Machine Learning

Lab Assignment 6

Submitted by:

(101983046) SHRUTI MAHAJAN

COE-21

BE Third Year

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

TIET, Patiala

Jan-May 2021

1) Implement Multiclass Logistic Regression (step-by-step) on Iris dataset using one vs. rest strategy?

#Divide the dataset into input features (all columns except price) and output variable (price)

```
import numpy as np

import pandas as pd

from sklearn.datasets import load_iris

dataset = load_iris()

X = dataset.data

y = dataset.target
```

#scaling and inserting 1 column for Beta matrix

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X=scaler.fit_transform(X)

X=np.insert(X, 0, values=1, axis=1)
```

#splitting

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

#As there are 3 classifications "0, 1, 2", so make 3 columns of y_train

```
y_train = [[1 if cla == 0 else 0 for cla in y_train], [1 if cla == 1 else 0 for cla in y_train], [1 if cla == 2 else 0 for cla in y_train]]
```

#Applying the method

```
n=1000

alpha=0.01

m,k=X_train.shape

betas = []

for c in range(3):
```

```
    beta=np.zeros(k)
    for i in range(n):
        cost_gradient=np.zeros(k)
        z=X_train.dot(beta)
        predicted=1/(1+np.exp(-z))
        difference=predicted-y_train[c]
        for j in range(k):
            cost_gradient[j]=np.sum(difference.dot(X_train[:,j]))
        for j in range(k):
            beta[j]=beta[j]-(alpha/m)*cost_gradient[j]
    betas.append(beta)
betas
```

#Prediction

```
Y_predict = []
for i in range(3):
    Y_predict.append(1/(1+np.exp(-(X_test.dot(betas[i])))))
Y_label=np.zeros(len(Y_predict[0]))
for i in range(len(Y_predict[0])):
    if(Y_predict[0][i] > Y_predict[1][i]) and (Y_predict[0][i] > Y_predict[2][i]):
        Y_label[i] = 0
    elif(Y_predict[1][i] > Y_predict[0][i]) and (Y_predict[1][i] > Y_predict[2][i]):
        Y_label[i] = 1
    else:
        Y_label[i] = 2
print(y_test)
Y_label
```

#Accuracy

```
a = 0
for i,j in zip(y_test, Y_label):
    if i == j:
        a += 1
print(a/len(y_test))
```

2) Ridge Logistic Regression

Step-by-Step Logistic Regression (with no regularization; alpha=10; number of iterations=1000)

#Loading the dataset

```
import numpy as np
import pandas as pd
df = pd.read_csv("exam.txt", names = ["t1", "t2", "target"], index_col = None)
x = np.array(df.iloc[:,:2])
y = np.array(df["target"])
```

#Polynomial function of test1 and test2 scores upto degree 6

```
from sklearn.preprocessing import PolynomialFeatures
trans = PolynomialFeatures(degree=6)
x = trans.fit_transform(x)
x = np.insert(x, 0, 1, 1)
```

#Split the dataset

```
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=1)
```

#Applying logistic regression with alpha=10 and i=1000

```
n=1000
alpha=10
```

```
m,k=X_train.shape
beta=np.zeros(k)
for i in range(n):
    cost_gradient=np.zeros(k)
    z=X_train.dot(beta)
    predicted=1/(1+np.exp(-z))
    difference=predicted-y_train
    for j in range(k):
        cost_gradient[j]=np.sum(difference.dot(X_train[:,j]))
    for j in range(k):
        beta[j]=beta[j]-(alpha/m)*cost_gradient[j]
print(beta)

#Prediction
Y_predict=1/(1+np.exp(-(X_test.dot(beta))))
Y_label=np.zeros(len(Y_predict))

for i in range(len(Y_predict)):
    if(Y_predict[i]>=0.5):
        Y_label[i]=1

#Accuracy
TP=0
TN=0
FP=0
FN=0
Y_test=np.array(y_test).reshape(-1,1)
for i in range(len(Y_label)):
    if(Y_test[i]==1 and Y_label[i]==1):
        TP=TP+1
    if(Y_test[i]==1 and Y_label[i]==0):
        FN=FN+1
```

```
    if(Y_test[i]==0 and Y_label[i]==1):
        FP=FP+1
    if(Y_test[i]==0 and Y_label[i]==0):
        TN=TN+1
print(TP,FP,TN,FN)
accuracy=(TP+TN)/(TP+TN+FP+FN)

#For positive class:
precision_pos=TP/(TP+FP)
recall_pos=TP/(TP+FN)
f1_score_pos=2*precision_pos*recall_pos/(precision_pos+recall_pos)

#For negative class
precision_neg=TN/(TN+FN)
recall_neg=TN/(TN+FP)
f1_score_neg=2*precision_neg*recall_neg/(precision_neg+recall_neg)
```

Step-by-Step Logistic Regression (with ridge regularization; alpha=10; number of iterations=1000; lambda=0.2)

```
n=1000
alpha=10
lamda = 0.2
m,k=X_train.shape
beta=np.zeros(k)
for i in range(n):
    cost_gradient=np.zeros(k)
    z=X_train.dot(beta)
    predicted=1/(1+np.exp(-z))
    difference=predicted-y_train
    for j in range(k):
        cost_gradient[j]=np.sum(difference.dot(X_train[:,j]))
    for j in range(k):
```

```
        beta[j]=beta[j]*(1 - alpha*lamda/m)-(alpha/m)*cost_gradient[j]
print(beta)

#Prediction
Y_predict=1/(1+np.exp(-(X_test.dot(beta))))
Y_label=np.zeros(len(Y_predict))

for i in range(len(Y_predict)):
    if(Y_predict[i]>=0.5):
        Y_label[i]=1

#Accuracy
TP=0
TN=0
FP=0
FN=0
Y_test=np.array(y_test).reshape(-1,1)
for i in range(len(Y_label)):
    if(Y_test[i]==1 and Y_label[i]==1):
        TP=TP+1
    if(Y_test[i]==1 and Y_label[i]==0):
        FN=FN+1
    if(Y_test[i]==0 and Y_label[i]==1):
        FP=FP+1
    if(Y_test[i]==0 and Y_label[i]==0):
        TN=TN+1
print(TP,FP,TN,FN)
accuracy=(TP+TN)/(TP+TN+FP+FN)

#For positive class:
precision_pos=TP/(TP+FP)
recall_pos=TP/(TP+FN)
```

f1_score_pos=2*precision_pos*recall_pos/(precision_pos+recall_pos)

#For negative class

precision_neg=TN/(TN+FN)

recall_neg=TN/(TN+FP)

f1_score_neg=2*precision_neg*recall_neg/(precision_neg+recall_neg)