Project Report

Executive Summary

This project aims at the implementation of gradient descent algorithm for Linear Regression and Logistic Regression incorporating multiple parameters such as Learning rate (alpha), threshold, number of iterations etc. Based on the implementation, various experiments are performed to understand the relationship between the parameters and the training and test cost/accuracy. Conclusion and discussion are provided at the end of the implementation and experiments. The entire project is implemented using 'Python 3' (Jupyter Notebook).

Tasks

Task 1 – Data pre-processing and partitioning

The dataset is downloaded and is visualized for initial pre-processing. The dataset has 29 columns and 19735 rows with 1 target variable – 'Appliances' and 28 dependent variables.
Various columns are analysed for missing values and unusual number of zeroes. The 'lights' column has 15252 zeroes out of 19735. Hence, that column is dropped. Also, a correlation matrix is plotted and visualized using a heatmap for all the independent variables. The features – 'T6', 'T9', and 'rv' are dropped on the basis of high correlation. Hence, going forward, the data is cleaned and has a shape of '19735 X 22' (22 features are retained for analysis and experimentation).

Task 2 – Linear Regression Model Design

A linear regression model is designed as a function to model the energy usage of appliances. The function is designed in such a way that it has input parameters – β_0, β and the dataset. The function implements the model using the input parameters and returns the output data. The regression equation is given below:

```
# Linear Regression Model Function
def linear_regression_model(x, beta_0, beta):
    y_hat = (x @ beta) + beta_0
    return y_hat
```

Task 3 – Gradient Descent Algorithm (batch update)

The gradient descent algorithm is implemented with batch update rule. The sum of squared error cost function used in the class is used as the basis for computing the error/cost for all the operations in gradient descent algorithm. The initial beta values are reported below:

β_0 = 0.5
β = [0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5]

The gradient descent algorithm is implemented keeping in mind the various control parameters. All the parameters are taken as an input to the algorithm and the final β values, cost and No. of iterations are returned back for further analysis and processing.

Task 4 – Conversion to binary classification – Logistic Regression

The target variable – 'Appliances' is transformed to a binary variable using 60 as the median value for conversion. Utilizing this new transformation, we now have a binary classification problem which can be solved by the Logistic Regression model. The model is implemented using SGDClassifier package in scikit-learn library (linear_model). The model accepts a control parameter – learning rate, `*alpha*` as part of the implementation. Implementing the model, the accuracy is reported below:

Training Accuracy	0.720
Test Accuracy	0.724

Experimentation

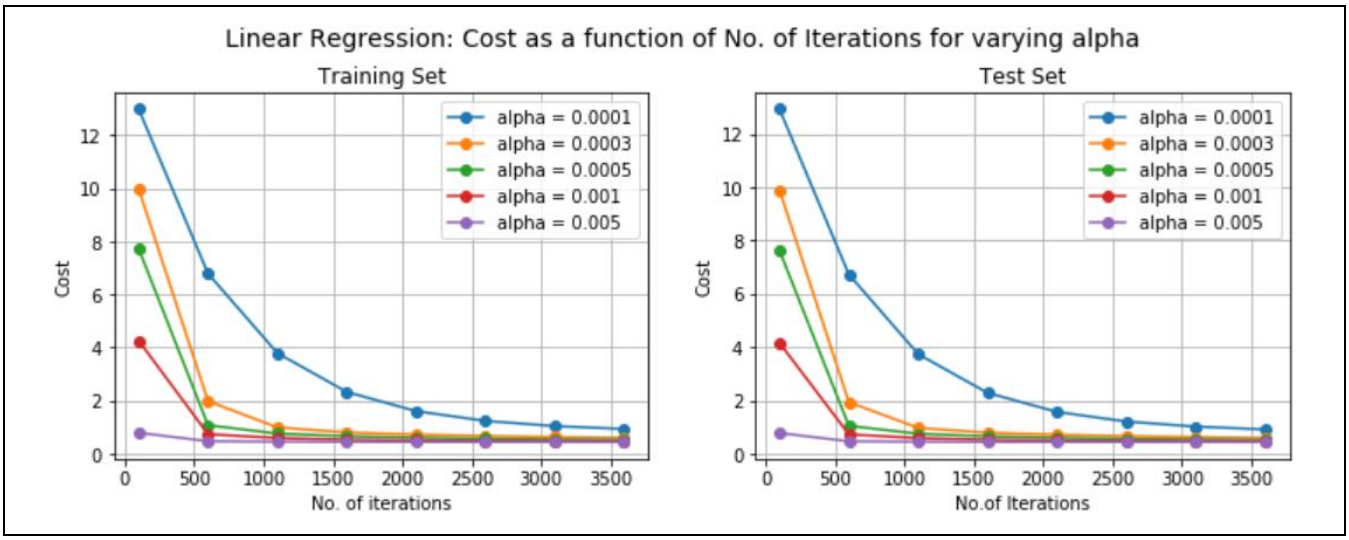
For all the experiments that are performed below, the initial β values are always 0.5. For each experiment, certain parameters are kept constant and some parameters are changed to decipher the relationship between COST and control parameters. The inferences are realized using plots and tables. The control parameters in research are explicitly mentioned for each experiment. Also, for both linear regression and logistic regression, in all the experiments, the data is scaled prior to analysis.

Experiment 1 – Varying control parameters for linear and logistic regression from a Cost perspective

Linear Regression: In this experiment, the *threshold* is fixed at a value of 0.00001. The reason such a value is chosen because for this value of *threshold*, we have the least cost. The reason is more profoundly visualized in Experiment 2. This experiment is performed to understand the relationship between Cost, No. of iterations and *alpha*. I have extracted the cost values for various value of no. of iterations and for varying *alpha* values.

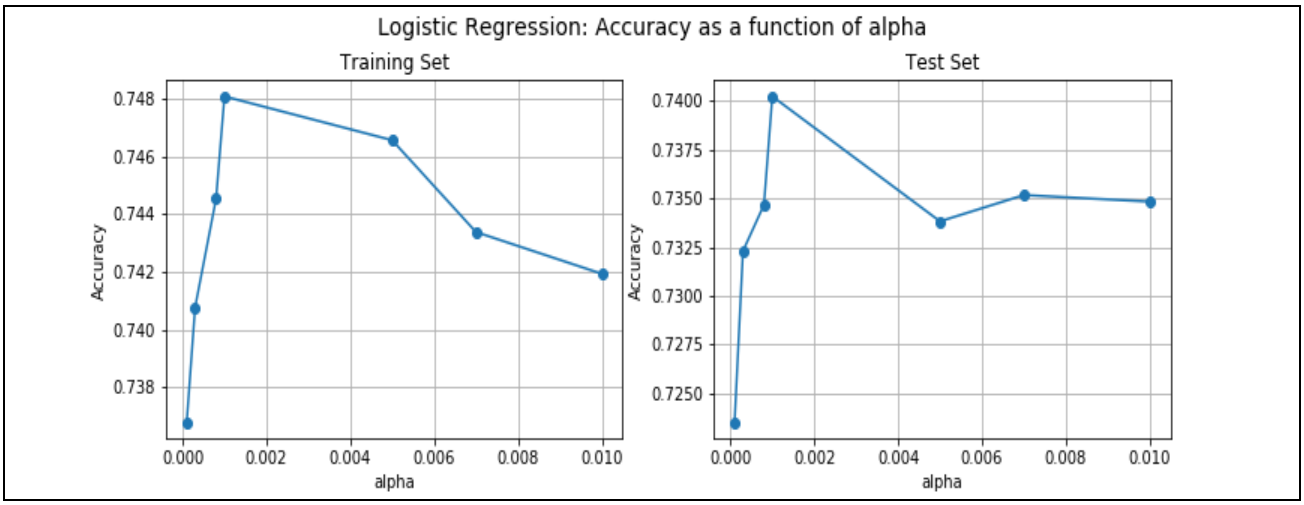
I have chosen a range of 100 to 4100 in steps of 500 for the 'No. of iterations', because I have experimented for *threshold* in Experiment 2 and it was inferred that for the least value of *threshold* taken at 0.00001, the COST converged at around 3600 iterations. Hence, that is my maximum limit. And, for the *alpha* values, I have chosen an array set of – [0.0001, 0.0003, 0.0005, 0.001, 0.005]. The prime reason for choosing this range is because for values lesser than 0.0001, the COST is high, almost equal to initial cost and for values higher than 0.005, the COST converges very quickly at <500 iterations, thereby denying any possible visualization.

The resultant plots are reported below:



We can infer from the above plot that, as the 'No. of iterations' increase the COST decreases. Also, we can see that the convergence gets steeper/faster for increasing values of *alpha* also indicated by the lesser number of iterations for increasing values of *alpha*.

Logistic Regression: For logistic regression, this experiment is performed with varying *alpha* values only, as it is implemented through a package. I have used the almost the same range of alpha values for the reason mentioned earlier and I have also included extra few values to provide more visualization. Implementing the model for varying *alpha* values, the resultant plots for training and test accuracy are reported below:



By looking at the plots, we can infer that the training and test accuracy spike at *alpha* = 0.001, recording the best accuracy value in the whole plot. The accuracies are low for low *alpha* values and rise exponentially for increasing values of *alpha.* After spiking at *alpha* = 0.001, the training and testing accuracy values decrease gradually for increasing values of *alpha.*

The best parameter values are obtained from this experiment and are reported below:

Linear Regression (*threshold = 0.00001)*	*alpha = 0.0001, No. of iterations = 3600* (least cost)
Logistic Regression	*alpha = 0.001* (max accuracy)

Experiment 2 – Varying thresholds for convergence for linear regression

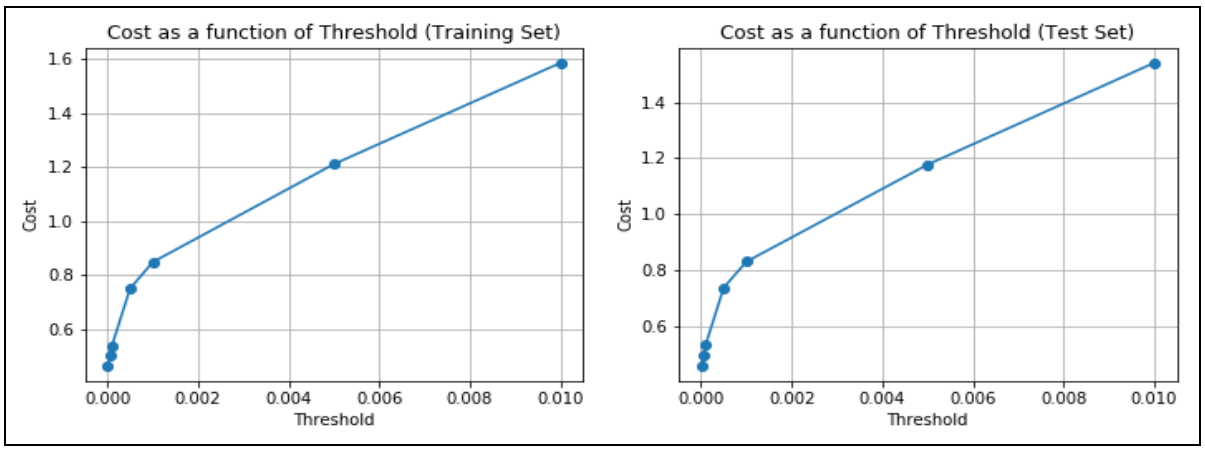
In this experiment, we try to establish the relationship between threshold and convergence for linear regression. I have inferred threshold as the parameter that sets an ideal difference value between successive costs as the stopping point or 'convergence' i.e. successive costs should differ by at least the 'threshold' value while running in the gradient descent algorithm, else we can safely say that convergence is reached.

In this experiment, I have chosen the range of values for *threshold* as given below:

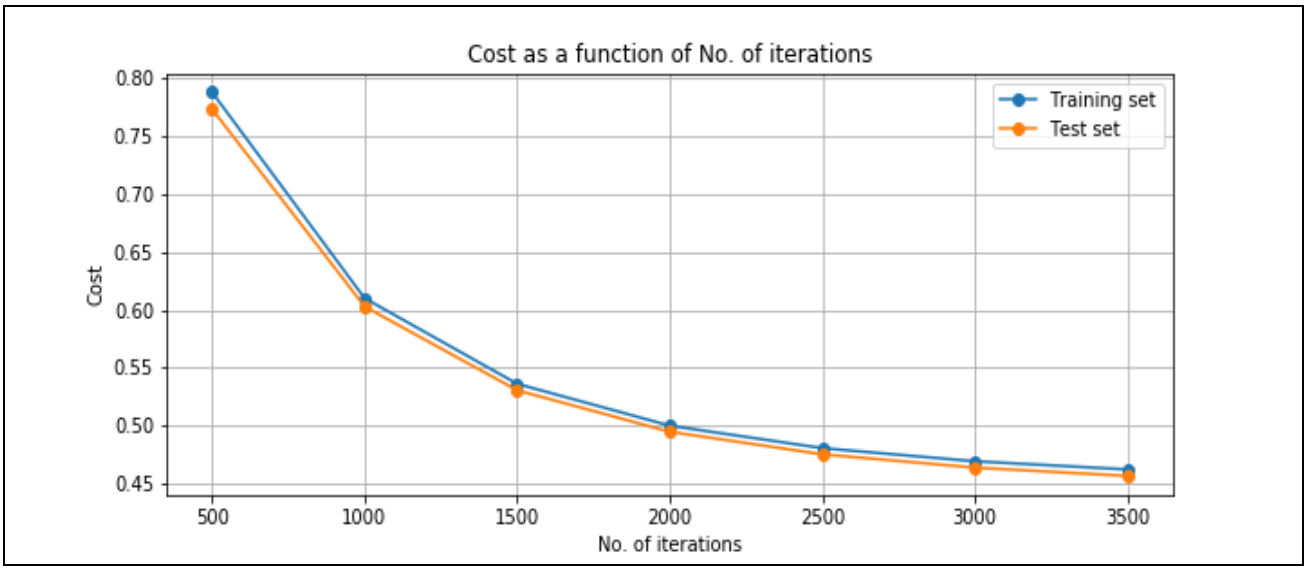
thresh_set = [0.00001, 0.00005, 0.0001, 0.0005, 0.001, 0.005, 0.01]

I have chosen such a set because for values lesser than 0.00001, the convergence takes very long and for values greater than 0.01 the convergence happens in <500 iterations. The maximum number of iterations is set fixed at 100,000 (to be safe and accommodate all *threshold* values). Implementing the gradient descent algorithm for varying threshold values, the below plot is obtained:

We can infer from the below plot that as *threshold* increases the COST also increases. COST here represents the cost at which convergence is reached. The plot is similar for both the Training and Testing sets. We have the least cost at *threshold* = 0.00001. Hence, the best *threshold* chosen is 0.00001.



Proceeding with the experiment, after choosing the best *threshold*, the train and test errors are plotted as a function of gradient descent iterations. The plot is reported below:



It can be inferred from the above plot that the test set is recording a lower cost for all values of *No. of iterations* as compared to the training set. In general, as the *No. of iterations* increase the COST decreases which is expected as the Cost function is convex, and we are approaching the global minimum as gradient descent algorithm iterates.

Experiment 3 – Exploring Cost for a random set of 10 features for linear and logistic regression

In this experiment, 10 features are randomly selected, and their train and test errors are computed by running the gradient descent algorithm. The values of *alpha* and *threshold* are the best-chosen values which are obtained from previous experiments. The initial β values are 0.5 as always.

The 10 random features are chosen by the `random()` package in Python. This package produces a different output in every iteration. Hence, the results obtained in a future iteration may not be consistent with the results presented here. But the behaviour and trends are more or less preserved and remain intact. The resultant dataset is fitted with both linear regression and logistic regression models. The final results are tabulated and reported below (with original set's train and test metrics):

Linear Regression	Cost (Training Set)	Cost (Test Set)
Original Set (22 features)	0.4611	0.4565
Random set (10 features)	0.4937	0.489

Logistic Regression	Training Accuracy	Test Accuracy
Original Set (22 features)	0.7202	0.724
Random set (10 features)	0.6247	0.6045

From the above table, it can be inferred that for Linear Regression and Logistic Regression, the dataset with randomly selected features doesn't perform as good as the original dataset with 22 features. The COST of the random set is higher than the original set for both the Training and Test sets. Similarly, in Logistic Regression, the Train and Test accuracy is lower for the random set as compared to the original. Looks like not all the information is captured by the random set and hence, the models aren't able to predict better.

The 10 random features that were chosen are reported below:

```
['T1', 'RH_2', 'T3', 'RH_5', 'T7', 'T8', 'RH_9', 'T_out', 'Windspeed',
'Visibility']
```

Experiment 4 – Exploring Cost for a "chosen" set of 10 features for linear and logistic regression

In this experiment, I am choosing the 10 best features of my choice and performing Linear Regression and Logistic Regression on the resultant dataset. The training and test metrics are obtained and are tabulated to serve as a comparison medium for inference and interpretation of the obtained results.

In my opinion, the 10 best features are reported below:

```
[ 'RH_1', 'RH_2', 'RH_3', 'RH_4', 'RH_5', 'RH_6', 'RH_7','RH_8', 'RH_9',
'RH_out']
```

After performing the Linear and Logistic Regression, the train and test metrics are captured and tabulated. The table is reported below, with all the values from original set, random set and chosen set:

Linear Regression	Cost (Training Set)	Cost (Test Set)
Original Set (22 features)	0.4611	0.4556
Random set (10 features)	0.4873	0.4813
My set (10 features)	0.4674	0.4603

Logistic Regression	Training Accuracy	Test Accuracy
Original Set (22 features)	0.7202	0.724
Random set (10 features)	0.6679	0.6511
My set (10 features)	0.7039	0.7007

Looking at the table, we can make the following inferences-

1. **My choice of features has** provided better results when compared to the **random set** as is evident by **random set's** higher training and test costs for Linear Regression and lower training and test accuracy for Logistic Regression.

   The main reason that my chosen set of features has provided better results can be inferred as that, the chosen features have captured better variance of the dataset as compared to the random set. The most important variables which exhibit variance are present in the chosen set and not in the random set. Hence, this has resulted in a better model which better predictive power and better train and test metrics.

2. **My choice of features has not** provided better results when compared to the **original set** of 22 features as is evident by **original set's** lower training and test costs for Linear Regression and higher training and test accuracy for Logistic Regression.

   Though the chosen set performed better than the random set, it did not perform as good as the original set. In my opinion, this is because the predictive power of the original set is better because it includes 22 features, thereby capturing almost all the variance in the dataset which would boost the resultant model's predictive power. The chosen set with 10 features is limited by the number of features selection and hence, as a result cannot capture the entire variance present in the dataset as good as the original dataset, thereby resulting in a poorer model.

Conclusion / Discussion

This section summarises and discusses the interpretation of the results obtained as part of the experimentation. Some of the key inferences are:

- **Linear Regression:** While implementing gradient descent for Linear Regression, we realize that the cost function is indeed a convex function with a global minimum. The COST keeps decreasing as the No. of iteration keeps increasing. The control parameter *threshold* determines when the gradient descent algorithm should stop iterating, thus establishing the condition of convergence. As *threshold* increases, the *No. of iterations* decreases as the convergence condition is met earlier. Also, the control parameter *alpha* determines the step size for updating β values, which indirectly determines the rate of convergence of the cost values. There can be a situation when the global minimum is not met at all because of a high *alpha* value. As *alpha* increases, the rate of convergence increases.

- **Logistic Regression**: In logistic regression, we have only one control parameter, *alpha.* As *alpha* increases, we saw that accuracy increased rapidly to spike at a point and then decrease gradually as *alpha* increased. There is no definitive relationship between *alpha* and Accuracy.

- Zooming out and approaching from overall perspective of the project implementation, it can be inferred from Experiment 3 & 4 that, to model/predict the energy usage, it is best to include as many features as possible to increase the predictive power of the resultant model.

- Having inferred the above point, we should also be mindful that a careful selection of "important" features can give a model that is as good as the model with all the features. In experiment 4, my choice of features included only 'Humidity' values at various locations in the house and outside the house. I felt that, people tend to use Air Conditioner more as the Humidity in the air increases. Although temperature also is a good candidate for selection, it is 'Humidity' which triggers the reaction. And, as expected, the model built out of the chosen features was able to predict as comparable to the original set with 22 features.

  Hence, in experiment 4, although the chosen set had a lesser accuracy than the original set, it was highly comparable. A model with 10 features does have the capability to produce a model comparable to a model with 22 features. Hence, it is important to choose the features of importance before building a model.

- Therefore, after experimentation, what matters the most in predicting energy usage, in my opinion, is 'Humidity' values. After which we can consider 'Temperature' at select areas and other weather parameters. Having performed the experimentation, I would list a good model built with 10 features as:

$$\text{Energy usage} = \beta_0 + \beta_1(RH\_1) + \beta_2(RH\_2) + \beta_3(RH\_3) + \beta_4(RH\_4) + \beta_5(RH\_5) + \beta_6(RH\_6) + \beta_7(RH\_7) + \beta_8(RH\_8) + \beta_9(RH\_9) + \beta_{10}(RH\_out)$$

  where the variables are described as below:

  RH_1, Humidity in kitchen area, in %
  RH_2, Humidity in living room area, in %
  RH_3, Humidity in laundry room area, in %

RH_4, Humidity in office room, in %
RH_5, Humidity in bathroom, in %
RH_6, Humidity outside the building (north side), in %
RH_7, Humidity in ironing room, in %
RH_8, Humidity in teenager room 2, in %
RH_9, Humidity in parents room, in %
RH_out, Humidity outside, in %

- In regard to modelling to get better results, I would prefer doing Principal Component Analysis. The reason behind this is, instead of going by intuition or randomness, we can let the algorithm determine which features are the best. By doing PCA, we not only shrink the number of features but also, at the same time capture all the variances present in all the features into a very minimal set of variables. I feel that it would be the best approach to get a better model with higher accuracy and lower costs.
  One more approach could be to find the feature importance for all the features to determine which of the variables are important (possessing high variance). This could help us in zeroing down the features that are needed for the model.