Model Fitting on a Loss Function

This lesson will focus on implementing the mean squared error loss function in Python and applying optimization to obtain the best performing model.

WE'LL COVER THE FOLLOWING

In the last lesson, we learned how loss functions can be used in theory to find out the best model for our problem. Now, we need to implement the mean squared error function in Python to our dataset.

Coding up the loss function

First, we will write the MSE (mean squared error) function, and then use that on our dataset.

```
def mse_loss(y_pred,y_actual):
  sq_error = (y_pred - y_actual)**2
  loss = sq_error.mean()
  return loss
```

We define a function that will work on two columns of dataframe, i.e., Series objects. We have two input series `y_pred` which has predicted values and `y_actual` which has the actual values. In **line 2**, we first calculate the error and then take the square by using the `**` operator. Since these are Series objects, Python will automatically subtract the corresponding `y_actual` and `y_pred` values. In the next line, we use the function `mean` on `sq_error` and return the `loss`.

Getting predictions

Now we will get predictions using the data that we have for a single model parameter and then compute the loss for that model.

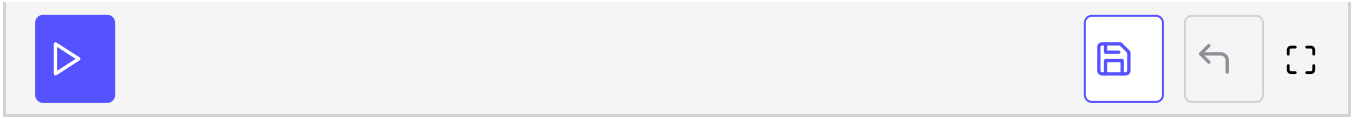
```
import pandas as pd

# define our loss fucntion
def mse_loss(y_pred,y_actual):
  sq_error = (y_pred - y_actual)**2
  loss = sq_error.mean()
  return loss

# read the file
df = pd.read_csv('tips.csv')

# compute predicted values
theta = 0.15
predicted_values = theta * df['total_bill']

#compute loss
loss = mse_loss(y_pred = predicted_values, y_actual = df['tip'])
print(loss)
```



We define our function, `mse_loss`, in **lines 4-7**. Then we read the data in **line 10**. After that we compute the predicted values in **line 14** by multiplying `theta` with `total_bill` column. This gives us predicted values for each row in the dataset. Next, we compute the loss in **line 17**. We use the function `mse_loss` defined above. We give `predicted_values` as `y_pred` and the column `tip` as `y_actual` to the function. Then, we print the loss returned from the function in the last line.

We get a loss value of approximately $1.19 which means that the average squared difference between the predicted value and the actual value is approximately $1.19.

Minimizing the loss function

Now that we know how to get loss values for a model, we need to find the model that best predicts our data by comparing losses for different models. We will set up some values for model parameter and compare the losses for all of these values.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

# define our loss fucntion
def mse_loss(y_pred,y_actual):
  sq_error = (y_pred - y_actual)**2
  loss = sq_error.mean()
  return loss

# read the file
df = pd.read_csv('tips.csv')

theta_values = [0.10,0.11,0.12,0.13,0.14,0.15,0.16,0.17,0.18,0.19,0.20]
losses = []
for theta in theta_values:
  # compute predicted values
  predicted_values = theta * df['total_bill']

  # compute loss
  loss = mse_loss(y_pred = predicted_values, y_actual = df['tip'])

  # Append the loss
  losses.append(loss)

# Find the minimum loss and corresponding theta
min_loss = min(losses)
min_loss_index = np.argmin(losses)
best_theta = theta_values[min_loss_index]

print('Model Parameters : ', theta_values)
print('Loss Values : ', losses)
print('Best Theta : ',best_theta)
print('Minimum Loss : ',min_loss)

# plot the losses against theta values
plt.plot(theta_values,losses)
plt.show()
```

Let's examine the long piece of code part by part. Our aim was to minimize the loss function by comparing losses for different values of the model parameter and then choosing the model parameter that gave the lowest loss.

We define our loss function in **lines 6-9**, as we did above. Then we store some values in a list for our model parameter, θ in **line 14**, that we will be using later on. We create an empty list and call it `losses` in the next line. This list will be used to store the losses that we get for all the model parameters.

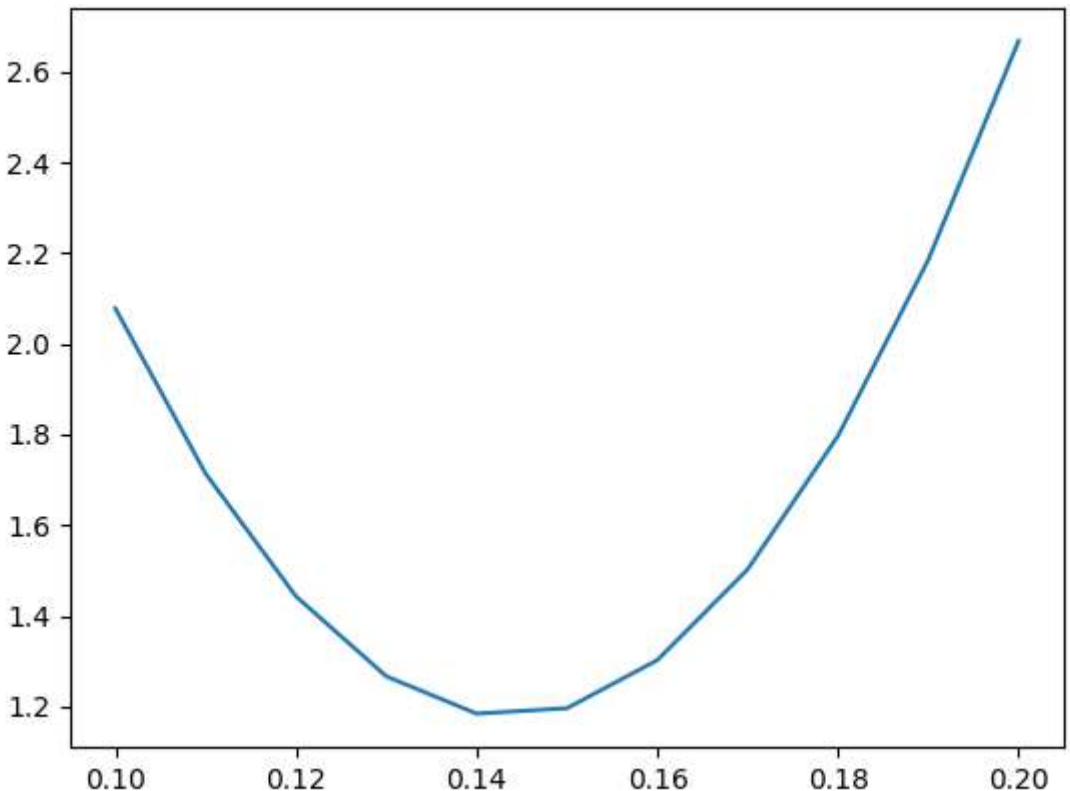
We use a `for` loop since we want to repeat the same process of computing the loss for different values of θ. In **line 16**, we start the `for` loop.

```
for theta in theta_values
```

This means that for every iteration of this `for` loop, the keyword `theta` will take a value from our list `theta_values`. Inside the loop, we compute the predicted values and the loss as we did before. In **line 24**, we store the loss that we calculated in the list of losses by using `append`.

After the loop finishes, `losses` will have the loss values for all thetas in `theta_values`. Now to find the minimum loss, we use the built-in function `min` in **line 27**. But we need to get the corresponding θ for it as well. So, we also find out the index of the minimum value from the `losses` list in **line 28** by using the function `argmin` from the `numpy` library that we imported in **line 2**. We index the list `theta_values` using the `min_loss_index` and retrieve the corresponding θ.

In **lines 31-34**, we display our findings. It is always a good idea to plot the loss values against model parameters. Since losses are not stored in a dataframe, we use the `plot` function from the `matplotlib` library. We give it `theta_values` as the first argument and `losses` as the second argument. It plots the first argument on the x-axis and the second on the y-axis.



We can confirm our results from the plot as well. We see that we encountered minimal loss when the model parameter was 0.14. It implied that the average squared error in the prediction was $\$1.18$. So, in this case, we got the best model at $\theta = 0.14$ by minimizing the mean squared loss function.

The curve that we plotted above is sometimes called the **error curve** or the **error surface**. The process of finding the best model according to a loss function is called **model fitting**. The goal of model fitting is to find the minimum of the error curve.

But, did you spot any issues with our implementation of model fitting? We will discuss a very important issue in the next lesson that will lead us to a technique called *gradient descent*.