Machine Learning I

Lecture 1

Nathaniel Bade

Northeastern University Department of Mathematics

Table of contents

What is Machine Learning?

Statistical Learning Theory

This course will use **statistical learning theory** to understand the mathematical aspects of **machine learning**.

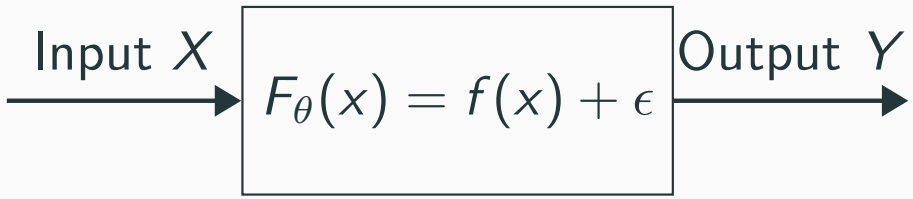
Statistical learning theory is roughly about understanding how much data is required to make predictions to a certain degree of accuracy. We will first understand some theoretical aspects of the theory, and then move to practical implementation using Python.

Both **machine learning** and **statistical modeling** deal with the estimation of parameters in mathematical models. Both theories make assumptions to achieve this.

Machine Learning vs Statistics

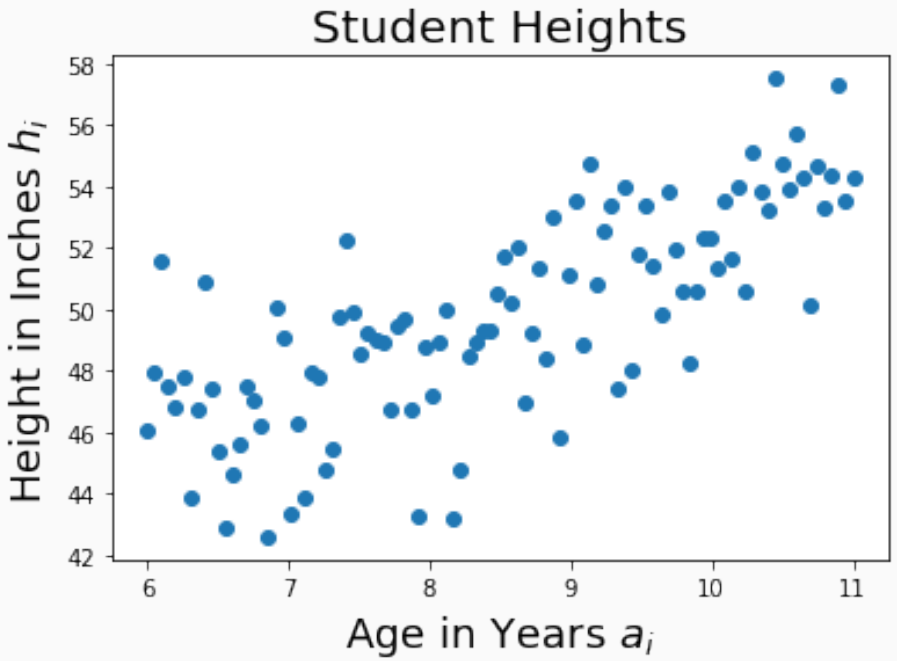
Statistical modeling starts with a sample space S and a set of probability distributions P on S. We then posit a mathematical model $F_\theta(x)$ depending on parameters θ to generate estimates about the likelihood of observed data.

For example, if ϵ is a random variable and $f : X \to Y$ is a labeling of data, a statistical model may take the form



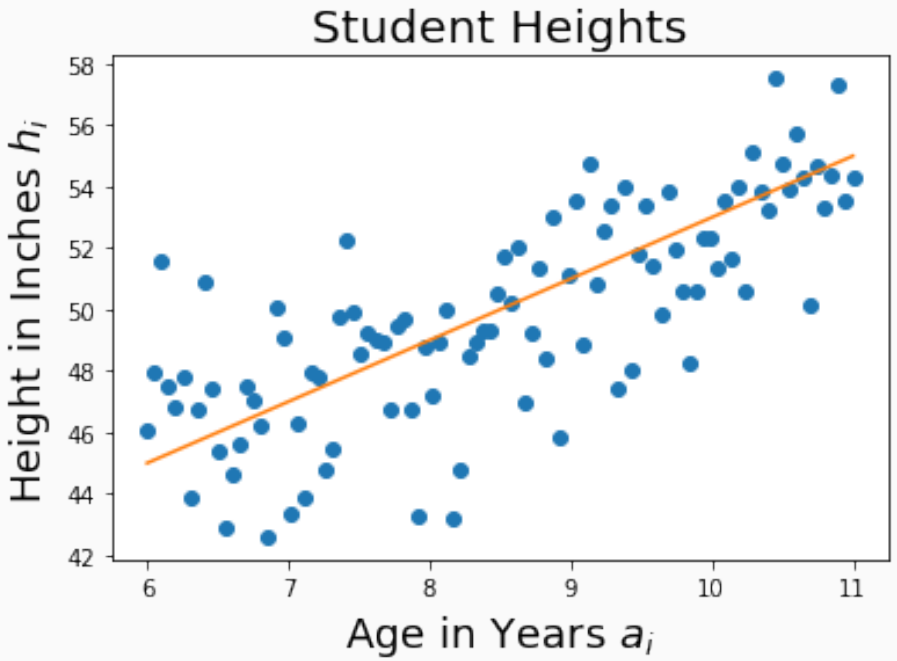
We then pick θ using **maximum likelihood estimation**. This requires knowing the probability distribution and model function used to generates the underlying data, or at least having a good guess.

Statistical Modeling



Example: Student height: In a school, students ages a_i are uniformly distributed between ages 6 and 11.

Statistical Modeling

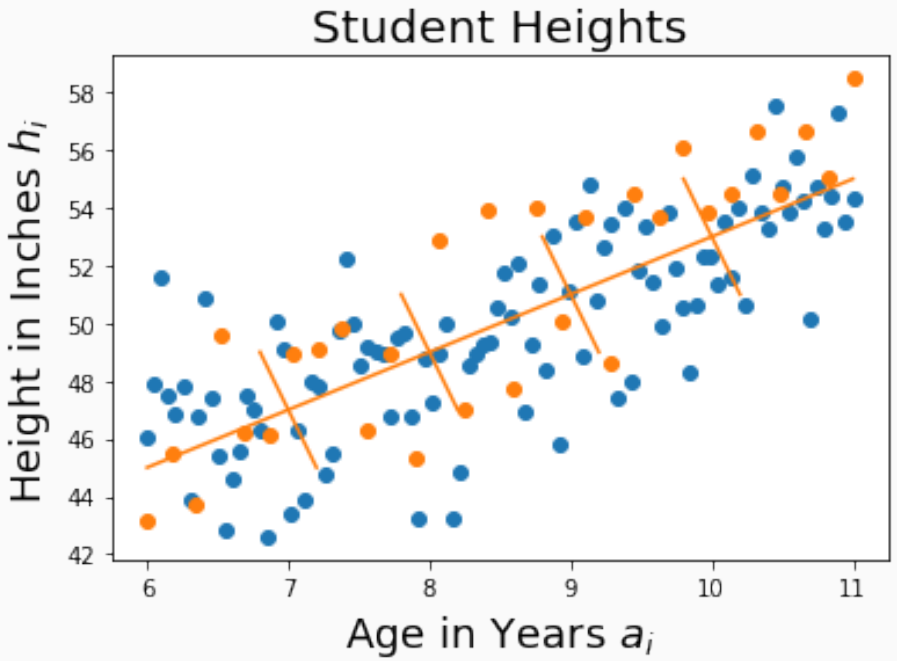


We might then predict height h_i of child i via their age a_i, using the model

$$h_i = b_0 + b_1 a_i + \epsilon_i \, .$$

We include a stochastic variable ϵ_i since h_i must fit all the data. To do statistical inference, we make the assumption that ϵ_i is Gaussian.

Statistical Modeling

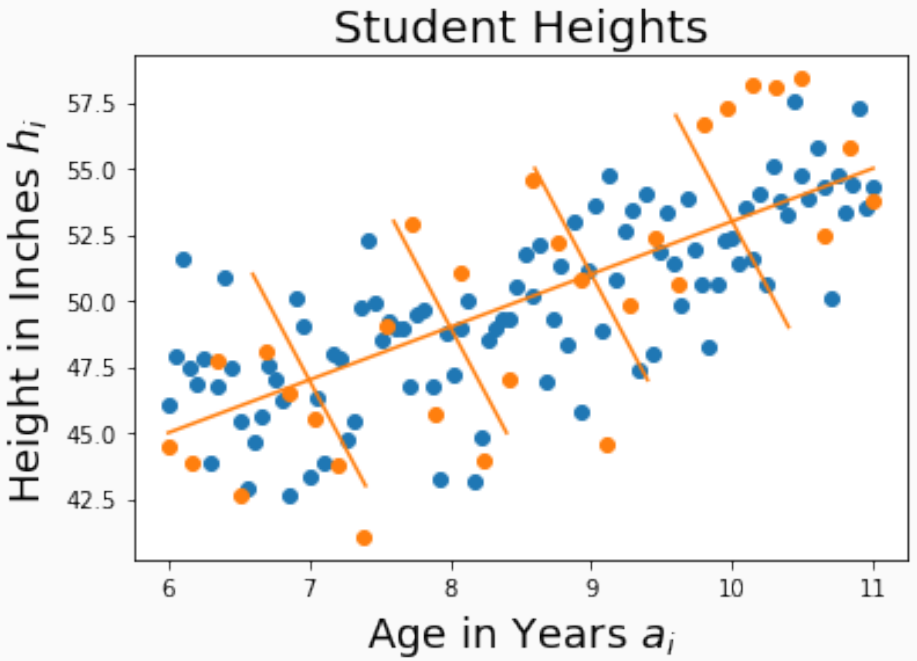


Model : $h_i = b_0 + b_1 a_i + \epsilon_i$

Sample space: $S = [6, 11] \times \mathbb{R}^+$ all possible pairs (a, h).

Parameters: $\theta = (\sigma_\epsilon^2, b_0, b_1)$. Each triple of values defines a distribution P_θ on X. P is the set of all such distributions.

Statistical Modeling



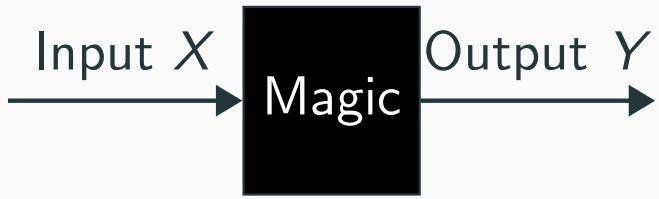
Model : $h_i = b_0 + b_1 a_i + \epsilon_i$

Sample space: $X = [6, 11] \times \mathbb{R}^+$ all possible pairs (a, h).

Parameters: $\theta = (\sigma_\epsilon^2, b_0, b_1)$. Each triple of values defines a distribution P_θ on X. P is the set of all such distributions.

Machine Learning vs Statistics

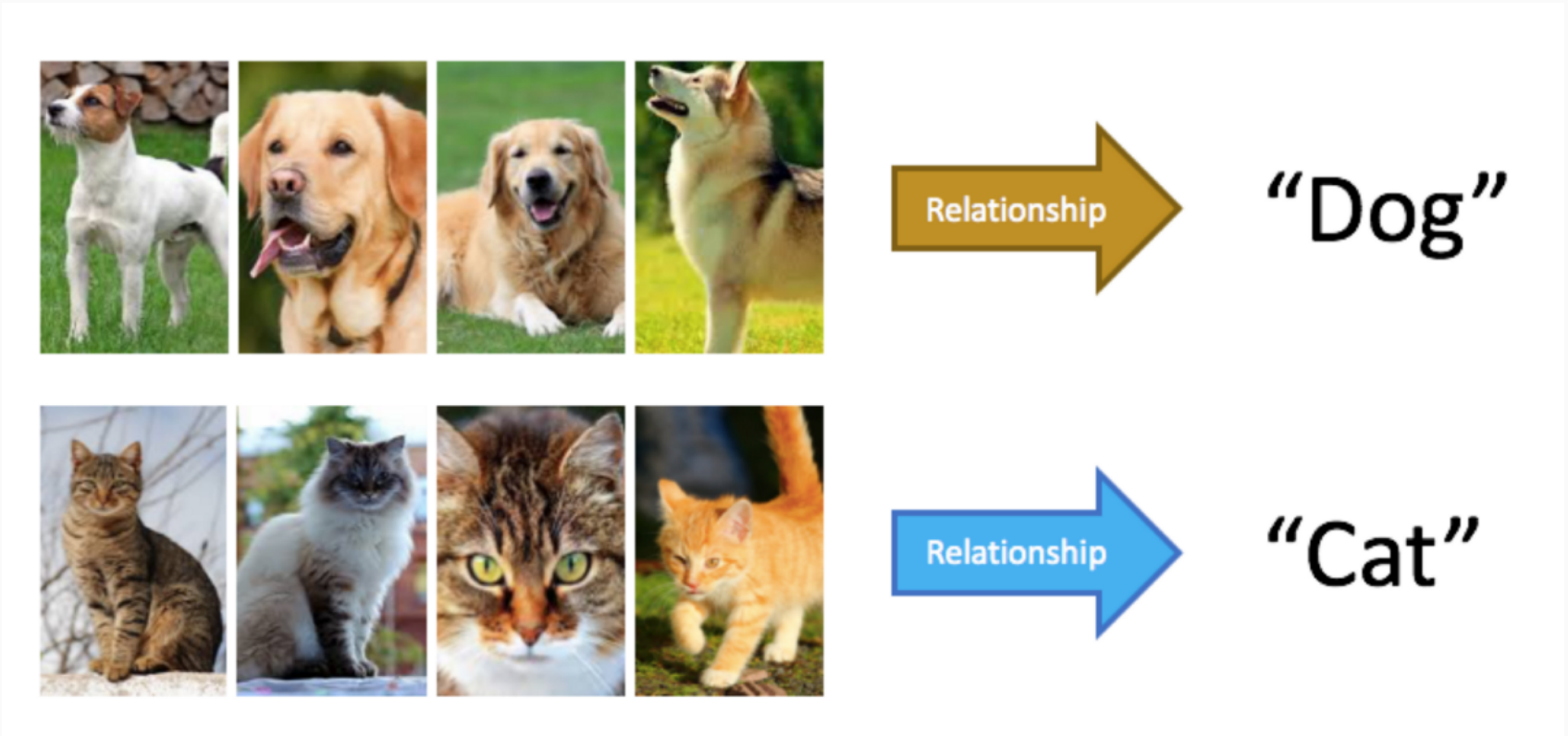
Machine learning is used in problems where the underlying distributions are unknown or unknowable. Typical examples include image classification, voice recognition, or gene expression.



Machine learning uses black-box methods that strive for accuracy regardless of the underlying distribution.

Main Question: For a given class of models, how many data points do we need in order to be certain (probabilistically) that the error in our model is small *no matter what the underlying distribution*.

Machine Learning Examples: Cats and Dogs



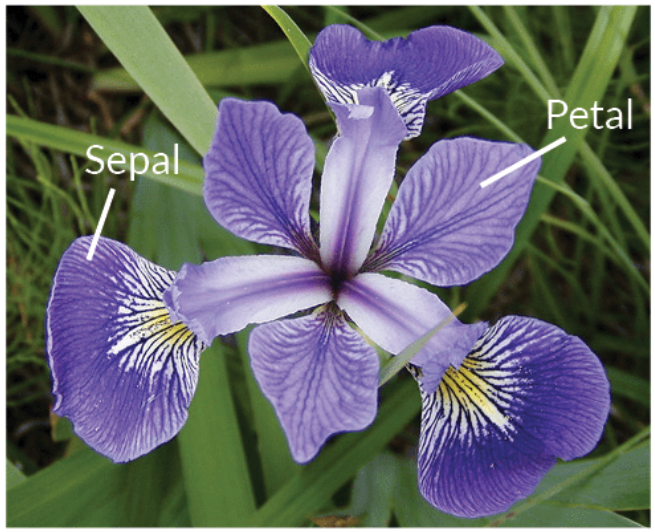
Example: In image classification, input $X = \mathbb{R}^{3\times 100\times 200}$ and output $Y = \{\text{"Dog"}, \text{"Cat"}\}$.

How do we find the distribution on the state space $S = X \times Y$?

Machine Learning Examples: Computer Vision

Example: For multilable classification, input $X = \mathbb{R}^{20\times 20}$ and output $Y = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Above is the famous MNIST data set of handwritten numbers.

Machine Learning Examples: Iris Classification



Iris Versicolor

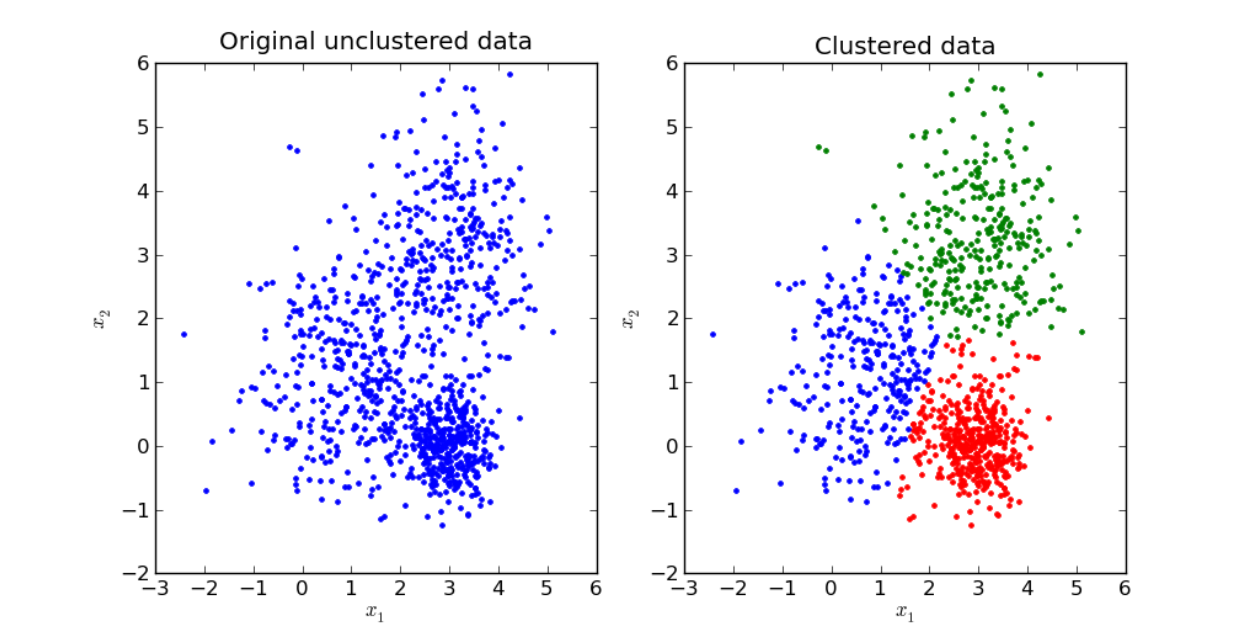
Iris Setosa

Iris Virginica

Sepal Length	Sepal Width	Petal Length	Petal Width	Species
5.1	3.5	1.4	0.2	I. Setosa
7.0	3.2	4.7	1.4	I. Versicolor
		⋮		

Example: Classification by Attributes: Input is $X = \mathbb{R}^4$ and output is a finite set $\mathcal{G} = \{\text{Virginica, Setosa, Versicolor}\}$.

Machine Learning Examples: Market Segmentation

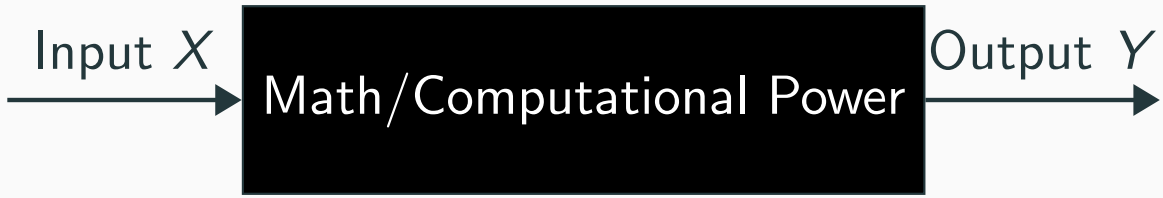


UserID	Products
0x131432	{"Leather Hat", "Dvvorsh Black Belt", "Dukes Hat Polish",...}
0x152341	{"Tinga XL Beach Towel", "Speco Meat Thermometer",...}

Example: Clustering: Input is $X = \mathbb{P}(\{\text{All Products}\})$, the powerset of the set of all products.

Machine Learning vs Statistics

Machine learning is used in problems where the underlying distributions are unknown or unknowable.



Main Question: For a given class of models, how many data points do we need in order to be certain (probabilistically) that the error in our model is small *no matter what the underlying distribution*.

Terminology and Notation

Two Types of Machine Learning

Supervised Learning: Given a sample set of *labeled* data, can we predict the labels on new unlabeled data from the same domain?

Examples: Image classification, classification by features.

Unsupervised Learning: Given a set of *unlabeled* data, can we find structure within the data?

Examples: Clustering, dimensional reduction.

Variable Types

There are two large families of variable types:

Qualitative Variables: Variables that only take discrete values, usually in a set of descriptive **classes**. Also called **categorical** or **discrete** variables, or **factors**. There could be no additional ordering or structure on the classes.

Examples: Iris name, "Cat" or "Dog", items purchased, the integers 0 to 9 in MNIST.

Quantitative Variables: Variables that take quantitative values, with some values larger than others and close values designating similar characteristics. Also called **numerical**.

Examples: Body temperature, grayscale value of a pixel, stem length, height.

Regression

Regression analysis is a set of procedures for estimating the relationships between independent variables (inputs) and dependent variables (outputs).

In HTF, **regression** specifically refers to prediction when the outputs are quantitative. If the outputs are qualitative, this is referred to as **classification**.

In SB, **regression** refers to any *functional* relationship between the domain of the inputs X and the range of the outputs Y, regardless of variable type.

This ambiguity can be found throughout the field.

Inputs and Output

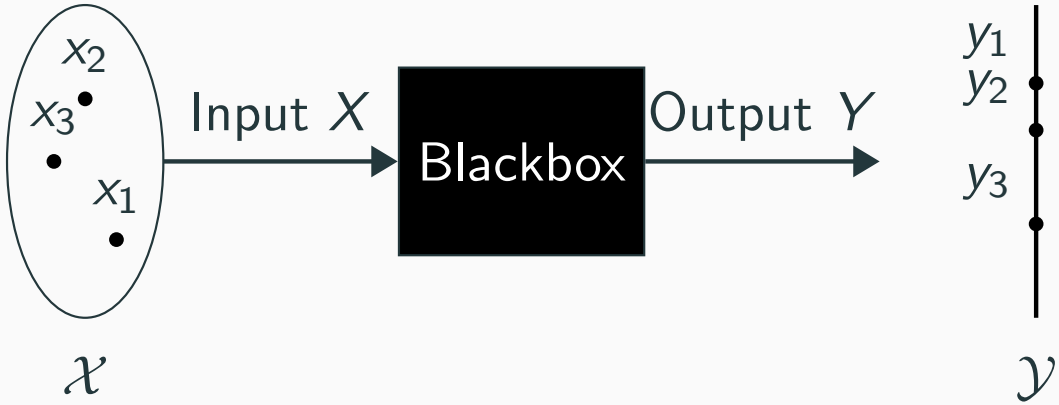
Input variables will be denoted by X. If X is a vector, its components will be X_j. The domain of these variables will be denoted $\mathcal{X}$.

In general, **output variables** will be denoted by Y. If we need to distinguish that the output is qualitative we will denote it by G. The domain of these variables will be $\mathcal{Y}$ and $\mathcal{G}$ respectively.

We use uppercase X, Y, G to refer to generic aspects of the variable and lowercase to denote observed values. For example, x_i denotes the i'th *observed* value of X and is a vector if X is.

All vectors are assumed to be column vectors.

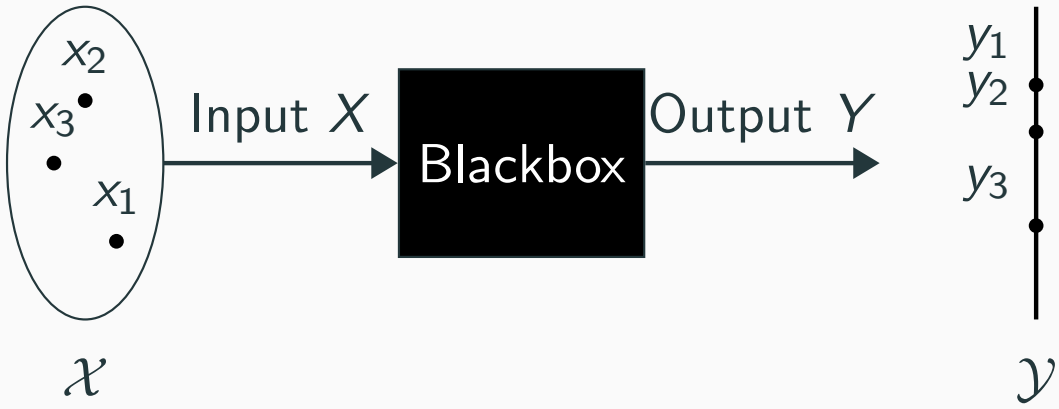
Inputs and Output



We assume that we have access to a set of N observations $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$, for $i = 1, \ldots, N$. This set of observations is called the **training set** or **training data**.

N will almost always denote the number of observed in points (data points in the training set).

Inputs and Output

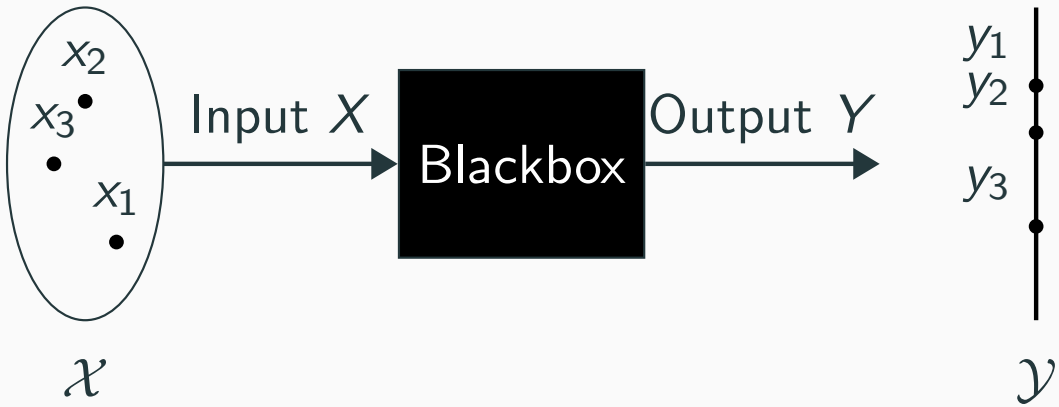


Matrices will be represented by bold uppercase letters. For example, a set of N input p vectors x_i, $i = 1, \ldots, N$ could be represented as $\mathbf{X}$.

We will only bold a vector when it has N components. This is to distinguish the p vector of inputs x_i from the N vector $\mathbf{x}_j$ consisting of all observations on the variable X_j.

Since all vectors are assumed to be column vectors, the i'th row of $\mathbf{X}$ is x_i^T.

Inputs and Output



Assume we have come to a regression function by analyzing the training data (x_i, y_i). The predicted output of the regression function will be denoted $\hat{Y}$.

First Example: Linear Regression

Linear Regression

The Problem: Given a vector of inputs $X^T = (X_1, \ldots, X_p)$, we want to predict the output Y via a linear model

$$\hat{Y} = \hat{\beta}_0 + \sum_{j=1}^{p} X_j \hat{\beta}_j \,.$$

The hats here denote fit constants.

If we include a constant variable 1 in X and write $\hat{\beta} = (\beta_0, \ldots, \beta_p)$, we can write

$$\hat{Y} = \hat{\beta}^T X$$

Linear Regression

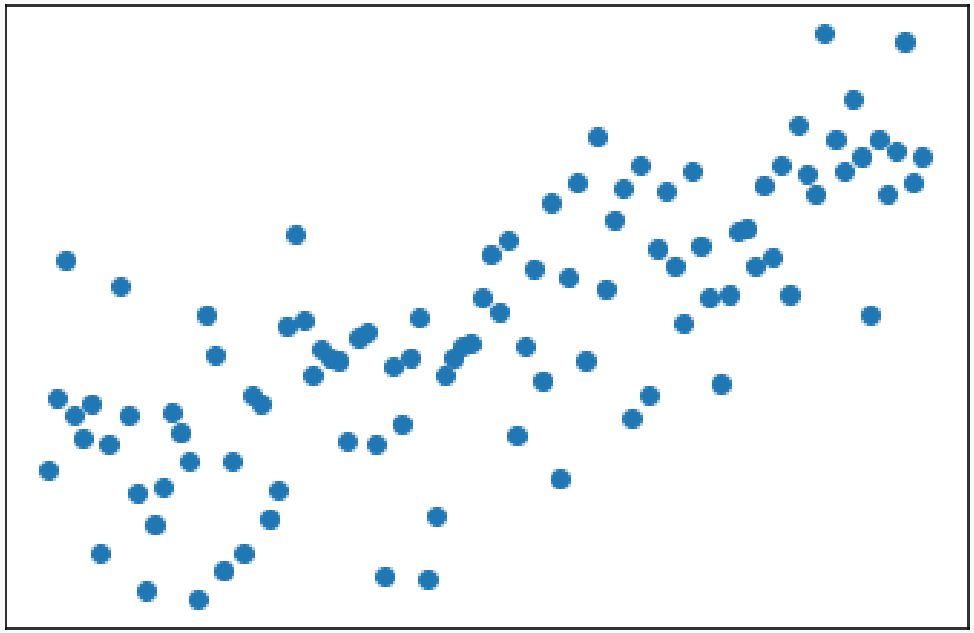
Note that $\hat{Y}$ need not be a scalar: if $\hat{Y}$ is a K vector then $\hat{\beta}$ is an $p \times K$ matrix

$$\begin{bmatrix} \hat{Y}_1 \\ \vdots \\ \hat{Y}_K \end{bmatrix} = \begin{bmatrix} \hat{\beta}_{0,1} \\ \vdots \\ \hat{\beta}_{0,K} \end{bmatrix} + X_1 \begin{bmatrix} \hat{\beta}_{1,1} \\ \vdots \\ \hat{\beta}_{1,K} \end{bmatrix} + \ldots + X_N \begin{bmatrix} \hat{\beta}_{p,1} \\ \vdots \\ \hat{\beta}_{p,K} \end{bmatrix}$$

$$\hat{Y} = \begin{bmatrix} \hat{\beta}_{0,1} & \cdots & \hat{\beta}_{0,p} \\ \vdots & \ddots & \vdots \\ \hat{\beta}_{0,K} & \cdots & \hat{\beta}_{p,K} \end{bmatrix} \begin{bmatrix} X_1 \\ \vdots \\ X_p \end{bmatrix}$$

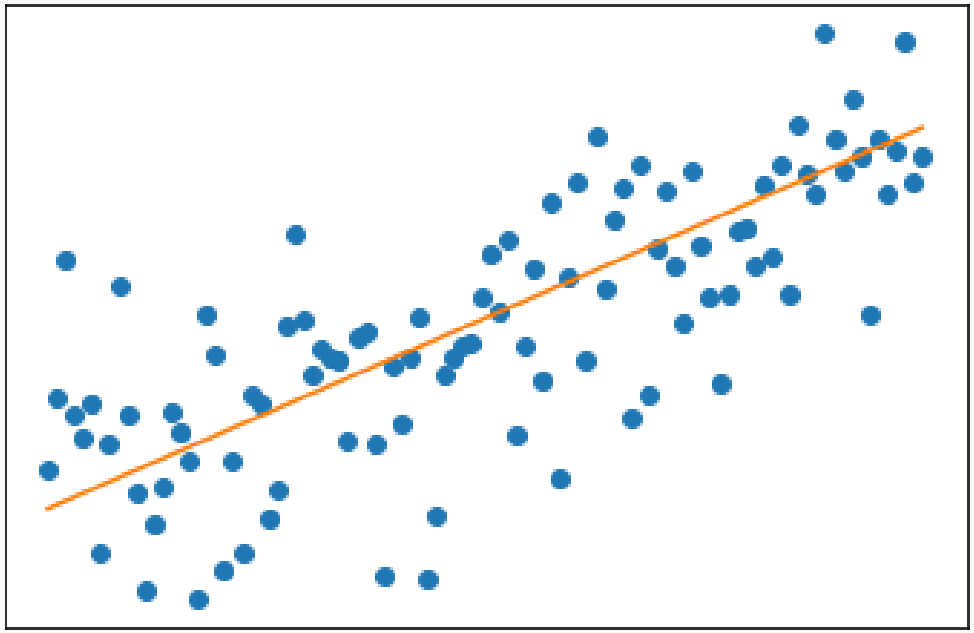
$$\hat{Y} = \hat{\beta}^T X$$

Linear Regression



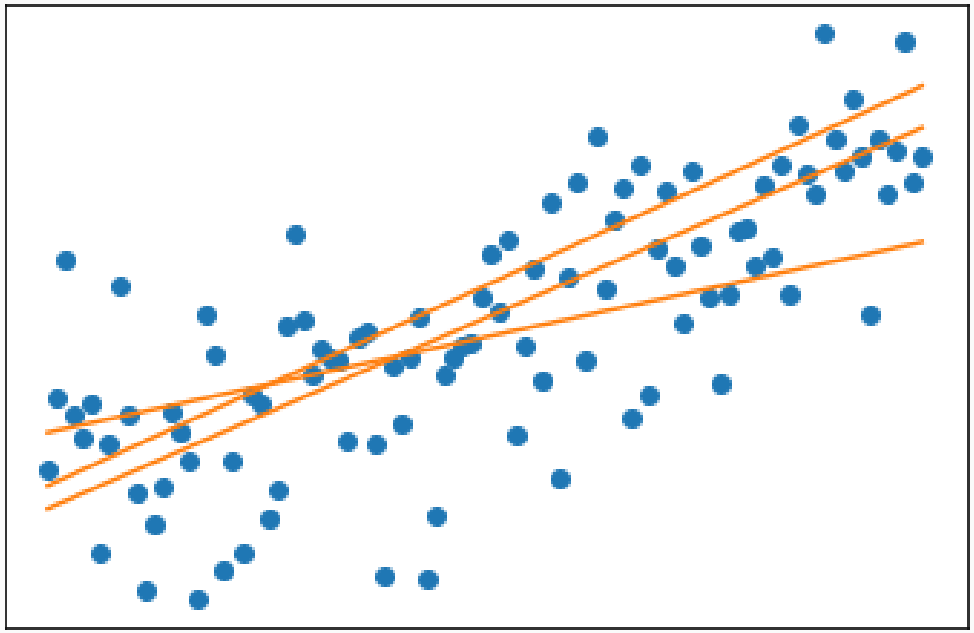
How do we find $\hat{\beta}$ given a specific set of data? That is, how do we **fit** the linear model to data points (x_i, y_i)?

Linear Regression



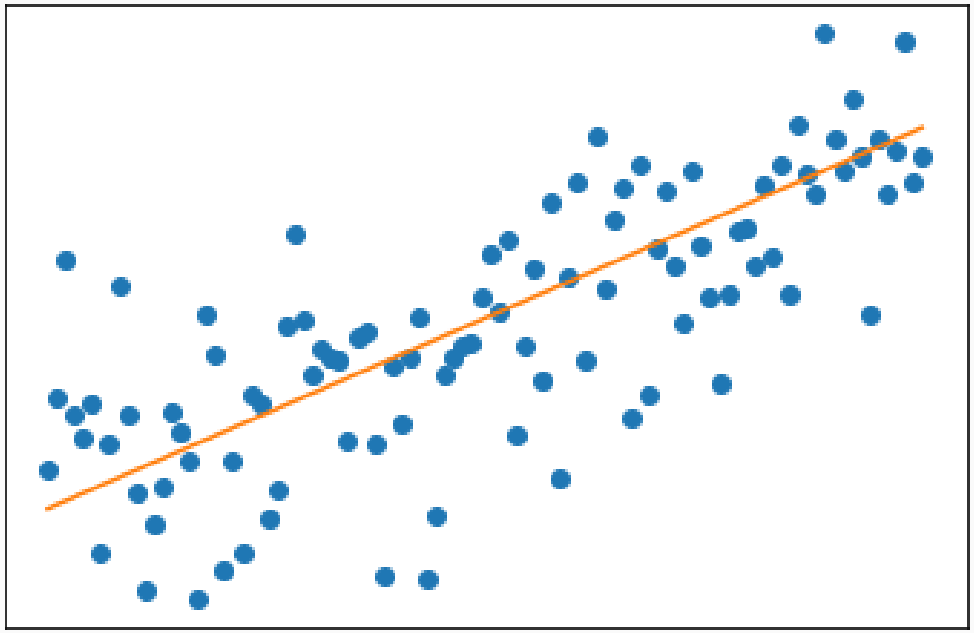
How do we find $\hat{\beta}$ given a specific set of data? That is, how do we **fit** the linear model to data points (x_i, y_i)?

Linear Regression



How do we find $\hat{\beta}$ given a specific set of data? That is, how do we **fit** the linear model to data points (x_i, y_i)?

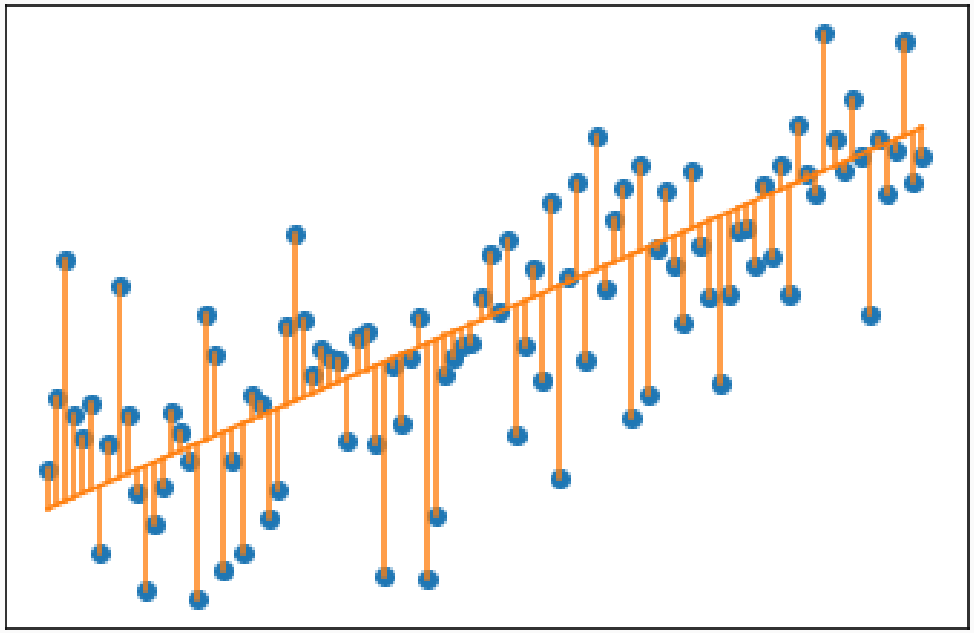
Linear Regression



There are many ways. The most popular is by minimizing the **residual sum of squares** over the training set

$$\text{RSS}(\beta) = \sum_{i=1}^{N} ||y_i - \beta^T x_i||^2 .$$

Linear Regression



There are many ways. The most popular is by minimizing the **residual sum of squares** over the training set. If Y is a scalar,

$$\text{RSS}(\beta) = \sum_{i=1}^{N} (y_i - \beta^T x_i)^2 \,.$$

With a little bit of work, we can write

$$\text{RSS}(\beta) = \sum_{i=1}^{N} (y_i - \beta^T x_i)^2$$

in matrix form as

$$\text{RSS}(\beta) = (\mathbf{y} - \mathbf{X}\beta)^T (\mathbf{y} - \mathbf{X}\beta)\,.$$

Proof: First,

$$\mathbf{y} - \mathbf{X}\beta = \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} - \begin{bmatrix} -x_1- \\ \vdots \\ -x_N- \end{bmatrix} \begin{bmatrix} \beta_0 \\ \vdots \\ \beta_p \end{bmatrix} = \begin{bmatrix} y_1 - \beta^T x_1 \\ \vdots \\ y_N - \beta^T x_N \end{bmatrix}\,.$$

Then

$$(\mathbf{y} - \mathbf{X}\beta)^T(\mathbf{y} - \mathbf{X}\beta) = \begin{bmatrix} y_1 - \beta^T x_1 & \dots & y_N - \beta^T x_N \end{bmatrix} \begin{bmatrix} y_1 - \beta^T x_1 \\ \vdots \\ y_N - \beta^T x_N \end{bmatrix}$$

$$= \sum_{i=1}^{N} (y_i - \beta^T x_i)^2$$

$$= \mathrm{RSS}(\beta)\,,$$

as claimed.

Linear Regression

Minimizing RSS(β) becomes a problem of calculus. The minimum will occur when the partial derivatives are 0:

$$0 = \frac{\partial}{\partial \beta_i} \text{RSS}(\beta)\,.$$

Homework: Show that the $p \times K$ derivatives

$$\frac{\partial}{\partial \beta} \text{RSS}(\beta) := \frac{\partial}{\partial \beta_{ij}} \text{RSS}(\beta)$$

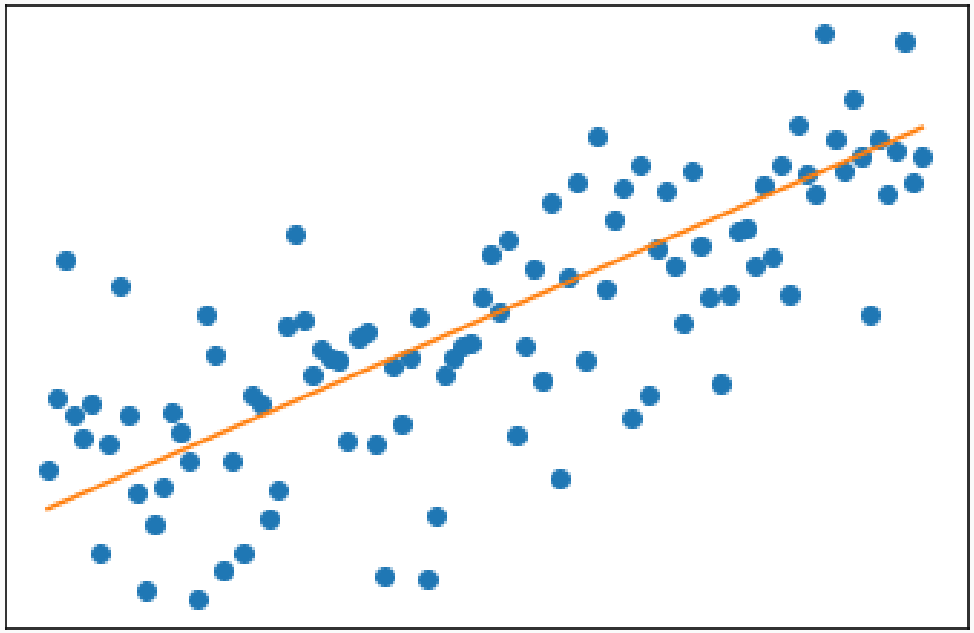
can be written compactly as

$$\frac{\partial}{\partial \beta} \text{RSS}(\beta) = -\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta) - (\mathbf{y} - \mathbf{X}\beta)^T\mathbf{X}\,.$$

Show that this implies that $\frac{\partial}{\partial \beta} \text{RSS}(\beta) = 0$ when

$$\mathbf{X}^T(\mathbf{y} - \mathbf{X}\beta) = 0\,.$$

Linear Regression



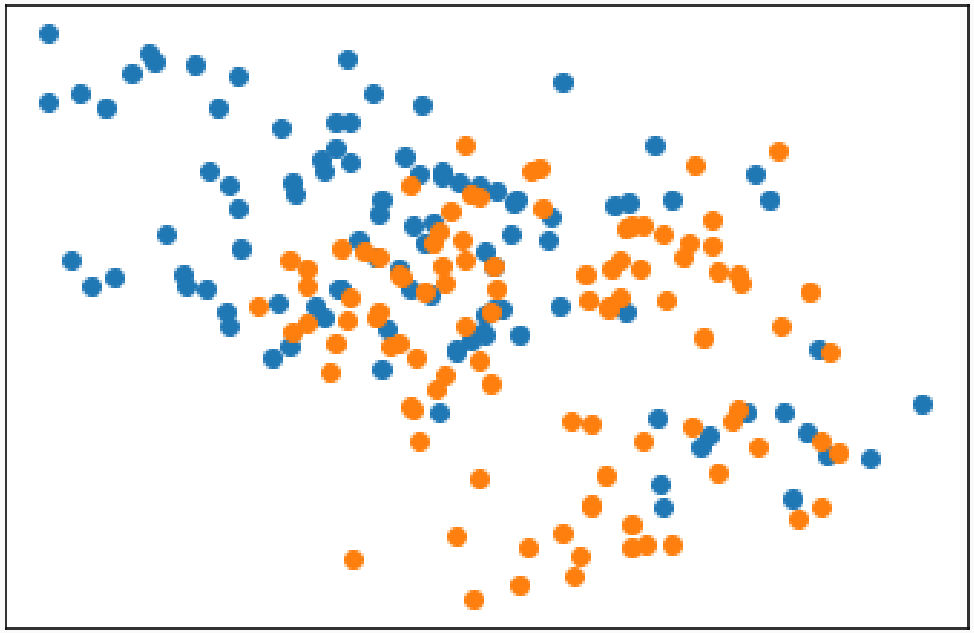
Solving $\mathbf{X}^T(\mathbf{y}-\mathbf{X}\beta) = 0$ for β yields an explicit equation for the parameters

$$\hat{\beta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}\,.$$

It doesn't seem to take much data to get a decent answer here, but we do have to invert a matrix. The solution is also relatively **stable** - it would take the addition of many more data points to dramatically change it.

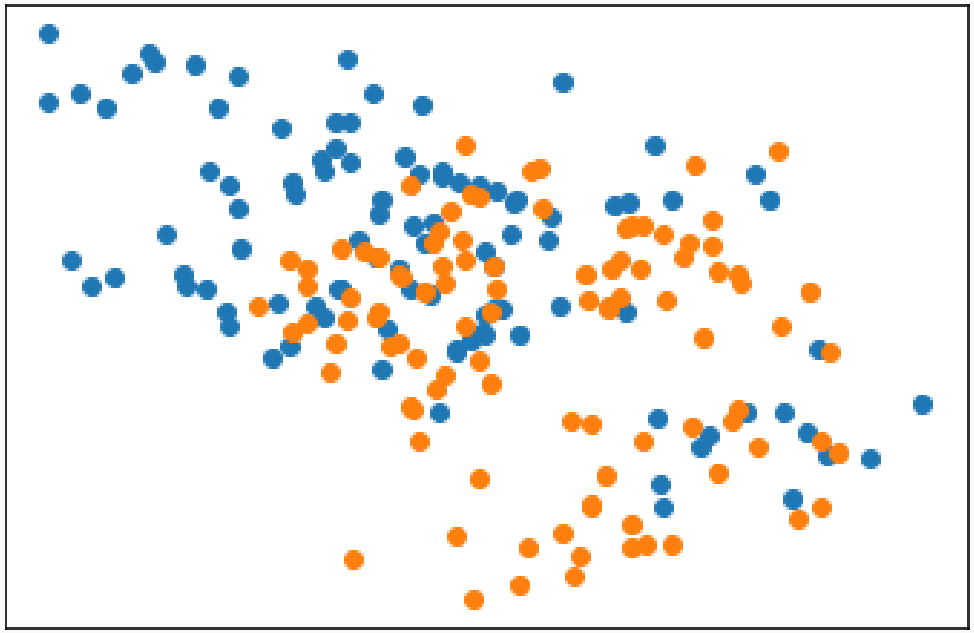
Second Example: Binary Classification

Binary Classification



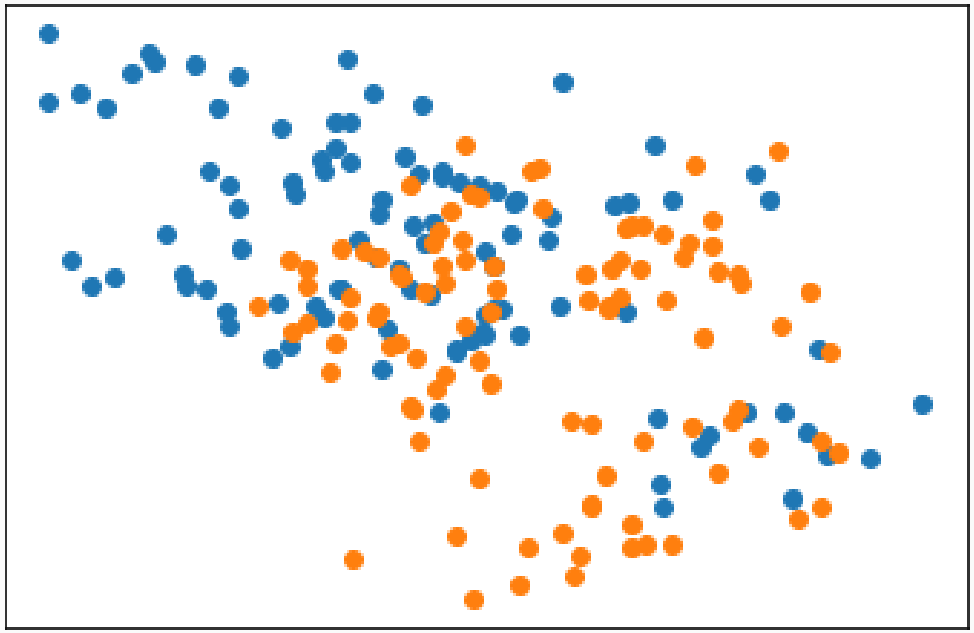
We have here a scatter plot of two colored training data in $\mathcal{X} = \mathbb{R}^2$. The inputs are X_1 and X_2 and the outputs Y take values in $\mathcal{G} = \{\text{Orange}, \text{Blue}\}$.

Binary Classification



For any point (X_1, X_2) we want to produce a probability $\hat{Y}$ that the point is labeled "Orange" and probability $1 - \hat{Y}$ that the point is labeled "Blue."

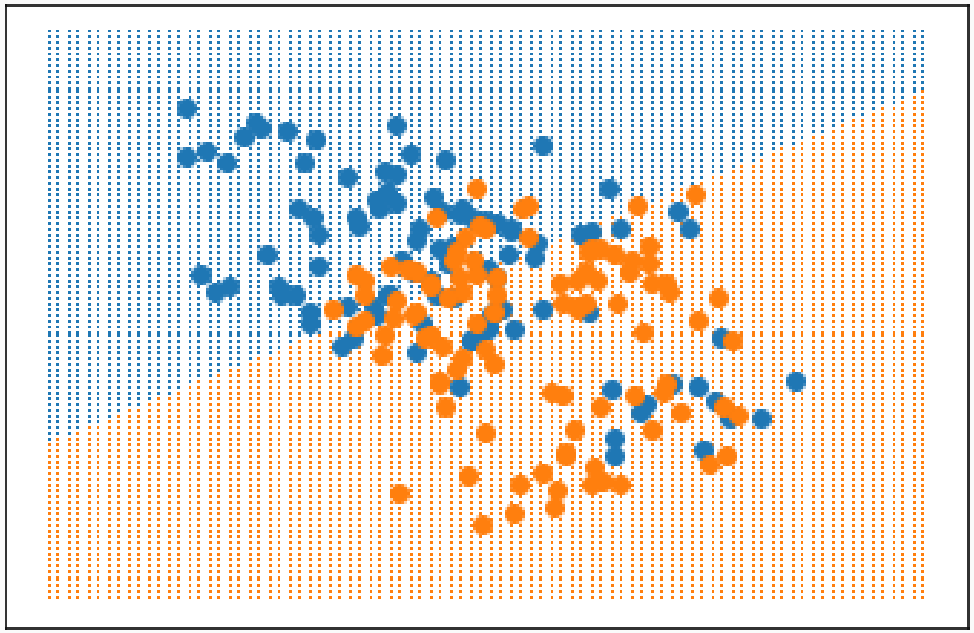
Binary Classification



In the binary case, we may minimize RSS(β) with $Y \in \{0, 1\}$. We assign a color to (X_1, X_2) by selecting

$$\hat{G} = \begin{cases} \text{Orange} & \text{if } \hat{Y} > .5 \\ \text{Blue} & \text{if } \hat{Y} \leq .5 \end{cases}$$

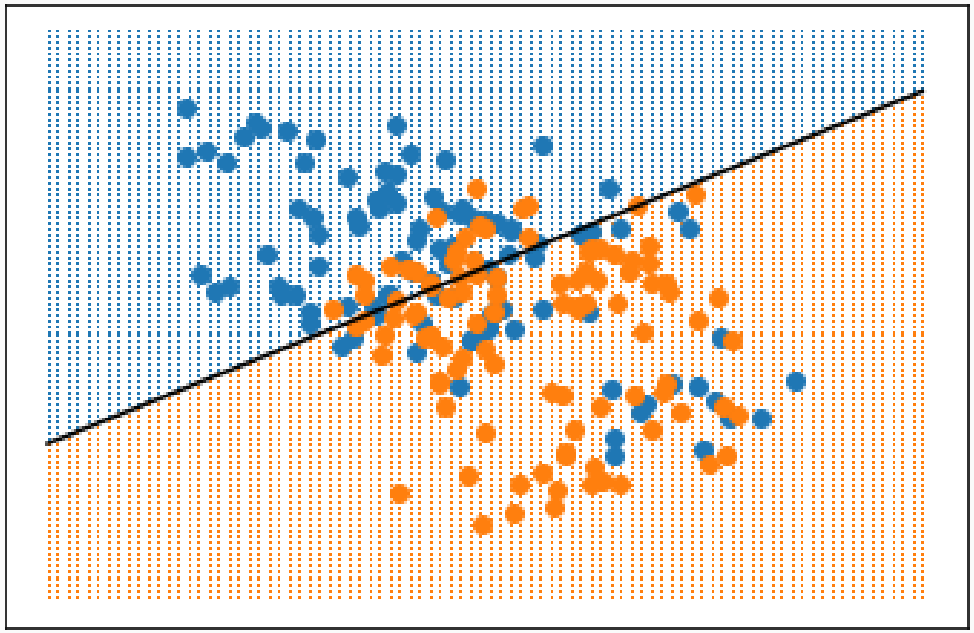
Binary Classification



In the binary case, we may minimize RSS(β) with $Y \in \{0, 1\}$. We assign a color to (X_1, X_2) by selecting

$$\hat{G} = \begin{cases} \text{Orange} & \text{if } \hat{Y} > .5 \\ \text{Blue} & \text{if } \hat{Y} \leq .5 \end{cases}$$

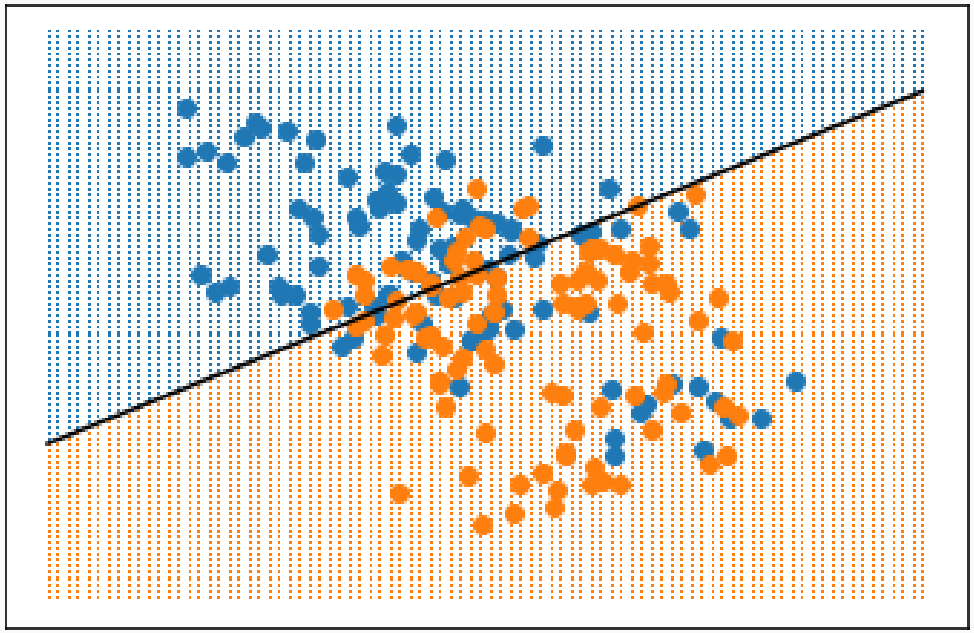
Binary Classification



In this case, the two classes are separated by a linear **decision boundary** $\{x : \hat{\beta} x^T = .5\}$.

Question: What do you think of this model? How could it be improved? Can we minimize the apparent errors?

Binary Classification

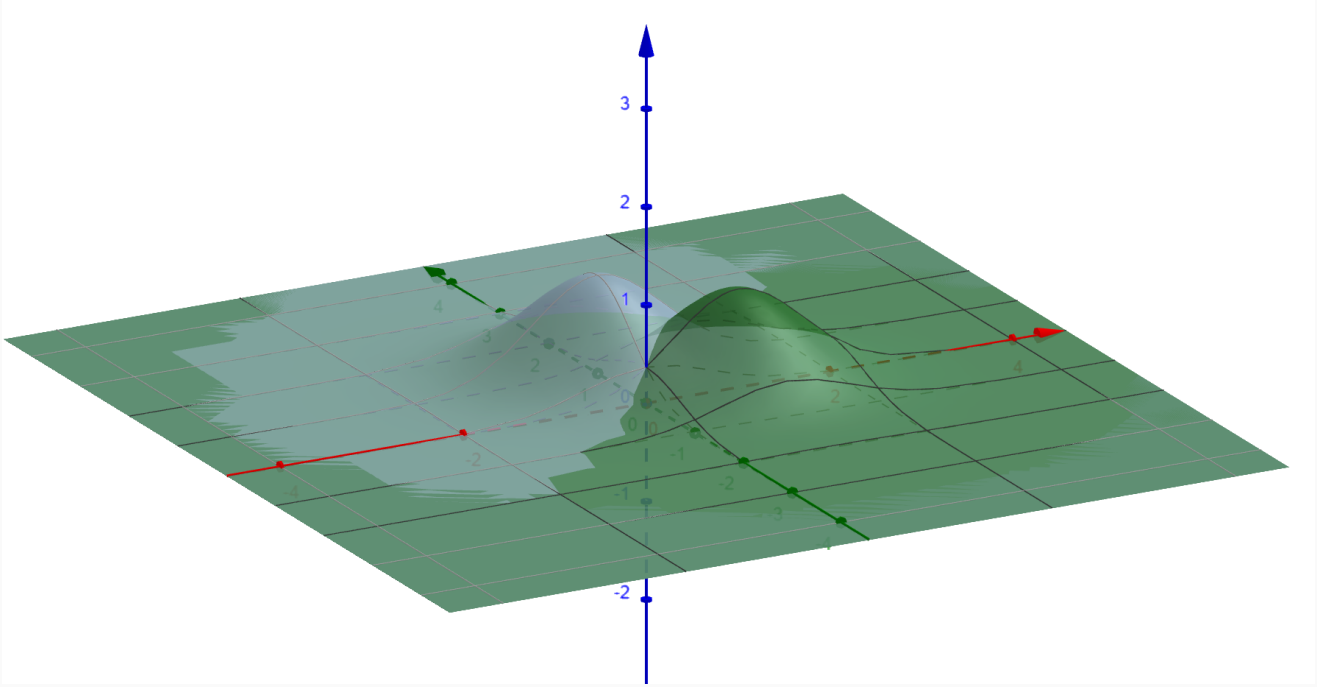


The theoretical accuracy of the linear classifier will depend on the mechanism of data generation. Consider the following two cases:

Scenario 1: The data was generated from a pair of multivariate normal distributions with different means.

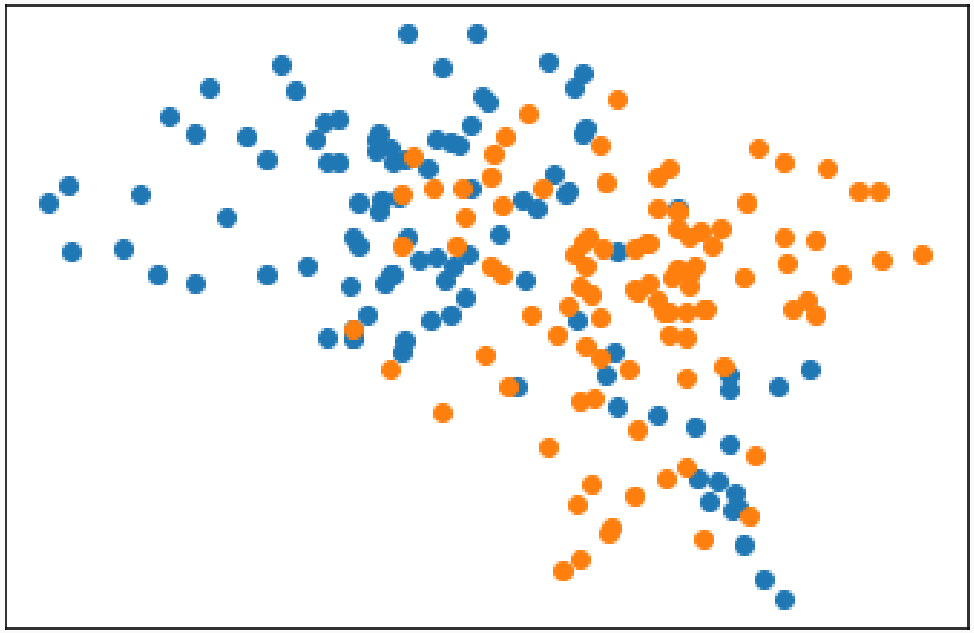
Scenario 2: The data was generated from many low variance distributions with normally distributed means.

Binary Classification



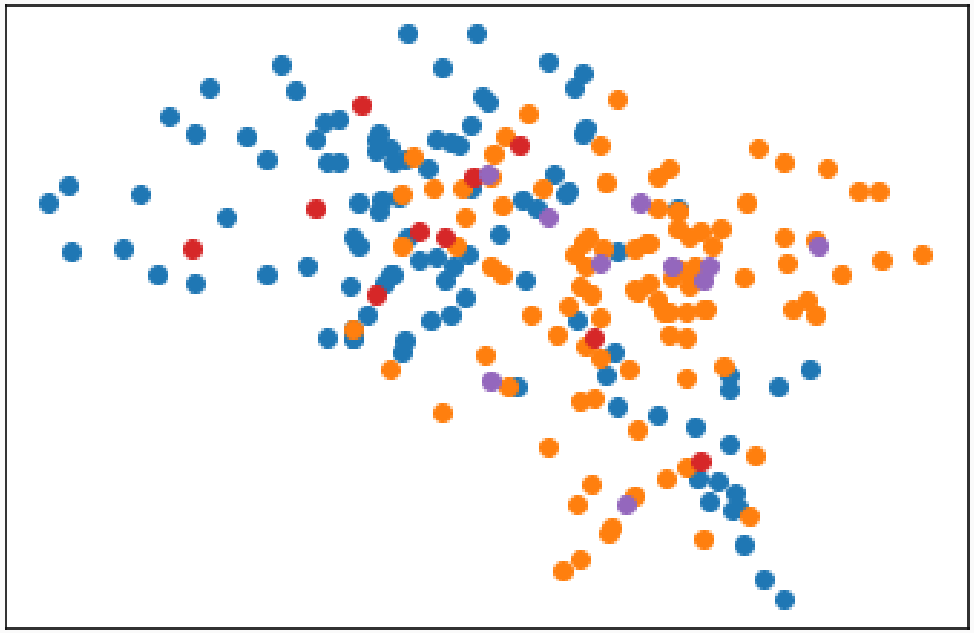
Scenario 1: If data was generated from a pair of multivariate normal distributions with different means, the linear classifier might be the best we can do.

Binary Classification



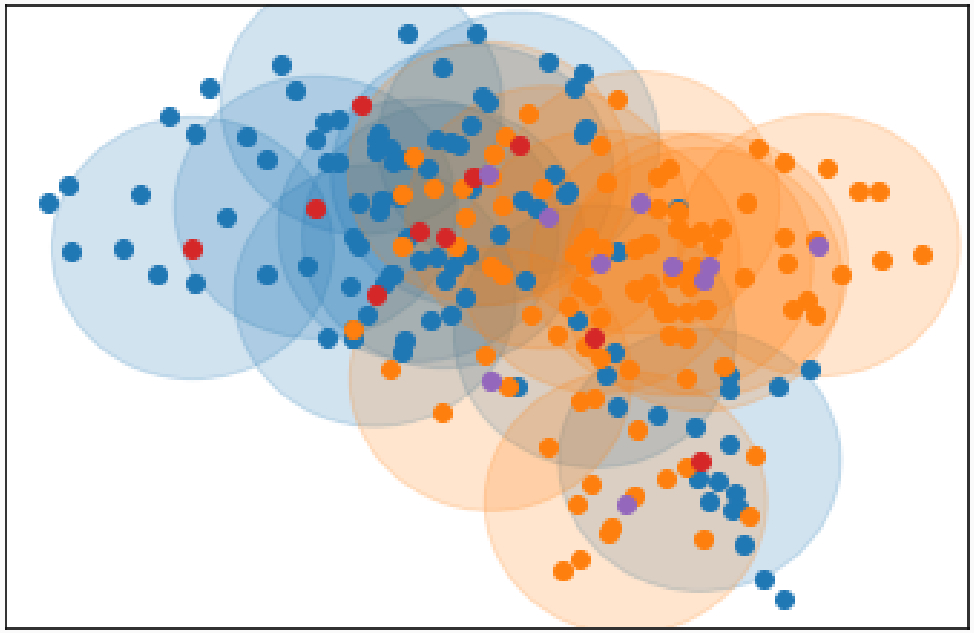
Scenario 2: If data was generated from many low variance distributions with normally distributed means, then the best classifier requires a much more nuanced approach.

Binary Classification



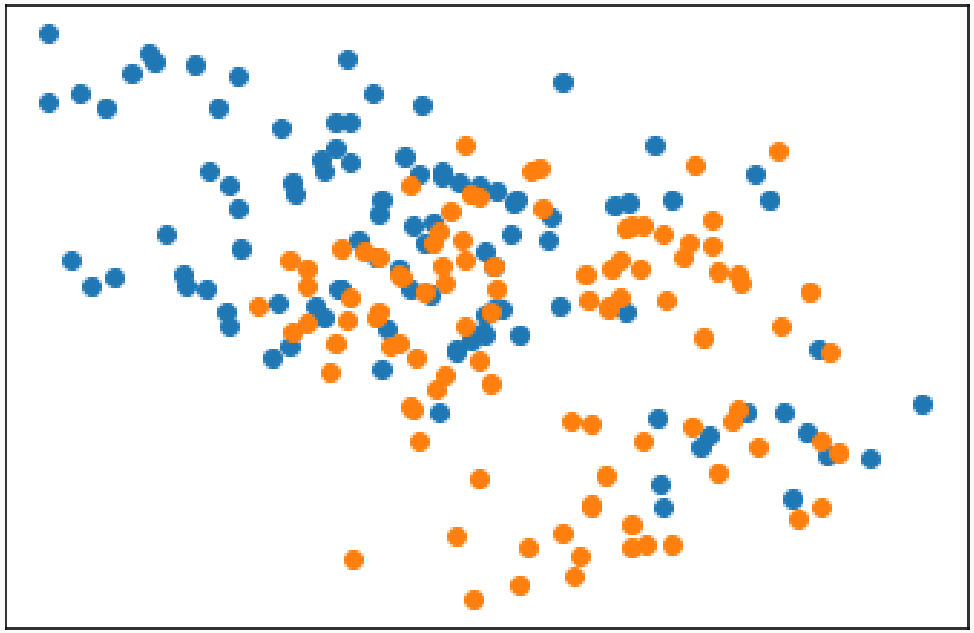
Scenario 2: If data was generated from many low variance distributions with normally distributed means, then the best classifier requires a much more nuanced approach.

Binary Classification



Scenario 2: If data was generated from many low variance distributions with normally distributed means, then the best classifier requires a much more nuanced approach.

Binary Classification: Nearest Neighbors

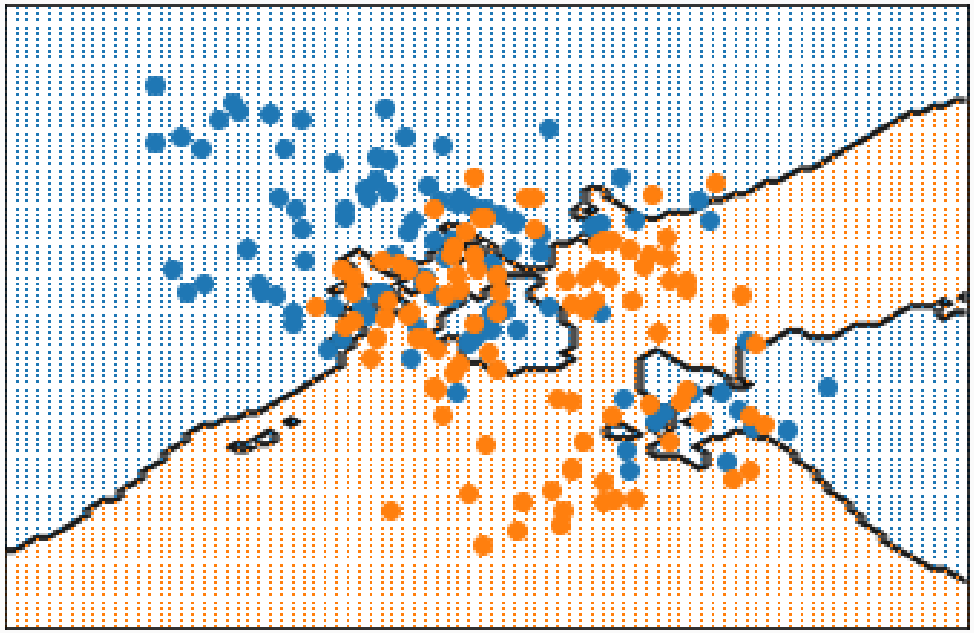


One way to complicate the boundary is the **k-nearest neighbors** method. For any pair of input values X, we determine a labeling by

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

where $N_k(x)$ are the k points in the training set closest to x.

Binary Classification: Nearest Neighbors

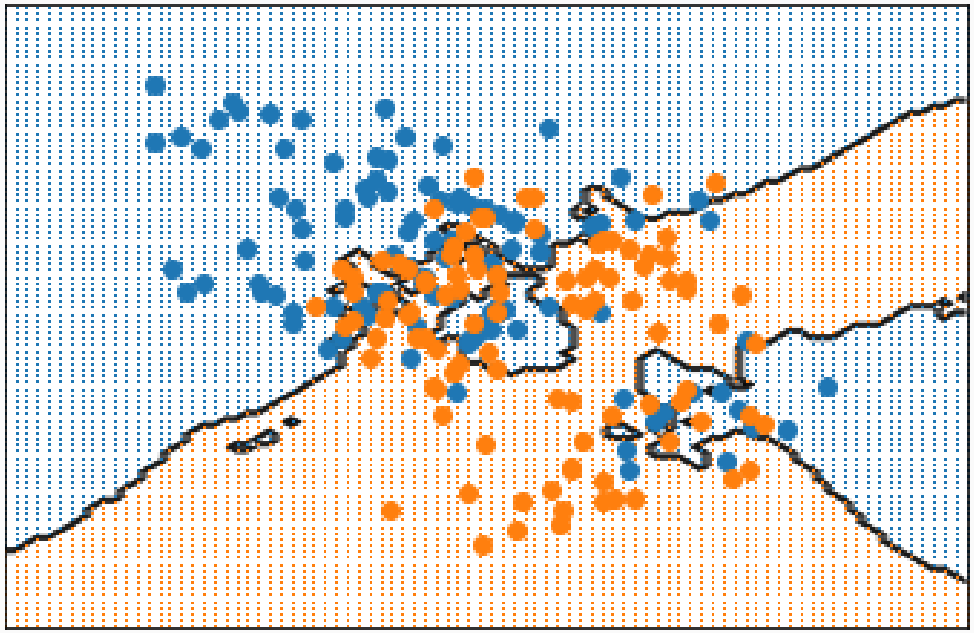


One way to complicate the boundary is the **k-nearest neighbors** method. For any pair of input values X, we determine a labeling by

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

where $N_k(x)$ are the k points in the training set closest to x.

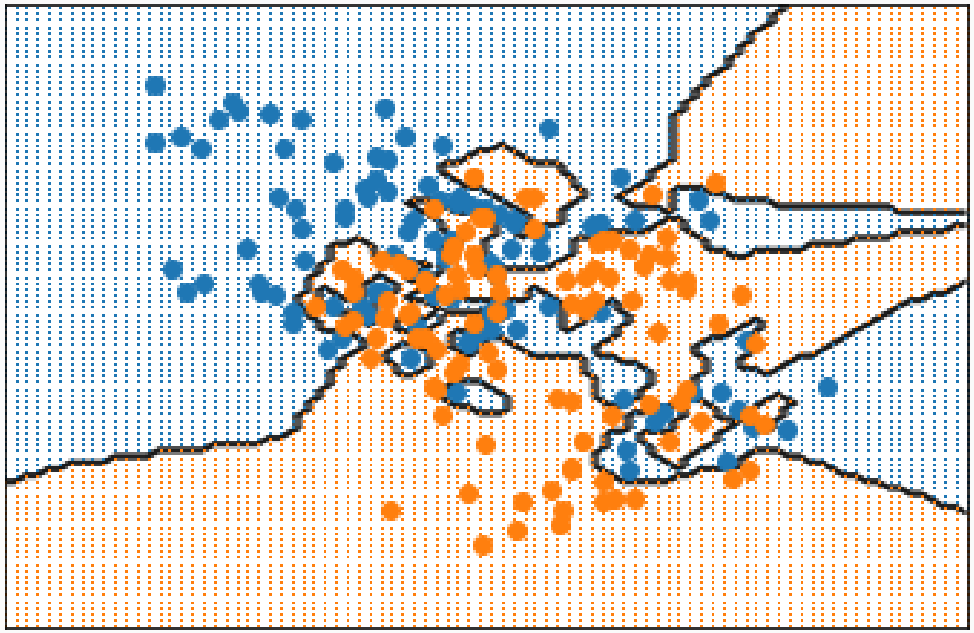
Binary Classification: Nearest Neighbors



Notice that the boundary is much more jagged in this case, but we also have fewer misclassifications of the training data. Such misclassifications are roughly the **training error**.

The k nearest neighbors algorithm yields our first example of **overfitting**. The training error is roughly an increasing function of k. When $k = 1$ the training error is 0. When $k = N$ the training error is roughly 50%.

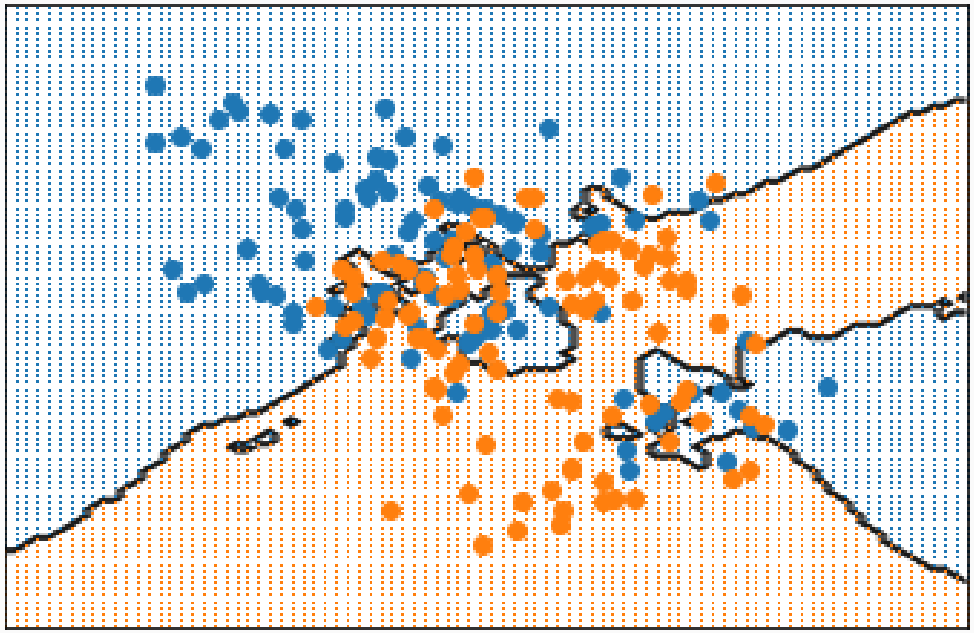
Binary Classification: Nearest Neighbors



$\mathbf{k = 1}$

When $k = 1$ the training error is 0. When $k = N$ the training error is roughly 50%.

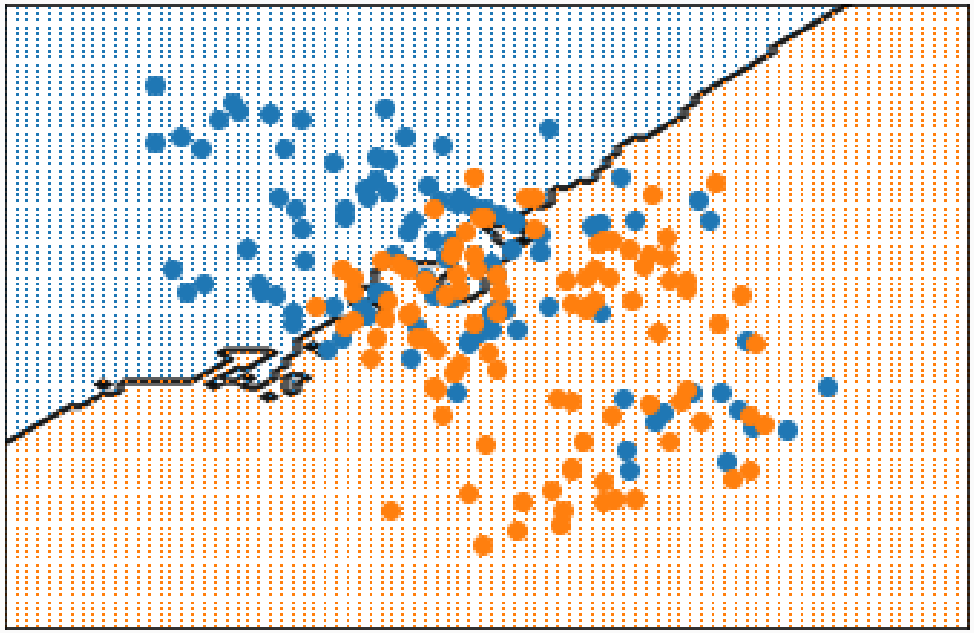
Binary Classification: Nearest Neighbors



$\mathbf{k = 10}$

When $k = 1$ the training error is 0. When $k = N$ the training error is roughly 50%.

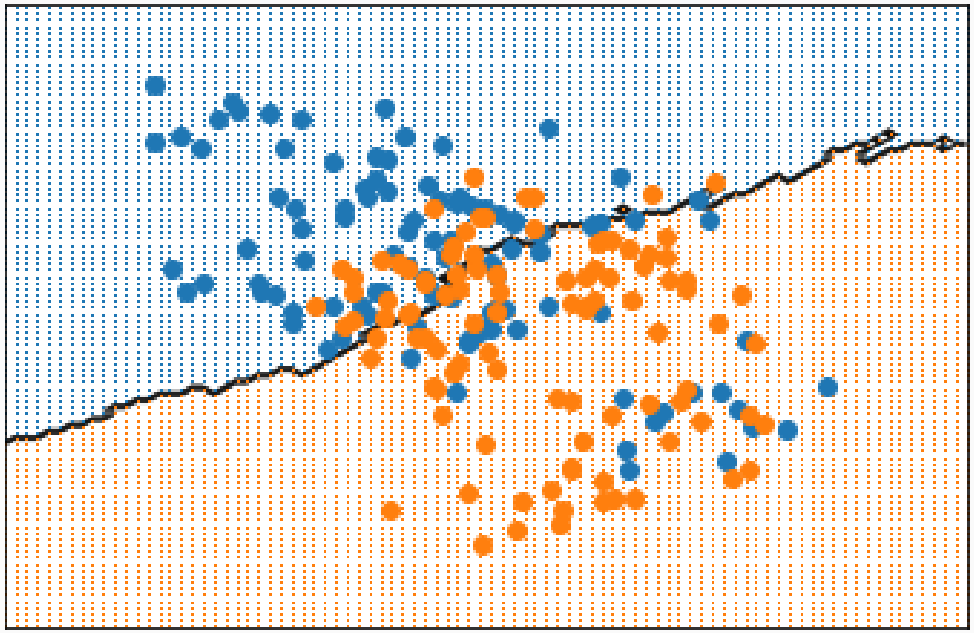
Binary Classification: Nearest Neighbors



k = 50

When $k = 1$ the training error is 0. When $k = N$ the training error is roughly 50%.

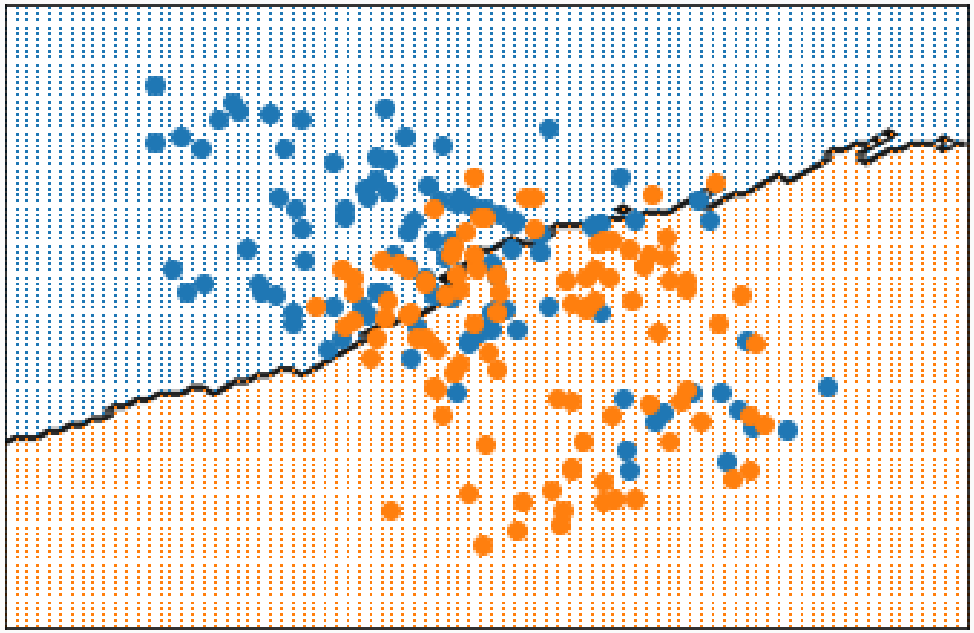
Binary Classification: Nearest Neighbors



$\mathbf{k = 100}$

When $k = 1$ the training error is 0. When $k = N$ the training error is roughly 50%.

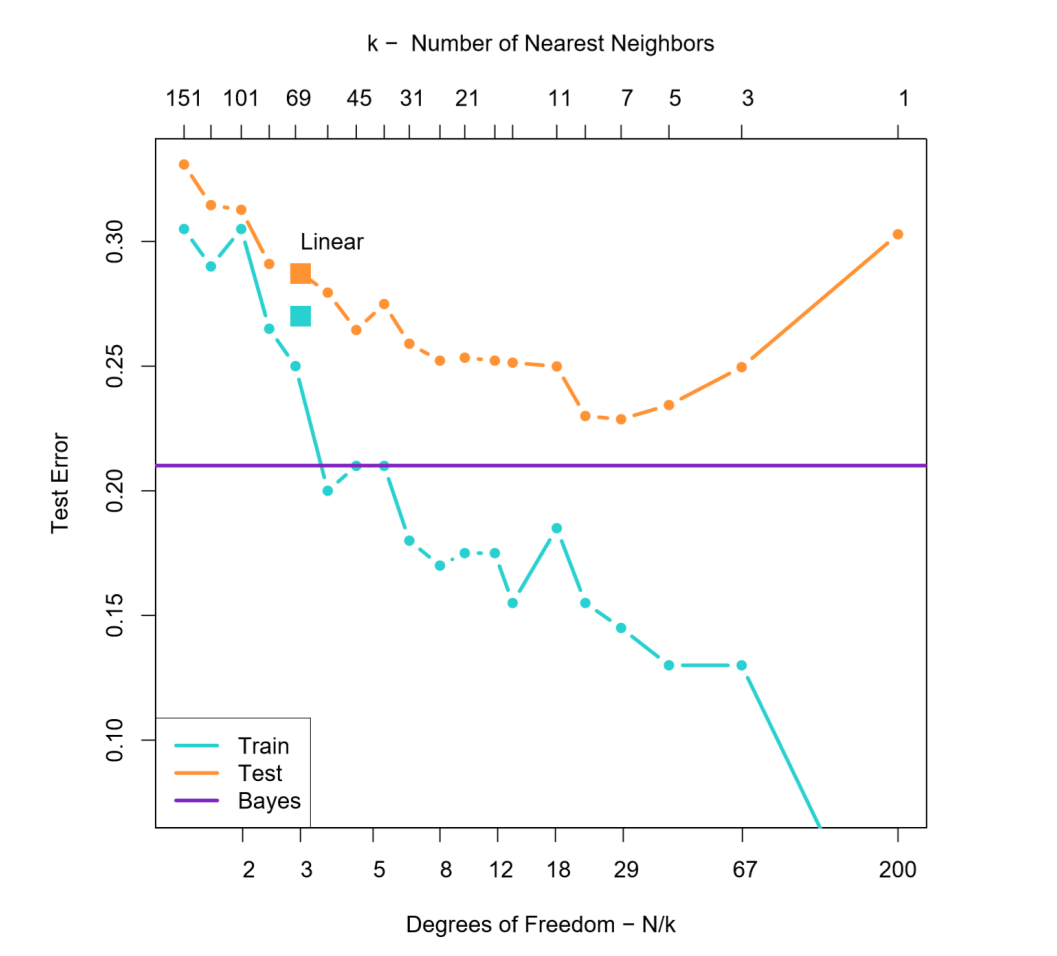
Binary Classification: Nearest Neighbors



$$\mathbf{k = 100}$$

It appears that k-nearest neighbors has a single parameter k, but we can't fit this parameter using lease squares since we will always just get $k = 1$.

Binary Classification: Comparison



Comparing the linear model to the k-nearest neighbors, we see that while the training error decreases as N/k increases, the true error is relatively high on both ends.

A Peek at Statistical Learning Theory

Statistical Learning Theory

Statistical learning theory is the study of principled decision making procedures for the choice of regression function.

Statistical learning theory lives properly in the world of random variables and probability spaces.

On the standard framework we to try to find a function that minimizes some loss function with respect to all available training data. But to make that method meaningful, we have to understand what loss minimization would imply for an arbitrary distribution on our data set.

Statistical Learning Theory

Let $X \in \mathbb{R}^p$ be a real random input vector, let $Y \in \mathbb{R}$ be a real random output vector, and let $P(X, Y)$ be the joint probability distribution. To find a function $f(X)$ that predicts Y we must define a loss function $L(Y, f(X))$.

As a first step, consider the **squared error loss** $L(Y, f(X)) = (Y - f(X))^2$. We want to find an f that minimizes the expectation value of the loss.

The expectation value of $L(Y, f(X)) = (Y - f(X))^2$ is

$$\begin{aligned} EPE(f) &= E(Y - f(X))^2 \\ &= \int [y - f(x)]^2 \Pr(dx, dy)\,, \end{aligned}$$

or the integral over the joint probability distribution.

Recalling that $\Pr(X, Y) = \Pr(Y|X)\Pr(X)$, we write

$$\begin{aligned} \int [y - f(x)]^2 \Pr(dx, dy) &= \int [y - f(x)]^2 \Pr(Y|X)\Pr(X)\,, \\ &= E_X\big(\, E_{Y|X}[Y - f(X)]\,\big) \end{aligned}$$

So

$$EPE(f) = E_X\big(E_{Y|X}[Y - f(X)] \big) .$$

If f can be arbitrary, it is sufficient to minimize at each x:

$$f(x) = \text{argmin}_c E_{Y|X}([Y - c]^2 | X = x) .$$

Since Y and c are independent, the minimum can be found by differentiating. Indeed,

$$0 = \frac{d}{dc} E_{Y|x}([Y - c]^2) = \frac{d}{dc}\Big(E_{Y|x}(Y^2) - 2cE_{Y|x}(Y) + c^2 \Big)$$

implies that

$$f(x) = E_{Y|X}(Y|X = x) .$$

The expression

$$f(x) = E_{Y|X}(Y|X = x)$$

is the **regression function**. When "best" is measured by squared average error, the "best prediction function" is the conditional mean.

Now suppose we choose $L(Y, f(X)) = |Y - f(X)|$. We find only one difference from the analysis about, namely that $E_{Y|X}(|Y - c||X = x)$ is minimized when $c = \text{median}(Y)$. Why?

Statistical Learning Theory: k-Nearest Neighbors

The k-nearest neighbor method attempts to directly implement

$$f(x) = E_{Y|X}(Y|X = x)\,.$$

At each point x, it estimates $f(x)$ by averaging over all y_i with input x_i. Since often there are no x_i, it average over the closest points:

$$\hat{f}(x) = \text{Ave}(\, y_i \,|\, x_i \in N_k(x))\,.$$

Here, expectation is estimated by averaging over **sample data** and minimize at x is estimated by minimizing "near" x.

Given enough training data this is a pretty good, if locally constant, estimate. But there are computational problems on the horizon.

Statistical Learning Theory: Linear Regression

For linear regression, we assume that

$$f(x) = E_{Y|X}(Y|X = x) \approx \beta^T X$$

is approximately linear. This is known as a **model based** approach: we restrict the class of $f(x)$ by specifying its form, but as a result we can estimate $f(x)$ as before.

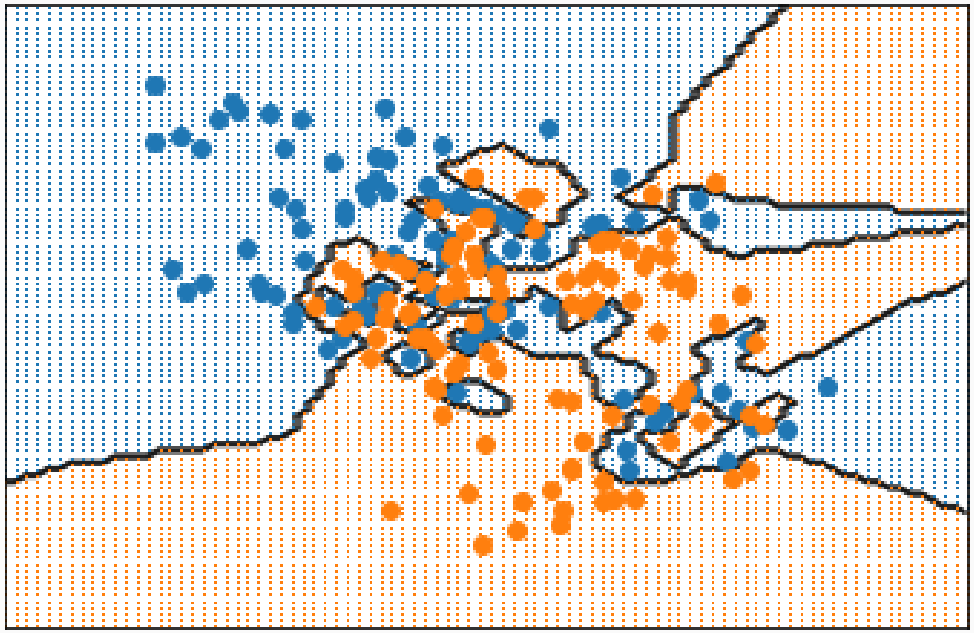
We could of course specify more complicated models, for example an additive model like

$$f(X) = \sum_{j=1}^{p} f_j(X_j)\,.$$

In general, how does restricting $f(x)$ to a class of models effect trainability? Fit?

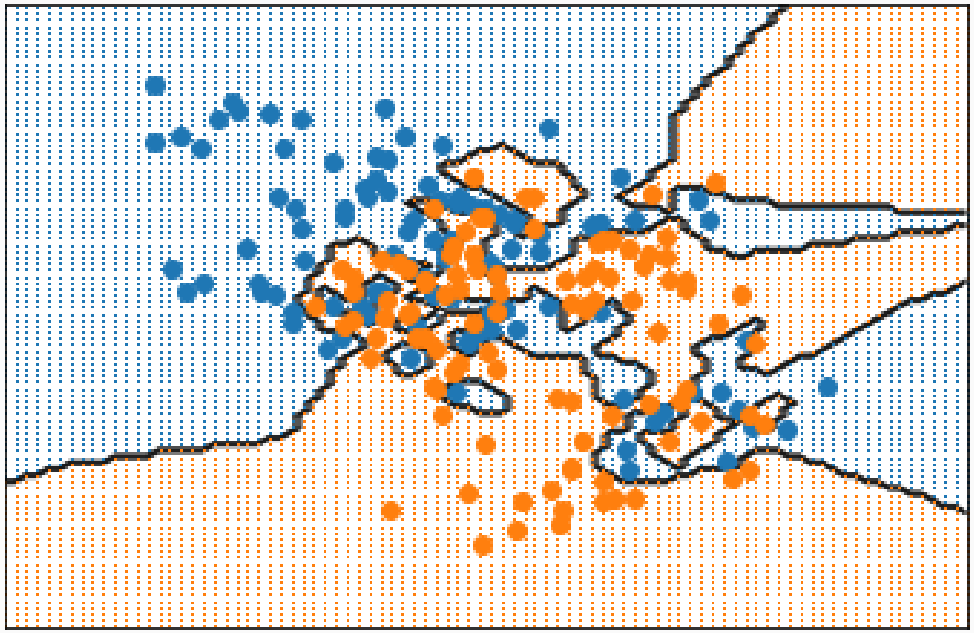
Bias-Variance Trade Off

Bias-Variance Trade Off



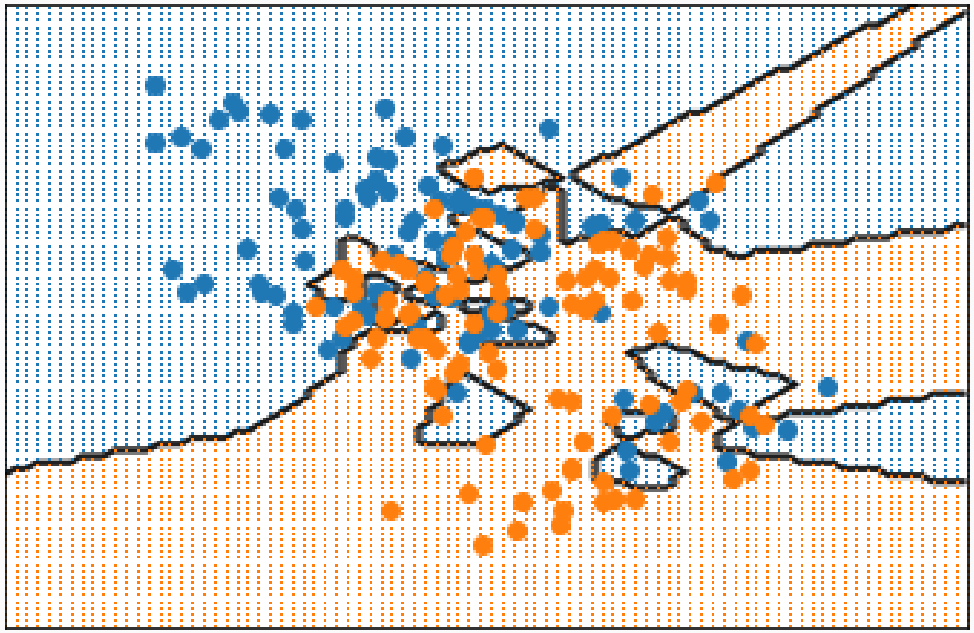
There is another disadvantage to using 1 nearest neighbors: it is very sensitive to the choice of training data.

Bias-Variance Trade Off



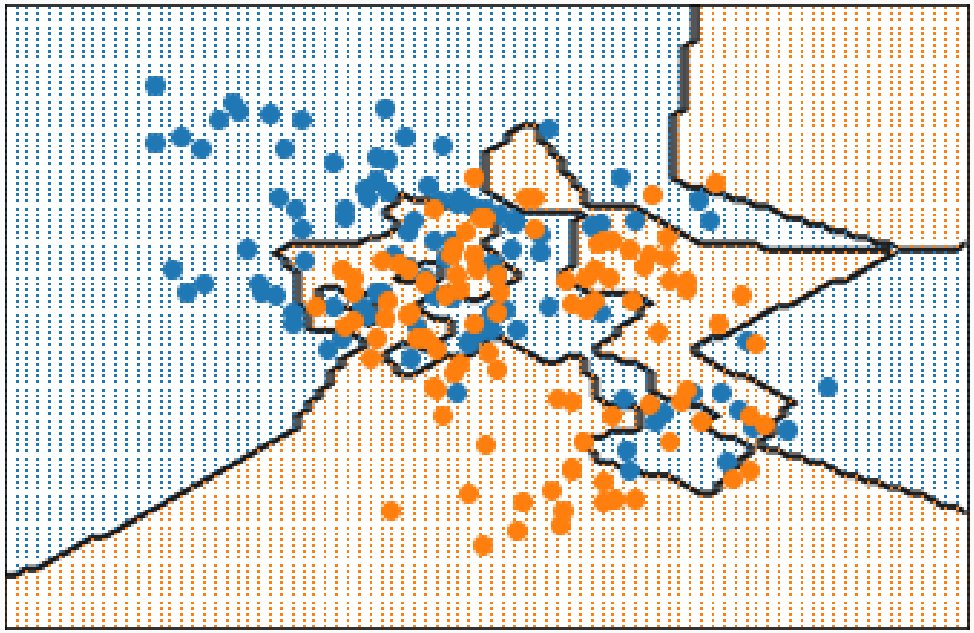
Taking two samples of the training data, 1NN produces two drastically different regressions, especially in the region with the most points.

Bias-Variance Trade Off



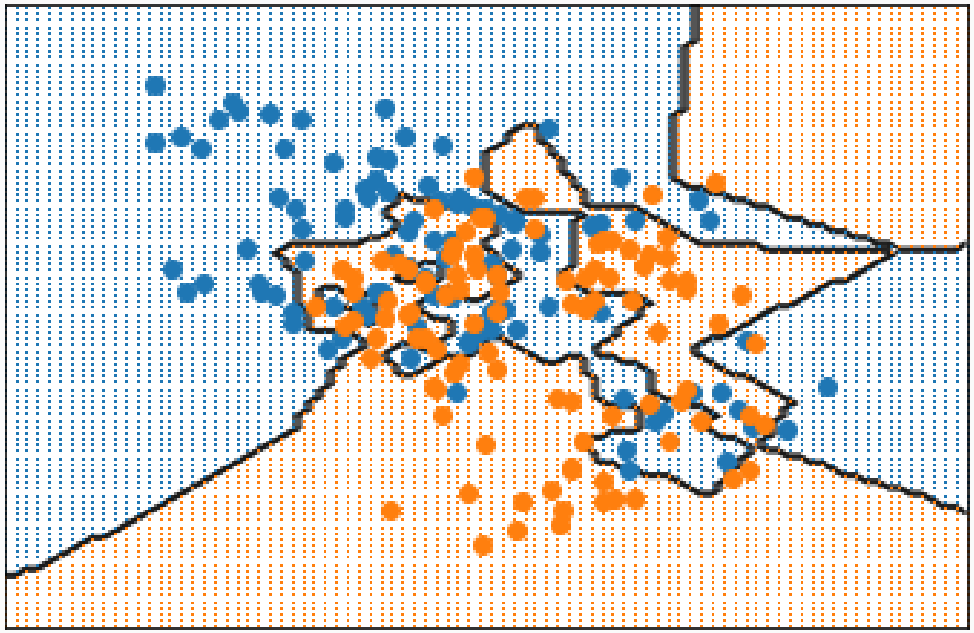
Taking two samples of the training data, 1NN produces two drastically different regressions, especially in the region with the most points.

Bias-Variance Trade Off



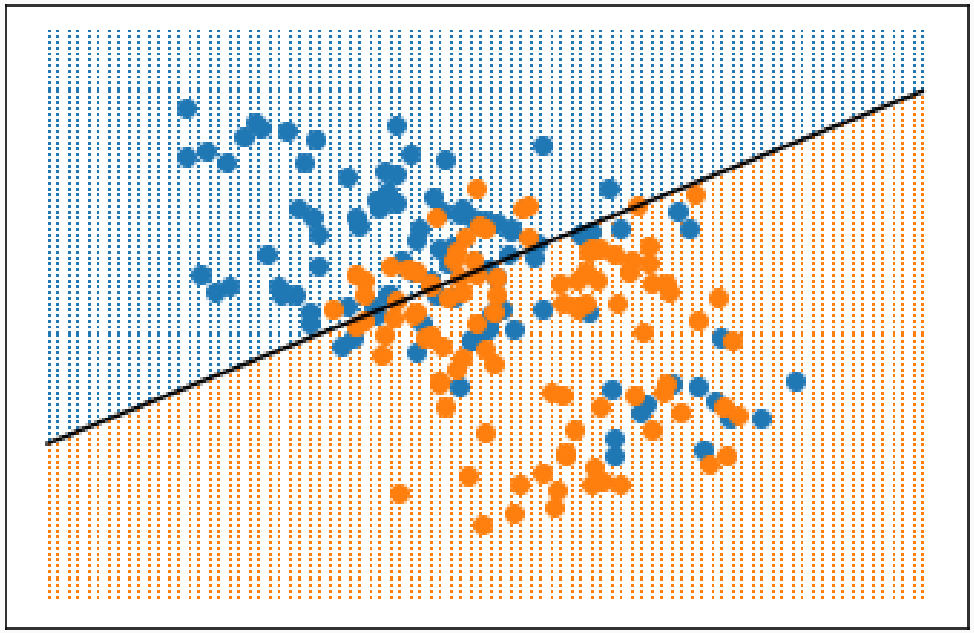
Taking two samples of the training data, 1NN produces two drastically different regressions, especially in the region with the most points.

Bias-Variance Trade Off



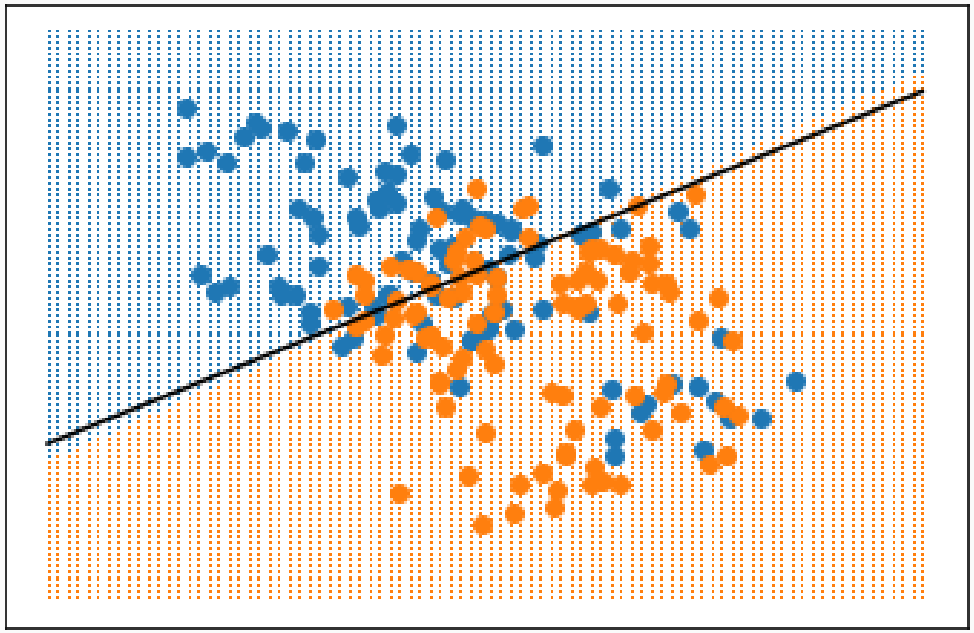
This sensitivity to training data is known as the variance of the model.

Bias-Variance Trade Off



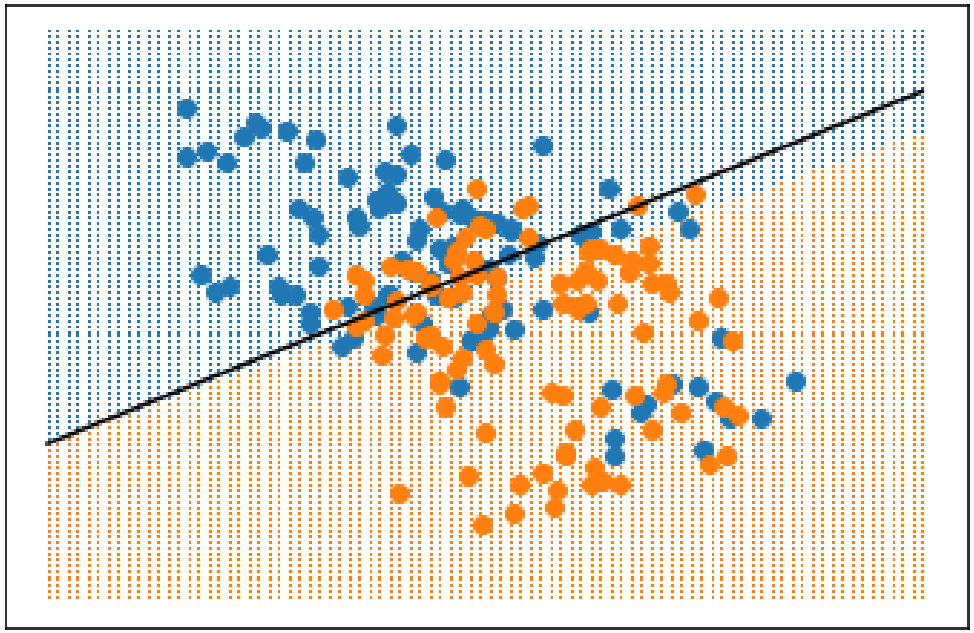
By contrast, if we take the same half of the training data and recompute the linear classifier the change in the fit is very low, especially in the region with the most data points.

Bias-Variance Trade Off



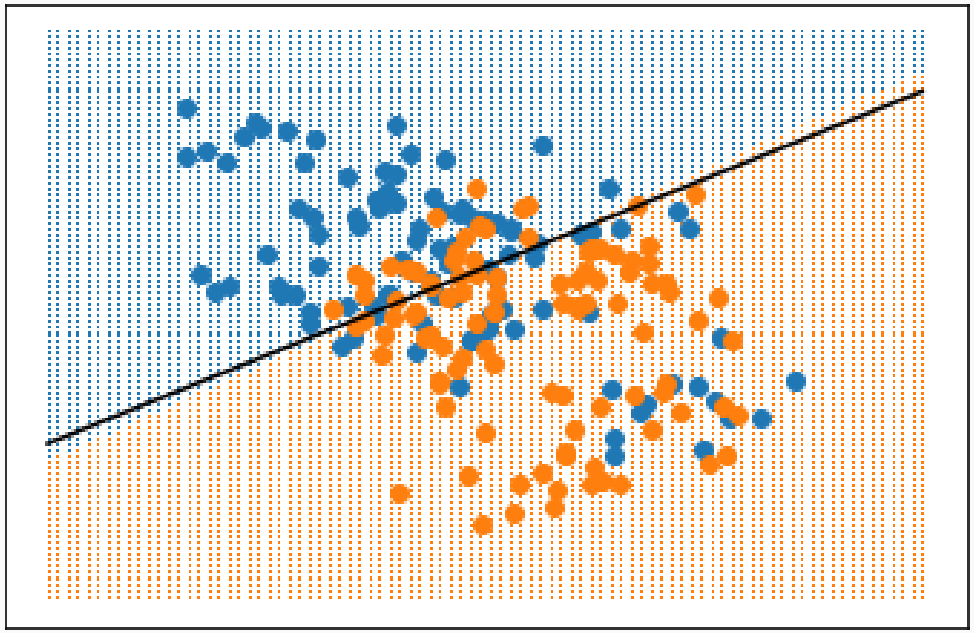
By contrast, if we take the same half of the training data and recompute the linear classifier the change in the fit is very low, especially in the region with the most data points.

Bias-Variance Trade Off



By contrast, if we take the same half of the training data and recompute the linear classifier the change in the fit is very low, especially in the region with the most data points.

Bias-Variance Trade Off

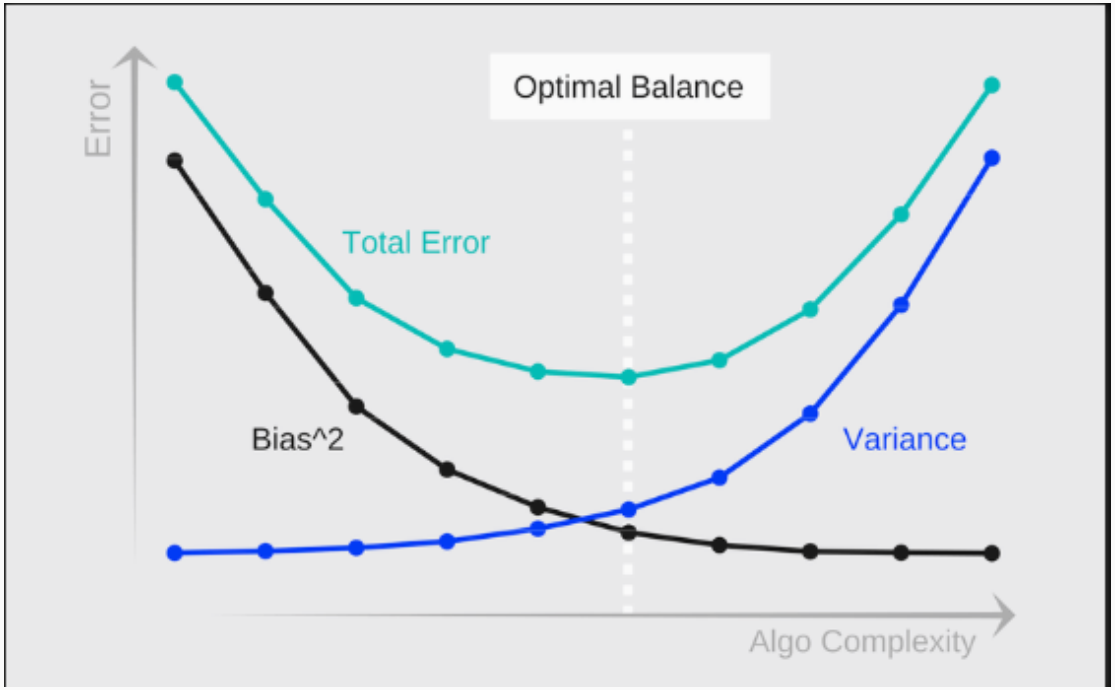


This contrast between the two algorithms is know as the **bias-variance trade off**.

For a class of models, the **bias** roughly is the expected error of the "best" classifier in the model class given a random set of training data.

The **variance** is roughly the sensitivity of the model to the training data.

Bias-Variance Trade Off

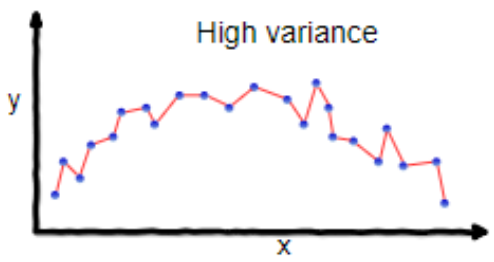


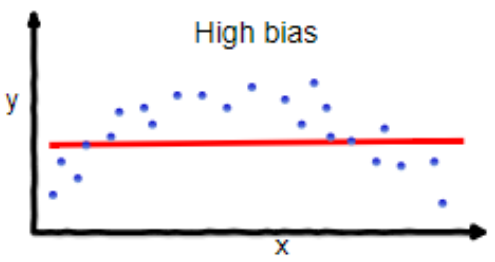
The idea is that the total error Err can be split into three parts:

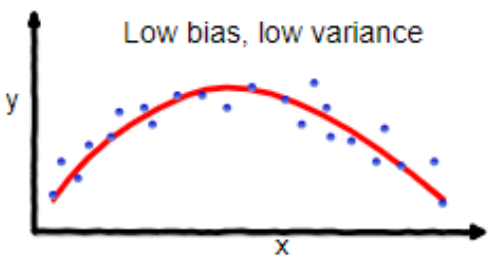
$$\text{Err} = (\text{Bias})^2 + \text{Var} + \text{Irreducible Error}$$

The best fit lies somewhere in between the extreme ends.

Bias-Variance Trade Off







overfitting **underfitting** **Good balance**

The idea is that the total error Err can be split into three parts:

$$\text{Err} = (\text{Bias})^2 + \text{Var} + \text{Irreducible Error}$$

The best fit lies somewhere in between the extreme ends. We will make this rigorous in the coming weeks.

Categorical Output and the Bayes Classifier

Bayes Classifier

If the output is a categorical variable G, we can give an analysis similar to the continuous case.

If the output has $|\mathcal{G}| = K$ classes, the loss function can be represented by a $K \times K$ matrix $\mathbf{L}$, where $L(k, \ell)$ is the cost of misclassifying an observation of $\mathcal{G}_k$ as $\mathcal{G}_\ell$.

The expected prediction error is

$$EPE = E[\, L(G, \hat{G}(X))] \,,$$

which we can again condition to write

$$EPE = E_X \left(\sum_{k=1}^{K} L(\mathcal{G}_k, \hat{G}(X)) \cdot \Pr(\mathcal{G}_k | X) \right) .$$

Bayes Classifier

We can minimize

$$EPE = E_X \left(\sum_{k=1}^{K} L(\mathcal{G}_k, \hat{G}(X)) \cdot \Pr(\mathcal{G}_k|X) \right) .$$

pointwise by finding

$$\hat{G}(x) = \text{argmin}_{g \in \mathcal{G}} \sum_{k=1}^{K} L(\mathcal{G}_k, \hat{G}(X)) \cdot \Pr(\mathcal{G}_k|X = x) .$$

If **L** is 0 on the diagonal and 1 elsewhere, this minimizer becomes

$$\hat{G}(x) = \text{argmin}_{g \in \mathcal{G}} \, 1 - \Pr(\mathcal{G}_k|X = x) ,$$

or

$$\hat{G}(x) = \max_{g \in \mathcal{G}} \Pr(\mathcal{G}_k|X = x) .$$

That is, $\hat{G}(x)$ is just the highest probability label for x.

Bayes Classifier

The function

$$\hat{G}(x) = \max_{g \in \mathcal{G}} \Pr(\mathcal{G}_k | X = x) \, .$$

is called the **Bayes classifier**. The error rate of the Bayes classifier is called the **Bayes rate**.

The Bayes classifier may not be the most desirable classifier for discrete data: Imagine that you need to detect cancer cells in an MRI scan. A false positive or two may not matter very much, but a false negative could have devastating consequences.

In that case, the 1-0 loss function would not be appropriate, and so the minimizing function would not be the Bayes classifier.

The Curse of Dimensionality

The Curse of Dimensionality

Of our two models, we found that the linear model was stable, but very biased. On the other, the k-nearest neighbors is unstable but much less biased. Since stability can be overcome with a large enough data set, why don't we always use k-nearest neighbors?

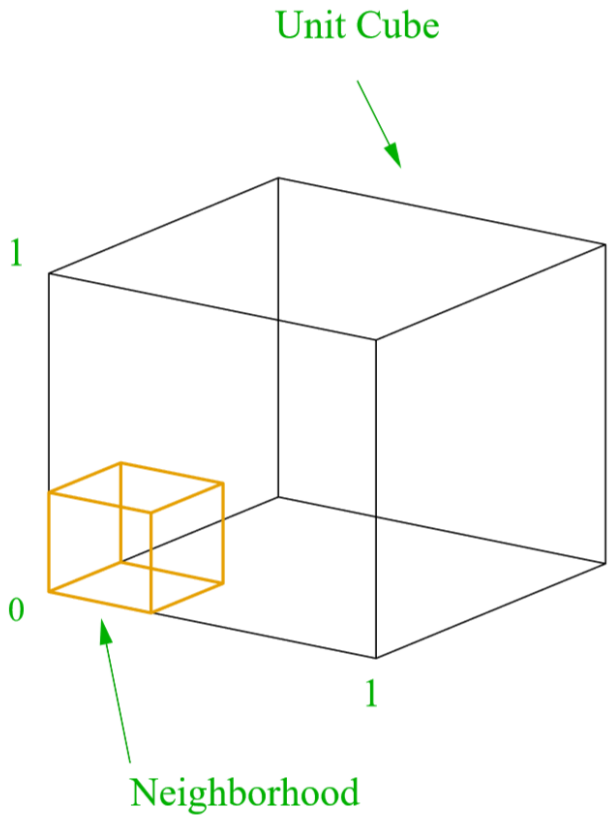
Geometry in higher dimensions often behaves counter-intuitively, and as the dimension of our input vector gets large things, computational problems arise.

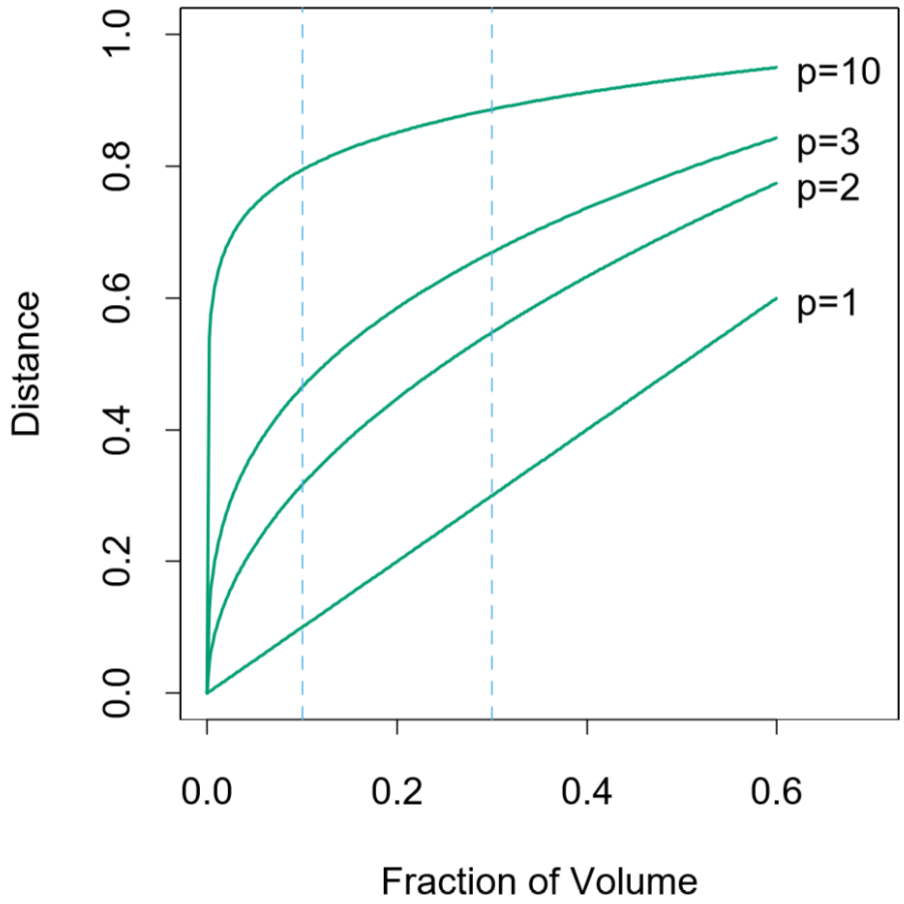
The Curse of Dimensionality

Example: Assume a p dimensional input with uniformly distributed datapoints in the unit cube. For a cubical neighbor around x_i to capture a proportion r of the data points, it will have to cover a proportion r of the volume, so the length of each edge will be $e_r = r^{\frac{1}{p}}$.

In 10 dimensions $e_{.01} = .63$, so capturing 1% of the data requires covering over half the range for each input. It's not clear that this is now a "local" average we're taking.

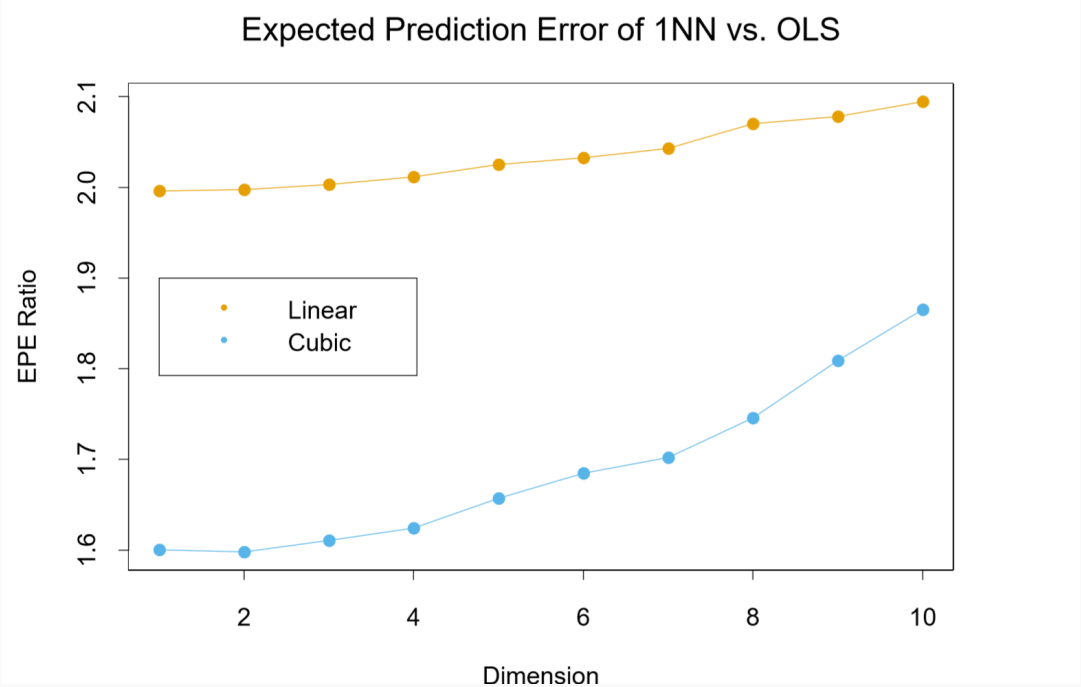
The Curse of Dimensionality





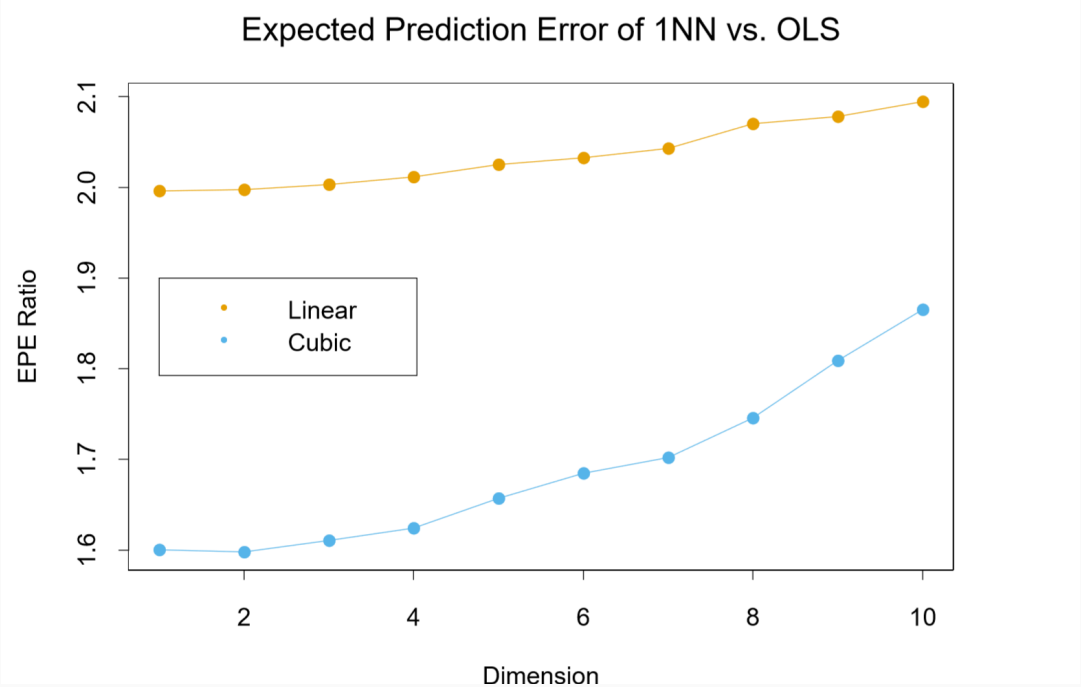
Indeed, we see that as p increases, length of a side required to probe the same fraction of volume increases drastically.

The Curse of Dimensionality



However, the linear classifier can be shown to avoid these problems. The graph above plots the ratio of the expected prediction error for 1 nearest neighbors relative to least squares linear model. In the orange line, the data was generated by $f(x) = x_1$, while for the blue line the data was generated by $f(x) = \frac{1}{2}(x_1 + 1)^3$. $N = 500$ data points were used in training.

The Curse of Dimensionality



By relying on rigid assumptions, the linear model has very little bias and is quite stable even for large dimensional data sets, whereas k-nearest neighbors is much less stable. But if the assumptions are wrong all conclusions may be incorrect.

We will see in this course that there is a whole range of models that live between these two extremes and serve to most efficiently access one kind of data or another.

Course Structure and Policies

Course Texts

This course will follow a pair a text books:

> **High level:** *The Elements of Statstical Learning*. Trevor Hastie Robert Tibshirani, Jerome Friedman.
>
> **Low level:** *Understanding Machine Learning: From Theory to Algorithms*. Shai Shalev-Shwartz, Shai Ben-David.

There is also a practicum book which is worth having if you're interested in machine learning:

> **Implementation:** *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. Aurélien Géron.

Society and impact:

> **Implementation:** *Weapons of math destruction*. Cathy O'Neil.

Course Policies

The course grade will be divided up as follows:

30% Homework

30% Labs

40% Final Project

Homework

Homeworks will be due Tuesday in class in weeks 3, 5 and 7. You are free to discuss homework with me in office hours. You may work in study groups, but all work turned in must be your own. Homework will be partially graded on presentation, and extra points will be awarded for homework completed in latex.

Labs

Labs will use Python and will be distributed as Jupyter Notebooks. They will be turned in through Blackboard. There will be roughly 6 labs beginning in week 4. The labs will be partially in class, and should take no longer than the allotted class time in theory, but extra credit will be given for going above and beyond, both in presentation and in results.

Project

The final project will consist of a project report (roughly 5 pages) and presentation (roughly 10 minutes).

Project groups should contain 3-4 people.

Projects can be of one of three forms:

A computational analysis of a data set using sufficiently complicated or novel techniques from this course.

A theoretical presentation of a topic not covered in this course with a case study.

Industry project working with an external company on real world data.

National laboratory project - Using machine learning to automatically set parameters an algorithms used in national security.

I am more than happy to discuss possible project in any of these categories with you.

Project

Rough timeline:

- Group Selection: January 26
- Project Proposal: February 9
- First Draft: March 15
- Final Draft: April 5
- Presentations: April 9 - 19.

These may change depending on the loss function.