Department of Computer Science and Engineering (Data Science)

Subject: Machine Learning – I (DJ19DSC402)

AY: 2022-23

Bhuvi Ghosh
60009210191

Experiment 5

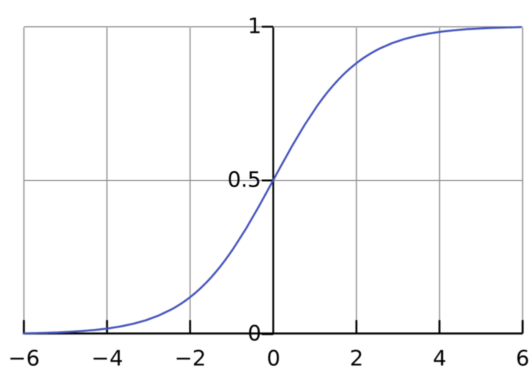
(Logistic Regression)

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. $$J(\theta) =- \frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} log(h_\theta(x^{(i)})) + (1 - y^{(i)}) log(1 - h_\theta(x^{(i)}))\right]$$

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 Scikit-learn, 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.

Q1)

```
[1] import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    from numpy import log,dot,exp,shape
```

```
[2] from sklearn.datasets import make_classification
```

```
[3] X,y = make_classification()
    from sklearn.model_selection import train_test_split
    X_tr,X_te,y_tr,y_te = train_test_split(X,y,test_size=0.3)
```

```
[4] def standardize(X_tr):
        for i in range(shape(X_tr)[1]):
            X_tr[:,i] = (X_tr[:,i] - np.mean(X_tr[:,i]))/np.std(X_tr[:,i])
```

```
def F1_score(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
```

```
class LogidticRegression:
    def sigmoid(self,z):
        sig = 1/(1+exp(-z))
        return sig
    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)
        lis = []
        for i in self.sigmoid(z):
            if i>0.5:
                lis.append(1)
            else:
                lis.append(0)
        return lis
```

```
standardize(X_tr)
standardize(X_te)
obj1 = LogidticRegression()
model= obj1.fit(X_tr,y_tr)
y_pred = obj1.predict(X_te)
y_train = obj1.predict(X_tr)
#Let's see the f1-score for training and testing data
f1_score_tr = F1_score(y_tr,y_train)
f1_score_te = F1_score(y_te,y_pred)
print(f1_score_tr)
print(f1_score_te)
```

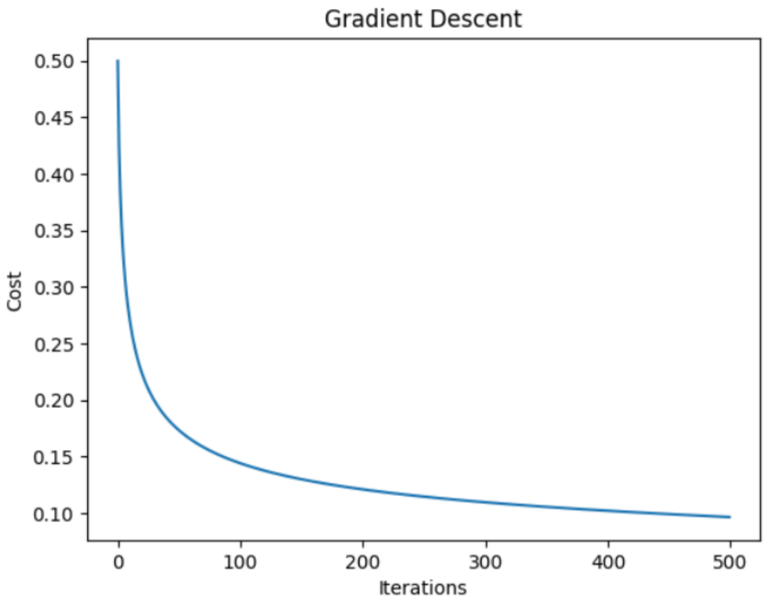
```
0.9315068493150684
0.7333333333333333
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score
model = LogisticRegression().fit(X_tr,y_tr)
y_pred = model.predict(X_te)
f1_score(y_te,y_pred)
```

```
0.6875
```

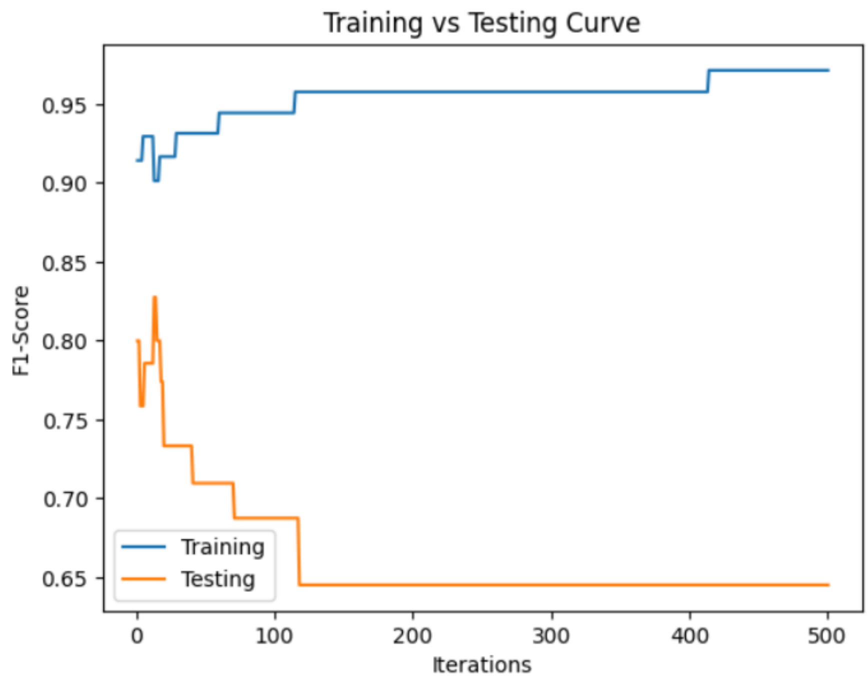
```
def plot_gradient_descent(X, y, alpha=0.001, iter=400):
    obj = LogidticRegression()
    cost_list = obj.fit(X, y, alpha=alpha, iter=iter)
    plt.plot(np.arange(iter), cost_list)
    plt.xlabel('Iterations')
    plt.ylabel('Cost')
    plt.title('Gradient Descent')
    plt.show()
```

```
plot_gradient_descent(X_tr, y_tr, alpha=0.01, iter=500)
```



[11]
```python
def plot_training_testing_curve(X_tr, y_tr, X_te, y_te, alpha=0.001, iter=400):
    obj = LogidticRegression()
    train_scores = []
    test_scores = []
    for i in range(1, iter+1):
        obj.fit(X_tr, y_tr, alpha=alpha, iter=i)
        y_train = obj.predict(X_tr)
        y_test = obj.predict(X_te)
        train_score = F1_score(y_tr, y_train)
        test_score = F1_score(y_te, y_test)
        train_scores.append(train_score)
        test_scores.append(test_score)
    plt.plot(np.arange(1, iter+1), train_scores, label='Training')
    plt.plot(np.arange(1, iter+1), test_scores, label='Testing')
    plt.xlabel('Iterations')
    plt.ylabel('F1-Score')
    plt.title('Training vs Testing Curve')
    plt.legend()
    plt.show()
```

```python
plot_training_testing_curve(X_tr, y_tr, X_te, y_te, alpha=0.01, iter=500)
```



Q2)

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
[346] df=pd.read_csv('/content/train.csv')
```

```
df.head()
```

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

5 rows × 25 columns

```
[348] df.drop(columns='Unnamed: 0',inplace=True)
```

```
[349] df.drop(columns='id',inplace=True)
```

```
df.info()
```

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

```
[351] from sklearn.preprocessing import LabelEncoder
      labelencoder=LabelEncoder()
      df['satisfaction']=labelencoder.fit_transform(df['satisfaction'])
```

```
[352] df['satisfaction'].astype(int)
```

```
0         0
1         0
2         1
3         0
4         1
         ..
103899    0
103900    1
103901    0
103902    0
103903    0
Name: satisfaction, Length: 103904, dtype: int64
```

```
[353] def find_categorical_columns(df):
        cat_cols = []
        for col in df.columns:
          if pd.api.types.is_categorical_dtype(df[col]):
            cat_cols.append(col)
          elif pd.api.types.is_object_dtype(df[col]):
            cat_cols.append(col)
        return cat_cols
```

```
[354] cat_columns=find_categorical_columns(df)
```

```
[355] def encode_categorical(df, cat_cols):
        df=pd.get_dummies(df, columns=cat_cols, prefix=cat_cols, prefix_sep='_')
        return(df)
```

```
[356] cat_cols
```

```
['Gender', 'Customer Type', 'Type of Travel', 'Class']
```

```
[357] cat_cols
```

```
['Gender', 'Customer Type', 'Type of Travel', 'Class']
```

```
[358] df=encode_categorical(df,cat_cols)
```

```
[361] df.dropna(inplace=True)
```

```
[362] x=df.drop(columns='satisfaction').values
      y=df['satisfaction'].values
```

```
[363] from sklearn import linear_model
      model=linear_model.LogisticRegression(C=1e40, solver='newton-cg')
      model.fit(x,y)
```

```
LogisticRegression
LogisticRegression(C=1e+40, solver='newton-cg')
```

```
[364] lr=linear_model.LogisticRegression(penalty='l2', C=1.0, solver='lbfgs', max_iter=1000)
```

```
lr.fit(x,y)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
LogisticRegression
LogisticRegression(max_iter=1000)
```

```
[366] df1=pd.read_csv('/content/test.csv')
```

```
[366] df1=pd.read_csv('/content/test.csv')
```

```
[368] df1.dropna(inplace=True)
```

```
[369] df1.drop(columns='Unnamed: 0',inplace=True)
```

```
[370] df1['satisfaction']=labelencoder.fit_transform(df1['satisfaction'])
```

```
[371] cat_cols=find_categorical_columns(df1)
```

```
[372] df1=encode_categorical(df1,cat_cols)
```

```
[373] df1.head()
```

	id	Age	Flight Distance	Inflight wifi service	Departure/Arrival time convenient	Ease of Online booking	Gate location	Food and drink	Online boarding	Seat comfort	...	satisfaction	Gender_Female	Gender_Male	Customer Type_Loyal Customer	Customer Type_disloyal Customer
0	19556	52	160	5	4	3	4	3	4	3	...	1	1	0	1	0
1	90035	36	2863	1	1	3	1	5	4	5	...	1	1	0	1	0
2	12360	20	192	2	0	2	4	2	2	2	...	0	0	1	0	1
3	77959	44	3377	0	0	0	2	3	4	4	...	1	0	1	1	0
4	36875	49	1182	2	3	4	3	4	1	2	...	1	1	0	1	0

5 rows × 29 columns

```
[374] df1.drop(columns='id',inplace=True)
```

```
[375] x=df1.drop(columns='satisfaction')
      y=df1['satisfaction']
```

Q3)

Accuracy without regularisation:

```
[376] y_pred=model.predict(x)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but LogisticRegression was fitted without feature names
  warnings.warn(
```

```
[377] from sklearn.metrics import accuracy_score
      accuracy_score(y,y_pred)
```

```
0.8717027768122658
```

Accuracy with Lasso regularisation:

```
[378] y_pred=lr.predict(x)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/base.py:432: UserWarning: X has feature names, but LogisticRegression was fitted without feature names
  warnings.warn(
```

```
[379] accuracy_score(y,y_pred)
```

```
0.8688062410690148
```

Conclusion: Logistic regression has successfully been performed with & without Lasso regularisation.