به نام خدا

گزارش فاز سوم

پروژه هوش محاسباتی

پوریا ناظمی و طهورا سعیدی

بخش اول

در بخش اول به دنبال این هستیم مدل svm مناسبی برای طبقه بندی داده ها که تصاویری از اعداد یک رقمی باشند، بسازیم.
برای انتخاب هایپرپارامتر ها، ابتدا kernel مناسب را یافتیم.

از بین کرنل های قابل اجرای زیر، دقت مدل خام در کرنل rbf ، بیشتر بدست آمد. از این رو این کرنل را انتخاب کردیم.

```
### kernel selection

kernels = ['rbf', 'sigmoid', 'poly', 'linear']
for each_kernel in kernels:
  clf = svm.SVC(kernel = each_kernel)
  clf.fit(train_data, train_labels)
  predicted_labels = clf.predict(test_data)
  print(accuracy_score(test_labels, predicted_labels))
```

```
0.899
0.853
0.8662
0.8588
```

پس از انتخاب این کرنل، بایستی دو پارامتر اصلی C و gamma مقداردهی کنیم و مقدار مناسب را بیابیم. پارامتر C میزان تاثیر پنالتی طبقه بندی اشتباه یک داده را مشخص میکند که اگر میزان پنالتی در نظر گرفته شده بیشتر باشد، مدل سعی میکند کمتر داده ای را در طبقه بندی دیگری قرار دهد. Trade off این فرآیند بر سر over fit شدن است.

میزان gamma اگر کم باشد overgeneralized و اگر زیاد باشد overfit اتفاق میفتد.

توضیح پارامتر C و تاثیر آن: لینک

توضیح پارامتر gamma و تاثیر آن: لینک

با استفاده از grid search میزان مناسب برای این دو پارامتر را می یابیم:

```
c = [1,2,3,4,5,10,15,20,30]
gammas = [0.01,0.05,0.1,0.2,0.3,0.4,0.5,0.8,'scale']

clf = svm.SVC(kernel = 'rbf')
param_grid = dict(C=c, gamma=gammas)
grid = GridSearchCV(clf, param_grid, cv=3, n_jobs=-1)
grid.fit(X_train, y_train)
print(grid.best_params_)
```

```
{'C': 10, 'gamma': 'scale'}
```

با در نظر گرفتن gamma = scale میزان گاما توسط خود مدل با توجه به scale داده ها انتخاب میشود.

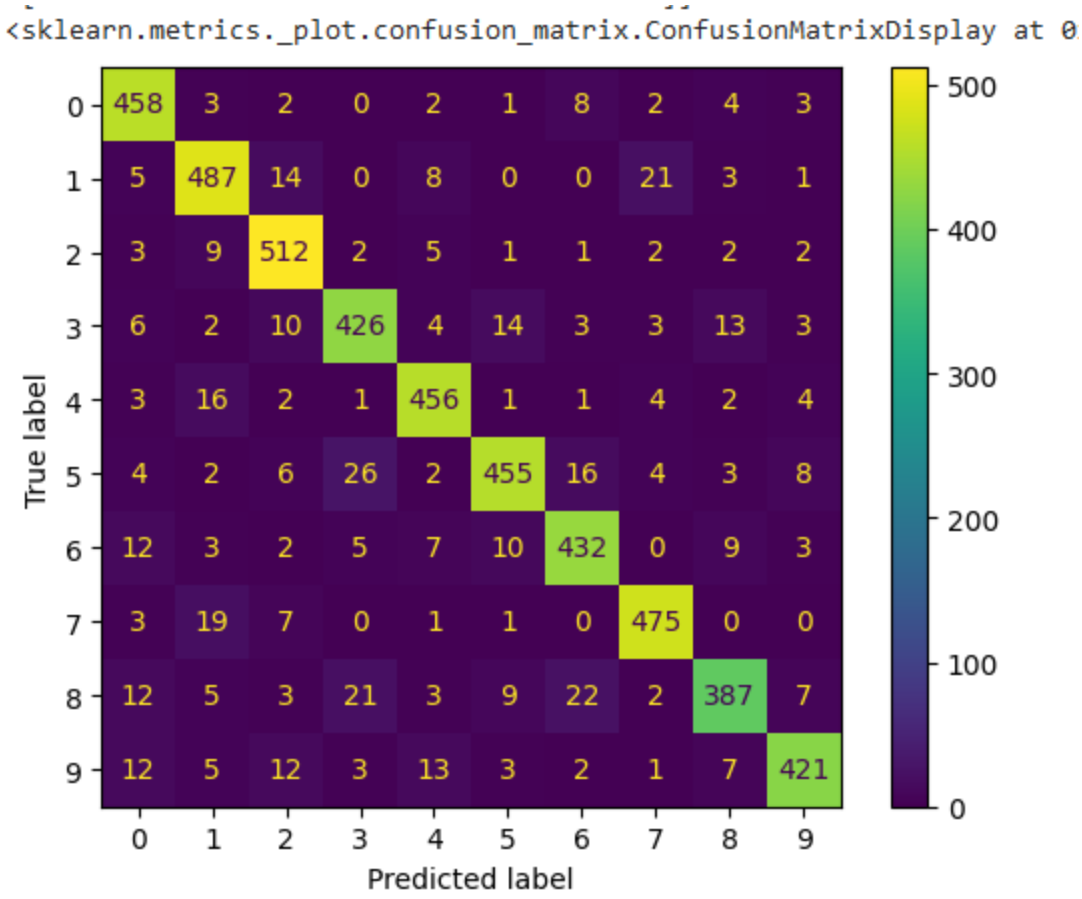
مدل را با پارامترهای گفته شده روی داده ها اجرا میکنیم. نتایج ارزیابی و ماتریس گمراهی به شرح زیر است:

```
clf = svm.SVC(kernel ='rbf', C = 10, gamma = 'scale')
clf.fit(train_data, train_labels)
```

```
SVC
SVC(C=10)
```

```
predicted_labels = clf.predict(test_data)
print(recall_score(test_labels, predicted_labels, average="micro"))
print(accuracy_score(test_labels, predicted_labels))
```

```
0.9018
0.9018
```



بخش دوم

در بخش دوم ابتدا باید پارامتر های شبکه را tune کنیم. یک پارامتر تعداد نورون های لایه میانی است و پارامتر دیگر l1_regularization است که برای جلوگیری از overfit است.

توضیح پارامتر l1_regularization و تاثیر آن: لینک

با استفاده از grid search مقادیر مناسب برای این دو پارامتر را می یابیم:

```
input_tensor = tf.keras.Input(shape=(1024,))
accuracy = []
hidden_sizes = list([24,32,64,100,70,80,90,120,150,200])
reg_l1_params = list([10e-6, 10e-5, 10e-4, 10e-3])
for hidden_size in hidden_sizes:
  for reg_l1_param in reg_l1_params:

    hidden_layer_1 = tf.keras.layers.Dense(units=hidden_size, activation=tf.nn.relu,
                                           name = 'hidden_layer',
                                           activity_regularizer=tf.keras.regularizers.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,verbose=0, batch_size=32, validation_data=(test_data, test_labels))
    test_loss, test_acc = model.evaluate(test_data, test_labels)
    accuracy.append({'reg_l1_param': reg_l1_param, 'hidden_size' : hidden_size, 'acc':test_acc})
    print('reg_l1_param:', reg_l1_param, ' hidden_size:', hidden_size, ' Test accuracy:', test_acc)

    # Plot the model architecture
    #plot_model(model, to_file='model.png', show_shapes=True)
```

در نهایت با توجه به خروجی هایپر تیونینگ برای دو پارامتر به مقادیر زیر میرسیم:

```
sorted_list = sorted(accuracy, key=lambda k: k['acc'], reverse=True)
print(sorted_list)
```

```
[{'reg_l1_param': 0.001, 'hidden_size': 150, 'acc': 0.8907999992370605},
```

شبکه را با پارامترهای یافت شده اجرا میکنیم و به accuracy زیر میرسیم:

```
157/157 [==============================] - 1s 5ms/step - loss: 0.4015 - accuracy: 0.8922
reg_l1_param: 0.001  hidden_size: 150  Test accuracy: 0.8921999931335449
```

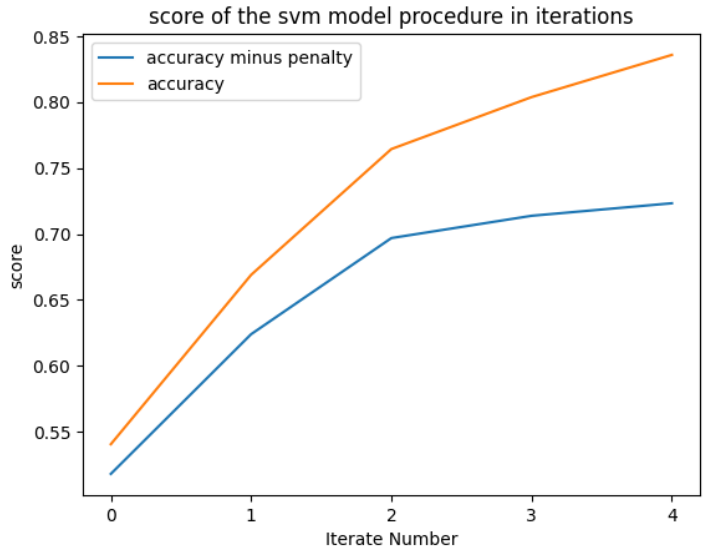
حال باید از مجموع 1024 فیچر موجود با استفاده از شبکه داده شده با پارامترهای بدست آمده در مرحله قبل، به تعداد کمتری فیچر برسیم و مدل svm را با تعداد فیچر جدید اجرا کنیم و accuracy و همچنین تاثیر تعداد فیچر ها به عنوان penalty به دنبال بهترین حالت باشیم.

به این منظور همانطور که گفته شده اهمیت فیچر ها را بر اساس میانگین وزن اختصاص یافته به هر فیچر در ورودی لایه میانی (hidden layer1) مشخص میکنیم و هر فیچری که میانگین وزن بیشتری داشته باشد، فیچر تاثیر گذارتری بوده است. تعدادی از فیچر های برتر را انتخاب میکنیم و دوباره شبکه را بر روی باقی فیچرهای باقیمانده اجرا میکنیم، این بار نیز طبق همان فرآیند قبلی تعدادی از فیچر ها را بر می گزینیم و این کار انتخاب فیچر را تا آنجا پیش میبریم که accuracy – penalty قابل قبول برای مدل svm برسیم. (در هر iteration مدل Svm را با فیچر های انتخاب شده train میکنیم و دقت مدل و همچنین دقت با تاثیر پنالتی تعداد فیچر را بررسی میکنیم)

پارامتر های قابل تامل این است که چند بار iteration انجام دهیم و در هر بار چند فیچر برتر را انتخاب کنیم. تابعی با همین دو پارامتر با نام feature_selection میسازیم و خروجی را بر اساس score بدست آمده تحلیل میکنیم.

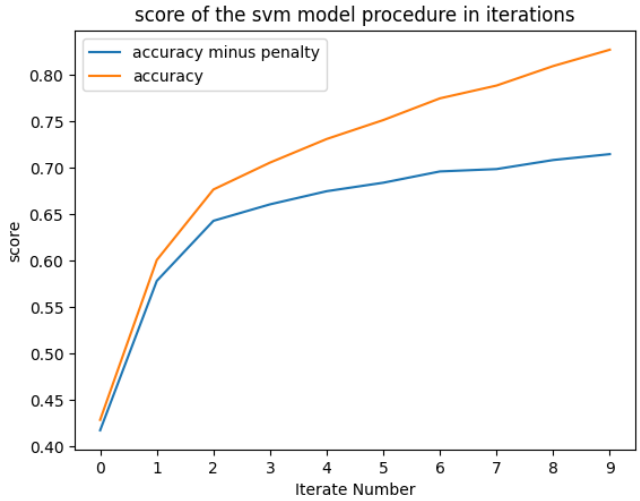
برای یافتن مقدار مطلوب دو پارامتر number of iteration و number of features selected in each iteration، ابتدا نکته قابل بررسی این است که برای رسیدن به یک تعداد مشخص فیچر اگر راه های مختلفی را پیش ببریم، آیا خروجی یکسانی بدست می آید یا نه. برای این منظور 150 فیچر نهایی رو در نظر میگیریم. یکبار با 5 ایتریشن و هربار انتخاب 30 فیچر برتر، یکبار با 10 ایتریشن و هر بار انتخاب 15 فیچر برتر و بار دیگر با 30 ایتریشن و هر بار انتخاب 5 فیچر برتر پیش میرویم و نتایج را بررسی میکنیم.

```
acc minus penalty:[0.5181, 0.624, 0.6969, 0.7138, 0.7232999999999999]
acc:[0.5406, 0.669, 0.7644, 0.8038, 0.8358]
```

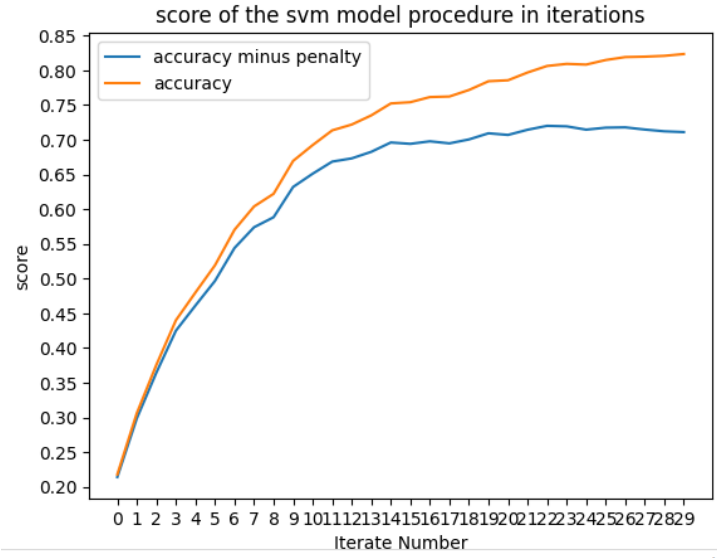


5 iterate 30 feature select

```
acc_minus_penalty:[0.41715, 0.5781000000000001, 0.6428499999999999, 0.6606, 0.67475, 0.6839, 0.6960500000000001, 0.6986, 0.70835, 0.7147]
acc:[0.4284, 0.6006, 0.6766, 0.7056, 0.731, 0.7514, 0.7748, 0.7886, 0.8096, 0.8272]
```



10 iterate 5 feature select



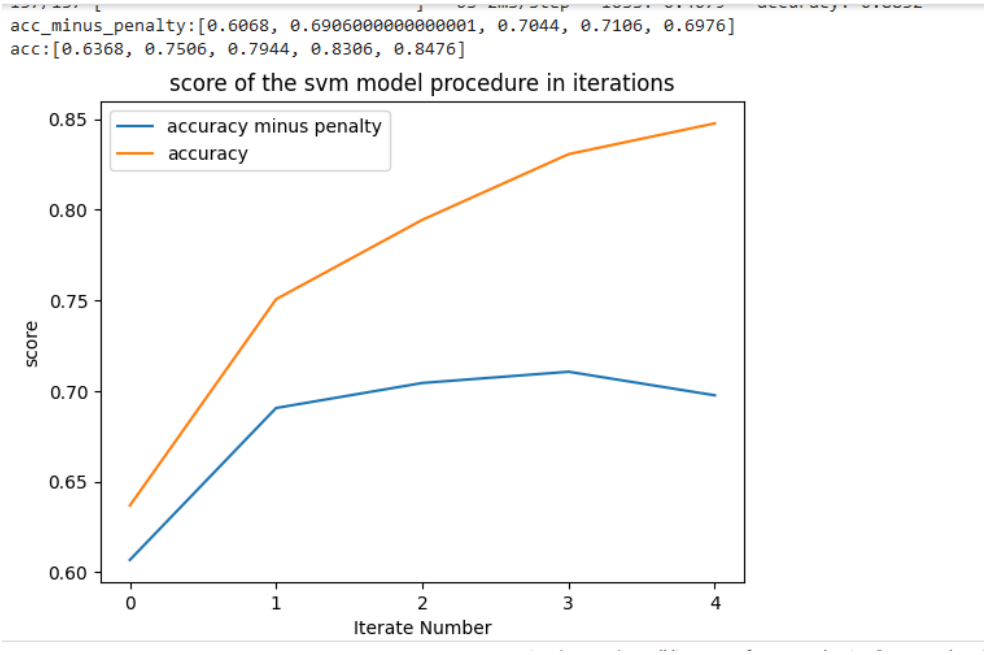
30 iterate 5 feature select

نتایج بدست آمده گر چه در نهایت 150 فیچر را شامل میشوند، اما یکسان نیستند.

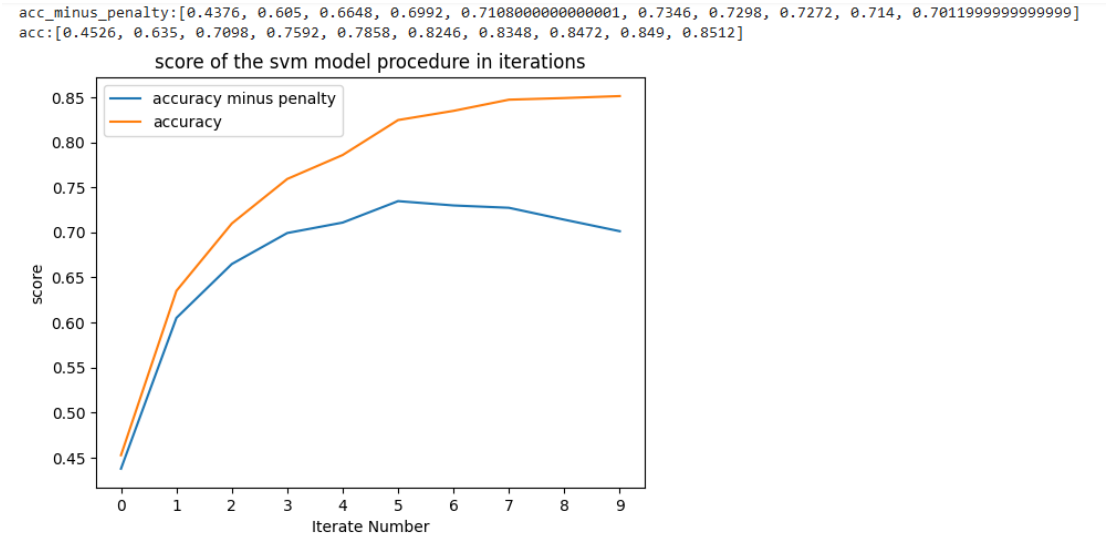
از حالت سوم، 5*30 نکاتی که بدست می آید این است گرچه افزایش فیچرهای مدل که در حالت افزایش iteration ها هم پیش می آید، دقت مدل رو افزایش میدهد، اما لزوما افزایش iteration ها سبب افزایش میزان امتیاز همراه با پنالتی تعداد فیچر نمیشود و در واقع میزان افزایش دقت نمیتواند تاثیر پنالتی از جایی به بعد از بین ببرد و تاثیر مثبت داشته باشد.

با توجه به نتایج بدست آمده، نکته دیگری که باید توجه کنیم تعداد فیچر نهایی است، در آزمایشاتی که برای train شبکه انجام شد، به این نتیجه رسیدم که با حدود 200 فیچر برتر کل، دقت شبکه به میزان کافی است ولی وقتی به میزان 100 کاهش

میدهیم، افت دقت شبکه به نسبت زیاد میشود. رنج تقریبا مطلوب تعداد فیچر نهایی جهت accuracy مناسب مدل svm تقریبا همین حدود میتوان نتیجه گرفت. چندین حالت دیگر را در همین رنج تست میکنیم.

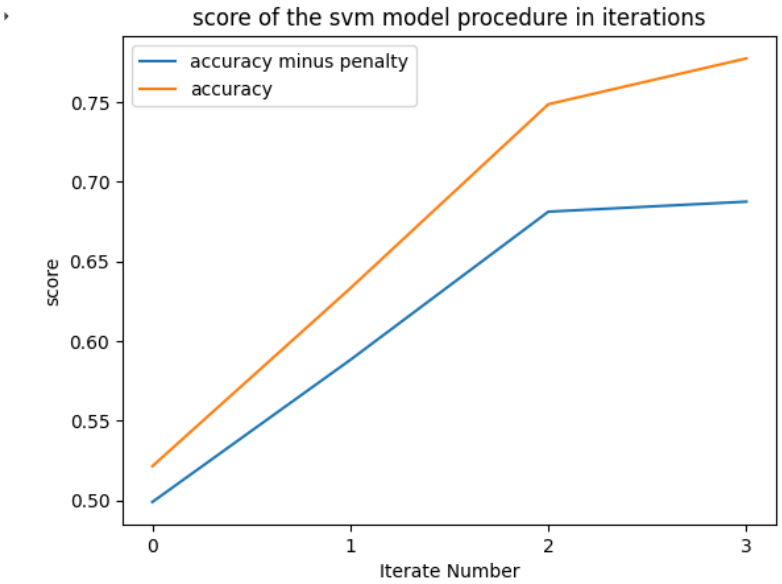


5 iterate 40 features select



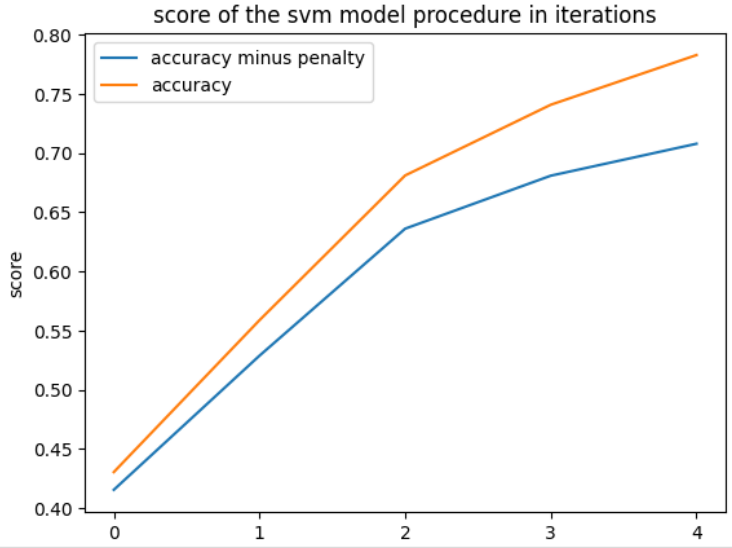
10 iterate 20 features select

```
acc_minus_penalty:[0.49889999999999995, 0.5882, 0.6813, 0.6876]
acc:[0.5214, 0.6332, 0.7488, 0.7776]
```



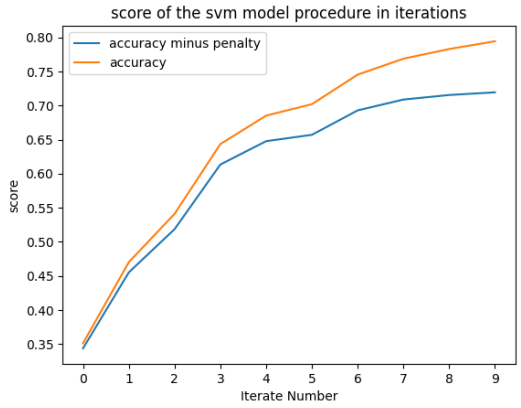
4 iterate 30 features select

```
acc_minus_penalty:[0.4154, 0.5287999999999999, 0.6362, 0.681, 0.7080000000000001]
acc:[0.4304, 0.5588, 0.6812, 0.741, 0.783]
```



5 iterate 20 features select

```
acc_minus_penalty:[0.34349999999999997, 0.45499999999999996, 0.5185000000000001, 0.6134, 0.6477, 0.6569999999999999, 0.6929, 0.7088000000000001, 0.7155, 0.
acc:[0.351, 0.47, 0.541, 0.6434, 0.6852, 0.702, 0.7454, 0.7688, 0.783, 0.7944]
```



10 iterate 10 features select

نمودار برخی حالات که صراحتا نامطلوب بودن آن ها را نمایش میدهد. از طرفی میزان امتیاز با تاثیر پنالتی تعداد فیچر از حدود 72 درصد بیشتر بدست نیامده است که این مقدار در iteration هفتم انتخاب 20 فیچر در هر مرحله بدست آمده است. از طرفی دقت مدل svm بدون توجه به تعداد فیچر در بهترین حالت با تعداد فیچرهای محدود شده، حدود 85 درصد است. با شواهد فوق شبکه را بار دیگر با 6 ایتریشن و انتخاب هر بار 20 فیچر ران میکنیم و فیچر های انتخابی را استخراج میکنیم.

در نهایت مدل Svm با دریافت تنها 140 فیچر با accuracy 82 درصد طبقه بندی میکند و با در نظر گرفتن پنالتی تعداد فیچر به 72 درصد میرسد:

```
(5000, 1024)
157/157 [==============================] - 0s 2ms/step - loss: 0.3980 - accuracy: 0.8902
(5000, 1004)
157/157 [==============================] - 0s 2ms/step - loss: 0.3995 - accuracy: 0.8896
(5000, 984)
157/157 [==============================] - 0s 2ms/step - loss: 0.4067 - accuracy: 0.8886
(5000, 964)
157/157 [==============================] - 0s 2ms/step - loss: 0.4115 - accuracy: 0.8864
(5000, 944)
157/157 [==============================] - 0s 2ms/step - loss: 0.4146 - accuracy: 0.8840
(5000, 924)
157/157 [==============================] - 1s 3ms/step - loss: 0.4250 - accuracy: 0.8846
(5000, 904)
157/157 [==============================] - 0s 2ms/step - loss: 0.4229 - accuracy: 0.8866
140
accuracy:0.8274
accuracy_minus_penalty:0.7224
```