Explanatory Notes for 6.390

Shaunticlair Ruiz (Current TA)

Fall 2022

Contents

CHAPTER 1

Introduction - Explanatory Notes

These are the explanatory course notes produced by **Shaunticlair Ruiz**, a current TA who has also served as an LA in Spring 2022. They are intended to be **supplementary** to the official lectures notes.

The official course lecture notes are designed to be **minimal**, and present what the instructors think that you absolutely **need** to **know** to understand and interact with the current state of machine learning.

These notes, by contrast, are designed to provide more thorough **explanations**. We **explain** certain logical leaps, **break down** concepts into smaller parts, and try to make the notes more **accessible** to students who find the primary notes too dense.

These notes cover the **same** topics as the primary notes, just with a different **presentation**. Most of the explanations in this document are a reaction to **difficulties** that students have had in previous semesters.

If the concepts in these explanatory note chapters are **familiar** to you, or if you find them to be too **drawn-out**, you can **skim** sections that you're not concerned about.

We, again, stress that neither set of notes is more "advanced", as they cover the **same material**. They simply reflect **different** learning styles and backgrounds.

It may be helpful to refer to these explanatory notes as you digest the official lecture notes: the main section numbers (1.1, 1.2, 1.3...) should **match** with the official notes.

If you have any concerns or points of **confusion**, feel free provide **feedback** on this ongoing project.

1.0.1 What is machine learning?

Why study machine learning? To answer that, let's learn what machine learning really *is*. Fortunately, it's all in the name.

Machine learning is a broad field. We use **machines**, or computers, and give them data to **learn** from.

Why are we teaching machines? Same as why we want **people** to learn: so they can use that learning to make good **decisions**.

So, in short, we can say:

Concept 1

The main focus of **machine learning** is making **decisions** or **predictions** based on **data**.

1.0.2 Why do Machine Learning: The Benefits

Why use machine learning? What is it good for?

The techniques used in machine learning have many applications. It has become the best way to handle many different problems:

Based on speed, time to develop, "robustness", etc.

- Facial detection
- Speech recognition
- Language processing
- Many problems that involve data or signal processing

Different ML techniques have become the best way to handle many problems in many fields. As a result, it has become very popular!

1.0.3 The role of humans

If machines can solve all of these problems, where do humans play a role?

Well, these machines aren't (yet) able to **set themselves up** to solve these problems: humans have to set up the system so the machines can succeed. We call this **framing** the problem.

We'll use an example to help explain.

- A human has to **recognize** that there is a problem to solve.
  - **Example:** You want to have self-driving cars. Your problem: those cars need to be able to **watch** the road.

- They have to decide what kind of **solutions** you want to try, and use that as the basis for training.
  - **Example:** You decide to create a **model** that can replicate vision for our car.
  - This **model** represents the kinds of **solutions** you expect to work: a particular model will allow for a certain approach to a situation.
- They have to **gather** data to train with.
  - **Example:** You might gather **videos** from dashcam footage, or create a virtual simulator for your car to drive in.
- They have to choose the **algorithms** we'll use for learning: what **instructions** do we give our computer?
  - **Example:** To "train" your model, you could need to adjust it to perform **better**. How do you adjust it, using the videos?
- They have to look at the final result and **validate** whether it's a good enough solution to use.
  - **Example:** You **test** out your model in a car: does it notice obstacles?
- They have to consider the possible **ethics** or other consequences of this solution.
  - **Example:** What's the most "responsible" way of driving? When should a car prioritize its own safety, or the safety of pedestrians? How much control should the user have?

This is over-simplified, but it gives us a high-level view of what we'll need going forward.

These are all important steps, and they require the human in question to make smart and responsible choices. That's why you need to learn machine learning: in order to use it **effectively**, you have to **understand** it!

We want to understand machine learning, so we'll break it down into different parts.

We'll do this by **asking** ourselves a couple **questions**, and thinking about machines in the broadest sense we can: as the **solution** to a **problem**.

This breakdown is different from the one above!

1.0.4 What's the plan?

We know that, in machine learning, we want to make **decisions** or **predictions** using **data**.

Let's frame this more generally: we want to **solve** a **problem** presented to us, using our **machine** and some **data**.

This brings up some questions:

- What exactly is our **problem**?
- And **solution** do we want to use?

The answer depends on the situation, but we can break down these questions into simpler, easier ones.

1.0.5 The Problem

Simply put, our goal is to create a **machine** that **takes in** data and **spits out** some kind of results.

In that way, our machine is just a **function**.

The **problem**, then, is to reach that goal: to get our desired output from our input.

That means we're focused on what's **outside** of our machine - here, we don't know or care how the machine works, we just know what we've got (input), and what we want (output).

- **Assumptions**: What do we **assume** about our **problem**? What do we expect about our **data**, or our possible **solutions**? How do we use this knowledge?
  - This step is important because these assumptions can allow us to **simplify** the problem, and often, our approach **depends** on them.
  - **Example:** We might be looking at our patients (several adorable puppies), and **assume** that they are all the same **age**: we can simplify by not including age as a varible.

We often use these assumptions to come up with solutions: if they aren't true, your approach may fail!

- **Problem Class**: What are the **needs** of our particular problem? What **kind** of inputs and outputs are expected?
  - In this situation, "class" means, "**set** of things with something in **common**". So, our "problem class" tells us, "what **kind** of problem do we have"?
  - This is important for choosing our solution: our solution follows from the problem.

"**Which group** of problems does ours **fit into?**"

In order to answer a question, you need to know what you're being asked!

    * We might also use **existing** solutions to similar **problems** as inspiration for our own work.

  - **Example:** Our inputs are weight, blood pressure, and breed. Our output is a number: how long do they have to live? This will be a real number, in years.

- **Evaluation Criteria**: What is our goal? We know the **kind** of output we want (structure, type, etc.), but how do we measure the **quality** of an answer?

  - This evaluation criteria is crucial, both for telling our machine how to **improve**, and to **show** other humans how well it **performs**.

    Example: We could use the absolute difference between the lifetime predicted, and the lifetime the puppies actually experience.

These aspects together make up our problem, that we now need a **solution** for.

1.0.6 Solution Setup: What is a model?

Remember: our goal is to create a **machine** that **takes in** data and **spits out** some kind of results.

The **solution** is what's **inside** the machine - how do we do it? What approach do we use?

First, let's dig a little into what a **solution** is: we've mentioned before that our solution will often rely on a **model**, but what exactly *is* a model?

For our purposes, a model is a way to **simplify reality**: we strip away everything that doesn't matter, and just leave a system that can work *well enough*, in the ways that matter.

In machine learning, we sometimes care less about how **realistic** the model is, than its ability to get **good results**. That means our model is not always structured to match reality.

We do this sometimes because we don't *know* the true model, and sometimes because simulating the true model is too expensive and time-consuming.

Definition 2

A **model** is a way of mathematically **representing** a **system**.

This system is **simplified** to only include the **details** we care about and give us the level of **accuracy** we want.

We boil down a **system** into the values we **care about**, and how those values **affect** each other (in terms of math equations).

Example: A planetary model that simulates **gravity** between Mars and the sun may not account for the density of the planet, or everything that happens on the surface... but that might be good enough to predict the **length of a year** on Mars.

However, in this example, we knew all of the values of the model (the weight of the planet and sun, the distance from the sun...). We have no need for machine learning: the model is already **complete**.

Again, we emphasize that a model doesn't have to be structured to match reality - but if we know the true model, this can help.

In the problems we face, we **don't know** those values, or even always what **model** will work best. That's where the techniques we will learn come in.

1.0.7 The Solution

So now, we have a vague idea of what our solution might look like. So, let's break it into parts, like we did for the problem.

- **Model Type**: Will we make a model? What kinds of **data** will we **include** in our model?
  - Sometimes, a model isn't necessary: do we really need it? If we do, how do we **use** that model?
- **Model Class**: What **kind** of model will we use? What sort of **variables** will we use, and what **structure** will our math use?
  - Just like with problem classes, a model class is a "type" of model: a collection of models with similar structure.
  - We will spend much of this class exploring **different** model classes: each has benefits in different circumstances.
- **Algorithm**: Once we have a model, how do we "teach" it what we want it to know? We'll need a **procedure** for this - an algorithm.
  - Which algorithms we choose will affect how well our machine can learn: how quickly will it learn, and how good is the end result?

Now, we take a deeper dive into each aspect listed about, starting with our **base assumptions**.

1.1 Problem Class

"Problem class" is, from the name, the type of problem you are presented with: what inputs and outputs are expected?

But there is a second aspect of the problem we haven't discussed - what **data** is does the machine have available when it is **training**?

1.1.1 Supervised vs. Unsupervised

We know that, to train our computer, we have to give it **input** data, but how does the machine know whether it's doing well? We could, for example, give it an "**answer key**": the correct outputs we expect from it.

If we do, we call that **supervised** learning.

In a way, we're "supervising" it by giving it the right answer: we're guiding it and making sure it does what we want it to!

Or, do we find a different way to measure its success? We break it down into a few common cases:

- **Supervised Learning** is when train our machine using a set of **inputs** and the correct matching **outputs**.
  - **Example:** You show your machine a bunch of **pictures** (inputs), and then **label** what is in each picture: like a dog (output).
- **Unsupervised Learning** is when we **don't** give our machine the answers, and it has to guess without having a "correct" answer.
  - This is often used in cases where we don't know a "correct" answer in advance. For example, we might want to find some kind of **pattern** in our data, and we have no way of predicting that!
  - **Example:** You look at a bunch of animals (input) and try to invent species for groups of animals.

You don't need to know the species "cow" and "pig" to figure out that they're different from each other!

- There are other cases that we will save for the end of this section.

We'll list some common problem classes for each of these types. You don't have to memorize these types, but they will come back later in the course.

But first, one more detail.

1.1.2 How do we store our data?

Each data point typically contains **multiple** values: pieces of information you want to draw **conclusions** from.

We often standardize the information our machine receives by storing this information in a **vector**.

Being consistent makes it easier to develop the techniques we need!

Our models are usually made up of **equations**, so we want to be able to **compute** with these values. So, each variable will be represented with a real number, or multiple if necessary.

Thus, one data point is a **vector of real numbers**. Specifically, a **column vector**.

Notation 3

x is our **vector of inputs**.

It is a column vector. Its matrix shape is $(d \times 1)$.

Example: Suppose we have a data point x that's a vector of 3 numbers: its shape is (3×1). We write this as:

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \tag{1.1}$$

Since this is so common, we introduce some notation.

The real numbers are represented with $\mathbb{R}$. Since we're combining **multiple** real numbers, we use **exponent** notation to represent this.

Notation 4

The **set of length-d vectors** is written as $\mathbb{R}^d$.

Example: $\mathbb{R}^2$ represents all of the length-2 vectors: all of the vectors/points on the 2D plane.

So, we might say a data point $x \in \mathbb{R}^d$ if all we know is that x is a length-d vector.

1.1.3 Supervised Learning

- **Regression** takes in a vector of numbers, and predicts some **real number** as an output. Our goal is to correctly guess the desired output.
  - **Example:** You want to predict how much a worker makes based on their job, where they live, and how many years they've worked.
- **Classification** also takes in a vector of numbers, but outputs a **label**: we have a set of **classes**, and we want to **label** each data point as a member of one class.
  - This means our output is **discrete**: each class is separate output value, and we have k classes.
  - **Example:** You have several documents and want to **label** which **language** each is written in.

Notice that "where they live" isn't usually represented as a number: we often have to convert certain data types.

As before, a "class" is just another word for a group of related things.

1.1.4 Unsupervised Learning

- **Density Estimation** takes in data, and tries to approximate the **distribution** of that data: what is the chance of getting a new data point x?
  - **Example:** You want to get the **distribution** of human **heights** in a particular city.
- **Clustering** is when you want to sort data points into groups of similar points, without knowing the groups in advance.
  - **Example:** You want to sort patients with a disease into groups, where each group might need different treatments.
- **Dimensionality Reduction** is a bit different: the goal is to take a vector, and reduce the length of the vector, while still keeping the information that's important.
  - You may not need every dimension to store the information you need, so you can save on space and time by storing it in a smaller vector.
  - **Example:** You find out height has no effect on income, so you ignore height. Or maybe you find that having both education and literacy is redundant.
  - Notice that what information is relevant depends on what you're using it for.

We define "**distributions**" in section 1.2.

Since we often automate this process, real examples might not be so simple!

1.1.5 Other Types of Learning

Now, we turn to some types of learning that are, arguably, neither supervised nor supervised.

- **Reinforcement Learning** is used when you have an "environment" you can interact with. Different choices will change what that environment looks like, and may reward or punish you.
  - The goal is to pick the actions that give you the best rewards.
  - **Example:** You have a robot on Mars, and you want to move your robot (action) to reach certain goals (rewards)
  - This isn't **supervised** because you **don't know** the correct action. But it isn't fully unsupervised because you **do know** when you get a reward.
- **Sequence Learning** is used to take one sequence, and turn it into another. In these sequences, each output depends on all of the previous inputs.
  - This means you need to store information about previous inputs using a **state**.
  - **Example:** Predicting the next word in a sentence, based on the words so far. You **predict** one word for each new one you receive, so you return a **sequence**.

We represent the current environment with something called a **state**.

  - We're partly "supervised" by being given the output sequence, but we don't know what our states need to look like.

1.1.6 Types of Learning not covered in this class (Optional)

These will not be covered, but are worth mentioning.

- **Semi-supervised Learning** gives us some supervised training data that has been labelled, but also some that has not.
- **Active Learning** gives our computer the ability to **choose** which data points it receives: this is used when data is **expensive**, and we want to learn efficiently.
- **Transfer Learning** is used when we apply learning from one task to another, related task. That way, the new task can be learned faster.

1.2 Assumptions

Let's look at our underlying assumptions: the rest of this class relies on these assumptions.

1.2.1 An assumption about data

Let's return back to our original goal: we want to use **data** to teach our machine to give us **results** we want. Just like how a person might learn from their **experience** and use it to make **judgments**.

However, there's an **assumption** built in to this statement, one we need to look at more closely: we are assuming that **past** data allows us to predict **future** data.

This may seem obvious, but it isn't always: past data may not be **representative** of the future, for example.

- **Example:** We can't use the weather over the month of July to predict the weather in the month of December.

This often called the problem of **induction**: using the past to predict the future.

1.2.2 Is our data representative?

First, let's solve the problem presented above:

- **Example:** We got our weather from a **different** month than we're trying to predict.

So, it seems our problem is that our **data** and what we're trying to **predict** are from **two different sources**.

We want them to come from the **same source**, then. In this case, we could say we want them to be from the **same** month. Great. But how do we say this in general?

1.2.3 How do we compare data?

We got down to the real problem: we want our new data to be from a similar source to the old data. One month couldn't **represent** another, because they **behave** differently.

- **Example:** For different months, we get different rainy days, different temperature ranges, so on: they can't be compared.

In general, we need a way to describe what we mean by "different": what describes one of these months?

- **Example:** To us, all that matters is the weather: how **likely** are we have a rainy day, for example? In fact, we'd like to know how **likely** every outcome is.

We represent this with something called a **distribution**. A distribution gives us exactly what we just described: **how likely** different events are to occur.

This is how our system "behaves", in a way.

Definition 5

A **distribution** is a **function** that gives us the **probability** of different **outcomes**.

Why is it called a distribution? Well, we're taking the **odds**, and spreading them out (or **distributing** them) over multiple different outcomes!

Example: The **distribution** of outcomes on a coin is 50% chance of heads, 50% chance of tails.

Notice that distributions are **probabilistic**: outcomes have a certain **chance** of occurring. Otherwise, these problems would be simple.

1.2.4 Identically Distributed Data

We can think of this distribution as a **simplified** view of the **source** of our data. Each "outcome" is a data point; one we can use to **learn**.

We want our **past** data we **learn** from, and our **future** data with **test** with, to have the **same** distribution.

We also want different points in the **same** dataset (past *or* future) to be from the same **distribution**: if they aren't, then why are we lumping them together?

We want to focus on one problem at a time - one distribution.

We want them to be the "same", or **identical**: they have **exactly** the same chances for each outcome.

In other words: we want our sets of data to be **identically distributed**.

Definition 6

If two **data points**(or datasets) are **identically distributed**, then they have the **same** underlying **distributions**.

In other words, they have the **same probabilities** for each possible **outcome**.

Example: Two fair coins will behave the same as each other: they both have 50-50 odds. Thus, they're **identically distributed**.

1.2.5 Independence (Review)

There's a second assumption that is just as important: when we draw two different data points, we are also **assuming** that the results of one do not **affect** the other.

If one point **depended** on another, then there's no **new** information: you could have used the last point to guess this one.

This means you're **not learning**, which is a problem: you need many experiences to come to a good **conclusion**, that will apply well in the future.

Because we don't want the result of one data point to **depend** on another, we call this assumption **independence**.

Definition 7

Two **data points** are **independent** if **knowledge** of the outcome for one data point does not affect the **probabilities** for the other.

This definition is a bit informal: the proper definition is to say that, for two events A and B, $P(A)P(B) = P(A \text{ and } B)$

Example: If you flip two coins, knowing that one coins comes up heads does not tell you anything about the other coin: the two coin tosses are **independent**.

1.2.6 Independent and Identically Distributed

We combine both of these assumptions into our final result: we want our data points and data sets to be both **independent and identically distributed.**

Definition 8

IID, or **Independent and Identically Distributed**, means that if you draw two data points, they

- Come from the **same distribution**: they have the same **probabilities** for each outcome,
- They **aren't related** in any other way: they are **independent**, meaning the **outcome** of one **does not** affect the other.

Example: Based on the two examples above, flipping two coins (or rolling a die twice) is IID.

We shorten this to one acronym, which tells you how important it is: it is the base assumption in many different statistics, inference, and machine learning settings.

We will assume this to be true, and use that assumption throughout the class. We expect our data to be IID in most cases.

1.2.7 Estimation and Generalization

In this section, the main theme has been applying knowledge about **training** data to **new**, unfamiliar situations, like our **testing** data.

We have a word for this that we haven't used so far: **generalization**.

Definition 9

Generalization is the **problem** of applying **current** knowledge to **new** situations we've never seen before.

We want to be able to take the **specific** case of our training data, and apply it to the more **general** case of any of the possible **new** data.

A second problem is the **nature** of our training data: because we **randomly** select it, we don't have a perfect idea of what the true distribution looks like.

The randomness means that our sample will look a bit different each time we generate it. This creates some **noise**: something that interferes with what we're trying to focus on.

Just like how background **noise** can make it harder to listen to a phone call!

The problem of using our sample to **estimate** the true distribution, despite imperfect, "noisy" data, is **estimation.**

Definition 10

Estimation is the **problem** of taking **imperfect** data and using it to **estimate** the "true" information we're looking for.

1.2.8 Other Assumptions

There are some other assumptions we will make, that will will not go into as much detail on:

- We know the set of possible answers: the type of answer we should give back, whether number, label, making a choice...
  - If we don't know what kind of answer we're supposed to give, how can we build a model to give back that answer?

Imagine if you were supposed to write an essay, but could only answer with real numbers between 0 and 1 - this is what we want to avoid.

- Our problem is solvable: the "true" model can be represented and answered using our computer.

Here are some more which are less universal.

- The data is generated by a Markov chain.

If you don't know what this is, don't worry! We come back to it later.

- The data might be **adversarial**: designed to specifically exploit weaknesses in the machine.

Some of these assumptions are required in order to move forward at all. Others narrow down the options we have to work with, so we can find a good solution in a reasonable amount of time.

1.3 Evaluation Criteria

1.3.1 What is a loss function?

In order to solve our task, we want to be able to measure how our machine is performing. We do this by creating a measure of success or failure, called a **loss function**.

Definition 11

A **loss function** measures how **poorly** your machine is **performing** on a **task**.

The output is a **real number**. If your machine is performing **well**, then you will have a **low** output. And vice versa: if it is doing **badly**, it will have a **high** output.

Example: If you counted the number of questions you got **wrong** on a test, that could be a **loss function**.

A loss function usually has the **correct** and **predicted** guesses as inputs: it has to compare them to know how well it's doing.

Notation 12

Often, we will use g to represent our **guess** as to the correct answer: this is the output of our model; our **prediction**.

The **true answer** is often represented by either a or y.

Our **loss** is the function $\mathcal{L}$, so altogether, our computed loss is $\mathcal{L}(g, a)$.

1.3.2 Examples of Loss Functions

Different loss functions are useful for different situations.

- **0-1 Loss** is a simple kind of loss: if our answer is correct, the value is 0. If our answer is incorrect, the value is 1.

  This matches our earlier example of "number of questions wrong on a test".

$$\mathcal{L}(g, a) = \begin{cases} 0 & \text{if } g = a \\ 1 & \text{otherwise} \end{cases}$$

  - This kind of loss is often used for **discrete** situations, where there are k options and one is correct - like on a multiple-choice test.

- **Linear loss** is the **absolute difference** between your answer and the correct one.

$$\mathcal{L}(g, a) = |g - a| \tag{1.2}$$

- **Square loss** is the **square difference**.

$$\mathcal{L}(g, a) = (g - a)^2 \tag{1.3}$$

  - Because the slope increases as you get further away from 0, it punishes large errors more aggressively than small errors.

- **Asymmetric Loss** punishes some outcomes more than others. It may be worse to miss a heart attack, than to expect one and be wrong.

$$\mathcal{L}(g, a) = \begin{cases} 1 & \text{if } g = 1 \text{ and } a = 0 \\ 10 & \text{if } g = 0 \text{ and } a = 1 \\ 0 & \text{otherwise} \end{cases}$$

1.3.3 How to use loss

We want to reduce loss as much as we can: in other words, **minimize** it. But there are lots of ways to do that.

We could the "worst-case" loss, or the average loss, etc...

In this class we will minimize the **expected loss**: the average loss we would *expect* based on the probability of each outcome.

We do this because, over the **long-term**, the **expected loss** should reflect what we actually get.

This is called the **law of large numbers**: if you have a large number of trials, the average value should be close to the expected value.

Concept 13

In most machine learning problems, we want to **minimize** our **expected loss**.

But we also need to be careful when choosing our loss function: if we get to choose how we're grading ourselves, then we need to pick an accurate way to measure progress!

1.4 Model Type

Now, we start on the solution. Do we choose to use a model? If we do, there are some other details we have to consider.

1.4.1 No Model

A model allows us to **simplify** what we learn from our data. So, if we don't use a model, we have to use our data **directly**.

One way to do this is to simply average some known data points that "seem" similar to the newest query. This is called the **nearest neighbor** approach.

When we say "similar", there are multiple ways to interpret this, but often we use distance in $\mathbb{R}^d$ space.

Example: You measure a chemical's physical properties, and **label** it based on which one you've seen before is the most **similar**.

1.4.2 Models using Parameters

These days, we're much more likely to **use** a model to make our prediction.

But, as we mentioned before, a model can be adjusted, and for almost any problem, we'll need to adjust it to fit our needs. This is the process of **training**.

How do we adjust a model? Our models will be a **function**, that has several values it uses to do calculations on our inputs. For example, here's a simple model:

$$f(x) = A\sin(Bx) + C \tag{1.4}$$

In this case, we have one input variable, x. And we have three values that don't change based on the input: A, B, and C. These values are called **parameters**.

Definition 14

Parameters are the **non-input constants** in a model that can be **adjusted**.

Example: When using the linear equation $f(x) = mx + b$, m and b are your parameters.

You can think of a parameter as a dial on a machine that you can "tune" to different values, like a radio.

Adjusting these parameters will change how the model behaves - different outputs for each input - but it keeps the same overall **structure**.

By structure, we mean the formula: the way variables and parameters **interact**. The above model will (almost) always be **different** from

"Almost" because, if $A = B = 0$ for both cases, they're both the constant function $f(x) = C$.

$$f(x) = Ax^2 + Bx + C \tag{1.5}$$

They both have three parameters, and one input, but they are different models.

1.4.3 Prediction Rule

Our goal is to use one of these equations to **directly** calculate our prediction. For that reason, we call this equation our **prediction rule**, but more often, we will call it our **hypothesis**.

Definition 15

A **hypothesis** is the **function** that defines our model, using a fixed number of **parameters**.

The **output** of our hypothesis is typically the **prediction** our model is designed to create.

1.4.4 Hypothesis Notation

For simplicity's sake, we often lump all of our **parameters** into a single **vector**, θ. Just like for x, we'll use a **column vector**.

Notation 16

θ is our **vector of parameters**.

It is a column vector. Its matrix shape is $d \times 1$.

Often, a vector is represented by bolding a variable ($\mathbf{x}$), or putting an arrow over it ($\vec{x}$). Since we work with vectors so often in this class, we will omit this notation.

Example: Here is a vector θ with 4 parameters.

$$\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \end{bmatrix} \tag{1.6}$$

Similarly, we lump all of our inputs into a single **vector**, x.

But, if we have **multiple** data points, we need to label them **separately**.

Notation 17

$x^{(i)}$ is the i^{th} **data point**, represented as a vector.

Sometimes, you may instead see the notation x_i.

x is the **input** to our hypothesis h, but since θ (our parameters) can be **adjusted**, we can think of it as a **second** "type" of input.

To represent this, we use $f(a; b)$ notation: a is our input to a single **function**, but b allows

us to describe a whole **family** of functions (by adjusting parameters).

Notation 18
Our **hypothesis** is shown in the form $h(x;\theta)$.

1.4.5 Fitting

The process of **adjusting** our model (i.e. its **parameters**) to match our data is called **fitting**.

As we mentioned before, our goal is typically to **minimize** expected loss. But this expected loss is based on knowing the **true** distribution of our data. We call this loss our **test error**.

We call it this because we're "testing" our machine in the real world.

Since we usually don't know the true distribution, we have to settle for our best guess - the **training data** that we've gathered.

Instead, we could minimize the **training** error: we average it out, to see our performance. Let's write that out.

The loss for our i^{th} data point is $\mathcal{L}(g^{(i)}, a^{(i)})$. So, we average out n of those points:

$$\frac{1}{n}\sum_{i=1}^{n}\mathcal{L}(g^{(i)}, a^{(i)}) \tag{1.7}$$

Let's write this in terms of x and y. a is just another name for y.

Our guess is given by the hypothesis, so $g^{(i)} = h(x^{(i)};\theta)$.

In this equation, we'll leave off θ, to allow for non-parametric hypotheses.

Key Equation 19
The **expected loss** for a hypothesis is:

$$\frac{1}{n}\sum_{i=1}^{n}\mathcal{L}(h(x^{(i)}), y^{(i)})$$

This is the equation we would **minimize** with θ.

1.4.6 Overfitting

But, we have to be careful - we mentioned before that the randomness of our sampling can introduce **noise**.

If we too heavily emphasize the current values, we may not **generalize** well to new data. This problem is called **overfitting**, and we will talk about it a lot in this course.

Definition 20

Overfitting happens when we fit **too strongly** to a particular dataset.

Because we focus too much on that **dataset**, our machine **learns** incorrect facts about the overall distribution.

This makes our model worse at **generalizing** to new situations.

Example: You want to know what cats are like. By coincidence, you see three black cats in a row. You assume all cats are probably black: you've **overfit** to your data.

In this course, we will discuss many ways to tackle **overfitting**.

1.5 Model Class

In this section, we'll assume we're using a model.

1.5.1 Hypothesis Class

As we mentioned before, changing our **parameters** will change the specific model we have, but it will have the same overall **structure**.

Models with the same overall equation are put in the same **model class**. Since our models are defined by their **hypothesis**, we will more often talk about **hypothesis class**.

Definition 21

A **hypothesis class** is a collection of **hypotheses** with the **same type of equation**: the only difference between them is the **value** of their **parameters**.

Another way to view it is that the **hypothesis class** represents all of the **possibilities** for a model class: we can get every option based on our **parameters**.

Another way to say "same type of equation" is "**same functional form**".

Example: Every hypothesis of the form $mx + b$ is in the same **hypothesis class**.

1.5.2 Expressiveness

Note that some hypothesis classes are capable of things that others are **not**. For example, our linear function $mx + b$ could never produce a **parabola** x^2.

That means if our problem **requires** a more complicated model, then we can't ever get a good result!

This can be summarized by **expressiveness** or "richness" of a hypothesis class.

Definition 22

If one **hypothesis class** is more **expressive** than another, it can represent a **larger** collection of possible hypotheses.

Sometimes, if a problem can't be solved in one model class, it might be solvable in a more **expressive** one.

Example: **Quadratic** equations ($Ax^2 + Bx + C$) are more expressive than **linear** equations ($mx + b$). Every linear equation can be **created** using quadratics, but not the other way around.

1.5.3 Choosing Model Classes

So, the question is - which model class should you use for a given problem?

Your first instinct might be to use the most **expressive** one you can. However, this can become very **expensive** to compute, because there are many more options you have to explore.

It's also more likely to overfit! We'll discuss why another time.

Often, it is already **known** what kinds of models work well for what kinds of problems - we'll explore some of those options in this class.

As an ML researcher gains more **experience**, they can use that experience to make **educated** guesses: they may look at multiple possible models, and pick one based on theory or practice.

Research on this is ongoing: we continue to develop new model classes to try to better handle new and old problems!

Choosing the **class** of model we want is called the **model selection** problem. Choosing the **parameters** for our model, on the other hand, is **model fitting**.

1.5.4 Our Linear Model

We will start this class off using one of the simplest models we know: one that only uses **addition** and **scalar multiplication**.

We have our input variables, $x_1, x_2, x_3 ...$ that we can combine using these two operations. We can add them together, add a constant, or multiply by a constant.

We can write this in general as:

$$h(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + ... + \theta_d x_d \tag{1.8}$$

Where θ_i are our parameters.

1.5.5 Linear Model: Vector Form

θ and x, both being vectors, are being multiplied in a way that looks similar to the **dot product**: multiplying together elements, and then adding.

$$h(x) = \theta_0 + \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_d \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} \tag{1.9}$$

So, we can rewrite it more compactly this way:

$$h(x) = \theta_0 + \theta \cdot x \tag{1.10}$$

Note that this looks very similar to the $y = mx + b$ formula, our original linear function!

Note that, in order for the dot product to work, x and θ must have the same **shape**.

Concept 23

When using a linear model, x **and** θ must have the **same shape**. They both have length d.

Meaning, they are both $d \times 1$ column vectors.

1.5.6 Linear Model: Cleaning Up

Unfortunately, we had to leave θ_0 out to make it work: if we want to talk about **all** parameters, we'll instead use the symbol Θ.

Notation 24

We represent the **parameters** of our **linear** equation as $\Theta = (\theta, \theta_0)$

We'll swap out the dot product for matrix multiplication, because this allows us to use matrices instead of just vectors (which we'll need later!)

$$h(x) = \theta_0 + \begin{bmatrix} \theta_1 & \theta_2 & \cdots & \theta_d \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} \tag{1.11}$$

In order to make the matrix multiplication work, we have to take the **tranpose** θ^T.

Finally, we condense our vectors into symbols.

Key Equation 25

The **linear model** has a hypothesis of the form

$$h(x) = \theta^T x + \theta_0 \tag{1.12}$$

This is the form you will use through a significant portion of this course - it's good to get used to it!

1.5.7 Other Models

We will explore several different kinds of models in this course.

In general, we will assume that we have a fixed, finite number of parameters. Models that don't have this restriction are called **non-parametric** models. We will use them in only one chapter.

Instead, we will focus on our **linear** model, and functions we can apply to allow it to handle $\mathrm{non-linear}$ problems.

We will combine many of these "non-linear units" to create **neural networks** later on - arguably the most **powerful** tool in the ML arsenal, and key to machine learning's modern explosion in usage.

Many of the modern models used in complex and high-performing system are **variations** on neural networks, so we will give them all the attention they need.

1.6 Algorithm

Finally, once we have our model class and a tool for evaluating our model, we can finally begin the process of **fitting** our model.

This is where the problem of developing an **algorithm** comes in - we need to decide on what set of instructions will best help us find a good model.

Our problem will typically boil down to a kind of optimization: minimizing a loss function, or more often, a modified loss function called an **objective function**.

Different problems will require different algorithms and techniques: some are general-purpose optimizers, others are specially tailors for the needs of machine learning.

One of our most powerful tools will be **gradient descent**; so much so that it has its own devoted chapter.

But, we will leave that to the next chapters.

1.7 Overview of the Course

Here is a short summary of each chapter.

- **Introduction**: an introduction to the basic concepts of the course, and what to expect going forward.
- **Regression**: using our linear model to learn to make numeric predictions about future data.
- **Gradient Descent**: learning to use the gradient, our "multivariable derivative", to optimize functions, like loss.
- **Classification**: using our model to sort data into different classes, and introducing some non-linear functions into that model.
- **Feature Representation**: transforming the data we receive, both to make them usable by a computer, and expanding our hypotheses to non-linear functions.
- **Neural Networks**: showing how you can combine multiple non-linear functions, to create a much more powerful function for new, exciting problems.
- **Convolutional Neural Networks**: building on neural networks with convolution, making it easier to handle images, signals, and other problems.
- **Sequential Models**: introducing "states", a way to store information over time, and how to do decision-making using that information.
- **Recurrent Neural Networks**: We combine neural networks with states to build up a sequence of outputs over time, allowing us to do some language processing.

- **Reinforcement Learning**: making decisions in a changing environment, where some states and choices reward you more than other.
- **Non-parametric methods**: introducing some different tools, which are often cheaper to develop and sometimes just as effective as more complex methods.
- **Clustering**: trying to find hidden patterns and structures in data, and making that data easier to visualize for human usage.

1.8 Terms

- Machine Learning
- Problem Class
- Model
- Model Class
- Distribution
- Identically Distributed
- Independence
- IID
- Induction
- Generalization
- Estimation
- Supervised Learning
- Unsupervised Learning
- Regression
- Classification
- Loss Function
- Expected Loss
- Parameter
- Hypothesis
- Fitting
- Overfitting

- Hypothesis Class
- Expressiveness
- Linear Model