This git contains the solution for the step-by-step implementation in solving the assignment for Data Science Programming course at USF.

The goal is to create an ML model capable of predicting the number of dengue cases across two cities. The work is broken down in four assignments which cover different aspects of data modeling, visualization, feature engineering and model building.

The competition and website name are: 'DengAI: Predicting Disease Spread' published at the following website. https://www.drivendata.org/competitions/44/dengai-predicting-disease-spread/data/

**Problem description:**

Your goal is to predict the total\_cases label for each (city, year, weekofyear) in the test set. There are two cities, San Juan and Iquitos, with test data for each city spanning 5 and 3 years respectively. You will make one submission that contains predictions for both cities. The data for each city have been concatenated along with a city column indicating the source: sj for San Juan and iq for Iquitos. The test set is a pure future hold-out, meaning the test data are sequential and non-overlapping with any of the training data. Throughout, missing values have been filled as NaNs.

TYou are provided the following set of information on a (year, weekofyear) timescale:

**City and date indicators**

* city – City abbreviations: sj for San Juan and iq for Iquitos
* week\_start\_date – Date given in yyyy-mm-dd format

**NOAA's GHCN daily climate data weather station measurements**

* station\_max\_temp\_c – Maximum temperature
* station\_min\_temp\_c – Minimum temperature
* station\_avg\_temp\_c – Average temperature
* station\_precip\_mm – Total precipitation
* station\_diur\_temp\_rng\_c – Diurnal temperature range

**PERSIANN satellite precipitation measurements (0.25x0.25 degree scale)**

* precipitation\_amt\_mm – Total precipitation

**NOAA's NCEP Climate Forecast System Reanalysis measurements (0.5x0.5 degree scale)**

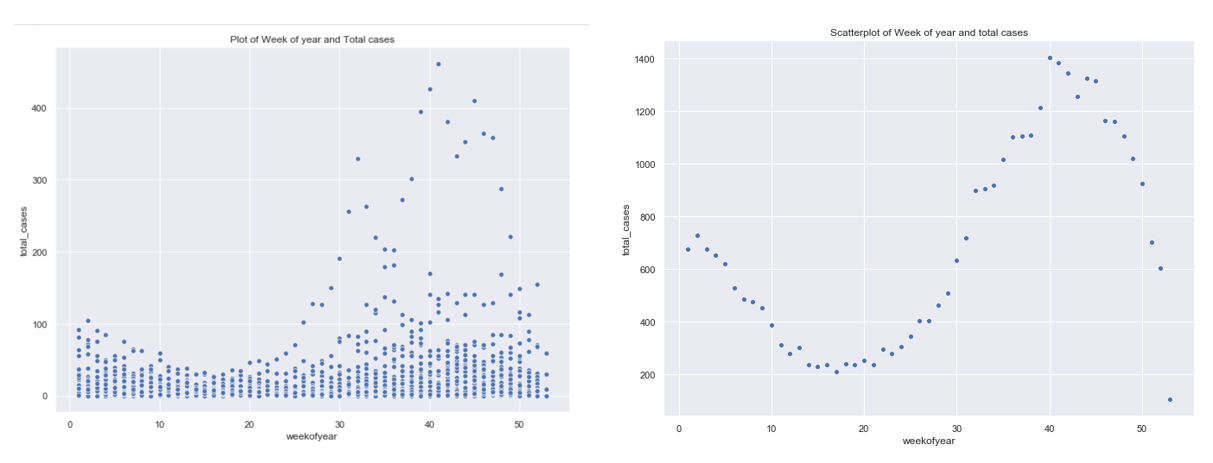
* reanalysis\_sat\_precip\_amt\_mm – Total precipitation
* reanalysis\_dew\_point\_temp\_k – Mean dew point temperature
* reanalysis\_air\_temp\_k – Mean air temperature
* reanalysis\_relative\_humidity\_percent – Mean relative humidity
* reanalysis\_specific\_humidity\_g\_per\_kg – Mean specific humidity
* reanalysis\_precip\_amt\_kg\_per\_m2 – Total precipitation
* reanalysis\_max\_air\_temp\_k – Maximum air temperature
* reanalysis\_min\_air\_temp\_k – Minimum air temperature
* reanalysis\_avg\_temp\_k – Average air temperature
* reanalysis\_tdtr\_k – Diurnal temperature range

**Satellite vegetation - Normalized difference vegetation index (NDVI) - NOAA's CDR Normalized Difference Vegetation Index (0.5x0.5 degree scale) measurements**

* ndvi\_se – Pixel southeast of city centroid
* ndvi\_sw – Pixel southwest of city centroid
* ndvi\_ne – Pixel northeast of city centroid
* ndvi\_nw – Pixel northwest of city centroid

**Assignment**

1. **Load the file 'dengue\_features\_train.csv', display the top 3 rows and observe the data. Then programmatically define the column names to make the following changes.**
2. **The predictor column (y-value) is present in the file 'dengue\_labels\_train.csv'. Read this file in a new dataframe and merge it with the above dataframe using city, year and weekofyear as join conditions.**

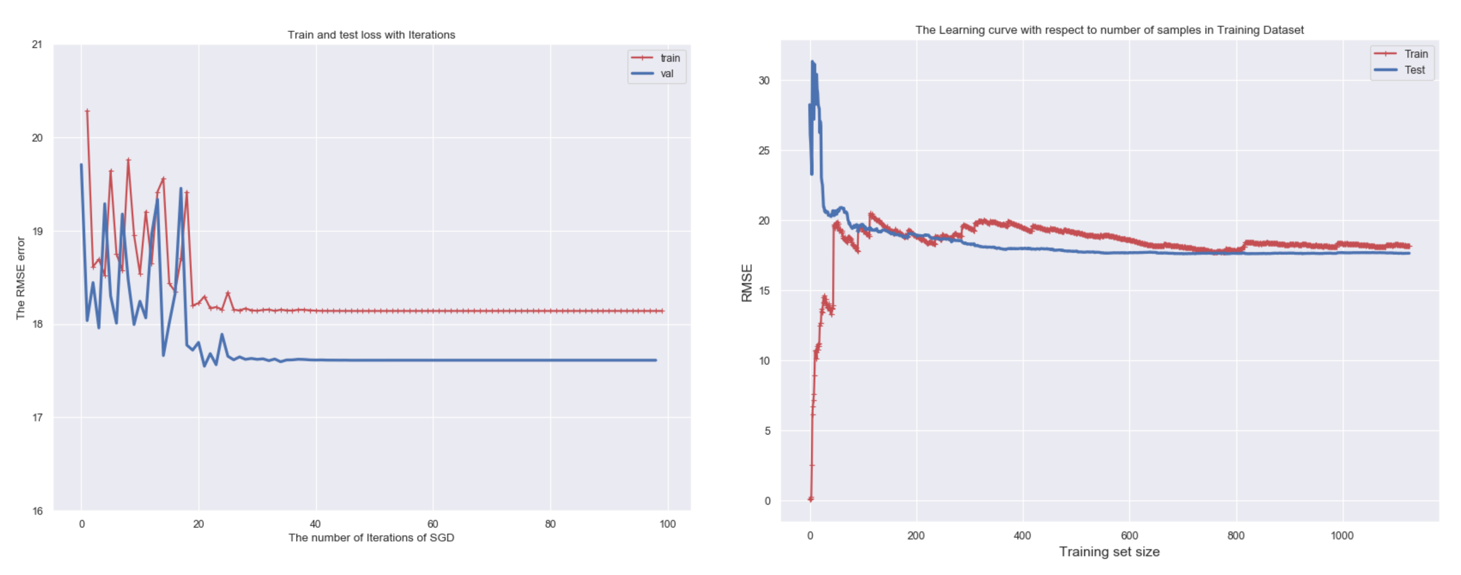


* The plot has week number on the x-axis and the total number of dengue cases on y-axis. It is clear that the number of cases have increased as the year cam to an end. Most cases were recorded between 30-50th week of the year. This looks normal and is left skewed.

1. **Preprocess the data (Encode the categorical features and Standardize the numerical features)**
2. **Build a stochastic gradient descent regressor, train the model and List the hyper-parameters that can be tuned in SGD.**

The Parameters that can be tuned for SGD are:

* Alpha - Constant that multiplies the regularization term. Defaults to 0.0001. This value penalizes the amount of regularization.
* l1\_ratio: The value of L1 regularization with respect to 1. The lower the value, the more L1 regularization in the fit
* learning\_rate - The constant that gets multiplied to the cost function.
* loss - The type of loss function used in calculating the error, in this training fit the loss is mean square error
* max\_iter- The number of iterations to update the final new co-efficient

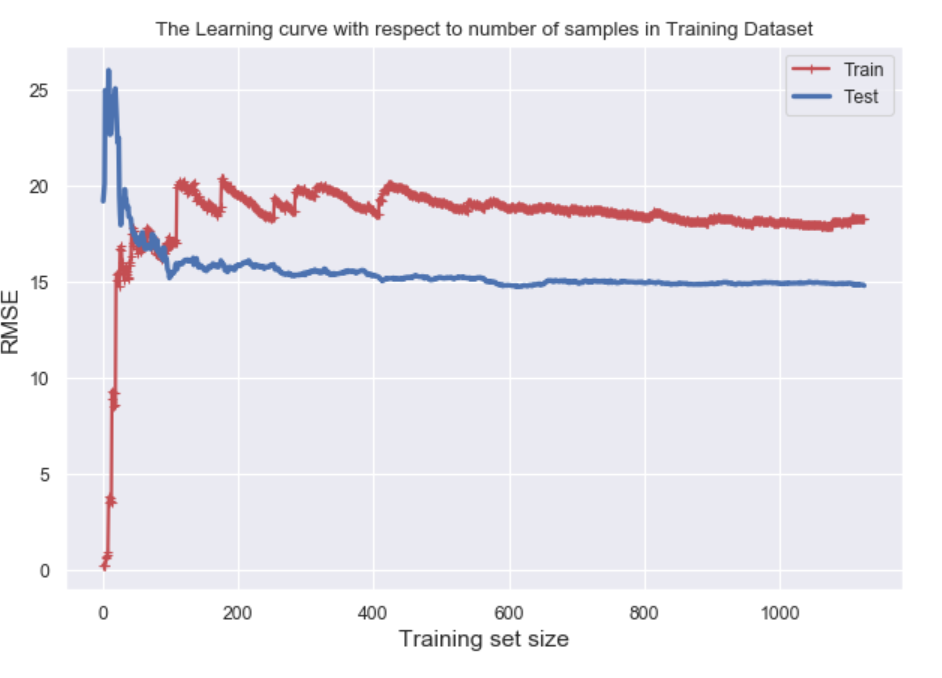
1. **Plot Learning curve and provide insights**

* We can see that error remained constant for training and test set when the number of iterations are 200. From this we can conlude that the increase in number of records in training set does nota add much value and just 200 records are enough to train the model and thus save resources.
* There is little fluctuation on the train error after 200 iterations, This might be because of the change in data and hence the model not able to predict well. Overall the data looks consistent and this looks like a generalized model, because the train error is not going down or up with increase in rows. Also, upto 40 iterations are enough to get to a steady train and test error.

1. **List which features you will choose in this model. Select the required columns in the dataframe and drop the others.**

* When two attributes give information about the same thing then they are duplicates and these might trick our model in terms of rule generation. As the two attributes have same data, the rules generated will be same thus, there wont be any use of having such duplicate columns
* So, looking at the variables, I am dropping the columns which have high corelation (0.9 >). These variables are :
* re\_an\_max\_air\_temp\_k - This is intercorelated with re\_an\_tdtr\_k. Hence I am dropping this as re\_an\_tdtr\_k is the difference between max and min temparature of the day
* re\_an\_dew\_point\_temp\_k - This is corelated with re\_an\_specific\_hd\_g\_per\_kg. This says that the pressure is highly dependent on temparature. Hence droppping the pressure

1. **Build a Linear SVR regressor, train the model and evaluate on a metric, list hyper-parameters and plot learning curve.**

* I chose RMSE error (Root Mean Square Error). It is measure on how far from the regression line the data points are. THe RMSE error gives low weightage to points around the best fit line and high weightage to points farther from the line. It basically tells us how concentrated the data is arounf the line of best fit. The high error on train and low on test says that the model is under fitting and we can increase its complexity.
* The Linear SVR is similar to the Traditional SVR but it supports only linear kernel and is implemented in terms of liblinear rather than libsvm.
* There are two main parameters to tune in the Linear version of SVR:
  + The choice of penalties - The penality is represented by C. This is the amount of influence slack variables has on the function. Slack variable is the non-linear points which are tough to classify. Hence, we will ignore these points. The C is the way of saying how much to ignore. With high C value the models puts penalities on the slack points and hence creates an over-complicated one.
  + Loss functions- The Lasso (L1) loss function is the standard way of entering the value in Linear SVR. By setting loss as epsilon insensitive we are saying it to start with L1 regularization and with epsilon of 0.3 I got least error
* From the learning curve we can see that the error in train and test samples has stabilized after 150 samples. But as this is a linear kernel, it was not able to reduce the error further and with increase in the sample size the model did not learn much

1. **Build a SVR model with Linear Kernel, train the model, evaluate and print the tuning parameters and plot learning curve to provide insights**

* I chose RMSE error (Root Mean Square Error). It is measure on how far from the regression line the data points are. THe RMSE error gives low weightage to points around the best fit line and high weightage to points farther from the line. It basically tells us how concentrated the data is arounf the line of best fit
* There are mainly Three parameters that can be tweaked to improve the performance of model
  + C- Regularization Parameter - The Cost function on the slack variable. This controls the train error and test error relationship. To have a highly fitted model C value should be high
  + gamma- The kernel co-efficient for Non-linear functions. The degree to which we can use the kernel, it is used to set the amount of N dimensions we can throw data and a complexity manager for kernel
  + Kernel - The type of Kernel for creating the hyperplane. This is the kernel matrix or kernel trick. There are different kernels A linear kernel throws the data in N dimensions and plots a linear curve, The Polynomial kernel increases the complexity and plots a curved hyperplane, Radial and hyperbolic ca learn any complex patterns.
  + The Learning curve of Linear SVM with kernel is similar to the linear SVR, as these two are build from the same concepts. It is expected to have same output. The only difference I noticed is the time taken to perform training. The Linear SVR took very less time when comoared to the SVR with Kernel. Apart from it the information is same as above. Stabilizing after 150 samples



1. **Add a new column called 'above\_average' with value 1 or 0. 1 if the total\_cases > median of total\_case and create an MLP classifier and explain the meaning of Precision, Recall and F1-Score and why these are used to evaluate Classification models (instead of using Accuracy as a metric). Evaluate the classifier using Precision, Recall and F1 score values**
   1. As the dataset is balanced we can rely on the accuracy as metric for evaluation.
   2. Along with that I believe False Negative score(Recall) is important. Because saying a city as having less number of cases is more dangerous than predicting a city as having more number of cases
   3. Cause I think if the model predicts correctly about the city having high number of cases then the medical team can be prepare well. But if the city having high number of cases is predicted by model as less than average then the mdical team might not take that city seriously and hence the people in that city might affect badly

**Case 1:**

Let's understand the metrics in terms of business context. Suppose you are owner of ferrari company and you are manufacturing limited edition super car. The head of marketing department has 10,000 customer details who they think to advertise. You have created a model which predicts whether a customer will buy the car or not. According to the model you will advertise to only those which the model tells as buyers. So in this case your model can do two mistakes

* Precision: Predicts non-buyer as buyer this is false positive (falsely predicting that the customer will buy)
* Recall : Predicts buyer as non-buyer this is false negative (falsely predicting that the customer will not buy)

Now which metric do you think is important? For this case,If model predicts a non-buyer as buyer then company will loose small amount by advertising to non-buyer and the amount they spent on advertising for that person will be low (at most 50$)..this is precision (falsely predicted as positive).. But on the other side of coin, If model predicts a buyer as non-buyer then the company is not going to advertise the car to that buyer and at the end the company is going to loose that customer who had the potential to buy that car. This is recall (falsely predicted as negative)..

So in this case the recall is the metric to optimize..

**Case 2:**

Let's put you in another shoe..You are manager of a branch and there are 4000 loan applications. You have created a model which predicts whether an applicant can be granted loan or not.. So in this case your model can do two mistakes

* Precision: Predicts non-eligible applicants as eligible this is false positive (falsely predicting that the applicant can be granted loan)
* Recall: Predicts eligible applicants as non-eligible this is false negative (falsely predicting that the loan application should be rejectet.)

**F1 Score**

F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. Intuitively it is not as easy to understand as accuracy, but F1 is usually more useful than accuracy, especially if you have an uneven class distribution. Accuracy works best if false positives and false negatives have similar cost. If the cost of false positives and false negatives are very different, it’s better to look at both Precision and Recall.