FRAMING

**Supervised Machine learning**

ML systems learn how to combine input to produce useful predictions on never-before-seen data

* **Label** is the variable we're predicting
  + Typically represented by the variable **y**
  + The label could be the future price of wheat, the kind of animal shown in a picture, the meaning of an audio clip, or just about anything.
  + Eg: Cat, Dog
* **Features** are input variables describing our data
  + Typically represented by the variables **{x1, x2, ..., xn}**
  + In the spam detector example, the features could include the following:
    - words in the email text
    - sender's address
    - time of day the email was sent
    - email contains the phrase "one weird trick."
  + Eg: nose, mouth, number of legs, color, presence/absence of a tail etc
* **Example** is a particular instance of data, **x**
  + We put **x** in boldface to indicate that it is a vector.
* **Labeled example** has {features, label}: (**x**, **y**)
  + Used to train the model
  + In our spam detector example, the labeled examples would be individual emails that users have explicitly marked as "spam" or "not spam."
* **Unlabeled example** has {features, ?}: (**x**, **?**)
  + Used for making predictions on new data
  + Once we've trained our model with labeled examples, we use that model to predict the label on unlabeled examples. In the spam detector, unlabeled examples are new emails that humans haven't yet labeled
* **Model** maps examples to predicted labels: **y'**
  + Defined by internal parameters, which are learned
  + A model defines the relationship between features and label. For example, a spam detection model might associate certain features strongly with "spam". Let's highlight two phases of a model's life:
  + Training means creating or learning the model. That is, you show the model labeled examples and enable the model to gradually learn the relationships between features and label.
  + Inference means applying the trained model to unlabeled examples. That is, you use the trained model to make useful predictions (y'). For example, during inference, you can predict medianHouseValue for new unlabeled examples.

|  |  |  |
| --- | --- | --- |
| X- input | Y-output | Application |
| Email | Spam(0/1)? | Spam filtering |
| Audio/Voice | Text transcripts | Speech to text translation |
| English | Spanish | Machine Translation |
| Ad, user-info | Click(0/1)? | Online advertisements |
| Image, radar info | Position of other cars | Self-driving cars |
| Image of phone | Defect(0/1)? | Visual inspection |

**Regression vs. classification**

A **regression** model predicts continuous values. For example, regression models make predictions that answer questions like the following:

* What is the value of a house in California?
* What is the probability that a user will click on this ad?

A **classification** model predicts discrete values. For example, classification models make predictions that answer questions like the following:

* Is a given email message spam or not spam?
* Is this an image of a dog, a cat, or a hamster?

A graph on a white board

Description automatically generated Y=WX+b

**L2 Loss** for a given example is also called squared error

= Square of the difference between prediction and label

= (observation - prediction)2

= (y - y')2

A graph with a line and a dotted line

Description automatically generated with medium confidence

It has long been known that crickets (an insect species) chirp more frequently on hotter days than on cooler days. For decades, professional and amateur scientists have cataloged data on chirps-per-minute and temperature. As a birthday gift, your Aunt Ruth gives you her cricket database and asks you to learn a model to predict this relationship. Using this data, you want to explore this relationship.

First, examine your data by plotting it:

A graph with red dots

Description automatically generatedA graph with red dots and blue lines

Description automatically generatedAs expected, the plot shows the temperature rising with the number of chirps. Is this relationship between chirps and temperature linear? Yes, you could draw a single straight line like the following to approximate this relationship:

True, the line doesn't pass through every dot, but the line does clearly show the relationship between chirps and temperature. Using the equation for a line, you could write down this relationship as follows:

where:

* is the temperature in Celsius—the value we're trying to predict.
* is the slope of the line.
* is the number of chirps per minute—the value of our input feature.
* is the y-intercept.

By convention in machine learning, you'll write the equation for a model slightly differently:

Y’= b+ w1 x1

where:

* is the predicted [label](https://developers.google.com/machine-learning/crash-course/framing/ml-terminology#labels) (a desired output).
* is the bias (the y-intercept), sometimes referred to as .
* is the weight of feature 1. Weight is the same concept as the "slope"  in the traditional equation of a line.
* is a [feature](https://developers.google.com/machine-learning/crash-course/framing/ml-terminology#features) (a known input).

To **infer** (predict) the temperature  for a new chirps-per-minute value , just substitute the  value into this model.

Although this model uses only one feature, a more sophisticated model might rely on multiple features, each having a separate weight (, , etc.). For example, a model that relies on three features might look as follows:



Training and Loss

Training a model simply means learning (determining) good values for all the weights and the bias from labeled examples. In supervised learning, a machine learning algorithm builds a model by examining many examples and attempting to find a model that minimizes loss; this process is called empirical risk minimization. Loss is the penalty for a bad prediction. That is, loss is a number indicating how bad the model's prediction was on a single example. If the model's prediction is perfect, the loss is zero; otherwise, the loss is greater. The goal of training a model is to find a set of weights and biases that have low loss, on average, across all examples.

 For example, Figure 3 shows a high loss model on the left and a low loss model on the right. Note the following about the figure:

* The arrows represent loss.
* The blue lines represent predictions.

A diagram of a graph

Description automatically generated

Clearly, the line in the right plot is a much better predictive model than the line in the left plot.

**Squared loss: a popular loss function**

The linear regression models we'll examine here use a loss function called **squared loss** (also known as **L2 loss**). The squared loss for a single example is as follows:

= the square of the difference between the label and the prediction

= (observation - prediction(**x**))2

= (y - y')2

**Mean square error** (**MSE**) is the average squared loss per example over the whole dataset. To calculate MSE, sum up all the squared losses for individual examples and then divide by the number of examples:A black and white math equation

Description automatically generated

where:

* (x,y) is an example in which
  + X  is the set of features (for example, chirps/minute, age, gender) that the model uses to make predictions.
  + Y is the example's label (for example, temperature).
* Prediction(x) is a function of the weights and bias in combination with the set of features .
* D is a data set containing many labeled examples, which are (x,y) pairs.
* N is the number of examples in D.

Although MSE is commonly-used in machine learning, it is neither the only practical loss function nor the best loss function for all circumstances.

How do we reduce loss?

Hyperparameters are the configuration settings used to tune how the model is trained.

Derivative of (y - y')2 with respect to the weights and biases tells us how loss changes for a given example

* Simple to compute and convex

So we repeatedly take small steps in the direction that minimizes loss

* We call these Gradient Steps (But they're really negative Gradient Steps)
* This strategy is called Gradient Descent

A diagram of a cloud

Description automatically generated

Weights: Imagine you're trying to predict the price of a house based on its size. You might think that the size of the house is important, but other factors like the number of bedrooms and bathrooms might also matter. In a machine learning model, weights are like the importance given to each of these factors. They're the numbers that the model adjusts to make predictions.

Bias: Now, let's say even for a house with zero size, there might be a base price due to factors like the location or the neighborhood. Bias is like that base price. It's an additional parameter in the model that allows it to make predictions even when all the other factors are zero.

In simpler terms:

Weights: They decide how much importance each feature (like size, number of rooms) gets in making a prediction.

Bias: It's like a base value that the model starts with. It helps in making predictions even when all the features are zero.

In summary, weights and biases are the knobs that a machine learning model adjusts to make accurate predictions based on the input data it's given.

Weights:

Let's say you believe that the number of hours studied has a significant impact on exam scores. However, you also think that factors like natural aptitude might play a role. In your model, weights are the factors that determine how much importance each input (like hours studied) gets in predicting the output (exam score).

For instance, you might assign a weight of 0.8 to the number of hours studied, indicating that it's very important, but a weight of 0.2 to natural aptitude, suggesting it's less important.

Bias:

Now, even if a student hasn't studied at all, they might still score a few points just because of their natural talent or luck. Bias accounts for such baseline factors that contribute to the output regardless of the input.

For example, you might set a bias of 10 points, meaning even if a student hasn't studied, they might still score 10 points just because of this baseline bias.

So, in this example:

Weights: Determine the importance of each input feature (like hours studied, natural aptitude).

Bias: Accounts for the baseline contribution to the output (like a minimum score regardless of studying).

Together, weights and biases allow the model to adjust its predictions based on the importance of different factors and account for baseline influences, ultimately aiming to make accurate predictions.

Hyperparameters: These are the settings or configurations that control the training process but are not learned from the data. Examples include the learning rate, which determines the size of the steps taken during optimization.

Derivative of Loss Function: When we calculate the derivative of the loss function with respect to the weights and biases, we get information about how the loss changes concerning these parameters. This information guides us in adjusting the weights and biases to minimize the loss.

Computational Simplicity and Convexity: The simplicity and convexity of the loss function mean that it's straightforward to compute its derivatives and that there's only one minimum (no local minima). This simplifies the optimization process.

Gradient Steps (Descent): Taking steps in the direction opposite to the gradient (negative gradient) of the loss function allows us to move towards the minimum of the loss function. These steps are called gradient steps, and the process is known as gradient descent.

In simpler terms:

Hyperparameters: Settings that control how the model learns.

Derivative of Loss: Helps us understand how to tweak the model parameters to reduce loss.

Gradient Descent: A strategy where we take small steps in the direction that decreases loss, guided by the derivative of the loss function.

1. **Hyperparameters**: Imagine you're setting up a robot to find the best path through a maze. Hyperparameters are like the settings you adjust before you let the robot loose. For example, how big the robot's steps should be.
2. **Derivative of Loss Function**: Think of this as the robot's sense of touch. As the robot moves, it feels the ground to understand if it's going uphill or downhill. Similarly, the derivative tells us if we're getting closer or farther away from the best solution as we adjust the model.
3. **Computational Simplicity and Convexity**: Picture the maze as a smooth, bowl-shaped valley. It's easy to figure out which way is down and where the lowest point is. This makes it easy for the robot to know which direction to go in the maze.
4. **Gradient Steps (Descent)**: Now, every time the robot feels the ground, it takes a small step downhill. These steps are like following the slope of the valley. Since we want to minimize the loss, we move in the direction that makes the loss smaller.

So, in simple terms:

* **Hyperparameters**: Robot's settings before starting.
* **Derivative of Loss**: Robot's sense of touch to know if it's going in the right direction.
* **Gradient Descent**: Robot's strategy of taking small steps downhill to find the lowest point in the maze.