Evaluating Regression Models

This lesson will focus on ways to evaluate the performance of regression Models.

WE'LL COVER THE FOLLOWING

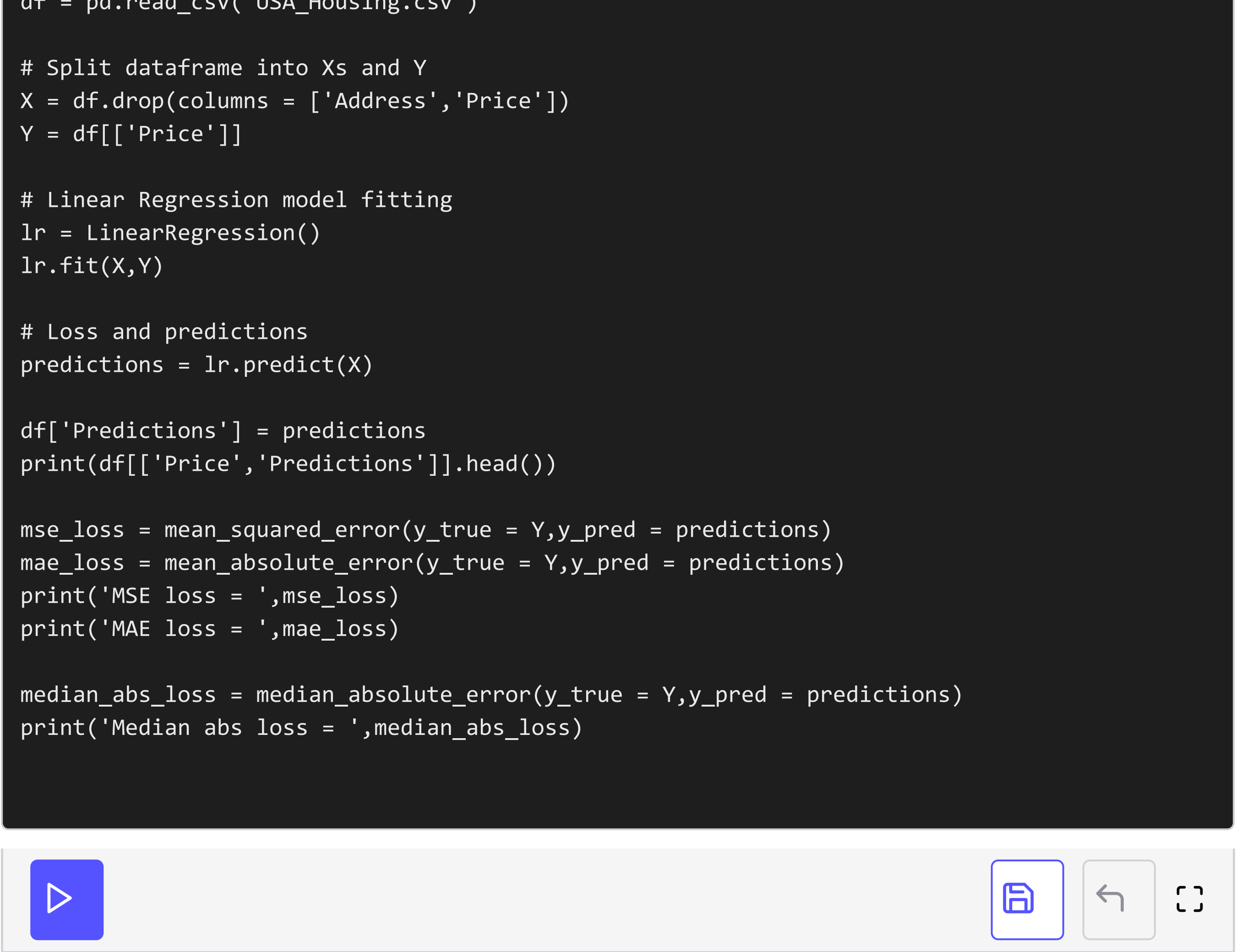
In the previous lessons, we learned how to make and fit linear regression models in Python. But we did not discuss ways to judge the performance of the models. In this lesson, we will focus on techniques used to evaluate the performance of linear regression models.

We will be using the same model that we used in the last lesson where we tried to predict house prices using the USA Housing Dataset.

Losses

We can evaluate the model performance by looking at different losses. We have already looked at mean squared loss.

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,mean_absolute_error,median_absolute_error
import numpy as np

df = pd.read_csv('USA_Housing.csv')
```



```
df = pd.read_csv('USA_Housing.csv')

# Split dataframe into Xs and Y
X = df.drop(columns = ['Address','Price'])
Y = df[['Price']]

# Linear Regression model fitting
lr = LinearRegression()
lr.fit(X,Y)

# Loss and predictions
predictions = lr.predict(X)

df['Predictions'] = predictions
print(df[['Price','Predictions']].head())

mse_loss = mean_squared_error(y_true = Y,y_pred = predictions)
mae_loss = mean_absolute_error(y_true = Y,y_pred = predictions)
print('MSE loss = ',mse_loss)
print('MAE loss = ',mae_loss)

median_abs_loss = median_absolute_error(y_true = Y,y_pred = predictions)
print('Median abs loss = ',median_abs_loss)
```

In **lines 3 and 4**, we have imported the `LinearRegression` class and `mean_squared_error` function. We read the data into a dataframe in **line 7**. Since we will not be using any non-numeric variables for prediction, we drop `Price` and `Address` and form a new dataframe `X` in **line 10**. It has all the variables that we can use in prediction. In **line 11**, we separate the actual values of `Price` in a dataframe called `Y`.

In **line 14**, we initialize the `LinearRegression` class and call the class object `lr`. We then use the `fit` function to fit our model in the next line. The `fit` function will find the best model for us and store the model parameters internally.

Now we get predictions using our fitted model in **line 18** using the `predict` function. Then we add a column `Predictions` in the dataframe in **line 20**. The next line will show us the actual values and predicted values of the top 5 rows side by side.

In **line 23**, we take the mean squared error and save it as `mse_loss` using the `mean_squared_error` function. In the next line, we take the mean absolute error using the `mean_absolute_error` function and save it as `mae_loss`. Both functions expect the same arguments, actual values (y_true) and predicted values(y_pred). We print these losses in the next two lines.

Interpreting losses

Losses are a good indication of the performance of the model. Now by looking at the losses, we can see that the model does not perform great. The more intuitive Mean Absolute Error of almost 81000 does not seem very good performance by the model on average.

However, some loss metrics, such as MSE and MAE, are greatly affected by outliers. For instance, there might be some outliers in the data that push the mean loss value up. Therefore, we also calculate the median absolute loss in **line 28**. We can see that the median absolute loss is almost 69000, which is a noticeable drop from the mean absolute error. This shows that we cannot always rely on loss functions to evaluate the performance, so we might need some other ways to look at the model's performance.

Plotting absolute error percentages

To check how well our model performed let's plot the absolute percentage error in each prediction

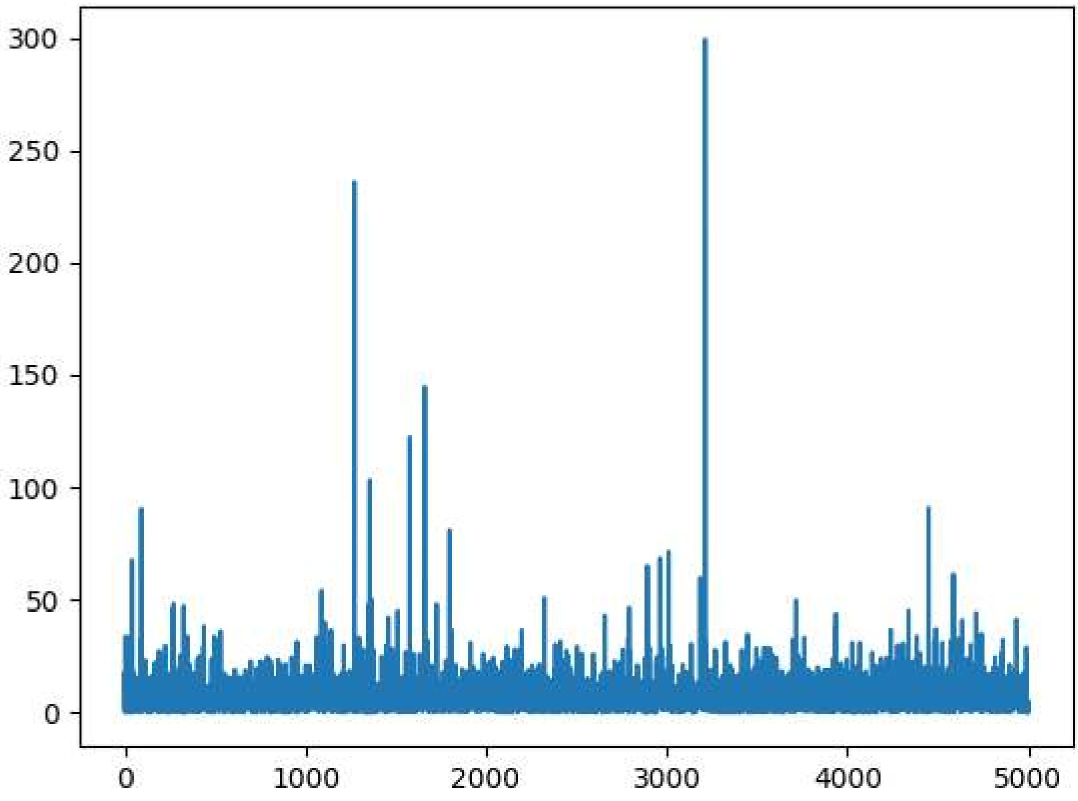
```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
import numpy as np

df = pd.read_csv('USA_Housing.csv')

# Split dataframe into Xs and Y
X = df.drop(columns = ['Address','Price'])
Y = df[['Price']]

# Linear Regression model fitting
lr = LinearRegression()
lr.fit(X,Y)

# Loss and predictions
predictions = lr.predict(X)

# Plot error %
errors = np.abs((Y-predictions) / Y) * 100
plt.plot(range(errors.shape[0]),errors)
```

After performing the regression, we compute absolute percentage errors in **line 21**. We use the following formula:

$$\%error = \frac{abs(actual - predicted)}{actual} * 100$$

We take the absolute error because an error can be both negative and positive. Then in **line 22**, we plot these errors. To plot, we use the function `plot` with `plt`. On the x-axis, we give a list of values using `range(errors.shape[0])`. The `errors.shape[0]` is the number of errors that we have. It is 5000 in this example. `range(5000)` gives us a consecutive list that starts from 0 and ends at (4999). We give the errors on the y-axis. Note that using `errors.plot(kind='bar')` would have given us the same plot as well but unlike this example, we do not always have our errors in a series or dataframe object, so we use the `plot` function from matplotlib.

Interpreting absolute error percentages



Looking at the absolute error percentages, we can see that most of the errors are within 15%. This error may be acceptable in this example.

We can also see some outliers that we were discussing in the above section. The model performs poorly on these examples. There might be some other factors that were responsible for the different prices of these houses which our model does not consider.

R^2 score

R-squared also known as **coefficient of determination** is a statistical measure that tells us the strength of the relationship between a model and the *dependent (predicted)* variable. It measures the proportion of the variation in the dependent variable that can be explained by the model. It is also known as the **goodness of fit** of a model.

Mathematically it is:

$$R^2 = 1 - \frac{Explained\ Variation}{Total\ Variation}$$

Its value ranges from 0 to 1. A value of 1 means 100% variation of the target variable can be explained by the model.

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
import numpy as np

df = pd.read_csv('USA_Housing.csv')

# Split dataframe into Xs and Y
X = df.drop(columns = ['Address','Price'])
Y = df[['Price']]

# Linear Regression model fitting
lr = LinearRegression()
lr.fit(X,Y)

# Loss and predictions
predictions = lr.predict(X)

# R2 Score
r2 = r2_score(y_true = Y, y_pred = predictions)
print('R2 = ',r2)
```

We will extend the code that we wrote in the above. After performing regression, in **line 21**, we calculate R^2 using the `r2_score` function that we imported in **line 3** from `sklearn.metrics`. It comes out to be approximately 0.91

Interpreting R^2

A value of 0.91 means that the variables in our model can account for 91% of the variation in the price of houses.

However, this does not mean that our model is 91% accurate. Higher R^2 values do not guarantee reliable and accurate models. We can see that in this example, we have a very high R^2 value but out model is not very accurate.

Contribution of each variable

Since our regression models are linear of the form:

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + ... + \theta_m x_m$$

we can see that each independent variable is multiplied by a model parameter θ. These model parameters can be thought of as weights of each variable. A variable with a higher value of weight contributes more to predicting the target variable, and a variable with a small weight does not contribute much. Thus, we can interpret the importance of independent variables to the target variable by their weights.

Takeaway

The big takeaway from our discussion in this lesson is that we cannot rely on a single metric to evaluate the performance of our model. Being better in one metric does not mean we will get good values for other metrics. However, if a model is poor in one metric such as losses, then we can disregard the model.

So far, we have learned how to make and evaluate a linear regression model for predicting numerical quantities. In the next lesson, we will look at regression for categorical variables.