**Machine Learning for Regression**

**Introduction**

Regression is usually the first of the machine learning techniques that students learn first.

In the last chapter, we discussed machine learning and its various types. From this chapter onwards, we will start with the practical aspects of ML and specially the implementation part.

In this chapter, we are going to discuss a supervised machine learning task called Regression where the value we are trying to predict will **be of continuous nature**.

**CRISP - DM PROCESS**

We will follow the CRISP-DM process of machine learning in this chapter to do a project to predict cancer death rate. **We discussed this process in the last chapter**. If you don’t remember, here are the six steps to do machine learning using this:

1. **Business Understanding**
2. **Data Understanding**
3. **Data Preparation and Exploratory data analysis**
4. **Modeling**
5. **Evaluation and Validation**
6. **Deployment**

Let us get started.

**Objective of this chapter**

Before we proceed, open [**this**](https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD)link. This link will open an **online hosted Jupyter Lab environment** and that can be used to run the code discussed in this lab and also try the exercises. You might have to wait a few minutes for it to fully load. Here is the full link of the same:

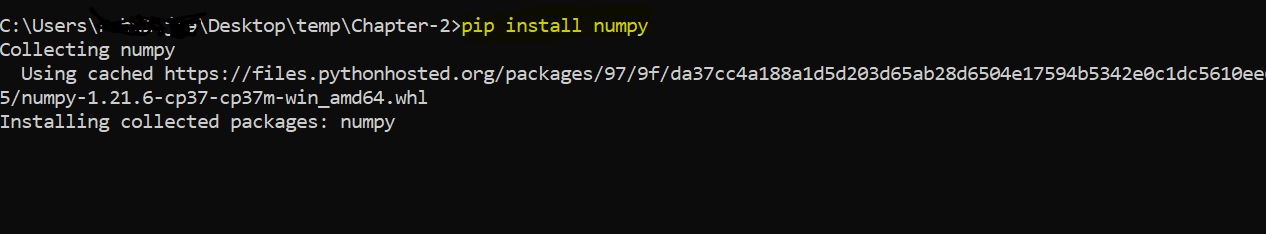
Link : **https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD**

We will do a guided project to predict the cancer death rate and this data comes from an American community survey hosted on data.world with the name Ordinary Regression Challenge. This data can be viewed [**here**](https://github.com/fenago/MLWorkshop/blob/main/Chapter-2/cancer_reg.csv). You don’t need to download this data as everything is available on the online hosted lab. The link of data is given below:

**https://github.com/fenago/MLWorkshop/blob/main/Chapter-2/cancer\_reg.csv**

We will be using python programming language to do this guided project and the libraries you would need are “Numpy”, “Skearn” and “Pandas”. **You don’t need to install anything as we will be providing you with a hosted notebook on binder which you can run to execute code as we mode further into this chapter**.

But if you are working on your local system, then make sure that you have numpy installed on your system. If it is not already installed, open a command prompt and enter the following command to install numpy:



**Machine Learning Process**

As discussed, We will follow the **CRISP-DM** steps to do the project on **cancer\_rate** prediction.

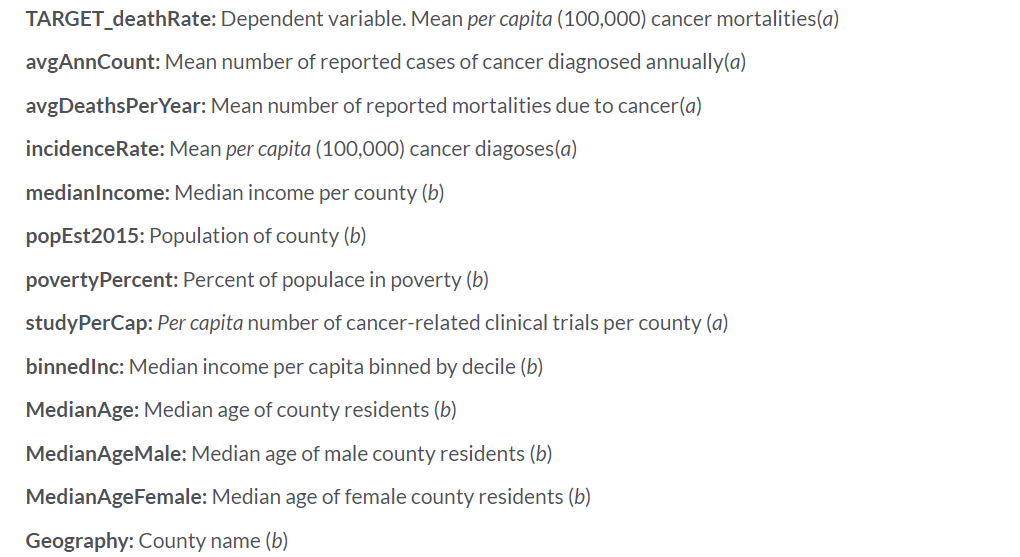
**Step.1 Business Understanding**

The objective of the guided project is to predict the cancer rate which is going to be an important tool for future research in health, specially calculating the health burden on the country. It will also help in allocation of appropriate funds for the health care and policy efforts.

Further, important features (of the final model) can be used to mitigate cancer rates by making a positive intervention to reduce the effect of that feature.

**Step.2 Data Understanding**

The data for this problem comes from data.world where the data were aggregated from a number of sources including the American Community Survey (**census.gov**), **clinicaltrials.gov**, and **cancer.gov**. Few rows of the data can be viewed below:



Since the data has many columns, it would be difficult to list all the columns on this page but you are encouraged to look at the data world documentation to view more. Here, the variable **TARGET\_deathRate** is the name of the column which we want to predict. This is also known as a dependent variable in machine learning.

Now, we will study remaining steps as a separate topics.

**Step.3 Data Preparation and Exploratory data analysis**

This is the step from where things get interesting:

**Objective of Exploration**

Objective of the exploration is to become acquainted with your data and try to look at it from each and every angle as possible. The key to best EDA is to :

1. Ask as many questions as possible about your data.
2. Try to answer those using your exploration
3. You need to be good in pandas to do these two steps effectively

**Let us look at some of the methods of exploration of this data.**

Although this is not an exhaustive list but some of the initial steps of any exploration is to get a quick feel of the data we have and it might involve answering the following questions:

1. *Number of rows and columns of the data*
2. *Number of categorical and continuous variables*
3. *Separating the target variable*
4. *Correlation plots of target vs numeric variable*
5. *Categorical data vs target (Looking at the group by average)*

So, by the end of this exploration, we might have a list of initial sets of numeric variables which can be then used for modeling.

**Reading the data**

**import numpy as np**

**import pandas as pd**

**cancer\_data = pd.read\_csv(r"https://raw.githubusercontent.com/fenago/MLWorkshop/main/Chapter-2/cancer\_reg.csv",encoding = "latin-1")**

Here, we are first importing the cancer\_reg.csv file, which is available in the online hosted jupyter file which you can also run on your own. Immediately after importing the data, we can:

1. View the shape of the data which will give us the idea about number of rows and columns
2. Also, we can look at the first few rows of the data to get a quick glance of the data.

**Missing Value analysis:**

Now that we have some idea about the dimension of our data, let us view how many missing values each of our data have. This is important because the modeling step cannot be executed without treating the missing values.

Missing values can be treated either by just dropping them or imputing those with some technique. Dropping does not always make sense if the number of missing values are quite large.

We can get look at missing values in any pandas dataframe using the following code:

**pd.isnull(cancer\_data).sum()**

**Missing value treatment**

Since the number of missing values are few, we can safely remove these columns. Let us do that on our cancer data.

**true\_false\_mask = pd.isnull(cancer\_data).sum().values == 0**

**new\_cancer\_data = cancer\_data[cancer\_data.columns[true\_false\_mask]]**

In the above code, first we are creating a true false mask which checks for whether the number of missing are zero and returns true for those columns where it is done.

Then, in the second line of code we are just selecting those columns wherever it is true.

Now, we will work with the new cancer data.

**Numeric columns and Categorical Columns**

The columns of dataframe can be printed using the .columns attribute of the pandas dataframe data.

There is a function in the pandas called “select\_dtypes” which makes it easy to subset the numeric columns and categorical columns for us.

In this function, we can pass the value of include as “object” to get all the categorical variables like this:

**<dataframe-name-here>.select\_dtypes(include="object").columns**

If on the other hand, you pass the value as “number”, you will get all the numeric column names.

**<dataframe-name-here>.select\_dtypes(include="number").columns**

**Correlation analysis of numeric variables**

Our target variable is **TARGET\_deathrate** which is numeric and we would be interested in finding the correlation between this variable and other numeric variables.

It is quite simple in pandas. We can just select all the numeric columns (We discussed this in the last section) and then call “.corr()” function on the data frame which has all the numeric columns.

Something like this:

**<data\_frame\_name>.corr()**

But this function has an issue, it produces all the pairwise correlations which sometimes is not desired. You can do something like the following:

**<data\_frame\_name>["col1 name", "col2 name"].corr()**

It will only give you the correlation between two provided columns.

**Exercise. 2.1 Reading the data and Missing Value Analysis**

Link : **https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD**

Open the Exercise 2.1 file from the lab [hosted here](https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD) (navigate to chapter 2) and do the following:

1. Import the pandas library as done below:

**import pandas as pd**

1. Read the cancer data using pandas.

**cancer\_data = pd.read\_csv(r"https://raw.githubusercontent.com/fenago/MLWorkshop/main/Chapter-2/cancer\_reg.csv",encoding = "latin-1")**

1. Check the shape of the cancer data using .shape attribute of pandas dataframe:

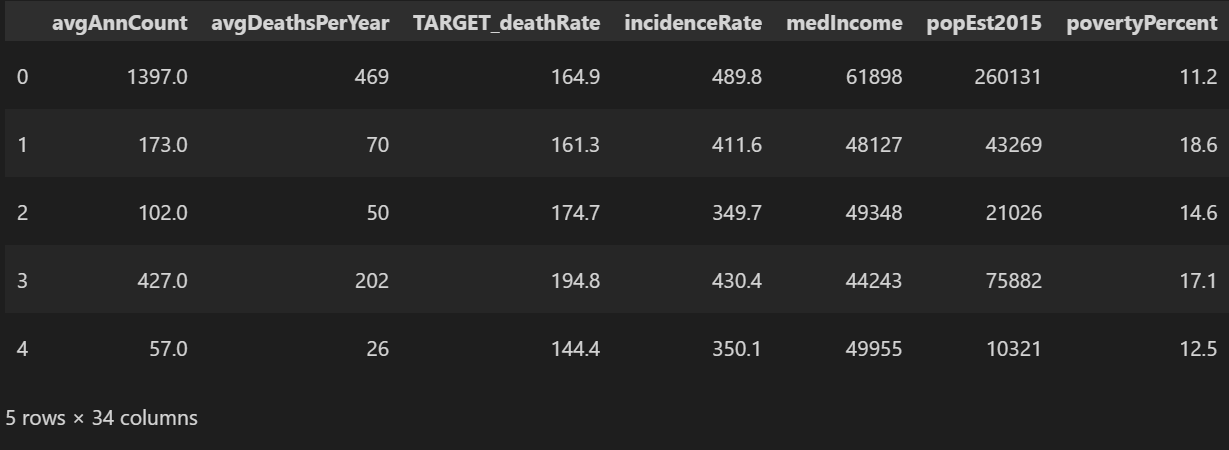
**cancer\_data.shape**

1. View the top few rows of the data.

**cancer\_data.head()**

The output should be the following: ‘





1. How many columns have missing values? To answer this question, we will use the “isnull” function of pandas:

**condition = pd.isnull(cancer\_data).sum().values != 0**

Now, we can just check the sum of above result and see how many columns have missing values as follows:

**condition = pd.isnull(cancer\_data).sum().values != 0**

**sum(condition)**

1. Name those missing columns. To name those columns which contain missing values, we just subset columns on the above condition as :

**cancer\_data.columns[condition]**

1. Check the shape of the new cancer data (after removing the missing columns). How many columns are left now?

To get the new cancer data after removing the missing columns, we can just use the above condition and make it == 0 instead of != 0 and use it to filter the data. It can be done as follows:

**true\_false\_mask = pd.isnull(cancer\_data).sum().values == 0**

**new\_cancer\_data = cancer\_data[cancer\_data.columns[true\_false\_mask]]**

1. Create a new variable called numeric\_vars which only contains numeric columns

We can use the dtypes method to do that which we discussed before. It will look like:

**numeric\_vars = cancer\_data.select\_dtypes(include="number").columns**

1. Create a new variable called categorical\_vars which only contains categorical columns. Just like above code, we can use the dtypes method but this time with a value of include = “object” :

**categorical\_vars = cancer\_data.select\_dtypes(include="object").columns**

1. **Store those variables in a new variable called “top\_correlated” which is highly correlated with the target. Store the top 5 of them.**

For this exercise, we will use the .corr method on pandas dataframe and subset on the target columns, so that we get all the correlations with this variable. Then we can sort these results in descending order so that all the top results show up at the beginning. It will also include its own correlation and we will just index it from 1 instead of 0 to avoid including this variable:

**only\_target\_corrs = new\_cancer\_data.corr()["TARGET\_deathRate"]**

**top5\_vars = only\_target\_corrs.sort\_values(ascending = False)[1:6].index**

**Step.4 Modeling**

So far we have done a little bit of exploratory data analysis on our cancer data. We have been able to remove the columns which contain the missing values, We also same the numerical columns and categorical columns in separate lists.

Also, I hope you would have been able to solve the challenge part of the last exercise. Now, here is the code to save the highly correlated variables in a variable because we will be using these sets of variables to create our model.

**only\_target\_corrs = new\_cancer\_data.corr()["TARGET\_deathRate"]**

**top5\_vars = only\_target\_corrs.sort\_values(ascending = False)[1:6].index**

Now, this new variable called “top5\_vars” will contain the top 5 correlated variables.

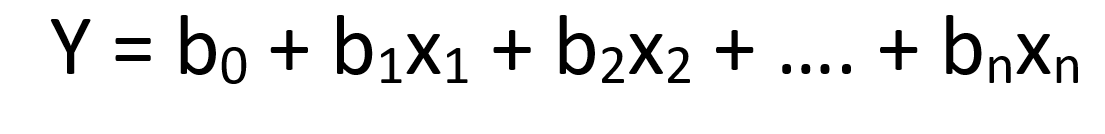
**Linear Regression**

There are many algorithms out there in machine learning which can be used on regression tasks but linear regression is the most simplest one and easy to understand. So, we will start with this.

We will build this algorithm first from scratch and then we will use the sklearn implementation of this.

**What is Linear Regression?**

Linear Regression, as the name suggests, is a simple algorithm for regression problems which relates a dependent variable to the independent variables linearly. So, we can look at the following equation to see this mathematically:



Here, Y is a representation of a dependent variable while all the x’s are for features (or independent variable). For our model, one possible linear regression model may look like:

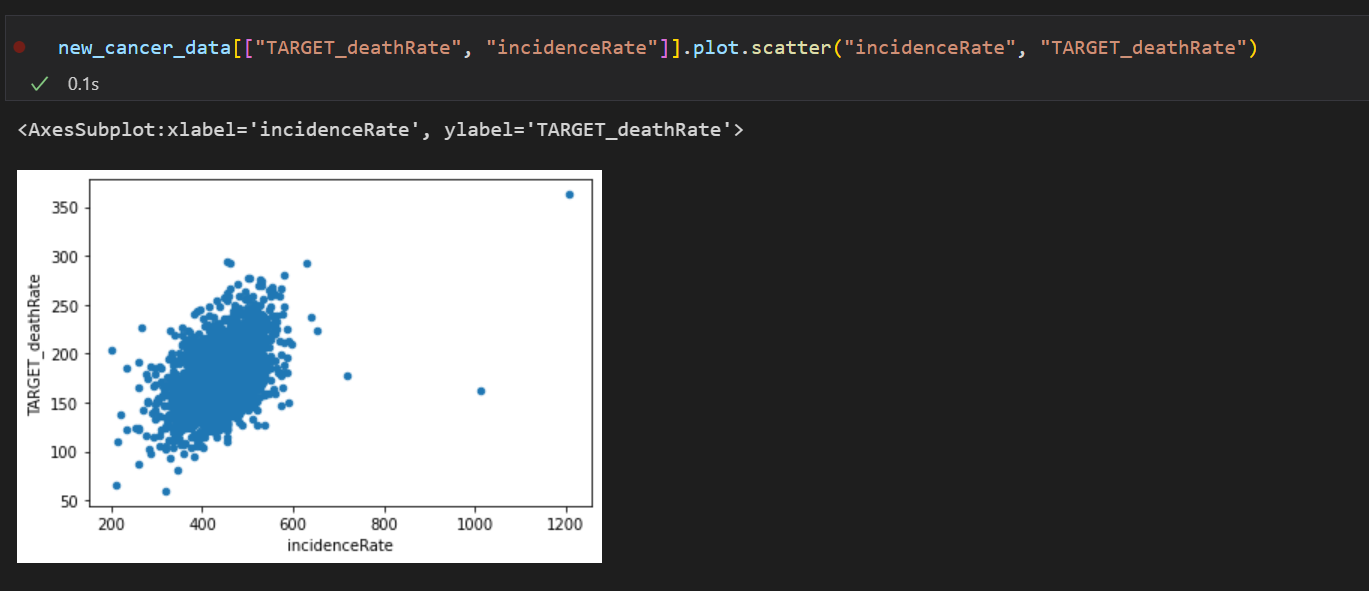
Cancer\_rate = b + b1 \* avgAnnCount

Here, cancer\_rate is linearly related to the average annual count of diagnosed cases. Note, here we have only considered one variable. But, we will build our first model using the top5 variable which we selected earlier (based on the top correlation).

**How does Linear Regression Make the prediction?**

Objective of Linear Regression is to estimate the unknown parameters (coefficients in this case) associated with the linear equation. Once the coefficients are estimated, we can make predictions on the new data by feeding that on the new data.

Let us see this through an example. One of the top variables is **incidenceRate.** Let us look at its relation on the scatter plot:



It looks very positive related to our target.

Suppose, our model has only one variable. So, it would look like:

**TARGET\_deathRate = b + b1 \* incidenceRate**

Suppose, after training the mode, we learned that b1 = 0.6 and bias (b) = 50. Then, suppose for a new example the incidenceRate is 500, our prediction would be calculated as follows:

**TARGET\_deathRate = 50 + 0.6 \* 500 = 350**

**TRAIN AND TEST SPLIT**

Before we actually start the modeling process, we need to set up some way to test our model. Suppose, we train our model on the whole data, we won’t have any way to test the model but just to use the model trained on the whole data as the production model which is not a good practice.

So, we first create some holdout data on which we can evaluate our model, (what methods we will use to evaluate our model is something we will discuss later in this chapter).

There is a package called sklearn which provides some utility functions for machine learning. One of that function is train\_test\_split which takes feature\_matrix (also called as X) and target array (also called as y) and one parameter specifying the size of the test data (called test\_size) and provides 4 different sets, we can call these X\_train, X\_test , y\_train and y\_test.

Here is how we can set up our splits:

**from sklearn.model\_selection import train\_test\_split**

**X = new\_cancer\_data[top5\_vars]**

**y = new\_cancer\_data["TARGET\_deathRate"]**

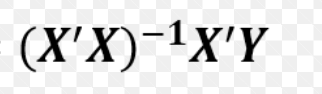
**X\_train,X\_test,y\_train,y\_test = train\_test\_split(X, y, test\_size=0.30)**

**Linear Regression From scratch**

To calculate linear regression from scratch, we can solve the normal equations (which are a set of equations we get while minimizing the squared sum of error).

This might be confusing for you. You can safely ignore this part as we will also look at the sklearn’s implementation.

So, in calculus, we differentiate the Squared sum of errors with respect to parameters and while minimizing we get the optimal value of the parameters (those coefficients we want). They are calculated by the following equation:



Yes, it is just that. If we solve the above equation, we will get the desired value of the coefficients.

But there is a catch, right now our X matrices only contain feature values. This is how it should be but it turns out when we are going to solve this equation, we also have to add a new columns in the front for 1s which will account for bias and hence the first value will be coefficient of this hypothetical column of ones term, or in other words, it would just be bias.

Let us look at the implementation using numpy:

**def lr\_from\_scratch(X, y):**

**new\_X = np.hstack((np.ones((X.shape[0],1)),X))**

**first\_term = np.dot(np.transpose(new\_X), new\_X)**

**second\_term = np.dot(np.transpose(new\_X), y)**

**result = np.dot(np.linalg.inv(first\_term), second\_term)**

**return result**

This function is taking two numpy arrays X and y and returning a result which holds the bias and coefficients. Let us understand each line one by one:

1. In the first line, we are creating a new X by adding 1s in the beginning
2. Then, we compute X’X and call it first\_term
3. Then, we compute X’y and call it the second term. We are using np.dot function which does matrix multiplication
4. Finally, we get the inverse of the first term and multiply it with the second term. Voila, We have our result.

**Exercise. 2.2 Linear Regression from scratch**

Link : **https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD**

Open the Exercise 2.2 file from the lab [hosted here](https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD) (navigate to chapter 2) and do the following. This will have the train and test splits made for you and also it will have the above implementation ready to use.

1. Run all the cells already present in the exercise 1.2, it just contains the code from the previous exercise.

**import pandas as pd**

**import numpy as np**

**cancer\_data = pd.read\_csv(r"https://raw.githubusercontent.com/fenago/MLWorkshop/main/Chapter-2/cancer\_reg.csv",**

**encoding = "latin-1")**

**true\_false\_mask = pd.isnull(cancer\_data).sum().values == 0**

**new\_cancer\_data = cancer\_data[cancer\_data.columns[true\_false\_mask]]**

**numeric\_vars = cancer\_data.select\_dtypes(include="number").columns**

**categorical\_vars = cancer\_data.select\_dtypes(include="object").columns**

**only\_target\_corrs = new\_cancer\_data.corr()["TARGET\_deathRate"]**

**top5\_vars = only\_target\_corrs.sort\_values(ascending = False)[1:6].index**

1. Split the new cancer data into train and test. As discussed, we do this using the train\_test\_Split function of sklearn. We provide the test\_size = 0.3:

**from sklearn.model\_selection import train\_test\_split**

**X = new\_cancer\_data[top5\_vars]**

**y = new\_cancer\_data["TARGET\_deathRate"]**

**X\_train,X\_test,y\_train,y\_test = train\_test\_split(X, y, test\_size=0.30)**

1. Use the lr\_from\_scratch function on X\_train and y\_train arrays. This function implements the Linear Regression from scratch using numpy:

**import numpy as np**

**def lr\_from\_scratch(X, y):**

**new\_X = np.hstack((np.ones((X.shape[0],1)),X))**

**first\_term = np.dot(np.transpose(new\_X), new\_X)**

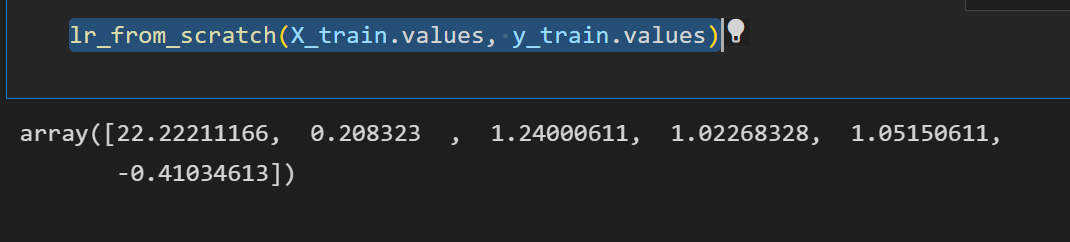
**second\_term = np.dot(np.transpose(new\_X), y)**

**result = np.dot(np.linalg.inv(first\_term), second\_term)**

**return result**

**lr\_from\_scratch(X\_train.values, y\_train.values)**

1. What values do you get?



1. **[Challenge]** Implement a function which takes one X and calculates the prediction using the coefficients we just found. It should be easy. We need to add the bias to the weighted sum of coefficients and matrix values. Try to do this problem on your own before looking at the solution

**Solution to the Challenge**

**def predict\_from\_lr\_scrach(X):**

**trained\_coefs = lr\_from\_scratch(X\_train, y\_train)**

**new\_X = np.hstack((np.ones((X.shape[0],1)),X))**

**predictions = np.dot(new\_X, trained\_coefs)**

**return predictions**

This function is first making a call to the lr\_from\_scratch function and getting the trained coefficients. Once that is done, it is again adding the 1s to the X input which is passed to the current function. Finally, we are multiplying the new\_X matrix with the trained coeff.

1. Use this function to make predictions on both training and testing sets, I.e X\_train and X\_test. Store your predictions in predictions\_train and predictions\_test respectively.

**train\_predictions = predict\_from\_lr\_scrach(X\_train)**

**test\_predictions = predict\_from\_lr\_scrach(X\_test)**

**Linear Regression Using Sklearn**

Now, the fun part. The Sklearn library makes it easy for us to implement ML algorithms.

We already have our train and test setup and just need to implement the sklearn part. Any model is trained in sklearn using the following steps:

1. Import the required model class from the package, here we need to import LinearRegression from sklearn.linear\_model

**from sub\_package import Model\_class**

1. Initialize the class by saying:

**model\_name = Model\_class()**

1. Call the fit method to train the model:

**model\_name.fit(X\_train, y\_train)**

1. To make the predictions:

**predictions\_test = model\_name.predict(X\_test)**

Yes, it was this simple. We can fit and predict using any model from sklearn using just the above four steps.

**Exercise. 2.3 Linear Regression using sklearn**

Open the Exercise 2.3 file from the lab [hosted here](https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD) (navigate to chapter 2) and do the following. This will have the train and test splits made for you.

1. Run the code which are available for you in this notebook
2. You just need to use the above four steps to train a LR model from sklearn

**from sklearn.linear\_model import LinearRegression**

**lr = LinearRegression()**

**lr.fit(X\_train, y\_train)**

1. Compute the train and test predictions using the .predict method we just discussed. Save these into predictions\_train\_sklearn and predictions\_test\_sklearn

**train\_predictions\_sklearn = lr.predict(X\_train)**

**test\_predictions\_sklearn = lr.predict(X\_test)**

1. Compare these predictions with our own implementation, these should be exactly the same.

**train\_predictions = predict\_from\_lr\_scrach(X\_train)**

**test\_predictions = predict\_from\_lr\_scrach(X\_test)**

1. **[Challenge]** Call the .coef\_ and .intercept\_ attribute of model\_name class and make sure that model coefficient of sklearn implementation and our earlier implementation are the same.

**lr.coef\_**

**Step.5 Evaluation**

We have trained our model, predicted on train and tested using the trained model but how does our model perform?

To answer that, we need to look at evaluation metrics.

**What is the metric?**

Metric is one single value which tells you about the performance of your model. By itself, it has no worth but when you have two metric values, both can be compared. So, metrics are useful only when they are available for many algorithms so that all of those algorithms can be compared.

Metrics can be different for different ML tasks. There can be two ML tasks:

1. **Regression**
2. **Classification**

Since the task is regression in this chapter, let us continue with that.

In regression, our basic objective is to predict the observations as closely to the true values as possible. Think about it for a second. Let us say I have one model which predicts one observation as 14 while the true is 13.

**Can you think of a metric here?**

We can just take the difference between these two and say metric = 14 - 13 = 1

Now, let us say we have another model which predicts for the same observations, 13.7.

Here, the same metric will be 0.3. Hence, the second model wins because it has low error.

First three metrics i.e MAE, MSE and RMSE are based on difference of the true value and predicted values.

Since, we don’t have just one observation but rather many observations, we need a way to combine those. We can just take the average (mean) of those.

**MAE (Mean absolute error)**

Here, we take the difference between true and predicted values and then we take absolute (to make it positive). Then, we take the mean of all the values and we get MAE.

**MSE (Mean squared error)**

Here, everything is the same but we take square here rather than absolute.

**RMSE(Root mean squared error):**

It just takes the square root of MSE

Here is a brief calculation showing these three metrics calculation for a dummy data:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Obs** | **actual** | **predicted** | **actual -predicted** | **absolute** | **squared** | **RMSE** |
| **1** | **13** | **9** | **4** | **4** | **16** | **3.67** |
| **2** | **10** | **13** | **-3** | **3** | **9** |  |
| **3** | **23** | **21** | **2** | **2** | **4** |  |
| **4** | **25** | **20** | **5** | **5** | **25** |  |
| **Mean** |  |  |  | **3.5** | **13.5** |  |
|  |  |  |  | **MAE** | **MSE** | **sqrt(MSE)** |

Let us now discuss the following two:

1. **RMSLE**
2. **MAPE**

RMSLE is just taking the root of mean squared **log error instead of the usual squared that we take in RMSE. It is usually not required because for a regression task usually the following four metric are sufficient:**

1. **MAE**
2. **MSE**
3. **RMSE**
4. **R2**

**MAPE on the other hand takes the mean absolute percentage error which means it simply does not take difference but percentage of each observation and then calculate the mean absolute.**

Finally, let us see the R2 which is actually quite an important metric but should not be looked at alone.

R2 is the percentage of variance explained by the model. So, if r2 is 0.7, it means that our current model is able to explain the 70% variation in the value which we are trying to predict and leaving the 30% to random causes.

It should not be looked at alone because sometimes it may happen that although R2 is large but other error metrics are also large. Also, r2 can simply increase by adding redundant features which does not help the model and adding to the garbage only.

Ideal values of errors should be 0 and R2 should be 1. But it is usually the case. Also, MSE will be usually greater than MAE and RMSE because other two (MAE and RMSE) are on the same scale as the data while MSE is not (Since difference has been squared) while for RMSE, after averaging we again take the root which makes the effect of squaring go away.

How do we choose the best model based on metric?

1. If we are looking at just one metric, then we can just look at which model performs best for that metric.
   1. For example, if I am looking at MAE, then I can simply look at that model which has the least MAE.
   2. Or if I am looking at R2, then I can look at the model which has the greatest R2.
2. If we are looking at more than one metric, then we can look at which model is winning in most of the cases.
   1. So, for example if I have a model which is best according to 2 metric while the other one is best on 4 metric then I will choose the later one
3. Different metrics should be interpreted differently. For error related metric, we want to have low values while for R2, we want to have high values.

**Using RMSE**

Let us use RMSE to report the evaluation of our model. We need two sets of values to compute the RMSE i.e predicted and actual values.

**RMSE from scratch**

As we discussed earlier, we can calculate RMSE by taking the square root of the sum of squared difference of actual and predicted value. It should be easy to implement.

**RMSE using sklearn**

As you might have guessed, sklearn also has functions for computing various metric values.

It can be imported as:

from sklearn.metrics import mean\_squared\_error

**Exercise. 2.4 RMSE using numpy and sklearn**

Link : **https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD**

Open the Exercise 2.4 file from the lab [hosted here](https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD) (navigate to chapter 2) and do the following. The train and test predictions have been provided to you.

1. Rull all the code
2. Create a function to compute root mean squared error

**def rmse\_from\_scratch(y\_true, y\_actual):**

**squares = np.square(y\_true - y\_actual)**

**sum\_sq = np.sum(squares)**

**sum\_sq /= len(y\_train)**

**return np.sqrt(sum\_sq)**

1. Use the function to compute the RMSE on train and test predictions

**rmse\_from\_scratch(train\_predictions, y\_train)**

1. Use the sklearn function to compute rmse. Don’t forget to add squared=True.

**from sklearn.metrics import mean\_squared\_error**

**mean\_squared\_error(y\_train, train\_predictions, squared=False)**

1. Compare both the results. They should be the same.
2. **[Challenge] Research and find out how to compute other metrics for regression which we discussed using sklearn.**

**Step. 6 Deployment**

Finally, if you are satisfied with the evaluation values you are getting in the previous step and trying all the different algorithms, you can finalize the model. After finalizing, we can take one of the following steps:

1. Share the predictions with the appropriate team, so that they can take appropriate action based on the predictions.
2. Create a web based framework to serve our predictions to users

**Activity**

Link : **https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD**

Open the Activity 1.ipynb file from the lab [**hosted here**](https://mybinder.org/v2/gh/fenago/MLWorkshop/HEAD)(navigate to chapter 2) and do the following. The objective is to predict life expectancy. You need to perform the following steps on this data:

1. Read the data in Pandas
2. Check the number of rows and columns.
3. Find out all the columns which have missing values
4. Find out all the columns which are numeric columns
5. Find out all the columns which are categorical columns
6. Remove all missing columns from the data
7. Use the .corr method to find out the pairwise correlation matrix of all the numeric columns
8. Select the target variable and Sort according to values.
9. Extract the top 10 features which are correlated with the target
10. Use them to train a Linear Regression model and get the coefficients
11. Calculate the RMSE using sklearn to evaluate your model.

**Project**

Perform an EDA and regression modeling steps on the Kaggle Vehicle Price data which is available [**here**](https://www.kaggle.com/datasets/nehalbirla/vehicle-dataset-from-cardekho).

Data link: **https://www.kaggle.com/datasets/nehalbirla/vehicle-dataset-from-cardekho**

**Summary**

In this chapter, we looked at machine learning for Regression and followed the CRISP DM approach we discussed in the last chapter.

We looked at various techniques to do exploratory data analysis and also how to handle missing values.

We created our own Linear Regression model from scratch and also used sklearn’s API to do the same.

We also looked at what metrics are and how they help us evaluate the model. We looked at calculating RMSE for our data both from scratch and using the sklearn function.

All this while, we discussed key concepts and then practiced that with some exercises.

Finally, we closed the chapter with an activity for you to try.