

k -Nearest Neighbors, Curse of Dimensionality, and Naive Bayes

Introduction

In this lecture we'll introduce another classification technique, k -Nearest Neighbors.

First we'll introduce the notion of **parametric** and **nonparametric** models.

Parametric vs. Nonparametric Models

There are many ways to define models (whether supervised or unsupervised).

However a key distinction is this:

- does the model have a *fixed number* of parameters or
- does the number of parameters *grow with the training data*.

Parametric models

- have fixed parameters independent of the training data
- have the advantage of (often) being faster to use
- have the disadvantage of making strong assumptions about the nature of the underlying data distributions.

Nonparametric models

- have parameters that grow with the size of the training data
- are more flexible
- can be computationally intractable for large datasets

Parametric Models

- **Linear Regression:** Assumes a linear relationship between the input variables and the output.
- **Logistic Regression:** Used for binary classification problems.
- **Polynomial Regression:** Extends linear regression by considering polynomial relationships.
- **Support Vector Machines (SVM):** With a linear kernel.
- **Naive Bayes:** Based on Bayes' theorem with strong independence assumptions.
- **Generalized Linear Models (GLM):** Extends linear models to allow for response variables that have error distribution models other than a normal distribution.

Nonparametric Models

- **K-Nearest Neighbors (KNN):** Classifies a data point based on how its neighbors are classified.
- **Decision Trees:** Splits the data into subsets based on the value of input features.
- **Random Forest:** An ensemble of decision trees.
- **Support Vector Machines (SVM):** With non-linear kernels like RBF.

k -Nearest Neighbors

When I see a bird that walks like a duck and swims like a duck and quacks like a duck, I call that bird a duck.

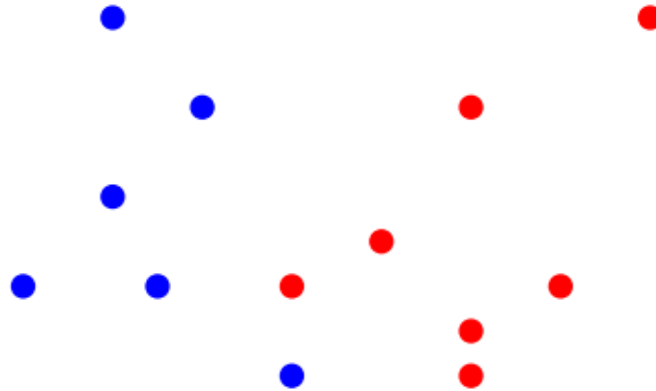
–James Whitcomb Riley (1849 - 1916)



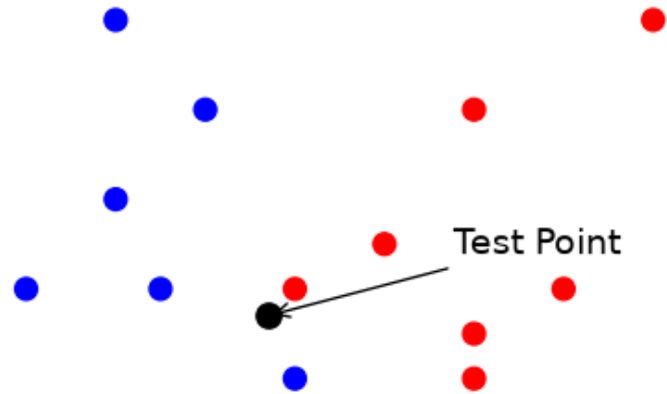
Like any classifier, k -Nearest Neighbors is trained by providing it a set of labeled data. However, at training time, the classifier does very little. It just stores away the training data.

Example: Training Points - 2 Classes

► Code



► Code

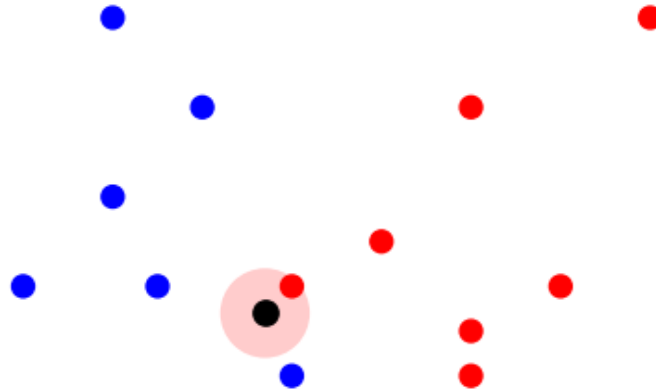


At test time, simply

- “look at” the k points in the training set that are nearest to the test input x , and
- make a decision based on the labels on those points, e.g. majority vote

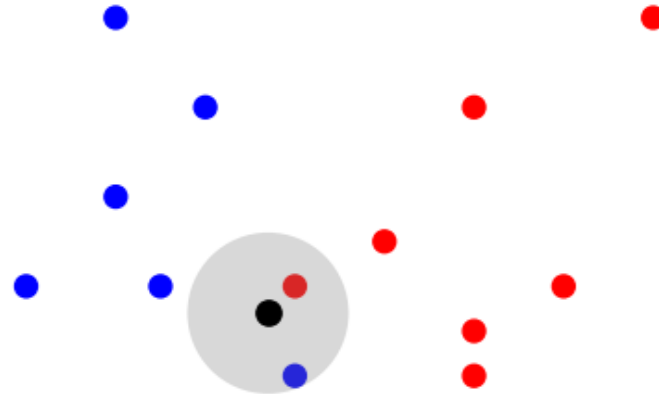
1-Nearest Neighbor

► Code



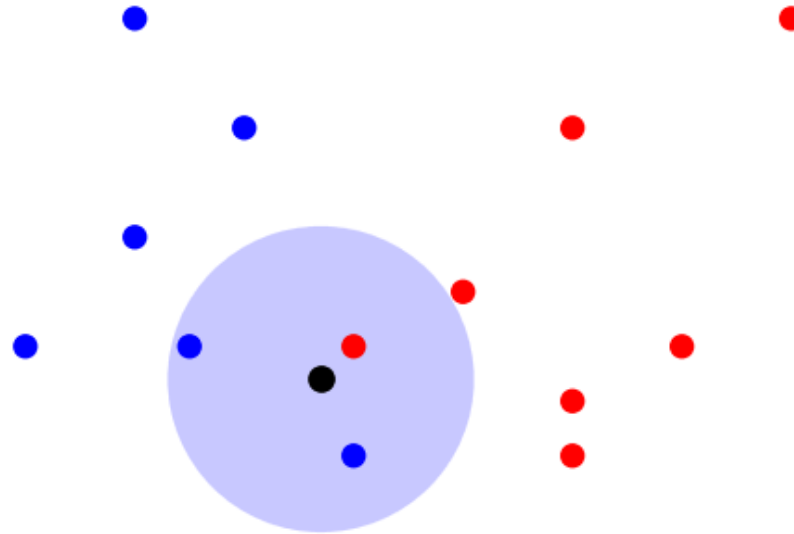
2-Nearest Neighbors

► Code



3-Nearest Neighbors

► Code



Hard vs. Soft Classification

k -Nearest Neighbors can do either **hard** or **soft** classification.

- As a **hard classifier**, it returns the majority vote of the labels on the k Nearest Neighbors.

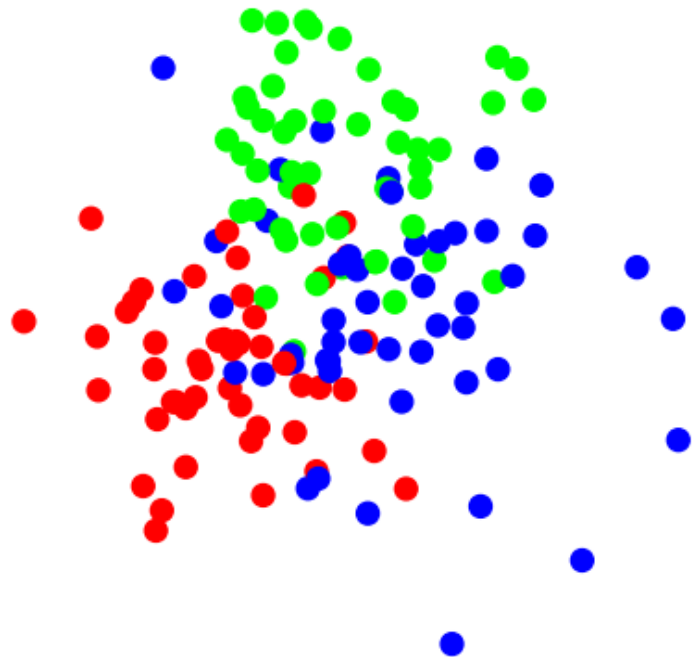
Which may be indeterminate in the case of a tie.

Model Selection for k -NN

Varying k

Consider this dataset where items fall into three classes:

► Code

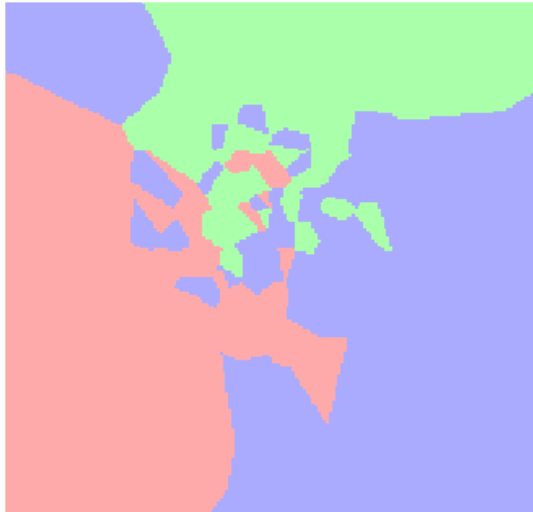


- Let's observe how the complexity of the resulting model changes as we vary k .
- We'll do this by plotting the **decision regions**. These show how the method would classify each potential test point in the space.

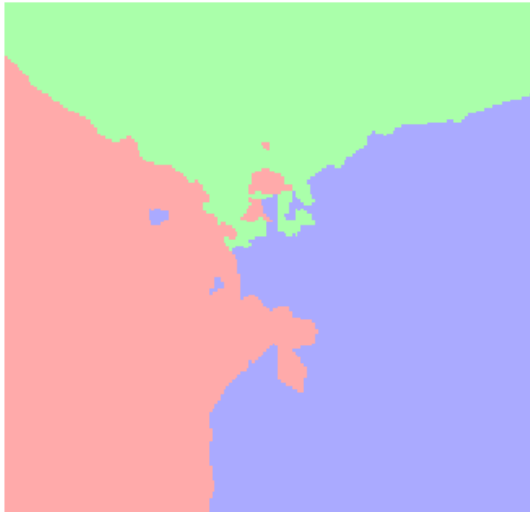
► Code

► Code

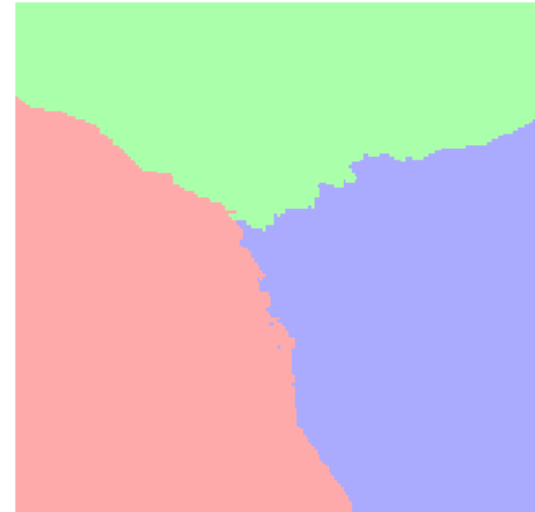
Decision Regions for $k = 1$



Decision Regions for $k = 5$



Decision Regions for $k = 25$



Notice how increasing k results in smoother decision boundaries.

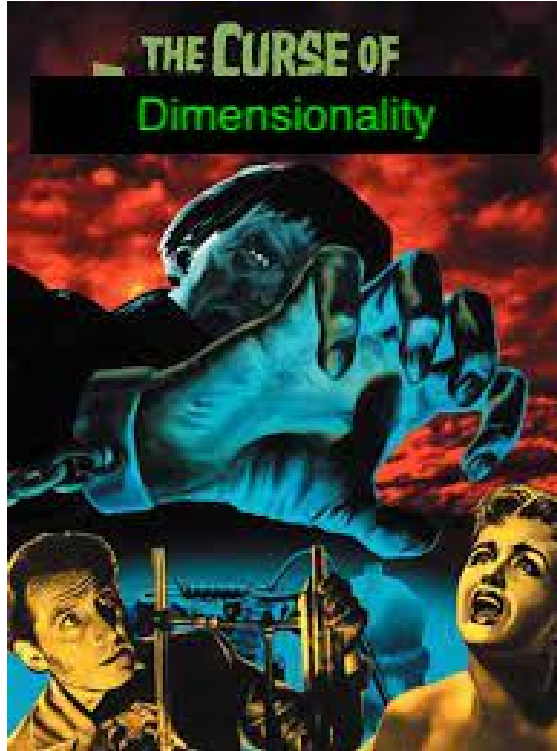
These are more likely to show good generalization ability.

Challenges for k -NN

Working with a k -NN classifier can involve some challenges.

The Curse of Dimensionality

The Curse of Dimensionality



The **curse of dimensionality** is a somewhat tongue-in-cheek term for serious problems that arise when we use geometric algorithms in high dimensions.

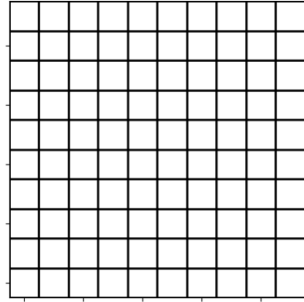
There are various aspects of the curse that affect k -NN.

The phrase ‘curse of dimensionality’ was coined by the mathematician Richard Bellman in 1957 in his book *Dynamic Programming*.

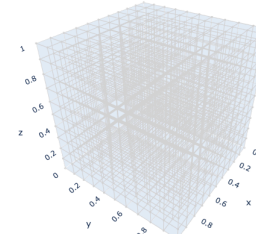
Points are far apart in high D

- k -NN relies on there being one or more “close” points to the test point x .
- In other words, we need the training data to be relatively dense, so there are “close” points everywhere.
- Unfortunately, the amount of space we work in grows exponentially with the dimension d .

Points are far apart in high D



10x10 grid with 10^2 bins



10x10x10 grid with 10^3 bins

Points are far apart in high D

One very intuitive way to think about it is this.

In order for two points to be close in $\mathbb{R}^d$, they must be close in **each** of the d dimensions. As the number of dimensions grows, it becomes harder and harder for a pair of points to be close in **each** dimension.

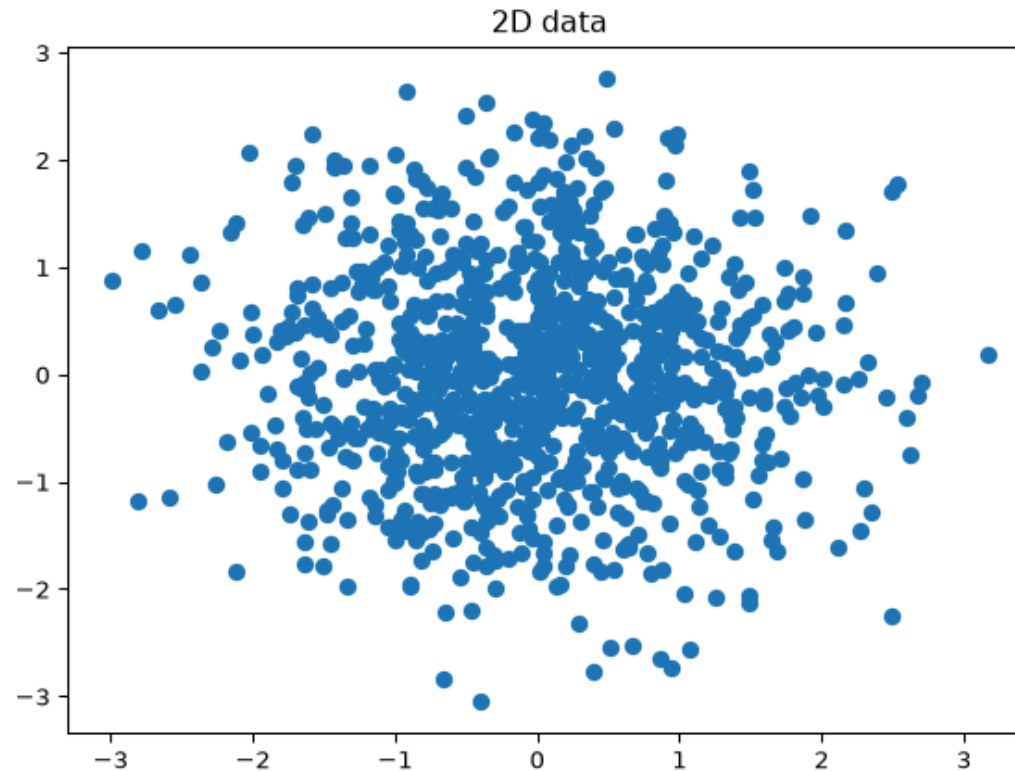
Distances shrink in high dimensions

- Contrarily, distances shrink in high dimensions. An example:
- Let's generate 1000 normally distributed points in 2D, 3D, 100D, and 1000D and look at the ratio of the largest to smallest distance between points.

► Code

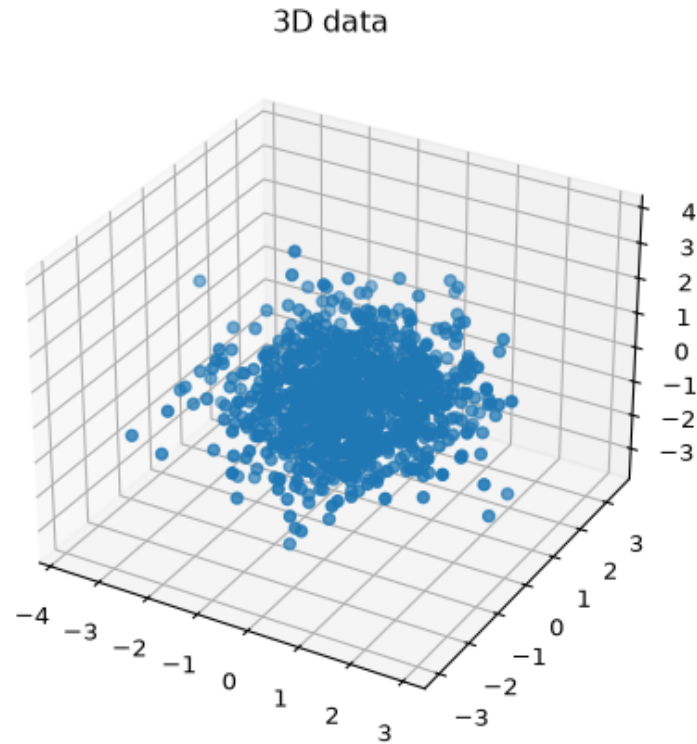
Scatter plot of 2D data

► Code



Scatter plot of 3D data

► Code



Closest/farthest distance ratios

Define a function to calculate the ratio of the largest to smallest distance between points in a dataset.

► [Code](#)

And then calculate the Euclidean distance ratio for each dataset.

► [Code](#)

```
Ratio of largest to smallest distance 2D: 2699.982  
Ratio of largest to smallest distance 3D: 356.654  
Ratio of largest to smallest distance 100D: 1.974  
Ratio of largest to smallest distance 1000D: 1.228
```

Manhattan distance ratio

► Code

```
Ratio of largest to smallest distance 2D (Manhattan): 3694.569  
Ratio of largest to smallest distance 3D (Manhattan): 315.760  
Ratio of largest to smallest distance 100D (Manhattan): 2.032  
Ratio of largest to smallest distance 1000D (Manhattan): 1.255
```

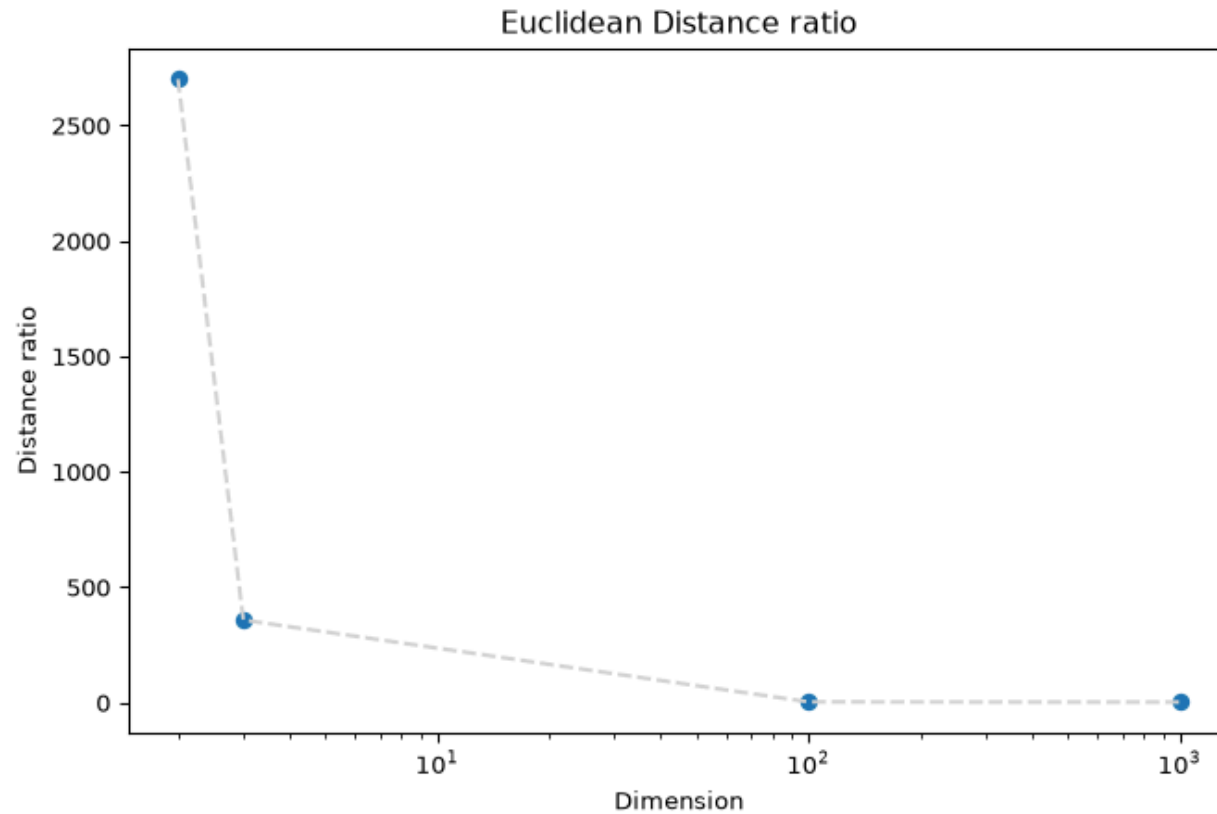
Cosine distance ratio

► Code

```
Ratio of largest to smallest distance 2D (Cosine): 76853891711.879  
Ratio of largest to smallest distance 3D (Cosine): 251995.519  
Ratio of largest to smallest distance 100D (Cosine): 2.606  
Ratio of largest to smallest distance 1000D (Cosine): 1.351
```


Euclidean distance ratios

► Code

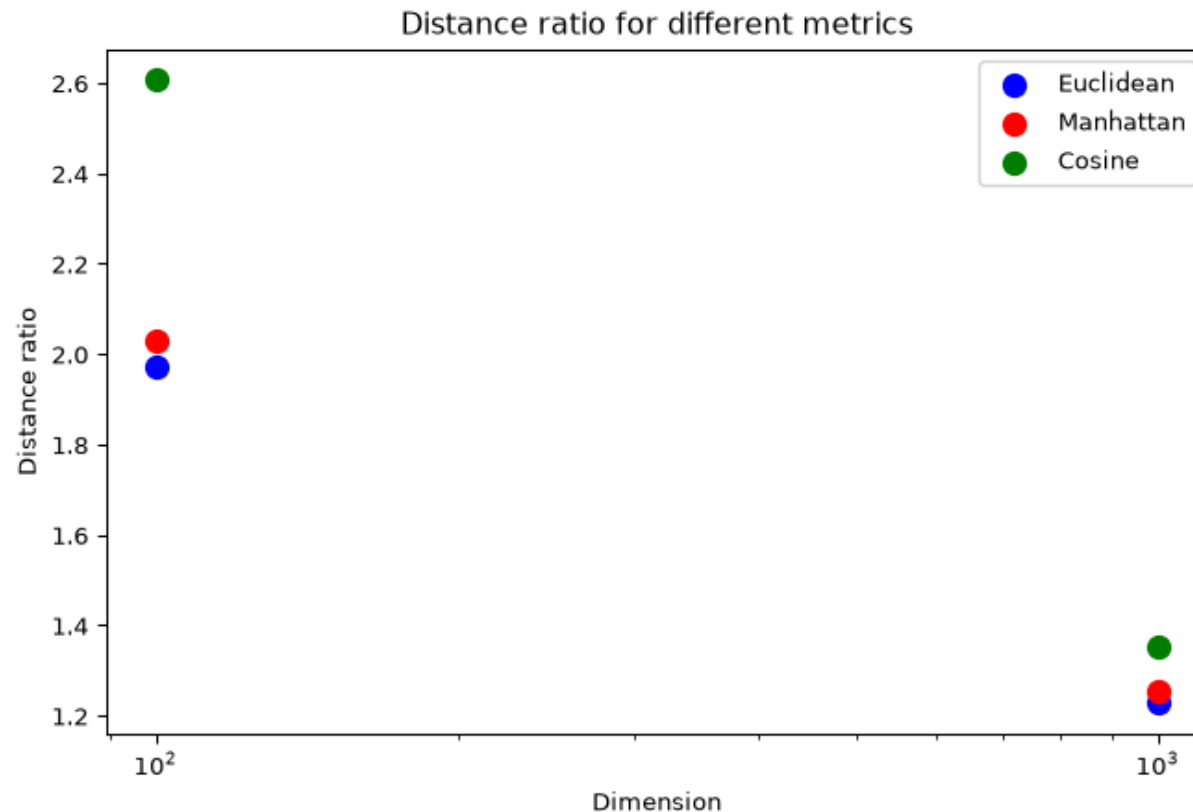


You see that the ratio of the largest to smallest distance drops dramatically as the

Compare distance ratios

Let's compare the distance ratios for the Euclidean, Manhattan, and Cosine metrics in 100d and 1000d all in the same plot.

► Code



Volume of a hypersphere

The above phenomenon is related to the strange phenomenon of the volume of a hypersphere (or n-ball) in increasing dimensions.

Now, the volume of a *hypersphere* in 2D and 3D is:

- For $d = 2$, πr^2 , which is π for a unit radius.
- For $d = 3$, $\frac{4\pi}{3} r^3$, which is $\frac{4}{3}\pi$ for a unit radius.

The [general equation](#) for a hypersphere of radius R in d dimensions is:

$$V_d(R) = \frac{\pi^{d/2}}{\Gamma\left(\frac{d}{2} + 1\right)} R^d,$$

where Γ is Euler's gamma function and simplifies to $\Gamma(d) = (d - 1)!$ for all positive integers d and has a compact representation for postive half integers.

For a fascinating exploration of the properties of unit spheres in high dimension, see [An Adventure in the Nth Dimension](#) by Brian Hayes in *American Scientist*.

Hypersphere volume in increasing dimensions

Here's a function to calculate the volume of a hypersphere of a given radius and dimension.

► [Code](#)

And now let's calculate the volume of a hypersphere of unit radius for dimensions 1 to 20 and store it in a list.

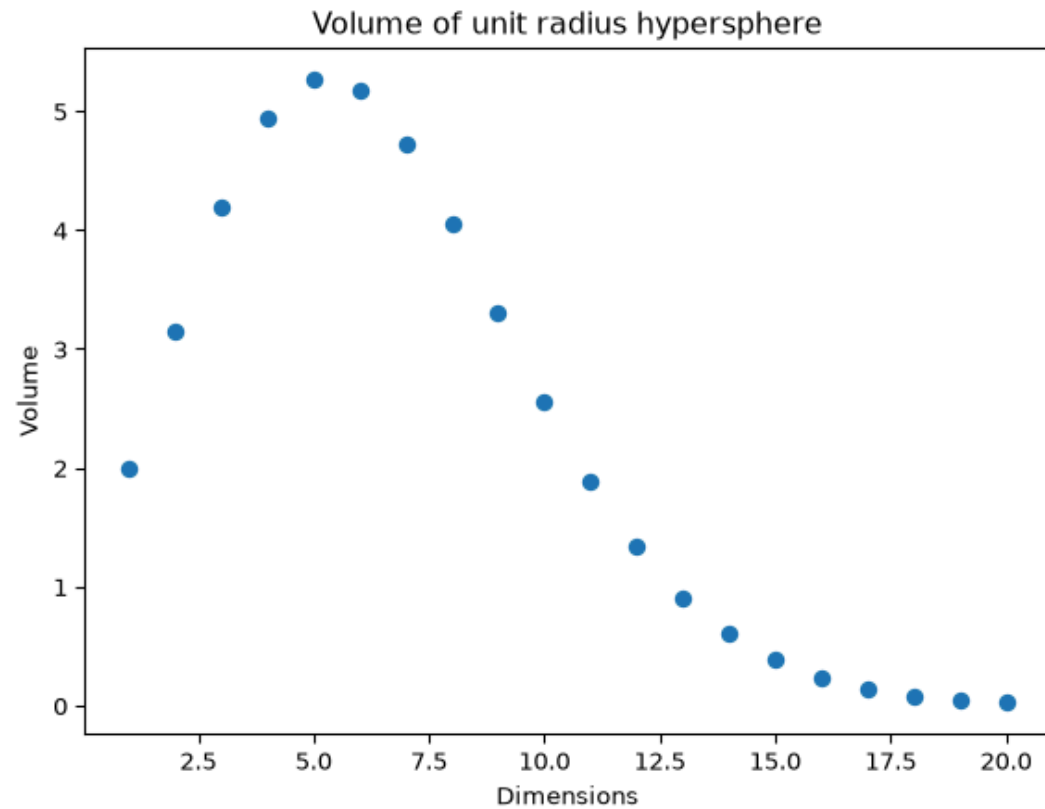
► [Code](#)

```
1: 2.0, 2: 3.1, 3: 4.2, 4: 4.9, 5: 5.3, 6: 5.2, 7: 4.7, 8: 4.1, 9: 3.3, 10: 2.6, 11: 1.9, 12: 1.3,  
13: 0.9, 14: 0.6, 15: 0.4, 16: 0.2, 17: 0.1, 18: 0.1, 19: 0.0, 20: 0.0,
```

Plot volumes of hyperspheres

And plot the results.

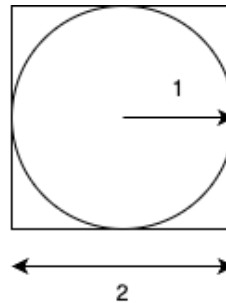
► Code



Ratio of unit hypersphere volume and enclosing hypercube volume

Another strange phenomenon to think about is via another example.

Assume you have data *uniformly* distributed in a hypercube with side length 2 which will exactly fit a unit hypersphere.

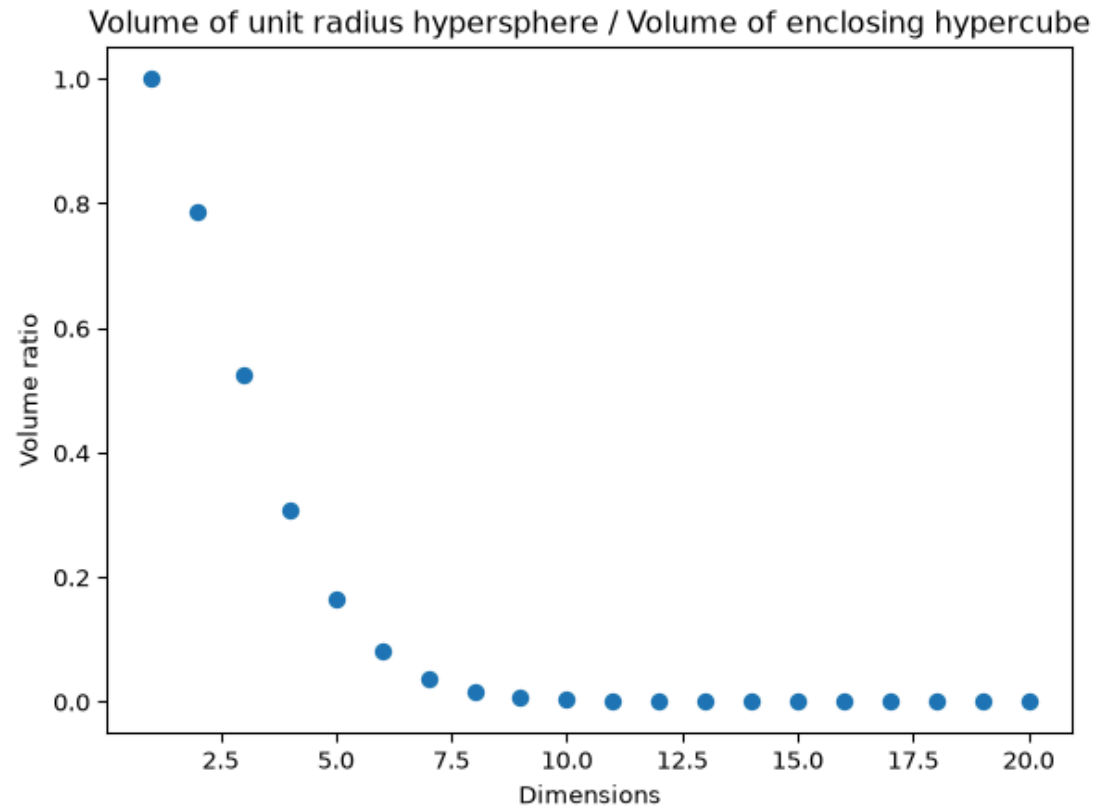


As the dimension grows, we want to know what fraction of the data is within unit distance of the center.

► [Code](#)

Plot volume ratios

► Code



The fraction quickly goes to zero as the dimension increases.

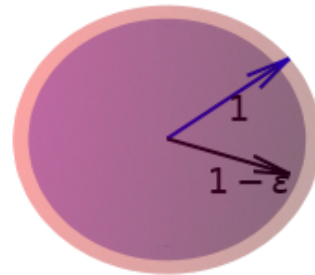
Proportion of hypersphere in outer shell

Another strange thing about high dimensions is that most of the volume of a hypersphere is in the outer shell.

We can compare the volumes of hyperspheres of radius 1 and $1 - \epsilon$ for a small ϵ .

► Code

Inner and Outer Hyperspheres



Proportion of volume in outer shell

Let's create a function to calculate the proportion of the volume of a hypersphere that is in the outer ϵ of the radius.

► Code

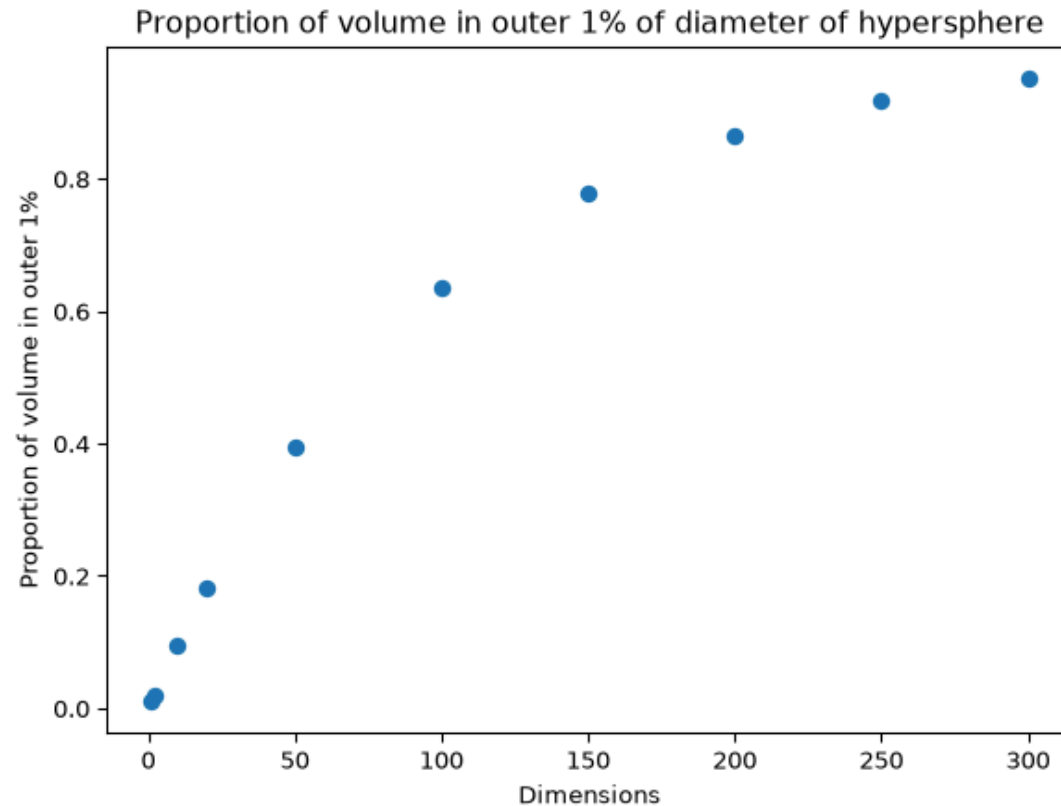
Let's calculate the proportion of the volume in the outer 1% of the radius for dimensions 1 to 300.

► Code

```
Proportion of volume in outer 1 percent of radius in 1 dimensions =0.010
Proportion of volume in outer 1 percent of radius in 2 dimensions =0.020
Proportion of volume in outer 1 percent of radius in 10 dimensions =0.096
Proportion of volume in outer 1 percent of radius in 20 dimensions =0.182
Proportion of volume in outer 1 percent of radius in 50 dimensions =0.395
Proportion of volume in outer 1 percent of radius in 100 dimensions =0.634
Proportion of volume in outer 1 percent of radius in 150 dimensions =0.779
Proportion of volume in outer 1 percent of radius in 200 dimensions =0.866
Proportion of volume in outer 1 percent of radius in 250 dimensions =0.919
Proportion of volume in outer 1 percent of radius in 300 dimensions =0.951
```

Plot outer shell proportion

► Code



By the time we get to 300 dimensions most of the volume is in the outer 1 percent.

Fraction of points in outer shell

What is the fraction f_d of all the points that are within a unit distance, but **not** within a distance of $1 - \epsilon$?

Let

$$K_d = \frac{\pi^{d/2}}{\Gamma\left(\frac{d}{2} + 1\right)}$$

so that the volume of a hypersphere of radius R is $K_d \times R^d$.

Then for $R = 1$ and $R = 1 - \epsilon$ we have:

$$\begin{aligned} f_d &= \frac{\text{Volume of Shell}}{\text{Volume of unit hypersphere}} \\ &= \frac{K_d \times (1)^d - K_d \times (1 - \epsilon)^d}{K_d \times (1)^d} \\ &= 1 - (1 - \epsilon)^d \end{aligned}$$

Curse of Dimensionality Example

Here's another example.

We create 100 points, scattered at random within a d -dimensional space.

We will look at two quantities as we vary d .

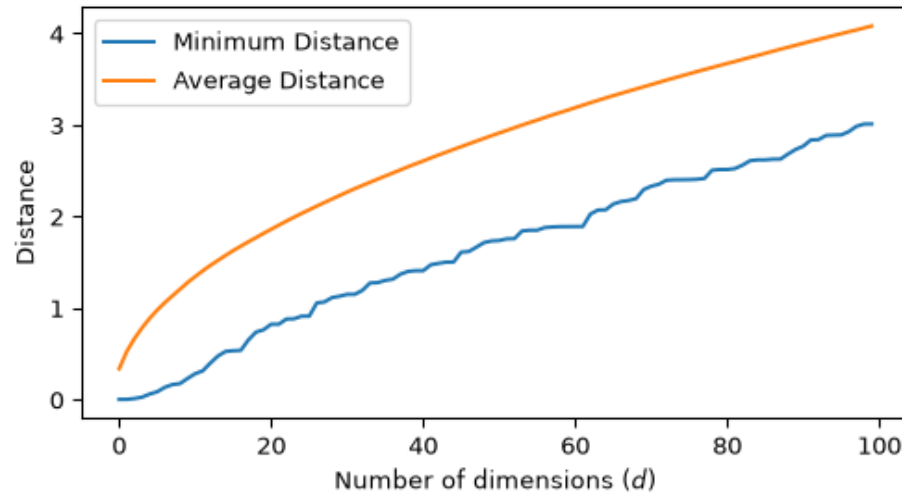
- The **minimum** distance between any two points.
- The **average** distance between any two points.

The following example is based on *Data Science from Scratch*, Joel Grus, Second Edition, Chapter 12.

► Code

► Code

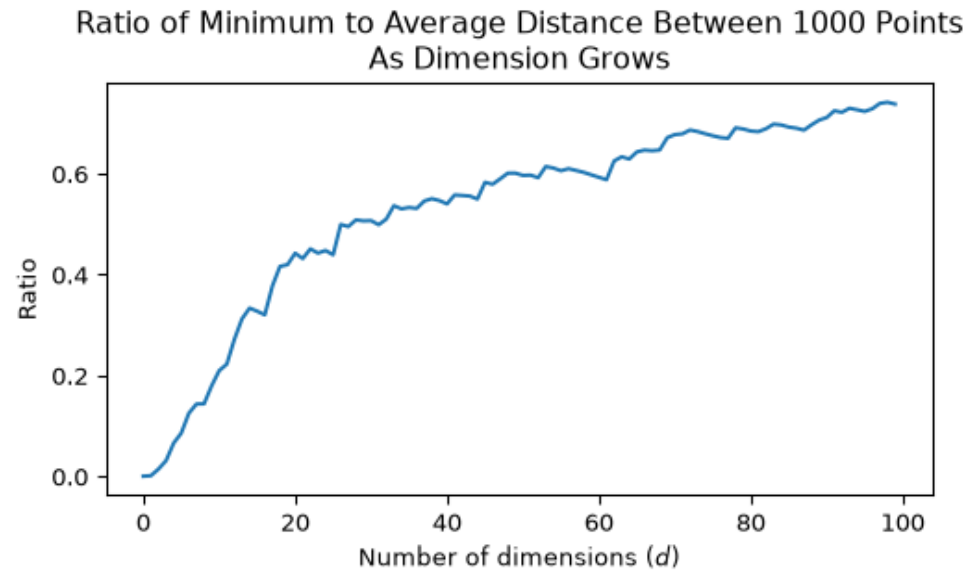
Comparison of Minimum Versus Average Distance Between 1000 Points
As Dimension Grows



The average distance between points grows. However we also observe that the minimum distance between points grows at a similar rate.

Let's look at the ratio between the average distance between points and the minimum distance between points.

► Code



This shows that, for any test point x , the distance to the **closest** point to x , gets closer and closer to the **average** distance between points.

If we used a point at the average distance for classifying x we'd get a very poor classifier.

Implications of the Curse

For k -NN, the Curse of Dimensionality means that in high dimensions most points are nearly the same distance from the test point.

This makes k -NN ineffective. It cannot reliably tell which are the k nearest neighbors and its performance degrades as d increases.

More on Model Selection

Recall that each value of k provides a different model.

Binary classification

In the simple case of a binary classifier, we can call

- one class the 'Positive' class or index 1
- the other class the 'Negative' class or index 0.

The most basic measure of success for a classifier is **accuracy**:

- **Accuracy** is the fraction of test points that are correctly classified.

Precision and Recall

		Classifier Prediction	
		Positive	Negative
Actual Value	Positive	True Positive	False Negative
	Negative	False Positive	True Negative

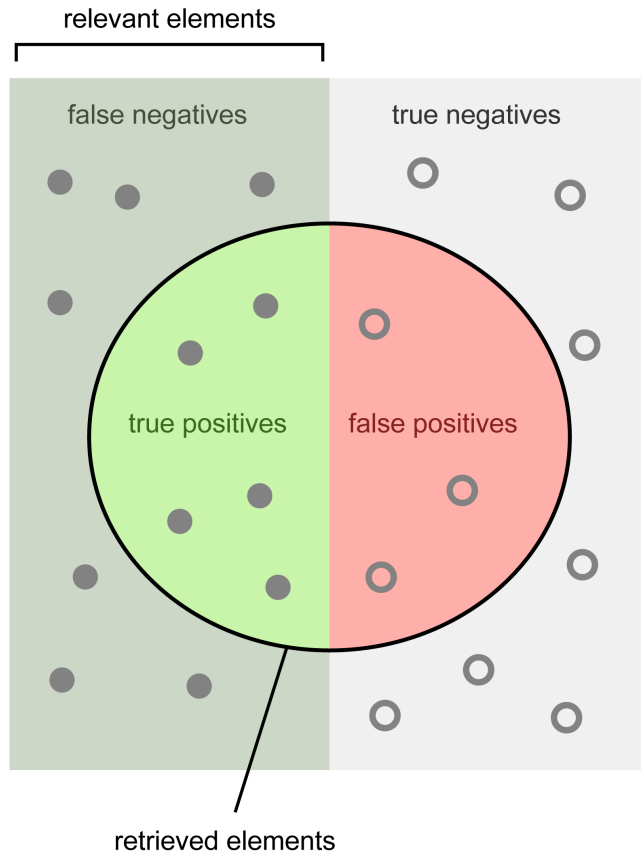
A better way to measure the classifier's performance is using a Confusion Matrix.

Diagonal elements represent successes and off diagonals represent errors.

Using the confusion matrix we can define some more useful measures:

- **Recall** - defined as the fraction of actual positives correctly classified
 - $TP / (TP + FN)$
- **Precision** - defined as the fraction of classified positives correctly classified
 - $TP / (TP + FP)$

Precision and Recall Illustration



How many retrieved items are relevant?

$$\text{Precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

How many relevant items are retrieved?

$$\text{Recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$

Evaluating k - Nearest Neighbors

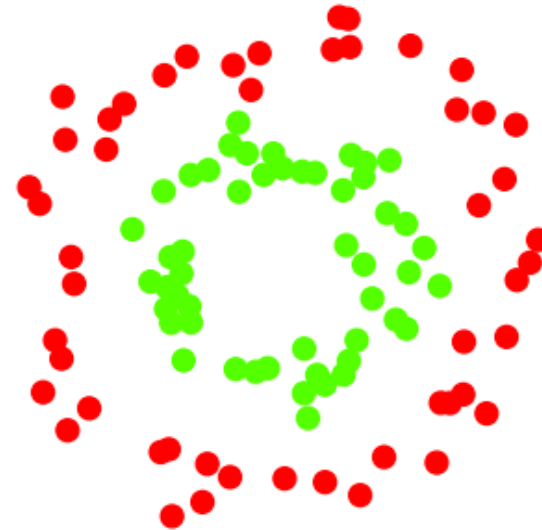
First we'll generate some synthetic data to work with.

► Code

```
Shape of data: (100, 2)  
Unique labels: [0 1]
```

Here is what the data looks like.

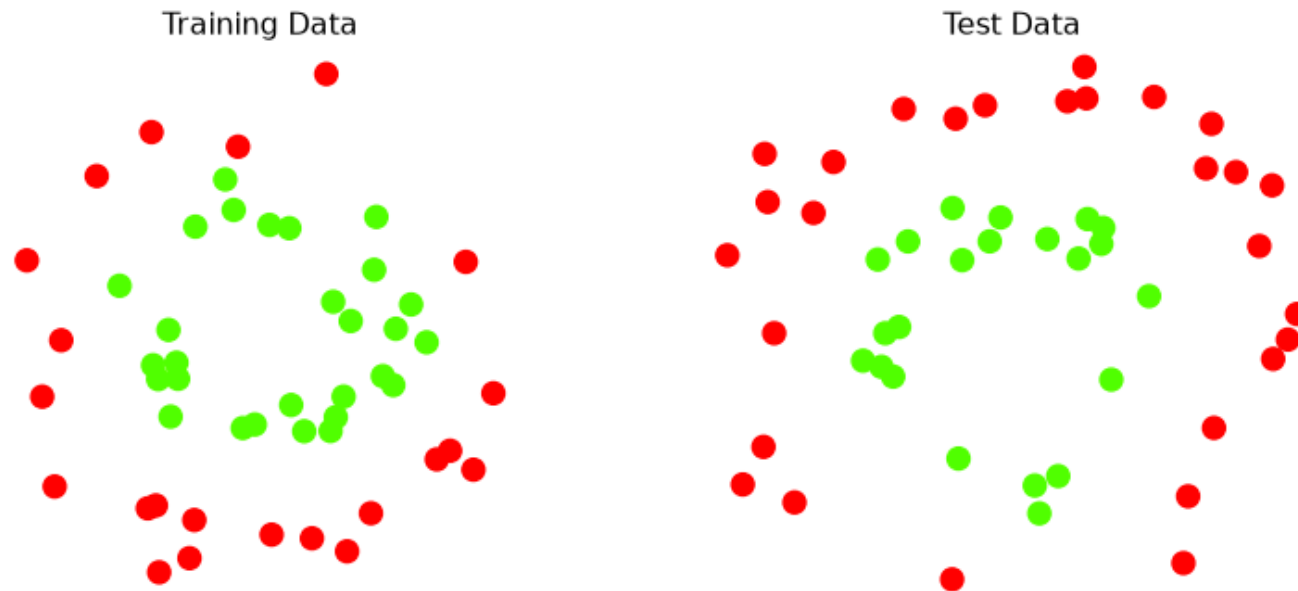
► Code



Recall that we always want to test on data separate from our training data.

For now, we will do something very simple: take the first 50 examples for training and the rest for testing.

► Code



For our first example, we will classify the points (in the two classes) using a k -NN classifier. We will specify that $k = 5$, i.e., we will classify based on the majority vote of the 5 nearest neighbors.

► Code

As we have seen previously, the `scikit-learn fit()` function corresponds to **training** and the `predict()` function corresponds to **testing**.

► Code

```
Accuracy on test data: 0.72
```

Accuracy of 72% sounds good – but let's dig deeper.

Confusion Matrix

We'll call the red points the Positive class and the green points the Negative class.

Here is the confusion matrix:

► [Code](#)

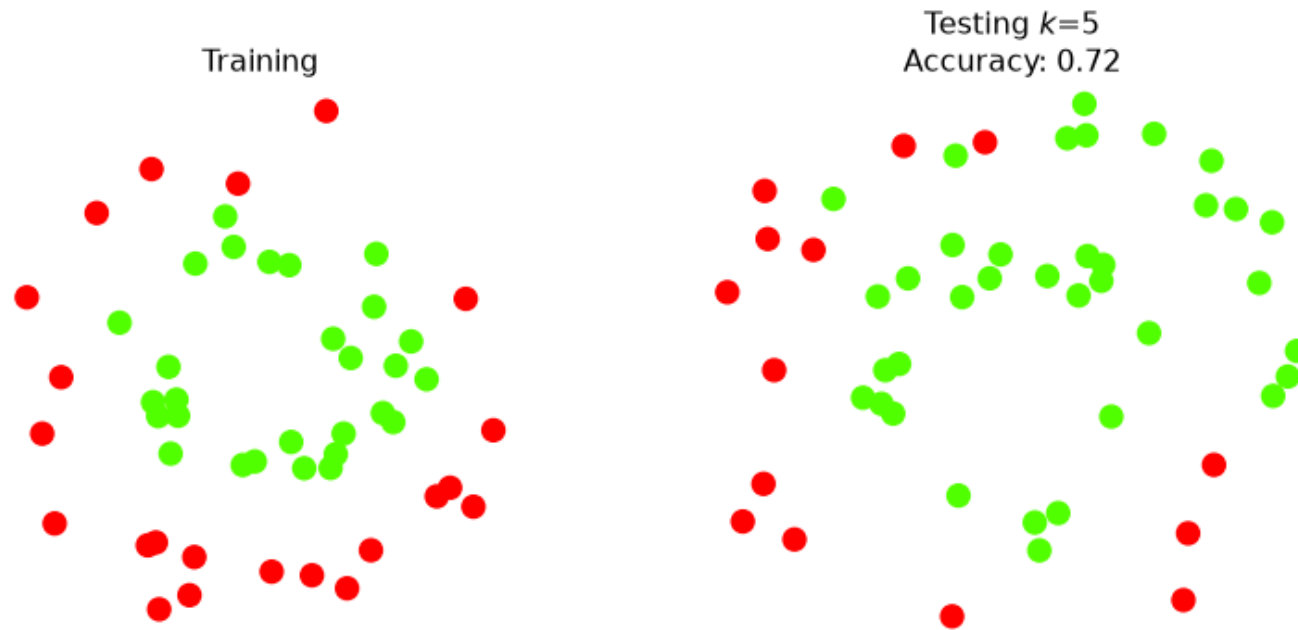
	Predicted 1	Predicted 0
Actual 1	14	14
Actual 0	0	22

Looks like the classifier is getting all of the Negative class correct, but only achieving accuracy of 50% on the Positive class.

That is, its **precision** is 100%, but its **recall** is only 50%.

Let's visualize the results.

► Code



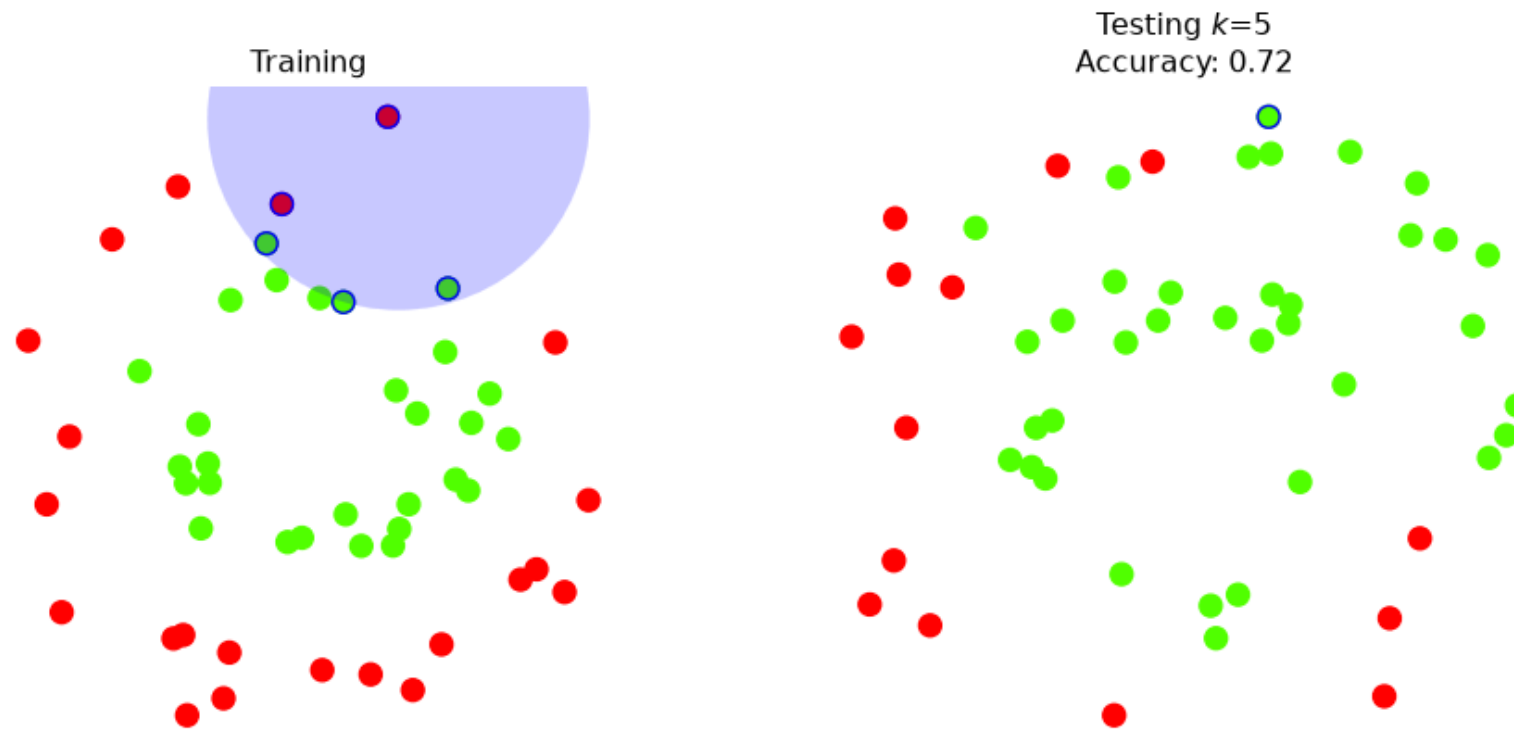
Indeed, the Positive (red) points in the upper half of the test data are all classified incorrectly.

Let's look at one of the points that the classifier got wrong.

► Code

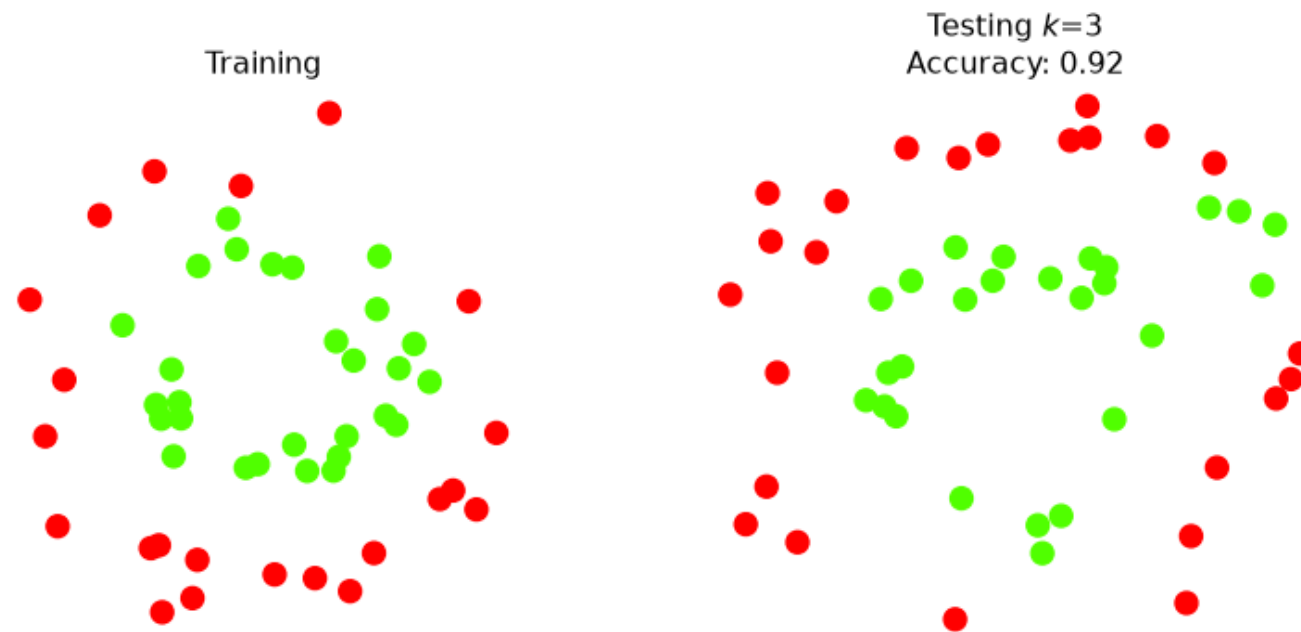
```
/var/folders/j_/hxxgy5dd7655k_416s6t9kgc0000gq/T/ipykernel_9543/3451066026.py:8: UserWarning: You passed both c  
and facecolor/facecolors for the markers. c has precedence over facecolor/facecolors. This behavior may change  
in the future.
```

```
plt.scatter(X_train[neighbors,0], X_train[neighbors,1],
```



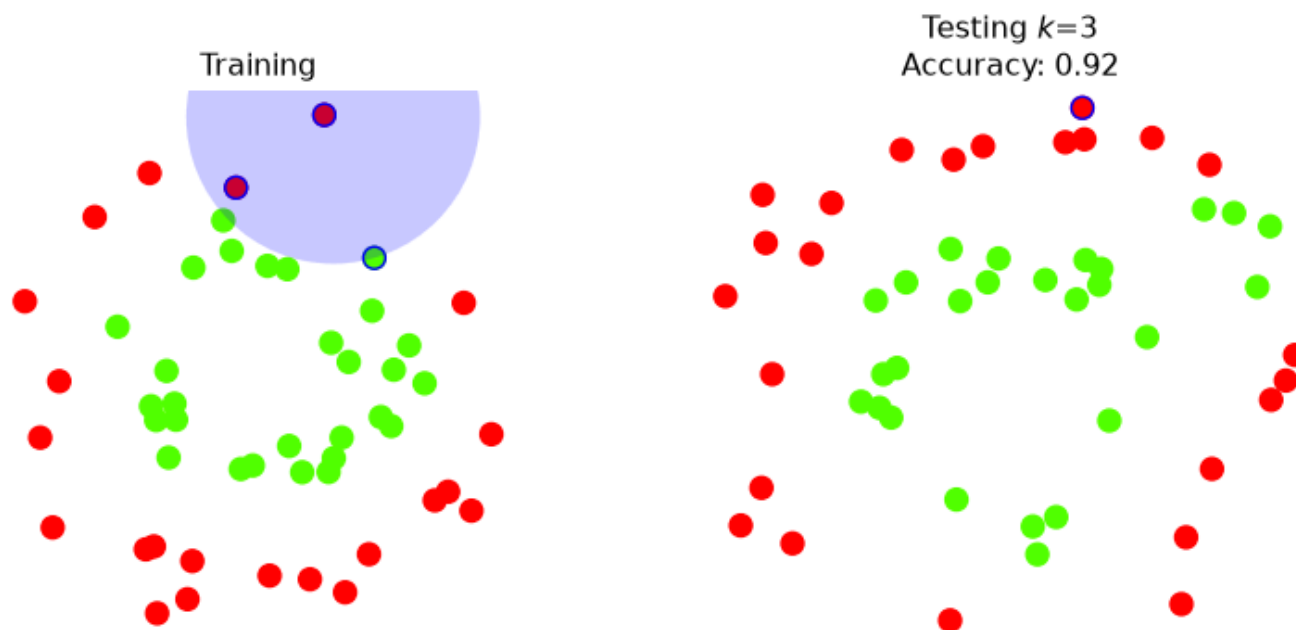
For comparison purposes, let's try $k = 3$.

► Code



And let's look at the same individual point as before.

► Code



Train-Test Splitting

So how confident can we be that the test accuracy is 92% in general?

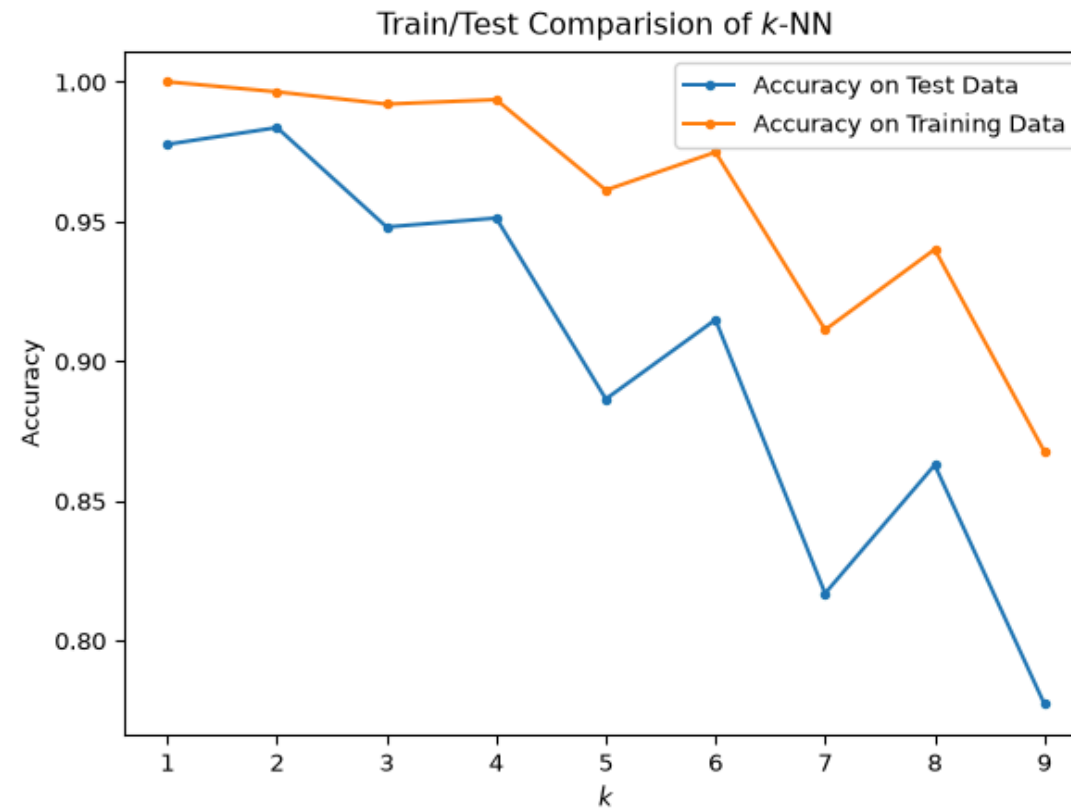
What we really need to do is consider **many** different train/test splits.

Thus, the proper way to evaluate generalization ability (accuracy on the test data) is:

1. Form a random train/test split
2. Train the classifier on the training split
3. Test the classifier on the testing split
4. Accumulate statistics
5. Repeat from step 1 until enough statistics have been collected.

► Code

► Code



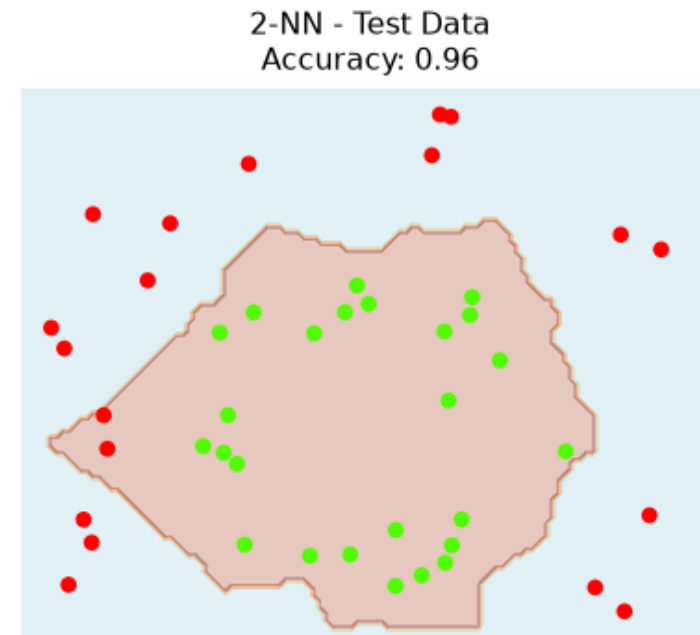
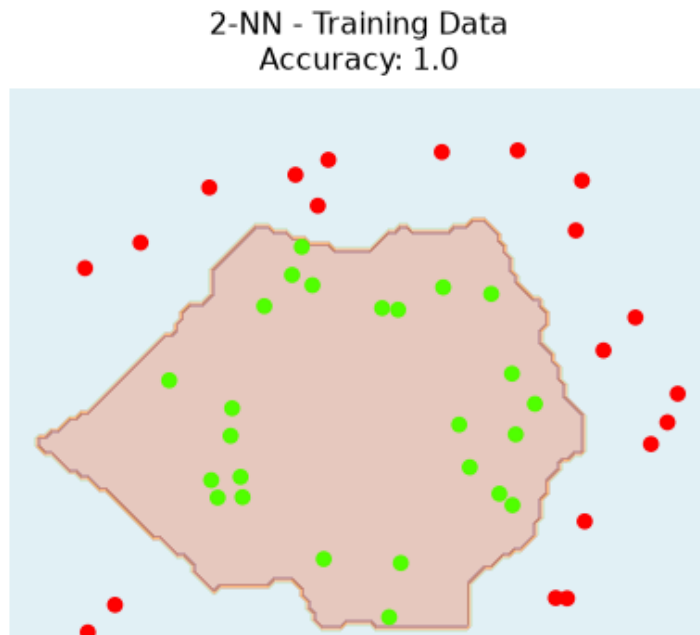
Based on the generalization error, i.e., accuracy on the held-out test data, it looks like $k = 2$ is the best choice.

Here is the **decision boundary** for k -NN with $k = 2$.

► Code

► Code

Ignoring fixed x limits to fulfill fixed data aspect with adjustable data limits.
Ignoring fixed x limits to fulfill fixed data aspect with adjustable data limits.



Real Data

To explore a few more issues, we'll now turn to some famous datasets that have been extensively studied in the past.

The Iris Dataset

The Iris dataset is a famous dataset used by Ronald Fisher in a classic 1936 paper on classification.



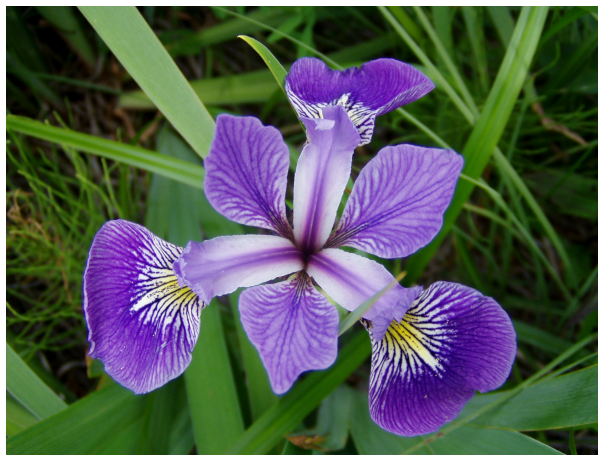
R. A. Fisher

Quoting from Wikipedia:

The data set consists of 50 samples from each of three species of Iris (Iris setosa, Iris virginica and Iris versicolor). Four features were measured from each sample: the length and the width of the sepals and petals, in centimetres. Based on the combination of these four features, Fisher developed a linear discriminant model to distinguish the species from each other.



I. setosa



I. versicolor



I. virginica

Load Iris Dataset

```
1 iris = datasets.load_iris()
2 X = iris.data
3 y = iris.target
4 ynames = iris.target_names
5 print(X.shape, y.shape)
6 print(X[1, :])
7 print(iris.target_names)
8 print(y)
```

[illegible]

Hyperparameters: k-NN

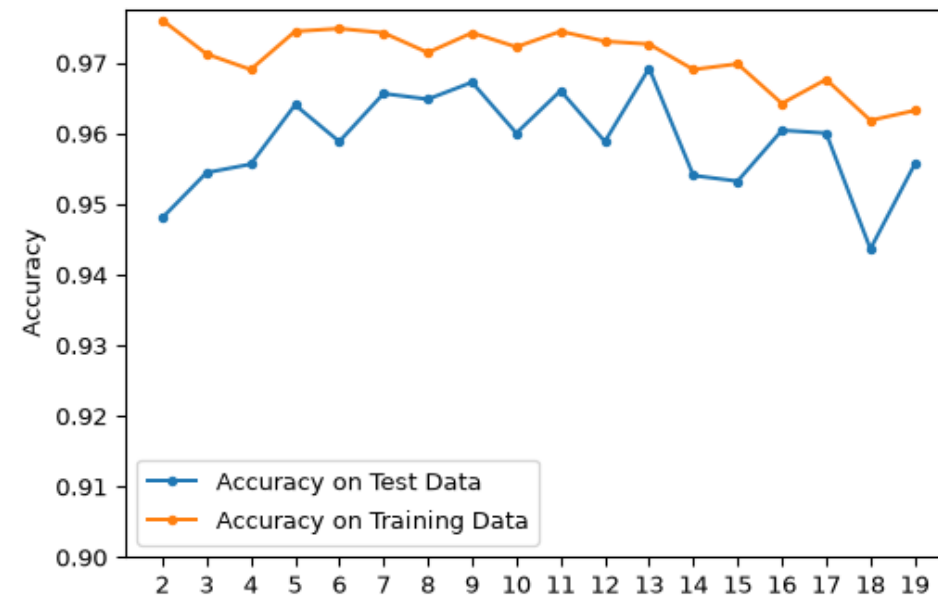
First, we'll explore setting the hyperparameters. We start with k -NN.

To set the hyperparameter k , we evaluate error on the test set for many train/test splits:

► Code

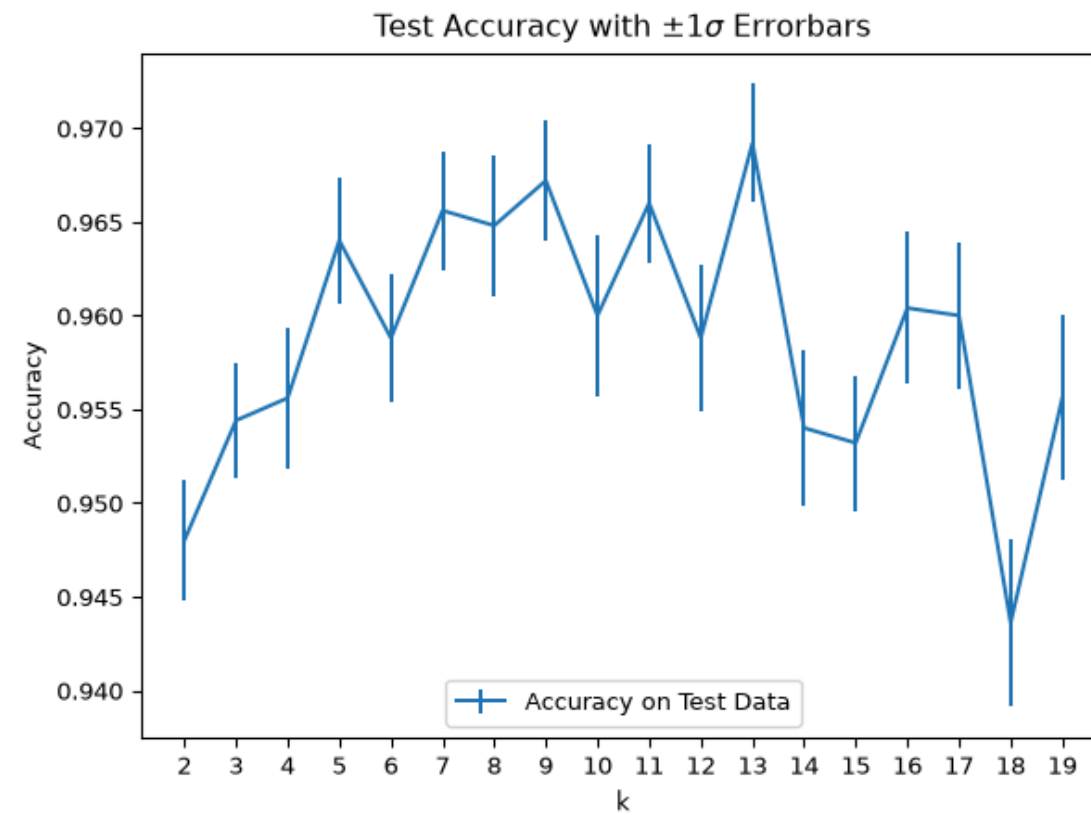
► Code

Max Test Accuracy at $k = 13$ with accuracy 0.969



It looks like $k = 13$ is the best-performing value of the hyperparameter.

► Code

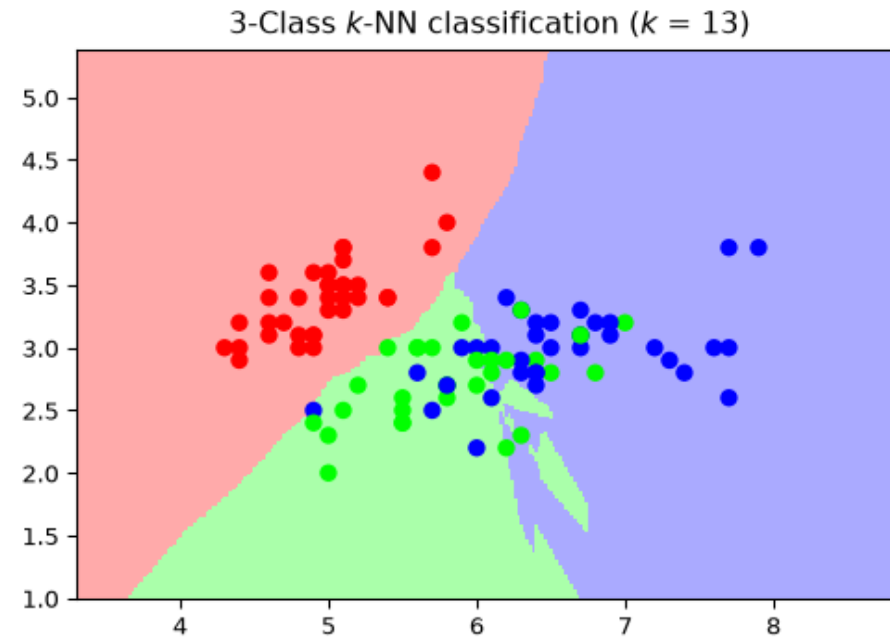


It looks like $k = 13$ is a reasonable value, although a case can be made that 9 and 11 are not statistically distinguishable from 13.

To gain insight into the complexity of the model for $k = 13$, let's look at the decision boundary.

We will re-run the classifier using only two (of four) features for visualization purposes.

► Code



There are a few artifacts, though overall this looks like a reasonably smooth set of decision boundaries.

Hyperparameters: Decision Tree

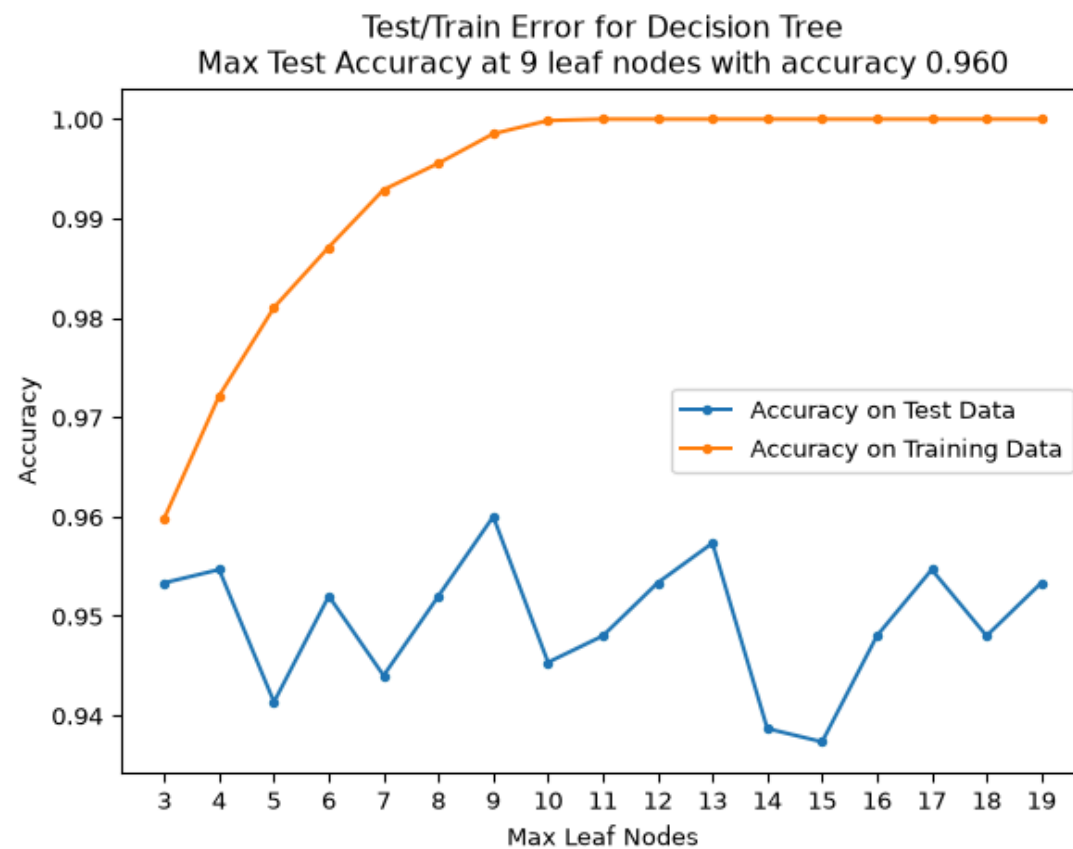
How do we control the complexity of a Decision Tree?

There are a variety of ways (see the [sklearn](#) documentation) but the simplest one is to control the number of leaf nodes in the tree.

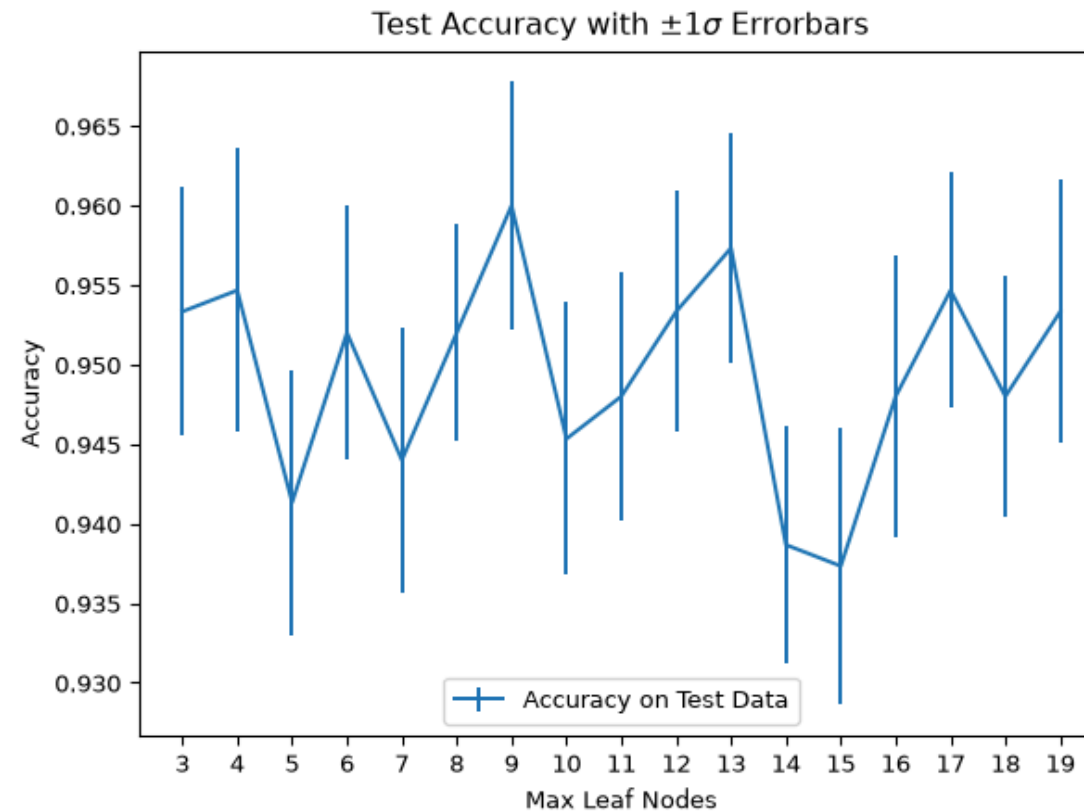
A small number of leaf nodes is a low-complexity model, and a large number of nodes is a high-complexity model.

► Code

► Code



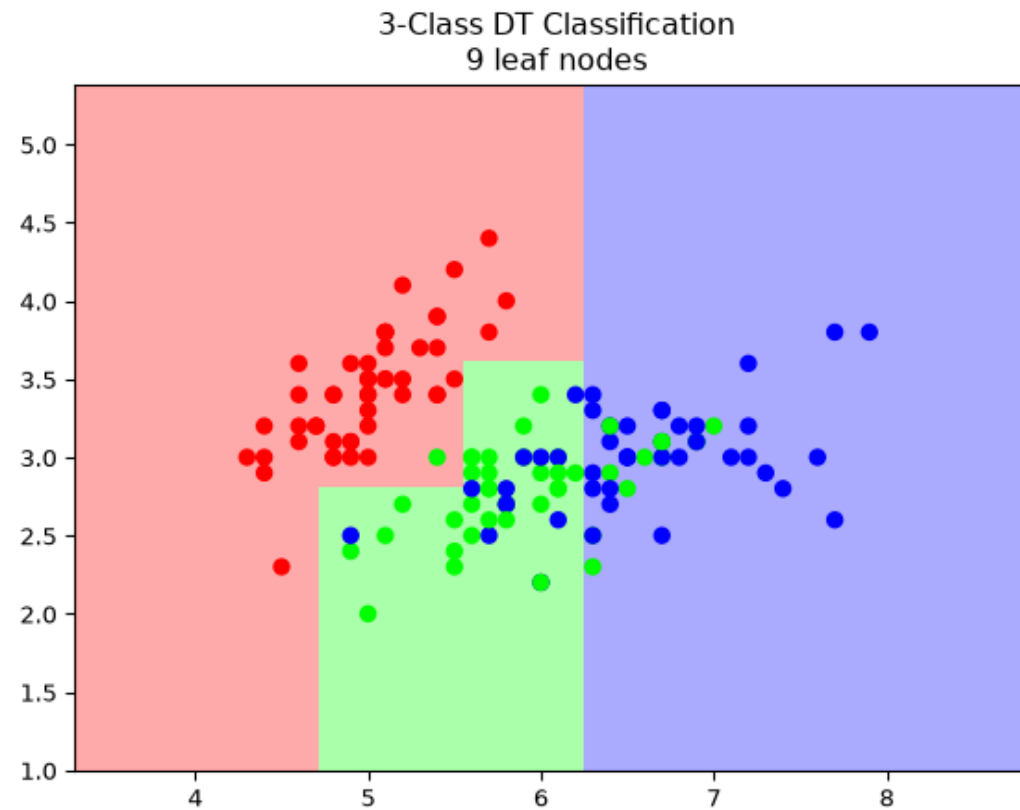
► Code



It looks like 9 leaf nodes is appropriate, but we would be justified to choose 4 or 13 as well.

Here we visualize the decision boundary for the DT:

► Code



MNIST dataset

NIST used to be called the “National Bureau of Standards.” These are the folks who bring you the reference meter, reference kilogram, etc.

NIST constructed datasets for machine learning of handwritten digits. These were collected from Census Bureau employees and also from high-school students.

This dataset has been used repeatedly for many years to evaluate classifiers. For a peek at some of the work done with this dataset you can visit this [link](#).

MNIST Digits

► Code

```
Data shape: (1797, 64)
Data labels: [1 5 0 ... 9 1 5]
Unique labels: [0 1 2 3 4 5 6 7 8 9]
```

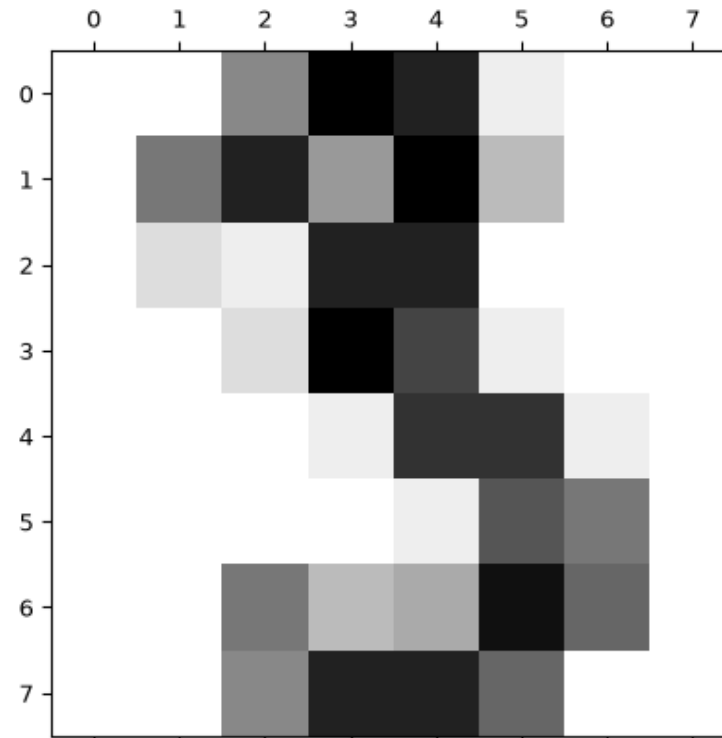
An individual item is an 8×8 image, encoded as a matrix:

► Code

```
array([[ 0.,  0.,  7., 15., 13.,  1.,  0.,  0.],
       [ 0.,  8., 13.,  6., 15.,  4.,  0.,  0.],
       [ 0.,  2.,  1., 13., 13.,  0.,  0.,  0.],
       [ 0.,  0.,  2., 15., 11.,  1.,  0.,  0.],
       [ 0.,  0.,  0.,  1., 12., 12.,  1.,  0.],
       [ 0.,  0.,  0.,  0.,  1., 10.,  8.,  0.],
       [ 0.,  0.,  8.,  4.,  5., 14.,  9.,  0.],
       [ 0.,  0.,  7., 13., 13.,  9.,  0.,  0.]])
```


Here we show the matrix as an image.

► Code



It is easier to visualize if we blur the pixels a little bit.

► Code



Here are some more samples from the dataset.

► [Code](#)



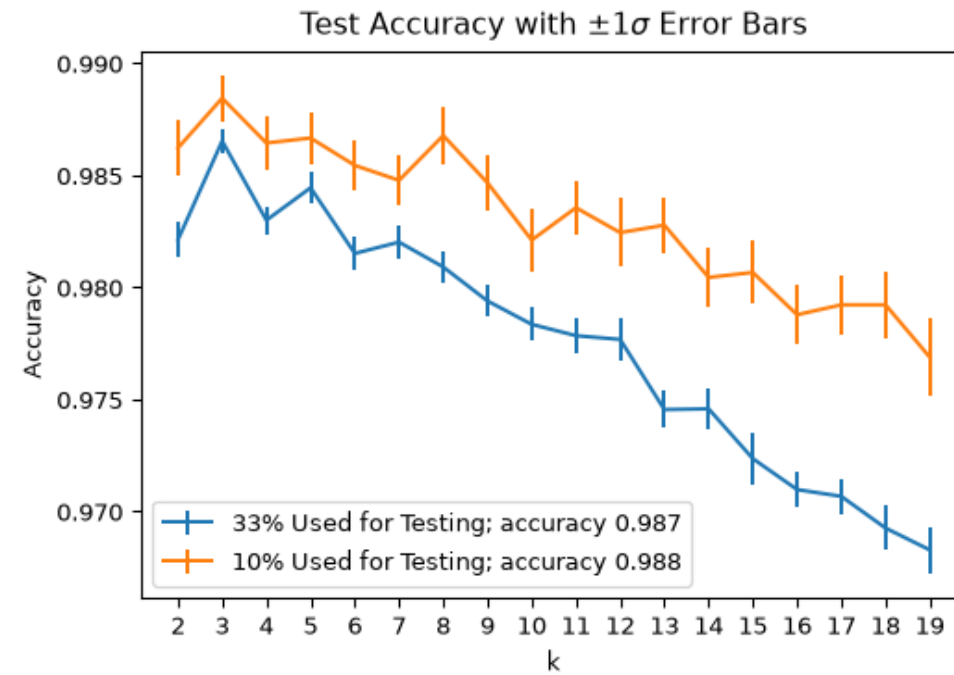
Although this is an 8×8 image, we can reshape it to a vector of length 64.

To do model selection, we will again average over many train/test splits.

However, since the performance of k -NN degrades on large sets of data we may decide to limit the size of our testing set to speed up testing.

► Code

► Code



These plots illustrate show two important principles.

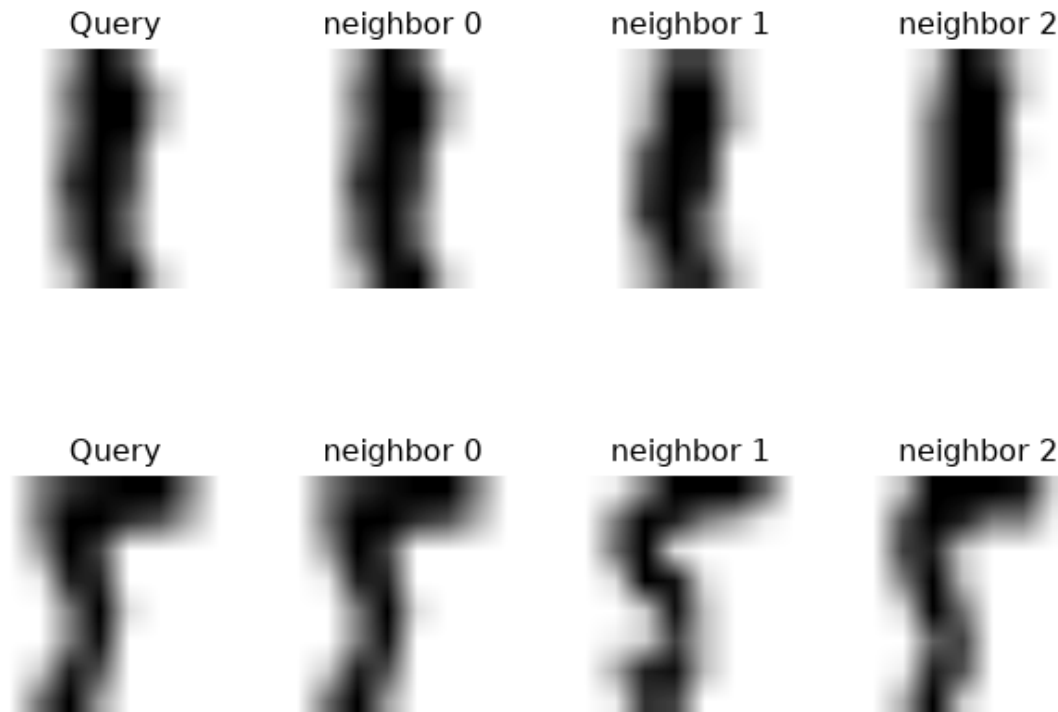
- With more training data the classifier performs better.
- With less testing data, the testing results are more variable. Though testing is faster.

The key decision here is what value to choose for k . So it makes sense to use the 33% test split, because the smaller error bars give us better confidence in our decision.

We can get a sense of why k -NN can succeed at this task by looking at the nearest neighbors of some points.

► Code

► Code



Naive Bayes

k -NN classifies by *distance*. We close with a very different – and equally classic – way to attack the classification problem: via **probability**.

Given a data point $\mathbf{x}$, we want the label y . The key tool is **Bayes' rule**.

Bayes Rule

Let A, B be events from a sample space Ω . Recall that Bayes' rule states

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}.$$

This was derived in our [Probability and Statistics Refresher](#), and we used it in [GMMs](#) to compute the *unknown* $P(A|B)$ from *known* $P(B|A)$, $P(A)$, and $P(B)$.

Using Bayes Rule in a Classifier

Let's start with a simple example

A doctor knows that meningitis causes a stiff neck 75% of the time, i.e., $P(S|M) = 0.75$

If the doctor *also* knows the following information

Probabilistic Classification

Now let a data point $\mathbf{x}$ have attributes $x_1, \dots, x_d$ and let the possible classes be $C_1, \dots, C_k$.

Bayes' rule says

$$P(C_i|\mathbf{x}) = \frac{P(\mathbf{x}|C_i)}{P(\mathbf{x})} \cdot P(C_i),$$

and since $P(\mathbf{x})$ does not depend on C_i , the most probable class (the **MAP** estimate) is

$$\hat{C} = \arg \max_{C_i} P(x_1, \dots, x_d|C_i) \cdot P(C_i).$$

The “Naive” Assumption

Naive Bayes **assumes** the attributes are **conditionally independent** given the class:

$$P(x_1, \dots, x_d | C_i) = P(x_1 | C_i) \cdot P(x_2 | C_i) \cdots P(x_d | C_i).$$

This is *naive* – features are rarely independent – but it turns 10^{20} bins into 20×10 , and it works surprisingly well in practice.

Worked Example

Classify a new taxpayer with **Refund = No, Married** (ignoring income).

From the table, $P(\text{No}) = 7/10$ and $P(\text{Yes}) = 3/10$, and

$$\begin{aligned} P(\text{Refund}=\text{No}|\text{No}) &= 4/7, & P(\text{Married}|\text{No}) &= 4/7, \\ P(\text{Refund}=\text{No}|\text{Yes}) &= 3/3, & P(\text{Married}|\text{Yes}) &= 0/3. \end{aligned}$$

<i>Tid</i>	Refund	Marital Status	Taxable Income	Evade
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

That is Naive Bayes in a nutshell:

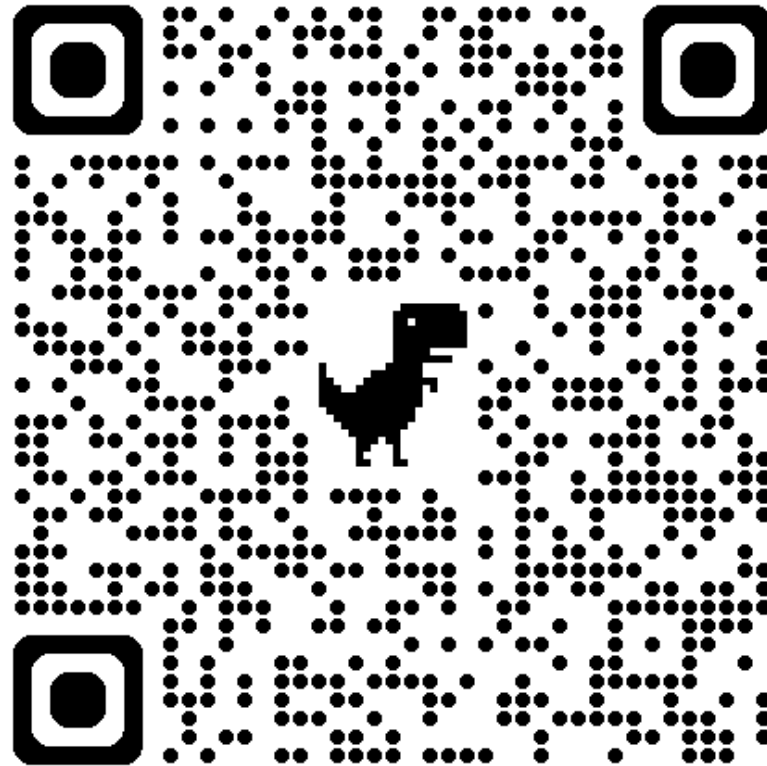
- classification via probability and Bayes' rule,
- training is just counting (or fitting a simple density) per feature and class,
- it scales to high dimensions because of the independence assumption – which is also its principal weakness.

The full treatment, including continuous features and **Support Vector Machines**, is in [Naive Bayes and SVM](#).

Summary

We covered the following topics:

- k -NN algorithm
- Curse of Dimensionality
- Worked with real datasets to perform model selection
- Metrics like precision and recall for evaluating classifier performance
- Naive Bayes: classification via Bayes' rule and the conditional-independence assumption



<https://bit.ly/3NeAwgx>