

# Module 1 - Prediction Problems

- 1.1 Introduction to Machine Learning
- 1.2 Nearest Neighbor Classification
- 1.3 Improving the Performance of Nearest Neighbor
- 1.4 Useful Distance Functions for Machine Learning
- 1.5 A Host of Prediction Problems

## 1.1 Introduction to Machine Learning

内容与 M0 - Introduction and Course Information 笔记中的 Introduction 部分一模一样。

### POLL

How prepared do you feel for this course?

### RESULTS

<input type="radio"/>	Very prepared. This should be easy.	22%
<input checked="" type="radio"/>	Somewhat prepared. I did alright in the previous course.	55%
<input type="radio"/>	Somewhat unprepared. The last course was quite a challenge.	21%
<input type="radio"/>	Completely unprepared. What am I doing here?	2%

# 1.2 Nearest Neighbor Classification

## Topics we'll cover

- 1 What is a classification problem?
- 2 The training set and test set
- 3 Representing data as vectors
- 4 Distance in Euclidean space
- 5 The 1-NN classifier
- 6 Training error versus test error
- 7 The error of a random classifier

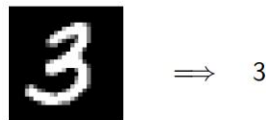
So today, we will get a feel for machine learning by looking at **one of the oldest and most enduring methods of classification: Nearest Neighbor.**

- This will let us understand what a classification problem is and also introduce some basic machine learning terminology like training set, test set, training error, and test error.
- We'll see **how data can be represented as vectors and Euclidean space and how we can compute distances in spaces like this.**

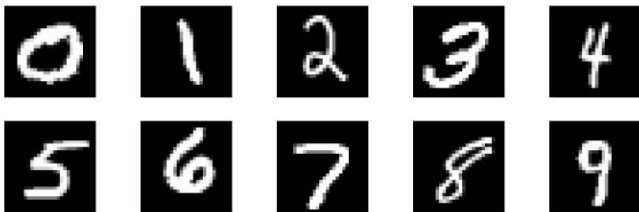
All this will give us the background we need to put the nearest neighbor classifier into action.

## The problem we'll solve today

Given an image of a handwritten digit, say which digit it is.



Some more examples:



## The machine learning approach

Assemble a data set:

1 4 1 0 1 1 9 1 3 4 8 5 7 2 6 8 0 3 2 2 6 4 1 4 1  
8 6 6 3 5 9 7 2 0 2 9 9 2 9 7 2 2 5 1 0 0 4 6 7  
0 1 3 0 8 4 1 1 1 5 9 1 0 1 0 6 1 5 4 0 6 1 0 3 6  
3 1 1 0 6 4 1 1 1 0 3 0 4 7 3 2 6 2 0 9 7 7 9 9  
6 6 8 9 1 2 0 8 6 7 0 8 5 5 7 1 3 1 4 2 7 9 5 5 4  
6 0 2 0 1 7 7 3 0 1 8 7 1 1 2 9 9 1 0 8 9 9 7 0 9  
8 4 0 1 0 9 7 0 7 5 9 7 3 3 1 9 7 2 0 1 5 5 1 7 0  
6 5 1 0 7 5 5 1 8 2 5 5 1 8 2 8 1 4 3 5 8 0 9 0 9  
4 3 1 7 8 7 5 2 1 6 5 5 4 6 6 5 5 4 6 0 3 5 4 6 0  
5 5 1 8 2 5 5 1 0 8 5 0 3 0 4 7 5 2 0 4 3 9 4 0 1

The MNIST data set of handwritten digits:

- **Training set** of 60,000 images and their labels.
- **Test set** of 10,000 images and their labels.

**And let the machine figure out the underlying patterns.**

So, the specific problem we will solve today is the following. We're given a picture or image of a handwritten digit and we want to say what digit it is. This, for example, is a three. Here are some more examples: we have a zero, a one, a two, **and so on and so forth. So how might we approach this problem?**

Here's an idea. A zero has got one loop in it. An eight has got two loops. A one has a single straight line. A four has three straight lines, and so on. What if we had a piece of software that took an image and figured out how many loops it had and how many straight lines, and also the relative positions of these loops and straight lines. Maybe we could then use this information and write down a bunch of simple rules to decide what digit it is. **Actually, people tried this a long time ago and they ran into a lot of problems.**

- So first of all, handwritten digits are super noisy and so it's hard to robustly pull out this information about the loops and lines and so on.
- Then, there's a lot of variability in the way people write fours, sevens, nines, et cetera. What this meant was that a huge number of rules were needed to account for all the different special cases and then, after all this trouble, the systems didn't really work well at all.

**So what we'll look at today is an entirely different approach, the machine learning approach. Rather than trying to figure out the underlying patterns ourselves, we'll just let the machine do it for us.**

Now, in order to figure out the patterns, what the machine needs above all else is a huge amount of data. So what we do is **we assemble a large data set of handwritten digit images, each labeled with the correct digit.** So the **MNIST data set** has got 60,000 images of handwritten digits. Here's a smattering of them. And we can use this training set to learn a classifier. A function that takes an image and then outputs what digit it thinks it is. MNIST also has a separate test set of 10,000 images along with their labels, and we can use this test set to assess how good our classifier really is.

总结:

## Nearest neighbor classification

Training images  $x^{(1)}, x^{(2)}, x^{(3)}, \dots, x^{(60000)}$

Labels  $y^{(1)}, y^{(2)}, y^{(3)}, \dots, y^{(60000)}$  are numbers in the range 0 – 9

```
1 4 1 0 1 1 9 1 5 4 8 5 7 2 6 8 0 3 2 2 6 4 1 4 1
8 6 6 3 5 9 7 2 0 2 9 9 2 9 7 2 2 5 1 0 0 4 6 7
0 1 3 0 8 4 1 1 1 5 9 1 0 1 0 6 1 5 4 0 6 1 0 3 6
3 1 1 0 6 4 1 1 1 0 3 0 4 7 5 2 6 2 0 0 9 7 7 9 9
6 6 8 9 1 2 0 8 6 7 8 8 5 5 7 1 3 1 4 2 7 9 5 5 4
6 0 2 0 1 8 7 3 0 1 8 7 1 1 2 9 9 1 0 8 9 9 7 0 9
8 4 0 1 0 9 7 0 7 5 9 7 3 3 1 9 7 2 0 1 5 5 1 9 0
6 5 1 0 7 5 5 1 8 2 5 5 1 8 2 8 1 4 3 5 8 0 1 0 9
4 3 1 7 8 7 5 2 1 6 5 5 4 6 6 5 5 4 6 0 3 5 4 6 0
5 5 1 8 2 5 5 1 0 8 5 0 3 0 4 7 5 2 0 4 3 9 4 0 1
```



How to **classify** a new image  $x$ ?

- Find its nearest neighbor amongst the  $x^{(i)}$
- Return  $y^{(i)}$

## The data space

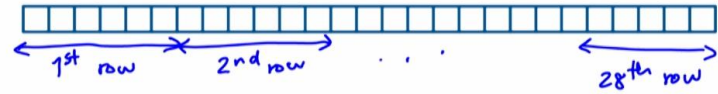
How to measure the distance between images?



MNIST images:

- Size  $28 \times 28$  (total: 784 pixels)
- Each pixel is grayscale: 0-255

Stretch each image into a vector with 784 coordinates:



- Data space  $\mathcal{X} = \mathbb{R}^{784}$
- Label space  $\mathcal{Y} = \{0, 1, \dots, 9\}$

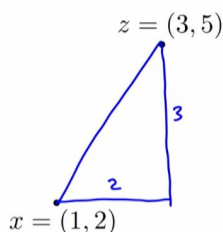
So what kind of classifiers might we try? Well, the simplest one imaginable perhaps, is **Nearest Neighbor**. **What happens in this case is that when we get a new image to classify, say this one over here, we get some new image X. We go through our training set of 60,000 images and we find the one that's closest to X. Then we simply return the label of that image.** That's it.

Now, there are some details that we have to work out. If we are looking for the one that's closest to X, it means we have some notion of distance between images. How are we representing images on the computer anyway? So let's look into that.

- So first off, we will represent images as vectors, okay? Now an MNIST image is 28 pixels by 28. So it's 28 pixels across, 28 pixels high. That means the total number of pixels is 28 and 28 which is 784.
- And each pixel is grayscale. So it's a value in the range zero to 255 where zero means black and 255 means white. For example, the pixels in the upper corner over here are all zero, whereas some of the pixels in the middle over here are probably much closer to 255.
- What we'll do with an image like this in order to make it into a vector is to simply stretch it out into one long, 784 dimensional vector. Something like this. And how do we stretch it out? Well, we begin by just copying down the first row. So we copy down the first row over here. Then we copy down the second row. And all the way to the last row. So these initial positions are all zeros. Somewhere in the middle, we have numbers like 200 and towards the end, we have zeros again. So we've taken the image and converted it into a 784 dimensional vector. Our data space then, which we're gonna denote by script X is 784 dimensional Euclidean space and we'll often write it like this:  $\mathbb{R}$  to the 784th. The label space just consists of the possible labels, zero to nine.

## The distance function

Remember Euclidean distance in two dimensions?



$$\sqrt{2^2 + 3^2} = \sqrt{13}$$

## Euclidean distance in higher dimension

Euclidean distance between 784-dimensional vectors  $x, z$  is

$$\|x - z\| = \sqrt{\sum_{i=1}^{784} (x_i - z_i)^2}$$

Here  $x_i$  is the  $i$ th coordinate of  $x$ .

Now that we have a specific vector representation, we also have to decide how we're going to compute distances between vectors and the most common, or default distance function is perhaps just Euclidean distance.

So let's recall how this works in two dimensions. When you have two points, the Euclidean distance between them is just the length of the line connecting them. So it's the length of this line. And what is that length? Well, if you look at these two points, X and Z, along the first coordinate, they differ by two and along the second coordinate, they differ by three. So the length of the line, the distance from X to Z is simply the square root of two squared plus three squared which is the square root of 13. That's the Euclidean distance between X and Z in two dimensions.

Now of course we aren't working in two dimensions. We're working in a much higher dimensional space but the basic idea is the same. When you want to compute the distance between two vectors, X and Z, you simply find out how much they differ on each individual coordinate, you square these values, you add them up and then you take the square root of the whole thing. That's Euclidean distance. Good.

## Nearest neighbor classification

Training images  $x^{(1)}, \dots, x^{(60000)}$ , labels  $y^{(1)}, \dots, y^{(60000)}$

1 4 1 6 1 1 9 1 3 4 8 5 7 2 6 8 0 3 2 2 6 4 1 4 1  
8 6 6 3 5 9 7 2 0 2 9 9 2 9 7 2 2 5 1 0 0 4 6 7  
0 1 3 0 8 4 1 1 4 5 9 1 0 1 0 6 1 5 4 0 6 1 0 3 6  
3 1 1 0 6 4 1 1 1 0 3 0 4 7 3 2 6 2 0 0 9 7 7 9 9  
6 6 8 9 1 2 0 4 6 7 8 5 5 7 1 3 1 4 2 7 9 5 5 4  
6 0 1 0 1 3 7 3 0 1 8 7 1 1 2 9 9 1 0 8 9 9 7 0 9  
8 4 0 1 0 9 7 0 7 5 9 7 3 3 1 9 7 2 0 1 5 5 1 9 0  
6 5 1 0 7 5 5 1 8 2 5 5 1 8 2 8 1 4 3 5 8 0 9 0 9  
4 3 1 7 8 7 5 5 1 6 5 5 4 6 6 5 5 4 6 0 3 5 4 6 0  
5 5 1 8 2 5 5 1 0 8 5 0 3 0 4 7 5 2 0 4 3 9 4 0 1

3

To classify a new image  $x$ :

- Find its nearest neighbor amongst the  $x^{(i)}$  using Euclidean distance in  $\mathbb{R}^{784}$
- Return  $y^{(i)}$

How accurate is this classifier?

Now we have a representation of the images as vectors in 784 dimensional space and we have a distance function between images. So we're ready to use nearest neighbor. **Each time we get a new image, we simply find its nearest neighbor using Euclidean distance in 784 dimensional space and we return the label of this training image.**

So how good is this classifier? Well, let's look at some numbers. First of all, what is the **error rate** of the classifier on the training points? So we have these 60,000 training images. For any training point, its nearest neighbor in the training set is itself. So it'll definitely get the right label. **So the error rate on the training set is zero. What that means is that training error is not a good predictor of future performance. It in general is something that is overly optimistic. That's why we have a separate test set. If we compute the error on those separate 10,000 points, that's really a much better indication of how well this classifier is gonna perform in practice.**

Now, what kind of test error might we expect? Well, **let's do a little toy experiment.** Suppose that we use the classifier that was completely random. When it was given an image, it didn't even look at the image but just randomly chose a number from zero to nine. What would be the error rate of a classifier like this? Well, whatever the correct label is, the chance that it randomly picks that correct label is 10%. **So a random classifier has got an error rate of 90%. We certainly want to do better than that.** Now let's see how well nearest neighbor does. On the test set, its error rate is 3.09%. That means that out of the 10,000 points, it gets 309 of them wrong. That's not too bad for such a simple method.

## Examples of errors

Test set of 10,000 points:

- 309 are misclassified
- Error rate 3.09%

Examples of errors:

Query					
NN					

Let's look at some of the mistakes that it makes.

This query, for example. When it was looking for its nearest neighbor, this is the point it found. So it thought it was a four.

Look at this one. Its nearest neighbor in the training set turned out to be this point and so it thought it was an eight, and so on and so forth.

These errors are all quite understandable once you see what the nearest neighbor classifier is doing. So, we have our first classifier now.

And next time, we'll see how to make it better.

POLL

Which of the following is likely to most increase the Euclidean distance between two identical number "1"s?

RESULTS

- ☐ If one of the numbers was shifted right by 5 pixels 23%
- ☒ If one of the numbers was shifted down by 5 pixels 7%
- ☒ Both will yield the same distance 49%
- ☐ Not enough information is provided 21%

计算 Euclidean distance 的公式，只与相对应的每个元素的差值有关，而与元素的位置无关。故这个问题所说的两种情况的 Euclidean distance 结果是一样的。

总结：



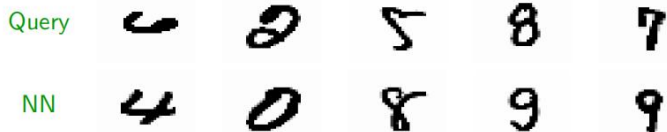
# 1.3 Improving the Performance of Nearest Neighbor

## Recall: nearest neighbor on MNIST

- Images of handwritten digits, represented as vectors in  $\mathbb{R}^{784}$ .
- Labels 0 – 9
- Training set: 60,000 points; test set: 10,000 points

Test error of nearest neighbor using Euclidean distance: 3.09%

Examples of errors:



Ideas for improvement: (1)  $k$ -NN (2) better distance function.

So last time, we saw the basics of nearest neighbor classification. And now we'll see how to take it to the next level.

So if you remember, we tried out nearest neighbor on the MNIST data set of handwritten digits. And we found that it gets pretty reasonable performance, an error rate of 3.09% on a separate test set. And these are some examples of the mistakes that it made.

How can we improve its performance even further? Let's look at two standard ways of doing this.

- The first is to move to  $K$ -nearest neighbors.
- The second is to look for better distance functions.

## $K$ -nearest neighbor classification

To classify a new point:

- Find the  $k$  nearest neighbors in the training set.
- Return the most common label amongst them.

MNIST:	$k$	1	3	5	7	9	11
	Test error (%)	3.09	2.94	3.13	3.10	3.43	3.34

In real life, there's no test set. How to decide which  $k$  is best?

## Cross-validation

How to estimate the error of  $k$ -NN for a particular  $k$ ?

### 10-fold cross-validation

- Divide the training set into 10 equal pieces.  
Training set (call it  $S$ ): 60,000 points  
Call the pieces  $S_1, S_2, \dots, S_{10}$ : 6,000 points each.
- For each piece  $S_i$ :
  - Classify each point in  $S_i$  using  $k$ -NN with training set  $S - S_i$
  - Let  $\epsilon_i$  = fraction of  $S_i$  that is incorrectly classified
- Take the average of these 10 numbers:

$$\text{estimated error with } k\text{-NN} = \frac{\epsilon_1 + \dots + \epsilon_{10}}{10}$$

So  $K$ -nearest neighbor classification, this is a very simple idea. When doing nearest neighbor, instead of simply looking for the very closest point in the test set. Find the closest three points, or the closest five points. They each have a label. Return the majority label, or the most common label. Let's see how this does on MNIST.

Okay. So, when  $K$  equals one, this is just the same nearest neighbor classifier we saw last time. And it has an error rate of 3.09%. When  $K$  equals three what we're doing in order to classify a new image is finding its three closest images in the training set. And returning their majority label. If we do this, the error rate goes down slightly, to 2.49%. When we try larger values of  $K$ , five, seven, nine, and so on, the error rate starts going up again. Now one thing that's important to mention over here, is that these errors are measured on the separate test set. Measuring errors on the training set is really a very poor indication of future performance. So on this particular data set, varying  $K$  helps a little bit, but not dramatically. In other cases, it sometimes helps a lot.

But how do we choose the right value of  $K$ ? In this toy example we had a separate test set, but that's not something we typically have in real life. We'd only have the training set and we've already made it clear that error on the training set is rather a poor indication of error in practice. How do we deal with this? This is a rather tricky problem. And it's a problem that's not limited to nearest neighbor. This is something we see over and over again in machine learning. Many of the methods we study will have parameters, like  $K$ , that need to be set correctly. If we set them well, the method works well. If we set them poorly, the method works poorly. And we have to somehow set them using the training set alone. A standard way of doing this is by something called **cross-validation**. So let me show you how this works for  $K$ -nearest neighbor.

Suppose we want to evaluate a particular choice of  $K$ . For example, three nearest neighbor,  $K$  equals three. We want to estimate the error rate of three nearest neighbor, but using only the training set. How would we do this? So here's how **10-fold cross-validation** would work. We take our training set, those 60,000 points, and we divide it into ten equal chunks. So ten chunks of 6,000 points each. Let's call these  $S_1, S_2, S_3, S_4$ , all the way to  $S_{10}$ . Now for each of these  $S$  sub  $i$ 's, we'll do the following. We'll take just that chunk and we'll think of it as the test set. And we'll think of the remaining nine chunks as the training set. And for each of the 6,000 points in  $S$  sub  $i$ , we will classify it using the remaining 54,000 points. That will give us an error rate, for  $S$  sub  $i$ , which we'll call  $\epsilon$  sub  $i$ , and once we've done this for each of these pieces, we have these 10 error rates,  $\epsilon$  one through  $\epsilon$  ten. Each of these individually is a fairly decent estimate for the error rate of  $K$ -nearest neighbor. To get an even better estimate, we average the 10 of them, and so this is our final estimate of the error of  $K$ -nearest neighbor for a specific value of  $K$ . Now of course, we will want to experiment with multiple values of  $K$ . We'll try  $K$  equals one,  $K$  equals three,  $K$  equals five and so on, and in each case, for each of these values of  $K$ , we'll run 10-fold cross-validation, we'll get an error estimate and then we'll pick the  $K$  with the lowest error estimate, and that's how we find the  $K$ -nearest neighbor.

Now what I've described over here is called 10-fold cross-validation because we divide the data set into 10 equal pieces. But we could also do five fold cross-validation, where divide it into five pieces. So the 60,000 points, will be divided into pieces of 12,000 points, or we could do eight-fold cross-validation or four-fold cross-validation for instance. But let's us choose  $K$ .

总结:

## Another improvement: better distance functions

The Euclidean ( $\ell_2$ ) distance between these two images is very high!



Much better idea: distance measures that are invariant under:

- Small translations and rotations. e.g. **tangent distance**
- A broader family of natural deformations. e.g. **shape context**

Test error rates:

$\ell_2$	tangent distance	shape context
3.09	1.10	0.63

So we've seen one approach to improving nearest neighbor by looking at more neighbors, by looking at K-nearest neighbors instead of just one. Another strategy which works very well in many settings is to look for better distance functions. Now when we were running nearest neighbor on MNIST, we used Euclidean distance, but **actually Euclidean distance is not a good choice for images in general**. So look at these two images for instance, they are actually exactly the same, except one of them is slightly scrolled to the right. Okay? Not a big difference to us, but what it means is that their vector representations are completely different and so there's actually a significant L2 distance between them. Clearly Euclidean distance is not a great choice for images. So what we'd like is to come up with a different distance function that doesn't behave badly in this way.

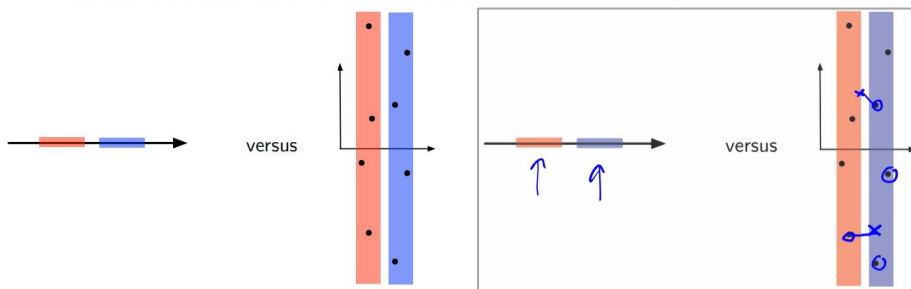
- We'd like a distance function, which doesn't change if you take one of the images and translate it slightly or rotate it slightly, and people have looked into this and have come up with such a distance function called **tangent distance**.
- Even better, would be a distance function that is invariant under an even larger family of deformations, not just small translations and rotations but maybe, for example, taking a line and making it slightly curved. One example of such a distance function is called **shape context**.

These are better distance functions and what happens when you use these for doing nearest neighbor, what happens to the performance on MNIST? And as we can see, it improves dramatically. With shape context, for example, the error rate drops well below 1%. **In general, in using domain knowledge to pick better data representations and better distance functions and better similarity functions, can be very, very beneficial in classification**, and so something that might be well worth the effort if you really want an accurate classifier.

## Related problem: feature selection

Feature selection/reweighting is part of picking a distance function.

**And, one noisy feature can wreak havoc with nearest neighbor!**



**Now one aspect of choosing a distance function is first just choosing the right features.**

In the case of MNIST, the features we were using were the 780 individual pixels in the image and these are all features that are arguably relevant to classification. **But in many other situations, the data has lots of features that are completely irrelevant to the classification task at hand.**

So for example, suppose you are building a classifier that takes loan applications and tries to decide whether the person is at risk of default. Now the data in this case, the loan application contains a lot of useful information, like age and income and so on, but also contains a lot of information that's completely irrelevant

like social security numbers if you don't remove these features, they can wreak havoc with nearest neighbor. The distance computations could be completely dominated by these features.

So here's a pictorial depiction of this. So on the left over here we have a data set that's just in one dimension and it has two labels. Red and blue. The points with smaller X value are red, the points with higher X value are blue. They're nicely separated and nearest neighbor will work extremely well on this one dimensional data. On the right over here, we decide to add in an extra feature along the X2 axis. This feature is completely irrelevant, and it completely wrecks the performance of nearest neighbor. So for example, let's say that these dots over here are the training points. How would this point be classified? Suppose this is a test point. Well its nearest neighbor is this, so it would be classified as red. Uh oh, that's not good. How would this point be classified? Well its nearest neighbor is this, so it would be classified as blue, not good at all.

**So feature selection is very important prior to using nearest neighbor.**

**总结:**

## Algorithmic issue: speeding up NN search

Naive search takes time  $O(n)$  for training set of size  $n$ : slow!

Luckily there are data structures for speeding up nearest neighbor search, like:

- 1 Locality sensitive hashing
- 2 Ball trees
- 3  $K$ -d trees

These are part of standard Python libraries for NN, and help a lot.

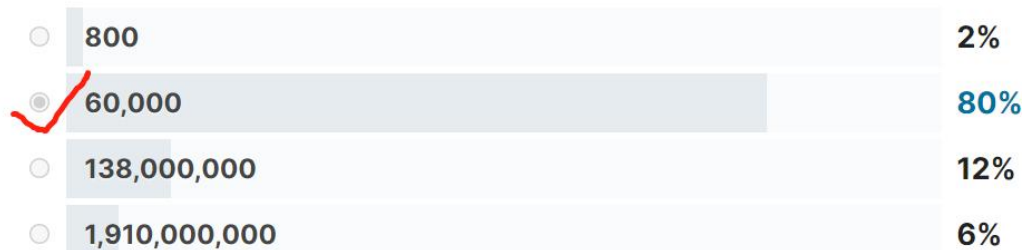
Now so far, we've been talking about **improving the statistical performance** of nearest neighbor, including improving the classification accuracy of nearest neighbor. But another thing is very important is **the algorithmic side of things**. **Nearest neighbor search can actually be rather slow if the training set is large**. When we were doing nearest neighbor on MNIST, for example, in order to classify a new image, we had to look through 60,000 training images. This is not pleasant, can you imagine if the training set instead had a million images or 100 million images? It would render nearest neighbor completely impractical. Well formally, there's a training set of size  $N$ . A naive search for the nearest neighbor takes time proportional to  $N$ , which is very bad. The good news is that this problem is something that has been researched for several decades now and people have come up with a variety of data structures for significantly speeding up nearest neighbor search. These data structures have names like locality sensitive hashing, **ball trees**,  **$K$ -d trees**, and **there's a whole lot more of them**. **The main thing is to be aware of them. They're part of standard libraries and Python for example, and in some cases, they can reduce the search time from something like  $N$  to something more like  $\log N$ , and in general they are quite helpful.**

Well, nearest neighbor has been our introduction, our first window into machine learning and we've used it to discuss basic concepts like training error and test error, feature selection, and cross-validation. Very soon we will come across all sorts of other methods for classification, but nearest neighbor is the oldest of them all and one of the simplest and most flexible.

### POLL

Suppose brute-force linear search is used to answer a single 1-NN query on the MNIST dataset. Approximately how many distance computations will be performed?

### RESULTS



### FEEDBACK

60,000



## 1.4 Useful Distance Functions for Machine Learning

### Topics we'll cover

- 1  $L_p$  norms
- 2 Metric spaces

So when we were discussing nearest neighbor classification, we kept emphasizing how important it is to pick the right distance function.

Now it turns out that distance functions are critical to many different types of machine learning and that there are a few distance functions that keep popping up again and again. So what we'll be doing today is to look at two families of such distance functions, the  **$L_p$  norms** and **metric spaces**.

### Measuring distance in $\mathbb{R}^m$

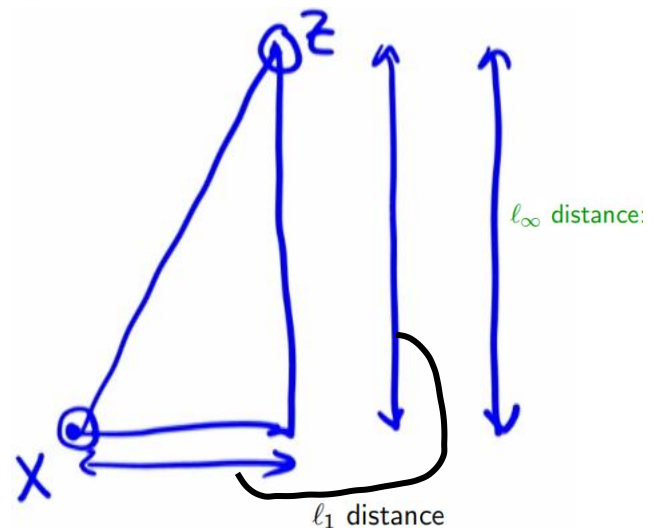
Usual choice: **Euclidean distance**:

$$\|x - z\|_2 = \sqrt{\sum_{i=1}^m (x_i - z_i)^2}.$$

For  $p \geq 1$ , here is  $\ell_p$  distance:

$$\|x - z\|_p = \left( \sum_{i=1}^m |x_i - z_i|^p \right)^{1/p}$$

- $p = 2$ : Euclidean distance
- $\ell_1$  distance:  $\|x - z\|_1 = \sum_{i=1}^m |x_i - z_i|$
- $\ell_\infty$  distance:  $\|x - z\|_\infty = \max_i |x_i - z_i|$



Let's start with  **$L_p$  norms**. What's a good way to measure distance in  $m$  dimensional Euclidean space? Well, there's L2 distance, which we saw last time. If you take out a ruler and you measure the distance between two points, that's L2 distance. It's a very simple, intuitive distance function and in many cases, it's the default choice. But it turns out that L2 is just one member of a much larger family of distance functions, called the  **$L_p$  distances**. And here's the general form of an  $L_p$  distance. So  $p$  can be any number from one to infinity. And to compute the  $L_p$  distance between two vectors,  $x$  and  $z$ , you'll look at their difference along each coordinate, you raise it to the  $p$  power, you add this up across all the coordinates, and then you take the  $p$  root of the whole thing. So, when you plug in  $p$  equals two for example, you just get back the previous formula, the Euclidean distance.

But when you plug in a different value of  $p$ , like  $p$  equals one, for instance, you get a different distance function. For  $p$  equals one, you get **L1 distance**, which is something that is also used a lot in machine learning. Okay, so what is this distance function? Well, let's say we have two points,  $x$  and  $z$ , so this is  $x$ , and that's  $z$  over there. L2 distance is just distance as the crow flies (crow fly, 直线距离), that's the one we've seen. That's a very simple distance. In L1 distance, you also wanna get from  $x$  to  $z$ , but you're not allowed to go as the crow flies. You are forced to go along horizontals and verticals. So the L1 distance from  $x$  to  $z$  is this, plus this. It's the sum of those two legs. So mathematically, what we do, is that we look at the difference along each coordinate, and we simply add up those differences. So this is the difference along the first coordinate, this is the difference along the second coordinate, and we add those two up and that's L1 distance.

A very important distance function. Another interesting choice is **L infinity distance**, which is the last one here in green. What L infinity distance does, is to simply look at the single coordinate along which the distance between  $x$  and  $z$  is the greatest. The single coordinate with the largest difference  $x_i$  minus  $z_i$ . So in the case of these two points over here,  $z$  and  $x$ , the L infinity distance between them, would simply be this. So you might wonder how we can plug in  $p$  equal to infinity in that formula over there. We can't,  $p$  has to be a finite number. But what we can do is to take larger and larger values of  $p$  and then look at the limit. And that's how we get our L infinity distance.

Okay, so these are three functions to bear in mind. L1, L2, and L infinity.

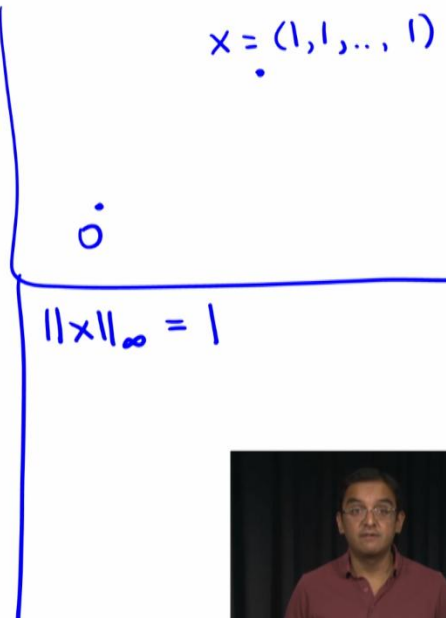
总结:

## Example 1

Consider the all-ones vector  $(1, 1, \dots, 1)$  in  $\mathbb{R}^d$ .  
What are its  $\ell_2$ ,  $\ell_1$ , and  $\ell_\infty$  length?

$$\begin{aligned}\|x\|_2 &= \sqrt{1^2 + 1^2 + \dots + 1^2} \\ &= \sqrt{d}\end{aligned}$$

$$\begin{aligned}\|x\|_1 &= |x_1| + \dots + |x_d| \\ &= d\end{aligned}$$

$$x = (1, 1, \dots, 1)$$


$$\|x\|_\infty = 1$$



Let's see some examples of these now. Let's talk about lengths of vectors. So the length of a vector is simply its distance from the origin. Let's say we have a vector over here, which is all ones, and this is in  $d$  dimensional space. The vector consists of  $d$  1s. The length of the vector is its distance from the origin, the all zeros point. And we wanna compute its length in  $L_2$ ,  $L_1$ , and  $L$  infinity.

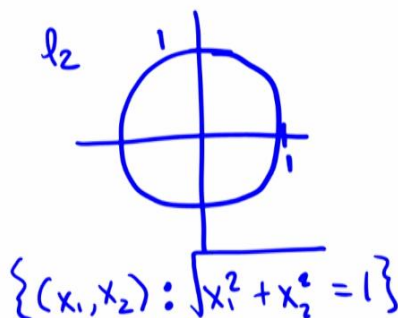
- So let's do  $L_2$  first, since that is the most familiar distance function. So we wanna compute the  $L_2$  norm of  $x$ . Well, plugging into the formula, the  $L_2$  norm is the square root of the first coordinate squared plus the second coordinate squared, all the way to the  $D$ th coordinate squared. Okay, so the  $L_2$  norm of  $x$ , is just the square root of  $d$ .
- What about the  $L_1$  norm of  $x$ . Let's draw this line over here. The  $L_1$  norm of  $x$  is the first coordinate absolute value, plus the second coordinate, plus all the way to the  $D$ th coordinate. And so the  $L_1$  norm is  $d$ .
- Now, let's do the  $L$  infinity norm. The  $L$  infinity norm of  $x$  is the coordinate along which the value is the largest, and in this case, one could pick any of the coordinates. So the  $L$  infinity norm is just 1.

So the three norms give very different values.

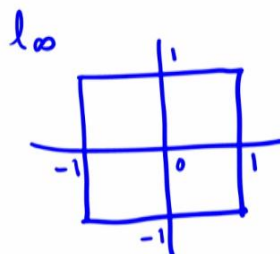
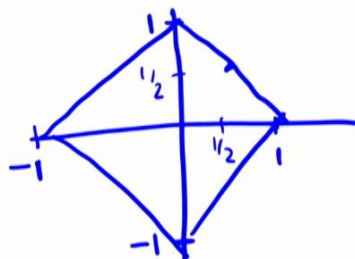
## Example 2

In  $\mathbb{R}^2$ , draw all points with:

- 1  $\ell_2$  length 1
- 2  $\ell_1$  length 1
- 3  $\ell_\infty$  length 1



$$\ell_1: \{(x_1, x_2) : |x_1| + |x_2| = 1\}$$



Let's look at another example. So now we are in  $\mathbb{R}^2$ , which is the plane.

- And we wanna draw all points whose  $L_2$  length is one. So that's something that you might be familiar with, that's just a circle. Every point whose length is one. So in the  $L_2$  case, we have a circle. The unit circle. Okay, so this point is one, that point is one, and formally, what we're looking for, is all points  $x_1, x_2$ , whose  $L_2$  norm is one, in other words, for which  $x_1^2 + x_2^2 = 1$ . And this is a circle. This is the formula for the unit circle. So that's the familiar case.
- Now let's try  $L_1$ . So we want all points whose length is one. So we want all points in the plane, all points  $x_1, x_2$ , and we want the length to be one. So the absolute value of  $x_1$  plus the absolute value of  $x_2$ , equals one. Let's see what points these are. Okay so let's draw the plane over here. Okay, so one example of such a point is the point  $1, 0$ . Because the coordinates add up to one. Another one is the point  $0, 1$ . Now we're taking absolute value, so we can also do  $-1, 0$  and  $0, -1$ . So these are four points that lie on this shape. But we also have, for example,  $1/2, 1/2$ , which lies in the middle over here. And when we finally join all these points together, we see that it looks like this. It has a diamond shape. Okay, my diamond is a little bit skewed, but you can imagine what it's supposed to look like. So that is the unit ball for the  $L_1$  norm.
- And what is it for  $L$  infinity? Well, this is what it turns out to be. Instead of a diamond, it just turns out to be a box. So that's  $0, 1, -1$ . And maybe you can check for yourself to see that that's really correct.

总结:

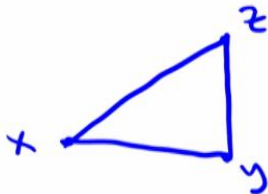
# Metric spaces

$$d(x, x') = 3.6$$

Let  $\mathcal{X}$  be the space in which data lie.

A distance function  $d : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$  is a **metric** if it satisfies these properties:

- $d(x, y) \geq 0$  (nonnegativity) ←
- $d(x, y) = 0$  if and only if  $x = y$  ←
- $d(x, y) = d(y, x)$  (symmetry) ←
- $d(x, z) \leq d(x, y) + d(y, z)$  (triangle inequality) ←



Okay, so we've talked a little bit about  $L_p$  norms. And these are distance functions that really come up a lot, especially  $p$  equals one to an infinity. So they're gonna turn out to be very useful. But there are many situations in which we need other distance functions. So for example, distance functions that are fine-tuned to a particular domain. Or distance functions between objects that aren't even vectors. You know, distance functions between strings or graphs. These sort of things come up all the time in machine learning. So is there a broader family of distances, something that, for example, can be a distance function on an arbitrary space, not necessarily a space of vectors. And there are a few such families, and perhaps the most important of them are **distance metrics** (距离度量). So let's go over the definition of this.

So we have  $X$  which is any data space. It could consist of vectors. It could be trees. It could be strings. Any space, an arbitrary space. Now we have a distance function on  $X$ , so it's a distance function where you give it two objects in  $X$  and it returns a value. You know, maybe 3.6 or something like that. Now, this distance function is called a metric if it happens to satisfy four properties. Four basic properties.

- The first is that the distances should never be negative. That sounds fairly reasonable.
  - The second is, that the distance from a point to itself should be zero, and moreover, these are the only cases in which the distance should be zero. So it should not be the case that the distance between two different points is zero.
  - The third property is that the distances should be symmetric. So the distance from  $x$  to  $y$  should be the same as the difference from  $y$  to  $x$ .
  - The final property is the triangle inequality. So what that says is that if you take any three points,  $x$ ,  $y$ , and  $z$ , then the distance from  $x$  to  $z$  is at most the distance from  $x$  to  $y$ , plus the distance from  $y$  to  $z$ . And if that holds, it satisfies the triangle inequality.
- So, any distance function that happens to satisfy these four properties is called a metric. And if we find that the distance function that we are using is a metric, it's useful, because there are all sorts of things we can do with it. For example, last time, we talked about methods for fast nearest neighbor search. These data structures like ball trees and  $k$ -d trees and locality-sensitive hashing and so on. Now, many of those methods work only for Euclidean distance. Work only if you're doing nearest neighbors in Euclidean space. But some of them work for arbitrary metrics. So if the distance function you've chosen happens to be a metric, then you can use these to do fast nearest neighbor search. So, it's very useful to be able to choose distances that are metrics.

So let's look at some examples of metric distances.

## Example 1

$$\mathcal{X} = \mathbb{R}^m \text{ and } d(x, y) = \|x - y\|_p$$

$$d(x, y) = \sum_{i=1}^m |x_i - y_i|$$

Check:

- $d(x, y) \geq 0$  (nonnegativity) ✓
- $d(x, y) = 0$  if and only if  $x = y$  ✓
- $d(x, y) = d(y, x)$  (symmetry) ✓
- $d(x, z) \leq d(x, y) + d(y, z)$  (triangle inequality) ✓

$$|x_i - z_i| \leq |x_i - y_i| + |y_i - z_i|$$

sum over all  $i$

And the first example is the  $L_p$  norms. It turns out that any  $L_p$  distance is a metric. So let's pick one concretely. Let's say  $L_1$  distance. So let's say that the distance between two points is the  $L_1$  distance, and let's say that the points are in  $m$  dimensional space. So we take the sum over the  $m$  coordinates of the difference between the two vectors along that coordinate. That's the  $L_1$  distance. Why is this a metric? Well, in order to check that, we just have to go down those four properties and check them one by one. So first of all, is this, can this ever be negative? No, because of the absolute value. So first property is okay. If  $x$  equals  $y$ , is this zero? Yeah, if  $x$  equals  $y$ , this is zero. If this is zero, does that mean  $x$  equals  $y$ ? Well, if this thing is zero, it means that all of these absolute values are zero, which means that  $x$  is equal to  $y$ . The second property's fine as well. Symmetry, is the distance from  $x$  to  $y$  the same as the difference from  $y$  to  $x$ , yes. What about the triangle inequality? Does  $L_1$  distance satisfy the triangle inequality? Well, one thing that's definitely true is that if you look at any one coordinate, the distance, the difference between  $x_i$  and  $z_i$  is at most the difference between  $x_i$  and  $y_i$  plus the difference between  $y_i$  and  $z_i$ . And now, we just sum over all coordinates. And that gives us the triangle inequality. So  $L_1$  distance satisfies all the properties of a metric, and all the  $L_p$  distances do.

## Example 2

$$\mathcal{X} = \{\text{strings over some alphabet}\} \text{ and } d = \text{edit distance}$$

Check:

- $d(x, y) \geq 0$  (nonnegativity)
- $d(x, y) = 0$  if and only if  $x = y$
- $d(x, y) = d(y, x)$  (symmetry)
- $d(x, z) \leq d(x, y) + d(y, z)$  (triangle inequality)

$$\mathcal{X} = \{A, C, G, T\}^*$$

$$x = A C C G T$$

$$y = C C G T$$

$$d(x, y) = \# \text{ of insertions, deletions, substitutions to get from } x \text{ to } y.$$

$$d(x, y) \geq 0$$

$$d(x, y) = 0 \Leftrightarrow x = y$$

$$d(x, y) = d(y, x)$$

$$\text{triangle inequality}$$

Now let's look at an example where the input space does not consist of vectors. So let's say that we're dealing with strings over some alphabet. For instance, if we are dealing with DNA sequences, then what we have are strings over A, C, G, and T. That's the input space. What the star means is strings of arbitrary length over this alphabet. So we have two strings. Let's say A-C-C-G-T and C-C-G-T. So we have these two strings. And now we want a distance function. What is the distance between  $x$  and  $y$ ? Now there are many ways in which we can define this, but from the biologist's point of view, they will look at these two, and say, "Well, you know, actually, these are very similar "because you can get from  $x$  to  $y$  by just deleting A." If you were to remove A, they become the same. Or equivalently, if you were to take  $y$ , and just insert an A at the beginning, you would get  $x$ . So this kind of distance function that takes into account insertions, deletions, and also substitutions, is called **edit distance**. And it turns out that it's a metric. So let's just define it. The edit distance between two strings,  $x$  and  $y$ , is the number of insertions, deletions, and substitutions needed to get from  $x$  to  $y$ . So why is this a metric? Well, once again, we just have to go through the properties. So first of all, is it always positive? Yes. Is it the case that it's zero if and only if  $x$  equals  $y$ ? Again, obviously, yes. Is it symmetric? Is the number of steps to go from  $x$  to  $y$ , the number of insertions, deletions, and substitutions to go from  $x$  to  $y$  the same as to go from  $y$  to  $x$ ? Yes, it is, because a deletion is the reverse operation of an insertion. And finally, does it satisfy the triangle inequality? It does and that's something you can also convince yourself. So here's an example of a distance function that's really over a fairly arbitrary space. A distance function between strings, but it turns out to be a metric, which means that we can do all sorts of nice things with it.

总结:



## A non-metric distance function

Let  $p, q$  be probability distributions on some set  $\mathcal{X}$ .

The **Kullback-Leibler divergence** or **relative entropy** between  $p, q$  is:

$$d(p, q) = \sum_{x \in \mathcal{X}} p(x) \log \frac{p(x)}{q(x)}.$$

$\downarrow$

$$p = \left(\frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8}\right)$$
$$q = \left(\frac{1}{6}, \frac{1}{3}, \frac{1}{3}, \frac{1}{6}\right)$$
$$d(p, q) = \frac{1}{2} \log \frac{1/2}{1/6} + \frac{1}{4} \log \frac{1/4}{1/3} + \frac{1}{8} \log \frac{1/8}{1/3} + \frac{1}{8} \log \frac{1/8}{1/6}$$



So the properties of a metric seem really minimal. It should be nonnegative. It should be symmetric. Should satisfy the triangle inequality. I mean, would we ever want to use a distance function that's not a metric? And, unfortunately, it turns out the answer is yes. And here, what I've shown here is **the prototypical example of a distance function that's very widely used and is nothing close to being a metric**. This is the **relative entropy**, or the **K-L divergence** and it's one of the most standard distance functions between probability vectors. So what's a **probability vector**? Well, a probability vector is just a vector that gives probabilities over different outcomes. So let's say we have four possible outcomes. An example of a probability vector is something like 1/2, 1/4, 1/8, 1/8. It says the probability of outcome one is 1/2. The probability of outcome two is 1/4. The probability of outcome three is 1/8. The probability of outcome four is 1/8. **So it's a bunch of positive numbers that add up to one**. And another probability vector over these four outcomes might be 1/6, 1/3, 1/3, 1/6. **So how do we measure the distance between these two probability distributions?** Well, one very natural way of doing that would be simply to look at the L1 distance between the vectors  $p$  and  $q$ . Or the L2 distance. And both of those are perfectly reasonable choices. But, it turns out that in machine learning, very very often, the distance measure of choice is the K-L divergence, which is summarized in this formula over here. So let's see what that works out to in this case. So the distance between  $p$  and  $q$  is the sum over all coordinates, so we're now gonna sum over the four coordinates. The first coordinate, we take  $1/2 \log 1/2$  over  $1/6$ . So we're looking at the first coordinate over here. We're comparing the two numbers  $1/2$  and  $1/6$ . Now we look at the second coordinate. We get  $1/4$ , that's  $p$  of  $x$ ,  $\log$ ,  $p$  of  $x$  is  $1/4$ , over  $1/3$  plus  $1/8 \log 1/8$  over  $1/3$ , plus  $1/8 \log 1/8$  over  $1/6$ . Very strange. **Now, thankfully, it turns out, that this can never be negative. But that's about where the good news ends. This is not a symmetric distance. So the distance from  $p$  to  $q$  is in general not the same as the difference from  $q$  to  $p$ . And it doesn't come close to satisfying the triangle inequality. But, it's a distance function we use all the time.**

Okay, so what we've done today is to talk a little bit about distance functions, and to be honest, we've really just scratched the surface in terms of organizing the space of possible distances. **Now complementary to distance functions are similarity functions**. And that's something that we'll be seeing a lot more of later on in the course.

POLL

Given the vector  $v = (1, 3, 5, 7, 9)$ , what is the  $\ell_1$  length of  $v$  ?

RESULTS

各元素减去0 (与原点为参考),  
取绝对值,  
然后累加



FEEDBACK

25

总结:

## 1.5 A Host of Prediction Problems

### Topics we'll cover

- 1 Machine learning versus algorithms
- 2 A taxonomy of prediction problems
- 3 Roadmap for the course

Our very first concrete machine learning problem was classifying handwritten digits. This is an example of a prediction problem. What I want to do today is to take a step back and to talk about prediction problems in a slightly more general and abstract way.

So, we'll begin with a few remarks about the difference between machine learning and algorithms, I'll then formalize the notion of a prediction problem and talk about a way in which we can organize this space, and finally, I'll end with a roadmap of the rest of this class.

### Machine learning versus Algorithms

A central goal of both fields:

*develop procedures that exhibit a desired input-output behavior.*

- **Algorithms:** input-output mapping can be precisely defined.  
Input: Graph  $G$ , two nodes  $u, v$  in the graph.  
Output: Shortest path from  $u$  to  $v$  in  $G$
- **Machine learning:** mapping cannot easily be made precise.  
Input: Picture of an animal.  
Output: Name of the animal.

Instead, provide examples of (input,output) pairs.

Ask the machine to *learn* a suitable mapping itself.

Okay, machine learning versus algorithms. So, in today's computer-driven world, two core technologies are machine learning and algorithms. What is the similarity or difference between these?

Well, interestingly, it turns out that both fields have a common central goal and that is to develop procedures, little pieces of code, that exhibit the desired input/output functionality.

- In algorithms, for example, people have spent decades looking for good ways to find shortest paths in graphs. So, the input is a graph along with two nodes in it, and the desired output is the shortest path between those two nodes. **An algorithm is a precise set of steps that leads from the input to the desired output.**
- In machine learning, we also want to come up with procedures that go from an input to a desired output, **but the sort of problems that we're dealing with are of a different nature so that it is very hard to list a precise set of steps that will get us to that output.** Okay, so for example, let's say the input is the picture of an animal and the desired output is the name of the animal. What are some precise steps that will do this for us? It's hard to imagine, **in fact, the problem isn't even precisely defined and so, rather than try and come up with the algorithm ourselves, what we do is to collect a whole bunch of XY examples of input/output pairs and then we ask the machine to figure out a mapping on its own.**

# Prediction problems: inputs and outputs

Basic terminology:

- The input space,  $\mathcal{X}$ .  
E.g.  $32 \times 32$  RGB images of animals.
- The output space,  $\mathcal{Y}$ .  
E.g. Names of 100 animals.



After seeing a bunch of examples  $(x, y)$ , pick a mapping

$$f : \mathcal{X} \rightarrow \mathcal{Y}$$

that accurately recovers the input-output pattern of the examples.

Categorize prediction problems by the type of **output space**:

(1) discrete, (2) continuous, or (3) probability values

So, this is a prediction problem. There is an input space, we'll always call it  $X$ , for example, the space of all images of animals, and then there's an output space, for example, names of animals. **We have in mind some desired mapping from inputs to outputs, but it's not something that's possible for us to necessarily specify in a precise way and so we collect a training set of  $XY$  examples.**

The learning machine takes this training set and uses it to pick a mapping from  $X$  to  $Y$ . A function that takes the image of an animal and returns the name of the animal, or it returns the name that it believes to be correct. Typically, the way a learning algorithm works is by looking for a function, a mapping, that does well on the training set.

Now, there are many, many different types of prediction problems out there and one way in which it's common to categorize them is according to the type of output space, and there are three cases that we typically distinguish.

- When the **output space is discrete**,
- when it's **continuous**,
- and when it consists of **probability values**.

It turns out that these three cases require somewhat different methods.

## Discrete output space: classification

### Binary classification

E.g., Spam detection  
 $\mathcal{X} = \{\text{email messages}\}$   
 $\mathcal{Y} = \{\text{spam, not spam}\}$

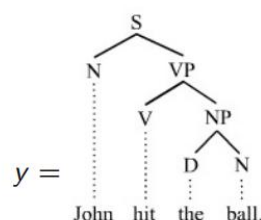
### Multiclass

E.g., News article classification  
 $\mathcal{X} = \{\text{news articles}\}$   
 $\mathcal{Y} = \{\text{politics, business, sports, ...}\}$

### Structured outputs

E.g., Parsing  
 $\mathcal{X} = \{\text{sentences}\}$   
 $\mathcal{Y} = \{\text{parse trees}\}$

$x = \text{"John hit the ball"}$



So, let's look at the first case. This is when the outputs are discrete and this is a case that we call **classification**. The simplest setting is **binary classification** where there are just two possible outputs, good or bad, plus or minus, yes or no. In spam detection, for example, the inputs  $X$  are email messages and the desired output is just spam or not spam. There are just two choices, it's binary. Very often, in classification problems, there are more than two possible outputs. So, for instance, maybe the input is a news article and the desired output is the subject matter of the article, politics, business, technology, sports, entertainment, et cetera. So, that's called **multiclass classification** and there are also cases in which the desired output is discrete but it's something more complex that has some combinatorial structure to it. In parsing, for example, the input is a sentence, like John hit the ball, and the desired output is the parse tree for that sentence, so this entire object over here. So, the output space is still finite, it's still discrete but it's more complex and it has a certain kind of structure to it. **What we'll be doing is devoting a large amount of attention to binary classification because this is a nice and simple setting in which to study prediction problems. It'll then turn out that the methods we develop can generalize quite easily to the other two settings as well, to multiclass and structured outputs.**

总结:

## Continuous output space: regression

- **Pollution level prediction**

Predict tomorrow's air quality index in my neighborhood

$\mathcal{Y} = [0, \infty)$  ( $< 100$ : okay,  $> 200$ : dangerous)

- **Insurance company calculations**

What is the expected life expectancy of this person?

$\mathcal{Y} = [0, 120]$

What are suitable predictor variables ( $\mathcal{X}$ ) in each case?

So we've been talking about categorizing prediction problems by the type of output space, so when the outputs are discrete, we call it a classification problem. When the outputs are continuous, then we have a **regression problem**. Let's look at a couple of examples.

- So, the first example here is, suppose we wanna predict the pollution level tomorrow and we're interested in this because, for instance, it will help us decide whether we are gonna let the kids go out tomorrow or whether we should keep them indoors. So, how does one measure pollution level? Well, one common way of doing it is by something called the **air quality index**, so this is a number, you know. A positive number that's less than a hundred means the air quality is not too bad. If it's more than a hundred, that means that it's not good, and if it's more than 200, it means that the air is absolutely dangerous. So, the output space now consists of positive numbers, it's a continuous space. Even if we only predict integer values, we still think of it as a continuous space because, for example, a prediction of a hundred is very close to a prediction of 101 or a prediction of 102, and is very far from a prediction of 200. **In other words, the Ys lie on a scale.**
- Let's look at another example. Insurance company calculations. So, one of the things that an insurance company is interested in when you apply for a policy is how much longer do they expect you to live, and this determines, for example, how much they would charge you. So here, the number we wanna predict, the Y, is the age at which you're going to die. Let's say it's something in the range zero to 120 and again, this is something that we could always round to the nearest integer, but we still think of it as a continuum because these numbers lie along a scale. Now, one interesting question is what there is to sort of think about is, what are suitable predictor variables for these kinds of regression problems? Okay, so for example, life expectancy. What are the sort of variables or what are the sort of pieces of information that might be helpful in determining this? Well, this is actually a very well-studied area and the sort of information that people use is your age, your gender, women tend to live longer, whether you smoke or not, do you have high blood pressure and are you taking medicine for it, do you have high cholesterol or are you taking medicine for it, do you smoke, and a few other such things.

## Probability estimation

$\mathcal{Y} = [0, 1]$  represents **probabilities**

Example: Credit card transactions

- $x$  = details of a transaction
- $y$  = probability this transaction is fraudulent

**Why not just treat this as a binary classification problem?**

So, the final kind of output space we'll look at is when the Ys represent probabilities. **So here, the output space is literally the range zero to one. Now, this seems a little bit like regression because it is a continuous range, but it turns out that this particular case does require specialized methods and so we tend to treat it separately.** So, an example here is credit card fraud detection, so X, the input, consists of the details of a credit card transaction. What is the amount of the purchase? What is being bought? What is the name of the merchant? What is the zip code? And so on. And Y here, the output we wish to predict, is what is the probability that this transaction is fraudulent? So, this is a **probability estimation problem**. One interesting question over here is, why not just think of this as binary classification? Why are we trying to predict the probability? Why not just say there are two possible labels, fraudulent or legitimate, and that's what we wanna predict? So, that's a good



question. It turns out that the reason we wanna use probabilities is because the results of this prediction are gonna be used as part of a larger decision-making framework and it's just one of the pieces of information that go into the decision. So, for instance, if a transaction has got a high probability of being fraudulent, then of course the transaction should be denied, but if the transaction has a low probability of being fraudulent, should it be denied or not? If it has just a small but significant probability of being fraudulent, should we accept it or deny it? Well, it might depend on other factors like the amount of the transaction. If it's just a small transaction, then maybe it's okay to take the risk. If it's a large transaction, then it should probably be denied. So, there are many factors that go into the decision and in order to assess the risks correctly, it's very useful to have not just a binary prediction, fraudulent or legitimate, but an actual probability value.

## Roadmap for the course

- 1 Solving prediction problems  
Classification, regression, probability estimation
- 2 Representation learning  
Clustering, projection, dictionary learning, autoencoders
- 3 Deep learning

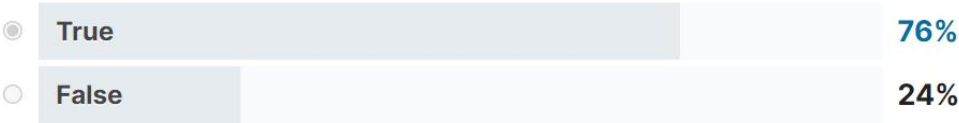
Okay, so we've been talking a lot about prediction problems and indeed, this is the bread and butter of machine learning, and we'll be spending a lot of time on this in the course. Is there anything other than prediction problems? Well, very often we see the dataset and we want to understand it better, and don't necessarily have a specific prediction task in mind. We just want to know, is there interesting structure in the data? Are there clusters in it, for example? So, the sort of topics that come under this, which we can call representation learning, are things like clustering, projection, dictionary learning, and so on. And we'll spend a little bit of time on this and we'll end with deep learning, which is, in a sense, a way of combining both representation learning and prediction problems.

Okay, well, that is a little bit of an overview of the course. Hopefully it's clear what lies ahead and next item we'll begin with the systematic treatment of classification.

### POLL

The output space for a prediction problem consists of integers in the range 0 to 100 representing student grades. For machine learning purposes, this is best thought of as a continuous output space.

### RESULTS



### FEEDBACK

True