به نام خدا

گزارش کار و نتایج پروژه دوم درس یادگیری ماشین

عنوان پروژه:

پیاده سازی کلاس بندی کننده لوجستیک

زبان برنامه نویسی استفاده شده:

پایتون

استاد مربوطه:

سرکار خانم دکتر شعاران

تهیه کننده:

رضا فرهنگی

دی 1399

با اعمال کلاس بندی کننده لجستیک روی داده های تایتانیک:

ابتدا نتایج روی داده های ولیدیشن بررسی شد و نرخ آموزش بهینه به صورت 0.1 به دست آمد. این نرخ آموزش با تعداد ایپاک 500 عدد بدست آمد. دقت روی داده های ولیدیشن در این حالت 78 درصد مشاهده شد.

سپس با ترکیب داده های ولیدیشن و ترین:

آموزش الگوریتم روی این داده ها انجام شد و روی داده های تست اعتبار سنجی شد که دقت 74 درصد را نتیجه داد.

```
importing libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
importing data
titanicdata = pd.read_csv('titanicdata.csv')
titanicdata.head()
```

	Survived	Pclass	Sex	Age
0	0	3	2	22.0
1	1	1	1	38.0
2	1	3	1	26.0
3	1	1	1	35.0
4	0	3	2	35.0

```
titanicdata.describe()
```

	Survived	Pclass	Sex	Age
count	891.000000	891.000000	891.000000	891.000000
mean	0.383838	2.308642	1.647587	29.699118
std	0.486592	0.836071	0.477990	13.002015
min	0.000000	1.000000	1.000000	0.420000
25%	0.000000	2.000000	1.000000	22.000000
50%	0.000000	3.000000	2.000000	29.699118
75%	1.000000	3.000000	2.000000	35.000000
max	1.000000	3.000000	2.000000	80.000000

```
preprocessing data
```

```
data = titanicdata.to_numpy()
mean_Pclass = np.sum(data[:,1])/len(data[:,1])
sigma_Pclass = np.std(data[:,1])
mean_Sex = np.sum(data[:,2])/len(data[:,2])
sigma_Sex = np.std(data[:,2])
mean_Age = np.sum(data[:,3])/len(data[:,3])
sigma_Age = np.std(data[:,3])
print('mean_Pclass:{}, sigma_Pclass:{}, mean_Sex:{}, sigma_Sex:{}, mean_Ag
e:{}, sigma_Age:{} '.format(mean_Pclass,sigma_Pclass,mean_Sex,sigma_Sex,me
an_Age,sigma_Age))
```

normalize data

```
for i in range(0,len(data)):
  data[i,1] = (data[i,1] - mean_Pclass) / sigma_Pclass
  data[i,2] = (data[i,2] - mean_Sex) / sigma_Sex
  data[i,3] = (data[i,3] - mean_Age) / sigma_Age
```

dividing input and target

```
input = data[:,1:]
target = data[:,0]
```

change class to +1 and -1

```
change_target = list(map(lambda target: 1 if target == 1 else -1, target))
target = np.array(change_target)
```

add attribute x0

```
zero = np.zeros((len(input),4))
x0 = np.ones((len(input),1))
x0 = x0.reshape(len(input),)
zero[:,0] = x0
zero[:,1:] = input
input = zero
```

Logestic_Regression_Classifier

```
class Logestic_Regression_Classifier():
  def __init__(self,learning_rate,epochs):
    self.learning_rate = learning_rate
    self.epochs = epochs
    def train(self,x,y,learning_rate=0.001,epochs=500):
      self.m , self.n = x.shape
      self.w = np.zeros(self.n)
      self.x = x
      self.y = y_other
      self.x1 = x[:,0]
      self.x2 = x[:,1]
      self.x3 = x[:,2]
      self.x4 = x[:,3]
      for i in range(self.epochs):
        self.update_weights()
      return self
```

```
  def update_weights(self):
    for j in range(0,len(self.x)):
      for k in range(0,4):
        A = 1/(1+np.exp(self.y[j]*(self.w(k)*self.x[j][k])))
      sum = 0
      sum += (A*(self.y[j]*self.x1[j])
      gd1 = -sum/self.m
      du = 0
      du += (A*(self.y[j]*self.x2[j])
      gd2 = -du/self.m
      suu = 0
      suu += (A*(self.y[j]*self.x3[j])
      gd3 = -suu/self.m
      duu = 0
      duu += (A*(self.y[j]*self.x4[j])
      gd4 = -duu/self.m
      gd = [gd1,gd2,gd3,gd4]
    self.w = np.transpose(self.w) - self.learning_rate*np.transpose(gd)
    return self
  def predict(self,x):
    teta = 1/(1+np.exp(-(x.dot(self.w))))
    y = np.where(teta>0.5,1,-1)
    return y
```

dividing data to train: 60% , validation: 20% , test: 20%

```
x_train, x_other, y_train, y_other = train_test_split(input, target, test_
size=0.4, random_state=0,shuffle=True)
x_valid, x_test, y_valid, y_test = train_test_split(x_other, y_other, test
_size=0.5, random_state=0,shuffle=True)
```

classify

```
classify = Logestic_Regression_Classifier(learning_rate=0.01,epochs=500)
classify.train(x_train,y_train)
y_pred = classify.predict(x_valid)
correctly_classified = 0
count = 0
for count in range(np.size(y_pred)):
  if y_valid[count] == y_pred[count]:
    correctly_classified += 1
    count +=1
  print('acc on valid: {}'.format(correctly_classified/count)*100)
  print(correctly_classified)
```

Tune Learning Rate

```
best_acc = 0
best_lr = 0
for q in range(200):
  lr = 10 ** np.random.uniform(-5 , 0)
```

```
  classify = Logestic_Regression_Classifier(learning_rate=lr,epochs=500)
  classify.train(x_train,y_train)
  y_pred = classify.predict(x_valid)
  correctly_classified = 0
  count = 0
  for count in range(np.size(y_pred)):
    if y_valid[count] == y_pred[count]:
      correctly_classified += 1
      count +=1
    acc = correctly_classified/count)*100
    print('acc on valid: {}'.format(acc))
    print(correctly_classified)
    if acc > best_acc:
      best_acc = acc
      best_lr = lr
print('best_acc:{} , best_lr:{}'.format(best_acc,best_lr))
add valid data to main data train
training_x = np.zeros(len(x_train)+len(x_valid),len(x_train[0]))
training_y = np.zeros(len(y_train)+len(y_valid),1)
training_x[0:len(x_train),:] = x_train
training_x[len(x_train):len(x_train)+len(x_valid),:] = x_valid
training_y[0:len(y_train),:] = y_train
training_y[len(y_train):len(y_train)+len(y_valid),:] = y_valid
run lg classifier on test data
test = Logestic_Regression_Classifier(learning_rate=0.1,epochs=500)
test.train(training_x,training_y)
y_predicted = classify.predict(x_test)
correctly_classified = 0
count = 0
for count in range(np.size(y_predicted)):
  if y_test[count] == y_predicted[count]:
    correctly_classified += 1
    count +=1
  print('acc on test: {}'.format(correctly_classified/count)*100)
  print(correctly_classified)
```