A Simple Model

This lesson will introduce predictive modeling and will focus on how to construct a simple model with loss functions.

WE'LL COVER THE FOLLOWING

In the previous chapters of this course, we have looked at data cleaning, data exploration, and statistical inference. Now it is time to move to the last stage of the data science lifecycle, which is making predictions that help us in decision making. Until now, we have discovered interesting patterns in the data that we know were significant, but how do we use these patterns to predict future events? With this objective, we make predictions using *models*.

Modeling

A **model** is a representation of a system. It tries to approximate real-world phenomena. For instance, Isaac Newton gave us a model for tries to approximate gravity. We can make predictions using the model that how far or high a ball will go if we throw it with a certain force. In the gravity model, there are certain factors that affect the outcome such as the force with which the ball is thrown, or the mass of the ball, and so on. In the same way, we can make models to predict whether a certain client will default on the credit card payment next month or not, where the client's history of payments and some other factors may affect the outcome of our model.

There are many different ways of making models and measuring their effectiveness. But first, let's start with a very simple model.

Predicting waiter tips

We have the data of customers that paid a tip at a restaurant. We will try to make a model that predicts the tip paid by the customer. Let's load the dataset and look at it first.

```
# Tips Dataset
# total_bill: Total bill (cost of the meal), including tax, in US dollars
# tip: Tip (gratuity) in US dollars
# gender: Gender of person paying for the meal (male, female)
# smoker: Smoker in party? (0=No, 1=Yes)
# day: name of day of the visit
# time: time of visit (Lunch,Dinner)
# people: number of people of the party


import pandas as pd
df = pd.read_csv('tips.csv')

print(df.head())
print(df.describe())
```

Now a very simple model can be that the customer always pays **15%** of the total bill as the tip. So mathematically:

$$predicted\ tip = 0.15 * total\_bill$$

Here 0.15 is called our model **parameter**. If we denote the model parameter with θ, total_bill value with x and the predicted tip with $\hat{y}$ then the above equation becomes

$$\hat{y} = \theta x$$

So, our simple model becomes a mathematical function, $f(x)$, that takes in an input x and gives us the output of predicted tip.

Let's first create a column to check the percent tips.

```
import pandas as pd
import matplotlib.pyplot as plt
df = pd.read_csv('tips.csv')

# Calculate percent tip
df['percent_tip'] = df['tip'] * 100.0 / df['total_bill']
```

```
df['percent_tip'] = df['percent_tip'].round()


# Plot value counts of percent tip
df['percent_tip'].value_counts().plot(kind = 'bar')
```

We calculate the percentage tips in **line 6** by multiplying the `tip` by `100.0` and then dividing it by the `total_bill`. Afterward, we round the values so that they are easy to distinguish in **line** 7. Then we plot the counts of each value of `percent_tip` in **line 10**. We first retrieve the counts of each value in `percent_tip` by using the `value_counts` function and then plot them using the `plot` function.

By looking at the plot, we find out that our model that always predicted the tip to be 15% does not predict accurately in all cases. So, we can say that our model is not performing well. At this point, we need some way of measuring how far off the predictions are from the actual values. This is where *loss functions* come in.

Loss functions

A **loss function** is a mathematical function that takes in predicted values that we predicted using our model parameter and a set of actual values (Y) and tells us how well our model performs on the entire input data (set of x values: X). Loss functions output a single value known as **loss**, which tells us how much loss or error we are getting when using a certain model parameter. Since loss functions output loss, the smaller the loss value, the better the model performance.

Therefore, we can compare which values of the model parameters perform better than others by making predictions using different model parameters and giving those predicted values to a loss function and then choosing the model parameter that gets the *minimum* value from the loss function.

Hence, we choose the model that **minimizes** a loss function.

$$L(\theta, Y)$$

But how does a loss function compute loss? There are many ways to compute

loss, and several loss functions have been designed for several problems, but

perhaps the most common and intuitive is the **Mean Squared Error** loss function.

Mean squared error

As the name suggests, this loss function computes the *mean* of *squared error*. This is a very intuitive way of computing a loss. Let's see why, with an example.

Suppose that we choose our $\theta = 0.15$ as before. The first value of `total_bill` in our *tips* dataset is 16.99.

$$\hat{y} = \theta x$$

$$\hat{y} = 0.15 * 16.99$$

$$\hat{y} = 2.54$$

If we denote the actual value of `tip` for the first row by y, then the error in the prediction is:

$$y - \hat{y} = 1.01 - 2.54$$

$$y - \hat{y} = -1.53$$

We can repeat this for all values of inputs ($x_1, x_2, x_3, x_4, ..., x_n$) and compute all the errors. Since the errors can be positive and negative, we take the squares of the errors. Now that we have all the squared errors, we take their mean to get the average error in a prediction using the current θ.

$$L(\theta, Y) = mean\{(y_1 - \hat{y_1})^2, (y_2 - \hat{y_2})^2, (y_3 - \hat{y_3})^2, ..., (y_n - \hat{y_n})^2\}$$

$$L(\theta, Y) = \frac{1}{n}((y_1 - \hat{y_1})^2 + (y_2 - \hat{y_2})^2 + (y_3 - \hat{y_3})^2 + ... + (y_n - \hat{y_n})^2)$$

$$L(\theta, Y) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})^2$$

This was the theory behind loss functions. In the next lesson, we will implement this loss function to find out the best model for our problem of predicting tips in Python.