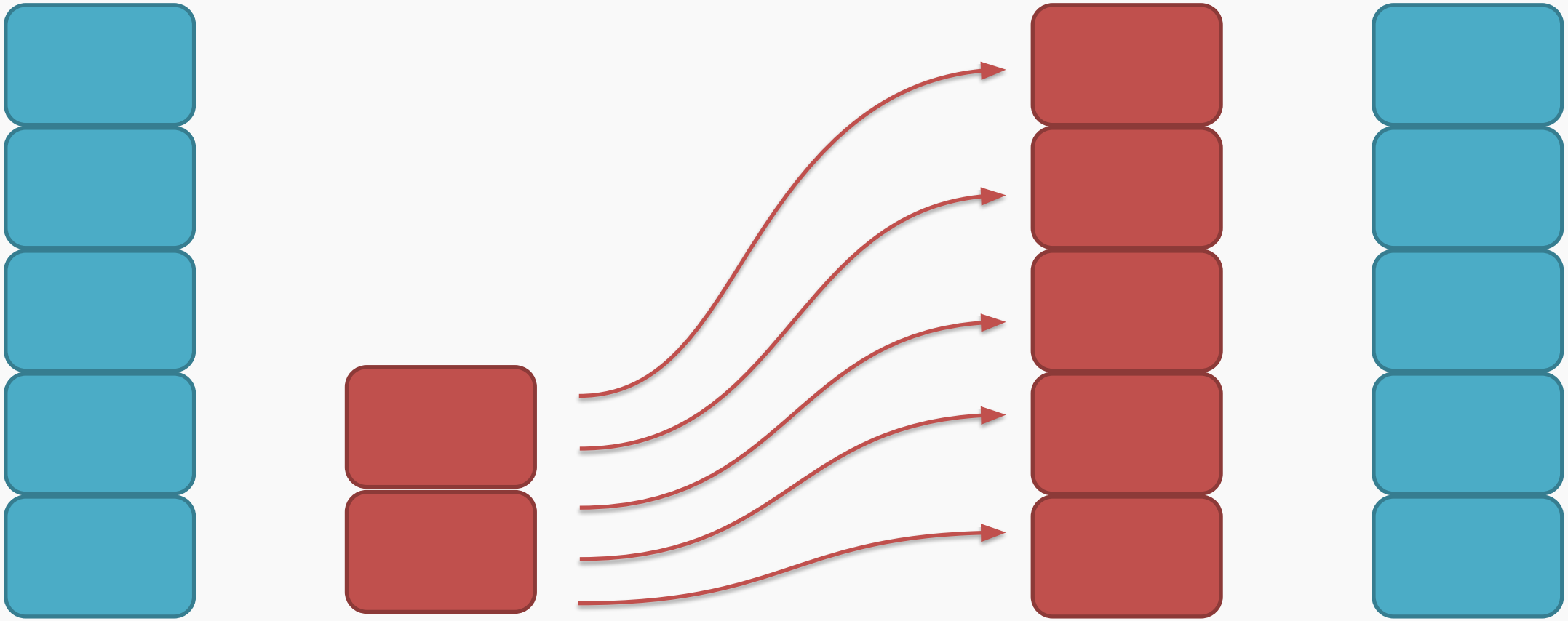
Dealing with Imbalanced Classes

1. Undersampling



Dealing with Imbalanced classes

2. Oversampling



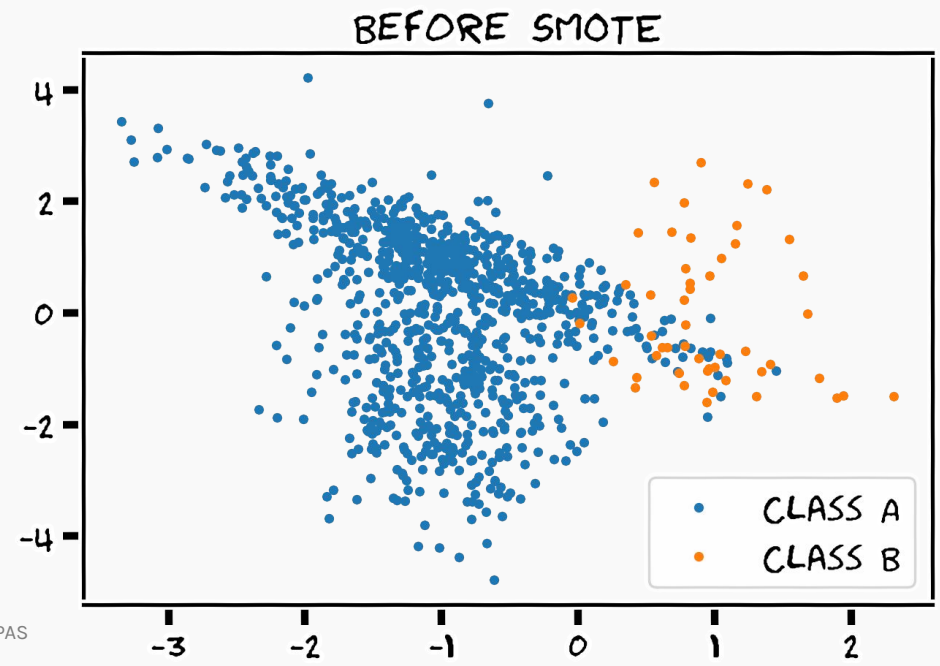
SMOTE (Synthetic Minority Oversampling Technique):

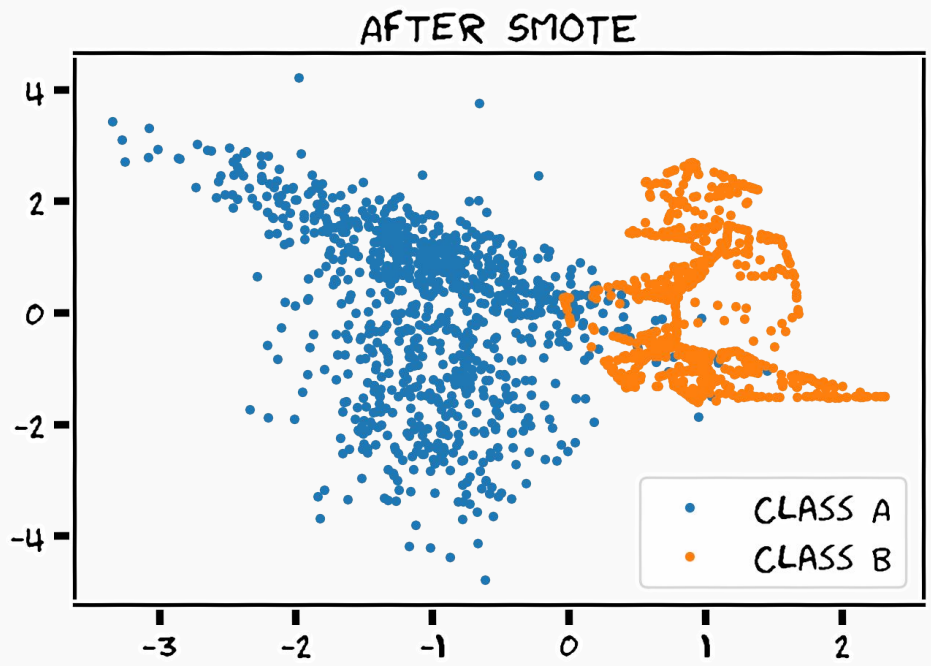
ii. SMOTE:

SMOTE is an improved alternative for oversampling.

SMOTE works by finding points that are closer in feature space.

Drawing a line between these points and generating new data points along this line.





Ensemble Methods: Boosting

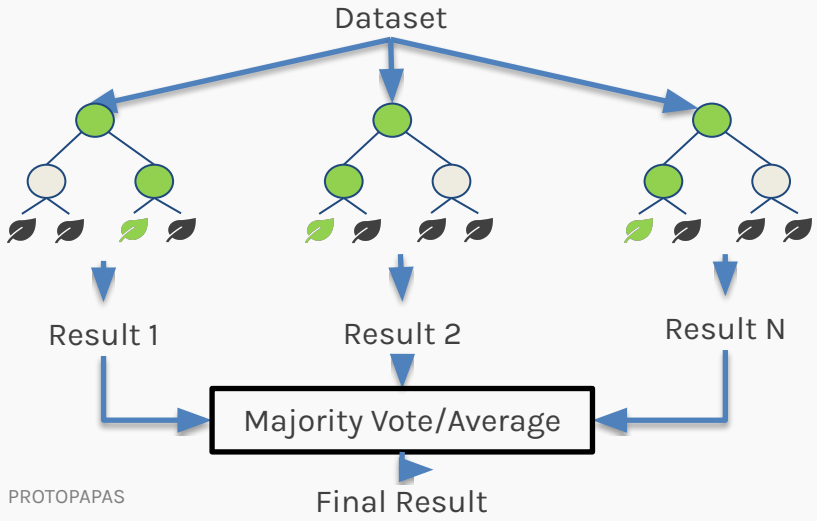
Boosting

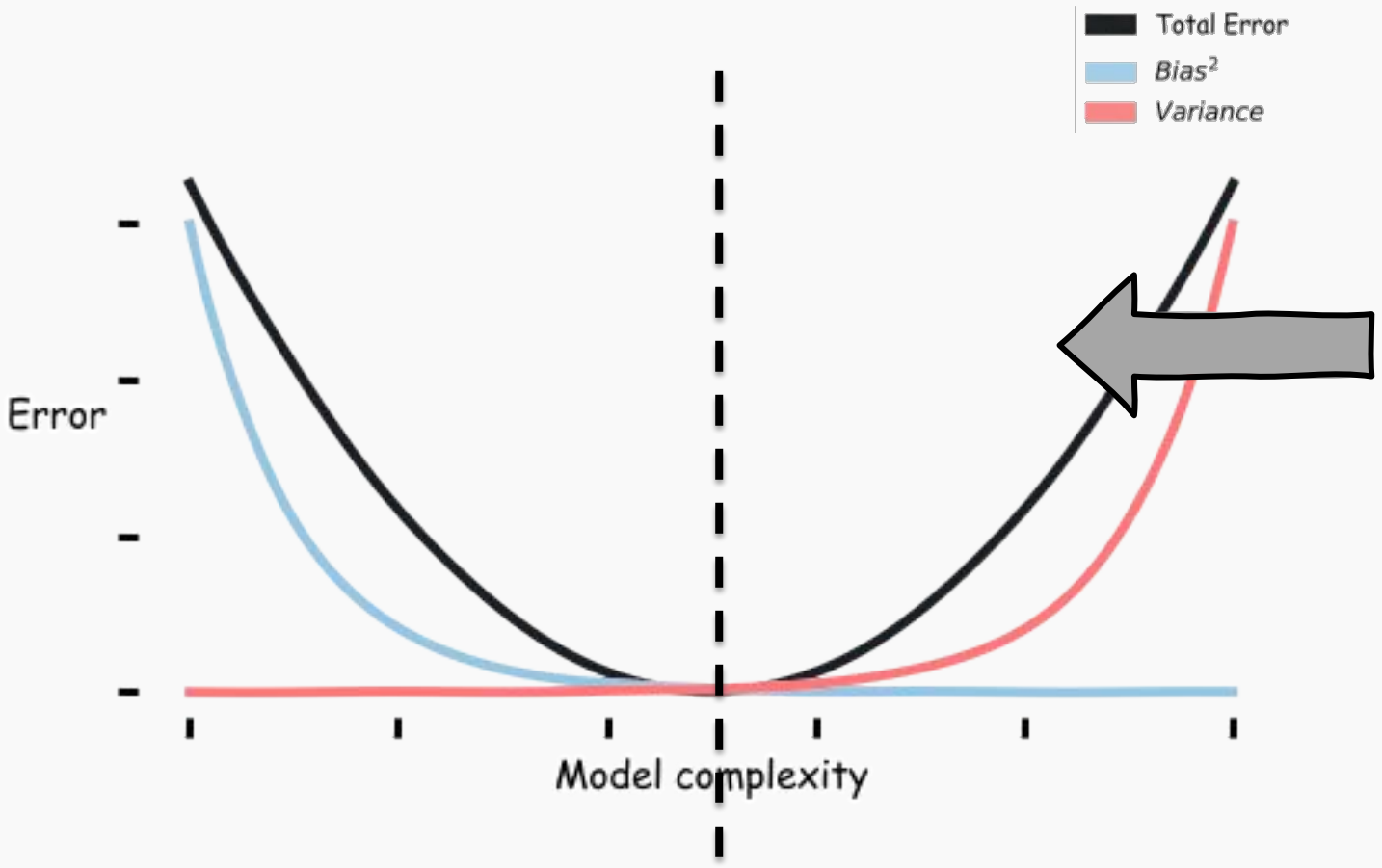
- Motivation: Iteratively Reduce Bias
- Gradient Boosting: Regression
- Boosting Hyperparameters
- AdaBoost: Classification

Option #1: Reduce Variance (Bagging & RF)

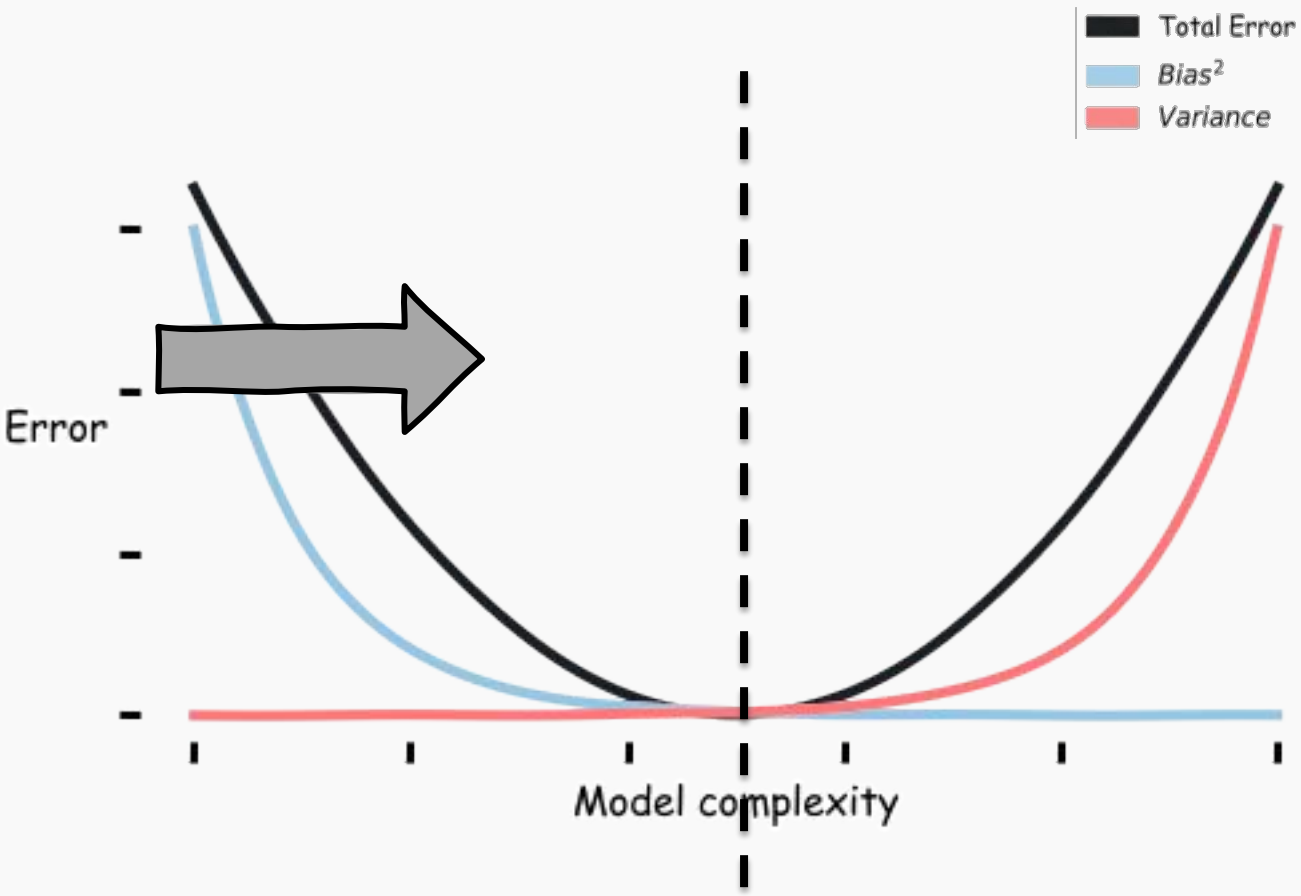
OPTION #1

Reduce variance





Option #2: Reduce Bias (Boosting!)



OPTION #2

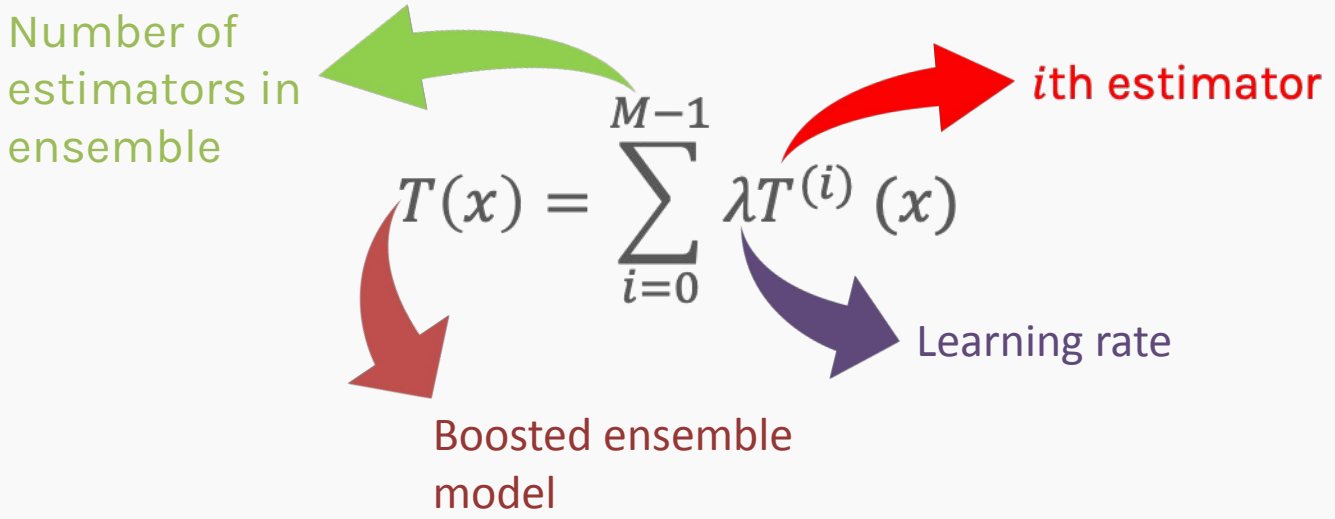
Reduce bias

?

Gradient Boosting

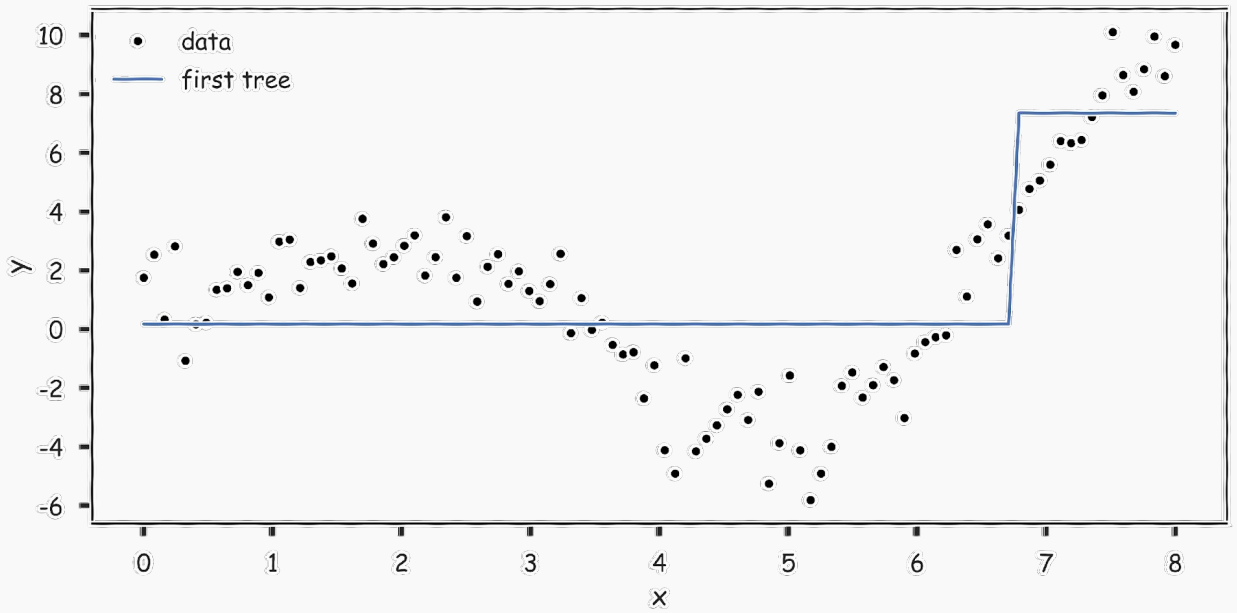
Learning from mistakes

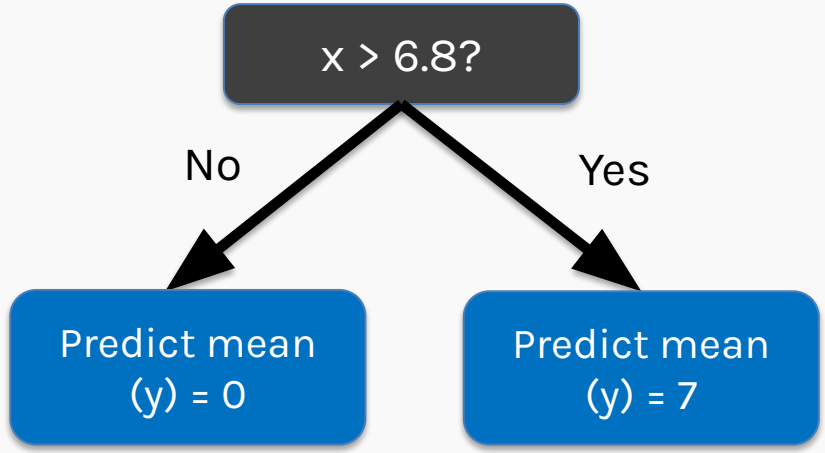
- In gradient boosting, "weak" base estimators are iteratively added to an ensemble after being **fit on the residuals** of the current ensemble.
- The contribution of each new estimator is scaled by a hyperparameter, λ, called the **learning rate**.
- Additional hyperparameters are the number of estimators in the ensemble and the complexity of the base estimators (e.g., max depth).



Gradient Boosting: Illustration

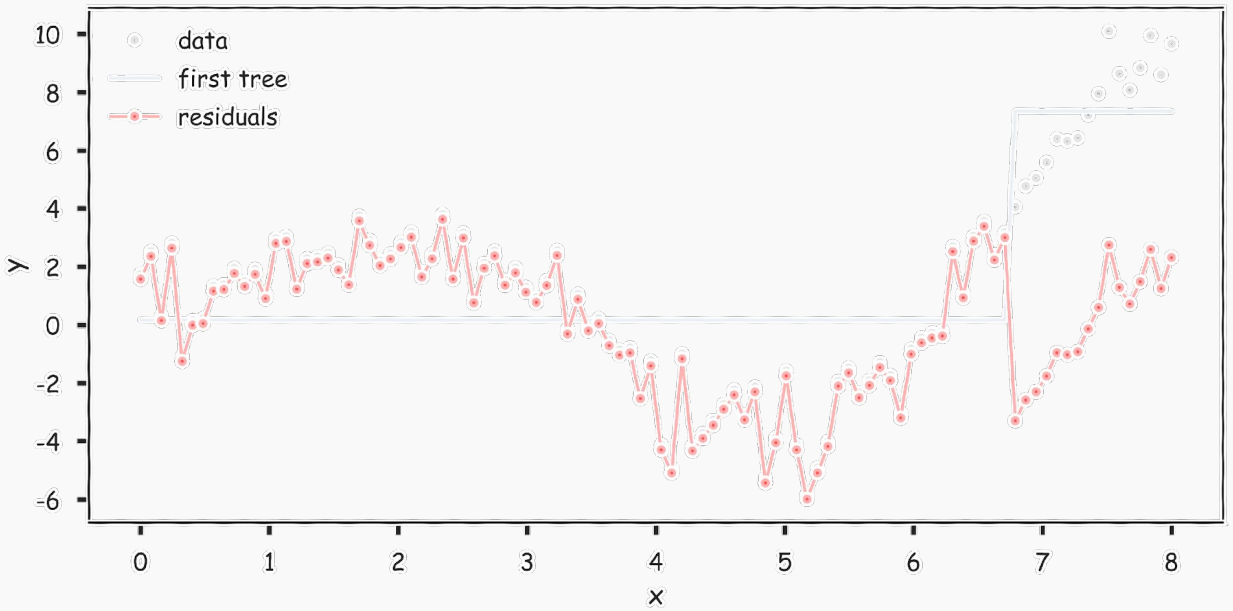
Step 1: Fit a simple model $T^{(0)}$ on the training data: $\{(x_1, y_1), \ldots, (x_N, y_N)\}$.

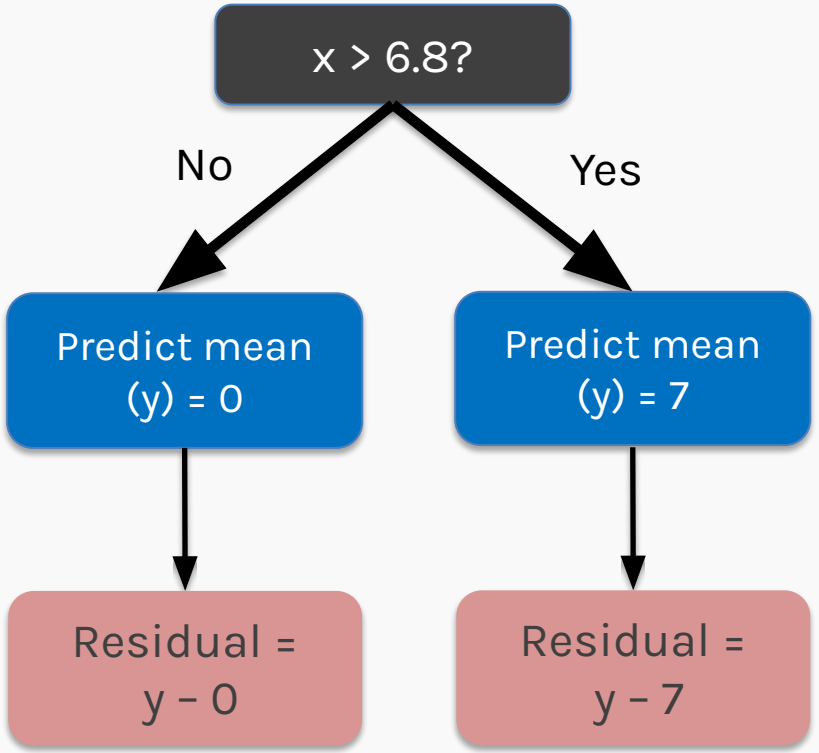




Gradient Boosting: Illustration

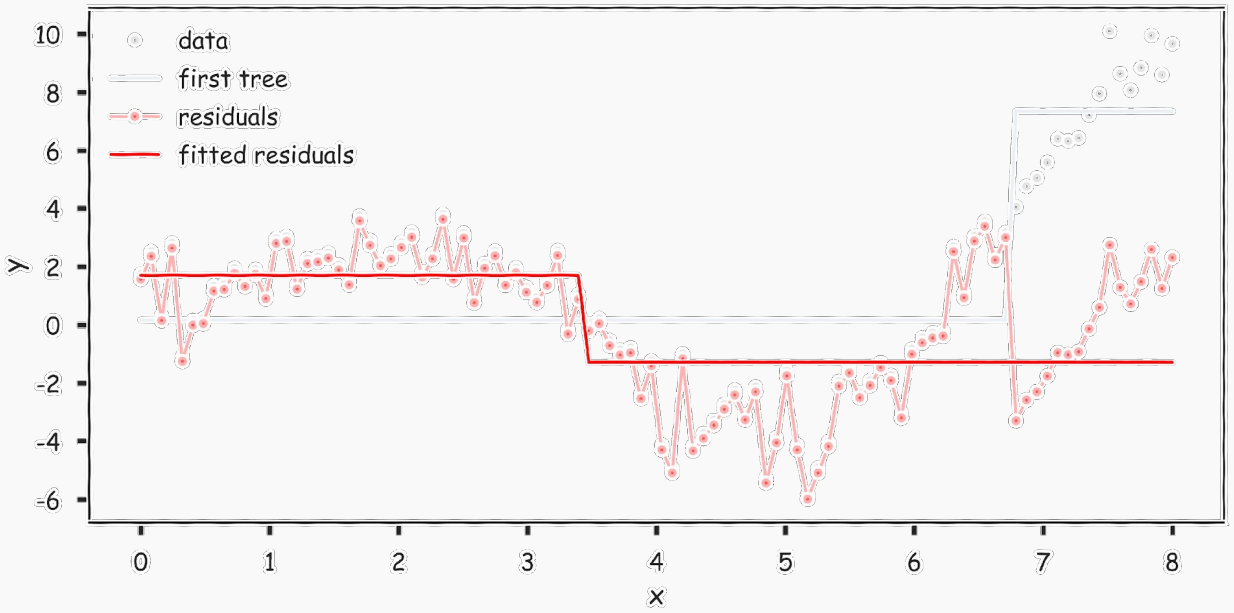
Step 2: Compute the residuals $\{r_1, \ldots, r_N\}$ for $T^{(0)}$. Set $T \leftarrow T^{(0)}$.

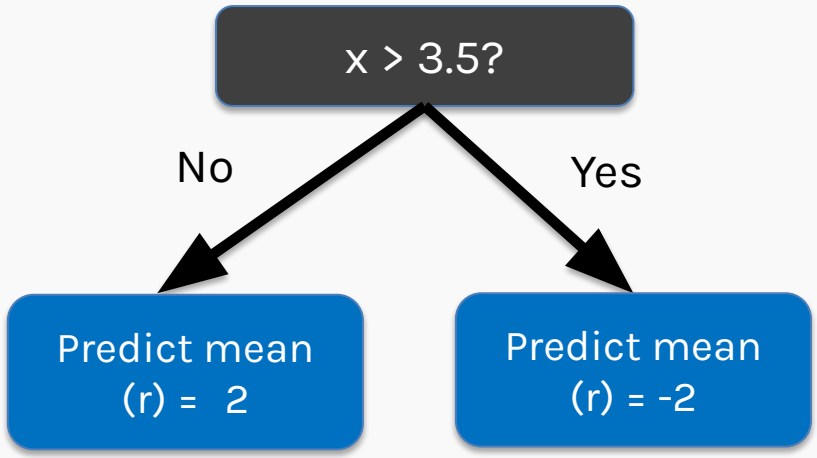




Gradient Boosting: Illustration

Step 3: Fit another model $T^{(1)}$ on: $\{(x_1, r_1), \ldots, (x_N, r_N)\}$.

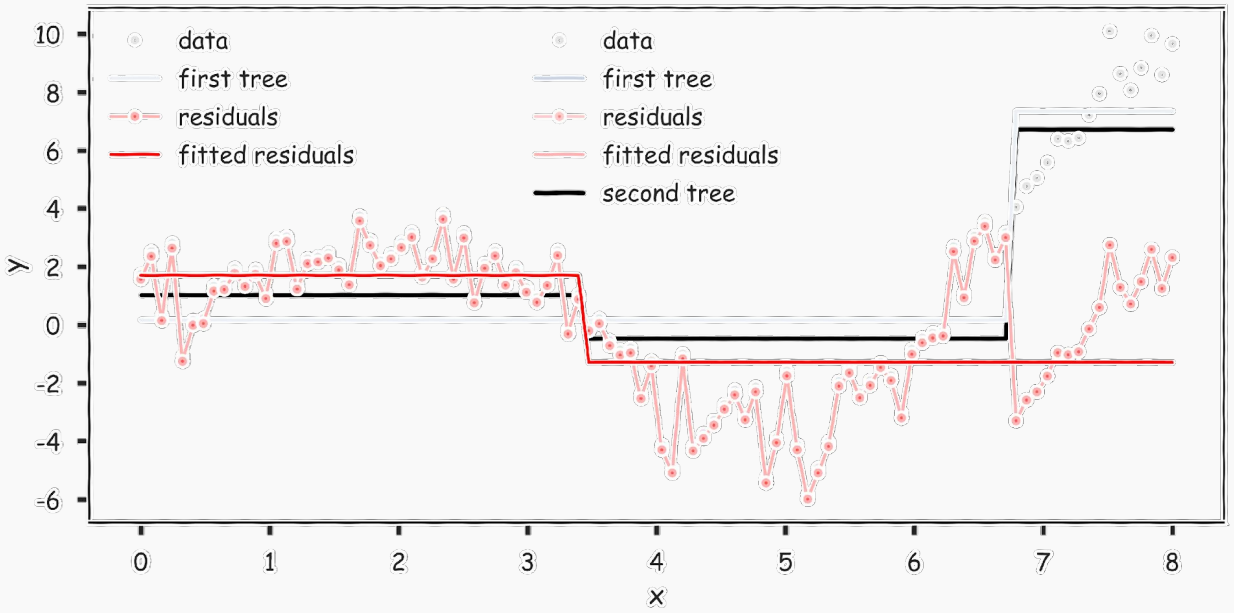


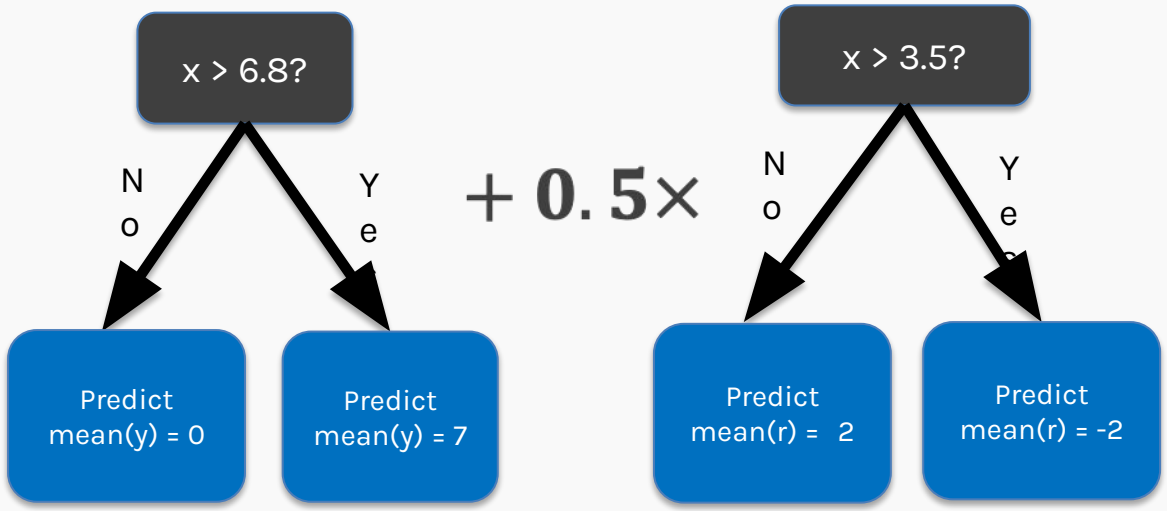


Gradient Boosting: Illustration

Step 4: Combine the two trees in step 1 and 3 by setting $T \leftarrow T + \lambda T^{(1)}$.

Assume $\lambda = 0.5$.

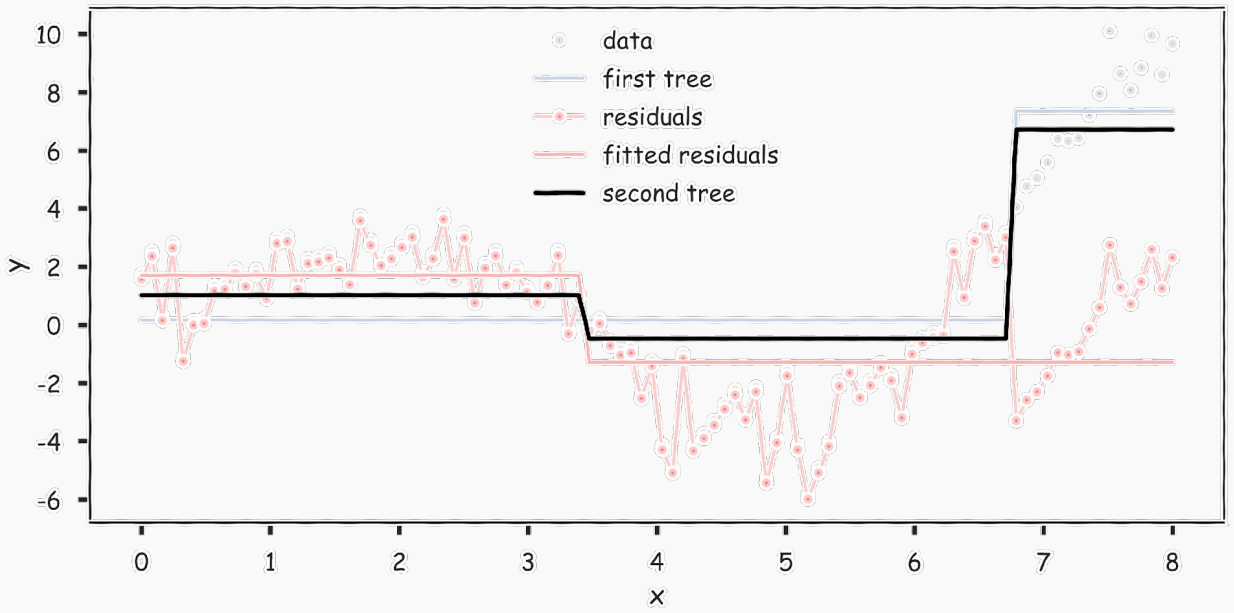


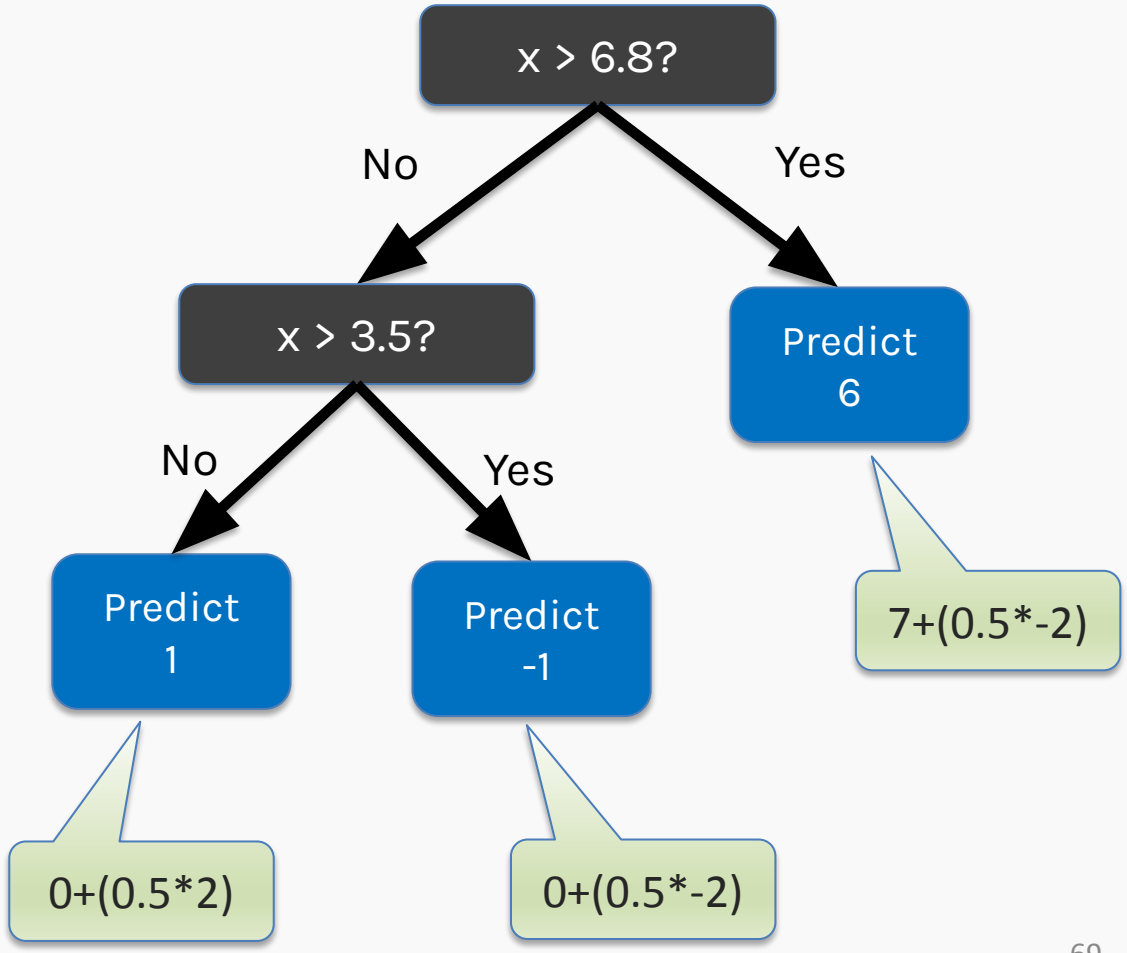


Gradient Boosting: Illustration

Step 4: Combine the two trees in step 1 and 3 by setting $T \leftarrow T + \lambda T^{(1)}$.

Assume $\lambda = 0.5$.

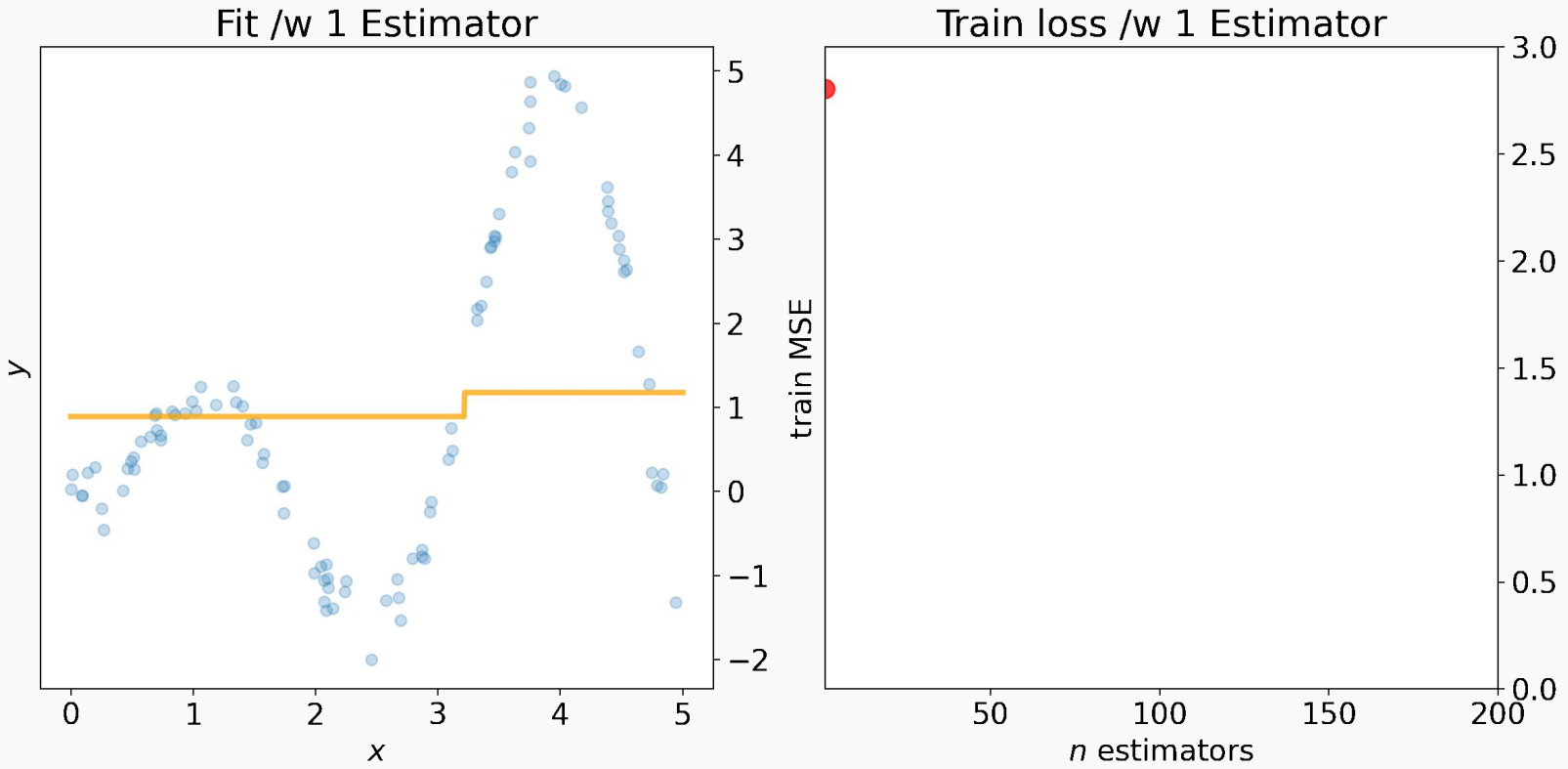




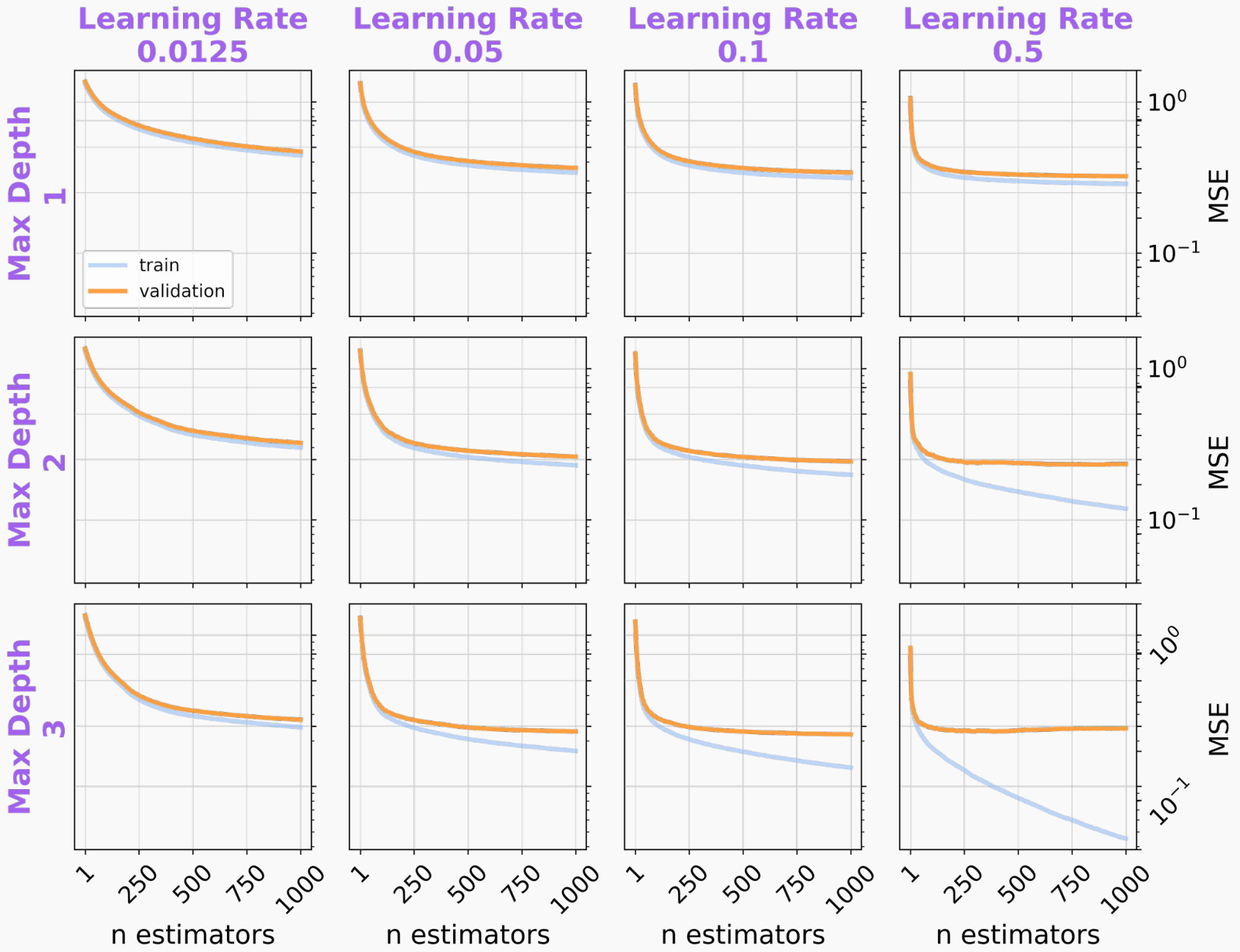
Gradient Boosting – fitting to residuals

Each new estimator is fit to approximate the residuals of the current ensemble.

$$r^{(i)} \leftarrow r^{(i-1)} - T^{(i)}(x)$$

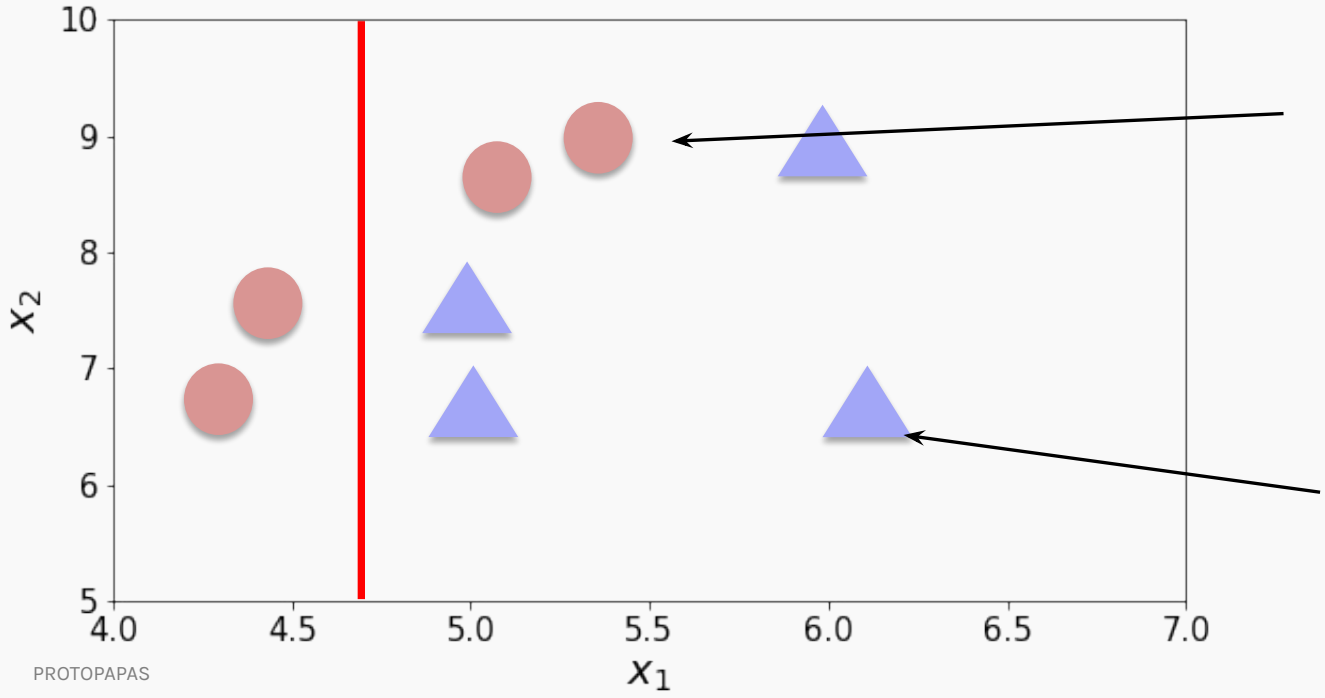


Gradient Boosting – hyperparameters



AdaBoost: Boosting for Classification

Adjust the weights assigned to each data point to ensure the next stump focuses on the points **misclassified by the previous stump**.



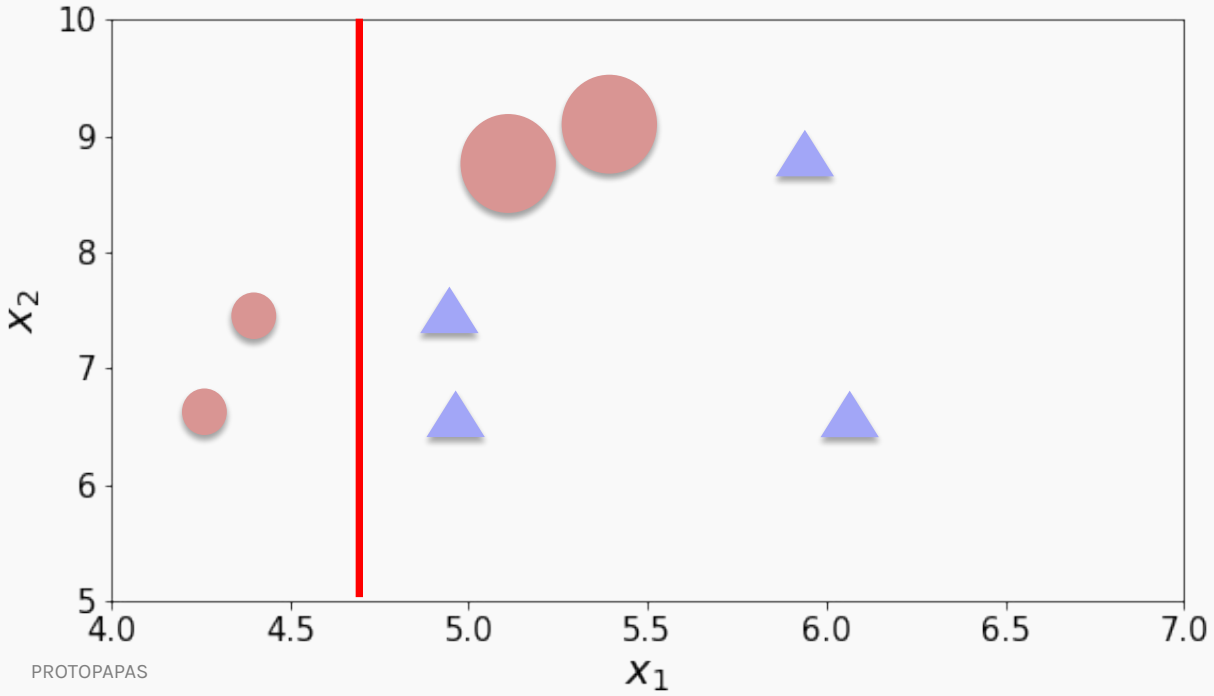
Increase weights of misclassified samples

Decrease weights of correctly classified samples

AdaBoost

Adjust the weights assigned to each data point to ensure the next stump focuses on the points **misclassified by the previous stump**.

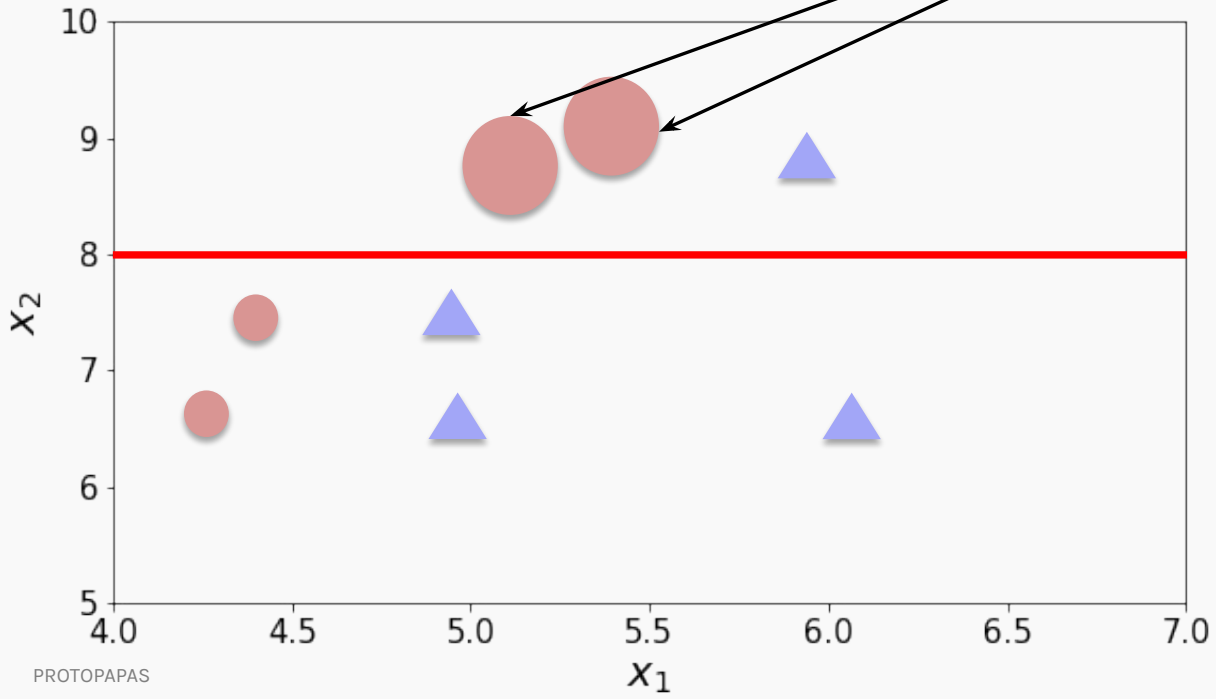
$$w_n^{(1)} \leftarrow \frac{w_n^{(0)} e^{-\lambda\, y_n\, S^{(0)}(x_n)}}{Z} = \frac{w_n^{(1')}}{Z}$$

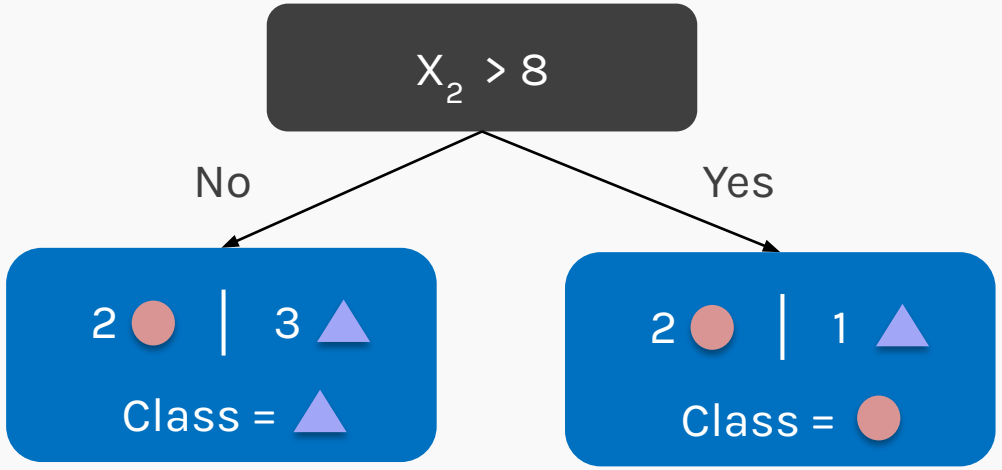


AdaBoost

Create another stump $S^{(1)}$ on the re-weighted data.

Notice that the $S^{(1)}$ has correctly classified the data points that $S^{(0)}$ misclassified.





AdaBoost – hyperparameters