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: 2022-23

Experiment 5

NAME: DIVYESH KHUNT **SAPID: 60009210116**

(Logistic Regression)

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

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.

From scratch o synthetic data

```
#Libraries
import numpy as np
from numpy import log,dot,e,shape
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
```

```
#synthetic dataset
X,y = make_classification(n_features = 4,n_classes=2)
X_tr,X_te,y_tr,y_te = train_test_split(X,y,test_size=0.1)
```

```
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])
```

```
class LogidticRegression:

    def sigmoid(self,z):
        return 1/(1+e**(-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 pred(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.pred(X_te)
y_train = obj1.pred(X_tr)
```

```
def f1(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 = 2*precision*recall/(precision+recall)
    return f1

F1_TR = f1(y_tr,y_train)
F1_TS = f1(y_te,y_pred)
print(F1_TR)
print(F1_TS)
```

```
0.8686868686868686
0.4444444444444445
```

```
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)
print(f1_score(y_te,y_pred))
```

```
0.9090909090909091
```

Using sklearn on IRIS dataset

```
#LOADING IRIS DATASET
from sklearn import datasets
import numpy as np
iris = datasets.load_iris()
x = iris.data
y = iris.target
```

```
from sklearn.model_selection import train_test_split
x_tr, x_ts, y_tr, y_ts = train_test_split(x, y, test_size=0.3, random_state=0)
```

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
sc.fit(x_tr)
x_tr_std = sc.transform(x_tr)
x_ts_std = sc.transform(x_ts)
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
lr = LogisticRegression(random_state=0, multi_class = 'auto')
lr.fit(x_tr_std, y_tr)
y_pred = model.predict(x_ts_std)
print(accuracy_score(y_ts, y_pred))
```

```
#Regularising model
#all the libraries are already imported in above code
iris = datasets.load_iris()
X = iris.data[:, [2, 3]]
y = iris.target
```

```
#split
x_tr, x_ts, y_tr, y_ts = train_test_split(X, y, test_size=0.3, random_state=0)
```

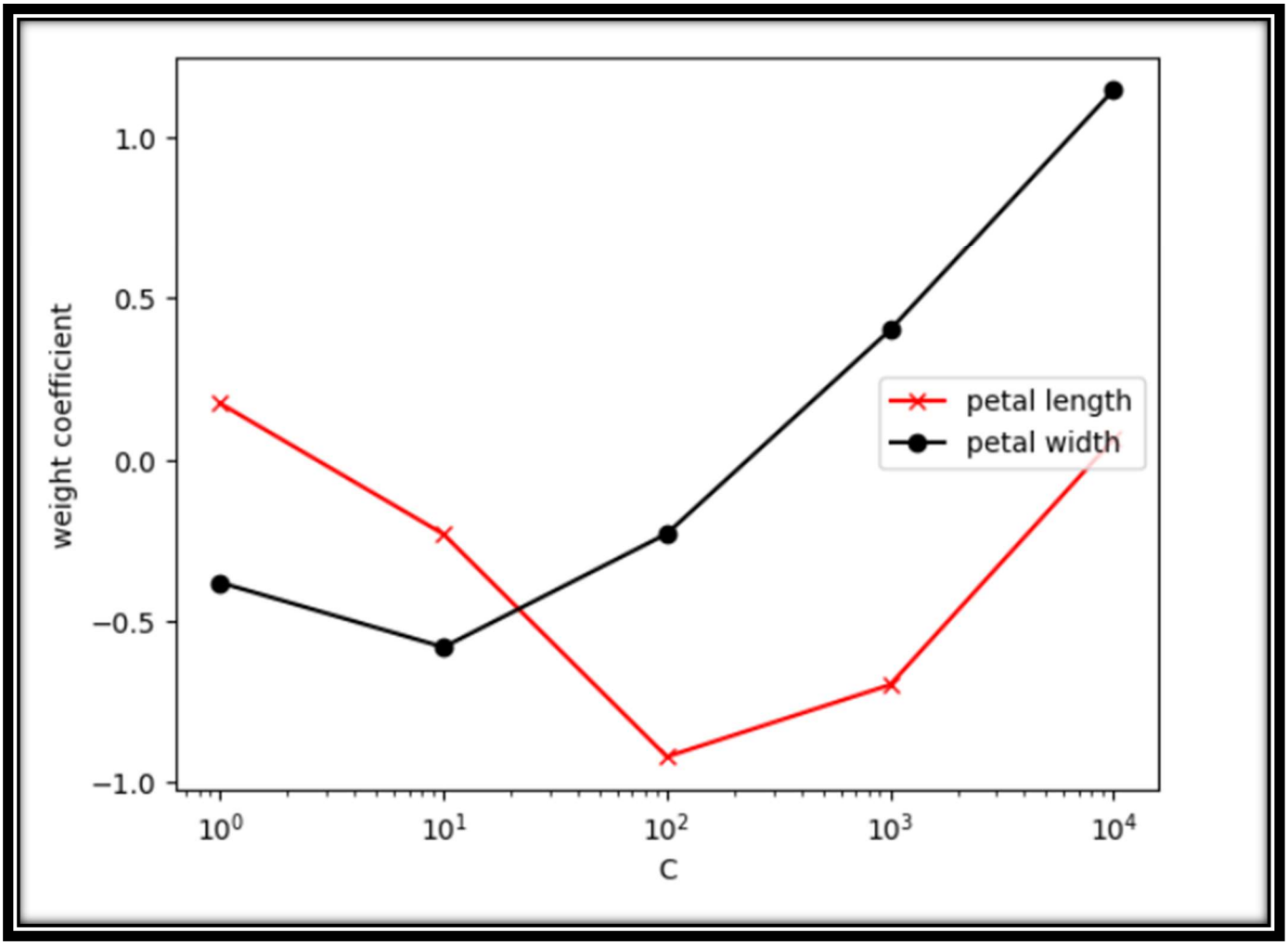
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
sc.fit(x_tr)
x_tr_std = sc.transform(x_tr)
x_ts_std = sc.transform(x_ts)
```

```
from sklearn.linear_model import LogisticRegression

weights, params = [], []
for c in np.arange(0, 5):
    lr = LogisticRegression(C=10**c, random_state=0)
    lr.fit(x_tr_std, y_tr)
    weights.append(lr.coef_[1])
    params.append(10**c)

weights = np.array(weights)
```

```
import matplotlib.pyplot as plt
plt.plot(params, weights[:, 0], color='red', marker='x', label='petal length')
plt.plot(params, weights[:, 1], color='black',  marker='o', label='petal width')
plt.ylabel('weight coefficient')
plt.xlabel('C')
plt.legend(loc='right')
plt.xscale('log')
plt.show()
```



```
#Logistic regression using sklearn
from sklearn import datasets
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from matplotlib.colors import ListedColormap
import matplotlib.pyplot as plt
```

```
iris = datasets.load_iris()
X = iris.data[:, [2, 3]]
y = iris.target
```

```
x_tr, x_ts, y_tr, y_ts = train_test_split(X, y, test_size=0.3, random_state=0)

sc = StandardScaler()
sc.fit(x_tr)
x_tr_std = sc.transform(x_tr)
x_ts_std = sc.transform(x_ts)

lr = LogisticRegression(C=1000.0, random_state=0)
lr.fit(x_tr_std, y_tr)
```

```
LogisticRegression
LogisticRegression(C=1000.0, random_state=0)
```

```
def plot_decision_regions(X, y, classifier, test_idx=None, resolution=0.02):
   markers = ('s', 'x', 'o', '^', 'v')
   colors = ('green', 'white', 'orange', 'blue', 'cyan')
   cmap = ListedColormap(colors[:len(np.unique(y))])

   x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
   x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
   xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
   np.arange(x2_min, x2_max, resolution))
   Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
   Z = Z.reshape(xx1.shape)
   plt.contourf(xx1, xx2, Z, alpha=0.4, cmap=cmap)
   plt.xlim(xx1.min(), xx1.max())
   plt.ylim(xx2.min(), xx2.max())

   x_ts, y_ts = X[test_idx, :], y[test_idx]
   for idx, cl in enumerate(np.unique(y)):
      plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],
               alpha=0.8, c=cmap(idx),
               marker=markers[idx], label=cl)
   if test_idx:
      x_ts, y_ts = X[test_idx, :], y[test_idx]
      plt.scatter(x_ts[:, 0], x_ts[:, 1], c='Yellow',
               alpha=1.0, linewidth=1, marker='o',
               s=55, label='test set')

X_combined_std = np.vstack((x_tr_std, x_ts_std))
y_combined = np.hstack((y_tr, y_ts))
```

```
plot_decision_regions(X_combined_std,
                      y_combined, classifier=lr,
                      test_idx=range(105,150))

plt.xlabel('petal length [standardized]')
plt.ylabel('petal width [standardized]')
plt.legend(loc='upper left')
plt.show()
```

```
<ipython-input-28-31076c4a4c11>:18: UserWarning: *c* argument looks like a single numeric R
  plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],
```