

## تحلیلی :

(1)

آرمان غفاریا  
99243056

① اگر فرض کنیم تعداد داده  $data$  set ما کم باشد ۱- می توان بر حجم آن افزود که با  $train$  های بیشتر قابلیت تعمیم پذیری بیشتری داشته باشد

۲- از  $Cross$   $Validation$  استفاده کنیم تا  $Validation$  ما بهبود پیدا کند و عملکرد بهتری داشته باشیم. مناسب  $splitting$

همچنین با در نظر گرفتن داده های تست متفاوت و جدا کردن آنها از داده های  $train$  می توان تعمیم پذیری مدل رگرسیون خطی را افزایش داد که بزرگ مدل داده خاص عادت نکند.

- مقدار درج مناسب هایپر پارامترها ← یک راه آزمون و خطا است که ببینیم مدل ما با مقادیر متفاوت هایپر پارامتر چطور عمل می کند و مقدار بهتر را قرار دهیم. همچنین می توان با روش های مثل جستجوی  $greedy$  هایپر پارامتر را تنظیم کرد.

- بررسی میزان  $overfit$  و املح آن هم کمک زیادی به تعمیم پذیری می کند

۲ معیار کاربردی هم  $mean squared error$  و  $Generalization$   $coefficient$  هستند

ساخته میزان  $mean squared error$  به عملکرد بهتر و تعمیم پذیری بهتر منجر می شود.

افزایش دادن مقدار ضریب تعمیم پذیری هم کمک می کند

۲ روی وجود دارد که بتوان روی دسته بندی دودویی را به حالت چندگانه تقسیم داد.

one-vs-one ← در این روی از بین دسته های موجود انتخاب های ۲ای ممکن را انجام می دهیم

برای هر کدام از آنها یک مدل دودویی آموزش داده می شود و بیشترین آن

به عنوان یک رأی در نظر گرفته می شود. مثلاً  
 دسته ۱ و ۲ ← رأی ۱  
 دسته ۱ و ۳ ← رأی ۲  
 ...

در نهایت بیشترین رأی پذیرفته و انتخاب می شود.

one-vs-rest ← در این روی برای هر دسته یک مدل دودویی آموزش می بیند به این معنا که یکسری در دسته موجود  
 جای گیرند سایرین در یک دسته دیگر

مدلی که بیشترین دقت را داشته باشد به عنوان بیشترین نهایی انتخاب می شود.

۵۷۵ ← مناسب برای وقتی که دیتای کمی برای آموزش داریم  
 ۵۷۷ ← مناسب برای وقتی که تعداد دسته ها زیاد است

۳ همانطور که در شکل مشخص است نمی توان یک روند خطی پیدا کرد که بر اساس داده های ورودی و مقادیر  
 بیشترین مناسب انتخاب دهد. (بهترین پیش به کمک خط ها قابل انجام نیست و اشکال غیر خطی اند)

اگر امارات زیادی به مدل خطی کنیم. ممکن است در اثر آموزش زیاد مدل دچار overfit شود  
 و برای رسیدن به دقت خوب فقط به همین مدل عادت کند و در داده های دیگر عملکرد بدی  
 داشته باشد.

در غیر اینصورت هم دقت مناسب نخواهیم داشت

④ در مواجهه با داده‌های نامتوازن (تعداد نمونه‌های مختلف بین دسته‌ها متفاوت است) می‌توان ارزیابی روش استفاده کرد تا مدل ما عملکرد بهتری داشته باشد

→ **Class weighting** در این روش به دسته‌های کم نمونه‌های کمتری دارند وزن بالاتری داده می‌شود تا تأثیر آن‌ها را در مدل ما داشته باشند.

- همچنین می‌توانیم از الگوریتم‌های مثل **Random Forest** استفاده کنیم که در مواجهه با داده‌های نامتوازن خوب عمل می‌کنند و تأثیر کلاس‌های کم حجم مناسب باشند.

- برای ارزیابی داده‌های نامتوازن از **Roc curve** استفاده کرد

- همچنین می‌توان داده‌ها را به حالت متوازن درآورد یعنی با نمونه‌های موجود در دسته‌های با نمونه کم نمونه اضافه شود یا از دسته‌های کم نمونه بیشتری دارند نمونه کم کرد.

↓  
(**duplication**) → با استفاده از این روش می‌توان نمونه‌ها را بیشتر کرد.

کدی :

(1)

الف) در کد انجام شد.

ب و پ)

```
[48] model = svm.SVC()           #create model
      model.fit(X_new, Y_new)

▼ SVC
SVC()

y_pred = model.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)      #find accuracy
print(accuracy)

0.7470046082949309

[50] y_pred = model.predict(x_test)
      f1 = f1_score(y_test, y_pred)           #find f1_score
      print(f1)

0.10294117647058824

cmatrix = confusion_matrix(y_test, y_pred)
print(cmatrix)                                # find confusion matrix

[[3179 1082]
 [ 16   63]]
```

آموزش داده شد.

مقدار accuracy برابر 74 درصد و مقدار f1\_score برابر با 10 درصد شد. همچنین مقادیر کانفیوژن ماتریس هم آورده شد.

$$\text{Precision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

$$F1 \text{ Score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

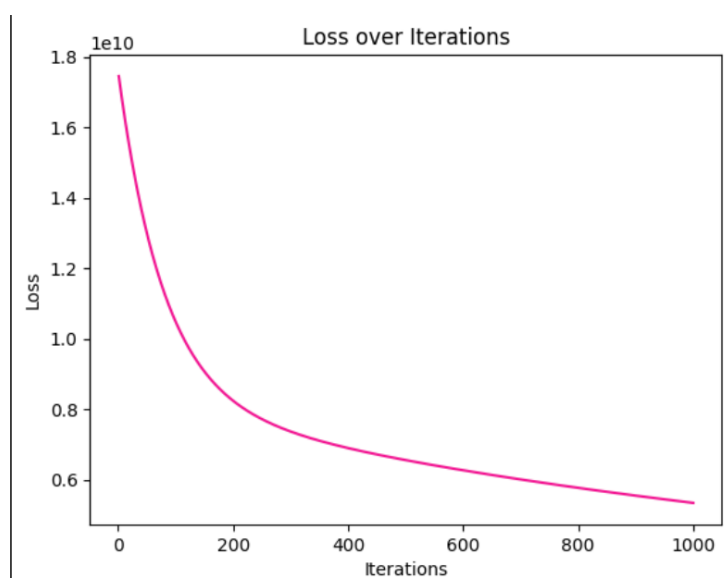
(2)

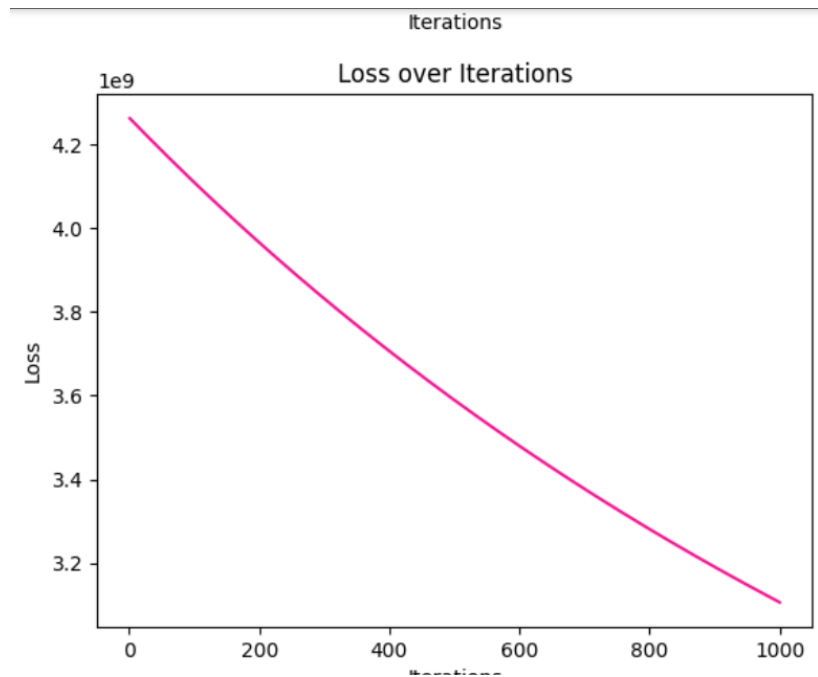
الف) در کد انجام شد.

ب)

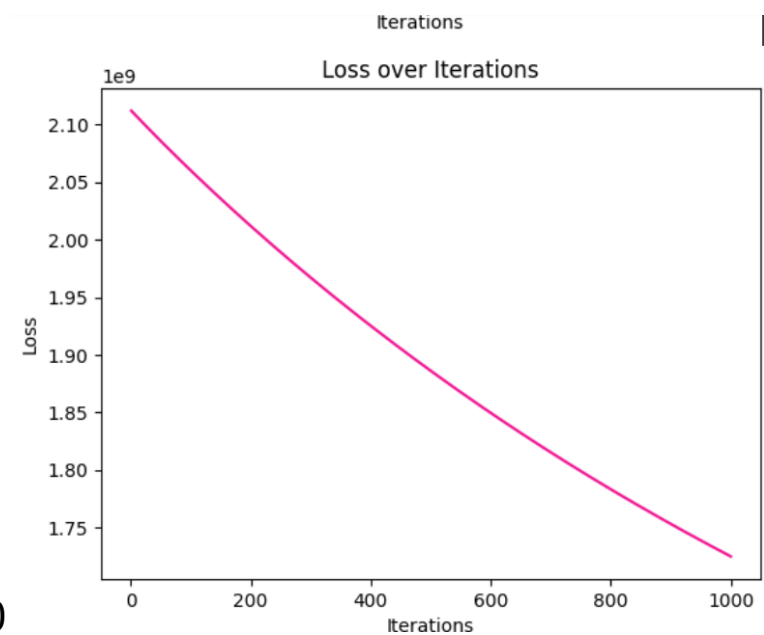
نمودار loss را برای درصد اسپلایت های متفاوت رسم میکنیم

10 درصد داده تست





30 درصد داده تست



60 درصد داده تست

مشاهده میشود که در 10 درصد داده تست هم نرخ کاهش بیشتری دارد هم در نهایت به مقدار کمتری رسیده است (به نسبت cost اولیه) پس عملکرد بهتری دارد.



(پ)

پیاده سازی در کد انجام شد . دقت را با r-square error گزارش میکنیم

```
▶ predict = np.dot(x1_test, W) + B
  r_square = r2_score(y1_test, predict)    # find r-square error
  print(r_square)
```

```
⇒ 0.6145017246529894
```

$$R^2 = 1 - \frac{\text{Sum of Squared Residuals}}{\text{Total Sum of Squares}}$$

where:

- Sum of Squared Residuals (SSR) is the sum of the squared differences between the actual values and the predicted values.
- Total Sum of Squares (SST) is the sum of the squared differences between the actual values and the mean of the actual values.

(ث)

```
data_set['region_0_2'] = data_set["region_0"]**2
data_set['region_1_2'] = data_set["region_1"]**2
data_set['region_2_2'] = data_set["region_2"]**2
data_set['region_3_2'] = data_set["region_3"]**2

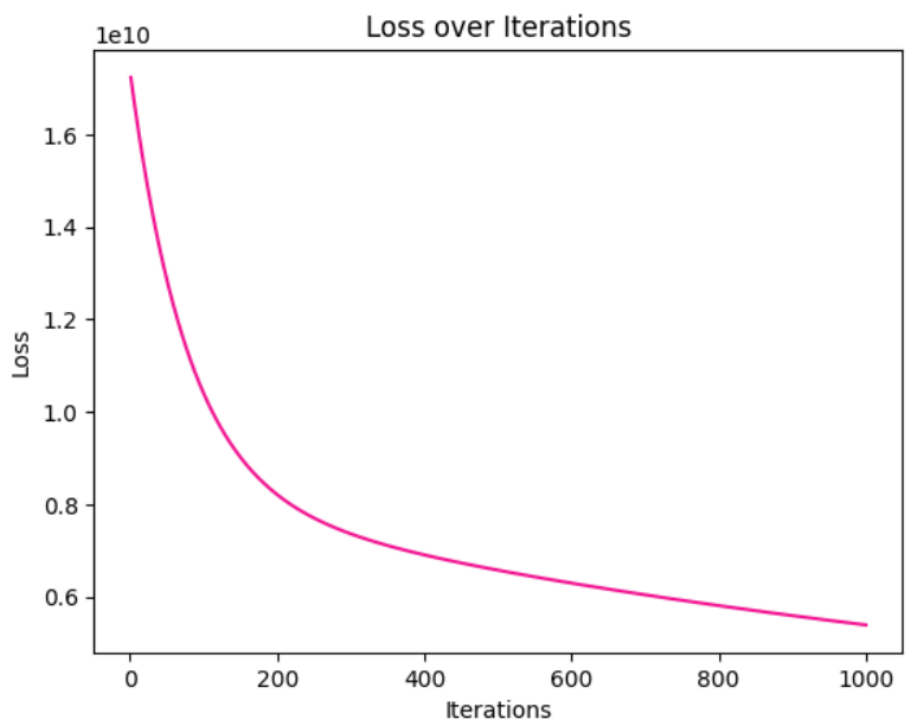
data_set['smoker_0_2'] = data_set["smoker_0"]**2
data_set['smoker_1_2'] = data_set["smoker_1"]**2    # add polonomials

data_set['sex_0_2'] = data_set["sex_0"]**2
data_set['sex_1_2'] = data_set["sex_1"]**2

data_set['bmi2'] = data_set["bmi"]**2

data_set['children2'] = data_set["children"]**2
```

فیچر های دیتا را به توان 2 رسانده و به دیتا اضافه میکنیم.



مشاهده میشود مجدد **loss** فانکشن نزولی است و نرخ کاهش خوبی دارد و به مقدار خوبی در نهایت رسیده است (به نسبت **cost** اولیه)

```
▶ predict = np.dot(x1_test, W) + B
r_square = r2_score(y1_test, predict)    # find r-square error
print(r_square)
```

0.6518596626251604

همچنین دقت مدل در حالت چندجمله ای بهتر شد.