Implementing Logistic Regression Model

Alok Patel Student ID : 260954024
Aishwarya Ramamurthy Student ID : 260963956
Katyayani Prakash Student ID : 260964511

Abstract

In this project, we investigated the performance of linear classification models on two benchmark datasets. We explored the effect of various parameters such as learning rate, epochs (number of iterations), number of features, and order of model on the overall accuracy of the model. We found that logistic regression approach achieved best accuracy when we optimized our learning rate and epochs, selected the best subset of features (by dropping redundant or non-uniform features), and increased the order of highly "important" features. We also found that better accuracy can be achieved if our training data has uniform class distribution, and higher number of features and observations. Lastly, we tested the performance of our model against Python's standard Logistic Regression & Naive Bayes classifiers.

1 Introduction

In this project, we have implemented the linear classifier - Logistic Regression - from scratch using Python programming language, developed on Google Colaboratory. Our model has been trained over 2 given datasets. Our goal was to correctly classify both datasets in binary classification manner into Classes 0 and 1.

We have applied a 5-step validation approach (each step consisting of extensive experiments) to arrive at the best model for both our training sets. We then compared our model's results with that of Python's SciKit-Learn based models of Logistic Regression and Naive Bayes. Our model's accuracy for both the datasets was comparable to the accuracy achieved by these standard models.

2 Overall Project Approach

The implementation of our overall project can be divided into 3 major parts :

2.1 Data Analysis

"The performance of a machine learning algorithm is only as good as the data used to train it" [1]. Thus, we attempted better comprehensibility of our raw data by performing: 1) Descriptive Analysis – determining the shape, class distribution, correlation, mean, standard deviation etc. of training data 2) Visual Analysis – plotting histograms, density plots and correlation matrices of datasets 3) Preprocessing Data – Normalising and Standardizing our data in hopes of achieving better accuracy.

All data obtained from this analysis has been attached in the Appendix A.

2.2 Building the logistic regression model

Once a thorough understanding, and consequent cleaning of raw data was done, the next step was to build a logistic regression model that would be used to train over this data. This was done with the help of classes and methods. Please refer Appendix B & C for more details on this.

Submitted as part of coursework for ECSE 551, Winter 2021. Do not distribute.

2.3 Validation and Experimentation

The last but perhaps the most crucial segment of our project entailed a series of experiments carried out to determine the highest possible accuracy that could be achieved for both the given datasets. This was achieved by :

1. Selection of optimal learning rate and epochs
2. Comparison of given data vs preprocessed data
3. Feature Selection
4. Model Selection
5. 10-Fold Cross Validation
6. Comparing our results with SciKit-learn based Logistic Regression & Naive Bayes models

3 Datasets

Orthopedic Patients dataset consists of 310 training examples (patients) characterized by 6 biomechanical features. Target (Output) variable "Class" has two labels : Abnormal (Class 0 - relevant to patients diagnosed with orthopedic disorders) and Normal (Class 1). Class 0 constitutes $\sim$68% (211/310) while Class 1 is only $\sim$32% (99/310) of the total instances. This skewed class distribution may affect the overall accuracy of our training model. Interestingly, highest correlation is observed between pelvic incidence and sacral slope in this dataset.

The second dataset comprises of 991 credit card transactions done by European card holders over two days. There are 28 features (numerical variables V1 - V28; nearly uniform distribution) obtained as a result of Principal Component Analysis (PCA) transformation and a non-transformed feature 'Amount' denoting the transaction amount. Class 0 ($\sim$49.65% of instances) represents a fraudulent transaction and Class 1 ($\sim$50.35% of instances) represents fair; class distribution is very uniform for this dataset which might contribute to better accuracy. The histogram visualization of both datasets (attached in Appendix A) show notable right and left-skewed data distributions.

Certain features from both the datasets have either been dropped or undergone basis expansions (squared or cubed) so as to improve the model fit and test for optimal accuracy.

4 Results

As stated in section 2, we divided our experiments in 5 segments, each segment helping us arrive at the most optimal parameter for our models (Segment 6 is for comparing our model with standard models). In this section, we state the results of these series of experiments. The best output of each experiment is highlighted in bold.

4.1 Selection of optimal learning rate and epochs

The first step was determining the most optimal learning rate which will yield us maximum accuracy. For this, we wrote a function wherein for a given set of features, target, number of iterations (epochs), and range of learning rates, a logistic regression model is fit and predicted. Accuracy is calculated for each element of the learning rate array and stored as a vector. The highest yielding accuracy element of the vector is printed as output. Finally, the accuracy and learning rate vectors are used to generate the required plot.

After finding the best learning rate, the next logical step was to test this against various epochs and see its dependence on accuracy. The below table summarizes our experimental results from the same.

In case of orthopedic patients, we reached our maximum accuracy at 500 epochs and while the accuracy remained the same for all epochs higher than that, the run-time increased considerably. Hence, we chose 500 as our optimal value.

In case of credit card transactions, we achieved highest accuracy at epochs = 1000 in our experiment. While going above may have improved our accuracy marginally, it would be at the cost of much higher runtime. Thus, we made the decision to stop at 1000.

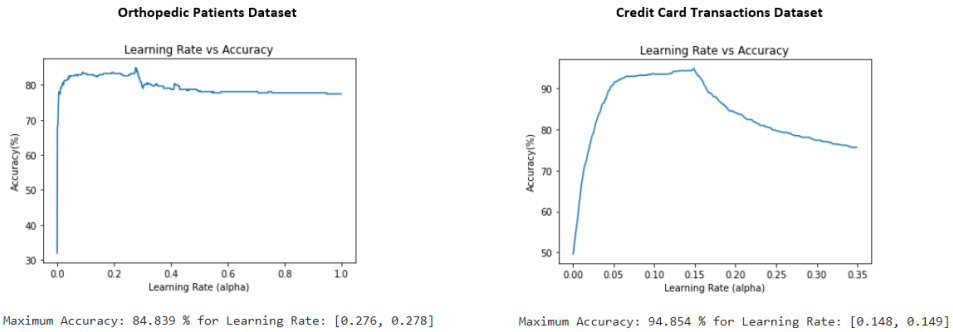

Figure 1: Learning Rate vs Accuracy for both datasets

Orthopedic Patients			Credit Card Transactions		
Learning Rate (alpha)	Epochs	Accuracy (%)	Learning Rate	Epochs	Accuracy (%)
0.28	**500**	**84.51612903**	0.15	500	94.34914228
0.28	1000	84.51612903	**0.15**	**1000**	**94.6518668**
0.28	100	71.61290323	0.15	100	82.94651867
0.28	10	68.06451613	0.15	10	54.69223007

Figure 2: Experimental results of varying epochs for optimal learning rate

4.2 Comparison of given data vs preprocessed data

In this set of experiments, we tested the accuracy of our original data versus the Standardized and Normalized datasets. We found our accuracy to be the highest for original data for both datasets.

For credit card set, we hypothesize the reason for this to be that features (except "Amount") have already been transformed by PCA. For orthopedic patients set, given features are uniformly distributed.

Orthopedic Patients (alpha = 0.28, epochs = 500)		Credit Card Transactions (alpha = 0.15, epochs = 1000)	
Data Type	Accuracy	Datatype	Accuracy
Original	**84.51612903**	**Original (PCA)**	**94.6518668**
Normalized	68.06451613	Normalized	74.47023209
Standardized	71.93548387	Standardized	87.58829465

Figure 3: Testing accuracy on given data vs processed data

4.3 Feature Selection

For feature selection, we developed an automated approach to recursively drop one feature at a time from our training data, and test the accuracy with this subset of features.

For orthopedic dataset, we found that the best accuracy was achieved for entire dataset.

For credit card dataset, we found that best accuracy was achieved when the feature “Amount” was dropped. This was expected, since Amount is the only feature which isn’t transformed by PCA. We also discovered that there are 11 redundant features in this dataset (cols 7, 12, 14, 17, 18, 19, 22, 23, 25, 26, 27) which when dropped, do not cause any change in accuracy. (Note: The table below lists only 7 most relevant deemed outputs out of the total 29 obtained).

4.4 Model Selection

After obtaining the optimal learning rate, epochs, dataset and features, the only thing remaining to be tested is the performance of this linear model against polynomial models. For this experiment, we only tested models of the order 2 and 3. For credit card set, the 2 features which yielded lowest

Orthopedic Patients (alpha = 0.28, epochs = 500)		Credit Card Transactions (alpha = 0.15, epochs = 1000	
Column Dropped	Accuracy	Column Dropped	Accuracy
None	**84.51612903**	None (7,12,14,17,18,19,22,23,25,26,27)	94.6518668
0	83.87096774	1	94.55095863
1	80.32258065	2	77.80020182
2	84.19354839	8	94.45005045
3	82.58064516	11	91.82643794
4	83.87096774	15	94.85368315
5	68.70967742	**28**	**95.0554995**

Figure 4: Results from the automated recursive feature drop code

accuracy after being dropped in above experiment (thus asserting their "importance" in the training set) were squared and cubed to test their accuracy. This was compared with the brute force analysis done on orthopedic set (all columns were powered sequentially) and it supported our hypothesis.

Orthopedic Patients (alpha = 0.28, epochs = 500)		
Model Name	Model Order	Accuracy
Model 1	**Linear**	**84.51612903**
	Square (col = 0)	31.93548387
Model 2	**Square (col = 1)**	**76.12903226**
	Square (col = 2)	67.74193548
	Square (col = 3)	65.48387097
	Square (col = 4)	68.06451613
Model 3	**Square (col = 5)**	**82.90322581**
	Cube (col = 0)	68.06451613
Model 4	**Cube (col = 1)**	**68.38709677**
	Cube (col = 2)	31.93548387
	Cube (col = 3)	68.06451613
	Cube (col = 4)	68.06451613
Model 5	**Cube (col = 5)**	**78.06451613**

Credit Card Transactions (alpha = 0.15, epochs = 1000)		
Model Name	Model Order	Accuracy
Model 1	**Linear (whole dataset)**	**94.6518668**
Model 2	**Linear (w/o col 28 "Amount")**	**95.0554995**
	Square (col = 2)	80.9283552
Model 3	**Cube (col = 2)**	**92.83551968**
Model 4	**Square (col = 11)**	**88.79919273**
Model 5	**Cube (col = 11)**	**93.64278507**
Model 6	**Linear (11 features dropped)**	**94.6518668**

Figure 5: Model Selection for 10-fold cross validation

The models written in bold above are the ones fetching us highest accuracies, and these will be tested with 10-fold cross validation to select our overall best model.

4.5 10-Fold Cross Validation

The final step was performing 10-fold cross validation on the above selected models, in order to find out the best model for our training data. We have plotted the average training and validation accuracy of each model for both datasets to zero in on our final model. Please refer Fig. 13 and 14 in Appendix A for detailed experiment results.

Figure 6: Accuracy of Training and Validation Set for all models for both datasets

From the above graph, it is clear that Model 3 – ***Polynomial Model of second order with feature "degree_spondylolisthesis" squared*** – is the best choice for Orthopedic Patients Dataset, with Average Training Accuracy = 81.43% and Average Validation Accuracy = 77.097%.

For Credit Card Dataset, our best model is Model 2 – ***Linear Model with "Amount" feature dropped***, with Average Training Accuracy = 95.17% and Average Validation Accuracy = 94.84%.

4.6 Comparing our results with SciKit-Learn Logistic Regression & Naive Bayes models

Our model's accuracy for both the data sets was comparable to the accuracy obtained from Sci-Kit Learn Libraries' of Logistic Regression and Naive-Bayes Classifier. Below are the results of our experiment. (Please note: We have only stated the Validation Accuracy of our model, in an attempt to keep up with the overall purpose of generalization).

Algorithm	Accuracy (%)	
	Orthopedic Patients (Model-3)	Credit Card Transactions (Model-2)
Logistic Regression (from scratch)	77.10	94.84
Logistic Regression (Scikit-based)	87.74	96.57
Gaussian Naive Bayes (Scikit-based)	78.06	91.73

Figure 7: Comparison of our model with SciKit-Learn Based Models

5 Discussion and Conclusion

By developing the code for Logistic Regression from scratch, and testing it rigorously for innumerable models, we can offer many insights:

- The run time of the code is dependent on epochs (no. of iterations), whereas the accuracy of the model is dependent on both the learning rate and epochs (along with other parameters such as features and order of the model).
- For Orthopedic Patients Dataset, various models displayed high bias & variance at many folds. There was also a sharp difference in the accuracies achieved by both datasets trained on the same model. We attribute this lower accuracy, and higher bias and variance in Orthopedic set to 1) uneven class distribution in training data, 2) fewer number of features, and 3) fewer observations.
- The Credit Card Dataset includes 11 redundant features. Their removal from the training data not only reduced the run-time of the algorithm but also decreased the cost of getting a new observation for training data.

Future work on this project can include experiments with different regularization techniques on Orthopedic Patients dataset to tackle the problem of high variance. For Credit Card dataset, we can use Dimension Reduction to get rid of the redundant features. We can also test other types of Gradient Descent Algorithms [10] such as Mini-batch and Stochastic Gradient Descent apart from the Batch Gradient Descent covered in this project. Models of orders higher than 3 can also be explored.

6 Statement of Contribution

This project was a team effort and every team member has participated in equal capacity to bring this task to completion. Alok Patel developed the class for Logistic Regression (with method fit()), explored Run time with varying epochs & Accuracy vs Epochs, implemented 10-fold cross validation for T models, and generated the ReadMe file for code. Katyayani Prakash was responsible for developing functions for feature and model selection (including basis expansion, dropping features), automating an approach to select a good subset of features, performing statistical and visual analysis on datasets, and in report planning and writing. Aishwarya Ramamurthy developed the accu_eval() and predict() methods, compared accuracy with Scikit-based Logistic Regression & Gaussian Naive Bayes classifiers, determined optimal learning rate (Learning Rate vs Accuracy plots), wrote a code for generating Confusion Matrix, and contributed to report writing.

References

[1] https://machinelearningmastery.com/data-preparation-is-important/

[2] https://likegeeks.com/seaborn-heatmap-tutorial/

[3] https://datatofish.com/confusion-matrix-python/

[4] https://www.datacamp.com/community/tutorials/naive-bayes-scikit-learn

[5] https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

[6] http://scipy-lectures.org/packages/statistics/index.html

[7] Machine Learning Mastery With Python: Understand Your Data, Create Accurate Models, and Work Projects End-to-End By Jason Brownlee

[8] Lecture Slides from Lec-3 to 9 of ECSE-551: Machine Learning For Engineers by Prof. Narges Armanfard

[9] An Introduction to Statistical Learning: with Applications in R https://linker2.worldcat.org/?jHome=https%3A%2F%2Fproxy.library.mcgill.ca%2Flogin%3Furl%3Dhttp%3A%2F%2Flink.springer.com%2F10.1007%2F978-1-4614-7138-7&linktype=best

[10] https://towardsdatascience.com/batch-mini-batch-stochastic-gradient-descent-7a62ecba642a

[11] https://www.youtube.com/watch?v=qtixXgeJUHE

[12] https://algorithmia.com/blog/developing-your-own-machine-learning-projects

Appendix-A

This section consists of statistical and descriptive analysis along with their visualizations relevant to this project.

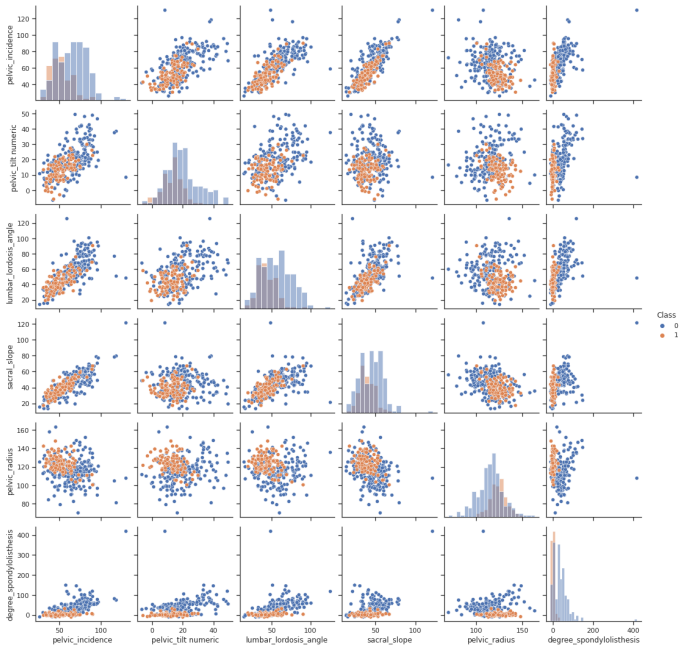

Figure 8: Scatter Matrix plot for Orthopedic Patients dataset.

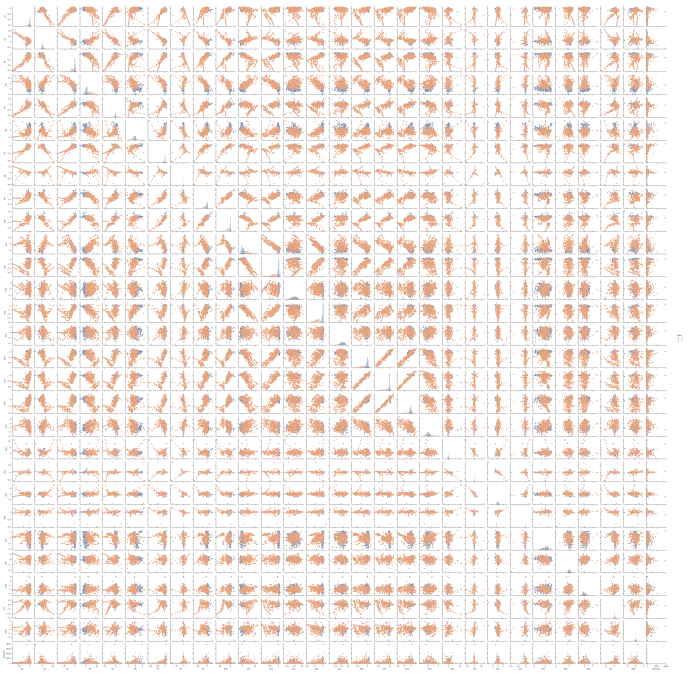

Figure 9: Scatter Matrix plot for Credit Card dataset.

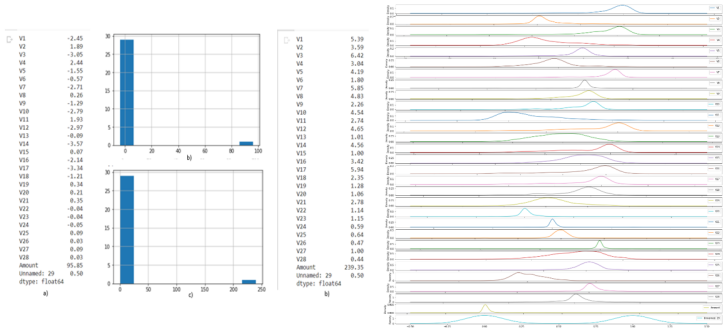

Figure 10: Credit card dataset: a) Mean values corresponding to 30 features, b) Mean histogram, c) Standard Deviation for 30 features,d) Standard deviation Histogram, (right) Density plot.

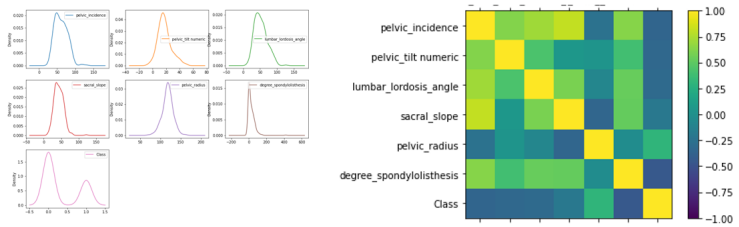

Figure 11: Orthopedic Patients dataset: Density plot(left), Correlation Matrix(right).

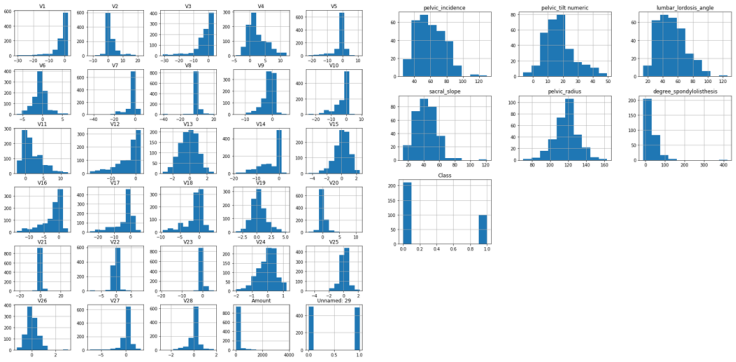

Figure 12: Histograms for Orthopedic Patients(left) and Credit Card(right) dataset.

Credit Card Data Set ML Accuracy for Different Models												
	Model - 1		Model - 2		Model - 3		Model - 4		Model - 5		Model - 6	
Range	Validation	Training	Validation	Training	Validation	Training	Validation	Training	Validation	Training	Validation	Training
0 to 99	93.94	95.07	95.96	95.07	96.97	93.83	91.92	92.15	96.97	93.5	93.94	94.84
99 to 198	89.89	95.4	89.89	95.29	87.88	94.73	86.87	92.71	89.89	94.39	90.91	95.63
198 to 297	95.96	93.61	95.96	94.73	92.93	94.06	81.82	92.825	91.92	93.94	95.96	93.72
297 to 396	89.89	93.72	91.92	95.29	94.95	94.058	91.92	92.38	92.93	93.83	88.89	93.72
396 to 495	86.87	93.83	91.92	95.29	91.92	94.17	85.86	92.94	88.89	93.83	85.86	93.83
495 to 594	95.96	94.39	96.97	95.18	95.96	93.49	95.96	92.94	96.97	93.16	95.96	94.39
594 to 693	93.94	94.73	94.95	95.52	89.89	94.39	90.91	93.72	88.89	94.058	93.94	94.73
693 to 792	85.86	94.73	89.89	95.85	83.84	94.51	83.84	94.28	84.85	94.28	85.86	94.84
792 to 891	93.94	93.83	100	94.62	100	93.5	100	92.49	100	92.94	92.93	93.72
891 to 991	93	93.83	97	94.84	96	93.49	97	92.7	96	93.27	92	93.71
Avg.	**91.93**	**94.32**	**94.84**	**95.17**	**93.034**	**94.024**	**90.61**	**92.91**	**92.73**	**93.72**	**91.62**	**94.32**
Model Type:	Linear Model		Without Amount Feature		Adding V3 cube Feature at end		Adding V12 Sqr. Feature at end		Adding V12 Cube Feature at end		Removing in total 11 features	
Note: In this case the learning rate is taken as 0.15 viz. proved to be optimum one, and number of epoch is taken as 1000. And, model 2 to 6 are taken without the Amount Feature.												

Figure 13: Accuracy obtained after 10-fold Cross Validation for T=6 Models

Orthopedic Patient Data Set ML Accuracy for Different Models										
	Model - 1		Model - 2		Model - 3		Model - 4		Model - 5	
Range	Validation	Training	Validation	Training	Validation	Training	Validation	Training	Validation	Training
0 to 31	35.48	87.1	74.19	78.49	19.35	82.8	93.55	65.59	48.39	81.36
31 to 62	38.71	86.74	83.87	77.78	77.42	84.59	100	64.52	54.84	80.64
62 to 93	96.77	77.78	83.87	73.12	93.55	81.72	96.77	65.23	96.77	75.98
93 to 124	100	78.49	100	70.97	90.32	74.91	96.77	65.69	100	77.06
124 to 155	100	77.78	96.77	80.29	100	75.27	87.097	66.31	100	75.63
155 to 186	96.77	77.06	70.97	66.67	93.55	82.08	70.97	68.46	100	75.63
186 to 217	87.09	78.14	93.55	81.0035	100	81.72	74.19	67.74	96.77	75.98
217 to 248	67.74	87.09	35.48	84.59	77.42	84.59	19.67	75.63	48.38	80.29
248 to 279	61.29	82.44	96.78	75.63	93.55	81.36	100	24.38	58.06	78.85
279 to 310	64.52	83.87	100	69.18	25.8	85.3	16.13	74.19	54.84	79.93
Avg.	**74.83**	**81.65**	**83.54**	**75.77**	**77.097**	**81.43**	**74.19**	**63.76**	**75.81**	**78.14**
Model Type:	**Linear Model**		**Sqr. of pelvic_tilt Feature at end**		**Sqr. of degree_sp Feature at end**		**Cube of pelvic_tilt Feature at end**		**Cube of degree_sp Feature at end**	
Note: In this case the learning rate is taken as 0.28 viz. optimum one, and number of epoch is taken as 500. But, as model-4 was having high bias it is set using alpha = 1										

Figure 14: Accuracy obtained after 10-fold Cross Validation for T=5 Models

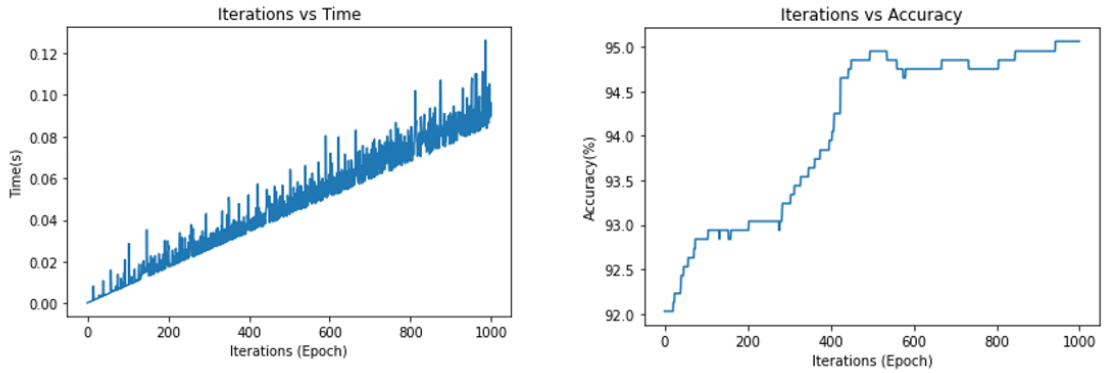

Figure 15: Plots for Credit Card Dataset;Learning rate = 0.15;Epoch range:0-1000.

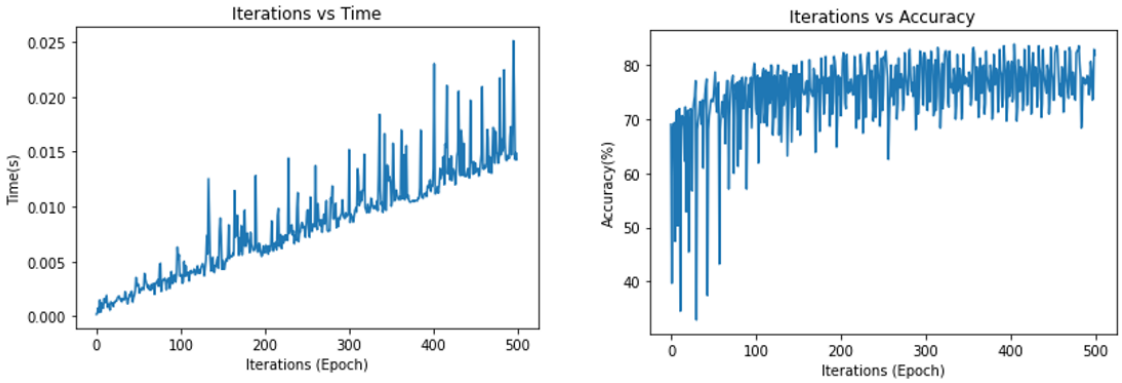

Figure 16: Plots for Orthopedic Patient Dataset;Learning rate = 0.28;Epoch range:0-500.

Appendix – B: ReadMe for the Code

ReadMe:

Note: The numbering below indicates the cell number and are written in a sequential manner.

1) First of all, we need to upload the data set by uploading the respective Excel sheets from the local drive, so for that please run the first cell and upload those files.

2) In the second section, the feature matrix and the output matrix is generated. So run it, as it will be used later for running the algorithm.

3) Run it to see the Descriptive Analysis of the Data Set.

4) Visual Analysis of the data is performed, once you run the code the result will be a set of Histogram plots.

5) 5th Cell will show the Descriptive Analysis of the Credit Card Data set.

6) 6th Cell will be used to run the Visual Analysis of Credit Card Dataset.

7) This function will drop a particular feature from the data set and return the new feature matrix.

8) This function is used for increasing the order of the feature and adding those to the feature matrix. Basically, it is used to make the model more complex by adding the polynomial terms such as cube or square of the feature(s).

9) This is the Class for Logistic Regression, and need to be run to perform the Model Training/Learning using Logistic Regression.

10) This includes the function which calcualtes accuracy of the Model by comparing one vector to the Other Vector.

11) This includes the functions which implement k-fold validation on the model.

12) This cell is used to generate the feature matrix for different models. We can add more models to this, currently 5 models are written for Orthopedic Patients and Credit Card problem.

13) This is the most important part of the code for k-fold implementation. Few lines are commented which can be uncommented to see the results. Also, in the last few lines the X_op_Tn and X_cc_Tn; where n varies from 1 to 5 for each models.

14) This part is used for generating plots depicting Accuracy measurement with respect to alpha (learning rate) and the number of iterations. Last few lines are commented and can be uncommented to see the results. Also, the first argument in the function call can be changed accordingly to requirement of generating a plot for the model chosen by the user.

15) Parameters varies in this section according to the experiment wanted to perform. The 6th Model for the Credit Card Data Set is performed here where in total 11 features were removed.

16) This is used to generate the Confusion Matrix for the data set after training.

17) Logistic Regression is performed using the Inbuilt Python Library SciKit-Learn.

18) Naive Bayes Classifier is peformed using the inbuilt Python Scikit-Learn Library.

19) The scatter plot for both the data set is performed in the Last Cell. Run to see th Scatter Plot.

Appendix – C: Python Code

Loading the data from the Local Drive

```
from google.colab import files
import numpy as np
import io
import pandas as pd

print("Please upload creditcard.csv file saved on your local drive")
ccfile = files.upload()
      #Uploading creditcard file
df_cc = pd.read_csv(io.BytesIO(ccfile['creditcard.csv']))
      #dataframe that stores creditcard.csv
#print(df_cc)

print("Please upload orthopedic_patients.csv file saved on your local driv
e")
opfile = files.upload()
      #Uploading orthopedic patients file
df_op = pd.read_csv(io.BytesIO(opfile['orthopedic_patients.csv']))
      #dataframe that stores orthopedic_patients.csv
#print(df_op)
```

Creating the X and y matrices from dataframes

```
array1 = df_cc.values
array2 = df_op.values

X_cc = array1[:,0:29]
      #Extracting feature from the Credit Card data set
Y_cc = np.vstack(array1[:,29])
      #Output vector for the credit card data set
X_op = array2[:,0:6]
      #Extracting feature from the Orthopedic data set
Y_op = np.vstack(array2[:,6])
      #Output vector for the Orthopedic patient data set

#Adding the dummy feature 1 to the end of both training sets.
X0 = np.ones((991,1))
X_cc = np.hstack((X_cc,X0))

X01 = np.ones((310,1))
X_op = np.hstack((X_op,X01))
```

Statistical analysis of data

```
#Descriptive Analysis of Orthopedic Patients Dataset
from pandas import set_option

shape = df_op.shape
print("Shape of Op training set = ",shape)

set_option('display.max_columns', None)
set_option('precision', 2)

description = df_op.describe()
print("\nData Description :\n",description)

class_counts = df_op.groupby('Class').size()
print("\nClass Distribution of Training Set\n",class_counts)

correlations = df_op.corr(method='pearson')
      #A correlation of -
1 or 1 shows a full negative or positive correlation among the features re
spectively.
print("\nCorrelation Matrix of Op\n",correlations)
      #A value of 0 shows no correlation at all
```

Visual Analysis of Data

```
from matplotlib import pyplot
import numpy

#Visual Analysis of Orthopedic Patients Dataset

#Histogram of all features
df_op.hist(figsize=[14,10])
pyplot.show()

#Density graph of all features
df_op.plot(kind='density', subplots=True, layout=(3,3), sharex=False,figsi
ze=[14,10])
pyplot.show()

#Correlation matrix
names = ['pelvic_incidence','pelvic_tilt numeric','lumbar_lordosis_angle',
'sacral_slope','pelvic_radius','degree_spondylolisthesis','Class']
fig = pyplot.figure()
ax = fig.add_subplot(111)
```

```
cax = ax.matshow(correlations, vmin=-1, vmax=1)
fig.colorbar(cax)
ticks = numpy.arange(0,7,1)
ax.set_xticks(ticks)
ax.set_yticks(ticks)
ax.set_xticklabels(names)
ax.set_yticklabels(names)
pyplot.show()
```

Descriptive Analysis of Creditcard Dataset

```
from pandas import set_option

shape = df_cc.shape
print("Shape of Creditcard training set = ",shape)

class_counts = df_cc.groupby('Unnamed: 29').size()
print("\nClass Distribution of Training Set\n",class_counts)
```

Visual Analysis of Creditcard Dataset

```
from matplotlib import pyplot
import numpy as np

mean_cc = df_cc.mean()
#print(mean_cc)
#mean_cc.hist()
#pyplot.show()
#mean_cc.plot(kind='density', subplots=True, layout=(3,3), sharex=False,fi
gsize=[14,10])

std_cc = df_cc.std()
print(std_cc)
std_cc.hist()
pyplot.show()
```

Feature selection of training sets

```
#This function drops one selected feature from our dataset
def featureDrop(X,col):
  X_dropped = np.delete(X, col, axis=1)
  return X_dropped
```

```
#This function takes the array, its column, and order as inputs and return
s an array which has the new feature appended as the last column.
def polyModel(X,index,order):
  n, m = X.shape
  new_col = np.zeros((n,1))
  for i in range(n):
    new_col[i][0] = pow(X[i][index],order)
  X_pow = np.append(X,new_col,axis = 1)
      #Adding new polynomial feature to the existing feature matrix
  return X_pow

#Calling the function for orthopedic patients dataset
X_op_sq = polyModel(X_op,5,2)
shape = X_op_sq.shape;
shape_old = X_op.shape;
print("Shape of linear model =", shape_old)
print("\nShape of polynomial model = ",shape)
print("\nPolynomial model with powered feature appended on the end =\n",X_
op_sq)

X_op_cub = polyModel(X_op,5,3)
print("\nPolynomial model with powered feature appended on the end =\n",X_
op_cub)
```

Class for training and the validation of the Orthopedic Patients and Credit Card Problem

Notes:

- The constructor of the class is made using the two Arguments i) X-Feature Matrix and ii) Y-Output Label Vector
- Later for the Fit function the arguments are number of epoch and learning rate.

```
import numpy as np

class LogisticRegression:
  def __init__(self, X, y):
    self.X = X
    self.y = y

  #This function is used to train the model
  def fit(self, epoch, alpha):
    n, m = self.X.shape
      #Getting the no. of training examples and no. of features
```

```
    Wk = np.zeros((m,1))
      #Initializing the weight vector with zeros
    k = 0

    loss = []

    while True:
      #if eps < 0.1:
      #Epsilon error is taken as 1%
      if k > epoch:
      #If the number of iterations k exceeds epoch, the algorithm is termi
nated.
        break

      a = np.dot(self.X,Wk)
      #Log-odd ratio is calculated
      y_hat = self.sigmoid(a)
      #Calling the sigmoid function to calculate the probability of the cl
ass

      dy = (1/n)*(self.y - y_hat)
      #Difference in the original and estimated target vector
      dW = (1/n)*(np.dot(self.X.T,dy))
      #Change in update in the Weights by minimization
      #loss.append(self.costfunction(self.X, self.y, Wk))

      Wk1 = Wk + alpha*dW
      #Updating the Weights vector
      k = k + 1
      #Increasing the iteration count
      error = np.square(Wk1 - Wk)
      #Calculating the difference in weights between two iterations

      #eps = error.max()
      #Getting eps as the maximum difference in the weight matrix(vector){
Infinity norm}; Required only when epsilon is used for convergence
      Wk = Wk1



    return Wk1
      #Function returns the updated weight vector

  #This function is used to calculate the estimated probability of the cla
ss
```

```
  def sigmoid(self, a):
    return 1/(1+np.exp(-a))

  def costfunction(self, X, y, W):
    a = np.dot(X, W)
    sig = self.sigmoid(a)
    n = len(X)
    class_1 = np.multiply(y, np.log(sig))
    class_0 = np.multiply((1-y), np.log(1 - sig))
    CEL = -np.sum(class_1 + class_0) / n
    return CEL

  #This function is used to predict our class for given input and weights
  def predict(self,X_v, weights):
    pred_vec = np.zeros((len(X_v),1));
      #Defining a predict vector initialized with zeros
    pred_vec = self.sigmoid(np.dot(X_v,weights))
      #Storing the predicted "Class" values into pred_vec
    for i in range(len(pred_vec)):
      #Checking for each output of each training example
      if pred_vec[i] >= 0.5:
      #It has feature of thresholding the output to 0 and 1.
        pred_vec[i] = 1
      else:
        pred_vec[i] = 0
    return pred_vec
```

Cell for the Accuracy Evaluation of the Model

- Function is defined for the Accuracy evalution of the output of the trained model with the actual output.

```
def Accu_eval(test_y,pred_y):
  cnt = 0;
      #Count variable
  if(len(test_y) == len(pred_y)):

    for i in range(len(test_y)):
      if(test_y[i] == pred_y[i]):
        cnt += 1
        acc_score = cnt / len(test_y)
      #Accuracy score = Total count/Length of test data set
    print("Accuracy Score = ", acc_score*100,"%")
      #Printing Accuracy score in percentage
```

```
    m=np.zeros((len(test_y,),2)).astype(int);
      #Initializing Matrix 'm' with zeros that stores the actual target an
d predicted target column wise
    m[:,0]=np.reshape(test_y,(len(test_y,)));
      #Storing the actual target vector into the 1st column
    m[:,1]=np.reshape(pred_y,(len(pred_y,)));
      #Storing the predicted target vector into the 2nd column

  else:
        print("Dimension mismatch of Test and Predicted data set")
  return acc_score
```

k-fold cross validation implementation using Functions

```
# kfold divides the data into validation and the training set
def kfold(X, Y, l):
  X_t, Y_t = kfoldtrain(X,Y,l)
  X_v, Y_v = kfoldvalidate(X,Y,l)
  return X_t, X_v, Y_t, Y_v

def ktrain(X, y, l):
      #Feature for training of the model
  X = np.delete(X,l,0)
  y = np.delete(y,l,0)
  return X, y

def kvalidate(X, y, l):
      #Output Vector for the validation of the model
  X = X[l,:]
  y = y[l,:]
  return X, y

#Function used to perform the k-fold
def kfold_All(X, Y, Alpha, Epoch, l):
  print("# Accuracy measuremnt for alpha = "+ str(Alpha) +", with Epoch =
"+ str(Epoch))
  avg1 = 0
  avg2 = 0
  for i in range(len(l)):
      #Loop do the K-fold validation
    X_t, Y_t = ktrain(X,Y,l[i])
    X_v, Y_v = kvalidate(X,Y,l[i])
    model = LogisticRegression(X_t,Y_t)
      #Making Object of the Class Logistic Regression
```

```
    weight = model.fit(Epoch, Alpha)
      #Learning the model
    test_y = model.predict(X_v, weight)
      #Output prediction of the model
    #print(test_y)
    print("\nFor "+str(i+1)+"th fold.")
    print("%% Validation accuracy %%")
    acc_val = Accu_eval(test_y, Y_v)
      #Validation Accuracy Calculation
    avg1 = acc_val + avg1
    #print(weight)
    test_y = model.predict(X_t, weight)
    print("%% Training accuracy %%")
    acc_trai = Accu_eval(test_y, Y_t)
      #Training Accuracy Calculation
    avg2 = acc_trai + avg2

  print("\nAverage Accuracy of Validation Set is : "+str((avg1/len(l))*100
))
  print("\nAverage Accuracy of Training Set is : "+str((avg2/len(l))*100))
```

Creation of Feature Matrix for T-Models

- This is used for the k-fold experimentation later.

```
# Feature Matrix for T models.

# Model-1
X_op_T1 = X_op
X_cc_T1 = X_cc

# Model-2
X_op_T2 = polyModel(X_op,1,2)
X_cc_T2 = featureDrop(X_cc,28)

# Model-3
X_op_T3 = polyModel(X_op,5,2)
X_cc_T3 = featureDrop(X_cc,28)
X_cc_T3 = polyModel(X_cc_T3,2,3)

# Model-4
X_op_T4 = polyModel(X_op,1,3)
X_cc_T4 = featureDrop(X_cc,28)
X_cc_T4 = polyModel(X_cc_T3,11,2)
```

```
# Model-5
X_op_T5 = polyModel(X_op,5,3)
X_cc_T5 = featureDrop(X_cc,28)
X_cc_T5 = polyModel(X_cc_T3,11,3)
```

Running the 10 fold cross validation for both the data set for different Models

```
#This range is for 10-folds of each Data Set.

#Orthopedic Patient problem.
op_k = [[i for i in range(0,31)],[i for i in range(31,62)],[i for i in ran
ge(62,93)],
        [i for i in range(93,124)],[i for i in range(124,155)],[i for i in
 range(155,186)],
        [i for i in range(186,217)],[i for i in range(217,248)],[i for i i
n range(248,279)],[i for i in range(279,310)]]

#Credit Card problem.
cc_k = [[i for i in range(0,99)],[i for i in range(99,198)],[i for i in ra
nge(198,297)],
        [i for i in range(297,396)],[i for i in range(396,495)],[i for i i
n range(495,594)],
        [i for i in range(594,693)],[i for i in range(693,792)],[i for i i
n range(792,891)],[i for i in range(891,991)]]

#fold_10 = kFoldCrossValidation()
      #When Class of K-fold was made it was used.

#Running k-fold for whole Credit Card Data set.
kfold_All(X_cc, Y_cc, 0.15, 1000,cc_k)

#Running k-fold for whole Orthopedic Patinet Data Set.
kfold_All(X_op, Y_op, 0.28, 500,op_k)

#Running k-fold for different T Models.[First op and second cc]
#kfold_All(X_op_T5, Y_op, 1, 500, op_k)
kfold_All(X_cc_T3, Y_cc, 0.15, 1000,cc_k)

#K-
fold for the 6th model of the Credi Card case where the 11 features are re
moved.
#kfold_All(X_cc_final, Y_cc, 0.15, 1000,cc_k)
```

Model Testing with different Epochs(No. of Iterations) and Learning rates

- In the second For loop the second argument is variable according to the data set.

```
# Input parameters (in this order): model_exp(Features,Target,Learning Rat
e (lower limit),Learning Rate (Upper limit),Step Size,No. of Iterations)
import matplotlib.pyplot as plt
import time

def model_exp(X,Y,alpha_range_lowlt,alpha_range_uplt,stp_size,itr):
      #For the Testing of the Model
    lea_rate=np.arange(alpha_range_lowlt,alpha_range_uplt,stp_size)

    acc_vec=np.zeros(len(lea_rate))
    t_1 = np.zeros(len(lea_rate))
    t_2 = np.zeros(itr)
    acc_vec1=np.zeros(itr)
    model1 = LogisticRegression(X,Y)
    iteration = []

    for i in range(len(lea_rate)):
      #To get the accuracy w.r.t. varying learning rate
      t1 = time.time()
      weights = model1.fit(itr,lea_rate[i])
      t2 = time.time()
      test_y1 = model1.predict(X, weights)
      acc_vec[i] = np.round((Accu_eval(test_y1, Y)*100),3)
      t_1[i] = t2 - t1

#In the fit function here, 0.15 is used for the CC and 0.28 for the OP.
    for i in range(itr):
      #To get the accuracy w.r.t. iteration
      t3 = time.time()
      weights1 = model1.fit(i+1,0.15)
      #Second argument will be changed accoring to the data set.
      t4 = time.time()
      test_y11 = model1.predict(X, weights1)
      acc_vec1[i] = np.round((Accu_eval(test_y11, Y)*100),2)
      iteration = iteration + [i]
      t_2[i] = t4 - t3

    #print("Learning Rates:\n",np.reshape(lea_rate,(len(lea_rate),1)),"\nA
ccuracy(%):\n",np.reshape(acc_vec,(len(acc_vec),1)))
    plt.plot(lea_rate,acc_vec);
    plt.xlabel('Learning Rate (alpha)'); plt.ylabel('Accuracy(%)')
```

```
    plt.title('Learning Rate vs Accuracy')
    plt.show()
    plt.plot(lea_rate,t_1)
    plt.xlabel('Learning Rate (alpha)'); plt.ylabel('Time(s)')
    plt.title('Learning Rate vs Time')
    plt.show()
    plt.plot(iteration,acc_vec1)
    plt.xlabel('Iterations (Epoch)'); plt.ylabel('Accuracy(%)')
    plt.title('Iterations vs Accuracy')
    plt.show()
    plt.plot(iteration,t_2)
    plt.xlabel('Iterations (Epoch)'); plt.ylabel('Time(s)')
    plt.title('Iterations vs Time')
    plt.show()
    ind = np.where(acc_vec == max(acc_vec))
    print("Maximum Accuracy:",max(acc_vec),"%","for Learning Rate:",list(l
ea_rate[ind].round(3)))

#Running the Model
model_exp(X_cc,Y_cc,0,1,0.01,1000)
#model_exp(X_op,Y_op,0,1,0.001,500)
#model_exp(X_cc_T2,Y_cc,0,1,0.001,1000)
#model_exp(X_op_T2,Y_op,0,0.35,0.001,500)
```

File to train and Validate for various Experiments

```
#This is used for random experiment of the different models with different
 parameters.

import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from numpy import set_printoptions
from sklearn.preprocessing import Normalizer

#This function is used to evaluate accuracy of our predicted output (y_hat
) with true output (y)
def Accu_eval(test_y,pred_y):
  cnt = 0;
        #Count variable
  if(len(test_y) == len(pred_y)):

    for i in range(len(test_y)):
      if(test_y[i] == pred_y[i]):
```

```
        cnt += 1
        acc_score = cnt / len(test_y)
        #Accuracy score = Total count/Length of test data set
    print("Accuracy Score = ", acc_score*100,"%")
        #Printing Accuracy score in percentage
    m=np.zeros((len(test_y,),2)).astype(int);
        #Initializing Matrix 'm' with zeros that stores the actual target
and predicted target column wise
    m[:,0]=np.reshape(test_y,(len(test_y,)));
        #Storing the actual target vector into the 1st column
    m[:,1]=np.reshape(pred_y,(len(pred_y,)));
        #Storing the predicted target vector into the 2nd column

  else:
        print("Dimension mismatch of Test and Predicted data set")
  return acc_score

#Pre-processing data functions

#This functions standardizes the dataset (mean 0, and std deviation 1)
def Standardize(X):
  scaler = StandardScaler().fit(X)
  rescaledX = scaler.transform(X)
  # summarize transformed data
  #set_printoptions(precision=3))
  return rescaledX

#This function normalizes the dataset
def Normalize(X):
  scaler = Normalizer().fit(X)
  normalizedX = scaler.transform(X)
  # summarize transformed data
  #set_printoptions(precision=3)
  return normalizedX

#Testing the effect of learning rate, feature selection and preprocessing
data on our accuracy

#Testing the model for epochs = 10 & alpha = 0.15
model1 = LogisticRegression(X_cc,Y_cc)
weights = model1.fit(10,0.15)
print("For epochs = 10 & alpha = 0.15:")
#print("The weights of our model are:\n",weights)
test_y1 = model1.predict(X_cc, weights)
Accu_eval(test_y1, Y_cc)
```

```
#print("Predicted value of Y is\n",test_y)


#Testing the model for epochs = 500 & alpha = 0.2
model2 = LogisticRegression(X_cc,Y_cc)
weights = model2.fit(500,0.2)
print("\nFor epochs = 500 & alpha = 0.2:")
#print("The weights of our model are:\n",weights)
test_y2 = model2.predict(X_cc, weights)
Accu_eval(test_y2, Y_cc)


#Testing the model for epochs = 1000 & alpha = 0.15
model3 = LogisticRegression(X_cc,Y_cc)
weights = model3.fit(1000,0.15)
print("\nFor epochs = 1000 & alpha = 0.15:")
#print("The weights of our model are:\n",weights)
test_y3 = model3.predict(X_cc, weights)
Accu_eval(test_y3, Y_cc)


#Testing the model for epochs = 100 & alpha = 0.3
model4 = LogisticRegression(X_cc,Y_cc)
weights = model4.fit(100,0.3)
print("\nFor epochs = 100 & alpha = 0.3")
#print("The weights of our model are:\n",weights)
test_y4 = model4.predict(X_cc, weights)
Accu_eval(test_y4, Y_cc)
print("\n\n")


#This loop drops one column of features in every recursion till all featur
es have been dropped once
n, m = X_cc.shape
print("n =",n)
print("m=",m)
for i in range(m):
  X_dropped = featureDrop(X_cc,i)
  print("Dropping features of column no: ",i)
  model_fs = LogisticRegression(X_dropped,Y_cc)
  weights = model_fs.fit(1000,0.15)
  test_y = model_fs.predict(X_dropped, weights)
  Accu_eval(test_y, Y_cc)
  print("\n")
```

```
#Deleting all the extra features from creditcard training set and testing
accuracy for that
X_cc_final = np.delete(X_cc, [7,12,14,17,18,19,22,23,25,26,27], axis=1)
print("X_cc after all dropped features is:",X_cc_final)
print("Shape of dataset is now :",X_cc_final.shape)
model10 = LogisticRegression(X_cc_final,Y_cc)

weights = model10.fit(1000,0.15)
print("\nFor epochs = 1000 & alpha = 0.15")
#print("The weights of our model are:\n",weights)
test_y10 = model10.predict(X_cc_final, weights)
Accu_eval(test_y10, Y_cc)
print("\n\n")
```

Confusion Matrix

- To run it please run one of the Logistic Regression algorithm for a particular case and then choose the argumments in the last line accordingly.

```
def Conf_mat(test_y,pred_y):
    m=np.zeros((len(test_y,),2)).astype(int);
        #Initializing Matrix 'm' with zeros that stores the actual target
and predicted target column wise
    m[:,0]=np.reshape(test_y,(len(test_y,)));
        #Storing the actual target vector into the 1st column
    m[:,1]=np.reshape(pred_y,(len(pred_y,)));
        #Storing the predicted target vector into the 2nd column

    column_values = ['Original_y', 'Predicted_y']
        # Creating the list containing column names
    df = pd.DataFrame(data = m,columns = column_values)
        # Dataframe creation
    df = df.to_string(index=False)

    #Confusion Matrix : Class 1 - positive; Class 0 - negative; Actual Tar
get Variable (Rows), Predicted Target Variable (Columns)
    conf_mat = pd.crosstab(m[:,0],m[:,1],colnames=['Predicted Target Varia
ble'],
                           rownames=['Actual Target Variable'], margins =
False)      #Crosstab() creates a combination of rows and columns with 0 a
nd 1, thus forming a table
    sns.heatmap(conf_mat,annot=True,cmap='Blues',fmt='d')
            #Heatmap displays the confusion matrix
    plt.title('Confusion Matrix');plt.show()
```

```
    accuracy=(conf_mat.iat[0,0]+conf_mat.iat[1,1])/conf_mat.to_numpy().sum
()          #Accuracy : (TP+TN) / (TP+FP+FN+TN);
    precision= conf_mat.iat[1,1]/(conf_mat.iat[1,1]+conf_mat.iat[0,1])
            #Precision : TP / (TP+FP)
    recall= conf_mat.iat[1,1]/(conf_mat.iat[1,1]+conf_mat.iat[1,0])
            #Recall : = TP / (TP+FN)
    spec= conf_mat.iat[0,0]/(conf_mat.iat[0,0]+conf_mat.iat[0,1])
            #Specificity : TN / (FP+TN)
    false_pos_rate = conf_mat.iat[0,1]/(conf_mat.iat[0,0]+conf_mat.iat[0,1
])          #False Positive Rate : FP / (FP+TN)
    print("Accuracy:%.2f"%(accuracy*100),"%","\nError:%.2f"%(100-
accuracy*100),"%")
    print("Precision:%.2f"%(precision*100),"%","\nRecall:%.2f"%(recall*100
),"%","\nSpecificity:%.2f"%(spec*100),"%")
    print("False Positive Rate:%.2f"%(false_pos_rate*100),"%","\nFalse Neg
ative Rate:%.2f"%(100-recall*100),"%")
    print("F1 measure:%.2f"%(2*((precision*recall)/(precision+recall))));

Conf_mat(test_y2, Y_cc)
```

Logistic Regression using inbuilt Python SciKit (sklearn) libraries

- For the Orthopedic Patinet the part is commented, you can uncomment it and run it.

```
import numpy as np
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
model = LogisticRegression()
model.fit(X_cc_T2, Y_cc.ravel())
p_pred = model.predict_proba(X_cc_T2);
y_pred = model.predict(X_cc_T2);
score_ = model.score(X_cc_T2, Y_cc);
conf_m = confusion_matrix(Y_cc, y_pred)
report = classification_report(Y_cc, y_pred)
# rescaledX_op = Standardize(X_op)
# model.fit(rescaledX_op, Y_op.ravel())
# p_pred = model.predict_proba(rescaledX_op);
# y_pred = model.predict(rescaledX_op);
# score_ = model.score(rescaledX_op, Y_op);
# conf_m = confusion_matrix(Y_op, y_pred)
# report = classification_report(Y_op, y_pred)
print("The weights are :\n",model.coef_)
      #weights
```

```
print("\nAccuracy Score is: ",score_)
      #accuracy score
print("\nConfusion matrix is\n",conf_m)
      #Confusion matrix
print("\nClassification Report is\n",report)
      #Classification report
```

Naive Bayes Classifier using inbuilt Python SciKit (sklearn) libraries

```
from sklearn.naive_bayes import GaussianNB
from sklearn import metrics

naive_bayes = GaussianNB()
# rescaledX_op = Standardize(X_op)
# naive_bayes.fit(normalizedX_op, Y_op.ravel())
# y_prd = naive_bayes.predict(normalizedX_op)
naive_bayes.fit(X_cc_T2, Y_cc.ravel())
y_prd = naive_bayes.predict(X_cc_T2)
print("Accuracy Score:",metrics.accuracy_score(Y_cc, y_prd))
```

Scatter Plots for both the datasets

- Please make the neccessary change in the 7th line of the cell.
- Results are also shown in the Appendix of the Report.

```
import matplotlib.pyplot as plt
import seaborn as sns
from IPython.display import Image

sns.set_theme(style="ticks")
# sns_plot = sns.pairplot(df_op,hue='Class',diag_kind='hist',height=2.0)
sns_plot = sns.pairplot(df_cc,hue='Class',diag_kind='hist',height=2.0)
      #Rename the Target column as 'Class' in creditcard.csv file before e
xecution.
sns_plot.savefig("pairplot.png")
plt.clf() # Clean parirplot figure from sns
Image(filename='pairplot.png') # Show pairplot as image
```

Code Ends