درس : مبانی وکاربرد های هوش مصنوعی

استاد: حسین کارشناس

دستیار آموزشی: پویا صامتی

پروژه فاز اول

رگرسیون خطی

اعضای تیم:

یونس ایوبی راد (4013613011)

پویا اسفندانی (4013613005)

مقدمه کد

در گام اول کتابخانه های numpy و pandas و matplotlib و ماژول mertics را فرا میخوانیم سپس از با کتابخانه pandas مقادیر را میخوانم و بعد از آن ایدی ها را حذف کرده و تمام ستون های train را در درون X_train(به جز FloodProbility) و مقادیر floodProbility را درون Y_train قرار میدهیم.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import metrics

# ------------------- Data Reading
train_data = pd.read_csv('train.csv')
test_data = pd.read_csv('test.csv')

# ------------------- Data Preprocess
x_train = train_data.iloc[:, 1:-1].values
y_train = train_data.iloc[:, -1].values

x_test = test_data.iloc[:, 1:-1].values
y_test = test_data.iloc[:, -1].values
```

اسکیل کردن

در گام اول اگر min_val و max_val به صورت پیش‌فرض None باشند، تابع با استفاده از X.min(axis=0) وX.max(axis=0) حداقل و حداکثر مقادیر هر ویژگی را در X محاسبه می‌کند.

سپس در گام بعدی تمام مقادیر را اسکیل میکند و انها را بین0-1 اسکیل میکند و انها را باز میگرداند.

```
1 usage
def min_max_scale(X, min_val=None, max_val=None):
    if min_val is None:
        min_val = X.min(axis=0)
    if max_val is None:
        max_val = X.max(axis=0)
    X_scaled = (X - min_val) / (max_val - min_val)
    return X_scaled, min_val, max_val


x_train, x_train_min, x_train_max = min_max_scale(x_train)
x_test = (x_test - x_train_min) / (x_train_max - x_train_min)

x_train = np.hstack((np.ones((x_train.shape[0], 1)), x_train))
x_test = np.hstack((np.ones((x_test.shape[0], 1)), x_test))
```

تابع Stochastic Gradient Descent

```
1 usage
def stochasticGradientDescent(X, Y, parameters, learning_rate, final_learning_rate, epochs):
    num_samples = len(X)
    mse_history = []
    learning_rate_history = [learning_rate]
    previous_mse = 0
    step = (learning_rate - final_learning_rate) / epochs
    for epoch in range(epochs):
        for i in range(num_samples):
            xi = X[i:i + 1]
            yi = Y[i:i + 1]
            gradient = np.dot(xi.T, np.dot(xi, parameters) - yi)
            parameters = parameters - learning_rate * gradient
        if epoch % 1 == 0:
            learning_rate = learning_rate - step
            learning_rate_history.append(learning_rate)
            mse = metrics.mean_squared_error(Y, np.dot(X, parameters))
            mse_history.append(mse)
            print(f"Epoch {epoch}, MSE: {mse}")
            distance = abs(mse - previous_mse)

            if distance < 0.0000001:
                return learning_rate_history, mse_history, parameters
            previous_mse = mse
    return learning_rate_history, mse_history, parameters
```

متغیر mse_history برای ذخیره تاریخچه خطا و متغیر learning_rate_history برای ذخیره تاریخچه نرخ یادگیری تعریف می‌شوند.

متغیر step، که به اندازه تفاوت بین learning_rate و final_learning_rate تقسیم بر تعداد تکرارها محاسبه می‌شود، برای کاهش تدریجی نرخ یادگیری در هر تکرار استفاده می‌شود.

الگوریتم یک حلقه تکرار (به تعداد epochs) را اجرا می‌کند. در هر epoch:

برای هر نمونه در X، گرادیان خطا محاسبه شده و پارامترهای مدل با استفاده از فرمول کاهش گرادیان به‌روزرسانی می‌شوند.

الگوریتم کاهش گرادیان بر اساس فرمول زیر پیاده شد.

$$\theta = \theta - \left(\frac{\alpha}{m}\right) * (X)(\theta^T X - Y)^T$$

در انتهای هر epoch، نرخ یادگیری کاهش یافته و به تاریخچه نرخ یادگیری اضافه می‌شود.

خطای میانگین مربعات بین مقادیر واقعی Y و مقادیر پیش‌بینی‌شده محاسبه شده و به تاریخچه خطا اضافه می‌شود.

اگر اختلاف بین خطای فعلی و خطای دوره قبلی کمتر از یک آستانه مشخص باشد (0.0000001)، الگوریتم متوقف می‌شود و تاریخچه نرخ یادگیری، تاریخچه خطا و پارامترهای به‌روز شده بازگردانده می‌شود.

در پایان حلقه یا در صورت فعال شدن شرط توقف، تابع تاریخچه نرخ یادگیری، تاریخچه خطا و پارامترهای نهایی مدل را بازمی‌گرداند.

تعیین ابرپارامتر ها و اجرای الگوریتم کاهش گرادیان :

```
learning_rate = 0.01
final_learning_rate = 0.0001
epochs = 10
alpha = 0.001
step = (learning_rate - final_learning_rate) / epochs
np.random.seed(100)
parameters = np.zeros(x_train.shape[1])
learning_rate_history, mse_history, out_parameters = stochasticGradientDescent(x_train, y_train, parameters,
learning_rate=learning_rate, final_learning_rate=final_learning_rate, epochs=epochs, alpha=alpha)
```

مقادیر ابر پارامتر ها :

Learning rate : 0.01

Final learning rate : 0.0001

Epochs : 10

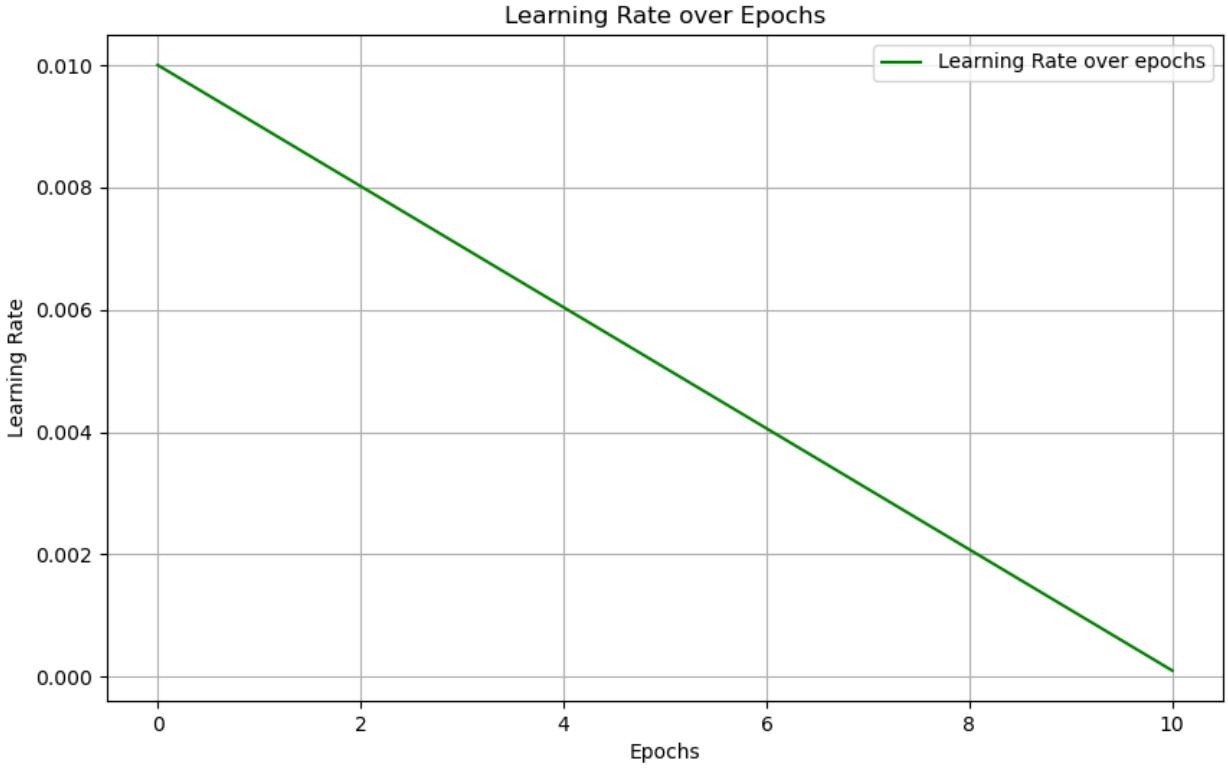
Alpha : 0.001

امتیازی اول:تغییر نرخ یادگیری

برای تغییر نرخ یادگیری میتوان از چندین روش استفاده کرد برای مثال میتوان آن را بر اساس MSE و یا epochs تغییر داد، ما بر اساس تغییر epochs این کار را انجام دادیم و تغییر به صورت خطی اتفاق می‌افتد، برای این کار یک learning rate نهایی و یک learning rate آغازین را قرار میدهیم، سپس در هر اپوک به طوری طول گام ها را کاهش میدهیم که در اثر گذشت epochs های داده شده لرنینت ریت آغازین به لرنینگ ریت نهایی تبدیل شود که این یک شیب کاهشی است.

امتیازی دوم: استفاده از توقف زود هنگام برای بهینه سازی

برای انجام دادن اینکار مهم ترین معیار که باید به آن توجه کرد MSE می‌باشد که برابر است با مربعات خطا ها و میتوان گفت که اگر این مقدار تغییرات از یک مقدار خیلی کوچک کمتر شد و یا بدتر از آن مقدار آن در epoch بعدی افزایش یافت یعنی زمان آن است که حلقه را بشکنیم و از آن بیرون بی‌آییم. برای این کار مقدار previous_mse را ساخته و مقدار 0 را در آن قرار میدهیم و سپس پس از محاسبه هر mse مقدار قبلی را با مقدار جدید مقایسه می‌کند و اگر مقدار از یک مقدار ثابت کوچک کمتر بود. از حلقه خارج می‌شود و مقادیر را بر می‌گرداند. در غیر این صورت به کار خود ادامه می‌دهد.



امتیازی سوم: استفاده از Lasso regularization

برای محاسبه کردن کردن regularization نوع اول باید $\lambda\sum_1^m |wi|$ را حساب کنیم و چون wi هرگز کمتر از 0 نیست و غیر ممکن است مقادیر ان منفی باشه پس نیازی به انجام دادن قدر مطلق نیست سپس پارامتر ها را مستقیم در عدد لاندا ضرب میکنیم وآنها را با gradiant جمع میکنیم.

```
gradient = np.dot(xi.T, np.dot(xi, parameters) - yi)
r1 = alpha * parameters
print(r1)
gradient += r1
parameters = parameters - learning_rate * gradient
```

تست برنامه و نتایج :

```
Epoch 0, MSE: 0.00041891179840586936
Epoch 1, MSE: 0.00041884308160584926
Epoch 2, MSE: 0.000418766885508848
Epoch 3, MSE: 0.00041867092462677293
Epoch 4, MSE: 0.0004185363516693552
Epoch 5, MSE: 0.00041843400022143558
Epoch 6, MSE: 0.0004180646866585709
Epoch 7, MSE: 0.00041772003187532926
Epoch 8, MSE: 0.0004173562878634854 6
Epoch 9, MSE: 0.0004169982340915812
R^2 Score: 0.8392214504996014
Mean Absolute Error: 0.016278497196258523

Process finished with exit code 0
```

R2 score : 0.8392

MSE : 0.016

نمودار تغییرات تابع هزینه بر حسب تعداد epoch