FEATURE SELECTION

Computational Intelligence

Hamed Masoudi

9922102135

Kourosh Sherkat

9912102638

June 15, 2023

Ferdowsi University of Mashhad (F.U.M)
Faculty of Engineering

بخش اول:

در بخش اول یک مدل SVM با تمام فیچر ها ترین می کنیم. ست کردن هایپر پارامتر ها برای این مدل خیلی مهم است که صحیح باشد چون در ادامه قصد داریم پس از فیچر سلکشن از این مدل استفاده کنیم برای ارزیابی عملکرد مان.

بعد از بررسی کرنل های مختلف و ضریب گاما و C به نتیجه زیر رسیدیم که بهترین دقت را برای کلسیفای کردن دیتا داشت.

```
model0 = SVC(C = 2.2 , gamma = 'scale' , kernel = 'rbf' , verbose = 1)
model0.fit(train_data , train_labels)
```

ما کرنل را بر روی rbf قرار دادیم و ضریب C را برابر با 2.2 قرار دادیم با این هایپر پارامتر ها به دقت زیر رسیدیم:

```
np.random.seed(123)
accuracy_score(test_labels , model0.predict(test_data)) ,
accuracy_score(train_labels , model0.predict(train_data))
```

برای محاسبه دقت مدل ترین شده را بر روی داده های ترین و داده های تست با توجه به کد بالا ارزیابی می کنیم. توجه داریم که دقت مهم برای ما دقت مدل بر روی داده های تست هست و نه داده های ترین.

دقت مدل ما بر روی داده های تست 0.90 بود و دقت مدل بر روی داده های تست 0.99 بود. این بهترین دقتی بود که ما توانستیم با مدل SVM بدست بیاریم.

ماتریس گمراهی برای هایپر پارامتر هایی که ما ست کردیم بصورت زیر است:

```
array([[458,   4,   2,   0,   2,   1,   7,   2,   4,   3],
       [  4, 491,  15,   0,   5,   0,   0,  21,   2,   1],
       [  1,   8, 511,   2,   6,   0,   1,   5,   3,   2],
       [  7,   3,  11, 428,   4,  13,   2,   2,  12,   2],
       [  3,  17,   2,   1, 457,   1,   0,   4,   2,   3],
       [  4,   2,   4,  28,   2, 452,  17,   5,   4,   8],
       [ 11,   1,   4,   4,   7,  10, 436,   0,   7,   3],
       [  3,  18,   5,   0,   1,   1,   0, 478,   0,   0],
       [ 13,   6,   2,  20,   3,   8,  23,   2, 386,   8],
       [ 14,   4,  10,   4,  12,   3,   2,   1,   5, 424]])
```

شکل 1. ماتریس گمراهی برای هایپر پارامتر های اصلی

مشخص است که مدل به خوبی تنظیم شده است و هایپر پارامتر ها درست ست شده اند چون در هر کلاس مقدار قابل توجهی را مدل درست پیش بینی کرده است.

اگر کرنل را ما بجای rbf بر روی Linear قرار بدهیم چون مدل ساده تر می شود دقتش بر روی داده های تست هم پایین تر می آید. دقت مدل با کرنل Linear بر روی داده های تست برابر با 0.85 است.

ماتریس گمراهی برای مدل SVM با کرنل خطی بصورت زیر است:

```
array([[449,   7,   1,   1,   5,   1,  13,   1,   4,   1],
       [  8, 467,  14,   2,  14,   0,   3,  28,   2,   1],
       [  6,   7, 500,   1,   8,   1,   4,   2,   7,   3],
       [  4,   2,  10, 402,   6,  26,   6,   2,  15,  11],
       [  5,  27,   2,   4, 437,   0,   7,   1,   4,   3],
       [  6,   2,   6,  32,   2, 438,  21,   6,   4,   9],
       [ 15,   1,   2,   9,  13,  24, 401,   1,  13,   4],
       [  5,  23,  10,   3,   3,   1,   0, 457,   3,   1],
       [ 18,   5,   5,  48,   7,  16,  18,   1, 344,   9],
       [ 17,   7,  14,  11,  14,   2,   2,   1,  12, 399]])
```

شکل 2. ماتریس گمراهی برای مدل SVM با کرنل خطی

قابل مشاهده است که عملکرد مدل ضعیف تر شده است به عنوان مثال برای کلاس اول مدل 449 دیتا را درست پیش بینی کرده است در صورتی که با کرنل rbf مدل 458 درست پیش بینی کرده بود این نشان می دهد که ما نباید از کرنل linear استفاده کنیم.

بخش دوم:

در این بخش مرحله ی انتخاب ویژگی را آغاز خواهیم کرد. برای این کار مراحل زیر را طی خواهیم کرد:

1. شبکه را ران کرده و سپس تعدادی از فیچرهای خوب (در این قسمت 25 فیچر برتر برداشته می شوند) را جدا می کنیم
2. مجددا شبکه را با فیچرهای باقی مانده آموزش می دهیم و مراحل قبل را تکرار می کنیم
3. این کار را آنقدر ادامه بدهیم که معیار score به بیشترین حد خود برسد

شبکه ی استفاده شده یک شبکه ی Fully Connected می باشد که پارامترهای hidden_size و reg_l1_param آن به شرح زیر می باشد: Hidden_size = 2048 , Reg_l1_param = 1.05e-4

برای انجام مراحل ذکر شده یک تابع حاوی شبکه می سازیم تا ورودی های لازم ، سایز ورودی را دریافت کند تا هر مرحله تغییر را بتواند کنترل کند. تابع ذکر شده به صورت زیر خواهد بود:

```
def training(train_data, train_labels,test_data, test_labels ,index):
  input_tensor = tf.keras.Input(shape=(index.shape[0],))
  hidden_size = 2048
  reg_l1_param = 1.05e-4
  hidden_layer_1 = tf.keras.layers.Dense(units=hidden_size,
activation=tf.nn.relu,
```

```
                                        name = 'hidden_layer',
                                        activity_regularizer=tf.keras.regu
larizers.l1(reg_l1_param))(input_tensor)

  output_layer = tf.keras.layers.Dense(units=10, activation=tf.nn.softmax,
name = 'classification_layer')(hidden_layer_1)

  model = tf.keras.Model(inputs=input_tensor, outputs=output_layer)
  model.compile(optimizer = 'adam',
loss='sparse_categorical_crossentropy', metrics=['accuracy'])
  history = model.fit(train_data, train_labels, epochs=10, batch_size=32 ,
validation_data=(test_data, test_labels))
  test_loss, test_acc = model.evaluate(test_data, test_labels)
  arg_max1 =
np.argmax(np.mean(model.get_layer('classification_layer').get_weights()[0]
, axis = 1))
  x =
np.sort(model.get_layer("hidden_layer").get_weights()[0][:,arg_max1])[::-
1]
  captured =
index[np.squeeze(np.array([np.argwhere(model.get_layer("hidden_layer").get
_weights()[0][:,arg_max1] == i) for i in x[:25]]))]
  return captured
```

کد بالا ابتدا دیتای مورد نیاز (که بسته به اینکه در کدام مرحله از انتخاب فیچر هستیم ممکن است تعداد فیچرهای آن فرق کند) و تعداد فیچرهای دیتا را دریافت می کند و ورودی شبکه را بر اساس *index.shape[0]* می سازد.

سپس دیتای دریافتی را به شبکه اصلی فید می کند و مدل شبکه را می سازد. حال در مرحله ی بعد، با استفاده از معیار میانگین ،

میانگین وزن های متصل از هر نورون در لایه ی *hidden_layer* به لایه ی *classification_layer* را محاسبه می کنیم و با مقایسه ی تمام 2048 نورون موجود در لایه ی *hidden_layer* شماره ی بهترین نورون را استخراج می کنیم. کد زیر این کار را انجام می دهد:

```
arg_max1 =
np.argmax(np.mean(model.get_layer('classification_layer').get_weights()[0]
, axis = 1))
```

پس از مشخص شدن بهترین نورون در hidden_layer ، وزن های نورون های ورودی متصل به آن را به ترتیب مرتب کرده وشماره ی 25 تا از بهترین فیچرها را جدا کرده و در متغیر captured ذخیره می کنیم. کد زیر کار گفته شده را انجام می دهد:

```
  x =
np.sort(model.get_layer("hidden_layer").get_weights()[0][:,arg_max1])[::-
1]
```

```
  captured =
index[np.squeeze(np.array([np.argwhere(model.get_layer("hidden_layer").get
_weights()[0][:,arg_max1] == i) for i in x[:25]]))]
```

در نهایت شماره ی این فیچر ها را تحت متغیر captured باز می گردانیم.

اما همانطور که قبلا ذکر شد باید در هر مرحله فیچرهای خوب جدا شده و باقی فیچر ها ترین شوند و این مراحل ادامه داشته باشد.کد زیر این کار را انجام می دهد:

```
index = np.arange(1024)
g_features = list()
captured = list()
for i in range(10):
  index = np.setdiff1d(index , captured)
  captured = training(train_data[:,index], train_labels
,test_data[:,index], test_labels , index)
  g_features.extend(captured)
```

در این کد،ابتدا یک لیست به نام index ساخته می شود که شماره ی تمام فیچر ها در آن قرار دارد.سپس یک لیست دیگر به نام g_features ساخته می شود که در هر مرحله فیچر های مناسب آن مرحله در آن قرار داده می شود.یک متغیر آغازین برای captured در نظر گرفته می شود تا شروعی بر کد باشد.یک حلقه ی for که بیانگر تعداد دفعاتی است که الگوریتم اجرا می شود تشکیل شده است.در این حلقه،در هر مرحله،شماره ی فیچر های خوب از لیست index توسط تابع setdiff1d حذف می شود تا باقی فیچر ها به داخل شبکه feed شوند و عملیات استخراج فیچر مجددا اجرا شود.در خط بعدی ترین اتفاق می افتد و شماره ی بهترین فیچر های آن مرحله به لیست g_features اضافه می شود. این حلقه 10 مرتبه تکرار می شود و با احتساب برداشت 25 فیچر در هر مرحله، سر انجام 250 فیچر به صورت کلی استخراج می کند.

برای محاسبه ی score از کد زیر استفاده خواهیم کرد:

```
model3 = SVC(C = 2.2, gamma = 'scale' , kernel = 'rbf')
print("Score" , "    Accuracy","    Features" , "\n")
for i in range(10):
  model3.fit(tf.squeeze(train_data[:,g_features[:((i+1)*x)]]),
train_labels)
  accuracy = accuracy_score(test_labels ,
model3.predict(tf.squeeze(test_data[:,g_features[:((i+1)*x)]])))
  score = accuracy - ((i+1) * x - x ) * 0.00075
  print("{:.4f}".format(np.round(score , 4)) ,"    ",
"{:.3f}".format(np.round(accuracy , 3)) ,"      ", (i+1)*x)
```

در این تکه کد ابتدا مدل اصلی که در بخش قبلی توضیح داده شد را مجددا با همان پارامتر های قبلی ایجاد می کنیم.سپس در هر مرحله 25 تا به فیچر های خود اضافه و مدل را با آن ترین می کنیم.در خط بعدی دقت مدل را بدست می آوریم و در خط بعدی score را محاسبه می کنیم:

$$score = accuracy - (n \times 25) \times 0.00075$$

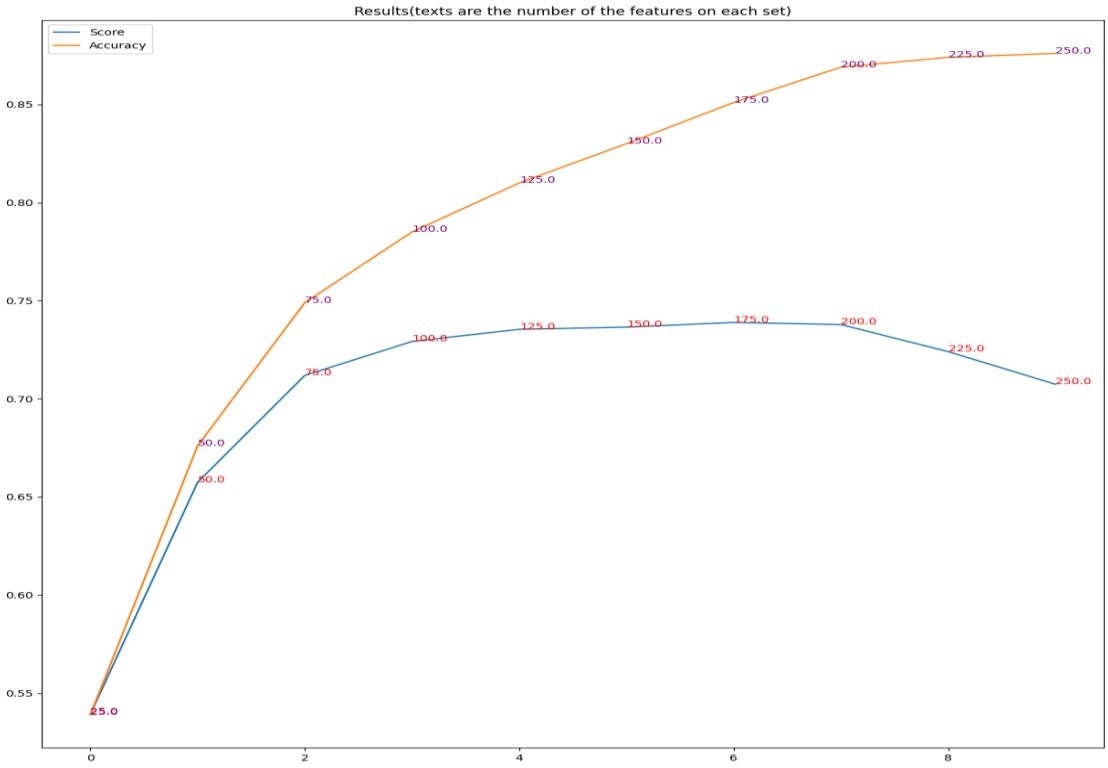
پس از محاسبه ی score، با مقایسه ی مقادیر آن در هر مرحله با تعداد فیچرهای مختلف ، مقدار max آن را پیدا می کنیم و تعداد فیچر مربوط به آن را بدست می آوریم. این مقدار همان تعداد فیچرهای ایده آل پس از استخراج فیچر می باشد.

نتایج بدست آمده برای این بخش به صورت زیر می باشد:

Score	Accuracy	Features
0.5392	0.539	25
0.6574	0.676	50
0.7119	0.749	75
0.7292	0.785	100
0.7354	0.810	125
0.7365	0.830	150
0.7389	0.851	175
0.7378	0.869	200
0.7240	0.874	225
0.7074	0.876	250

بنابراین بهترین حالت score در اینجا 0.7389 می باشد که 175 فیچر حاصل شده است.پس بهترین تعداد فیچر 175 تا است.

برای درک بهتر این داده ها را ما بصری سازی کردیم بصورت زیر:



شکل 3. بصری سازی امتیاز و دقت

دقت بدست آمده پس از اعمال انتخاب فیچر پایین تر از مقدار دقت اولیه(با تمام فیچرها) شده است اما نشان دهنده آن است که با 175 فیچر میتوان 85.5 درصد دقت را کسب کرد و بار محاسباتی مدل را کاهش داد اما به دلیل اهمیت همه ی فیچرها و correlation بین آنها دقت نهایی با دقت بدست آمده 5 درصد اختلاف دارند.

بصورت کلی ما توانستیم از بین 1024 فیچر 175 فیچر مهم را بدست بیاریم که دقت آن نسبت به حالتی که1024 فیچر داشتیم فقط 5 درصد کمتر است این کاملا ارزشمند است که ما 849 فیچر را استفاده نکنیم و همچنان دقتمان آنچنان تفاوت قابل توجهی نسبت به حالت اول که از 1024 فیچر استفاده می کردیم نداشته باشد.