BPS5231

Artificial Intelligence

For

Sustainable Building Design

L3

Ang Yu Qian

Assistant Professor

L03.1
Supervised 3

Recap

The Linear Classifier

Logistic Regression

Confusion Matrix

ROC Curve

Scikit-Learn

L03.2
Supervised 4

Decision Trees

L03.3
Simulations

Rhino

ClimateStudio

L03.1
Supervised 3

Recap

The Linear Classifier

Logistic Regression

Confusion Matrix

ROC Curve

Scikit-Learn

L03.2
Supervised 4

Decision Trees

L03.3
Simulations

Rhino

ClimateStudio

Today, we will cover

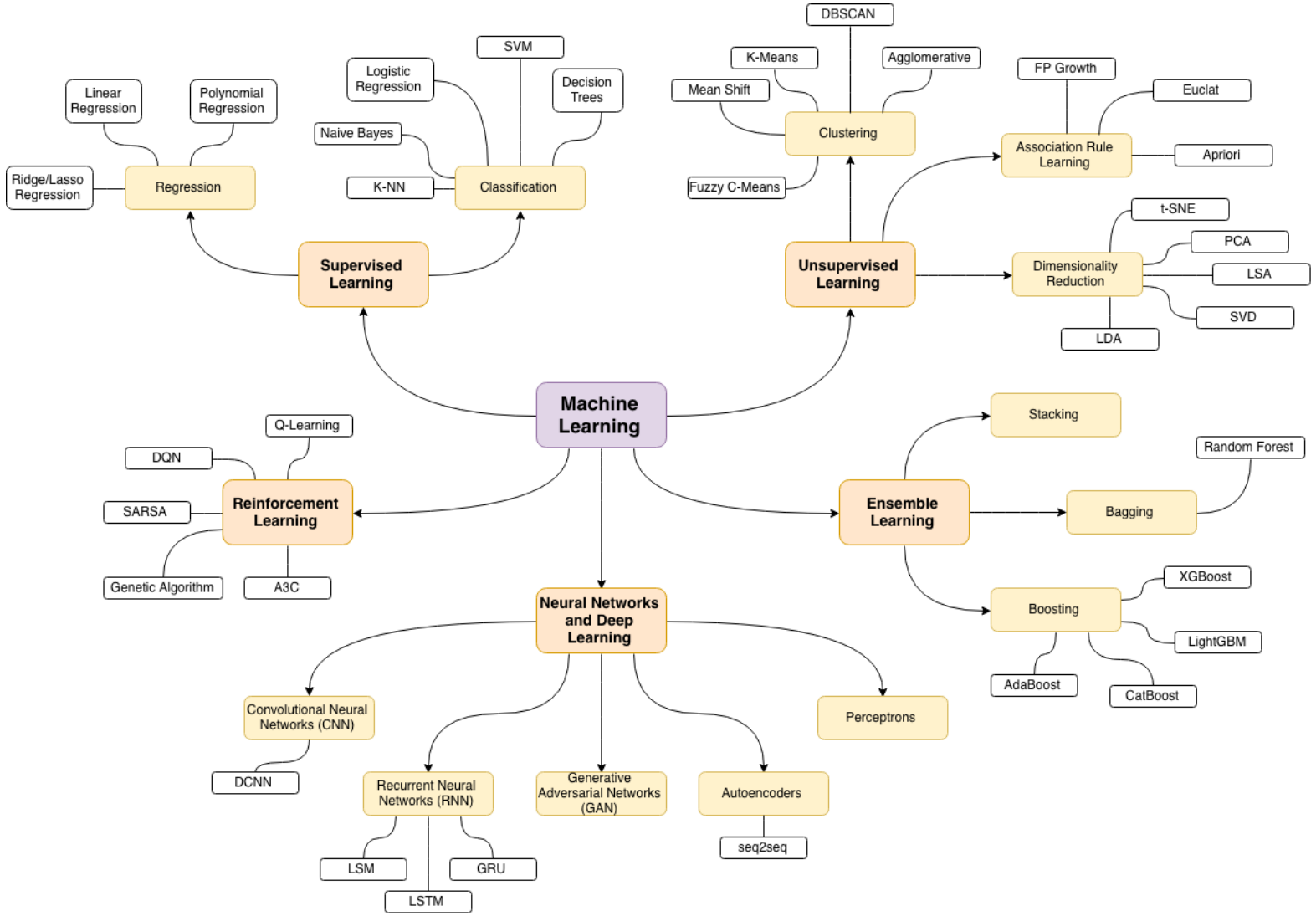


Image from Anil Jain, Google

SUPERVISED 1 SUPERVISE 2 DESIGN SPACE

Today, we will cover

Today →

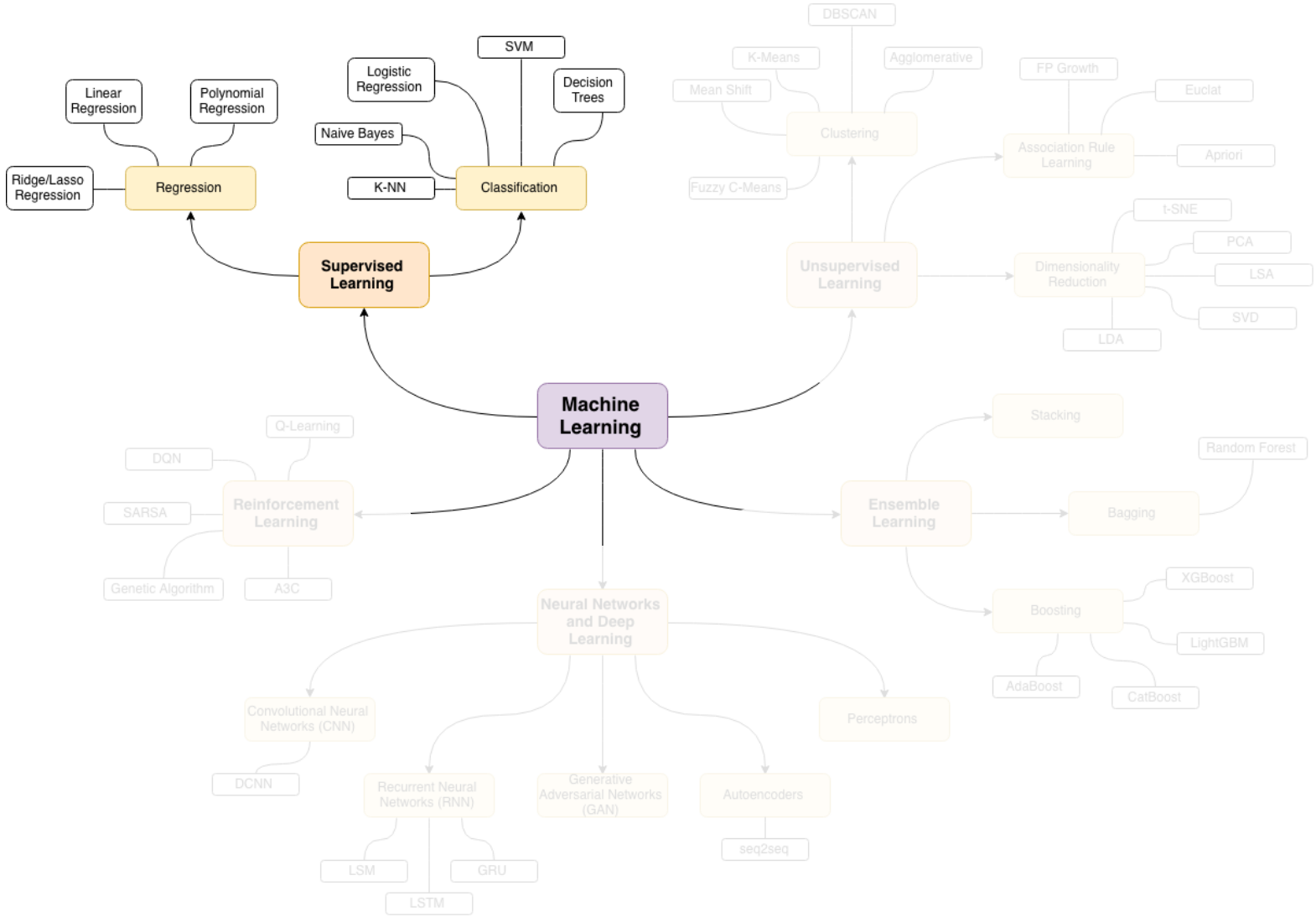
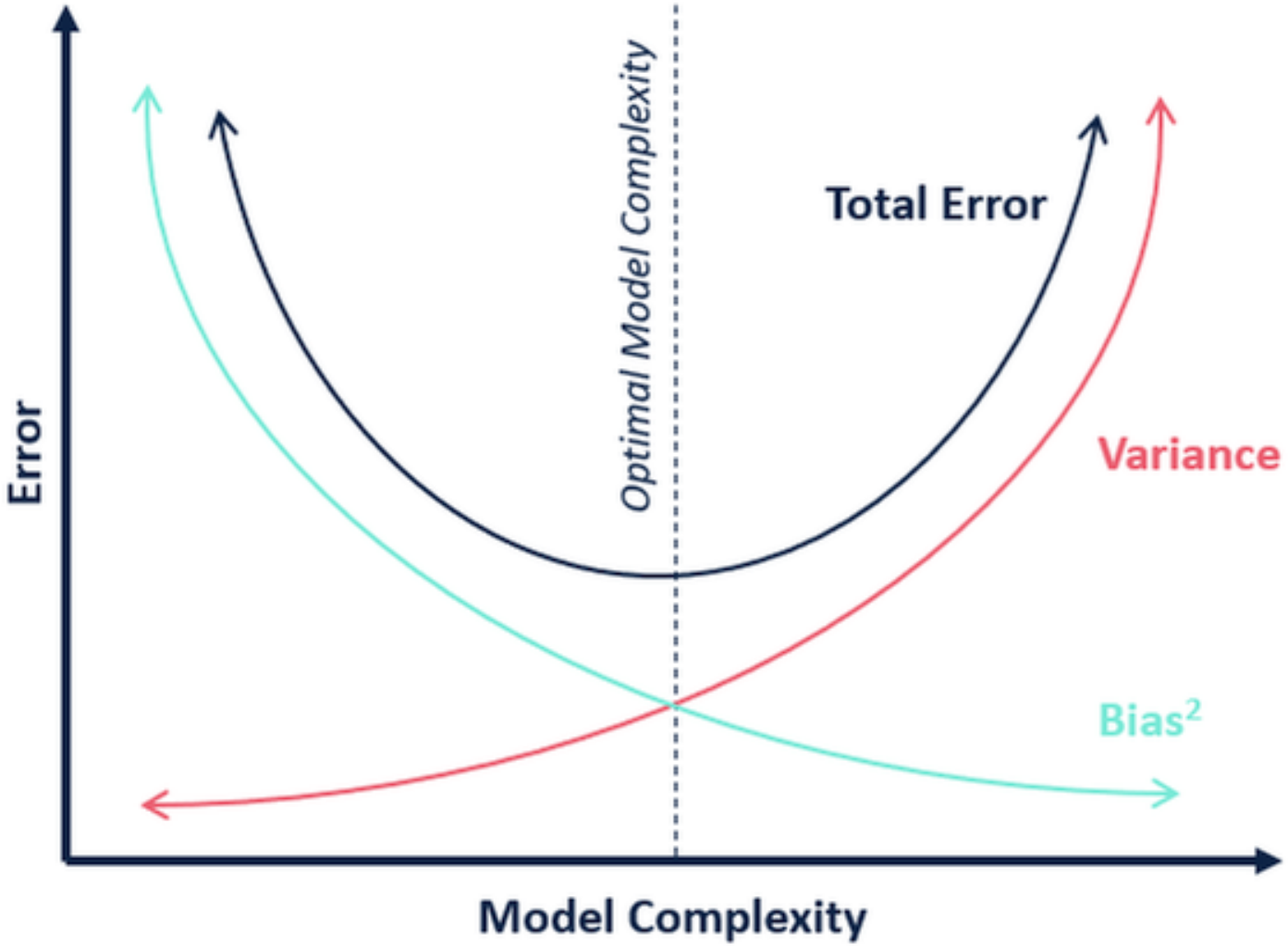


Image from Anil Jain, Google

Recap – Bias-Variance Trade-off



Linear Regression

$$Y = f(X) = \beta_0 + \beta_1 X + \varepsilon$$

Multiple Features/Predictors

$$Y = f(X) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n + \varepsilon$$

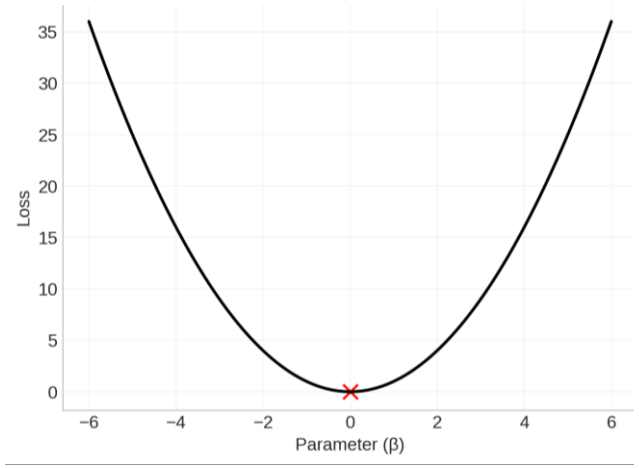
Polynomials

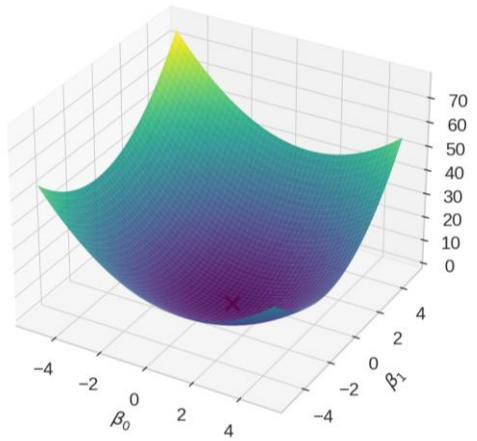
$$Y = f(X) = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3 + \ldots + \beta_d X^d + \varepsilon$$

Complex real-world problems are non-convex (e.g. neural networks)
Non-convex landscape have multiple local minima, saddle points, highly irregular “loss surfaces”
Analytical solutions usually impossible

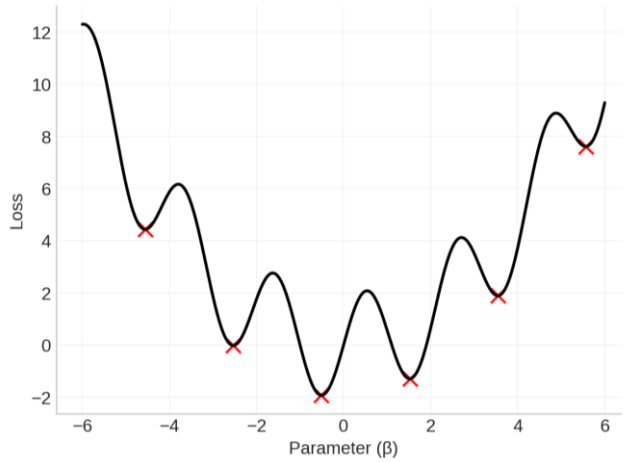
Here, we rely on iterative optimization methods (e.g., gradient descent)

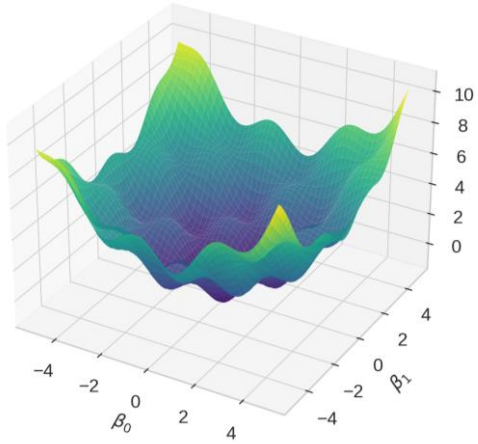
Convex



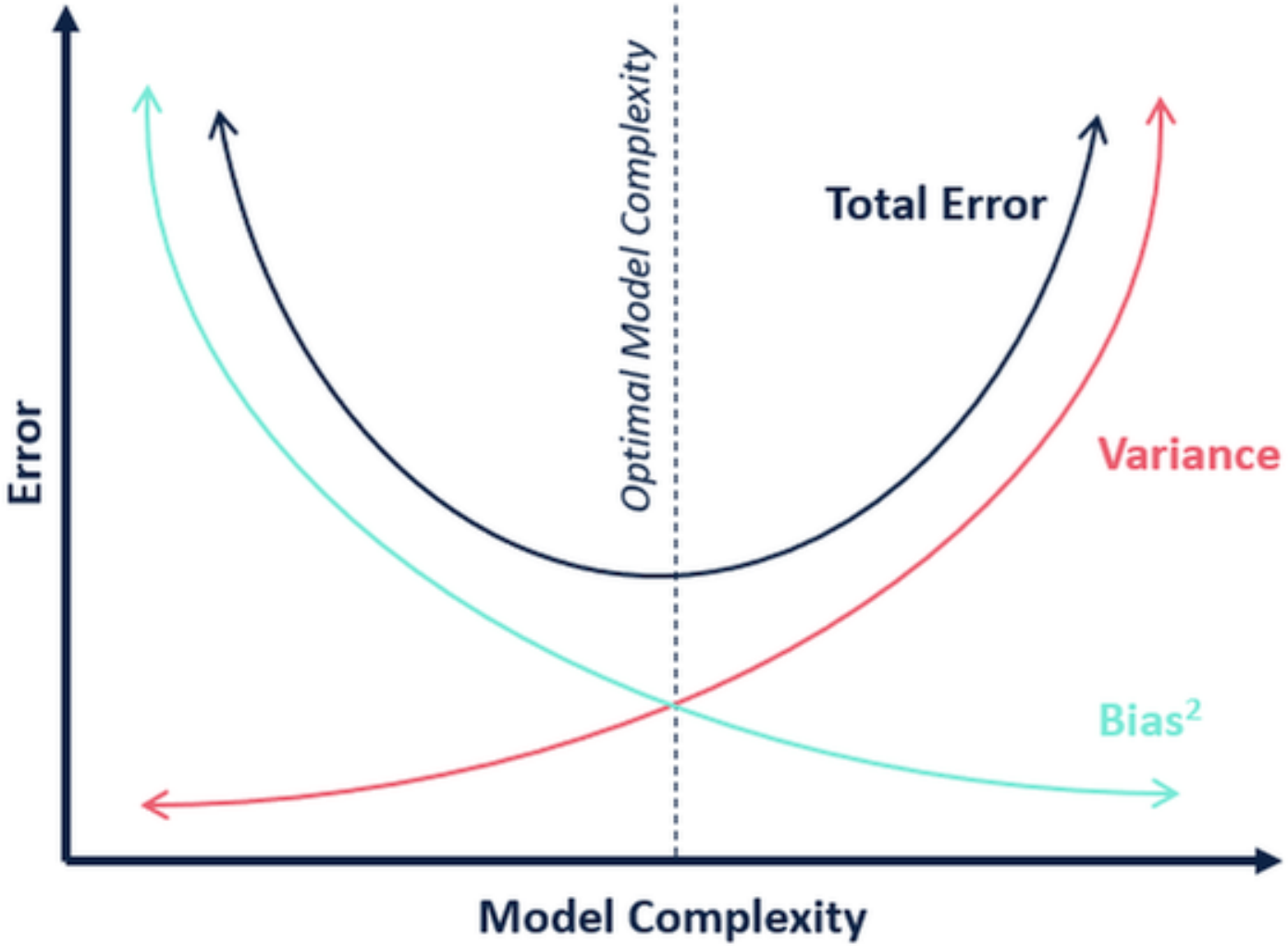


Non-Convex





Recap – Bias-Variance Trade-off



Parametric vs Non-parametric

Parametric Methods: They reduce the problem of estimating f(x) to finding a finite set of parameters
Example: In linear regression, we only need to find the intercept and the slope

Approach:
(1) Make some assumptions about the functional form of f (e.g., linear, quadratic, or …),
(2) Estimate the required parameters using training data

Nonparametric Methods: No assumption about the functional form of f

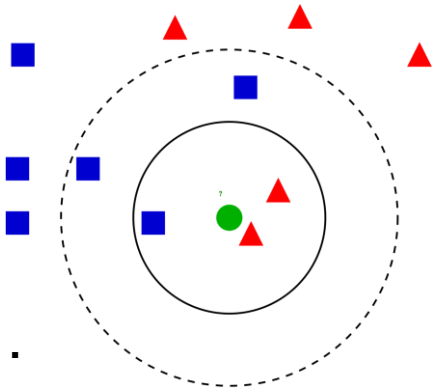
Advantage: They can fit a variety of different shapes of f to data

Disadvantages:
(1) Some harder than parametric methods,
(2) Require a good amount of data to accurately estimate f

K-Nearest Neighbors (KNN)

Idea: To predict the output for a new data point:

- Look at the K closest neighbors in the training data (using a distance metric, usually Euclidean).
- Aggregate their values:
  - Regression: average their target values.
  - Classification: majority vote of their labels.



Key property: No "training" in the classical sense – just storing the dataset.

Prediction = similarity-based lookup.

Intuition: *"Tell me who your neighbors are, and I'll tell you who you are."*

The Linear Classifier

$$h(x;\theta,\theta_0) = \text{sign}(\theta^T x + \theta_0) = \begin{cases} +1 & \text{if } \theta^T x + \theta_0 > 0 \\ -1 & \text{otherwise} \end{cases}$$

We can think of θ, θ_0 as specifying a hyperplane. It divides the space our points live in, into two half-spaces. The one that is on the same side as the normal vector is the *positive* half-space, and we classify all points in that space as *positive*. The half-space on the other side is *negative*, and all points in it are classified as *negative*.

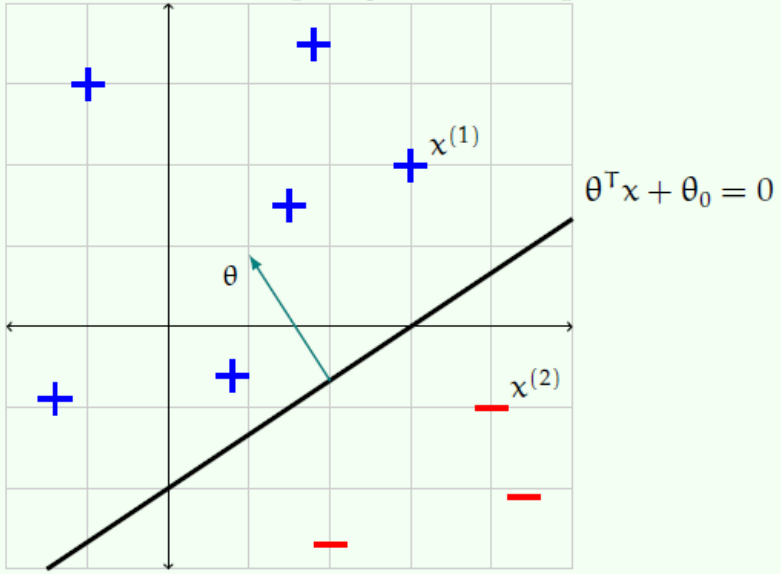
Example: Let h be the linear classifier defined by $\theta = \begin{bmatrix} -1 \\ 1.5 \end{bmatrix}, \theta_0 = 3$.

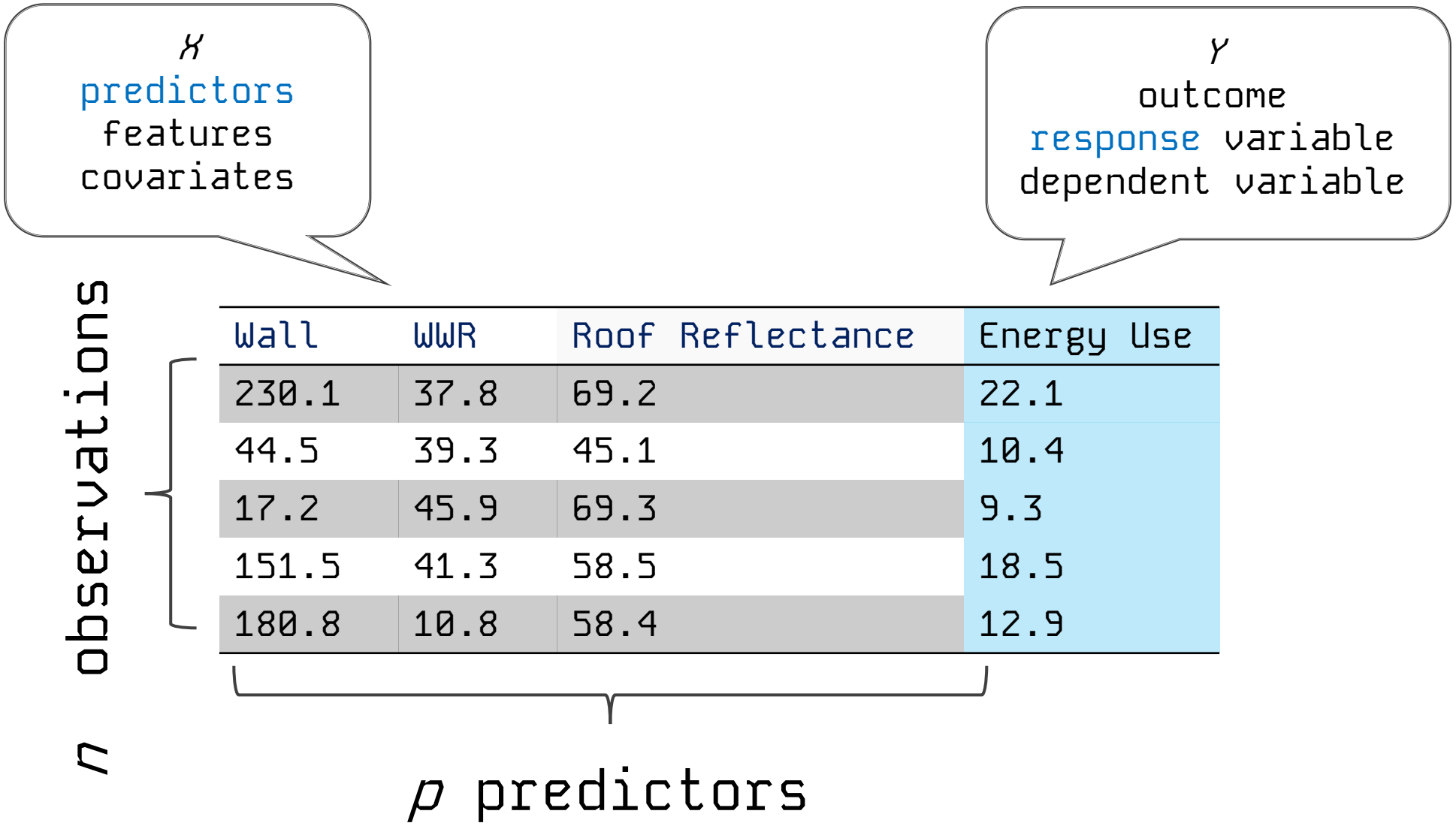
The diagram below shows several points classified by h. In particular, let $x^{(1)} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$ and $x^{(2)} = \begin{bmatrix} 4 \\ -1 \end{bmatrix}$.

$$h(x^{(1)}; \theta, \theta_0) = \text{sign}\left(\begin{bmatrix} -1 & 1.5 \end{bmatrix} \begin{bmatrix} 3 \\ 2 \end{bmatrix} + 3\right) = \text{sign}(3) = +1$$

$$h(x^{(2)}; \theta, \theta_0) = \text{sign}\left(\begin{bmatrix} -1 & 1.5 \end{bmatrix} \begin{bmatrix} 4 \\ -1 \end{bmatrix} + 3\right) = \text{sign}(-2.5) = -1$$

Thus, $x^{(1)}$ and $x^{(2)}$ are given positive and negative classfications, respectively.





Wall	WWR	Roof Reflectance	Energy Use
230.1	37.8	69.2	22.1
44.5	39.3	45.1	10.4
17.2	45.9	69.3	9.3
151.5	41.3	58.5	18.5
180.8	10.8	58.4	12.9

Logistic Regression

response variable *Y* is Yes/No

Age	Type	HVAC Type	Set Point	Wall Insulation	Smart Meter	WWR	Shading	Efficient
63	1	Split	24	2.5	Yes	25	Yes	No
67	1	Central	22	1.8	No	45	No	No
67	1	Central	25	3.2	Yes	30	Yes	No
37	1	District	25	2.0	Yes	55	No	Yes
41	0	District	23	1.5	No	35	No	Yes

Logistic Regression

These data contain a binary outcome of building dwelling units.
An outcome value of:

- *Yes* indicates efficiency (crossed a predefined efficiency threshold)
- *No* means inefficient

There are 8 predictors including:

- Building age
- Shading
- Window-to-wall ratio
- Set point
- HVAC system

and others

Logistic Regression

Up to this point, the methods we have seen have focused on modeling and predicting a quantitative response variable (for example, total building energy use in kWh, peak cooling load, or annual electricity consumption).

Linear regression (and its variants like Ridge and LASSO) perform well in these cases.

However, when the response variable is categorical (for example, whether a building is energy efficient or inefficient), the problem is no longer a regression problem, but instead a classification problem.

The goal in classification is to assign each building (an observation) into a category (efficient vs. inefficient) defined by the response variable Y, based on a set of predictor variables X (such as building age, HVAC type, window-to-wall ratio, etc.).

Why not linear regression?

Consider the following:

$$Y = \begin{cases} 1 \text{ if BPS student (BPS)} \\ 2 \text{ if Computer Science (CS)} \\ 3 \text{ otherwise} \end{cases}$$

The model would imply a specific ordering of the outcome, and would treat a one-unit change in y equivalent. The jump from $y = 1$ to $y = 2$ (BPS to CS) should not be interpreted as the same as a jump from $y = 2$ to $y = 3$ (CS to others).

Similarly, the response variable could be reordered such that $y = 1$ represents CS and $y = 2$ represents BPS, and then the model estimates and predictions would be fundamentally different.

If the categorical response variable was *ordinal* (had a natural ordering, like class year, Freshman, Sophomore, etc.), then a linear regression model would make some sense, but is still not ideal.

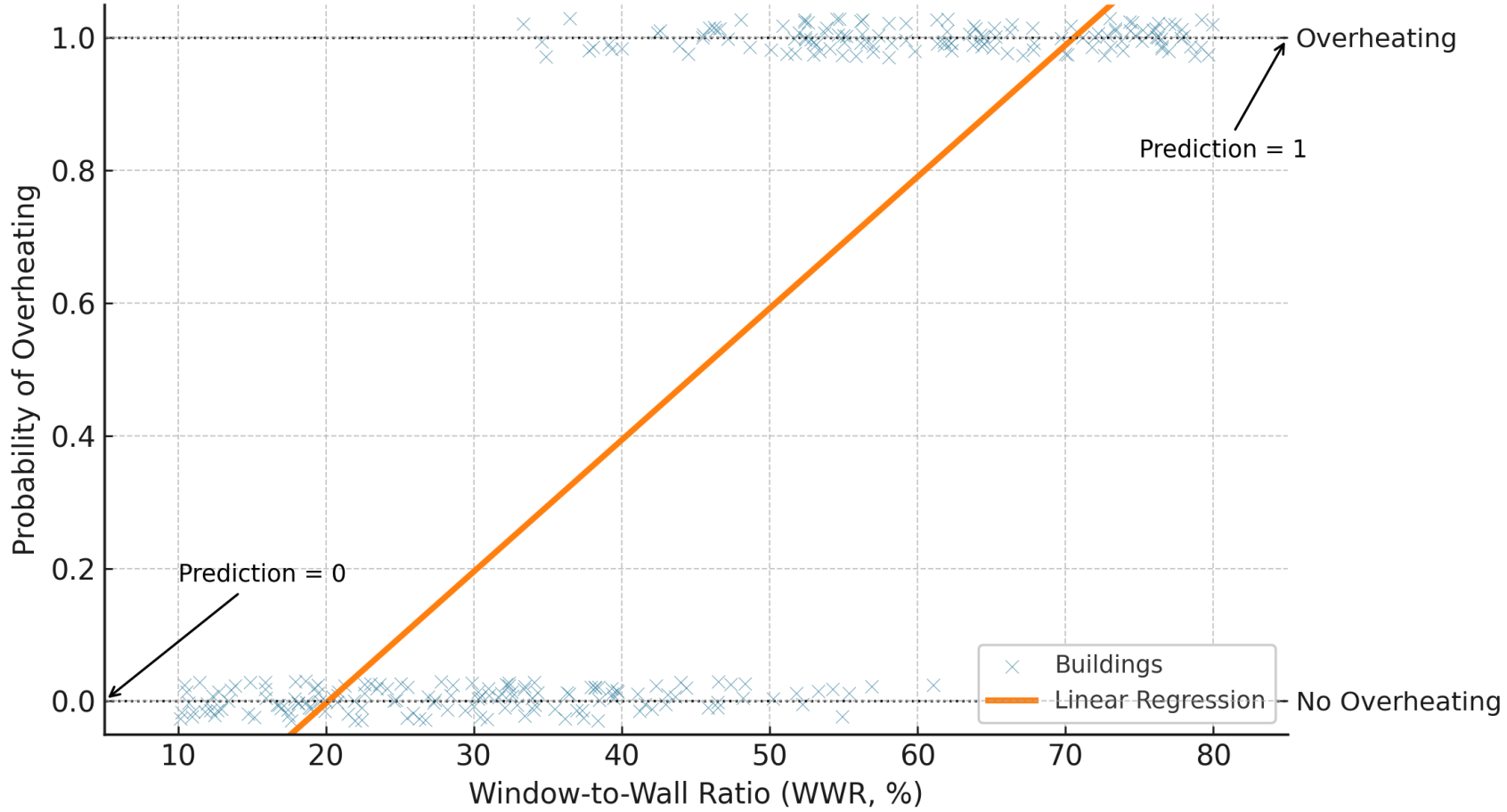
Logistic Regression

The simplest form of classification is when the response variable y has only two categories, and then an ordering of the categories is natural. For example, a NUS student could be categorized as (note, the $y = 0$ category is a "catch-all" so it would involve both living on campus students and those who live in other situations: off campus, etc):

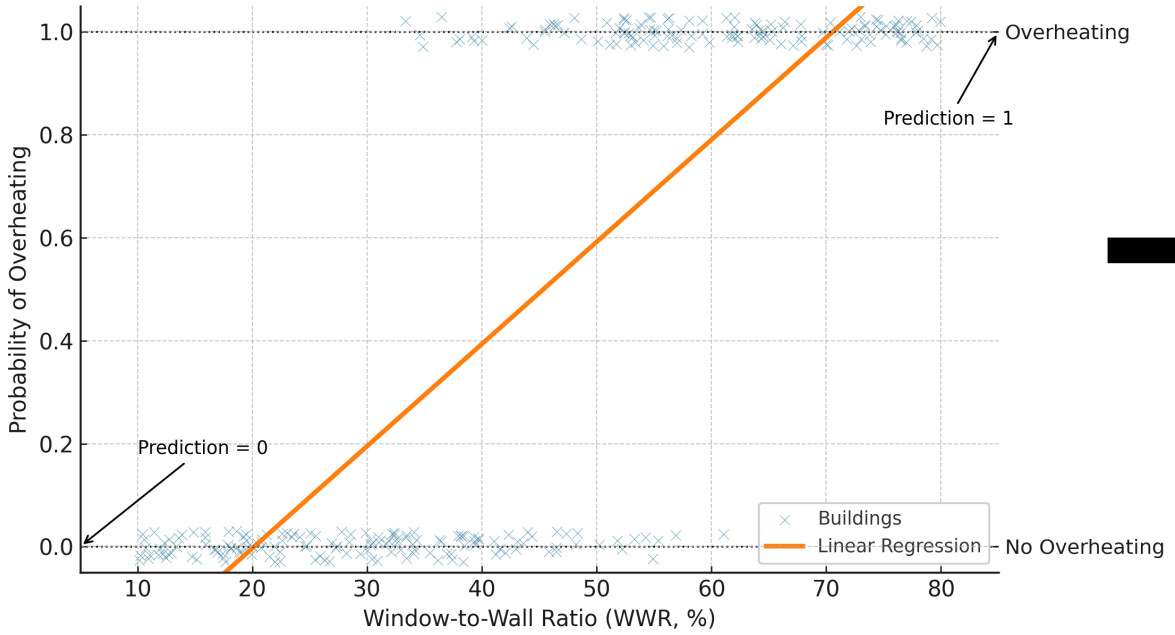
$$Y = \begin{cases} 1 \text{ if lives on campus} \\ 2 \text{ otherwise} \end{cases}$$

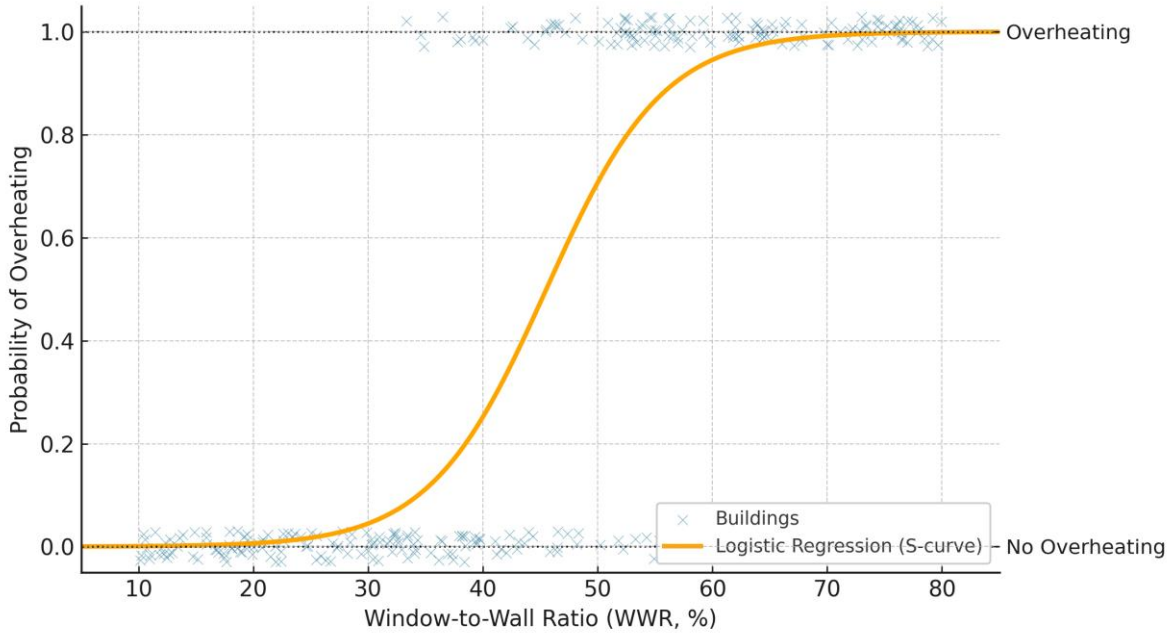
Linear regression could be used to predict y directly from a set of covariates (like gender, major, GPA, etc.), and if $\hat{y} \geq 0.5$, we could predict the student lives on campus and predict other houses if $\hat{y} < 0.5$.

What could go wrong with this linear regression model?



Think of a function that would do this for us:





Logistic Regression addresses the problem of estimating a probability, $P(y = 1)$, to be outside the range of [0,1]. The logistic regression model uses a function, called the *logistic* function, to model $P(y = 1)$:

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

Remember this!

As a result the model will predict $P(y = 1)$ with an S-shaped curve, which is the general shape of the logistic function.

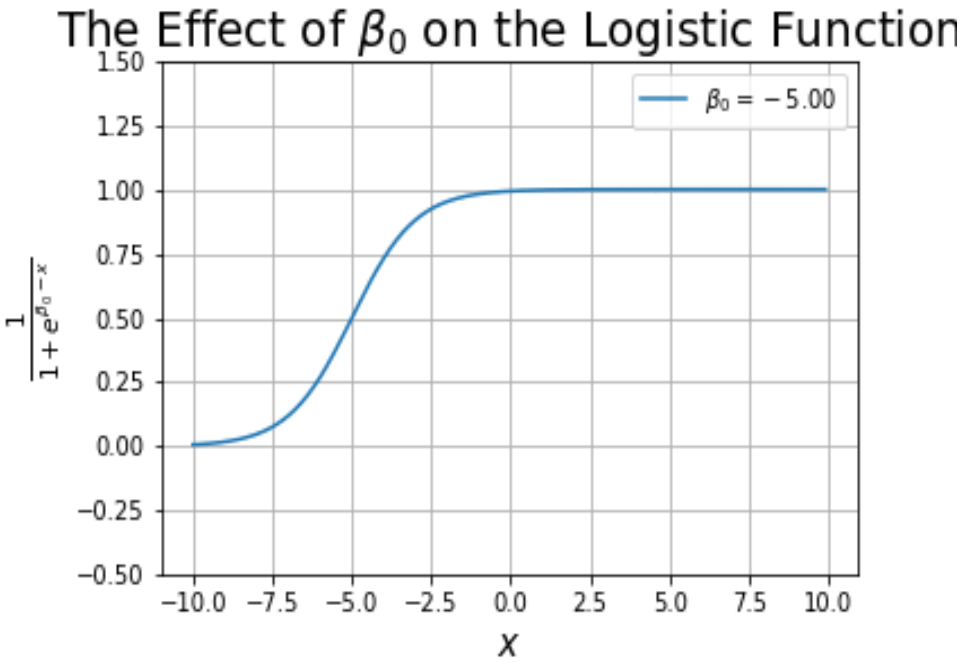
β_0 shifts the curve right or left by $c = -\frac{\beta_0}{\beta_1}$.

β_1 controls how steep the S-shaped curve is. Distance from ½ to almost 1 or ½ to almost 0 to ½ is $\frac{2}{\beta_1}$

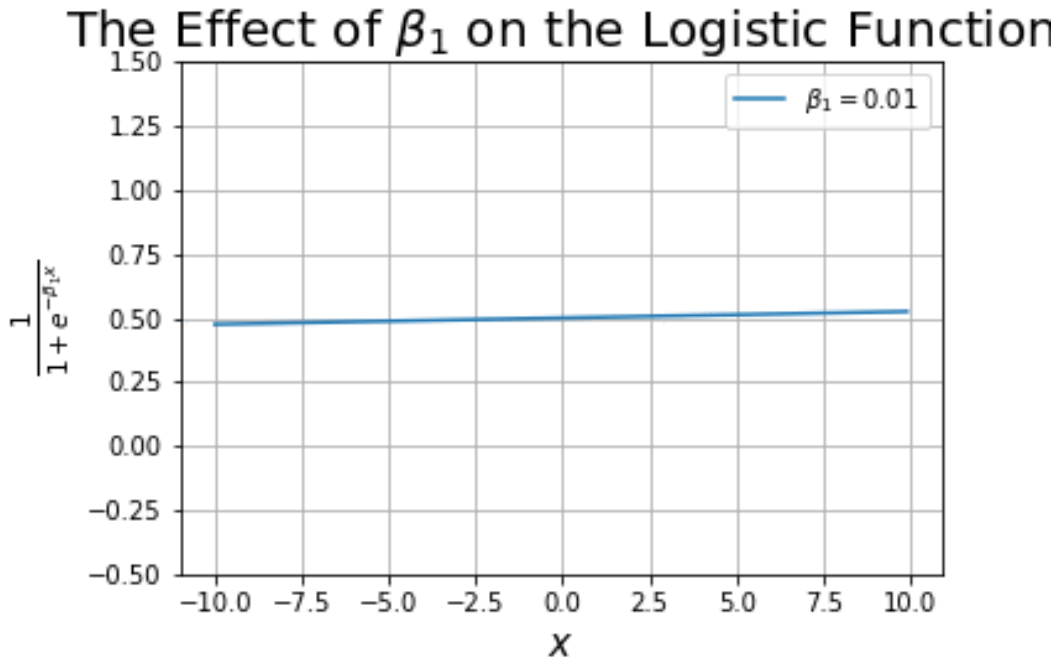
Note: if β_1 is positive, then the predicted $P(y = 1)$ goes from zero for small values of X to one for large values of X and if β_1 is negative, then the $P(y = 1)$ has opposite association.

Logistic Regression

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



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



There are 2 major types of error in classification problems based on a binary outcome. They are:

False positives: incorrectly predicting $\hat{Y} = 1$ when it truly is in $Y = 0$.
False negative: incorrectly predicting $\hat{Y} = 0$ when it truly is in $Y = 1$.

The results of a classification algorithm are often summarized in two ways:

1. a confusion matrix, sometimes called a contingency table
2. a receiver operating characteristics (ROC) curve.

Classification Performance – Confusion Matrix

When a classification algorithm (like logistic regression) is used, the results can be summarize in a (2 x 2) table as such:

	Predicted No Overheating ($\hat{Y} = 0$)	Predicted Truly Overheating ($\hat{Y} = 1$)
Truly No Overheating ($Y = 0$)	132	32
Truly Overheating ($Y = 1$)	56	83

The table above was a classification based on a logistic regression model to predict overheating.

What are the false positive and false negative rates for this classifier?

Precision

Of all predicted positives, how many are actually positive?

$$\text{Precision} = \frac{TP}{TP + FP}$$

High precision = few false alarms

Recall

Of all actual positives, how many did we correctly identify?

$$\text{Recall} = \frac{TP}{TP + FN}$$

High recall = few missed cases

F1 Score

Harmonic mean of Precision & Recall.
Balances the two when both matter.

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Good for imbalanced datasets

ROC Curves

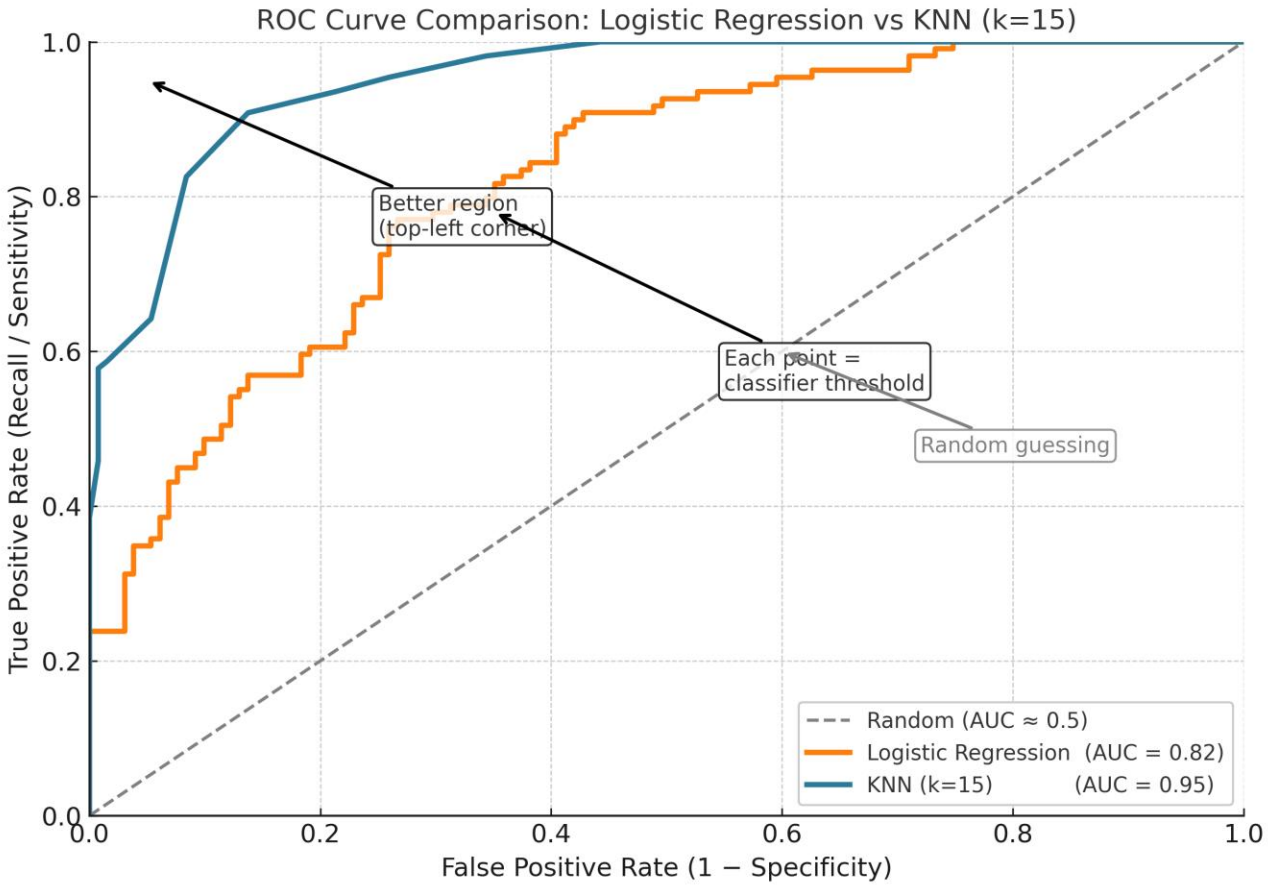
The Radio Operator Characteristics (ROC) curve illustrates the trade-off for all possible thresholds chosen for the two types of error (or correct classification).

The vertical axis displays the true positive rate, and the horizontal axis depicts the false positive rate.

What is the shape of an ideal ROC curve?

See next slide for an example.

ROC Curves



ROC Curve = trade-off between True Positive Rate (Sensitivity) and False Positive Rate (1 − Sensitivity)

Diagonal line = random guess (AUC ≈ 0.5). Any useful model should be above this

Top-left corner = ideal region → high sensitivity, low false positive

AUC (Area Under Curve) is a single number summary: closer to 1.0 = better

Scikit-Learn

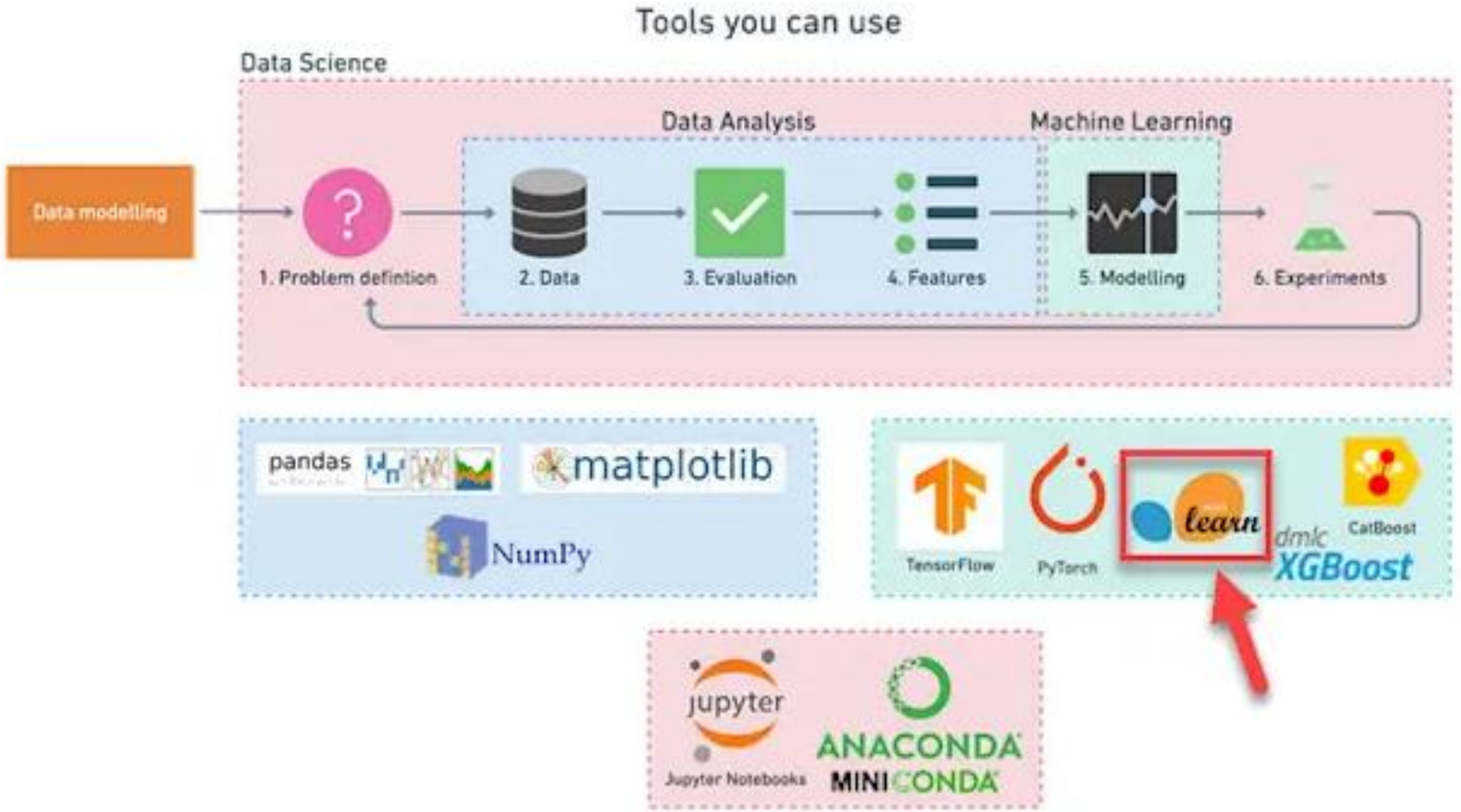
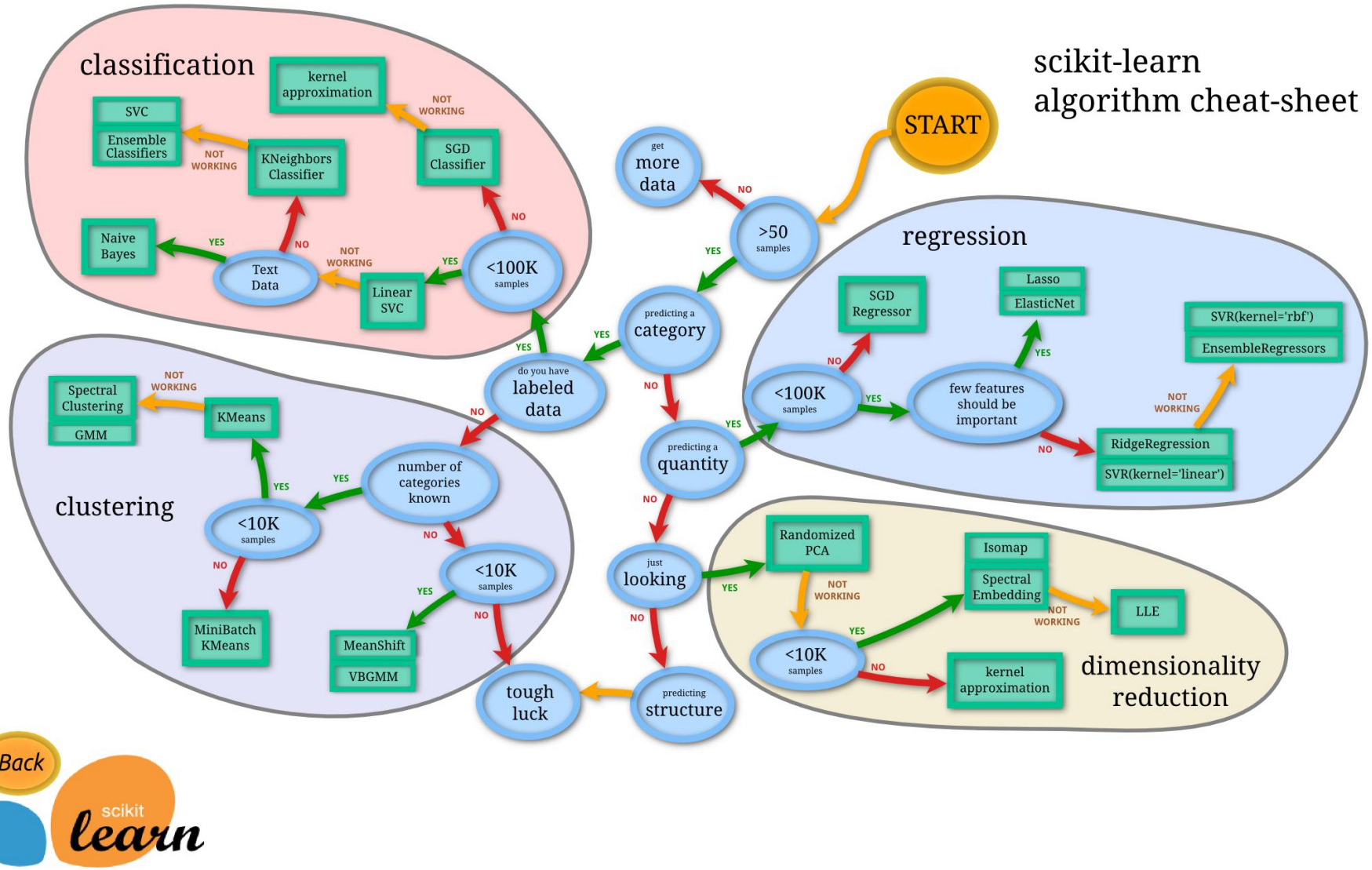


Image from ZTM

Scikit-Learn

Image from ZTM



SUPERVISED 1 SUPERVISE 2 DESIGN SPACE

Google Colab
https://bit.ly/BPS5231-L3

L03.1
Supervised 3

Recap

The Linear Classifier

Logistic Regression

Confusion Matrix

ROC Curve

Scikit-Learn

L03.2
Supervised 4

Decision Trees

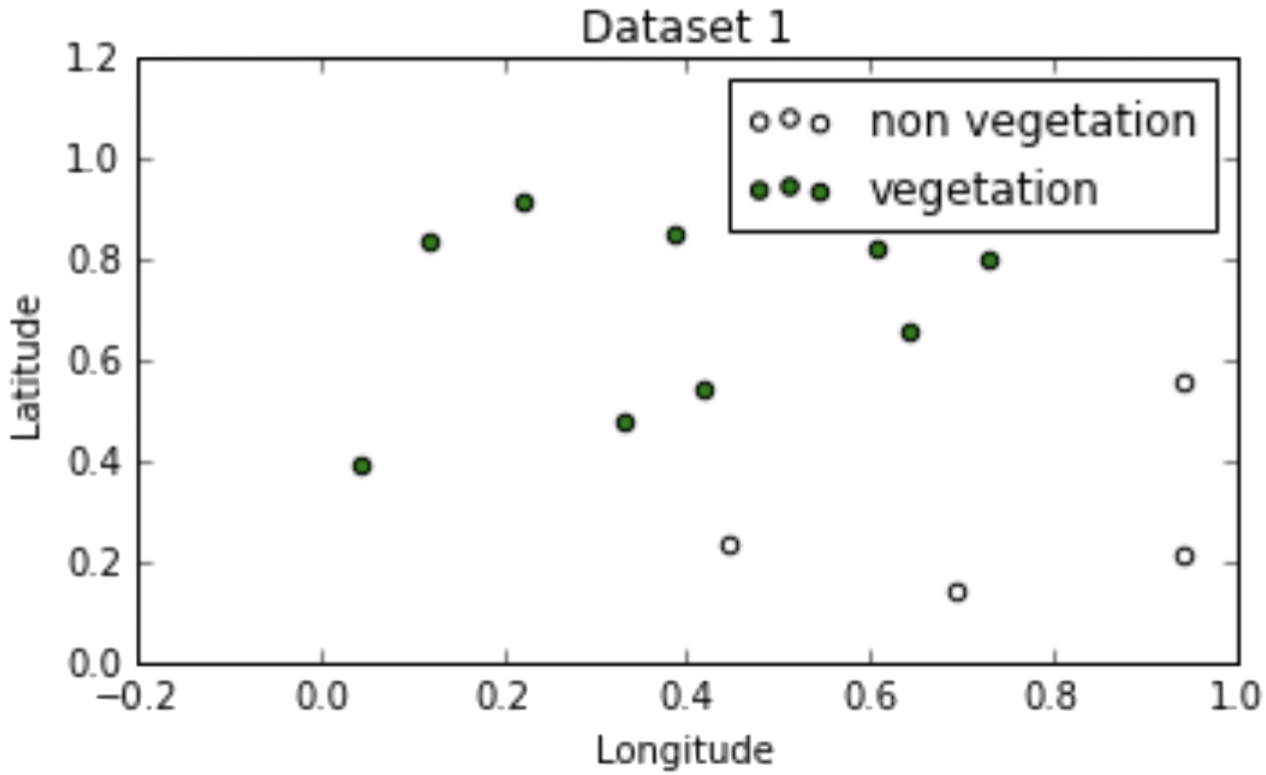
L03.3
Simulations

Rhino

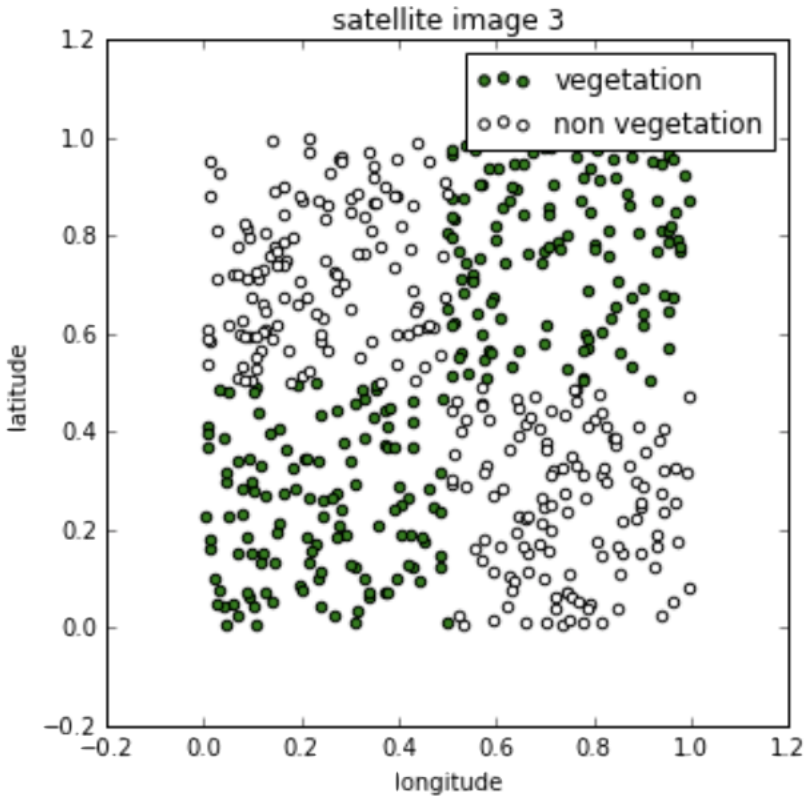
ClimateStudio

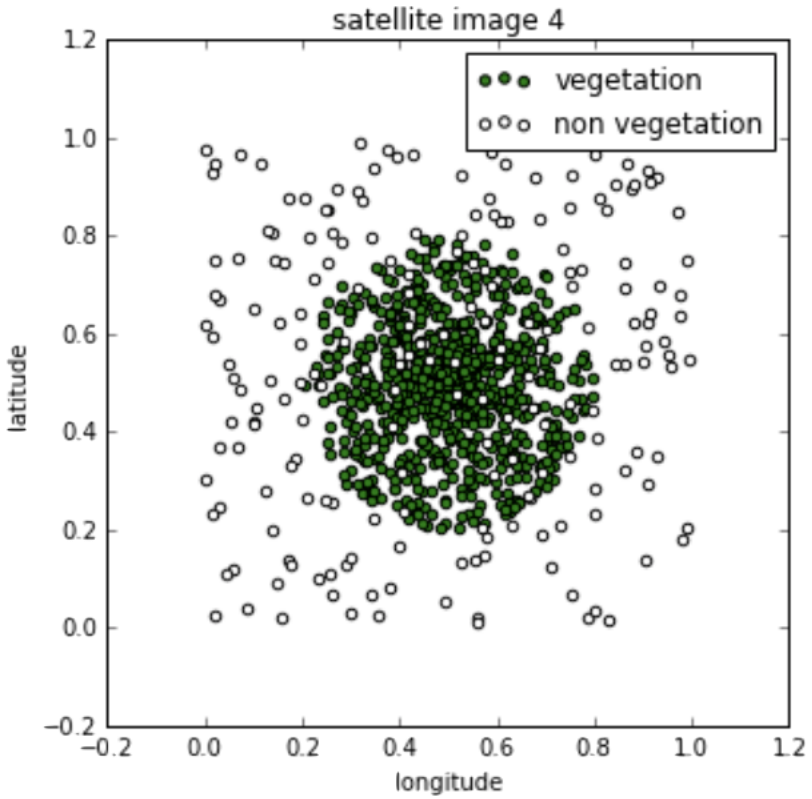
logistic regression for building classification boundaries works best when:

the classes are well-separated in the feature space
have a nice geometry to the classification boundary)

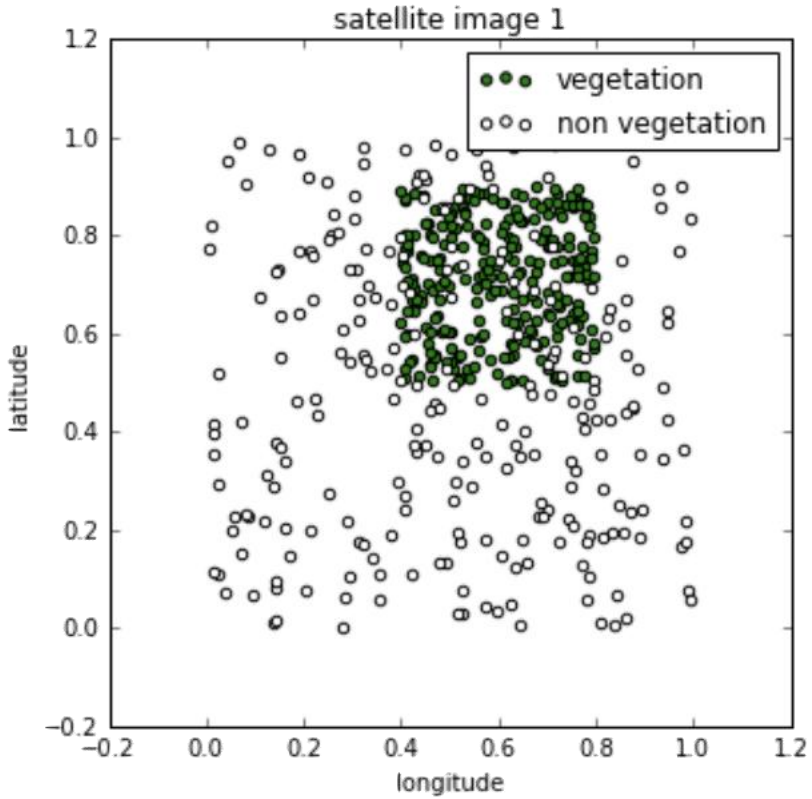


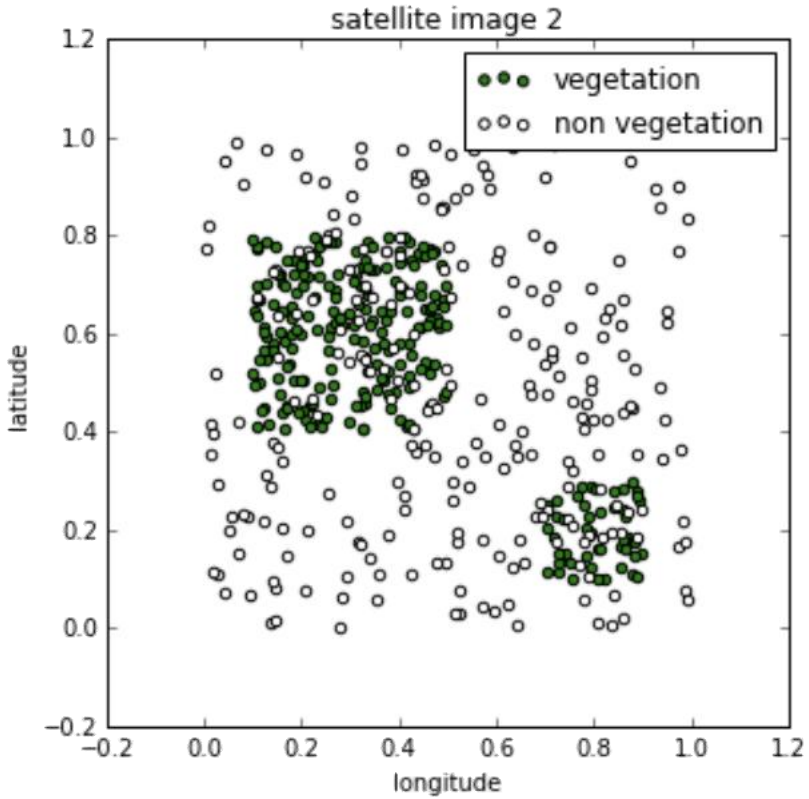
How about these?



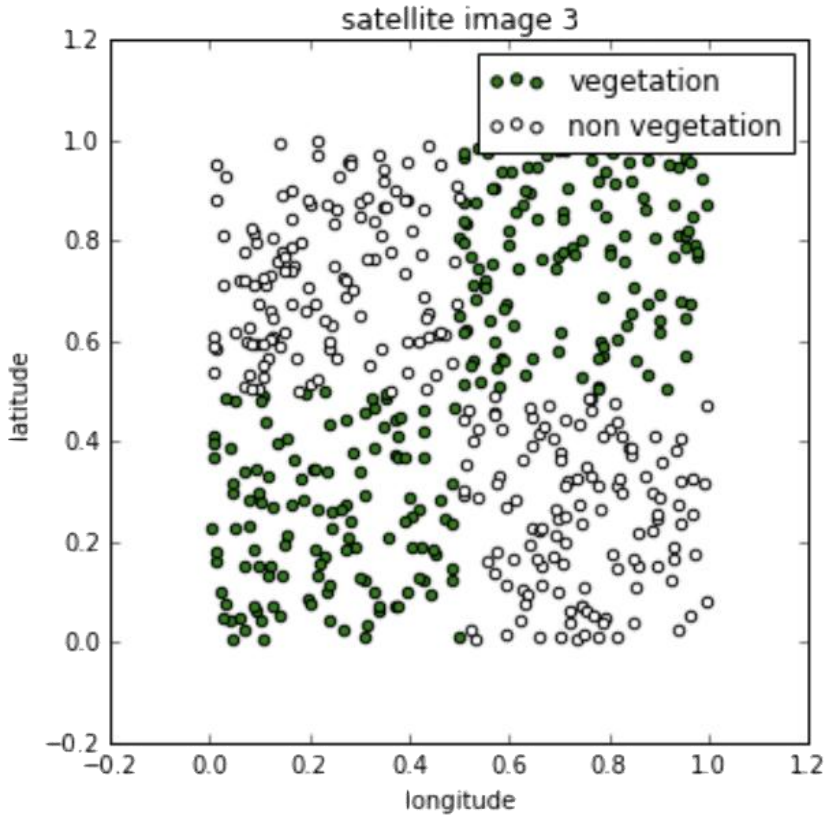


Or these?

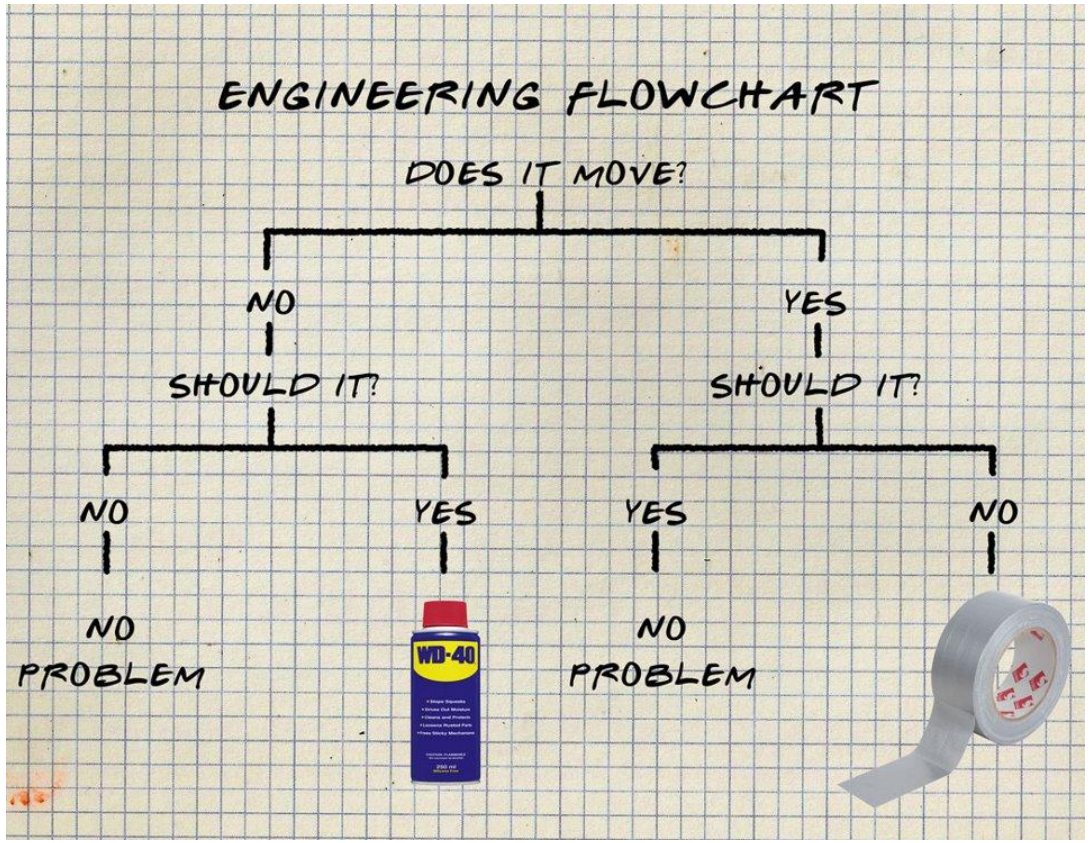

satellite image 1
vegetation
non vegetation
latitude
longitude
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2


satellite image 2
vegetation
non vegetation
latitude
longitude
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
1.2

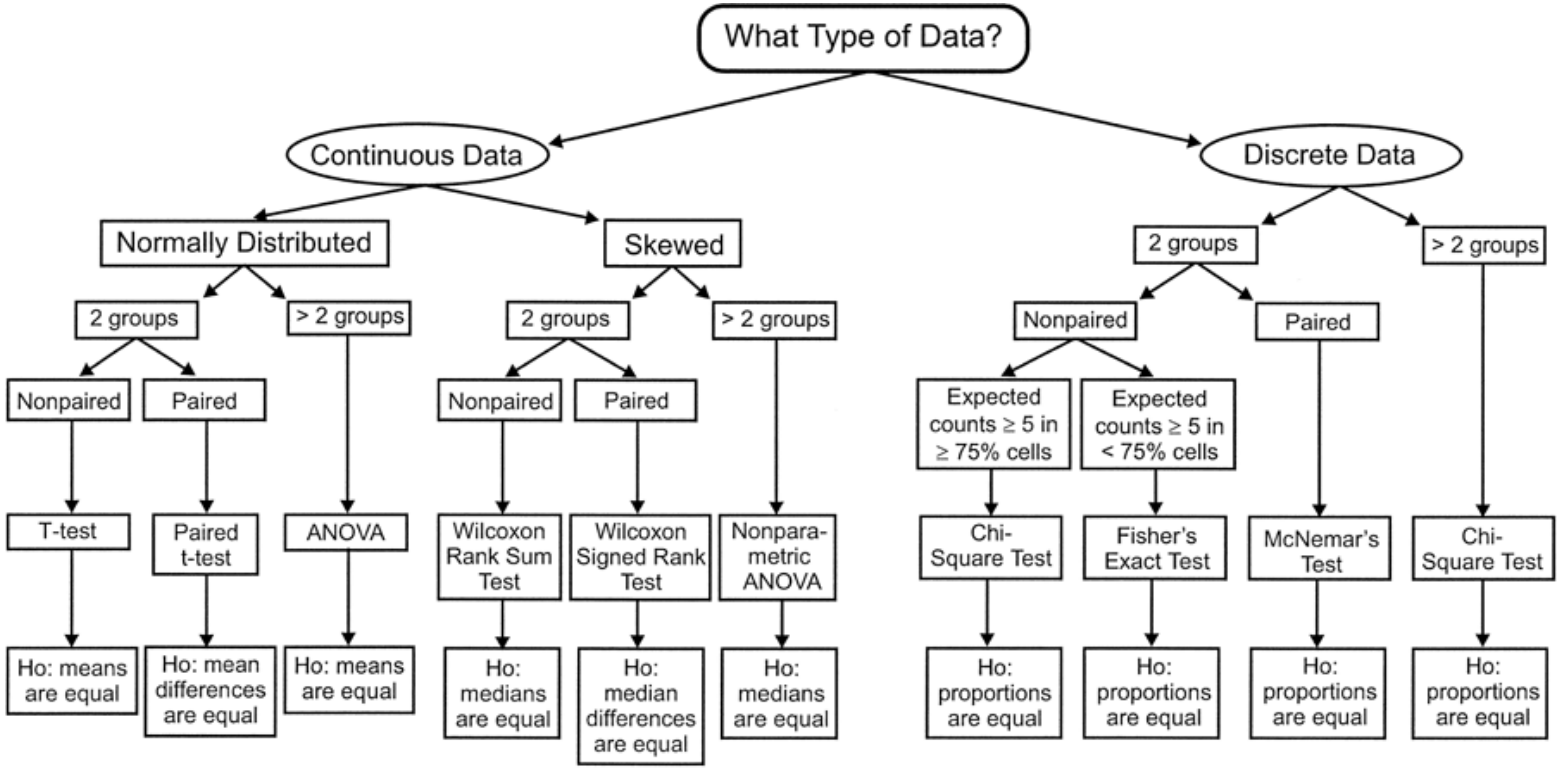
Notice that in all of the datasets the classes are still well-separated in the feature space, but *the decision boundaries cannot easily be described by single equations*:



People in every walk of life have long been using interpretable models for differentiating between classes of objects and phenomena:



Or in the [inferential] data analysis world:



Source: Waning B, Montagne M: *Pharmacoepidemiology: Principles and Practice*: http://www.accesspharmacy.com

It turns out that the simple flow charts in our examples can be formulated as mathematical models for classification and these models have the properties we desire; they are:

1. interpretable by humans
2. have sufficiently complex decision boundaries
3. the decision boundaries are locally linear, each component of the decision boundary is simple to describe mathematically.

Why do tree-based models still outperform deep learning on typical tabular data?

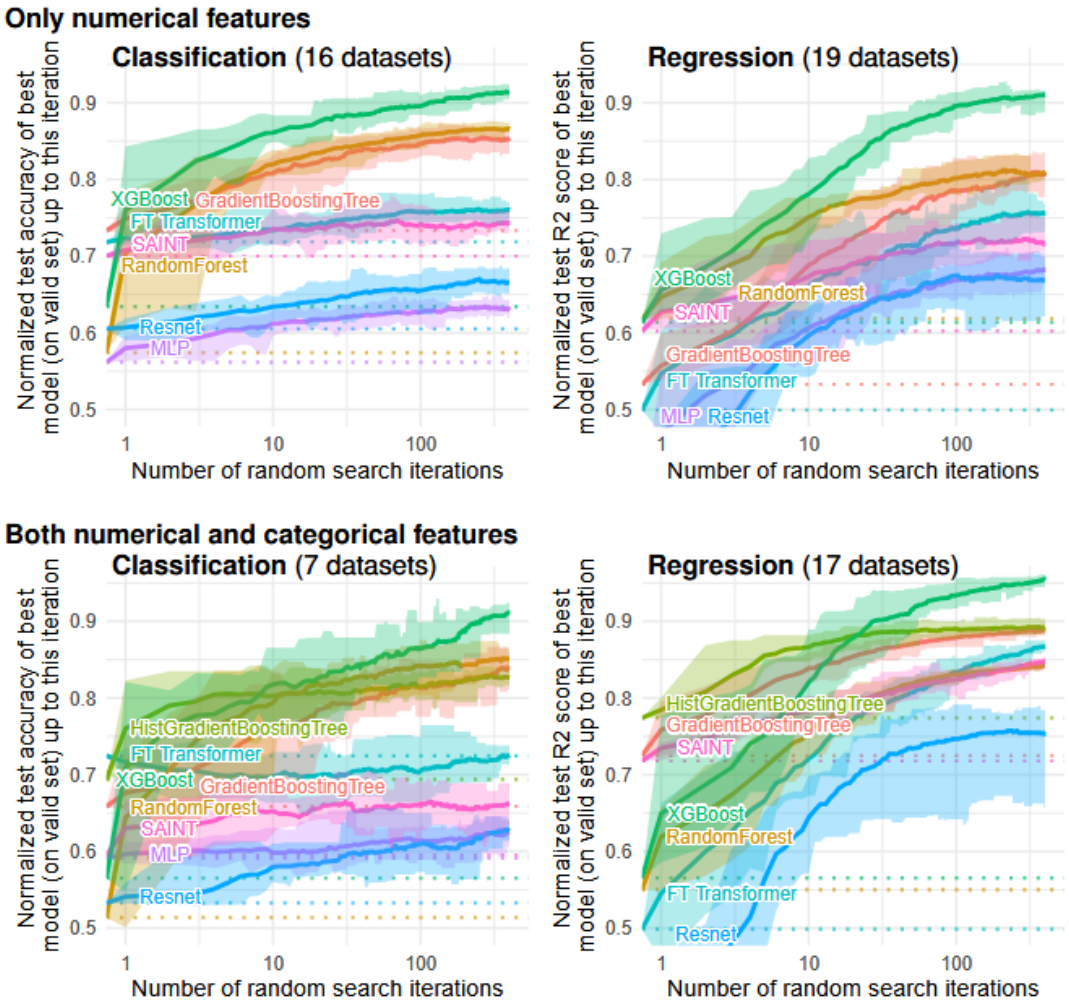
Léo Grinsztajn
Soda, Inria Saclay
leo.grinsztajn@inria.fr

Edouard Oyallon
MLIA, Sorbonne University

Gaël Varoquaux
Soda, Inria Saclay

Abstract

While deep learning has enabled tremendous progress on text and image datasets, its superiority on tabular data is not clear. We contribute extensive benchmarks of standard and novel deep learning methods as well as tree-based models such as XGBoost and Random Forests, across a large number of datasets and hyperparameter combinations. We define a standard set of 45 datasets from varied domains with clear characteristics of tabular data and a benchmarking methodology accounting for both fitting models and finding good hyperparameters. Results show that tree-based models remain state-of-the-art on medium-sized data (~10K samples) even without accounting for their superior speed. To understand this gap, we conduct an empirical investigation into the differing inductive biases of tree-based models and neural networks. This leads to a series of challenges which should guide researchers aiming to build tabular-specific neural network: **1.** be robust to uninformative features, **2.** preserve the orientation of the data, and **3.** be able to easily learn irregular functions. To stimulate research on tabular architectures, we contribute a standard benchmark and raw data for baselines: every point of a 20 000 compute hours hyperparameter search for each learner.



SUPERVISED 1 SUPERVISE 2 DESIGN SPACE

Flow charts whose graph is a tree (connected and no cycles) represents a model called a decision tree.

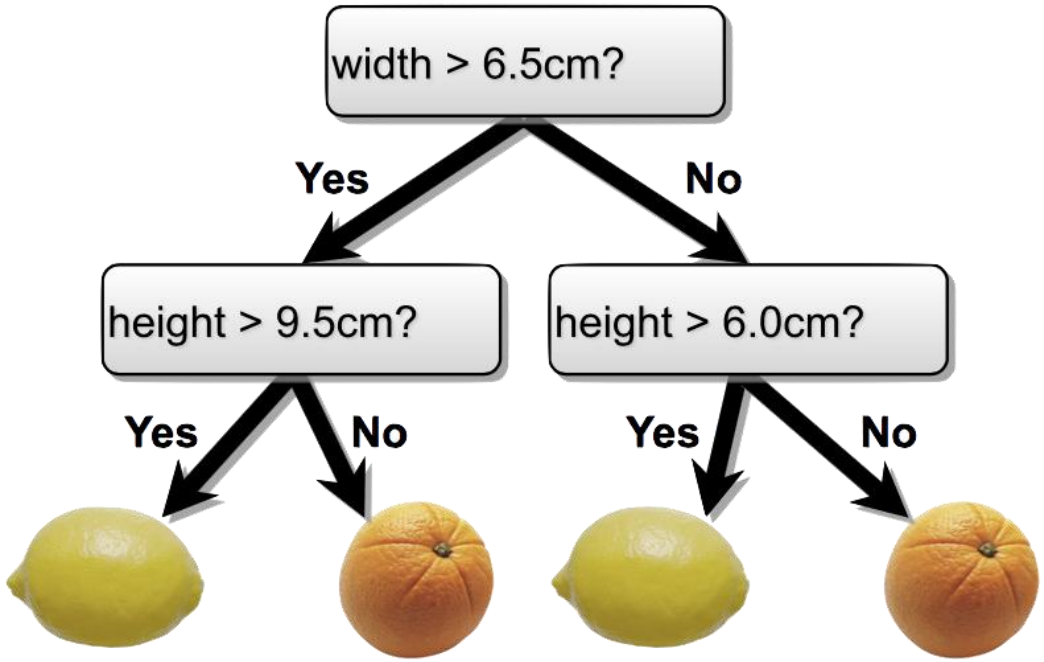
Formally, a decision tree model is one in which the final outcome of the model is based on a series of comparisons of the values of predictors against threshold values.

In a graphical representation (flow chart),

- the internal nodes of the tree represent attribute testing.
- branching in the next level is determined by attribute value (yes/no).
- terminal leaf nodes represent class assignments.

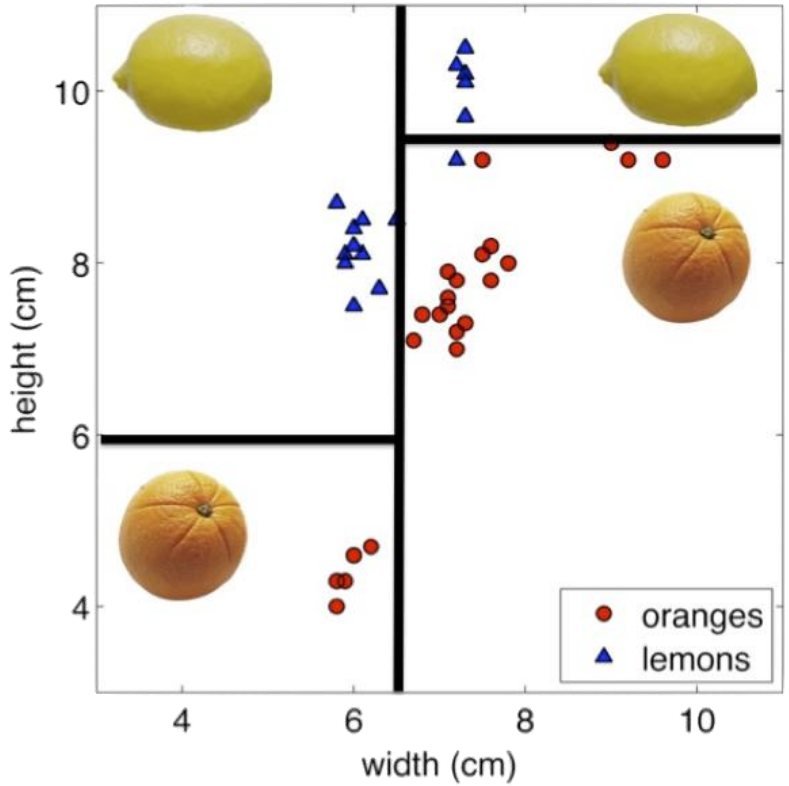
Flow charts whose graph is a tree (connected and no cycles) represents a model called a decision tree.

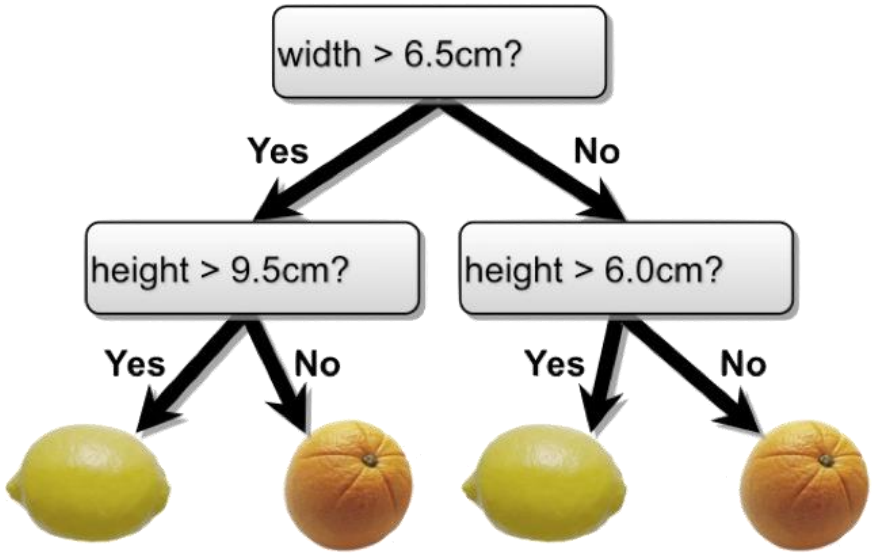
Formally, a decision tree model is one in which the final outcome of the model is based on a series of comparisons of the values of predictors against threshold values.



Decision Trees – The Geometry of Flow Charts

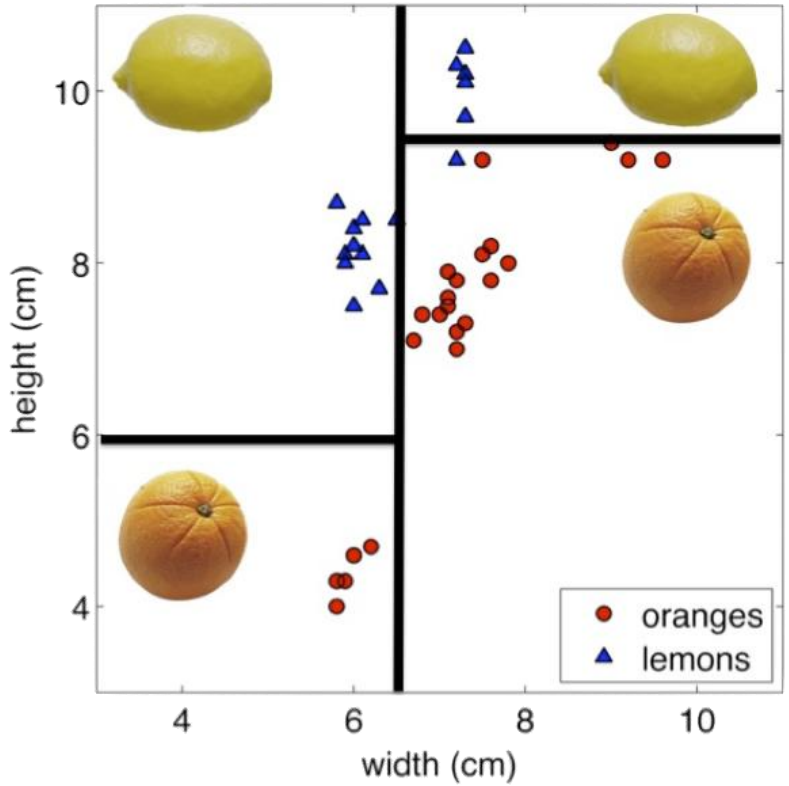
Every flow chart tree corresponds to a partition of the feature space by axis aligned lines or (hyper) planes. Conversely, every such partition can be written as a flow chart tree.

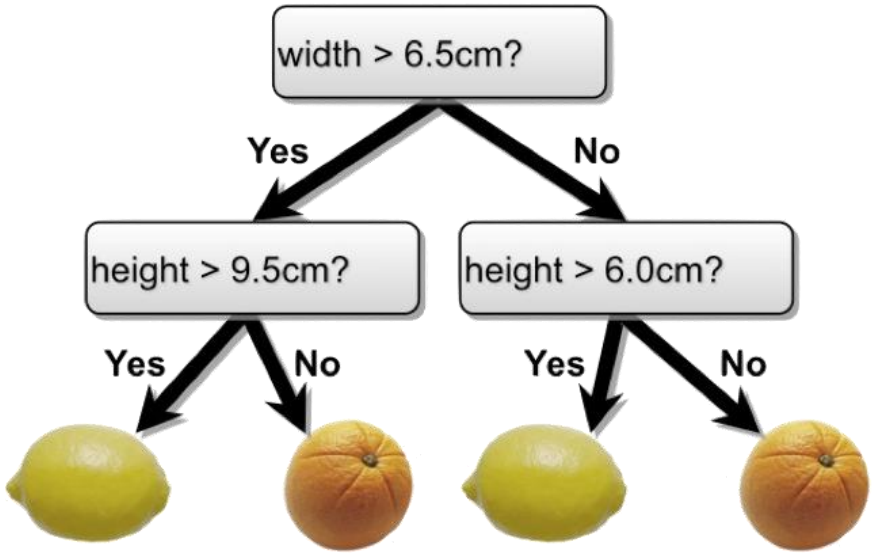




Decision Trees – The Geometry of Flow Charts

Each comparison and branching represents splitting a region in the feature space on a single feature. Typically, at each iteration, we split once along one dimension (one predictor). Why?





Given a training set, *learning* a decision tree model for binary classification means:

producing an *optimal* partition of the feature space with axis-aligned linear boundaries (very interpretable!),

each region is predicted to have a class label based on the largest class of the training points in that region (Bayes' classifier) when performing prediction.

Decision Trees — Splitting Criteria

While there is no ‘correct’ way to define an optimal split, there are some common sensical guidelines for every splitting criterion:

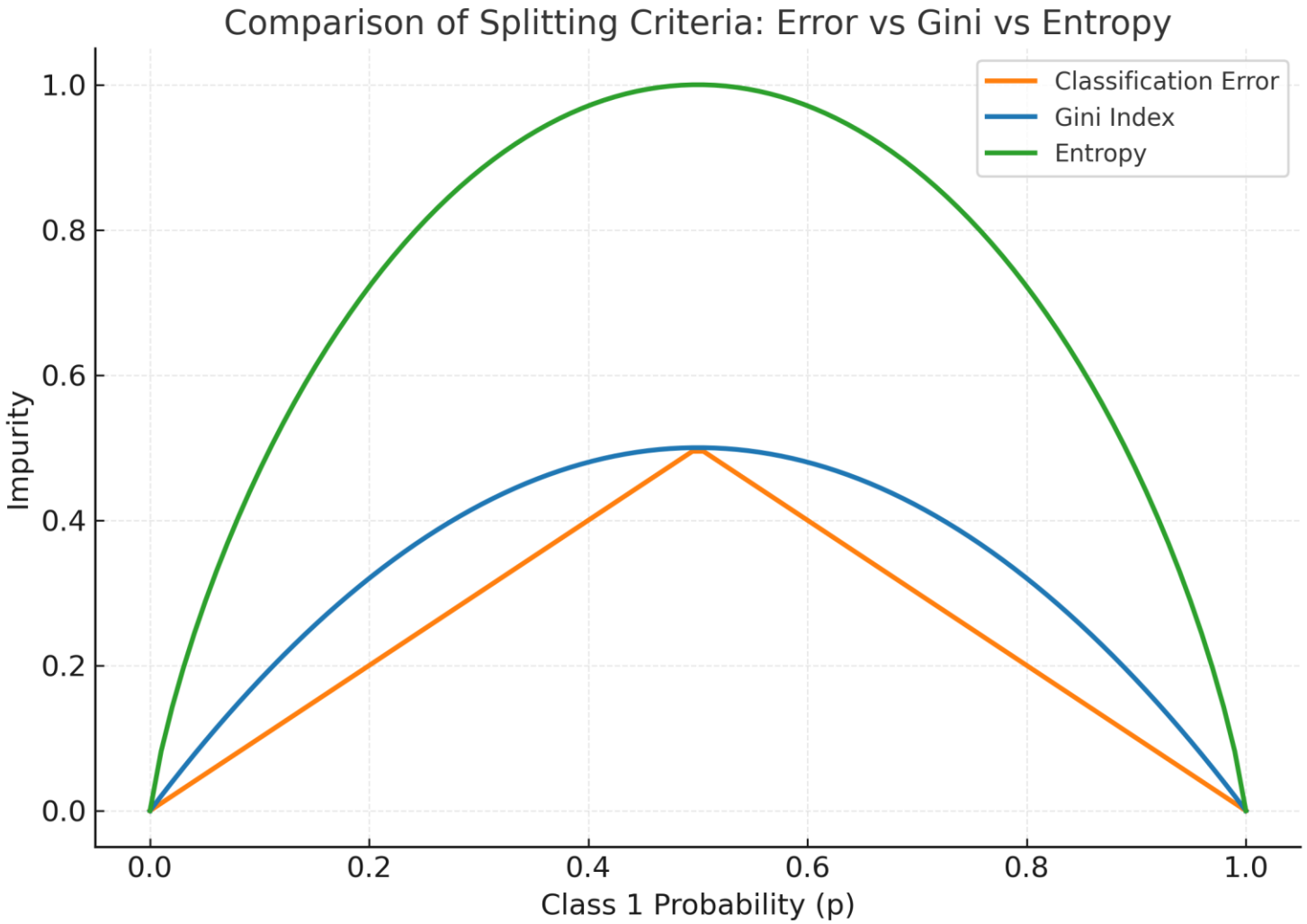
the regions in the feature space should grow progressively more pure with the number of splits. That is, we should see each region ‘specialize’ towards a single class.

the fitness metric of a split should take a differentiable form (making optimization possible).

we shouldn’t end up with empty regions - regions containing no training points.

Read up on different splitting criteria

Decision Trees — Splitting Criteria



We make some observations about our models:

(High Bias) A tree of depth 4 is not a good fit for the training data - it's unable to capture the nonlinear boundary separating the two classes.

(Low Bias) With an extremely high depth, we can obtain a model that correctly classifies all points on the boundary (by zig-zagging around each point).

(Low Variance) The tree of depth 4 is robust to slight perturbations in the training data - the square carved out by the model is stable if you move the boundary points a bit.

(High Variance) Trees of high depth are sensitive to perturbations in the training data, especially to changes in the boundary points.

Not surprisingly, complex trees have low bias (able to capture more complex geometry in the data) but high variance (can overfit). Complex trees are also harder to interpret and more computationally expensive to train.

Decision Trees – Stopping Conditions

Common simple stopping conditions:

Don’t split a region if all instances in the region belong to the same class.

Don’t split a region if the number of instances in the sub-region will fall below pre-defined threshold (min_samples_leaf).

Don’t split a region if the total number of leaves in the tree will exceed pre-defined threshold.

The appropriate thresholds can be determined by evaluating the model on a held-out data set or, better yet, via cross-validation.

What is the major issue with pre-specifying a stopping condition?

you may stop too early or stop too late.

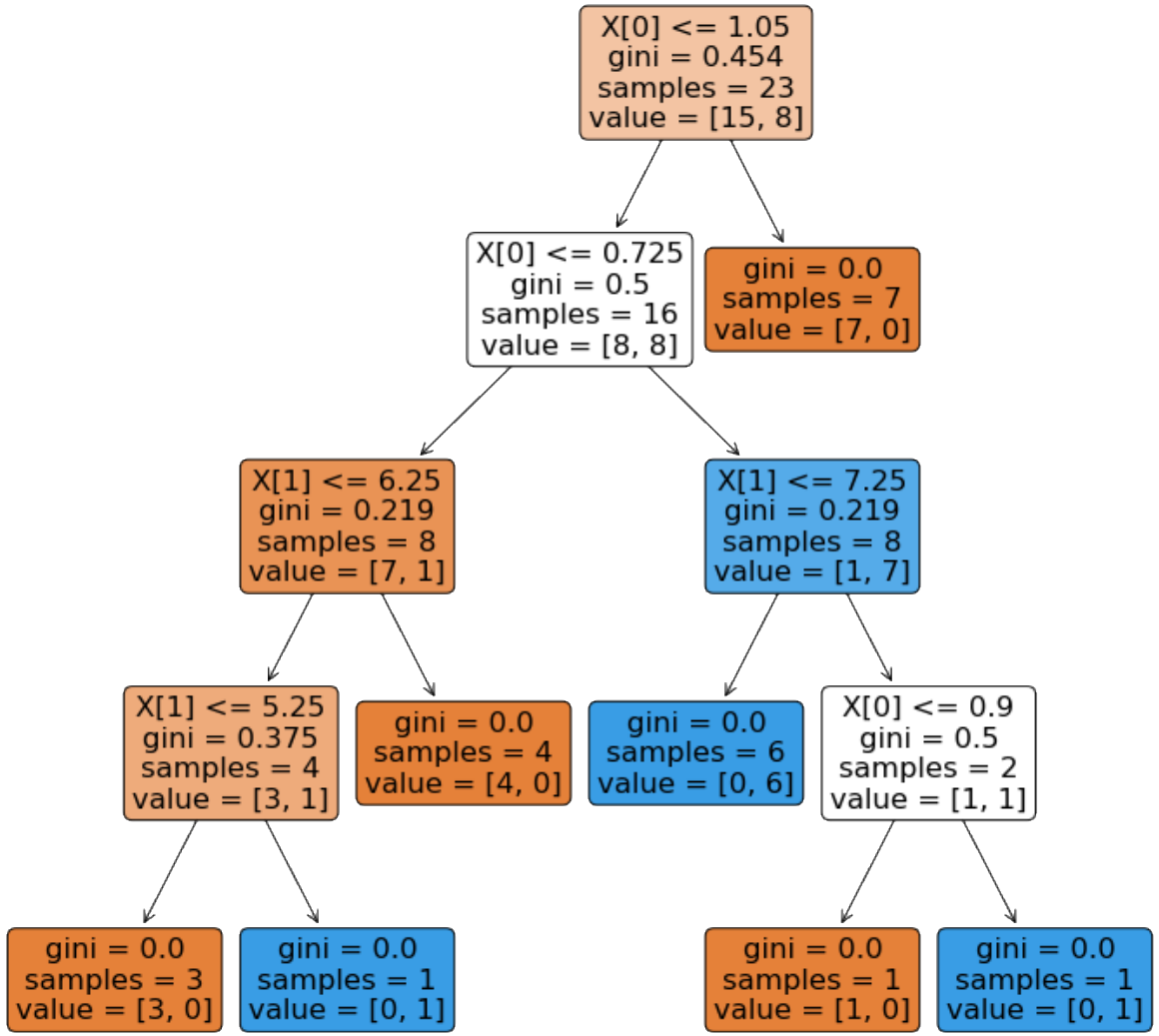
How can we fix this issue?

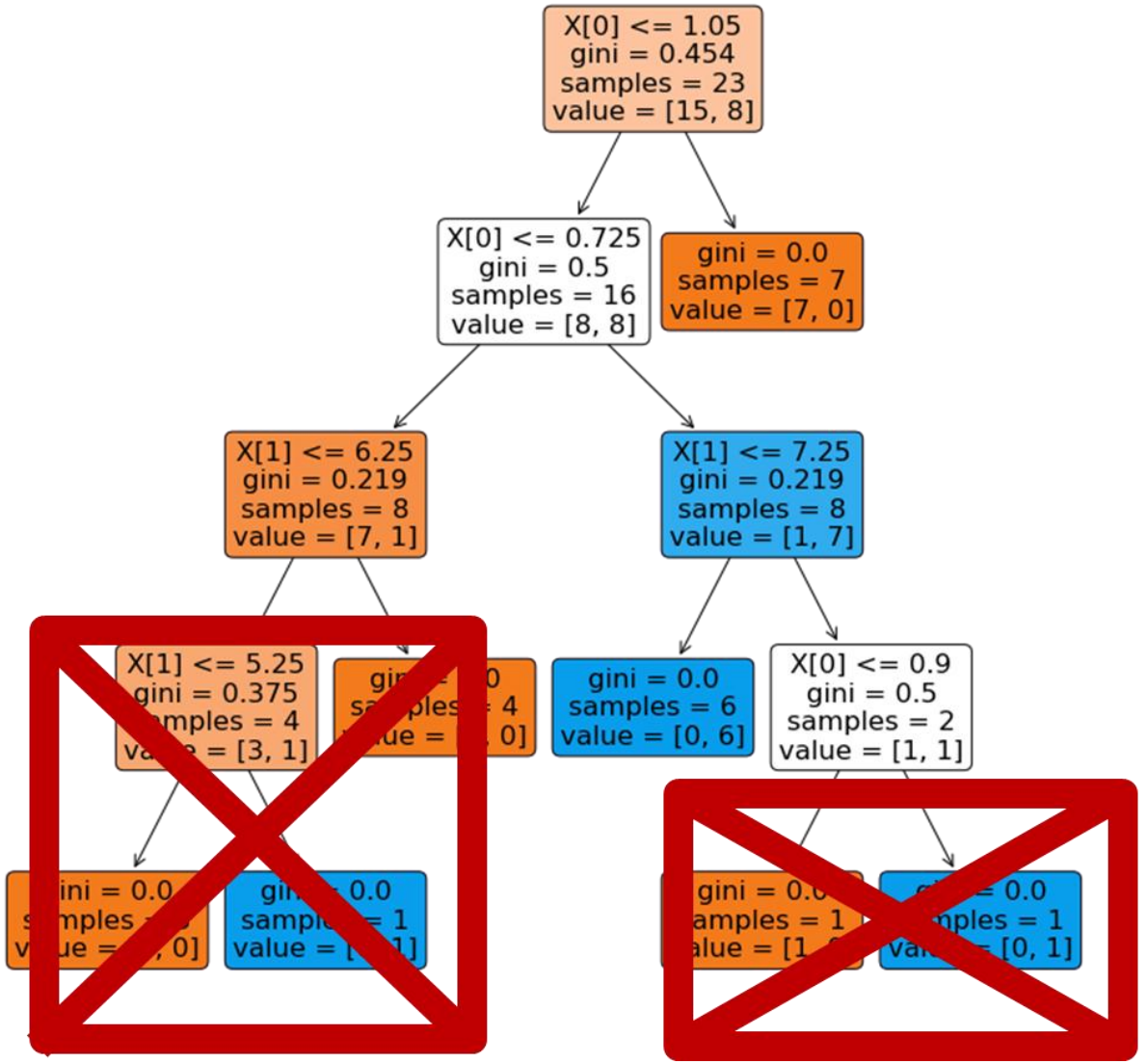
choose several stopping criterion (set minimal Gain(R) at various levels) and cross-validate which is the best.

What is an alternative approach to this issue?

Don't stop. Instead prune back!

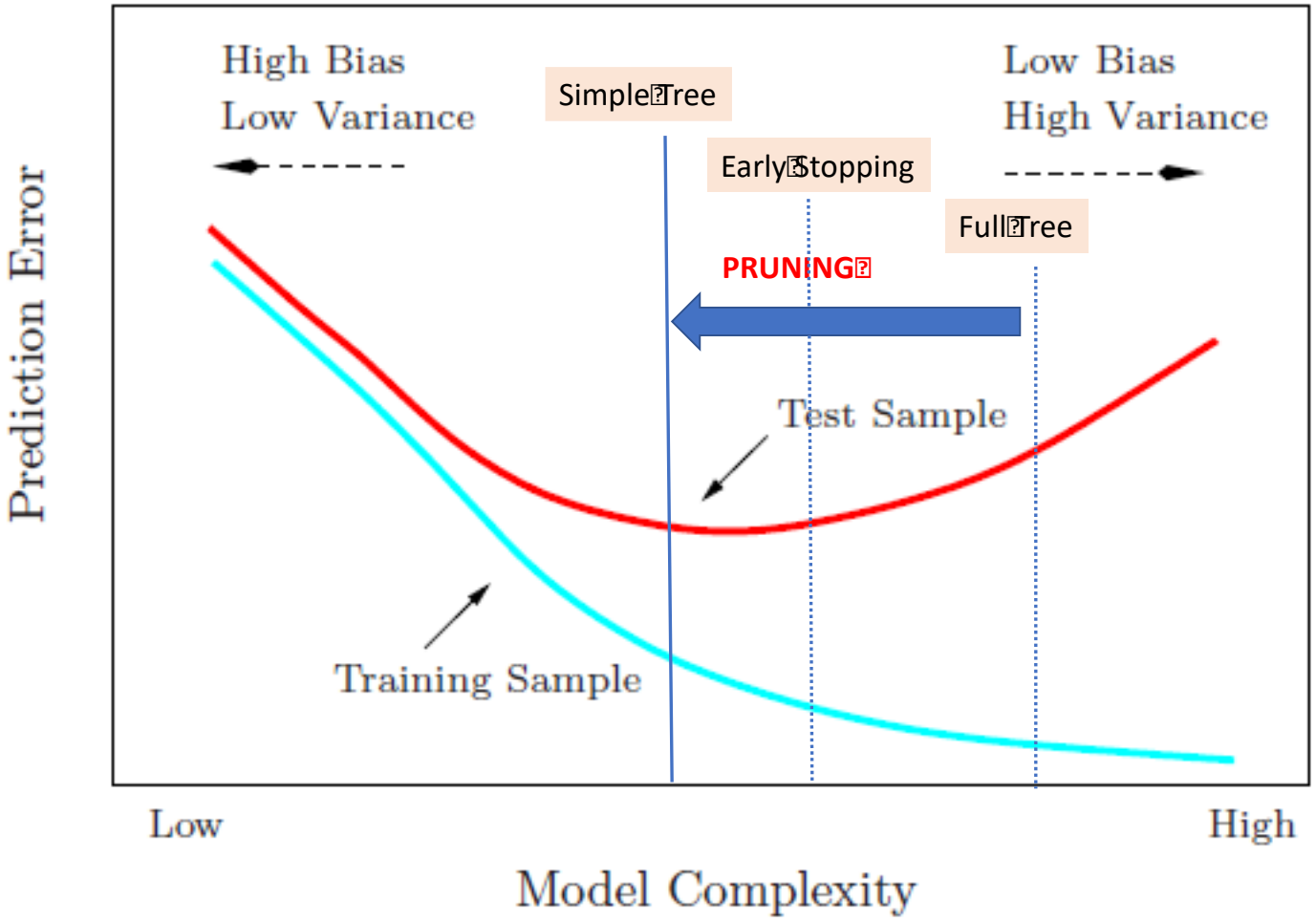
Decision Trees – Alternatives to Stopping Conditions (Pruning)





SUPERVISED 1 SUPERVISE 2 DESIGN SPACE

Decision Trees – Alternatives to Stopping Conditions (Pruning)



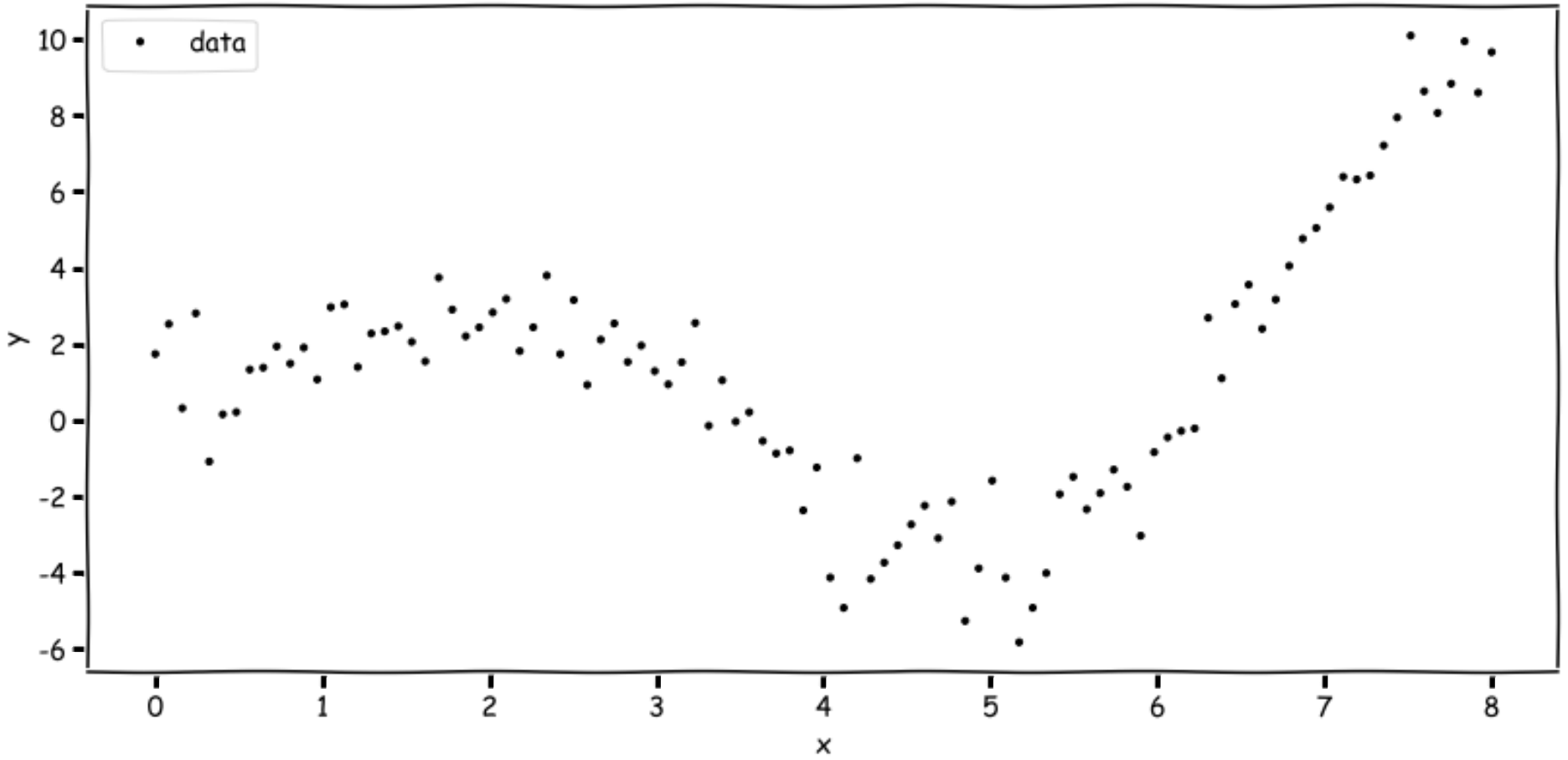
How can this decision tree approach apply to a *regression problem* (quantitative outcome)?

Questions to consider:

- What would be a reasonable loss function?
- How would you determine any splitting criteria?
- How would you perform prediction in each leaf?

A picture is worth a thousand words…

How do we decide a split here?



With just two modifications, we can use a decision tree model for regression:

1. The three splitting criteria we've examined each promoted splits that were pure - new regions increasingly specialized in a single class.

    A. For classification, purity of the regions is a good indicator the performance of the model.
    B. For regression, we want to select a splitting criterion that promotes splits that improves the predictive accuracy of the model as measured by, say, the MSE.

2. For regression with output in $\mathbb{R}$, we want to label each region in the model with a real number - typically the average of the output values of the training points contained in the region.

The learning algorithms for decision trees in regression tasks is:

1. Start with an empty decision tree (undivided features pace)
2. Choose a predictor j on which to split and choose a threshold value t_j for splitting such that the weighted average MSE of the new regions as smallest possible:

$$\underset{j,t_j}{\operatorname{argmin}} \left\{ \frac{N_1}{N} \operatorname{MSE}(R_1) + \frac{N_2}{N} \operatorname{MSE}(R_2) \right\}$$

or equivalently,

$$\underset{j,t_j}{\operatorname{argmin}} \left\{ \frac{N_1}{N} \operatorname{Var}(y | x \in R_1) + \frac{N_2}{N} \operatorname{Var}(y | x \in R_2) \right\}$$

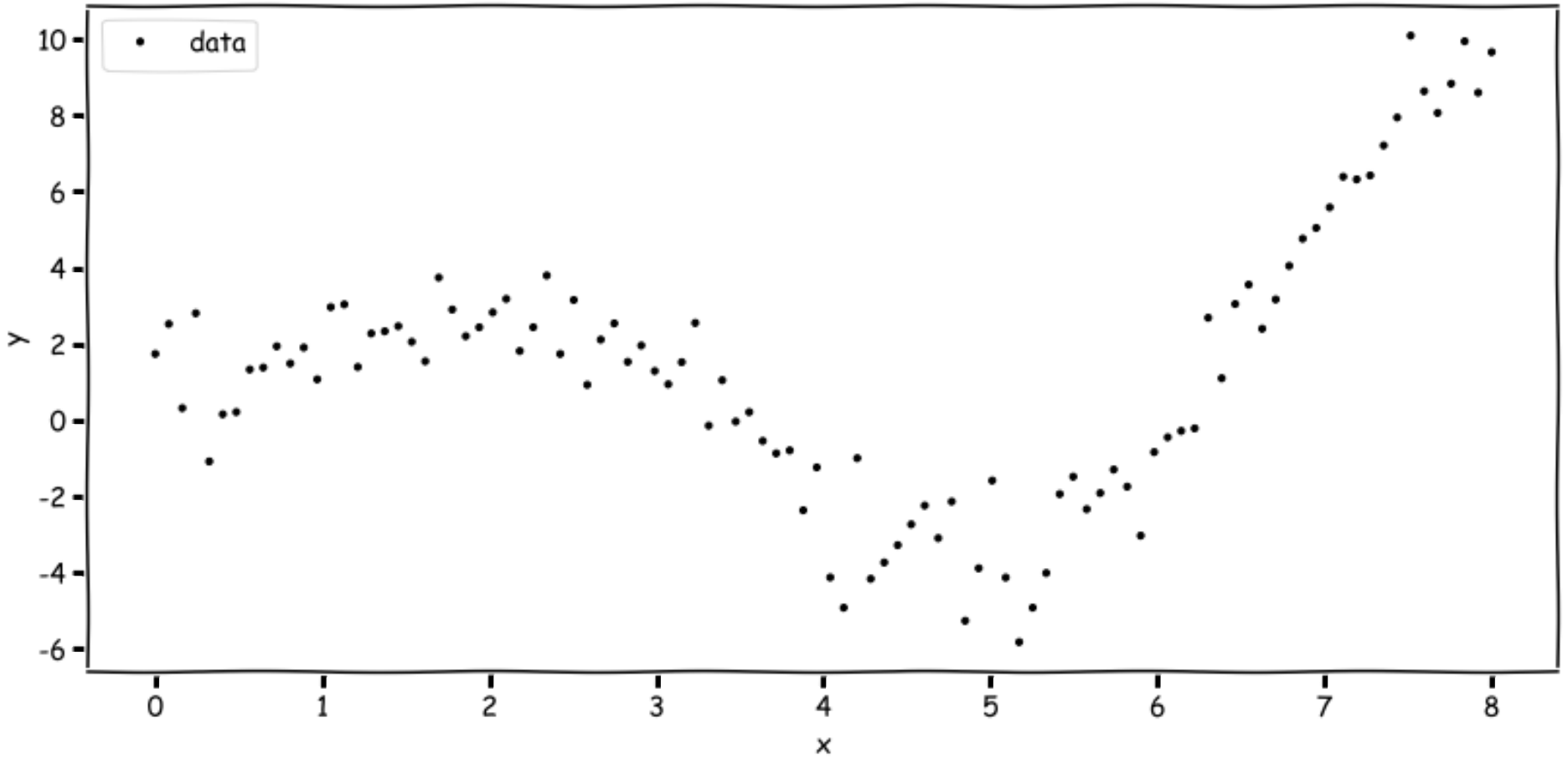
where N_i is the number of training points in R_i and N is the number of points in R.

3. Recurse on each new node until *stopping condition* is met.

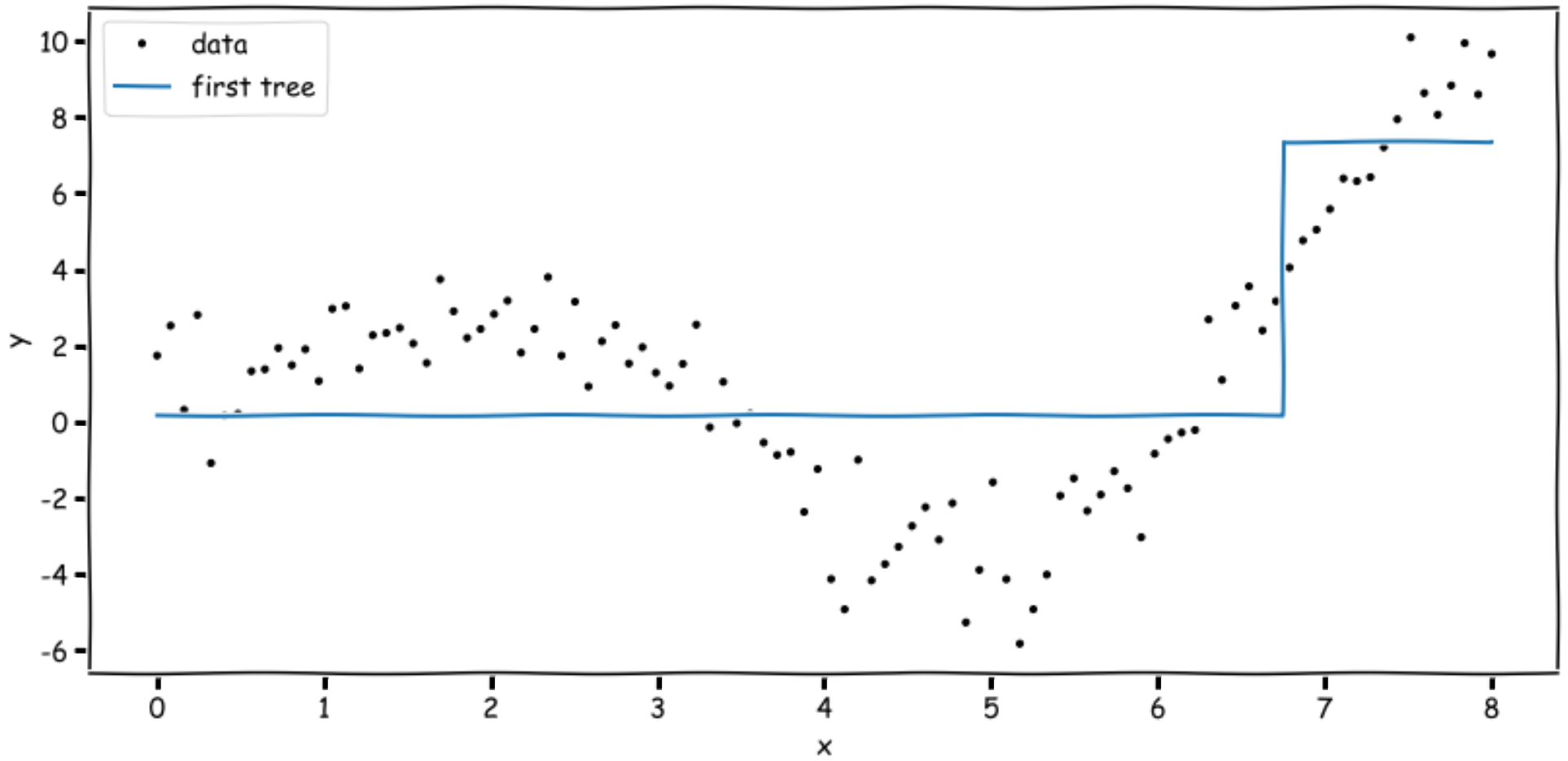
For any data point x_i

1. Traverse the tree until we reach a leaf node.
2. Averaged value of the response variable y's in the leaf (this is from the training set) is the $\hat{y}_i$.

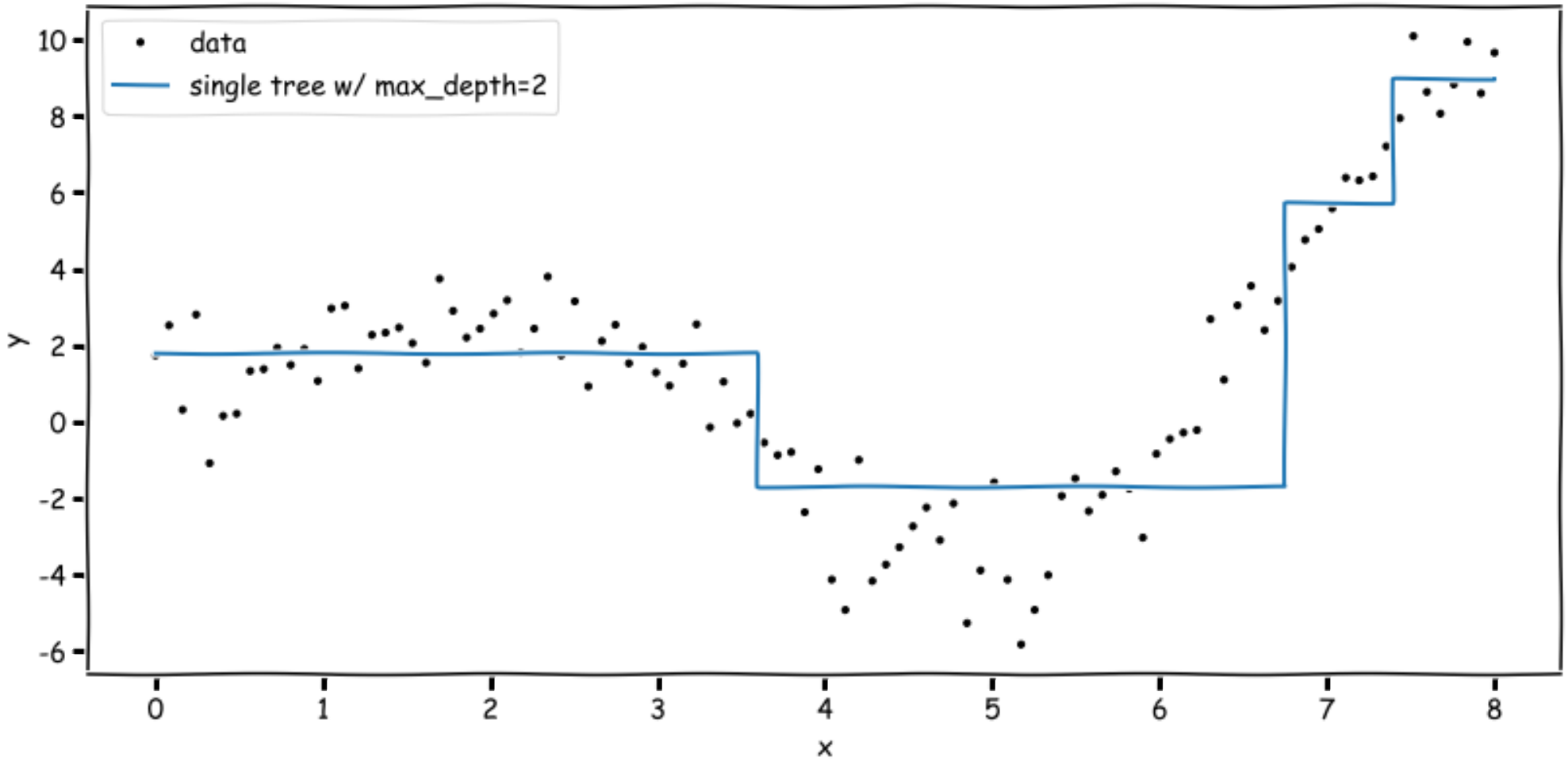
How do we decide a split here?



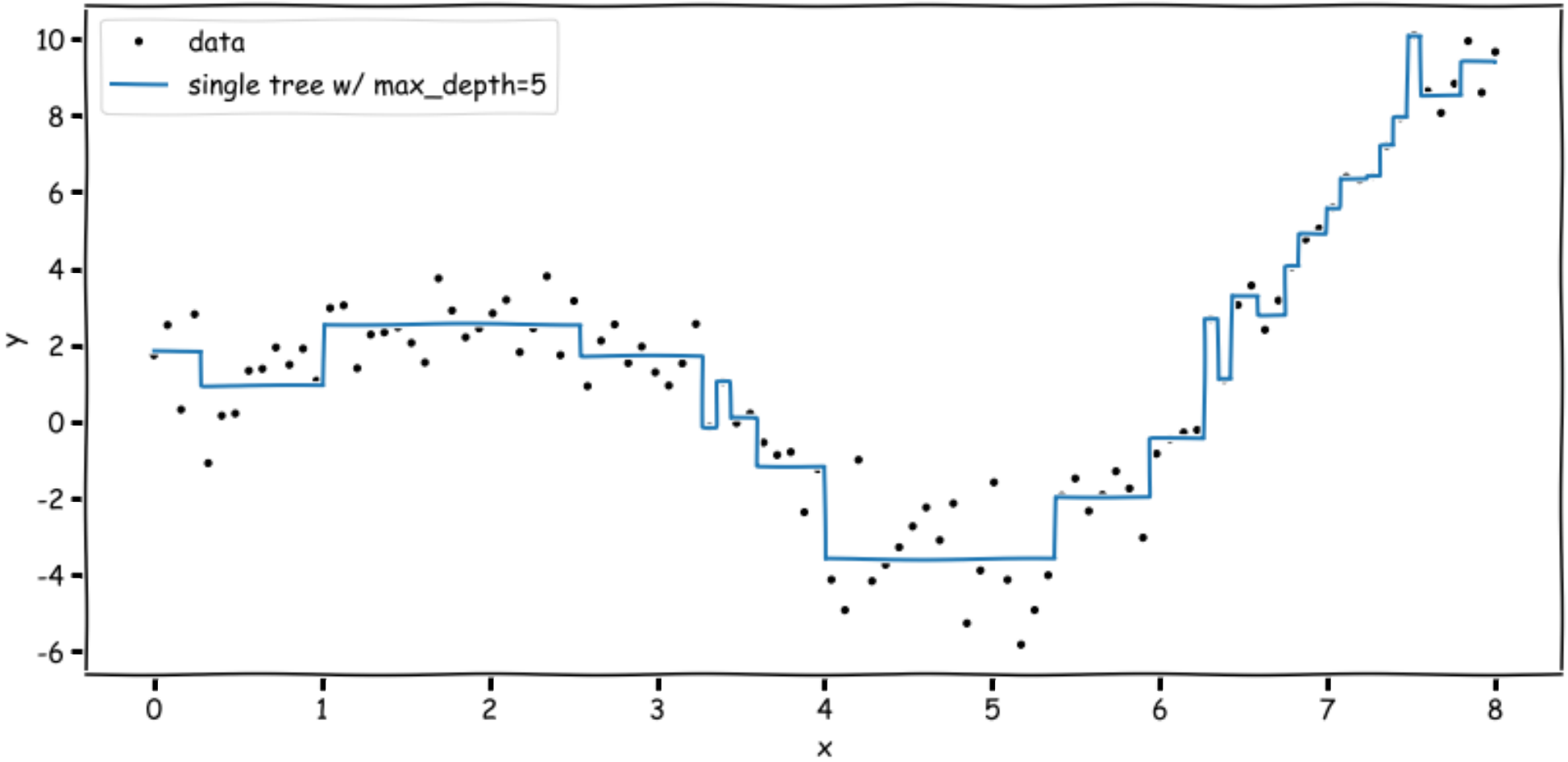
Max_depth = 1



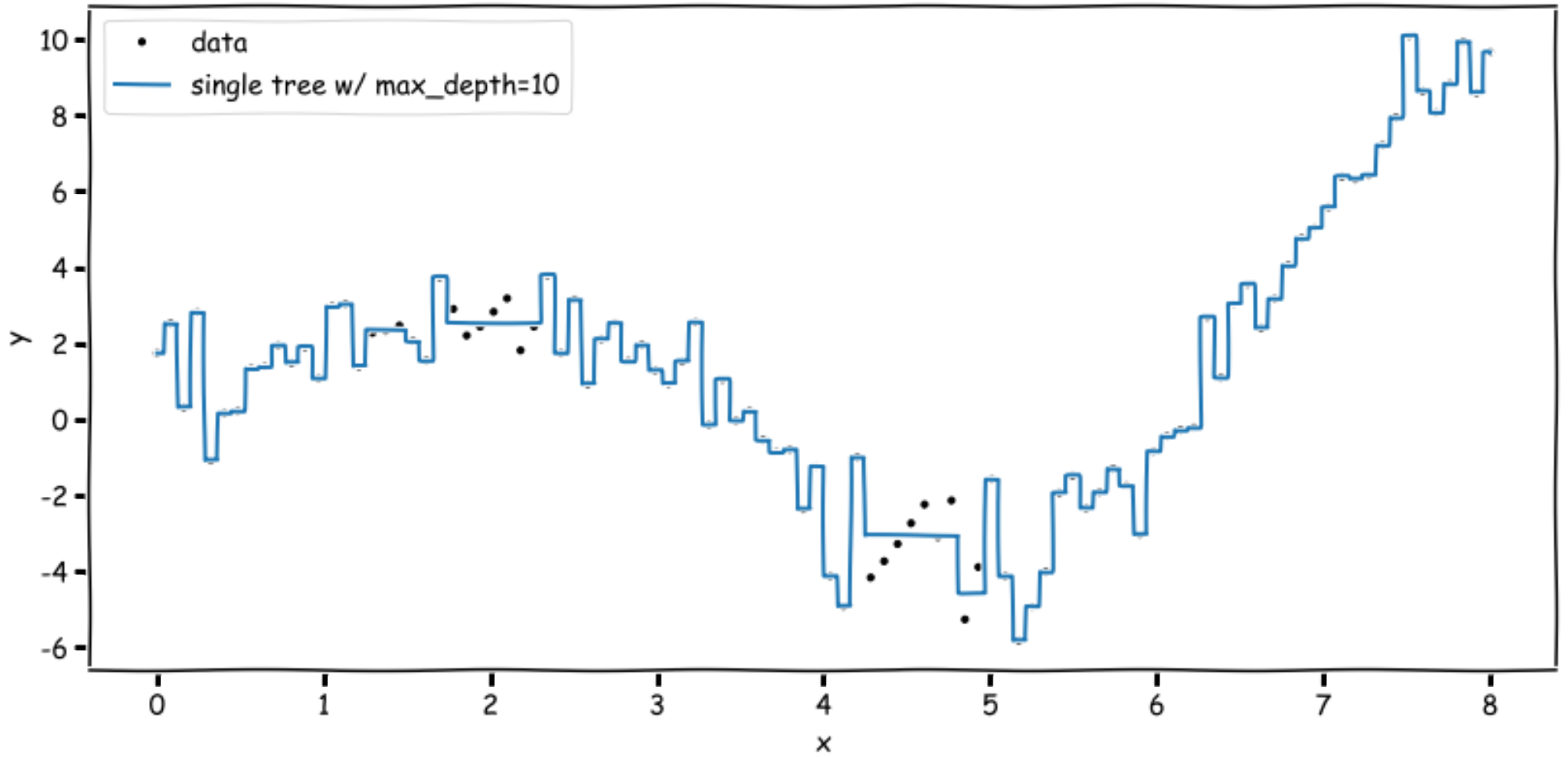
Max_depth = 2



Max_depth = 5



Max_depth = 10



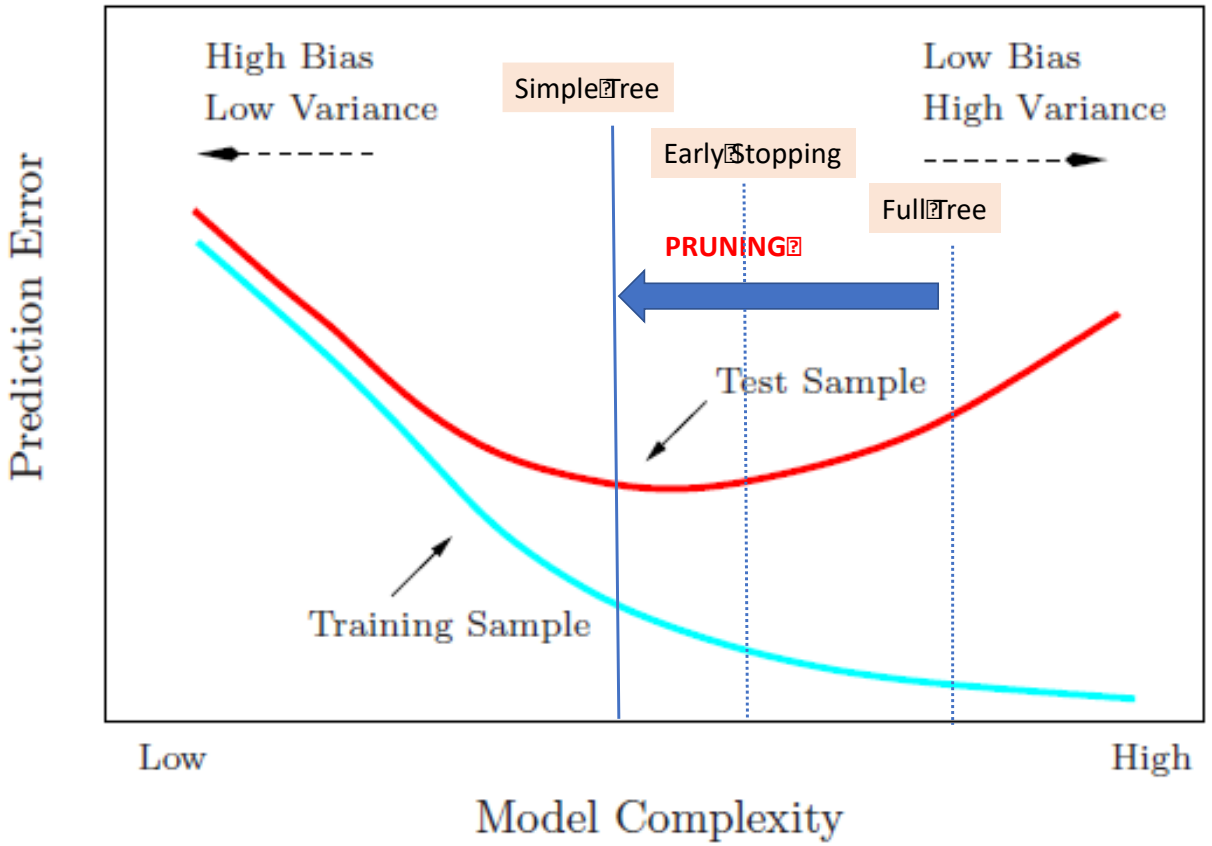
Most of the stopping conditions, like maximum depth or minimum number of points in region, we saw for classification trees can still be applied.

In the place of purity gain, we can instead compute accuracy gain for splitting a region R

$$\text{Gain}(R) = \Delta(R) = MSE(R) - \frac{N_1}{N} MSE(R_1) - \frac{N_2}{N} MSE(R_2)$$

and stop the tree when the gain is less than some pre-defined threshold.

Same issues as with classification trees. Avoid overfitting by pruning or limiting the depth of the tree and using CV



Sklearn Model

DecisionTreeRegressor

```
class sklearn.tree.DecisionTreeRegressor(*, criterion='squared_error',
splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1,
min_weight_fraction_leaf=0.0, max_features=None, random_state=None,
max_leaf_nodes=None, min_impurity_decrease=0.0, ccp_alpha=0.0, monotonic_cst=None)
```

A decision tree regressor. [source]

splitter : *{"best", "random"}, default="best"*

The strategy used to choose the split at each node. Supported strategies are "best" to choose the best split and "random" to choose the best random split.

max_depth : *int, default=None*

The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

For an example of how `max_depth` influences the model, see Decision Tree Regression.

min_samples_split : *int or float, default=2*

The minimum number of samples required to split an internal node:

- If int, then consider `min_samples_split` as the minimum number.
- If float, then `min_samples_split` is a fraction and `ceil(min_samples_split * n_samples)` are the minimum number of samples for each split.

Google Colab
https://bit.ly/BPS5231-L3

L03.1
Supervised 3

Recap

The Linear Classifier

Logistic Regression

Confusion Matrix

ROC Curve

Scikit-Learn

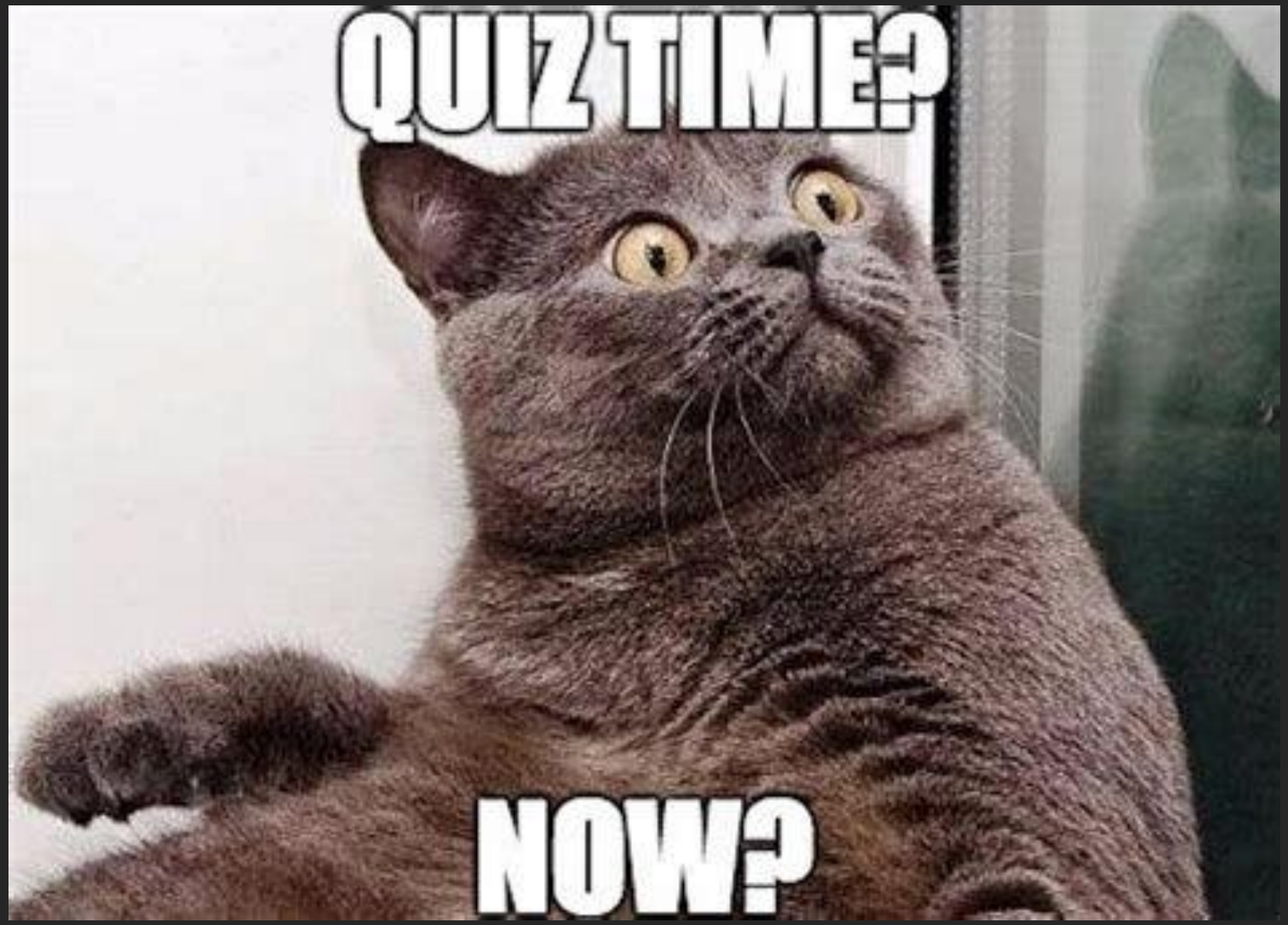
L03.2
Supervised 4

Decision Trees

L03.3
Simulations

Rhino

ClimateStudio


QUIZ TIME?
NOW?