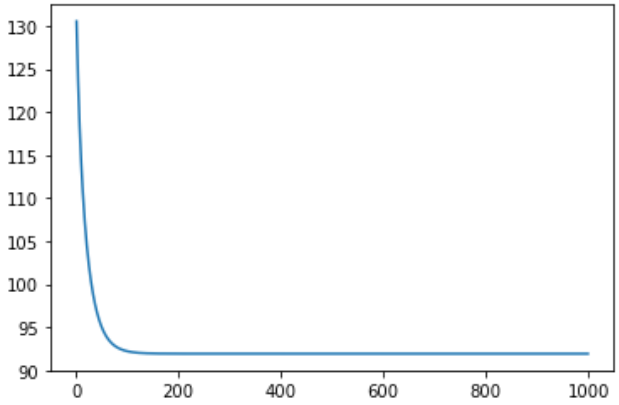
Program Description:

The program implements logistic regression along with gradient decent algorithm on synthetic data and the real-world data. We have calculated the error rate of test and training sets in different circumstances like adding the nominal variable's and concluded the performance of the model. Finally, we have split the data into ten splits and train the model in a loop ten times to evaluate the training testing error and the bias variance trade-off is discussed.

Q1 - Synthetic data

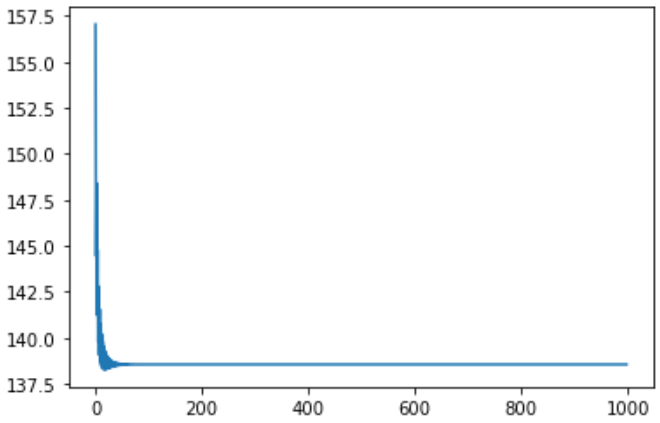


Plot 1 for synthetic data ℓ(D, λ(t)) versus T*epochs*

- Above is the plot ℓ(D, λ(t)), versus t, t = 1, . . . , T*epochs*,. As mentioned the number of epochs be Tepochs are set to 1000, the learning rate, η = 0.01, and the algorithm is initialized with λ (0) ∈ R d+1, λ(0) i = (−1)**i , i = 0, 1, . . . , d. As seen in the plot, the value of the objective function decreases as the increase in the no. of iterations i.e T*epochs*. The main role of the hyperparameters here is to find the minimum value for the function f(X, λ).
- Here the closer we get to the optimal value of intercept, closer the slope gets to zero. This means we should decrease the step size as we move closer to the optimal value.
- Gradient decent determines the step size by multiplying it to the learning rate ***η*** to get the new value of intercept.
- The algorithm has to stop when the step size is closer to zero. We can achieve that by applying a limit in the program for which the step size and the slope attains minima.
- This is when T*epoches* comes into picture, here we are controlling the algorithm that how many times it shall repeat/iterate through the process to get the optimum value. Generally, the value of number of iterations i.e T*epoches* is considered as 1000 or more.
- As mentioned, with the help of the learning rate ***η***, we are determining the step size of our algorithm i.e at what rate our step should decent to reach the minimum value. Here if the learning rate ***η*** is too large and the value of the iteration T*epoches* is too small, the function will reach to the minimum value faster than expected thus making the plot steeper towards minimum and might be skipping the exact value where the function should attain its minima.

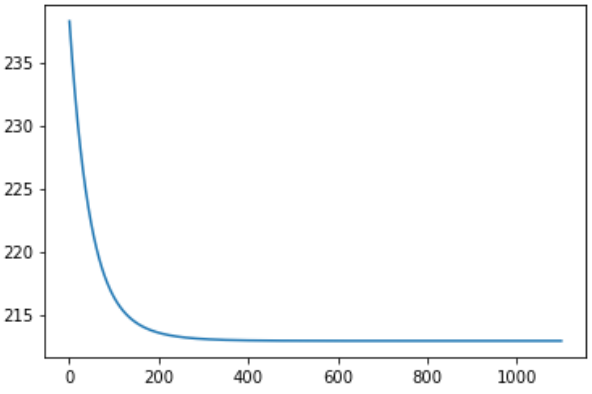
Q2 - Real-world data

- Real world data “auto.txt” has been imported in the program and a binary variable has been created for prediction based on a condition, if data[‘mpg’] >= 23 then 1 otherwise 0.
- Here the algorithm is initialized with Gaussian-distributed numbers with mean 0 and standard deviation ≈ 0.7.
  The data is split into two datasets viz. d_train and d_test, while d_train set is considered for gradient decent and to train the logistic regression model f(X, λ).
- The values of the hyperparameters $\boldsymbol{\eta}$ and T*epoches* has been change from what was given in the synthetic data.
- For $\boldsymbol{\eta}$ = 0.01 and T*epoches* = 1000, it was observed that these values for hyperparameters was too large, thus the plot is directly descending to 0 and remaining constant from there, as shown in the below plot.



Plot 2 for real world data ℓ(D, λ(t)) versus T*epochs* for $\boldsymbol{\eta}$ = 0.01 and T*epochs* = 1000

- As seen in the above plot, the algorithm overshoots and hence skipping the points where the function can attain the minima.
- After tuning for the hypermeters i.e after checking for multiple values it was observed that for $\boldsymbol{\eta}$ = 0.0001 and T*epoches* = 1100 we are getting the optimal values, as seen in the below plot.



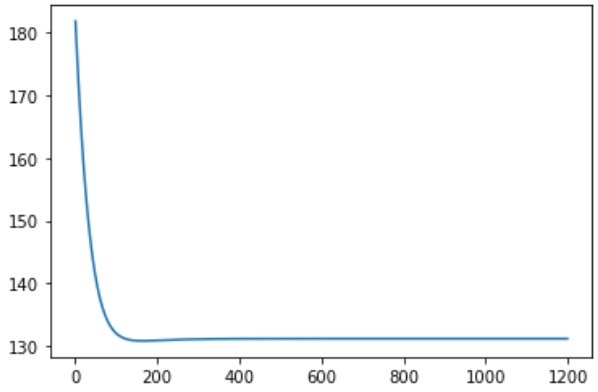
o

Plot 3 for real world data ℓ(D, λ(t)) versus T*epochs* for $\boldsymbol{\eta}$ = 0.0001 and T*epoches* = 1100

- Here, we are expecting ℓ(λ (t+1) , d_train) ≤ ℓ(λ (t) , d_train) as we are descending towards minima and trying to find the best fit or optimum value for σ.

Q3 - Nominal variables

- To complete this task a set of dummy variables are created for data['origin'] column, thus increasing the number of features.
- d_train and d_test is being treated as augmented version of the previous sets and the model has been trained.
- In this case, the hyperparameters are tuned for the values for $\boldsymbol{\eta}$ = 0.0001 and T*epoches* = 1200 making the algorithm converge as shown in the plot below:



Plot 4 for augmented data ℓ(D, λ(t)) versus T*epochs* for $\boldsymbol{\eta}$ = 0.0001 and T*epoches* = 1200

- To check the performance of the real world data with respect to the augmented data that we have created, we have used the below mentioned formula :
  - ER(λ, D) = (X (x,y)∈D (y − yˆ) **2) / |D| , yˆ = 1 if : (f(x, λ) > 0.5)
- By calculating the error rate multiple times and after training the values for random initialisations, it was observed that by increasing the number of features the error rate decreases.
- Hence, from the results obtained we can say that the by adding a nominal variable the performance of the model increases slightly as the error rate for the original data set is ER(λ, d_test) = 0.1504 while after adding the nominal variable the error rate becomes ER(λ, d_test) = 0.0321.
- To confirm above, also compared the error rates for training sets for both original set, as well as for augmented sets.
- ER for original set is ER(λ, d_train) = 0.1550 and for augmented set is ER(λ, d_train) = 0.0256, from these obtained results we can say that a better performance on training set can results in better performance in the test sets.

Q4 - Training and testing error

- In this section we have split the data into 10 parts and split it for d_train and d_test and train the model on d_train for i = 1,2….10., keeping the η and T*epoches* same as of section 2 (for real world data).
- The error rate for each iteration is given in the following table.

Sr No.	ER(λ∗i , d_train,i)	ER(λ∗i , d_train,i)
1	0.1165846959	0.1077313718
2	0.1218159838	0.1304493983
3	0.1071637626	0.1173343924
4	0.1526280297	0.1365524274
5	0.1028277286	0.1324739012
6	0.1184861864	0.1051918415
7	0.1472785368	0.06727214485
8	0.03909773357	0.09032613527
9	0.09617921247	0.09937796534
10	0.07067431223	0.1339950612

- From the above obtained results, we can say that for no pair of train and test error rates are matching but for pairs 1,4,6 and 7 shows that the testing data is performing well than the training data.
- Generally, when a model underfits its can be considered as *high biased* conversely when a model overfits it can be considered as *high variance*.
- In this scenario we have to tune the hyperparameters in such a way that model finds a best fit solution, or in other words, be optimal. This can be achieved by *Bias_variance trade-off.*
- From the obtained results, as the error rate on the training data is too low which means the accuracy is too high and by looking at the numbers it can be considered as ≈ 100, hence it is safe to say that the model is overfitting.
- But, if considering both the data sets i.e train and test data the error rate in both the cases is too low then the accuracy can be considered as too high. A model is considered as overfit if it is achieving highest accuracy in the training data but not preforms very well in the test data.
- In this case from the obtain results we are saying that the data is overfitting in training but after comparing both the sets i.r train and test sets its difficult to say if its overfits or a best fit optimal solution.
- Which ever the case is, we may have to tune the hyper parameter and consider the bias variance trade-off design to train the model.

References:

- Learning Rate in Gradient Descent — what could possibly go wrong | by ramsane | Medium
- Week 4 lab worksheet
- Week 4 slides
- How to Calculate the Bias-Variance Trade-off with Python (machinelearningmastery.com)