ML BOOTCAMP – REPORT

Yogita Singh
IIT(ISM) Dhanbad

➔ General:

- Importing a csv file in google colab.
  - Tried uploading the csv files using `google.colab import files`
    `Uploades = files.upload()`
    `#Choose a file`
    Problem: Took too long to upload a file every time you need to work on the code.
  - Tried uploading the training and test data set to google drive and then importing it to google colab.
    Problem: FileNotFound. There was some error in the path or directory.
  - Uploaded the test and training data to GitHub and then copied the url of the raw dataset to import the csv files.

- Extracting the predicted values of the test data from google colab.
  - Used
    `df_predicted = df.DataFrame(data =y_predicted, index=df_test.ids)`
    `df_predicted.to_csv()`
    Then I downloaded the csv file from the folder in google colab.

- Normalization: We use normalization to transform all our features to a similar scale. It, in turn, improves the performance and training stability of the model by giving us a smooth and circular gradient descent contour plot and hence a straight path to get the lowest cost possible.
  Here, We used z-score normalization in each of the different algorithms. It scales all the features in a small range and also accounts when there are outliers in our data. The formula for z-score normalization is:
  `X_norm = (X_train – mean)/(standard deviation)`

- The test data, for which the predicted values have to be obtained, also needs to be normalized with the same value of mean and standard deviation as that used in normalizing the training data. This is done so because the function that is obtained to predict the values used the normalized training data. Hence, to obtain correct prediction values, the test data also needs to be normalized.

➔ Linear Regression:

- In both of the regression algorithms, we have a gradient descent code. In a simple Linear Regression model, we try to find and fit a line such that the distance of the line from the data points is minimum using the training datasets.

  $$y=wx+b$$

  This line has 2 unknown terms, weight and bias that we find out using the gradient descent algorithm. A simple linear regression model works only when we have 1 input feature. We use a multiple linear regression model when we have more than 1 input features.
  In multiple linear regression, each of the input feature is appointed a weight which gets calculated using the gradient descent algorithm.

  $$Y=w_1x_1+w_2x_2+w_3x_3+\ldots\ldots\ldots+b$$

- In linear regression, we first train the model to find the best fit line to predict the value of y using the value of x. Linear Regression has a mean squared error cost function which is the summation of the squared difference between the predicted values and the actual values. We try to minimize this cost function by running a gradient descent function which lower the cost function by iteratively updating the values of weights and bias. With each iteration, the gradient descent function takes a particular step in the direction of the local minima. The value of this step is determined by a learning rate α. If the learning rate is too small, the function will take a long time to converge. And if the learning rate is too big, the function might start to diverge as it may skip over the local minima.

- I have used a learning rate of 0.01 for 1000 iterations. It takes approx 2 seconds to run and converges at a minimum cost of 4769.

➔ Polynomial Regression:

- Polynomial Regression is just Linear Regression but with some additional added features. The extra features from the dataset are extracted like x_1^2, x_1x_2, x_2^2 etc.

- To extract the features, I tried to use an n-number system approach (where n is the degree of the polynomial equation). An 'abc' number for x1,x2 and x3 would give a term such as $x_1^a.x_2^b.x_3^c$.
  For example, if I need a polynomial equation for 3 variables of degree 3, we will take a 9 bit(3^3) number in base 4. Starting from 000, the number will be incremented by 1 and the term obtained will be stored as a new feature.

0	0	1	->	x
0	0	2	->	x^2
0	0	3	->	x^3
0	1	0	->	y
0	1	1	->	xy

Etc.

- The extra features now obtained were normalized and the gradient descent function was run. Both Linear Regression and Polynomial Regression use the same mean squared error cost function.
- I ran gradient descent on a 5 degree polynomial as it gave me the lowest cost for less number of iterations, keeping the learning rate constant, as compared to other n degree polynomial. I got a total of 55 features.
  I used a learning rate of 0.17 for 20000 iterations. After 20000 iterations, the cost was approximately 105. If I increased the number of iterations, the cost approached zero but since it was taking a lot of time, I capped the number of iterations at 20000.
  I used a learning rate of 0.17 because when I tried to use a learning rate any higher, the cost started to increase significantly after a certain number of iterations and then started to give a nan value.

➔ Logistic Regression:

- Logistic Regression model is most often used for classification problems, however, it can be used for regression problems as well. It is used to classify dataset into one of the 'n' target variables. Unlike Linear and Polynomial Regression, Logistic Regression uses a log loss cost function.

- When logistic regression uses the mean squared error cost function, the curve for the cost comes out to be non-convex (or wiggly). This means that there are multiple local minimas which, in turn, doesn't give very accurate results. Hence, we use the log loss cost function which looks like:

  $$\text{Loss} = -y*\log(fw\_b)-(1-y)*\log(1-fw\_b)$$

  A sigmoid function is used to find the probability of a value being 1. A sigmoid function is used to find the predicted value because a sigmoid function maps any real value between 0 and 1.
  Now, In this loss function, if the probability value matches the target value, the loss comes out to be 0. Else the loss is high depending on how far the probability value is from the target value. For example, if the target value of a particular datapoint is 0, and the predicted value is also zero, loss = 0*log(0)-1*log1 = 0. However if the predicted value is 1, then the loss = 0*log(1)-1*log(0) which will come out to be infinity.

- The training and test data provided to us, however, has multiple target variables. So, to classify the different datapoints into the target variables, we use the one versus all

approach. In one versus all, we create N distinct binary classifiers (where N is the number of target variables), each classifier being used to determine a particular class. In each class, the target class is inputted as 1 while all the other classes are inputted as 0. Then, the logistic regression model runs as usual for each classifier. In turn, a probability of a datapoint belonging to that particular classifier/target is obtained. The classifier with the highest probability for each datapoint is selected to be the predicted value of a particular datapoint.

- While computing the loss, whenever the term 0*log(0) was encountered, a NaN value was returned because even though mathematically, absolute zero multiplied with any number should be 0, python considers 0*log 0 to be indeterminate. Hence I added a small value epsilon $\varepsilon = 10^{-5}$ in the log function to prevent the indeterminate form from happening.

- I used a learning rate of 0.06 for 1300 iterations which gave me an accuracy of 85.16333%. A higher learning rate resulted in an irregular cost curve and a lower learning rate took more time.

➔ KNN:

- KNN is a distance based algorithm. It measures the distance of the test data with all points present in the training dataset. It then takes the k nearest neighbours from it and counts the number of datapoints in each category. The test datapoint, in turn, belongs to the category which has the most number of datapoints.

- We measure the distance between two points using the formula:

$$\text{Distance} = \text{sqrt}(\sum(x-x_i)^2)$$

- To measure the accuracy of the model, I took a small part of my training data and ran KNN on it. In turn, I took the value of k=4 because I got a maximum accuracy of around 90%.