19 $\frac{m}{13/2}$

SDS | Page No.
Date

Machine Learning

Name : Preksha Patel
Sapid : 60004210126
Branch : Computer Engineering
Div : C2, Batch : 1

Experiment no. 3

Aim :- To implement Logistic Regression using gradient descent.

Description of Experiment :

Logistic Regression :

Logistic Regression is a statistical method used for binary classification tasks. It models the probability that an instance belongs to a particular class, often denoted as 0 or 1. The logistic function, also known as the sigmoid function, is employed to transform a linear combination of input features into a probability between 0 & 1.

The steps for performing logistic Regression are given as follows :

1. Problem statement :

Given a dataset $D = \{\{x^{(i)}, y^{(i)}\}\}_{i=1}^{m}$ where $x^{(i)} \in \mathbb{R}^n$ represents the feature vector of the i^{th} instance & $y^{(i)} \in \{0, 1\}$ denotes its corresponding binary label, the objective is to learn a binary classifier that predicts the probability $P(y=1|x)$ for each instance x based on its features.

2. Model Hypothesis :

We assume that the probability of an instance belonging to class 1 follows a logistic function (sigmoid function) as follows :-

$$P(y=1 | x; \theta) = \sigma(\theta^T x) = \frac{1}{1+e^{-\theta^T x}}$$

where θ is parameter vector of logistic regression model.

3. Loss function :

The loss function used for logistic regression is the binary cross-entropy loss, defined as :

$$J(\theta) = \frac{-1}{m} \sum_{i=1}^{m} [y^{(i)} \log(\sigma(\theta^T x^{(i)})) + (1-y^{(i)}) \log (1-\sigma(\theta^T x^{(i)}))]$$

$$+ \lambda \sum_{j=1}^{n} \theta_j^2$$

where λ is a regularization parameter to control overfitting.

4. Gradient descent optimization :

The parameter θ are updated iteratively using gradient descent to minimize the loss function. At each iteration, the gradients of the loss function with respect to the parameters are computed as follows :

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m} \sum_{i=1}^{m} (\sigma(\theta^T x^{(i)}) - y^{(i)}) x_j^{(i)} + 2\lambda\theta_j$$

The parameters then updated as : $\theta_j := \theta_j - \alpha \frac{\partial J(\theta)}{\partial \theta_j}$

$\because \alpha =$ learning rate.

5. Training Procedure : The mode is trained by iteratively updating the parameters using gradient descent until convergence or until a stopping criterion is met (e.g. maximum no. of iterations)

6. Model Evaluation :

After training, the model is evaluated on a separate test dataset to assess its performance using appropriate evaluation metrics such as accuracy, precision, recall, F_1-score, & ROC score.

7. Hyperparameter Tuning :
Hyperparameters such as the learning rate α and regularization parameter λ are tuned to optimize the model's performance through techniques like grid search or randomized search.

8. Cross-validation :
To ensure the robustness of the model, cross-validation techniques such as k-fold cross validation may be employed to estimate the model's performance on unseen data.

9. Deployment & Monitoring :
Once the model is trained and evaluated satisfactorily, it can be deployed for use in real-world application.

Conclusion :-
In Conclusion, the experiment to implement logistic regression without using sklearn and instead employing gradient descent proved to be a challenging yet rewarding endeavor. By meticulously coding the logistic regression algorithm & implementing gradient descent. Through iterative optimization, the model gradually learned to make accurate predictions, demonstrating the power of gradient descent in minimizing the cost function & fine-tuning the model parameters.

13/12

Name:-Preksha Patel
Sapid:-60004210126
Branch:-Computer Engineering
Div:-c2,Batch:-1

MACHINE LEARNING

EXPERIMENT-03

Code and output:-

USING LIBRARY

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

data=pd.read_csv("../content/framingham.csv")
data.head()
```

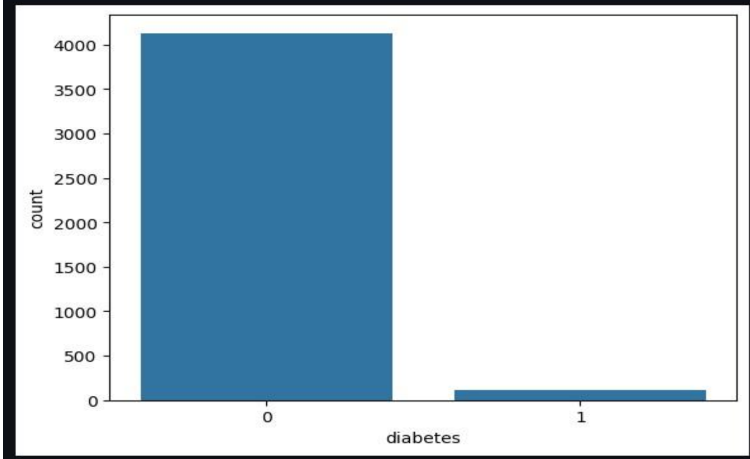
	male	age	education	currentSmoker	cigsPerDay	BPMeds	prevalentStroke	prevalentHyp	diabetes	totChol	sysBP	diaBP	BMI	heartRate	glucose	TenYearCHD
0	1	39	4.0	0	0.0	0.0	0	0	0	195.0	106.0	70.0	26.97	80.0	77.0	0
1	0	46	2.0	0	0.0	0.0	0	0	0	250.0	121.0	81.0	28.73	95.0	76.0	0
2	1	48	1.0	1	20.0	0.0	0	0	0	245.0	127.5	80.0	25.34	75.0	70.0	0
3	0	61	3.0	1	30.0	0.0	0	1	0	225.0	150.0	95.0	28.58	65.0	103.0	1
4	0	46	3.0	1	23.0	0.0	0	0	0	285.0	130.0	84.0	23.10	85.0	85.0	0

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4238 entries, 0 to 4237
Data columns (total 16 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   male             4238 non-null   int64
 1   age              4238 non-null   int64
 2   education        4133 non-null   float64
 3   currentSmoker    4238 non-null   int64
 4   cigsPerDay       4209 non-null   float64
 5   BPMeds           4185 non-null   float64
 6   prevalentStroke  4238 non-null   int64
 7   prevalentHyp     4238 non-null   int64
 8   diabetes         4238 non-null   int64
 9   totChol          4188 non-null   float64
 10  sysBP            4238 non-null   float64
 11  diaBP            4238 non-null   float64
 12  BMI              4219 non-null   float64
 13  heartRate        4237 non-null   float64
 14  glucose          3850 non-null   float64
 15  TenYearCHD       4238 non-null   int64
dtypes: float64(9), int64(7)
memory usage: 529.9 KB
```

```
sns.countplot(data,x='diabetes')
```

```
<Axes: xlabel='diabetes', ylabel='count'>
```




```
data =data.dropna()
```

```
data.isnull().sum()
```

```
male               0
age                0
education          0
currentSmoker      0
cigsPerDay         0
BPMeds             0
prevalentStroke    0
prevalentHyp       0
diabetes           0
totChol            0
sysBP              0
diaBP              0
BMI                0
heartRate          0
glucose            0
TenYearCHD         0
dtype: int64
```

```
data.shape
```

```
(3656, 16)
```

```
x=data.iloc[:,:8]
y=data.iloc[:,8]
y.head()
```

```
0    0
1    0
2    0
3    0
4    0
Name: diabetes, dtype: int64
```

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=42)
```

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
new_xtrain=sc.fit_transform(x_train)
new_xtest=sc.transform(x_test)
```

```
from sklearn.linear_model import LogisticRegression
classifier=LogisticRegression()
classifier.fit(new_xtrain,y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

```
y_pred=classifier.predict(new_xtest)
```

```
y_pred
```

```
array([0, 0, 0, ..., 0, 0, 0])
```

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test,y_pred)
```

```
array([[1070,    0],
       [  27,    0]])
```

```
from sklearn.metrics import accuracy_score
print('Accuracy =' ,accuracy_score(y_test,y_pred))
```

```
Accuracy = 0.975387420237 01
```

WITHOUT USING LIBRARIES

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn import datasets
import matplotlib.pyplot as plt
```

```
d = datasets.load_breast_cancer()
x, y = d.data, d.target
```

```
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size=0.2, random_state=1234)
```

```python
class LogisticRegression:

    def __init__(self, learning_rate=0.001, n_iters=1000):
        self.lr = learning_rate
        self.n_iters = n_iters
        self.weights = None
        self.bias = None

    def fit(self, X, y):
        n_samples, n_features = X.shape

        # init parameters
        self.weights = np.zeros(n_features)
        self.bias = 0

        # gradient descent
        for _ in range(self.n_iters):

            linear_model = np.dot(X, self.weights) + self.bias
            # apply sigmoid function
            y_predicted = self._sigmoid(linear_model)

            # compute gradients
            dw = (1 / n_samples) * np.dot(X.T, (y_predicted - y)) #derivative w.r.t weights
            db = (1 / n_samples) * np.sum(y_predicted - y)  #derivative w.r.t bias
            # update parameters
            self.weights -= self.lr * dw
            self.bias -= self.lr * db

    def predict(self, X):
        linear_model = np.dot(X, self.weights) + self.bias
        y_predicted = self._sigmoid(linear_model)
        y_predicted_cls = [1 if i > 0.5 else 0 for i in y_predicted]
        print(y_predicted_cls)
        return np.array(y_predicted_cls)

    def _sigmoid(self, x):
        return 1 / (1 + np.exp(-x))

def accuracy(y_true, y_pred):
    accuracy = np.sum(y_true == y_pred) / len(y_true)
    return accuracy
```

```python
itr=[]
acc=[]
```

```python
regressor = LogisticRegression(learning_rate=0.0001, n_iters=1000)
regressor.fit(xtrain, ytrain)
predictions = regressor.predict(xtest)
itr.append(1000)
```

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

```python
print("Logistic Regression classification accuracy:", accuracy(ytest, predictions))
acc.append(accuracy(ytest, predictions))
```

```
Logistic Regression classification accuracy: 0.9298245614035088
```

```python
regressor.weights
```

```
array([ 3.15267538e-02,  4.38592690e-02,  1.82394637e-01,  7.27657289e-02,
        2.81683690e-04, -1.58921860e-04, -5.94869592e-04, -2.47270611e-04,
        5.51932783e-04,  2.26761495e-04,  1.51071202e-04,  3.05608006e-03,
       -1.13197589e-03, -8.16912730e-02,  1.44483806e-05, -4.65383514e-05,
       -7.39608956e-05, -1.01074526e-05,  5.54703739e-05,  1.84514942e-06,
        3.33636766e-02,  5.58146134e-02,  1.82897269e-01, -9.88462272e-02,
        3.48754542e-04, -6.74411296e-04, -1.27433782e-03, -2.94192307e-04,
        7.76822709e-04,  1.96607498e-04])
```

```python
regressor.bias
```

```
0.004111914763563403
```