به نام خدا

تمرین چهارم مباحث ویژه

استفاده از رگرسیون لاجستیک چنددسته ای(تکنیک یکی در برابر همه)

و

استفاده از روش سافت مکس

استاد دولتشاهی

پریسا مبارک

40211415006

شرح تمرین:

میخواهیم با استفاده از دو روش رگرسیون لاجستیک چنددسته ای(تکنیک یکی در برابر همه) و سافت مکس ، دو مدل مختلف بر روی دیتاست هایی که در این تمرین داریم بسازیم و موارد زیر را انجام دهیم:

1) دقت را در دو روش مقایسه کنیم.

2) سرعت را در دو روش مقایسه کنیم.

روش رگرسیون لاجستیک چنددسته ای(تکنیک یکی در برابر همه) :

در تمرین دوم روش رگرسیون لاجستیک را پیاده سازی کرده بودیم. در این مثال دیتاست ما از چند کلاس(3کلاس که هرکدام یک نوع گل می باشد) تشکیل شده است. بنابراین از روش یکی در برابر همه استفاده میکنیم. در این روش برای هر کلاس بطور جداگانه ، یک مدل رگرسیون لاجستیک باینری آموزش داده می‌شود که بین آن کلاس و سایر کلاس‌ها تمایز قائل شود. در نهایت، برچسب جدید برابر با کلاسی قرار می‌گیرد که احتمال آن بیشترین احتمال بین تمام مدل‌های باینری است.

در ابتدا پس از وارد کردن توابع مورد نیاز و خواندن داده ها(مانند تمرینات قبل)،برای سهولت کار برچسب ها که بصورت رشته(نوع گل) میباشند را به اعداد صحیح تبدیل میکنیم:

```
# تبدیل نام کلاس‌ها(برچسب‌ها) به اعداد صحیح
df['variety'] = df['variety'].map({'Setosa': 1, 'Versicolor': 2, 'Virginica': 3})
```

در ادامه همانند تمرین دوم مراحل را انجام داده و توابع فرضیه و توابع مورد نیاز را مینویسیم .سپس مقدار دهی هارا نیز انجام میدهیم:

```
# مقداردهی اولیه پارامترها برای هر کلاس
alpha = 0.01
theta = np.random.randn(num_classes, xn_train.shape[1])
```

پس برای هرکلاس یک بردار پارامتر(به تعداد ویژگی ها و به علاوه یک(به علت عرض از مبدا)) داریم.بنابراین طبق کد بالا تتا را به صورت یک ماتریس تعریف کردم.

حال میخواهیم آموزش مدل را شروع کنیم. برای اینکار همانند کد های قبل یک تابع update_step داریم که کار آن یک مرحله آپدیت کامل هست. در واقع در این مثال ، این تابع برای تمام کلاس ها بطور کامل آپدیت پارامتر ها را انجام میدهد البته برای یک مرحله:

```
start_time = time.time() ############################## زمان شروع آموزش و تست
#آموزش مدل برای تمام کلاس ها
def update_step():
    global theta
    for cls in range(num_classes):
        y_train_cls = np.where(y_train == cls + 1, 1, 0)
        y_pred = h(xn_train, theta[cls,:])
        dtheta = np.zeros([xn_train.shape[1]])
        dtheta[0] = (y_pred - y_train_cls).mean()
        for i in range(1, xn_train.shape[1]):
            dtheta[i] = (xn_train[:, i] * (y_pred - y_train_cls)).mean()
        theta[cls,:] -= alpha * dtheta
```

تااینجا یک مرحله آموزش کامل صورت گرفته است.در ادامه باید تعداد دفعات آموزش را افزایش دهیم تا به تدریج آموزش اثر کند و دقت مدل نیز افزایش یابد. برای اینکار از یک حلقه استفاده میکنیم و در هرمرحله ، یک مرحله آموزش کامل تمام کلاس ها انجام میگیرد(با فراخوانی تابع آپدیت استپ) و سپس

دقت را محاسبه میکنیم. برای محاسبه دقت همانند کد زیر ابتدا برای داده های تستمان، خروجی پیش بینی شده را بدست می آوریم. به این صورت که برای هر کلاس ،داده های تستمان را به مدل میدهیم ومدل خروجی های پیش بینی شده را که بصورت احتمال هست برای داده های تست، میدهد(y_probs). بعد از اینکه این کار را برای تمام کلاس ها انجام دادیم،y_probs بصورت یک ماتریس بدست می آید که تعداد ستون های آن برابر با تعداد کلاس ها و تعداد سطرهایش نیز برابر با تعداد داده های تست هست.

در نهایت از احتمالات بدست آمده(ماتریس y_probs) برای هر داده تست(یعنی برای هر سطر)، argmax میگیریم. بااین کار برای هر داده تست(برای هر سطر)، کلاسی که بیشترین احتمال را دارد بدست می آوریم و در y_pred ذخیره میکنیم. بنابراین y_pred به عنوان خروجی های پیش بینی شده داده های تستمان بدست می آید.

در نهایت با استفاده از y_test که خروجی های واقعی داده های تستمان هست و y_pred ، دقت را محاسبه و در یک لیست ذخیره میکنیم. این کاررا برای 700 مرحله انجام دادم .

در خارج از حلقه نیز مقدار دقت نهایی را چاپ کردم و در ادامه زمان مدل را نیز حساب و چاپ کردم:

```
accuracy_list=[]
for i in range(700):
    update_step()
    y_probs = np.zeros((len(xn_test), num_classes))
    for cls in range(num_classes):
        y_probs[:, cls] = h(xn_test, theta[cls,:])
    y_pred_test = np.argmax(y_probs, axis=1)
    # محاسبه دقت مدل
    accuracy = np.mean(y_pred_test + 1 == y_test)
    accuracy_list.append(accuracy)
print("Accuracy:", accuracy)

end_time = time.time() ################################### زمان پایان آموزش و تست
test_training_time = end_time - start_time #### محاسبه زمان آموزش و تست
print("test_training_time:", test_training_time)
```

علت اینکه در کد بالا، پیش بینی و محاسبه دقت را در داخل حلقه انجام داد، این هست که میخوام در نهایت نمودار دقت را نمایش دهم:

```
# نمودار دقت مدل
plt.plot(accuracy_list, label='accuracy_list Set')
plt.xlabel('iteration')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

سپس نمودار خروجی پیش بینی شده و خروجی واقعی برای هر داده تست را نیز، نمایش میدهیم:

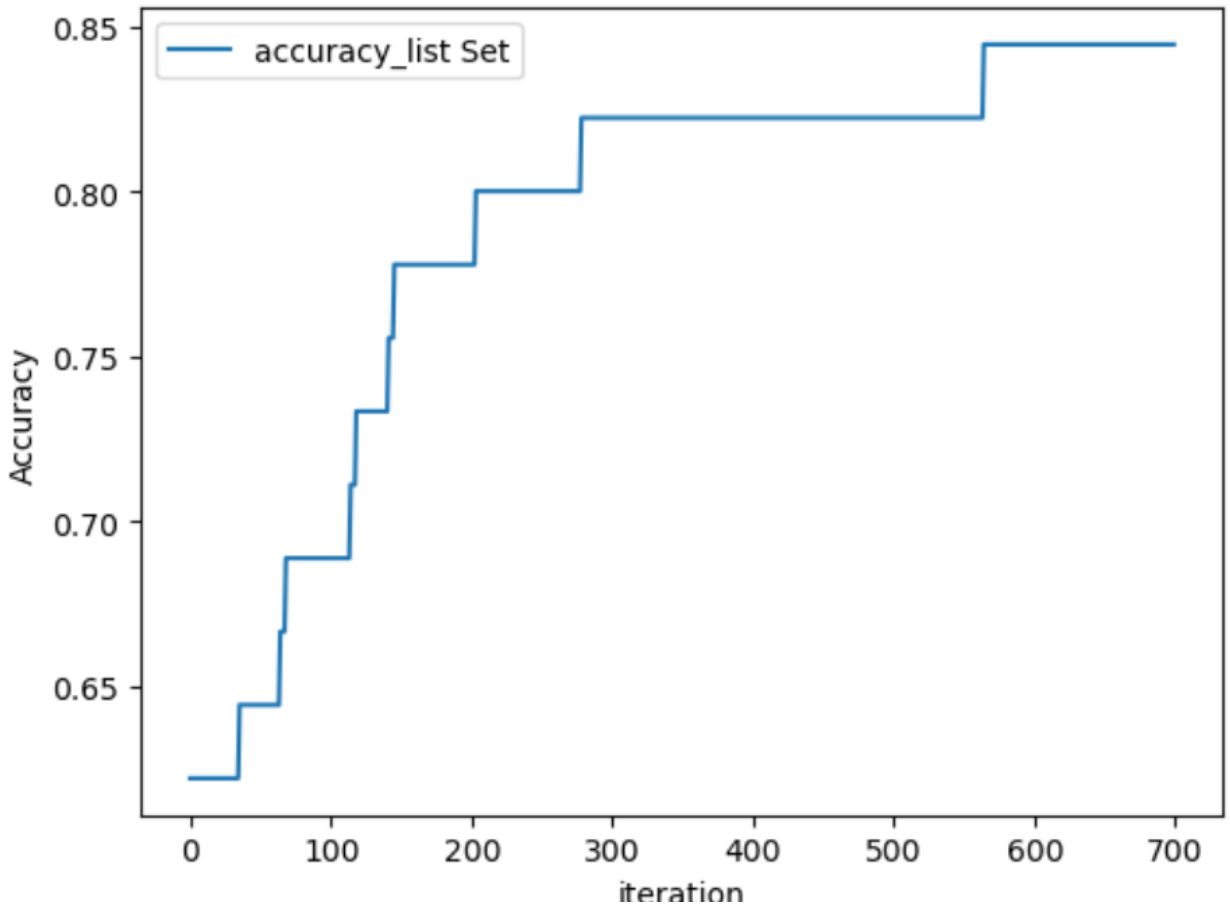
```
plt.scatter(range(len(y_test)), y_pred + 1, label='y_pred ', color='blue', marker='o', alpha=0.5)
plt.scatter(range(len(y_test)), y_test, label='y_test', color='red', marker='o', alpha=0.5)
plt.xlabel('Data Index')
plt.ylabel('Labels')
plt.legend()
plt.show()
```

خروجی های کد:

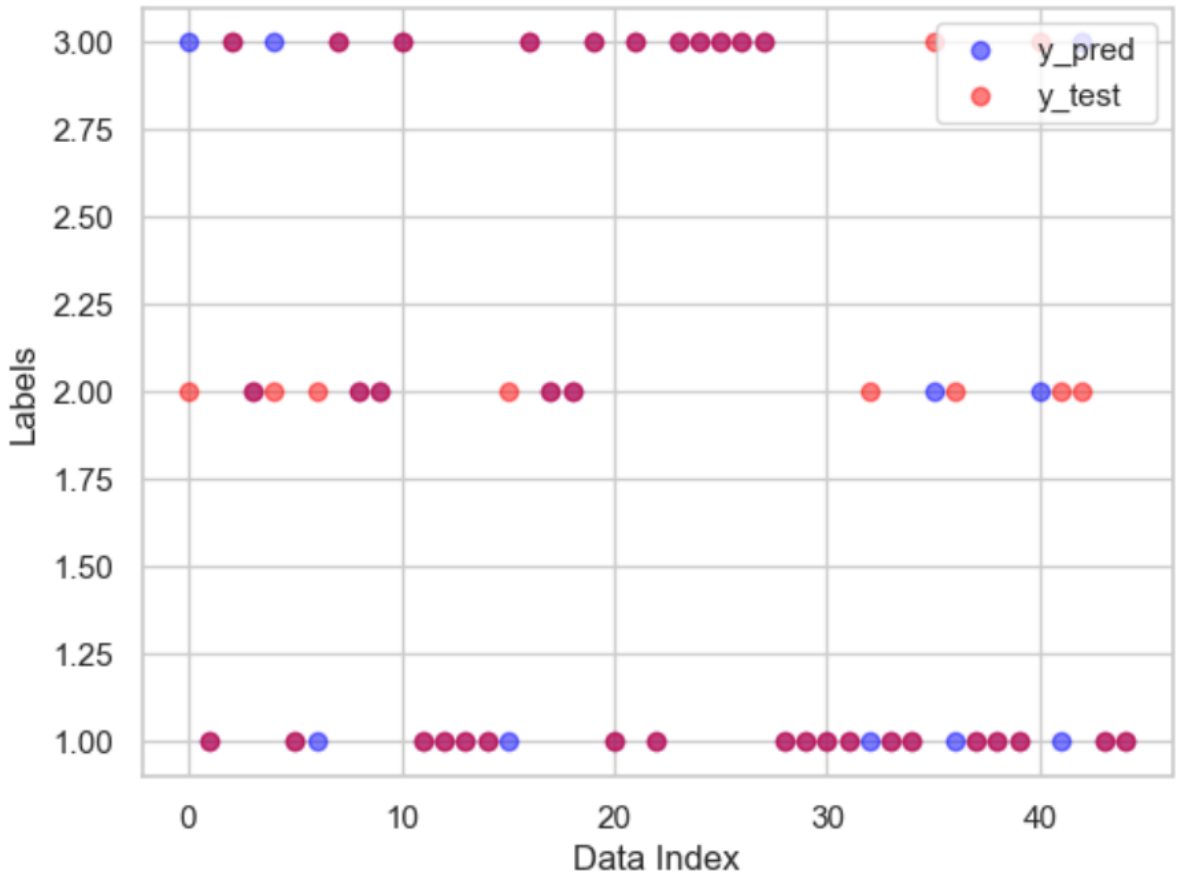
```
dataset:
      sepal.length  sepal.width  petal.length  petal.width    variety
0              5.1          3.5           1.4          0.2     Setosa
1              4.9          3.0           1.4          0.2     Setosa
2              4.7          3.2           1.3          0.2     Setosa
3              4.6          3.1           1.5          0.2     Setosa
4              5.0          3.6           1.4          0.2     Setosa
..             ...          ...           ...          ...        ...
145            6.7          3.0           5.2          2.3  Virginica
146            6.3          2.5           5.0          1.9  Virginica
147            6.5          3.0           5.2          2.0  Virginica
148            6.2          3.4           5.4          2.3  Virginica
149            5.9          3.0           5.1          1.8  Virginica

[150 rows x 5 columns]
Accuracy: 0.8444444444444444
test_training_time: 0.3247718811035156
```

طبق کد بالا دقت مدل 84 درصد میباشد که بسیار عالی هست.

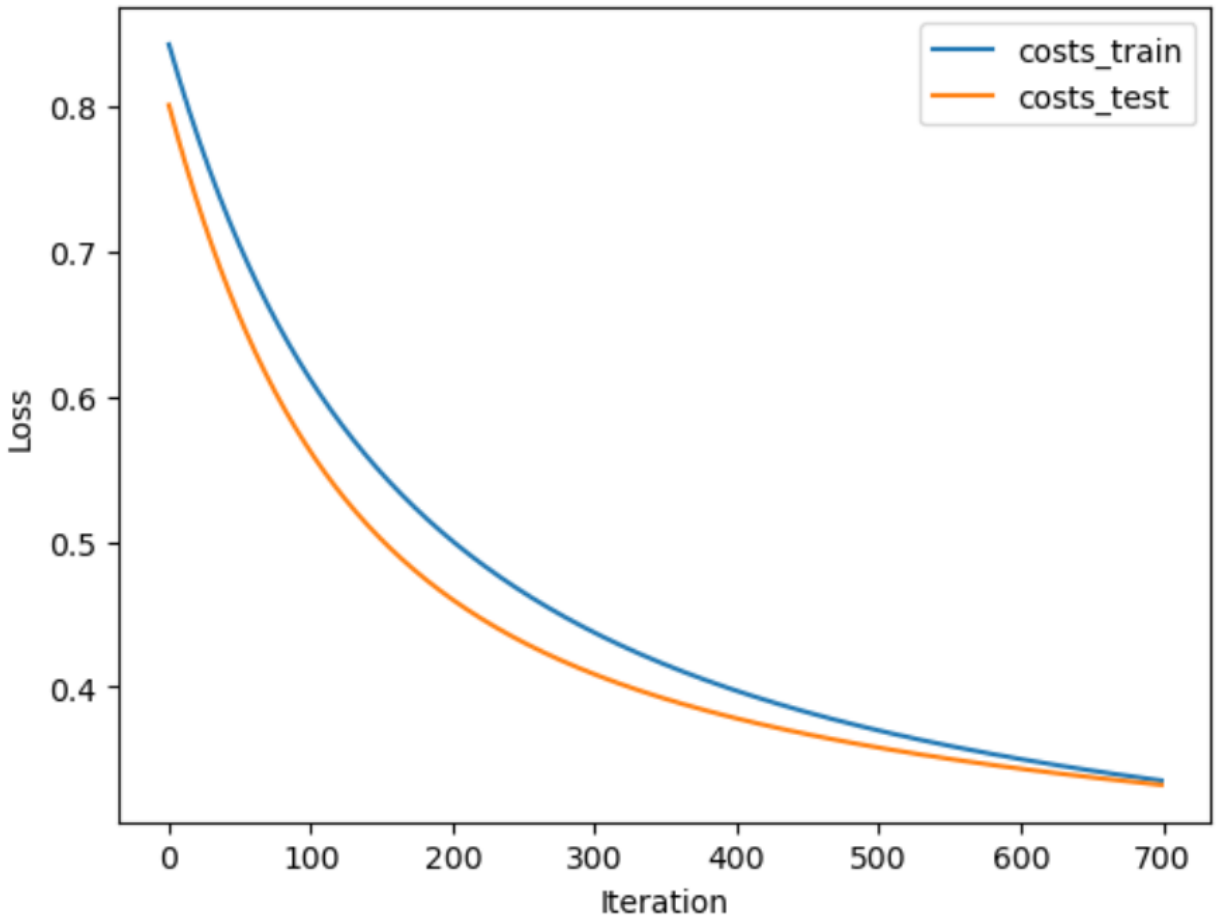
در ادامه نمودار دقت مدل به عنوان خروجی نمایش داده شده و مشاهده میکنیم که بعد از اتمام 700 مرحله دقت کاملا بالا میباشد:



نمودارزیر نیز مقایسه خروجی های پیش بینی شده و خروجی های واقعی برای داده های تست را نشان میدهد. در این نمودار،ستون افقی داده های تست و ستون عمودی خروجی های داده های تست(هم خروجی پیش بینی شده و هم خروجی واقعی) میباشد همچنین رنگ بنفش نشان دهنده ترکیب دو رنگ آبی کمرنگ(y_pred) و قرمز کمرنگ(y_test) میباشد و این به معنای روی هم قرا گرفتن دو رنگ هست یعنی خروجی پیش بینی شده و واقعی دقیقا یکی هستنند و در نتیجه یعنی برای آن داده تست، مدل خروجی را کاملا درست تشخیص داده است. بطور خلاصه رنگ بنفش نشاندهنده تشخیص درست مدل برای داده تست می باشد.



همچنین در صورت لزوم میتوانیم نمودار خطا را نیز نمایش دهیم:



روش سافت مکس :

الگوریتم softmax یک تابع فعال‌سازی است که بردار ورودی را به یک توزیع احتمالاتی که مجموع عناصر آن یک است تبدیل می‌کند. یعنی برای هر داده آموزشی، مجموع احتمالات پیش بینی شده به ازای هرکلاس، برابر یک میشود. برای تبدیل کد قبل به روش سافت مکس، باید تغییراتی در آن کد اعمال کنیم. ابتدا تابع فرضیه را تغییر میدهیم:

این تابع دو ماتریس را به عنوان ورودی میگیرد.ماتریس x ماتریسی هست با ابعاد:

(تعداد ویژگی ها*تعداد نمونه ها) و ماتریس theta ماتریسی هست با ابعاد:

(تعداد ویژگی ها*تعداد کلاس ها). خروجی تابع نیز، برای هر نمونه آموزشی، احتمالات پیش بینی شده به ازای هر کلاس را نشان میدهد. پس ابعاد خروجی: (تعداد کلاس ها*تعداد نمونه ها)

```python
# تابع فرضیه با روش Softmax
def h_softmax(X, theta):
    z = X.dot(theta.T)
    z -= np.max(z, axis=1, keepdims=True)  # برای جلوگیری از انفجار گرادیان
    exp_z = np.exp(z)
    return exp_z / np.sum(exp_z, axis=1, keepdims=True)
```

در ادامه تابع خطا را نیز باید به این صورت تغییر دهیم:

```python
# تابع خطا با روش Softmax
def softmax_loss(y_pred, y_true):
    m = len(y_true)
    log_likelihood = -np.log(y_pred[range(m), y_true-1] + 1e-10)  #لگاریتم احتمالات سافت مکس
    loss = np.sum(log_likelihood) / m
    return loss
```

در تابع خطای تصویر بالا، ماتریس y_pred (خروجی تابع فرضیه یعنی احتمالات پیش بینی شده برای تمام نمونه ها به ازای هر کلاس) و بردار y_true(برچسب های واقعی) را به عنوان ورودی میگیرد و خطا را محاسبه میکند. لازم به ذکر است در دستور y_pred[range(m), y_true-1]، با ترکیب این دو ترکیب اندیس، مقادیر مشخصی از y_pred که با اندیس‌های مشخص شده توسط range(m) و y_true-1 مطابقت دارند، استخراج می‌شوند. این مقادیر معمولاً به عنوان احتمالات پیش‌بینی شده برای برچسب‌های واقعی در مسائل دسته‌بندی استفاده می‌شوند.

در ادامه باید تابع آپدیت را تغییر دهیم. در این تابع در یک حلقه for برای هر کلاس،آپدیت پارامترها را انجام میدهیم.به این صورت:

برای هرکلاس ابتدا گرادیان به عنوان مجموع {تفاضل بین احتمالات پیش‌بینی‌شده برای آن کلاس و تابع نشانگر برای آن کلاس (یعنی اگر آن کلاس، برچسب واقعی باشد، 1 و در غیر این صورت 0)، ضرب شده در داده‌های آموزشی }محاسبه می‌شود. (درواقع عبارت y_train == k + 1 یک عبارت بولی هست که طول آن برابر با تعداد نمونه های آموزشی هست و بررسی میکند که آیا برچسب واقعی هرنمونه آموزشی، با آن کلاس برابر است یا خیر).پس بطور کلی در این تابع برای هرکلاس گرادیان محاسبه شده و پارامترهای آن کلاس از طریق گرادیان آپدیت میشوند.

```
# تابع به‌روزرسانی پارامترها با روش Softmax
def update_step_softmax():
    global theta
    y_pred = h_softmax(xn_train, theta)
    m = len(y_train)
    for k in range(num_classes):
            gradient = xn_train.T.dot(y_pred[:, k] - (y_train == k + 1))
            theta[k] -= (alpha/m ) * gradient
```

در ادامه همانند کد بخش اول مراحل زیر را برای 700 مرحله انجام میدهیم تا مدل به خوبی آموزش ببیند:

```
for i in range(700):
    update_step_softmax()
     # محاسبه خطاهای آموزشی و تست
    #costs_train.append(softmax_loss(h_softmax(xn_train, theta), y_train))
    #costs_test.append(softmax_loss(h_softmax(xn_test, theta), y_test))
    # پیش‌بینی خروجی دسته‌ها برای داده‌های تست
    y_probs = h_softmax(xn_test, theta)
    y_pred_test = np.argmax(y_probs, axis=1) + 1
    # محاسبه دقت مدل
    accuracy = np.mean(y_pred_test == y_test)
    accuracy_list.append(accuracy)
print("Accuracy:", accuracy)

end_time = time.time() ##################################### زمان پایان آموزش و تست
test_training_time = end_time - start_time #### محاسبه زمان آموزش و تست
print("test_training_time:", test_training_time)
```

طبق کد بالا در نهایت دقت و زمان محاسبه شده نیزچاپ میشود.

در ادامه نیز نمودار دقت و نمودار مربوط ب خروجی پیشبینی شده و خروجی واقعی داده های تست، را نمایش میدهیم.

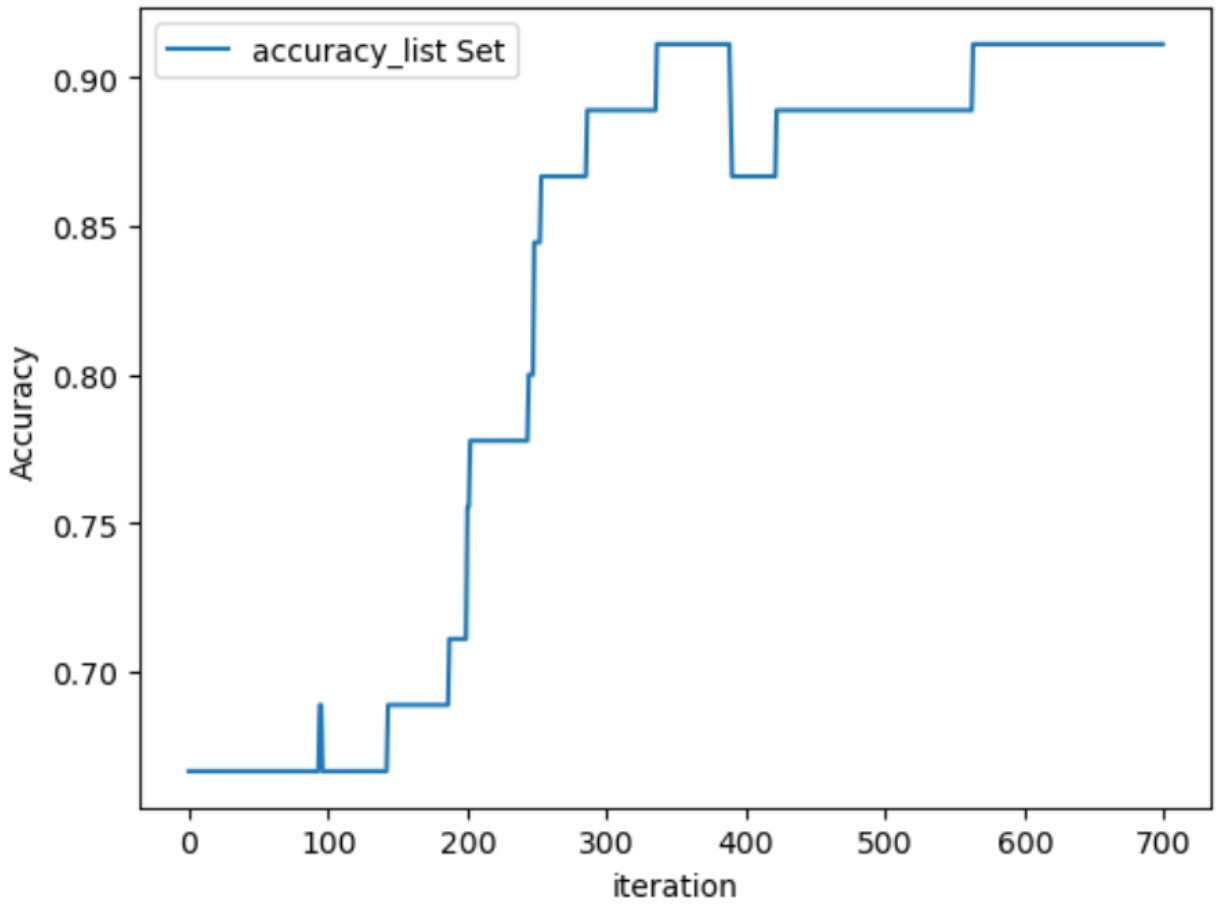
خروجی های این کد:

```
dataset:
      sepal.length  sepal.width  petal.length  petal.width     variety
0              5.1          3.5           1.4          0.2      Setosa
1              4.9          3.0           1.4          0.2      Setosa
2              4.7          3.2           1.3          0.2      Setosa
3              4.6          3.1           1.5          0.2      Setosa
4              5.0          3.6           1.4          0.2      Setosa
..             ...          ...           ...          ...         ...
145            6.7          3.0           5.2          2.3   Virginica
146            6.3          2.5           5.0          1.9   Virginica
147            6.5          3.0           5.2          2.0   Virginica
148            6.2          3.4           5.4          2.3   Virginica
149            5.9          3.0           5.1          1.8   Virginica

[150 rows x 5 columns]
Accuracy: 0.9111111111111111
test_training_time: 0.1530005931854248
```

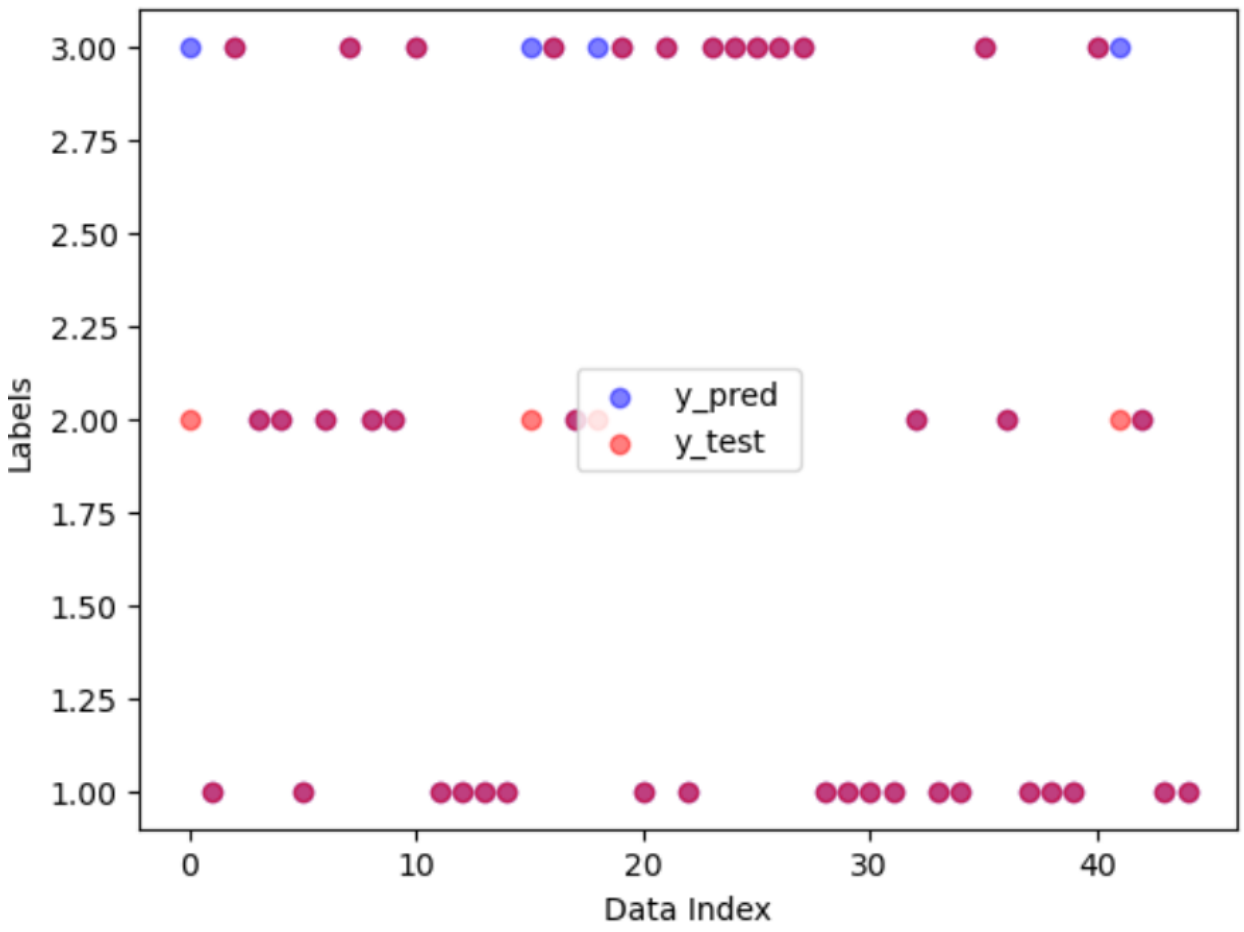
همانطور ک مشاهده میکنیم دقت و زمان از روش اول بهتره شده است.

هم چنین نمودار های خروجی در تصاویر زیر نمایش داده شده است.

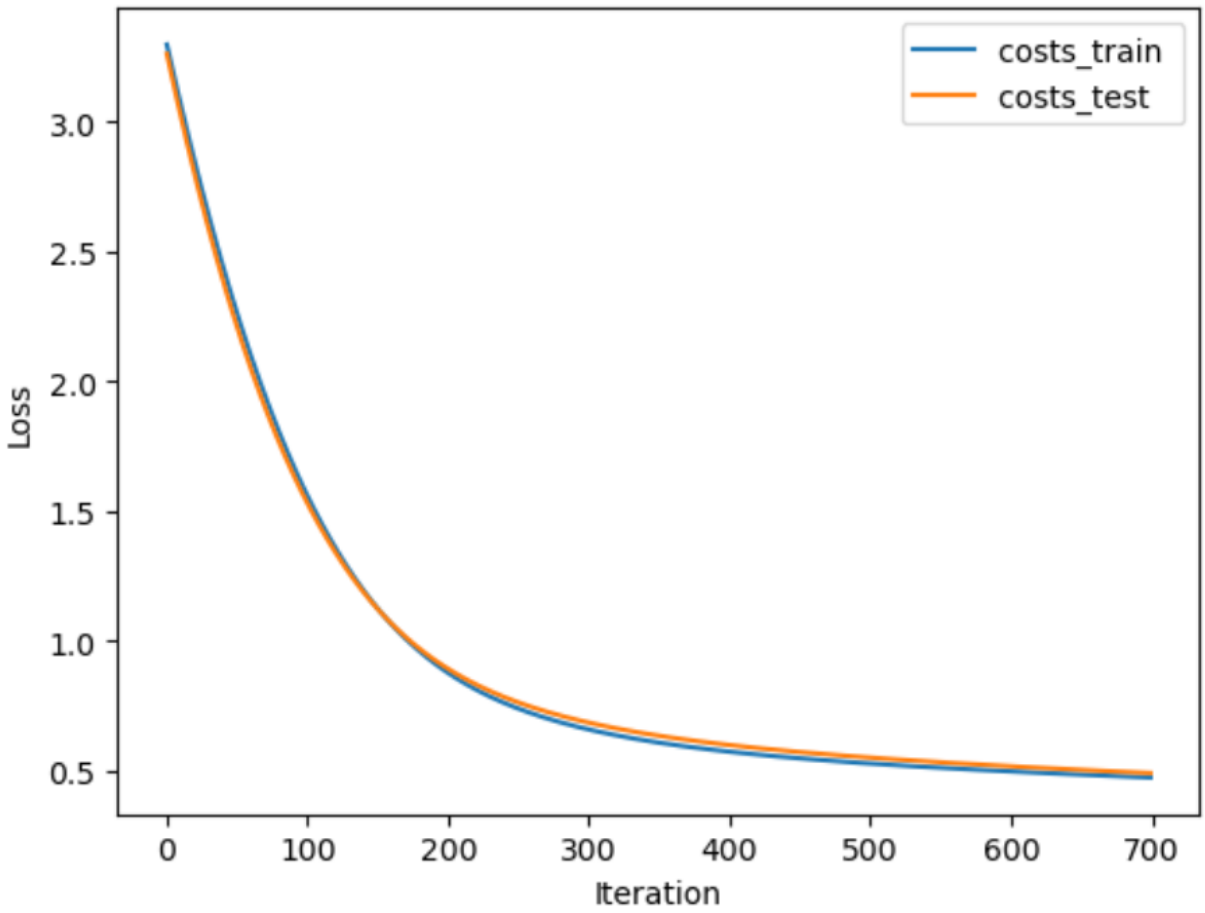
نمودار دقت روش سافت مکس:



نمودار مقایسه خروجی های پیش بینی شده و خروجی های واقعی برای داده های تست در روش سافت مکس:



همچنین میتوانیم درصورت لزوم نمودار خطا را نیز نمایش دهیم:



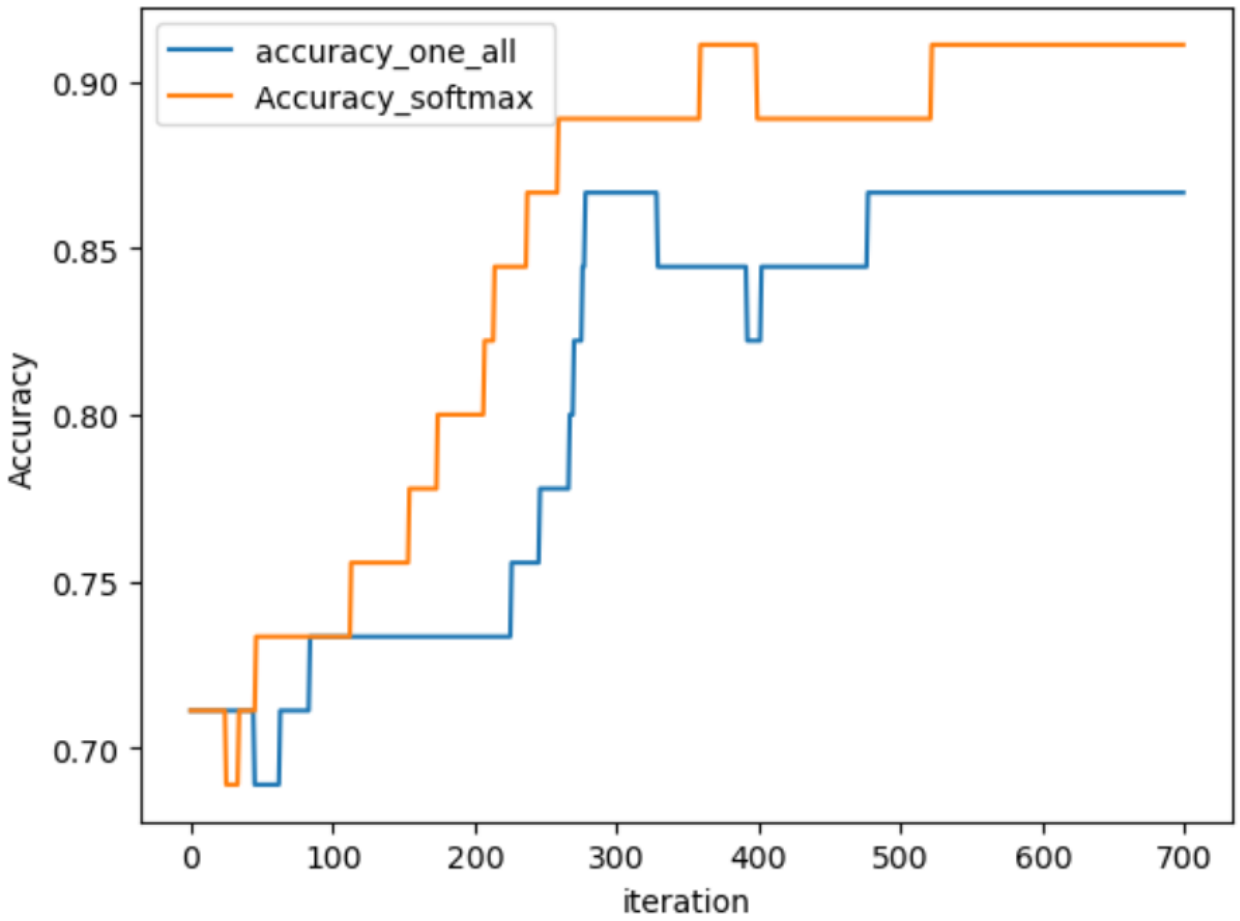
مقایسه دو روش:

برای مقایسه دو روش چون با هربار ران کردن برنامه، پارامترهای تصادفی متفاوتی بوجود می آیند، در کد سوم، دو کد را باهم ترکیب کردم و با یک بار ران کردن، نتایج و نمودار هر دو روش را هم زمان و در یک نمودار مشاهده میکنیم .

طبق خروجی زیر مشاهده میکنیم که روش سافت مکس عملکرد بهتری از لحاظ سرعت و دقت نسبت به روش رگرسیون لجستیک(روش یکی در برابر همه) دارد:

```
Accuracy_one_all: 0.8666666666666667
time_one_all: 0.3150143623352051
Accuracy_softmax: 0.9111111111111111
time_softmax: 0.16095542907714844
```

در تصویر زیر نیز نمودار دقت هر دو روش را مشاهده میکنیم:


accuracy_one_all
Accuracy_softmax
0.90
0.85
0.80
0.75
0.70
Accuracy
0
100
200
300
400
500
600
700
iteration