Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Department of Computer Science and Engineering (Data Science)

Subject: Machine Learning – I (DJ19DSC402)

AY: 2021-22

Experiment 5

(Logistic Regression)

Name: Dev Patel **SAP ID: 60009200016**

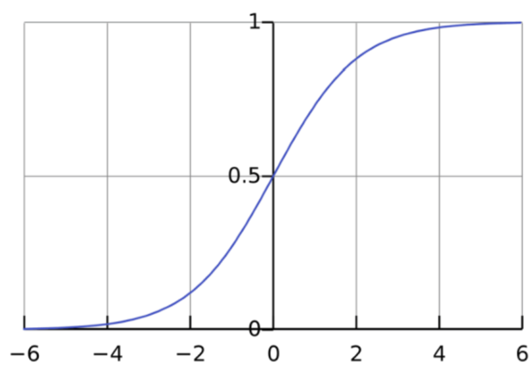
Batch: K/K1 **Date: 25/04/22**

Aim: Implement Logistic Regression on a given Dataset with binary and multiclass labels.

Theory:

Logistic Regression is a statistical approach and a Machine Learning algorithm that is used for classification problems and is based on the concept of probability. It is used when the dependent variable (target) is categorical. It is widely used when the classification problem at hand is binary; true or false, yes or no, etc. For example, it can be used to predict whether an email is spam (1) or not (0). Logistics regression uses the sigmoid function to return the probability of a label.

Sigmoid Function is a mathematical function used to map the predicted values to probabilities. The function has the ability to map any real value into another value within a range of 0 and 1.



The rule is that the value of the logistic regression must be between 0 and 1. Due to the limitations of it not being able to go beyond the value 1, on a graph it forms a curve in the form of an "S". This is an easy way to identify the Sigmoid function or the logistic function. In regards to Logistic Regression, the concept used is the threshold value. The threshold values help to define the probability of either 0 or 1. For example, values above the threshold value tend to 1, and a value below the threshold value tends to 0.

Type of Logistic Regression

1. Binomial: This means that there can be only two possible types of the dependent variables, such as 0 or 1, Yes or No, etc.
2. Multinomial: This means that there can be 3 or more possible unordered types of the dependent variable, such as "cat", "dogs", or "sheep"
3. Ordinal: This means that there can be 3 or more possible ordered types of dependent variables, such as "low", "Medium", or "High".

Binary Logistic Regression Major Assumptions

1. The dependent variable should be dichotomous in nature (e.g., presence vs. absent).
2. There should be no outliers in the data, which can be assessed by converting the continuous predictors to standardized scores, and removing values below -3.29 or greater than 3.29.
3. There should be no high correlations (multicollinearity) among the predictors. This can be assessed by a correlation matrix among the predictors. Tabachnick and Fidell (2013) suggest that as long correlation coefficients among independent variables are less than 0.90 the assumption is met. He aim of training the logistic regression model is to figure out the best weights for our linear model within the logistic regression. In machine learning, we compute the optimal weights by optimizing the cost function. **Cost function:** The cost function J(Θ) is a formal representation of an objective that the algorithm is trying to achieve. In the case of logistic regression, the cost function is called LogLoss (or Cross-Entropy) and the goal is to minimize the following cost function equation:
4. $$J(\theta) =- \frac{1}{m} \sum_{i=1}^{m} [y^{(i)} log(h_\theta(x^{(i)})) + (1 - y^{(i)}) log(1 - h_\theta(x^{(i)}))]$$

Gradient descent is a method of changing weights based on the loss function for each data point. We calculate the LogLoss cost function at each input-output data point. We take a partial derivative of the weight and bias to get the slope of the cost function at each point. (No need to brush up on linear algebra and calculus right now. There are several matrix optimizations built into the Python library and Scikitlearn, which allow data science enthusiasts to unlock the power of advanced artificial intelligence without coding the answers themselves). Based on the slope, gradient descent updates the values for the bias and the set of weights, then reiterates the training loop over new values (moving a step closer to the desired goal). This iterative approach is repeated until a minimum error is reached, and gradient descent cannot minimize the cost function any further. We can change the speed at which we reach the optimal minimum by adjusting the learning rate. A high learning rate changes the weights more drastically, while a low learning rate changes them more slowly.

Lab Assignments to complete in this session:

Use the given dataset and perform the following tasks:
Dataset 1: Synthetic Dataset
Dataset 2: IRIS.csv
Dataset 3: Airlines_Passanger.csv

1. Perform required Logistic Regression from scratch on Dataset 1. Compare the F1 score of the LR model built from scratch and built using python library.
2. Perform Multimodal classification on Dataset 2 using python library.
3. Compare the results of Logistic Regression model with and without regularization.

Code:

1. Perform required Logistic Regression from scratch on Dataset 1. Compare the F1 score of the LR model built from scratch and built using python library.

Using Python libraries

In [1]:

```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler,LabelEncoder
from sklearn.metrics import accuracy_score , confusion_matrix, f1_score
import pandas as pd
import numpy as np
from numpy import shape,dot,log
```

In [2]:

```
X,y = make_classification(n_features = 4,n_classes=2)
print(X,y)
```

```
[[-1.41297301e+00 -8.51966450e-01  1.66056955e-01  1.13254017e+00]
 [ 1.18749214e+00  7.75628630e-01 -1.35968500e-01 -9.91431468e-01]
 [-1.38118844e+00  2.00742160e+00  3.33314061e-01 -7.80488205e-01]
 [-1.46262038e+00 -6.10450303e-01  1.88234109e-01  9.91933089e-01]
 [ 1.57693542e+00  4.39629387e-01 -2.16102551e-01 -9.24228361e-01]
 [ 5.92089930e-01 -7.27946368e-02 -9.54599537e-02 -1.88940990e-01]
 [-5.64731232e-02  1.96486733e+00  1.26979587e-01 -1.28317384e+00]
 [-1.03797163e+00  1.16538264e+00  2.29825167e-01 -3.58453559e-01]
 [-9.43656602e-01  1.24986302e+00  2.20403334e-01 -4.52400344e-01]
 [-3.76819310e-01  4.21285194e-01  8.33267341e-02 -1.28942219e-01]
 [ 1.08465690e+00 -1.48167315e+00 -2.56048388e-01  5.49941350e-01]
 [ 2.88972542e+00 -3.15097879e+00 -6.34209329e-01  9.35827202e-01]
 [-1.10731719e+00  1.88817878e+00  2.84007303e-01 -8.11013775e-01]
 [-5.96088952e-01 -7.65623285e-01  4.55990654e-02  7.47738621e-01]
 [ 1.02944363e+00 -1.18283489e+00 -2.29564052e-01  3.73470113e-01]
 [ 1.39875298e+00  6.71834207e-01 -1.74714242e-01 -1.00712849e+00]
 [-5.37489701e-02  1.94236889e+00  1.25206054e-01 -1.26931374e+00]
 [-1.12250678e+00  7.91199970e-01  2.20301414e-01 -7.58967097e-02]
 [ 8.11958703e-01  1.15129444e+00 -5.55860283e-02 -1.09057141e+00]
 [-2.55447804e-01 -1.34551588e+00 -4.17113967e-02  9.96588718e-01]
 [ 4.97638028e+00 -2.69384452e+00 -9.27664922e-01 -2.04337937e-01]
 [-7.52909671e-02 -1.04949424e+00 -5.16021198e-02  7.27649382e-01]
 [-1.92735026e+00  1.34304591e+00  3.77328654e-01 -1.20048152e-01]
 [-2.17897497e+00  2.02975915e-01  3.47397818e-01  7.38473234e-01]
 [ 1.46976999e+00  6.21989450e-01 -1.88639187e-01 -1.00246744e+00]
 [ 3.95773886e-01  5.12708358e-02 -5.77926781e-02 -1.92705778e-01]
 [-3.73201633e-02 -1.38106940e-01 -2.57385343e-03  1.06741438e-01]
 [-9.93043875e-01 -3.00567463e+00 -2.81997177e-02  2.39553486e+00]
 [ 2.00777622e-01  2.43470353e-01 -1.62264801e-02 -2.42279993e-01]
 [ 1.14562872e+00 -2.29395332e+00 -3.14329734e-01  1.06532707e+00]
 [-9.66107027e-01  6.27926216e-01  1.86413716e-01 -3.00758664e-02]
 [ 1.64779842e+00 -2.79819547e+00 -4.21932655e-01  1.19915835e+00]
 [ 1.30820805e+00 -8.09310286e-01 -2.49956972e-01  1.35005146e-02]
 [-1.49024407e+00  6.71926210e-01  2.69687360e-01  1.50764871e-01]
 [ 2.58241318e+00 -4.72139232e-01 -4.25660838e-01 -7.21297253e-01]
 [ 7.63021261e-01 -1.14075765e+00 -1.86048778e-01  4.52292661e-01]
 [ 1.19488418e+00 -1.79462141e+00 -2.91844658e-01  7.13739332e-01]
 [-1.52647869e+00  3.35594127e+00  4.36849287e-01 -1.61845119e+00]
 [ 1.48087509e+00  5.64419877e-01 -1.93813327e-01 -9.68659016e-01]
 [-1.09316232e+00  8.20616763e-01  2.17558548e-01 -1.07208198e-01]
 [-9.28709062e-01  1.12486015e+00  2.10578388e-01 -3.75317292e-01]
 [-1.10646349e+00 -9.68848282e-01  1.11871801e-01  1.08736362e+00]
 [-2.82030987e-01 -2.08187434e+00 -8.19539290e-02  1.49661188e+00]
```

```
 [ 8.24367983e-01  1.12872701e+00 -5.88535163e-02 -1.08054740e+00]
 [ 3.75461635e-01 -5.23605631e-01 -8.92779813e-02  1.97486387e-01]
 [ 1.36123256e-01  1.76473587e+00  8.53050140e-02 -1.22736631e+00]
 [-5.30538307e-01  1.65572085e+00  1.81290197e-01 -8.87709023e-01]
 [-6.75190008e-01 -6.84824962e-01  6.26310528e-02  7.25746691e-01]
 [ 1.44238688e-01 -3.67773056e-01 -4.43286949e-02  1.86601082e-01]
 [ 8.64686529e-02  1.87340424e+00  9.94853260e-02 -1.27968267e+00]
 [ 1.24121434e+00 -9.30152468e-01 -2.46926927e-01  1.20661773e-01]
 [-1.15854913e+00  1.87856383e+00  2.91309129e-01 -7.84089330e-01]
 [-1.50168517e+00 -3.84162974e-01  2.07866579e-01  8.57205103e-01]
 [-1.10570970e+00  8.58274425e-01  2.21755770e-01 -1.27205497e-01]
 [-1.41524615e+00 -9.28730956e-01  1.61785094e-01  1.18446734e+00]
 [ 3.26377376e-01 -9.52998309e-01 -1.07578777e-01  5.02525377e-01]
 [-1.21466823e+00  1.08405241e+00  2.52108885e-01 -2.33580498e-01]
 [-3.88236772e-01 -1.42903224e+00 -2.63133243e-02  1.10531578e+00]
 [-1.19815336e+00  1.45188679e+00  2.71713573e-01 -4.84654838e-01]
 [-1.26659213e+00  1.23543237e+00  2.69209673e-01 -3.13372510e-01]
 [ 1.88833473e+00  1.55126547e-01 -2.81131311e-01 -8.59967451e-01]
 [ 7.81055407e-01 -6.53875451e-03 -1.20538450e-01 -3.08713517e-01]
 [-4.79785433e-01  4.67431999e-01  1.01943576e-01 -1.18339999e-01]
 [-9.05638984e-01  1.58756231e+00  2.34886141e-01 -6.92066177e-01]
 [-1.44506514e+00  3.53293294e-01  2.43554825e-01  3.44413133e-01]
 [ 4.55007579e-01  1.48264218e+00  1.92699229e-02 -1.16770658e+00]
 [-5.52260400e-01 -2.07172026e+00 -3.97749401e-02  1.59817580e+00]
 [-3.00352706e-01 -2.03247756e+00 -7.61617360e-02  1.47112745e+00]
 [ 1.43760384e+00 -1.02822061e+00 -2.83040361e-01  1.07119845e-01]
 [-1.18885827e+00  1.71127013e+00  2.85899658e-01 -6.60761114e-01]
 [ 6.37367624e-01  1.29463989e+00 -2.00998363e-02 -1.11585696e+00]
 [-1.15842982e-01 -1.92435848e+00 -9.80345041e-02  1.32531947e+00]
 [-9.09377300e-01 -7.68786916e-01  9.35997587e-02  8.75411897e-01]
 [ 1.88885496e-01  1.03123547e+00  3.30293461e-02 -7.61045431e-01]
 [-1.41817658e+00  1.58114207e+00  3.13340020e-01 -4.82366458e-01]
 [-1.47102124e+00 -5.34478612e-01  1.94100152e-01  9.44811109e-01]
 [ 5.11699732e-01  1.46227274e+00  9.32300971e-03 -1.17689257e+00]
 [-8.95658390e-01  8.92332878e-02  1.43145632e-01  2.99691125e-01]
 [-8.06216955e-01  1.88508328e-01  1.35364170e-01  1.97865536e-01]
 [ 2.10558906e+00 -4.14016619e-02 -3.26381929e-01 -8.16437851e-01]
 [-5.55776479e-01 -1.63273616e+00 -1.28055269e-02  1.30784558e+00]
 [ 6.33671912e-01  1.32591155e+00 -1.76486720e-02 -1.13515815e+00]
 [-9.60276542e-01 -3.41486025e-01  1.27154448e-01  6.11837992e-01]
 [ 9.58886852e-01 -1.80714150e+00 -2.56296447e-01  8.16651279e-01]
 [ 5.45106085e-01 -1.84505831e+00 -1.94929920e-01  1.00769969e+00]
 [ 1.57340965e+00  4.85217041e-01 -2.12815651e-01 -9.53111768e-01]
 [ 1.52197928e+00 -2.33290006e+00 -3.74566095e-01  9.40363150e-01]
 [ 3.00720729e-01  1.08358915e+00  1.89783382e-02 -8.40663853e-01]
 [ 9.94210836e-01 -1.40035829e+00 -2.37240173e-01  5.32153393e-01]
 [-1.27596461e+00  8.95605786e-01  2.50192495e-01 -8.37743383e-02]
 [-5.69160807e-01 -2.86870461e-01  7.02796693e-02  4.18776438e-01]
 [-3.20257892e-01  2.16531530e+00  1.79623673e-01 -1.31065831e+00]
 [-1.03196033e+00  1.12112479e+00  2.26235992e-01 -3.31450155e-01]
 [-1.19949574e+00  1.13732696e+00  2.52982333e-01 -2.75066998e-01]
 [-1.60766755e-01  2.03689430e+00  1.47358719e-01 -1.28923894e+00]
 [-5.86181684e-01 -2.16567386e+00 -4.02134178e-02  1.67421154e+00]
 [-1.45003746e-01  2.00981659e+00  1.43303811e-01 -1.27756172e+00]
 [ 1.43235657e+00 -9.16380275e-01 -2.75499981e-01  3.48963174e-02]
 [-7.54956395e-01  1.58452217e+00  2.11524567e-01 -7.50441663e-01]
 [-9.99786342e-02  2.06173984e+00  1.39503867e-01 -1.33011558e+00]] [0 1 1 0 1 0 1 1 1 0
0 0 1 0 0 1 1 1 1 0 1 0 0 0 1 0 0 0 0 0 1 0 0 0 1 0 0
 1 1 1 1 0 0 1 0 1 1 0 0 1 0 1 0 1 0 0 1 0 1 1 1 0 1 1 0 1 0 0 0 1 1 0 0 0
 1 0 1 1 1 1 0 1 0 0 0 1 0 0 0 1 1 1 1 1 1 0 1 0 1 1]
```

In [3]:

```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state = 41)
```

In [4]:

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)
print(X_train,X_test)
```

```
[[ 1.27199268 -1.74434257 -1.62244688  1.1935365 ]
 [ 0.12720056 -0.31410657 -0.21160512  0.27480421]
 [ 0.07919829  1.31704115  0.40279781 -1.51239401]
 [-0.07572448  1.45411339  0.5737652  -1.57386478]
 [-0.91203281 -0.75157419  0.45571536  1.37270985]
 [ 0.68202243  0.79148356 -0.25969875 -1.28189339]
 [ 0.25722461  0.74220707  0.05885548 -0.97729045]
 [-0.76433311  0.77224445  0.8774101  -0.410097  ]
 [-0.2587589   1.52949649  0.74516522 -1.55014905]
 [-0.24221929 -1.52569254 -0.34760239  1.84046518]
 [-0.85512155  0.80173706  0.95963628 -0.38954243]
 [-0.74516372  1.10900299  0.9812578  -0.79617032]
 [ 2.40848021 -2.33974717 -2.73169591  1.1880078 ]
 [ 0.43253127  1.01781619  0.01760594 -1.38710701]
 [ 0.80410796 -1.36169106 -1.11717474  1.04274874]
 [ 0.9592755  -1.7159968  -1.36510482  1.34585033]
 [-0.4655773  -0.25522498  0.27802526  0.55779404]
 [-0.45153441 -1.55425371 -0.19215055  1.99531964]
 [ 1.16960149  0.44252836 -0.76863886 -1.18018795]
 [-0.79056315 -0.29497465  0.52100596  0.79310952]
 [ 0.53388038  0.91857136 -0.0976226  -1.33623849]
 [-0.39131352  0.29376271  0.41329995 -0.09687702]
 [-0.11313674  1.41632321  0.58999934 -1.50980887]
 [-0.30575691  0.26017667  0.33376499 -0.10979967]
 [ 0.86273499 -0.90731545 -1.00296847  0.50257186]
 [-0.47972028 -1.62263393 -0.19402382  2.08799675]
 [-0.98049627  1.19903858  1.19919798 -0.75801376]
 [ 1.23783839  0.3643513  -0.85023406 -1.13329895]
 [ 0.9086128  -1.12481245 -1.11611498  0.71766602]
 [-0.77675332  0.8632226   0.91938428 -0.50405064]
 [-0.55367907 -0.54485962  0.24534881  0.93194855]
 [-0.73687064  0.0185065   0.58932357  0.41264542]
 [-0.055211   -0.810269   -0.24267868  0.93426767]
 [ 1.37653831 -2.08298849 -1.82480651  1.50897226]
 [-0.79540782  0.41057175  0.77417368  0.01070481]
 [ 0.69233355  0.77505879 -0.27365813 -1.26967551]
 [ 1.00020284 -1.35257883 -1.26904397  0.91731323]
 [ 0.12045729  1.2379514   0.34221661 -1.44862756]
 [-0.43348509  1.15860935  0.75228496 -1.03463204]
 [ 0.16429849  0.70410363  0.11888426 -0.88024655]
 [-1.59412398  0.93104189  1.58980136 -0.09895903]
 [ 0.31932832 -0.42752283 -0.40363767  0.2880719 ]
 [-1.17104142  1.10433028  1.31642882 -0.54057521]
 [-0.90098059  0.55081355  0.90723077 -0.08330889]
 [ 0.49932909 -0.09941879 -0.43004832 -0.18292992]
 [ 4.14232332 -2.00704099 -3.98539826 -0.20169668]
 [-0.08890648 -1.44700256 -0.44104733  1.6627453 ]
 [ 0.83345936 -1.06563081 -1.03576244  0.69598495]
 [ 1.19752338 -0.71338753 -1.19921617  0.08989697]
 [ 0.99406065  0.51807081 -0.60310913 -1.16105544]
 [-1.04508688  0.85271985  1.12789495 -0.33459483]
 [ 1.31765683  0.27352773 -0.94545819 -1.07914409]
 [-0.98821974  1.0102572   1.13859212 -0.54336443]
 [ 2.15312834 -0.39006516 -1.84073408 -0.83179898]
 [-0.85012665  0.76952583  0.94430259 -0.35662899]
 [-0.48795243 -0.60366532  0.17258467  0.95875369]
 [-0.20490691 -1.02571604 -0.20042349  1.26206773]
 [-0.91140646  0.57822112  0.92516216 -0.10768285]
 [ 0.17417989  0.13076153 -0.09154671 -0.24794283]
 [-0.92536349  0.52940376  0.91894886 -0.04514449]
 [ 0.27854323 -0.74003844 -0.48182242  0.65987247]
 [-1.24043038 -0.32603529  0.8658247   1.09217828]
 [-1.2149511  -0.43543628  0.8070117   1.19895795]
 [-0.9553118   1.32079635  1.22230835 -0.90833389]
 [ 0.65634427 -0.0511972  -0.53718879 -0.32891616]
 [ 0.46028927 -1.38928723 -0.85500438  1.27561048]
 [ 0.53695123  0.89581158 -0.10809448 -1.312713  ]
 [ 1.0386995  -0.72341105 -1.07714623  0.19443326]
 [-0.61995855  1.10679035  0.88145237 -0.86732201]
 [-1.20797066 -0.49072912  0.78195077  1.25639316]
 [-1.00194228  0.74254424  1.05483688 -0.23733934]
 [-0.0236603  -0.14695364 -0.03321989  0.17746629]
```

```
 [ 1.3147272   0.30670681 -0.93141588 -1.11434899]
 [-1.80320396  0.10128934  1.46193071  0.94746046]
 [-1.05287466  0.6053912   1.04664967 -0.05474624]
 [ 0.38542463  1.03264123  0.06010119 -1.37591055]
 [-0.45445599 -1.23475738 -0.0769317   1.64144704]
 [-0.91274216  1.32779419  1.19111345 -0.94115109]
 [ 0.64135934 -0.87669128 -0.81706232  0.5986457 ]
 [-1.19338366  0.21069159  1.01829219  0.4671554 ]] [[-0.09152862 -1.44579335 -0.55072358
1.41814613]
 [ 0.01074984  1.21108926  0.51731664 -1.15895102]
 [-1.1411389   2.0619632   1.86563988 -1.46349423]
 [ 0.0987147   1.16462702  0.42239936 -1.15334039]
 [ 1.9222728  -0.12955133 -1.68916465 -0.72157858]
 [ 1.24973405 -0.624904   -1.33320989  0.04616956]
 [ 1.38600099  0.29838053 -1.04761787 -0.89366831]
 [-1.04540428 -0.6524201   0.60440503  1.08135561]
 [-1.11057737  0.33059309  1.08707112  0.17314821]
 [-0.86535016  0.63080786  1.00926626 -0.22077457]
 [ 1.35887094 -0.76611588 -1.48729816  0.13277366]
 [ 1.73903298 -0.00277742 -1.47840662 -0.76184636]
 [-0.18110625 -1.02466632 -0.29157344  1.05617124]
 [-1.01859607  1.19207738  1.38341708 -0.68832277]
 [-1.04732152 -0.70193836  0.58450853  1.12939169]
 [-0.62065422 -0.5987637   0.2669304   0.84349487]
 [-0.53364521  0.01875607  0.46145116  0.21671946]
 [ 0.10101234  1.15011401  0.41413901 -1.14051887]
 [ 0.48015548 -0.06977134 -0.43819081 -0.14458496]
 [-0.69122151 -2.04170662 -0.30035946  2.24970963]]
```

In [5]:

```
LR = LogisticRegression()
LR.fit(X_train,y_train)
```

Out[5]:

```
LogisticRegression()
```

In [6]:

```
y_pred = LR.predict(X_test)
y_train_pred = LR.predict(X_train)
```

In [7]:

```
print("Using python libraries, F1_score train_set= ",f1_score(y_train,y_train_pred))
print("Using python libraries, F1_score in test_set= ",f1_score(y_test,y_pred))
```

```
Using python libraries, F1_score train_set=  0.8809523809523809
Using python libraries, F1_score in test_set=  0.9
```

In [8]:

```
confusion_matrix(y_test,y_pred)
```

Out[8]:

```
array([[9, 1],
       [1, 9]])
```

Using Code from scratch

In [9]:

```
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state = 41)
```

In [10]:

```
def standardize(X):
  for i in range(shape(X)[1]):
```

```
    X[:,i] = (X[:,i] - np.mean(X[:,i]))/np.std(X[:,i])
```

In [11]:

```
standardize(X_train)
standardize(X_test)
```

In [12]:

```
def f1Score(y,y_hat):
    tp,tn,fp,fn = 0,0,0,0
    for i in range(len(y)):
        if y[i] == 1 and y_hat[i] == 1:
            tp += 1
        elif y[i] == 1 and y_hat[i] == 0:
            fn += 1
        elif y[i] == 0 and y_hat[i] == 1:
            fp += 1
        elif y[i] == 0 and y_hat[i] == 0:
            tn += 1
    precision = tp/(tp+fp)
    recall = tp/(tp+fn)
    f1_score = 2*precision*recall/(precision+recall)
    return f1_score
```

In [13]:

```
print(np.mean(X_train[:,0]))
print(np.var(X_train[:,0])) #Checking if mean is 0 and variance is 1 after standardizatio
n
```

```
3.3306690738754695e-17
0.9999999999999998
```

In [14]:

```
class Logistic_Regression:

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

    def initialize(self,X):
        weights = np.zeros((shape(X)[1]+1,1))
        X = np.c_[np.ones((shape(X)[0],1)),X]
        return weights,X

    def fit(self,X,y,alpha=0.001,iter=400):
        weights,X = self.initialize(X)

        def cost(theta):
            z = dot(X,theta)
            cost0 = y.T.dot(log(self.sigmoid(z)))
            cost1 = (1-y).T.dot(log(1-self.sigmoid(z)))
            cost = -((cost1 + cost0))/len(y)
            return cost

        cost_list = np.zeros(iter,)
        for i in range(iter):
            weights = weights - alpha*dot(X.T,self.sigmoid(dot(X,weights))-np.reshape(y,
(len(y),1)))
            cost_list[i] = cost(weights)
        self.weights = weights
        return cost_list

    def predict(self,X):
        z = dot(self.initialize(X)[1],self.weights)
        l = []
        for i in self.sigmoid(z):
            if i>0.5:
                l.append(1)
            else:
```

```
            l.append(0)
        return l
```

In [15]:

```
lrs = Logistic_Regression()
lrs.fit(X_train,y_train)
```

Out[15]:

```
array([0.66980573, 0.64846376, 0.62892952, 0.61102598, 0.59459138,
       0.57947919, 0.56555747, 0.55270798, 0.54082506, 0.52981451,
       0.51959244, 0.51008418, 0.50122324, 0.49295041, 0.48521289,
       0.47796357, 0.47116031, 0.46476542, 0.45874509, 0.45306896,
       0.44770969, 0.44264267, 0.43784565, 0.43329851, 0.42898301,
       0.42488259, 0.42098219, 0.41726809, 0.41372775, 0.41034972,
       0.4071235 , 0.40403948, 0.40108882, 0.39826336, 0.39555562,
       0.39295868, 0.39046613, 0.38807206, 0.38577099, 0.38355781,
       0.3814278 , 0.37937656, 0.37739999, 0.37549427, 0.37365583,
       0.37188132, 0.37016764, 0.36851186, 0.36691123, 0.36536319,
       0.36386532, 0.36241535, 0.36101113, 0.35965067, 0.35833205,
       0.35705349, 0.35581329, 0.35460986, 0.35344169, 0.35230733,
       0.35120544, 0.35013474, 0.34909399, 0.34808205, 0.34709782,
       0.34614025, 0.34520834, 0.34430116, 0.3434178 , 0.34255741,
       0.34171916, 0.34090228, 0.34010602, 0.33932967, 0.33857256,
       0.33783403, 0.33711346, 0.33641026, 0.33572386, 0.33505371,
       0.3343993 , 0.33376013, 0.3331357 , 0.33252558, 0.3319293 ,
       0.33134645, 0.33077662, 0.33021943, 0.32967448, 0.32914143,
       0.32861992, 0.32810962, 0.32761021, 0.32712137, 0.32664281,
       0.32617423, 0.32571537, 0.32526594, 0.3248257 , 0.3243944 ,
       0.32397178, 0.32355763, 0.32315171, 0.32275382, 0.32236373,
       0.32198124, 0.32160617, 0.32123832, 0.3208775 , 0.32052355,
       0.32017629, 0.31983554, 0.31950116, 0.31917299, 0.31885086,
       0.31853465, 0.3182242 , 0.31791939, 0.31762006, 0.31732611,
       0.31703739, 0.31675379, 0.31647519, 0.31620148, 0.31593255,
       0.31566827, 0.31540856, 0.31515331, 0.31490242, 0.31465579,
       0.31441333, 0.31417495, 0.31394056, 0.31371007, 0.31348341,
       0.31326048, 0.31304121, 0.31282552, 0.31261334, 0.31240459,
       0.3121992 , 0.3119971 , 0.31179823, 0.31160252, 0.31140991,
       0.31122032, 0.31103371, 0.31085001, 0.31066916, 0.31049111,
       0.3103158 , 0.31014318, 0.3099732 , 0.3098058 , 0.30964094,
       0.30947856, 0.30931863, 0.30916108, 0.30900588, 0.30885299,
       0.30870235, 0.30855393, 0.30840769, 0.30826359, 0.30812158,
       0.30798163, 0.3078437 , 0.30770775, 0.30757375, 0.30744167,
       0.30731146, 0.30718311, 0.30705656, 0.30693179, 0.30680878,
       0.30668748, 0.30656787, 0.30644992, 0.3063336 , 0.30621888,
       0.30610574, 0.30599415, 0.30588407, 0.3057755 , 0.30566839,
       0.30556273, 0.3054585 , 0.30535566, 0.30525419, 0.30515408,
       0.3050553 , 0.30495782, 0.30486163, 0.30476671, 0.30467303,
       0.30458058, 0.30448933, 0.30439927, 0.30431038, 0.30422263,
       0.30413601, 0.30405051, 0.3039661 , 0.30388277, 0.3038005 ,
       0.30371927, 0.30363907, 0.30355989, 0.3034817 , 0.30340449,
       0.30332824, 0.30325295, 0.3031786 , 0.30310516, 0.30303264,
       0.30296101, 0.30289026, 0.30282038, 0.30275136, 0.30268318,
       0.30261583, 0.3025493 , 0.30248357, 0.30241864, 0.30235449,
       0.30229111, 0.3022285 , 0.30216663, 0.3021055 , 0.30204509,
       0.30198541, 0.30192643, 0.30186814, 0.30181055, 0.30175363,
       0.30169738, 0.30164179, 0.30158684, 0.30153254, 0.30147887,
       0.30142582, 0.30137338, 0.30132155, 0.30127031, 0.30121966,
       0.3011696 , 0.3011201 , 0.30107117, 0.3010228 , 0.30097498,
       0.3009277 , 0.30088095, 0.30083473, 0.30078904, 0.30074385,
       0.30069917, 0.30065499, 0.30061131, 0.30056811, 0.30052539,
       0.30048315, 0.30044137, 0.30040006, 0.3003592 , 0.30031879,
       0.30027882, 0.3002393 , 0.3002002 , 0.30016153, 0.30012329,
       0.30008546, 0.30004804, 0.30001102, 0.29997441, 0.29993819,
       0.29990236, 0.29986691, 0.29983185, 0.29979716, 0.29976284,
       0.29972889, 0.2996953 , 0.29966207, 0.29962919, 0.29959666,
       0.29956447, 0.29953262, 0.29950111, 0.29946993, 0.29943908,
       0.29940855, 0.29937834, 0.29934844, 0.29931886, 0.29928958,
       0.29926061, 0.29923194, 0.29920357, 0.29917549, 0.2991477 ,
       0.29912019, 0.29909297, 0.29906603, 0.29903937, 0.29901298,
       0.29898685, 0.298961  , 0.29893541, 0.29891007, 0.298885  ,
```

```
       0.29886018, 0.29883561, 0.29881129, 0.29878721, 0.29876338,
       0.29873979, 0.29871643, 0.29869331, 0.29867042, 0.29864776,
       0.29862533, 0.29860312, 0.29858113, 0.29855936, 0.2985378 ,
       0.29851646, 0.29849533, 0.29847441, 0.2984537 , 0.29843319,
       0.29841288, 0.29839277, 0.29837286, 0.29835314, 0.29833362,
       0.29831429, 0.29829514, 0.29827619, 0.29825741, 0.29823882,
       0.29822042, 0.29820218, 0.29818413, 0.29816625, 0.29814854,
       0.29813101, 0.29811364, 0.29809644, 0.29807941, 0.29806253,
       0.29804582, 0.29802927, 0.29801288, 0.29799665, 0.29798057,
       0.29796464, 0.29794886, 0.29793324, 0.29791776, 0.29790243,
       0.29788724, 0.2978722 , 0.2978573 , 0.29784254, 0.29782792,
       0.29781344, 0.29779909, 0.29778488, 0.2977708 , 0.29775685,
       0.29774303, 0.29772935, 0.29771579, 0.29770235, 0.29768904,
       0.29767586, 0.2976628 , 0.29764985, 0.29763703, 0.29762433,
       0.29761175, 0.29759928, 0.29758692, 0.29757468, 0.29756256,
       0.29755054, 0.29753863, 0.29752684, 0.29751515, 0.29750357,
       0.2974921 , 0.29748073, 0.29746946, 0.2974583 , 0.29744723,
       0.29743627, 0.29742541, 0.29741465, 0.29740398, 0.29739342])
```

In [16]:

```
y_pred = lrs.predict(X_test)
y_trainpred = lrs.predict(X_train)
print(y_pred,y_test)
```

```
[0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0] [0 1 1 1 1 0 1 0 0 1 0 1 0 1
0 0 1 1 0 0]
```

In [17]:

```
print("For model from scratch, F1_score train_set= ",f1Score(y_train,y_trainpred))
print("For model from scratch, F1_score in test_set= ",f1Score(y_test,y_pred))
```

```
For model from scratch, F1_score train_set=  0.8809523809523809
For model from scratch, F1_score in test_set=  0.9
```

The F1 scores for test set using synthetic dataset are:

Using code from scratch: 0.8181818181818182

Using python libraries: 0.8181818181818182

We get the same f1 score for both models.

2. Perform Multimodal classification on Iris Dataset using python library.

In [18]:

```
from sklearn import datasets
iris = datasets.load_iris()
iris = pd.DataFrame(data= np.c_[iris['data'], iris['target']],
                     columns= iris['feature_names'] + ['target'])
iris.head()
```

Out[18]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0.0
1	4.9	3.0	1.4	0.2	0.0
2	4.7	3.2	1.3	0.2	0.0
3	4.6	3.1	1.5	0.2	0.0
4	5.0	3.6	1.4	0.2	0.0

In [19]:

```
iris.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   sepal length (cm)  150 non-null    float64
 1   sepal width (cm)   150 non-null    float64
 2   petal length (cm)  150 non-null    float64
 3   petal width (cm)   150 non-null    float64
 4   target             150 non-null    float64
dtypes: float64(5)
memory usage: 6.0 KB
```

In [20]:

```
iris.corr()
```

Out[20]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
sepal length (cm)	1.000000	-0.117570	0.871754	0.817941	0.782561
sepal width (cm)	-0.117570	1.000000	-0.428440	-0.366126	-0.426658
petal length (cm)	0.871754	-0.428440	1.000000	0.962865	0.949035
petal width (cm)	0.817941	-0.366126	0.962865	1.000000	0.956547
target	0.782561	-0.426658	0.949035	0.956547	1.000000

In [21]:

```
X= iris.iloc[:,:-1]
y= iris.iloc[:,-1]
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.3,random_state = 1
)
```

In [22]:

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.fit_transform(X_test)
```

In [23]:

```
irislr= LogisticRegression(multi_class = 'multinomial')
```

In [24]:

```
irislr.fit(X_train,y_train)
```

Out[24]:

```
LogisticRegression(multi_class='multinomial')
```

In [25]:

```
y_pred = irislr.predict(X_test)
y_train_pred = irislr.predict(X_train)
```

In [26]:

```
print("F1_score train_set= ",f1_score(y_train,y_train_pred,average = 'micro'))
print("F1_score in test_set= ",f1_score(y_test,y_pred,average = 'micro'))
```

```
F1_score train_set=  0.9714285714285714
F1_score in test_set=  0.9555555555555556
```

3. Compare the results of Logistic Regression model with and without regularization.

In [27]:

```
train = pd.read_csv('/content/sample_data/train.csv')
train.head()
```

Out[27]:

	Unnamed: 0	id	Gender	Customer Type	Age	Type of Travel	Class	Flight Distance	Inflight wifi service	Departure/Arrival time convenient	...	Inflight entertainment
0	0	70172	Male	Loyal Customer	13	Personal Travel	Eco Plus	460	3	4	...	5
1	1	5047	Male	disloyal Customer	25	Business travel	Business	235	3	2	...	1
2	2	110028	Female	Loyal Customer	26	Business travel	Business	1142	2	2	...	5
3	3	24026	Female	Loyal Customer	25	Business travel	Business	562	2	5	...	2
4	4	119299	Male	Loyal Customer	61	Business travel	Business	214	3	3	...	3

5 rows × 25 columns

In [28]:

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 103904 entries, 0 to 103903
Data columns (total 25 columns):
 #   Column                             Non-Null Count   Dtype  
---  ------                             --------------   -----  
 0   Unnamed: 0                         103904 non-null  int64  
 1   id                                 103904 non-null  int64  
 2   Gender                             103904 non-null  object 
 3   Customer Type                      103904 non-null  object 
 4   Age                                103904 non-null  int64  
 5   Type of Travel                     103904 non-null  object 
 6   Class                              103904 non-null  object 
 7   Flight Distance                    103904 non-null  int64  
 8   Inflight wifi service              103904 non-null  int64  
 9   Departure/Arrival time convenient  103904 non-null  int64  
 10  Ease of Online booking             103904 non-null  int64  
 11  Gate location                      103904 non-null  int64  
 12  Food and drink                     103904 non-null  int64  
 13  Online boarding                    103904 non-null  int64  
 14  Seat comfort                       103904 non-null  int64  
 15  Inflight entertainment             103904 non-null  int64  
 16  On-board service                   103904 non-null  int64  
 17  Leg room service                   103904 non-null  int64  
 18  Baggage handling                   103904 non-null  int64  
 19  Checkin service                    103904 non-null  int64  
 20  Inflight service                   103904 non-null  int64  
 21  Cleanliness                        103904 non-null  int64  
 22  Departure Delay in Minutes         103904 non-null  int64  
 23  Arrival Delay in Minutes           103594 non-null  float64
 24  satisfaction                       103904 non-null  object 
dtypes: float64(1), int64(19), object(5)
memory usage: 19.8+ MB
```

In [29]:

```
train = train.drop('Unnamed: 0',axis =1)
```

```python
train = train.drop('id',axis =1)
train["Arrival Delay in Minutes"] = train["Arrival Delay in Minutes"].fillna(0)
```

In [30]:

```python
le = LabelEncoder()
train["Gender"]= le.fit_transform(train["Gender"])
train["Class"]= le.fit_transform(train["Class"])
train["Customer Type"]= le.fit_transform(train["Customer Type"])
train["Type of Travel"]= le.fit_transform(train["Type of Travel"])
train.head()
```

Out[30]:

	Gender	Customer Type	Age	Type of Travel	Class	Flight Distance	Inflight wifi service	Departure/Arrival time convenient	Ease of Online booking	Gate location	...	Inflight entertainment	On-board service
0	1	0	13	1	2	460	3	4	3	1	...	5	4
1	1	1	25	0	0	235	3	2	3	3	...	1	1
2	0	0	26	0	0	1142	2	2	2	2	...	5	4
3	0	0	25	0	0	562	2	5	5	5	...	2	2
4	1	0	61	0	0	214	3	3	3	3	...	3	3

5 rows × 23 columns

In [31]:

```python
X_train = train.iloc[:,:-1]
y_train = train.iloc[:,-1]
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
```

In [32]:

```python
test = pd.read_csv('/content/sample_data/test.csv')
test.head()
```

Out[32]:

	Unnamed: 0	id	Gender	Customer Type	Age	Type of Travel	Class	Flight Distance	Inflight wifi service	Departure/Arrival time convenient	...	Inflight entertainment	b se
0	0	19556	Female	Loyal Customer	52	Business travel	Eco	160	5	4	...	5	
1	1	90035	Female	Loyal Customer	36	Business travel	Business	2863	1	1	...	4	
2	2	12360	Male	disloyal Customer	20	Business travel	Eco	192	2	0	...	2	
3	3	77959	Male	Loyal Customer	44	Business travel	Business	3377	0	0	...	1	
4	4	36875	Female	Loyal Customer	49	Business travel	Eco	1182	2	3	...	2	

5 rows × 25 columns

In [33]:

```python
test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 25976 entries, 0 to 25975
Data columns (total 25 columns):
 #   Column                             Non-Null Count  Dtype  
---  ------                             --------------  -----  
 0   Unnamed: 0                         25976 non-null  int64  
 1   id                                 25976 non-null  int64  
 2   Gender                             25976 non-null  object 
 3   Customer Type                      25976 non-null  object 
 4   Age                                25976 non-null  int64  
 5   Type of Travel                     25976 non-null  object 
 6   Class                              25976 non-null  object 
 7   Flight Distance                    25976 non-null  int64  
 8   Inflight wifi service              25976 non-null  int64  
 9   Departure/Arrival time convenient  25976 non-null  int64  
 10  Ease of Online booking             25976 non-null  int64  
 11  Gate location                      25976 non-null  int64  
 12  Food and drink                     25976 non-null  int64  
 13  Online boarding                    25976 non-null  int64  
 14  Seat comfort                       25976 non-null  int64  
 15  Inflight entertainment             25976 non-null  int64  
 16  On-board service                   25976 non-null  int64  
 17  Leg room service                   25976 non-null  int64  
 18  Baggage handling                   25976 non-null  int64  
 19  Checkin service                    25976 non-null  int64  
 20  Inflight service                   25976 non-null  int64  
 21  Cleanliness                        25976 non-null  int64  
 22  Departure Delay in Minutes         25976 non-null  int64  
 23  Arrival Delay in Minutes           25893 non-null  float64
 24  satisfaction                       25976 non-null  object 
dtypes: float64(1), int64(19), object(5)
memory usage: 5.0+ MB
```

In [34]:

```
test = test.drop('Unnamed: 0',axis =1)
test = test.drop('id',axis =1)
test["Arrival Delay in Minutes"] = test["Arrival Delay in Minutes"].fillna(0)
le = LabelEncoder()
test["Gender"]= le.fit_transform(test["Gender"])
test["Class"]= le.fit_transform(test["Class"])
test["Customer Type"]= le.fit_transform(test["Customer Type"])
test["Type of Travel"]= le.fit_transform(test["Type of Travel"])
test.head()
```

Out[34]:

	Gender	Customer Type	Age	Type of Travel	Class	Flight Distance	Inflight wifi service	Departure/Arrival time convenient	Ease of Online booking	Gate location	...	Inflight entertainment	On-board service
0	0	0	52	0	1	160	5	4	3	4	...	5	5
1	0	0	36	0	0	2863	1	1	3	1	...	4	4
2	1	1	20	0	1	192	2	0	2	4	...	2	4
3	1	0	44	0	0	3377	0	0	0	2	...	1	1
4	0	0	49	0	1	1182	2	3	4	3	...	2	2

5 rows × 23 columns

In [35]:

```
X_test = test.iloc[:,:-1]
y_test = test.iloc[:,-1]
scaler = StandardScaler()
X_test = scaler.fit_transform(X_test)
```

L1 regularization

L1 regularization

In [36]:

```
lrl1 = LogisticRegression(penalty='l1',solver='liblinear')
```

In [37]:

```
lrl1.fit(X_train,y_train)
```

Out[37]:

```
LogisticRegression(penalty='l1', solver='liblinear')
```

In [38]:

```
y_pred = lrl1.predict(X_test)
y_train_pred = lrl1.predict(X_train)
```

In [39]:

```
print("F1_score train_set= ",f1_score(y_train,y_train_pred,average = 'micro'))
print("F1_score in test_set= ",f1_score(y_test,y_pred,average = 'micro'))
```

```
F1_score train_set=  0.8755774561133354
F1_score in test_set=  0.8710732984293194
```

L2 Regularization

In [40]:

```
lrl2 = LogisticRegression(penalty='l2')
```

In [41]:

```
lrl2.fit(X_train,y_train)
```

Out[41]:

```
LogisticRegression()
```

In [42]:

```
y_pred = lrl2.predict(X_test)
y_train_pred = lrl2.predict(X_train)
```

In [43]:

```
print("F1_score train_set= ",f1_score(y_train,y_train_pred,average = 'micro'))
print("F1_score in test_set= ",f1_score(y_test,y_pred,average = 'micro'))
```

```
F1_score train_set=  0.8756159531875577
F1_score in test_set=  0.8710732984293194
```

L1+L2 Regularization

In [44]:

```
lrl1l2 = LogisticRegression(penalty='elasticnet',solver='saga',max_iter=116,l1_ratio=1)
```

In [45]:

```
lrl1l2.fit(X_train,y_train)
```

Out[45]:

```
LogisticRegression(l1_ratio=1, max_iter=116, penalty='elasticnet',
                   solver='saga')
```

In [46]:

```
y_pred = lrl1l2.predict(X_test)
y_train_pred = lrl1l2.predict(X_train)
```

In [47]:

```
print("F1_score train_set= ",f1_score(y_train,y_train_pred,average = 'micro'))
print("F1_score in test_set= ",f1_score(y_test,y_pred,average = 'micro'))
```

```
F1_score train_set=  0.875606328919002
F1_score in test_set=  0.8711117955035418
```

Without Regularization

In [48]:

```
lrnoreg = LogisticRegression(penalty='none')
```

In [49]:

```
lrnoreg.fit(X_train,y_train)
```

Out[49]:

```
LogisticRegression(penalty='none')
```

In [50]:

```
y_pred = lrnoreg.predict(X_test)
y_train_pred = lrnoreg.predict(X_train)
```

In [51]:

```
print("F1_score train_set= ",f1_score(y_train,y_train_pred,average = 'micro'))
print("F1_score in test_set= ",f1_score(y_test,y_pred,average = 'micro'))
```

```
F1_score train_set=  0.8755870803818909
F1_score in test_set=  0.8711117955035418
```

In this case, the f1 scores with and without regularization are almost the same