Data-driven curve fitting using Linear Regression

Vaibhav Mahapatra
Data Science Department
Indian Institute of Technology Madras
Chennai, India
me19b197@smail.iitm.ac.in

***Abstract*—This article describes how Linear Regression can be used to fit a curve through a set of given data points. This document demonstrates how an empirical model can learn to predict or estimate quantities of interest based on past-accumulated data. We will first explore the theory behind this approach and then apply it on a real-world data set using Python and observe it's results.**

Improvement: Introduced Polynomial Regression in theory section. Elaborated on Kernel Method for Linear Regression and its pros and cons. Improved MSE of fit on Linear Regression.

I. Introduction

Linear Regression is an machine algorithm which comes under the subset of supervised learning. Regression aims at reliably predicting a continuous output based on various numerous and categorical inputs. It does so by learning from a set of previous data which is provided to the algorithm in input-output pairs. Its applications are multifarious. It can be used for descriptive analytics, i.e., to understand relationships between multiple variables. It can also be used for prescriptive analytics, i.e., using current data to predict and prescribe actions for the future.

Building Linear Regression Models broadly involve 3 iterative steps: Data preparation, Model fitting and Model validation. The final step of selecting the model is done after multiple iterations of the previous 3 steps and procuring satisfactory results. Typically, for the first iteration, it takes around 70% time to clean and prepare the data, 20% time to fit models (preferably multiple), and 10% time to validate the results of the model. The theory for the first of Data Cleaning and Preparation will not be explored in detail as it is a combination of intuition and statistical inferences. We will comprehensively explore the next steps of Model fitting and Validation in this article.

Regression Models can be fit in many ways. The most common method is to minimise the mean squared error(MSE) between the model's prediction and the ground truth of the output variable. This method of fitting is also called the *Ordinary Least Squares* fit. The explanation of why MSE is used as a metric for Regression is explained in the next section. The MSE can be minimised in various ways, namely by using *Matrix-Vector algebra*, *Gradient Descent*, etc. These approaches will be discussed with mathematical rigour in the next section.

The final section of this article will encapsulate a real-world use case of Linear Regression. We will build a Regression model on data procured from the USA Census Bureau. The process will help us formally establish how various factors can contribute to cancer incidence and mortality rates among American citizens. Broadly, we'll examine the influence of geographical and socioeconomical factors in predicting Cancer incidence and mortality rates. This article will contain various visual plots to help us intuitively process the various inter-relationships at play, along with the results of our approach.

II. Linear Regression

A. Defining the Problem

We have *n* samples of *d*-dimensional input vectors $x_1, x_2, ...x_d$, where each component corresponds to an observed quantity or feature. For each corresponding sample, we have a 1-dimensional output variable represented by *y*. This y is the ground truth. Our objective is to determine a functional map $\hat{f}(x)$, such that MSE, as defined below, is minimum:

$$\hat{y} = \hat{f}(x) = w_0 + w_1 x_1 + w_2 x_2 + ...w_d x_d \tag{1}$$

$$MSE = \frac{1}{n}\sum_{i=1}^{n} |\hat{y}_i - y_i|^2 \tag{2}$$

Finding the best functional map is equivalent to finding the best weights vector $w = (w_0, w_2, ...w_d)$

B. Mean Squared Error as a metric

Mean square error can be referred to as the *Loss Function* for this problem. The first explanation for why this metric is chosen can be an intuitive explanation. By looking at the loss function's definition in the previous subsection, we can see that we're effectively trying to minimise the *distance* or *error* between the prediction and the ground truth. Even though this choice seems intuitive, in this section, we will justify this choice with mathematical rigour.

Let

$$Y = w^T X + \epsilon$$

where w is some d-dimensional weights vector and ϵ can be characterised as some normal noise $N(0, \sigma^2)$, which is independent of X. With these assumptions, we can easily say

$$Y|X = x \ N(w^T x, \sigma^2)$$

$$P(Y = y|X = x, W) = P(\epsilon = y - w^T x|X = x, w)$$

$$= P(\epsilon = y - w^T x) = \frac{1}{\sqrt{2\pi)\sigma^2}} \exp\left(\frac{-1}{2\sigma^2}(y - w^T x)^2\right))$$

If we draw m independently and identically distributed samples $(x_1, y_1), (x_2, y_2)....(x_m, y_m)$ we can split the likelihood(L) of this occurrence of samples as follows:

$$P(y_1....y_m|x_1....x_m, w) = P(y_1|x_1, w)....P(y_m|x_m, w)$$

$$L(w) = \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi)\sigma^2}} \exp\left(\frac{-1}{2\sigma^2}(y_i - w^T x_i)^2\right))$$

$$log(L(w)) = \sum_{i=1}^{m} \left(\frac{-1}{2\sigma^2}(y_i - w^T x_i)^2\right)) + const.$$

$$-log(L(w)) = c\sum_{i=1}^{m}((y_i - w^T x_i)^2) + const.$$

From the above expression, we can see that to the obtain Maximum Likelihood Estimate of w, we have to minimize the Negative Log Likelihood. Doing so implies minimising the Sum of squared errors which basically implies minimising the MSE.

C. Determining Weights by Minimising MSE

1) Matrix-Vector formulation: The first method we will explore for minimising MSE is simply setting the gradient of the Loss function to zero and finding the weights corresponding to the Minima. Let us take an input (nxd) data matrix X, whose each row denotes each sample of data and each column denotes each attribute. We'll also represent the outputs and weights a vectors y (n-dimensional) and w (d-dimensional). Hence, we can represent the Loss(J) as a function of w as follows:

$$J(w) = \frac{1}{n}||Xw - y||^2$$

To minimise J(w), we'll set it's gradient with respect to w to zero

$$\nabla_w J(w) = \frac{2}{n} X^T (Xw - y) = 0$$

$$X^T X\hat{w} = X^T y$$

If $X^T X$ is invertible, then

$$\hat{w} = (X^T X)^{-1} X^T y \tag{3}$$

(3) can hence be directly implemented to determine the weights.

2) Gradient Descent: The above formulation is mathematically accurate but computationally heavy. This is because it not only involves matrix multiplications but also taking inverse of a (dxd) matrix. The time complexity of the problem will increase significantly with increase in data points and dimensions. To tackle this, the iterative algorithm like Gradient Descent is used. The main idea is to estimate w and improve it in each iteration based on the model's performance. Mathematically, this can be represented as:

$$W_{k+1} = W_k - \alpha \nabla_W J \tag{4}$$

In (4) α refers to a the step-size, a parameter chosen by the user, and k refers to the k^{th} iteration of the algorithm. From the equation, we can see that we're updating W in the steepest descent direction of the Loss Function J. This implies that with each iteration, we're updating W such that the loss J reduces. As we move closer to the minima, the gradient reduces and the parameter updation gets slower. We stop running the algorithm based on a user-set tolerance for either the change in weights, change in Loss value, etc. At the end of running the algorithm, we have an estimate of the weights which is *near* the optimum value of weights. From the practical point of you, this estimate suffices in the bigger picture.

D. Model Evaluation

In this section, we'll discuss the various methods of evaluating models and choosing the right model. Our objective is to choose the model with works the best on *unseen* data. To evaluate the model's performance, we first split the given data into 2 sets: training data and test data. In practice, a 70-30 or a 80-20 training-test split is fairly reasonable. The training set is used to train the model, whereas the test data acts as the *unseen* data to evaluate the performance of the model. A few methods of estimating the performance of a Regression model are mentioned below:

- Parity Plot: In this method, we plot the predictions on test data against the actual output values to visualise the model's fit. Ideally, for the *perfect* model, all points will lie on the $y = x$ line. But in reality, points on this plot will lie around and near this line. A model whose points are more tightly distributed around this line will have a better fit than the model whose points are more scattered. Fig. 1 will help visualise what this plot typically looks like.

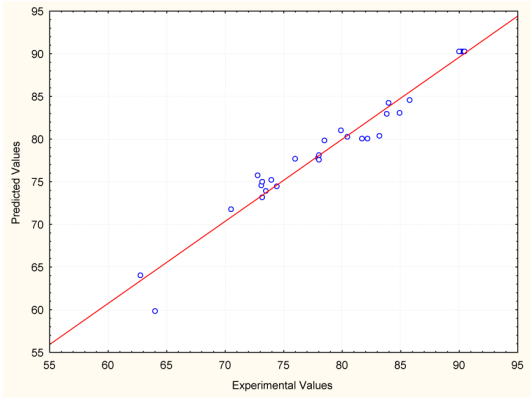

Fig. 1. Example of a Parity Plot

- Error metrics: Various metrics like Mean Absolute Error, MSE and R2_score can be used to evaluate models. We've defined MSE before as (2). The other metrics can be defined as follows:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|\hat{y}_i - y_i| \tag{5}$$

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \quad (6)$$

Similar to MSE, a model with less MAE will be a better fit than one with more. As for R2_score, we want a model with a score closer to 1.

III. Real Life Use-Case of Linear Regression

A. Dataset and Objectives

We will be working on the US census data for poverty and average income figures. Along with cancer incidence and mortality rates demographic data, we intend to use Regression to see how factors like geography, poverty, income background, access to insurance, etc. affect chances of incurring cancer. Broadly, the objective is to clean data, generate relevant features and use Regression to build a model that predicts cancer incident rate and mortality rate.

B. Data Pre-processing

Firstly, we will drop attributes that are either irrelevant or too sparse. Definition of the *Fips* attributes is unclear, with no clear impact on our target variables. Hence, these attributes are dropped from the dataset. We also drop all the sub-classes of *Med_Income* as they are heavily sparse. With too many *NaN* values, it is unreasonable to try and fill them with an estimate like mean or median. There is also some unknown data in the *Mortality_Rate* and *Incident_Rate* columns. Those instances or rows are dropped as only a fraction of the dataset have this issue. It shouldn't affect our analysis to a great extent. On analysing the correlation between various features, we found out the gender splits are irrelevant and do play a significant role. Hence, we removed the columns *M_With , M_Without, M_Poverty, F_Poverty, F_With, F_Without* and keep only the *All_With, All_Without and All_Poverty* columns. Some numeric columns have some inconsistencies in datapoints. These are dealt with using data and text processing.

C. Polynomial Regression

The same linear regression methodology can be extended to fit polynomials to the data. To implement this, we'll have to include an extra step before doing the OLS fit, that is to create a map from the linear space to the polynomial space. Let this feature mapping be represented by ϕ. We can represent the news equations as shown below:

$$X - - > \Phi(X)$$

$$w_{poly} = (\Phi(X)^T \Phi(X))^{-1} \Phi(X)^T y$$

This same method works in this case because the problem is still linear in the parameters we are trying to estimate. One drawback of this method is that as we increase the polynomial degree, the mapping will increase dimensions, making the OLS weights calculation computationally heavy.

D. Kernel Linear Regression

In the case that training points are less than the dimension of the data, or they are comparable, the Kernel interpretation can help solve the problem faster. It is a clever mathematical trick thanks to which, instead of dealing with dxd matrix $X^T X$, we can solve the same problem with a nxn matrix. Exact implementation of this can be seen in this site: https://web.mit.edu/modernml/course/lectures/MLClassLecture3.pdf

E. Data Visualization and Feature Engineering

On some preliminary analysis, I realised that feature like *All_With, All_Without and All_Poverty* are dependent on the overall population of the area we're looking at. To remove the influence of the population size, I normalised the features and generated new ones under the tags of *Poverty_Norm and Insurance_Norm*.

Now we'll analyse numeric data via data visualisations and correlations. I plotted some regression plots using seaborn and a correlation matrix as seen in Fig. 2, to intuitively draw insights on how the various variables are correlated. My main insights are as follows:

1) There seems to be a fairly significant positive correlation between our target variables (Incidence_Rate and Mortality_Rate) and Poverty in a region. This seems to be fairly intuitive as more poverty implies less access to proper nutrition and healthcare.
2) There's a negative correlation between Med_Income and our target variables. This goes hand in hand with our previous observation as those with more Income will have higher safety and chances of survival.
3) There doesn't seem to be a strong correlation between Insurance_Norm and our targets. The graph shows some slight negative correlation, but it doesn't seem significant.
4) There is a fairly strong negative correlation between Avg_Ann_Incidence, Avg_Ann_Deaths and our target variable

Subsequently, we'll analyse categorical data, mainly the *State* each row belongs to. I dropped the column *AreaName* as each row has a unique AreaName, which doesn't help us in our analysis. To see the influence of State on Incidence and Mortality Rates, I plotted bar plots in Fig. III-E and noted the following insights:

1) The distribution of both Incidence_Rate and Mortality_Rate seems to be fairly similar, but have a scale difference.
2) Broadly, we can bucket states into 3 classes, ones with < 50 Incidence_Rate, $>= 50$ but < 75 Incidence_Rate and $>= 75$ Incidence_Rate.

In line with my second observation, I bucketed the states into 3 classes and generated a categorical feature with label-encoding.

At the end of this analysis and feature engineering, we can summarise our finalised dataset as follows:

- Target Variables- Incidence_Rate, Mortality_Rate

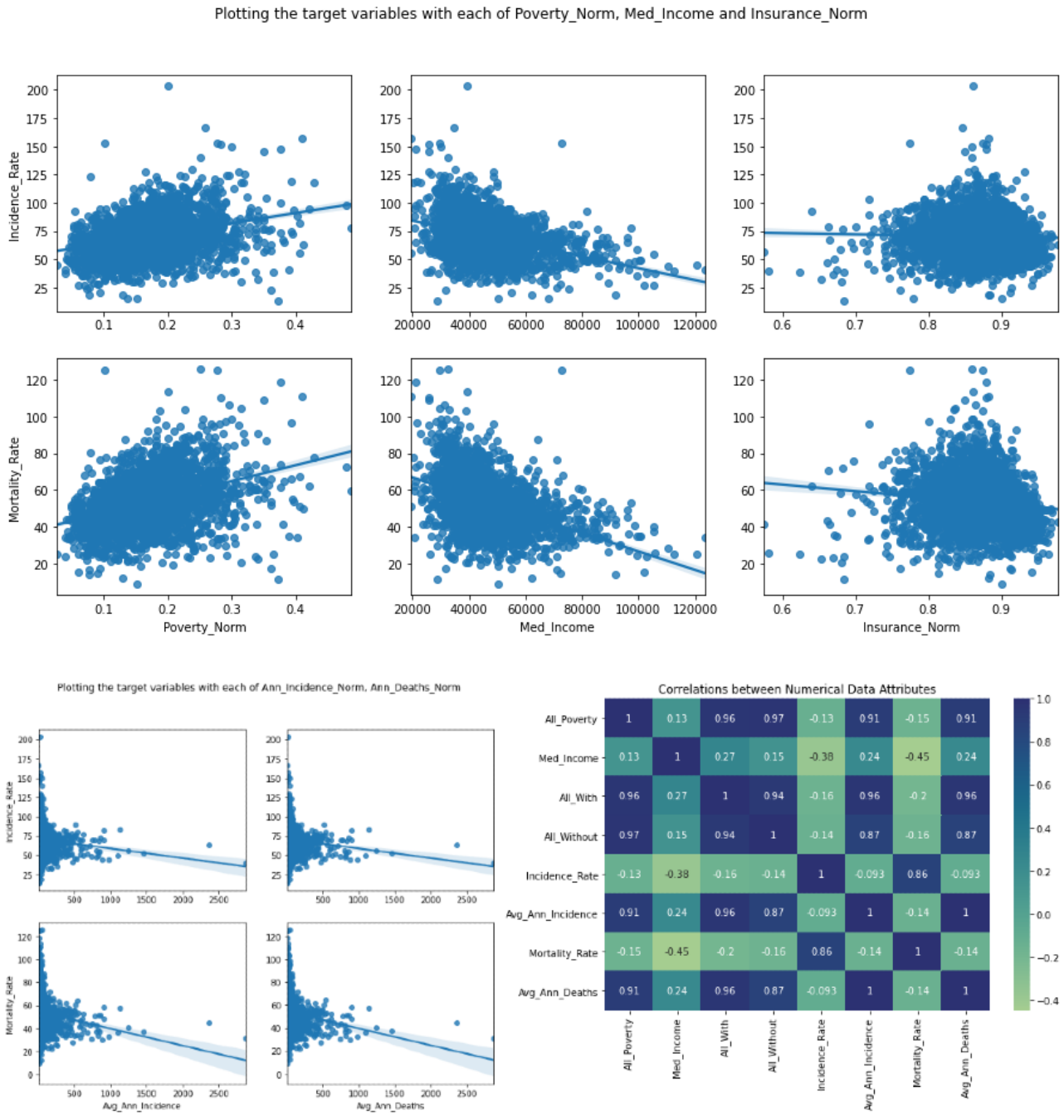

Fig. 2. Data Visualisation on Numeric Data

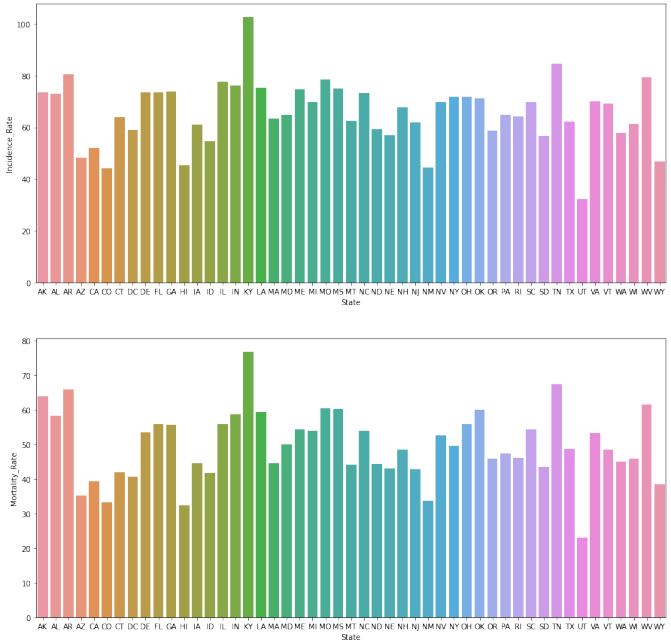

Fig. 3. Distribution of Target Variables Across States

- Numeric Attributes- Poverty_Norm, Insurance_Norm, Med_Income, Avg_Ann_Incidence and Avg_Ann_Deaths
- Catergorical Attribute- State

F. Model Fitting and Evaluation

The Model fitting process can be broken down into the following steps-

1) Choosing X & y- Select features from the dataset and store data in a matrix X. Store the outputs as a vector y
2) Polynomial transform(optional)- Generate polynomial features from X to capture non-linear relationships between X & y
3) Normalise data(optional)- Normalising all attributes of X in a small range like [0,1] can help increase the training speed of the Regression Model.
4) Train-test split- Split the X matrix and y vector into training and test matrices. Preferably jumble the rows before doing so, to make the model more robust.
5) Train model- train a Linear Regression Model on (X_train, y_train)
6) Evaluate model and reiterate- evaluate the model using various previously discussed metrics and parity plots. Redo steps with different features and polynomial degrees till we get the best performing model.

For both, predicting the Incidence_Rate and Mortality_Rate, I chose all the features and Polynomial Features of degree 2. These hyperparameters seemed to work the best for the final dataset I made. The test results for both models are summarised below.

The Parity Plots to graphically visualise the goodness of fit can be seen in Fig. 4 and Fig. 5

TABLE I
REGRESSION MODELS PERFORMANCE

Metric	Incidence_Rate Model	Mortality_Rate Model
MAE	10.59	8.13
MSE	191.90	111.95
RMSE	13.85	10.58

Fig. 4. Incidence_Rate Regression Parity Plot

IV. CONCLUSION

Linear Regression has enabled us to establish some relationships between the inputs and outputs. The data analysis we've done plus the models we built have captured underlying correlations between data but not to a great extend. To improve the prediction capacity, we can build more complex models on data which capture non-linear relationships between data. Examples of such models are Non-Linear Regression, Neural Networks, etc.

REFERENCES

[1] Kumari, Khushbu & Yadav, Suniti. (2018). Linear regression analysis study. Journal of the Practice of Cardiovascular Sciences. 4. 33. 10.4103/jpcs.jpcs_8_18.
[2] Schneider A, Hommel G, Blettner M. Linear regression analysis: part 14 of a series on evaluation of scientific publications. Dtsch Arztebl Int. 2010 Nov;107(44):776-82. doi: 10.3238/arztebl.2010.0776. Epub 2010 Nov 5. PMID: 21116397; PMCID: PMC2992018.
[3] https://www.analyticsvidhya.com/blog/2021/05/multiple-linear-regression-using-python-and-scikit-learn/
[4] https://towardsdatascience.com/polynomial-regression-with-scikit-learn-what-you-should-know-bed9d3296f2

Fig. 5. Mortality_Rate Regression Parity Plot