Practical-8

Objective:

Design a logistic regression classifier with gradient descent.

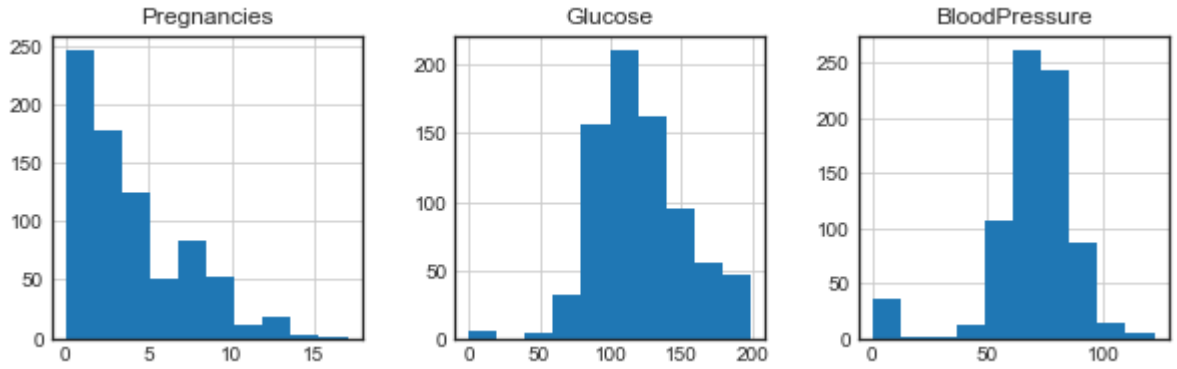
Description:

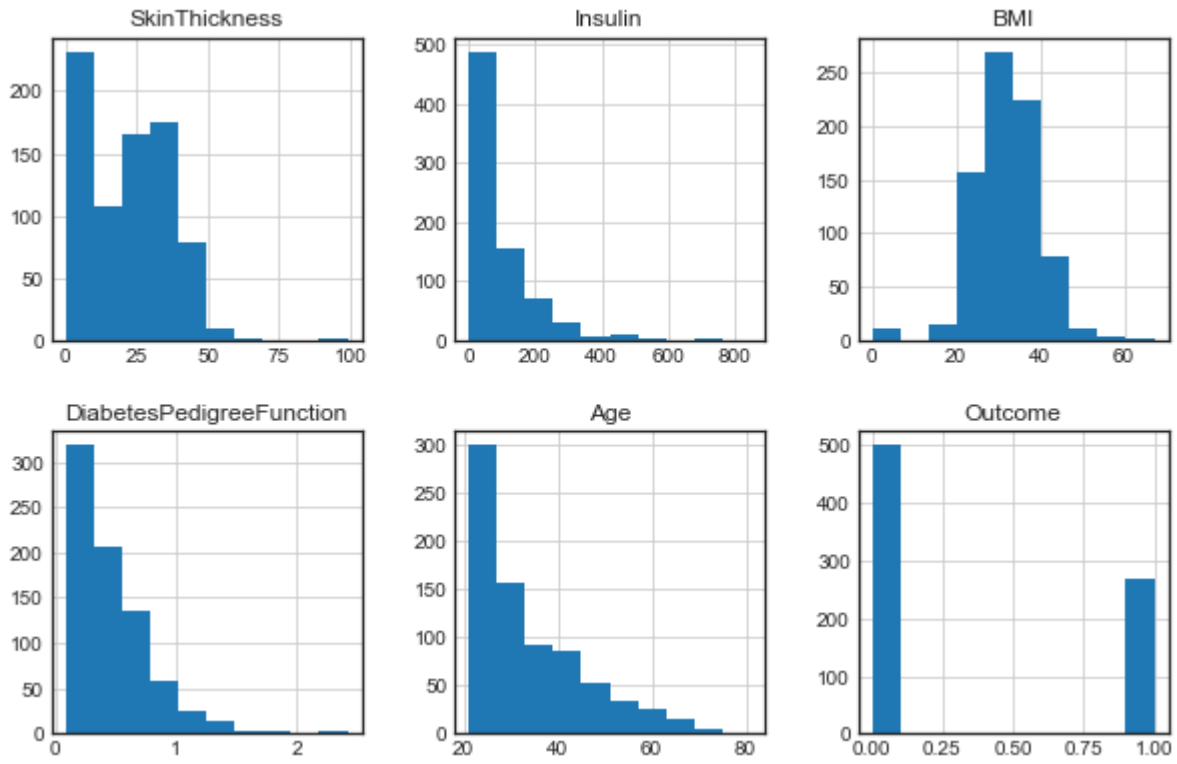
Given a data set with multiple features (can have multiple values) and a class(discrete values), Design a logistic regression classifier with gradient descent which will give out target value x for unknown examples.

Dataset:

1. Name: Diabetics prediction
2. Source: https://www.kaggle.com/kandij/diabetes-dataset
3. The data was collected and made available by “National Institute of Diabetes and Digestive and Kidney Diseases” as part of the Pima Indians Diabetes Database
4. There are 8 features and 1 class(Outcome).
5. There are 768 samples
6. Correlation with Outcome :

Pregnancies	0.221898
Glucose	0.466581
BloodPressure	0.065068
SkinThickness	0.074752
Insulin	0.130548
BMI	0.292695
DiabetesPedigreeFunction	0.173844
Age	0.238356
Outcome	1.000000





Algorithm:

1. Read data from csv using pandas
2. Covert dataset into list of x(data) and y(target)
3. Normalize x by using (0,1) normalization
4. Split the data into training and test data
5. Logistic Regression:
    - Initialize theta vector(size = number of features + 1) to zero
    - Calculate y_cap(ie: predictions) by performing dot product of x and theta and applying sigmoid function for the obtained value.
    - Calculate the cost function by using below formula:

      cost= - 1 /m * [y log(y pred) + (1-y) log(1-y pred)

      m = number of samples
      since we are doing multivariate here we will sum up error vector to get resultant error.

    - Update the value of theta by using following formula:

      θ = θ-(η /m) * (xt.error)

      error= (y - ypred)

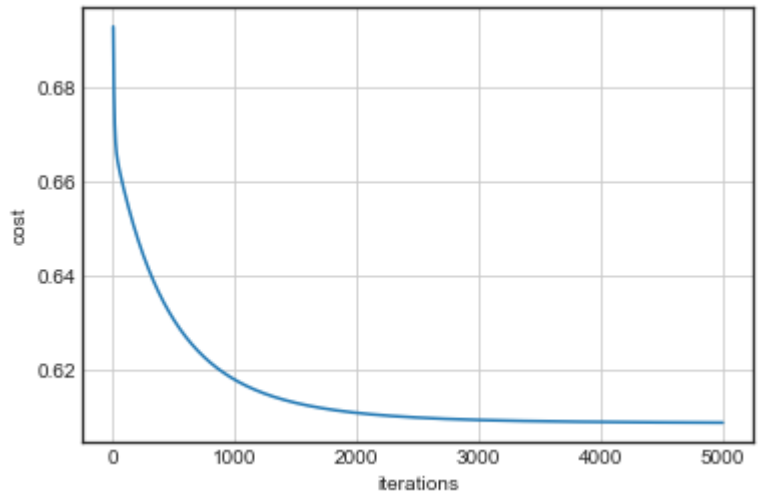
- Repeat (c-f) for specified amount of iterations

Implementation:

```
# 1 read data

data = pd.read_csv('./data/ diabetes2.csv')

# 2 split data into x and y

x = np.array(data.iloc[:,:-1].values)
y = np.array(data['Outcome'])

# 3 Normalize the data

def normalize(x):
 return x/np.max(x, axis=0),np.max(x, axis=0)
x,temp_max = normalize(x)

# 4 split data into test and training

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.30)
n=len(x_train[0])-1
m=len(x_train)

# 5 Logistic regression

def sigmoid(x):
 return 1/(1+np.e**(-x))

def logistic_regression(x, y, theta, lr, itr):
 cost_plot = np.zeros(itr)
 for i in range(itr):
 y_cap = sigmoid(np.dot(x, theta))
 cost = y * np.log(y_cap) + (1 - y)*np.log(1 - y_cap)
 cost = -1*(np.sum(cost) / m)
 cost_plot[i] = cost
 updated_theta = lr * np.dot(x.T,(y_cap - y)) / m
 theta = theta - updated_theta
 plt.plot(range(1, itr + 1), cost_plot)
 plt.grid()
 plt.xlabel("iterations")
 plt.ylabel("cost")
 return theta
```

```
theta = np.zeros(n+1)
itr = 5000
theta = logistic_regression(x, y, theta, 0.2 ,itr)
y_pred = sigmoid(np.dot(x, theta))

count=0

for i in range(len(y)):
 if(y_pred[i]>0.5 and y[i] == 1):
 count+=1
 continue
 if(y_pred[i]<=0.5 and y[i] == 0):
 count+=1
print(count/len(y))
```

Output:



Accuracy : 69.140625%

Theta values:

array([2.07826746, 2.43421778, -3.62649312, -0.02572106, 0.7349827 , -0.33453605, 0.7574637 , -1.07804988])