Supervised Machine Learning with Simple Linear Regression in Python

In the previous segment, we explored various supervised and unsupervised machine learning algorithms that help us make sense of data. This chapter delves into the practical implementation of supervised machine learning algorithms using Python.

But before we proceed, let's ensure our understanding of supervised learning is well-grounded. Supervised machine learning involves training models on labeled data. In this paradigm, we encounter two main types of variables: dependent and independent. The dependent variable, also known as the target variable, is what we seek to predict or comprehend. Conversely, independent variables, often termed features, are the inputs utilized to predict the dependent variable.

There are two primary categories of supervised learning problems: classification and regression.

Classification: Here, the answer (dependent variable) is like picking a category. Imagine sorting emails as spam or not spam. The category (spam/not spam) is what we're predicting, and the email content itself (features) helps decide the category.

Regression: This technique tackles continuous values. For instance, predicting house prices. The price (dependent variable) is a continuous value, and the model would use features like square footage, number of bedrooms, and location (independent variables) to predict the price of a new house.

And today we'll be focusing on a fundamental and widely used algorithm called Simple Linear Regression.

In this case study we will be using Simple linear regression to predict the salary based on the

years of experience. We will use the `salaries_dataset.csv` train and test our model.

We will start by opening a new notebook in Google Colab and naming it `simple linear regression`. You can name it anything that you want. Next, we will move to the files tab on the side panel of Google Colab and upload the provided dataset.

This dataset comprises 30 observations with two columns: the first is 'Years of experience', and the second is the salary, our dependent variable.

YearsExperienc	Salary
1.1	39343
1.3	46205
1.5	37731
2	43525
2.2	39891
2.9	56642
3	60150
3.2	54445
3.2	64445
3.7	57189
3.9	63218
4	55794
4	56957
4.1	57081
4.5	61111
4.9	67938
5.1	66029
5.3	83088
5.9	81363
6	93940
6.8	91738
7.1	98273
7.9	101302
8.2	113812
8.7	109431
9	105582
9.5	116969
9.6	112635
10.3	122391
10.5	121872

Essentially, it's a collection of data from a company's employees, detailing their respective years of experience and salaries. Each row corresponds to a different employee, offering insight into their tenure and earnings.

Our objective here is to construct a simple linear regression model. By meticulously analyzing the correlation between experience and salary, this model will be equipped to predict the salaries for new employees based on their years of experience. In essence, the model will recommend an appropriate salary for a new hire based on their experience level within the company.

After successfully uploading our dataset, we can proceed with our notebook.

Moving on, incorporating essential libraries is a crucial step before diving into machine learning tasks with Python. To import the essential libraries required for our model, namely: numpy, matplotlib, pandas, and some specific functions of sklearn, we'll begin by importing the necessary libraries for our Simple Linear Regression implementation. These libraries will help us handle data, visualize it, and build our machine learning model.

Google Collab

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

After importing these libraries, click on the play button, we will see a small green tick on the right side of the code chunk. This suggests that the code ran successfully.

Now that we have the essential libraries imported, it's time to load our dataset.

Google Collab

```
dataset = pd.read_csv('Salary_Data.csv')
```

Now that we have the essential libraries imported and our dataset loaded, let's proceed to extract the features and target variable.

Before extracting, we define our features (X) and target variable (y). In this case, X represents the 'YearsExperience' column, while y corresponds to the 'Salary' column.

Google Collab

X = dataset.iloc[:, :-1].values

y = dataset.iloc[:, -1].values

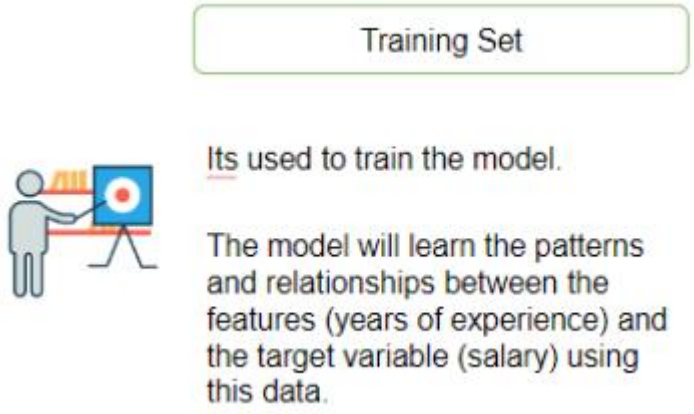
Here X represents the independent variables that our model will use to learn patterns and make predictions. In this example, it selects all columns except the last one ([:, :-1]), which corresponds to the 'YearsExperience' column, **and it's obtained by using the .iloc method to access the rows and columns of the dataset.**

And y represents the dependent variable, which is the quantity we are trying to predict. Here, it selects the last column ([:, -1]), which corresponds to the 'Salary' column.

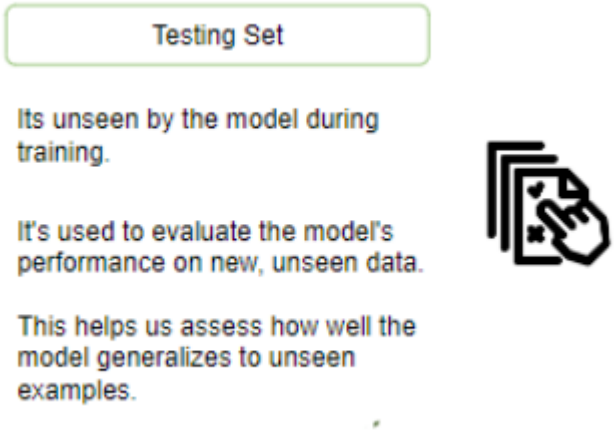
This division ensures that our model learns from the relationship between the features (independent variables) and the target variable (dependent variable), enabling it to make accurate predictions of the target variable based on the given features.

As we saw, we've separated our data into features (X) and the target variable (y). But before we can train a model to predict salaries based on years of experience, we need to split our data into two sets: a training set and a testing set.

The training set, as the name suggests, is used to train the model. The model will learn the patterns and relationships between the features (years of experience) and the target variable (salary) using this data.



The testing set, on the other hand, is unseen by the model during training. It's used to evaluate the model's performance on new, unseen data. This helps us assess how well the model generalizes to unseen examples.



Therefore splitting the dataset into training and testing sets allows us to train the model on one subset (training set) and evaluate its performance on another subset (testing set) that it hasn't seen before. This helps us gauge how well the model generalizes to new, unseen data.

We can accomplish this using the train_test_split function from the sklearn.model_selection module. This function randomly splits the dataset into training and testing sets according to a specified ratio, typically with a larger portion allocated for training.

Now we will split the dataset into test and train sets.

Here's how we can import the train_test_split function:

Google Collab

from sklearn.model_selection import train_test_split

Next we will utilize the train_test_split function from the sklearn.model_selection module to split our dataset into training and testing sets.

Google Collab

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 1)

Here X_train and y_train represent the features and target variable for the training set, respectively and X_test and y_test represent the features and target variable for the testing set, respectively.

test_size=0.2 parameter determines the proportion of the dataset that will be used for testing. Here, 0.2 means 20% of the data will be used for testing, and the remaining 80% will be used for training.

And lastly, random_state=1 ensures reproducibility. Setting a specific random_state ensures that every time you run the code, you'll get the same split of data. This can be any arbitrary number; here, it's set to 1. This is essentially like keying in an identification number to your train and test split.

The output of train_test_split is four variables:

X_train: This contains the training features.

X_test: This contains the testing features.

y_train: This contains the training target variable values. And

y_test: This contains the testing target variable values.

X_train: This contains the training features.

X_test: This contains the testing features.

y_train: This contains the training target variable values.

y_test: This contains the testing target variable values.

Now that we have split our dataset into training and testing sets, the data preprocessing phase is complete. We can move on to the next step, which is training the Simple Linear Regression model on the training set (X_train and y_train). The training set will be used to train our Simple Linear Regression model, while the test set (X_test and y_test) will be used to evaluate its performance.

All right, so the first thing we need to do is import the right class to build this simple linear regression model. The library we'll be using is Scikit-Learn, from which we'll get access to a specific module called linear_model. Within this module, we'll import the LinearRegression class. Our Simple Linear Regression model will be an instance of this LinearRegression class. In other words, we'll create an object of this class to represent our model.

Google Collab

from sklearn.linear_model import LinearRegression

This line of code imports the LinearRegression class from the linear_model module within Scikit-Learn library. By doing this, we're essentially telling Python that we want to use this specific class to build our linear regression model.

Now that we have the LinearRegression class, let's create an instance of it to represent our specific model!

Here's how we can do that:

Google Collab

regressor = LinearRegression()

In this step, we're creating an instance of the LinearRegression class, which represents our Simple Linear Regression model. We do this by assigning the LinearRegression() function call to a variable named regressor. This variable regressor now holds our model, and we can use it to access the functionalities provided by the LinearRegression class, such as training the model on our training data.

And that's only the building part. After successfully importing the necessary class and creating an instance to represent our model, the next step is to train this model on the training set (X_train and y_train). To train the model, Scikit-Learn provides a function called fit. This function essentially fits the model to the training data, allowing it to learn the patterns and relationships between the features and the target variable. Here's how we can use it:

Google Collab

regressor.fit(X_train, y_train)

In this step, we're calling the fit function on our regressor object, which is the method of LinearRegression class, passing in the training features (X_train) and the corresponding target variable values (y_train). This process enables our Simple Linear Regression model to learn from the training data and understand how the features (years of experience) relate to the target variable (salary).

Let’s run this code cell.

Now, our model has been trained on the training set, and it has learned the underlying patterns in the data. We're now ready to use this trained model to make predictions on new data or evaluate its performance on the testing set.

Now, we're going to proceed to the next step, predicting the test set result. And to do this we'll use the .predict() function to make

predictions on the test set (X_test). This will allow us to see how well our model generalizes to unseen data and assess its performance on data it wasn't explicitly trained on. We'll be comparing the predicted values with the actual salary values in the test set to evaluate the model's accuracy.

This can be done through:

Google Collab

y_pred = regressor.predict(X_test)

Here we're calling the .predict() function on our trained model (regressor) and passing the test features (X_test) as input. Remember, our model has learned the relationship between features (years of experience) and target variable (salary) during the training phase. Now, when we provide new, unseen features (from the test set) to the model through .predict(), it uses the learned relationship to predict the corresponding target variable (salary) for each data point in the test set.

The result of this prediction will be stored in the y_pred variable. This variable will now contain a list of predicted salary values for each data point in the test set.

In essence, this line of code uses our trained model to make predictions on unseen data (test set), allowing us to evaluate how well the model generalizes to new examples.

In the next step, we'll typically compare these predicted values (y_pred) with the actual salary values in the test set (y_test) to assess the model's accuracy and performance.

Now that we have both the predicted values (y_pred) and the actual salary values (y_test) for the test set, we can perform some data visualization to understand how well our model performed.

So first we’re going to visualize the Training set results and second we’re going to visualize the Test set results that help us understand how well our model is performing.

Visualizing the Training set results allows us to assess how effectively our model fits the training data, providing insights into its learning process and how well it captures the patterns in the data.

On the other hand, visualizing the Test set results helps us evaluate how well our model generalizes to new, unseen data. By comparing the model's predictions with the actual values in the test set, we can determine its performance on data it hasn't encountered during training.

Okay let’s do this:

Google Collab

```
plt.scatter(X_train, y_train, color = 'red')
plt.plot(X_train, regressor.predict(X_train), color = 'blue')
plt.title('Salary vs Experience (Training set)')
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.show()
```

plt.scatter(X_train, y_train, color='red'):This line creates a scatter plot using the training data. The scatter plot represents individual data points where each point corresponds to an employee's years of experience (X_train) on the x-axis and their corresponding salary (y_train) on the y-axis. Each data point is depicted as a red dot.

plt.plot(X_train, regressor.predict(X_train), color = 'blue'): This line plots the regression line for the training set. The regression line represents the predictions made by our trained model (regressor) based on the features (years of experience) in the training set. It's plotted in blue.

Next plt.title('Salary vs Experience (Training set)') sets the title of the plot to 'Salary vs Experience (Training set)', providing a clear indication of what the plot represents.

Then plt.xlabel('Years of Experience') labels the x-axis of the plot as 'Years of Experience', providing context for the data represented on this axis.

plt.ylabel('Salary') labels the y-axis of the plot as 'Salary', providing context for the data represented on this axis.

Lastly plt.show() displays the plot containing the scatter plot of training set data points and the regression line.

Now, we are going to do the same for the test set result, so we are going to copy the above code for visualizing the training set result and paste it into a new code cell.

[Copy the code and then make changes while speaking as mentioned below]

[Google Collab: this is the code after changes done to the copied code]

plt.scatter(X_test, y_test, color = 'red')

plt.plot(X_train, regressor.predict(X_train), color = 'blue')

plt.title('Salary vs Experience (Test set)')

plt.xlabel('Years of Experience')

plt.ylabel('Salary')

plt.show()

Now we're going to do the right replacements in this code in order to visualize the test set results.

Fisrt in the plt.scatter code line we'll replace X_train and y_train with X_test and y_test respectively to visualize the test set results.

However, in the plt.plot line (plt.plot(X_train, regressor.predict(X_train), color = 'blue')), we don't replace X_train. This is because we're still

plotting the regression line generated by our model based on the training set features (X_train).

Even though we're visualizing the test set results, we use the regression line generated from the training set because the model itself remains the same. It's the same trained model (regressor) making predictions; it's just that we're evaluating its performance on a different dataset (the test set).

Or we can say the purpose of using this line in the testing data plot is for comparison and evaluation. By overlaying the same prediction line on both the training and testing data plots, you can visually assess how well the model's predictions on the training data generalize to unseen data (testing data). It allows you to directly compare how the model performs on both datasets.

Since the same prediction line is used for both the training and testing data plots, it provides insight into how the model's predictions, learned from the training data, apply to new, unseen data (testing data). This comparison helps to evaluate the model's ability to generalize and make accurate predictions on data it hasn't seen before.

So, by keeping X_train in the plt.plot line, we ensure consistency in the representation of our model's predictions, using the same model trained on the training set to predict outcomes for the test set.

To visualize the predictions on the testing data directly instead of reusing the prediction line from the training data plot can certainly be done.

Both approaches have their merits and can provide valuable insights into the model's performance. However, using the prediction line from the training data plot can be particularly useful for comparing the model's behavior on both training and testing data in a single visualization. It offers a comprehensive view of the model's

generalization capabilities and its consistency across different datasets.

And finally, in the plt.title, we'll replace " Training set " with " Test set ", and everything else remains the same. And there you go, now we are ready to visualize the training set result and test set result.

Now we'll execute the 'Visualizing the Training set results' code cell, and after executing the code, we'll get a 2-D plot, with indeed the real salaries in these red points here, and the regression line containing the predicted salaries. And we clearly see that this regression line was calculated so that it comes as close as possible to the real salaries. And of course, for each of the years of experience here, in order to get the predicted salary, we have to project the years of experience to this blue regression line. So for example, the predicted salary corresponding to eight years of experience is about one hundred thousand dollars per year. That's how it works. And so we can clearly see that our predicted salaries are very close to the real salaries for most of them.

But that's on the training set, and that's important because our model was actually trained with these observations, i.e., with these years of experience and salaries. And now we would like to observe if we have the same results, or the same closeness of the regression line to the real salaries in the test set with which the model wasn't trained. We want to evaluate it on new observations. And that's exactly what the new graph will tell us, because now we are going to plot the real salaries of the test set and the predicted salaries of the same test set. So there we go, now let's run this 'Visualizing the test set results' code cell.

And let's see if we're still close to the real salaries even for new observations. And yes, absolutely, our predicted salaries which are on the blue line once again are very close to the indeed real salaries. So our simple linear regression model was able to do a good job at

predicting new observations. So congratulations, you built your very first successful Supervised machine learning model.