گزارش تمرین کامپیوتری سوم سیگنال سیستم دکتر اخوان

پاییز 1403

محمد مهدی صمدی (810101465)

سپهر جمالی (810101400)

بخش سوم)

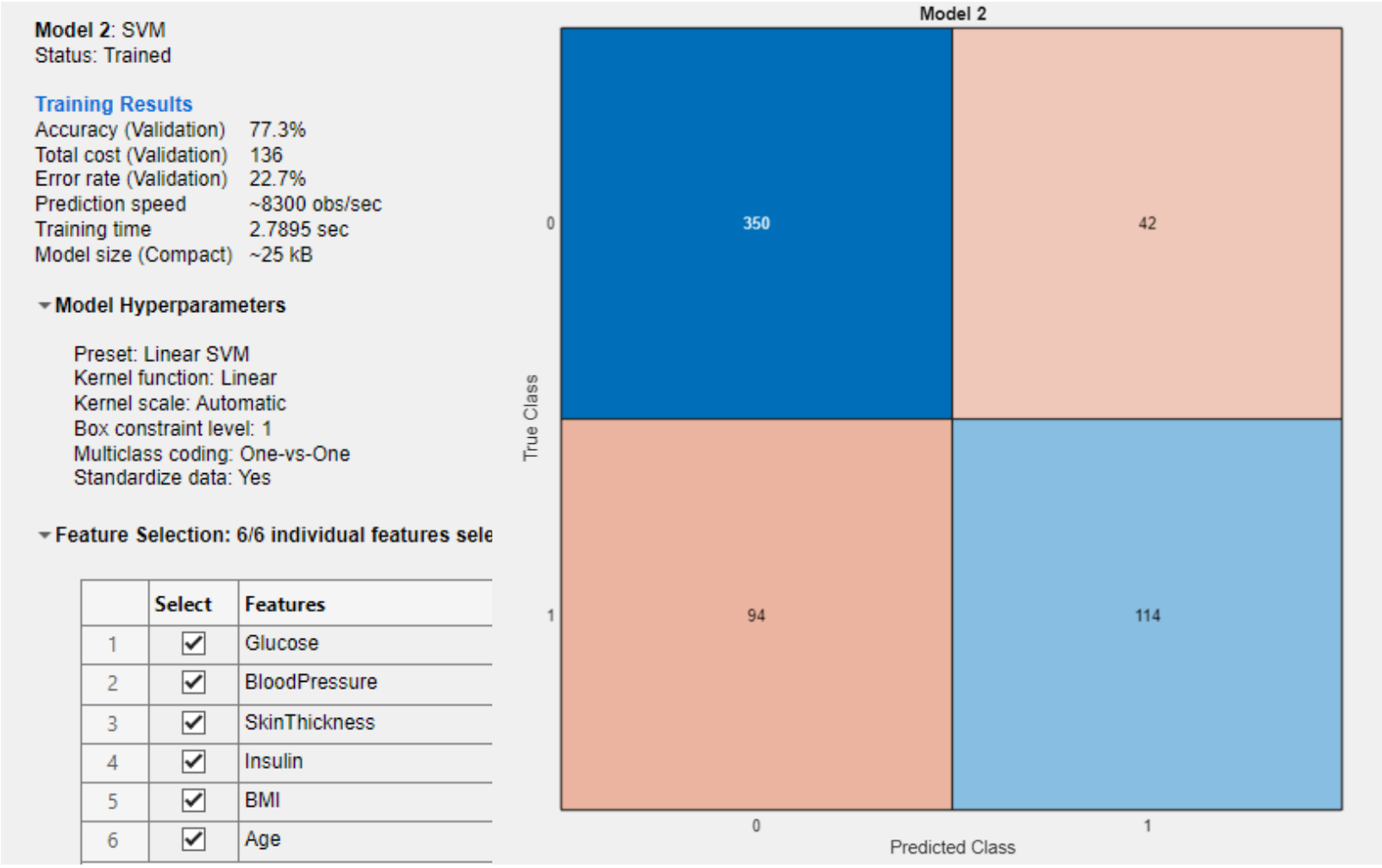
در ابتدا dataset را در محیط متلب load می‌کنیم. چند سطر اول را به عنوان نمونه مشاهده می‌کنیم.

	diabetestraining						
	Glucose	BloodPress...	SkinThickn...	Insulin	BMI	Age	label
	Number	Number	Number	Number	Number	Number	Number
1	Glucose	BloodPress...	SkinThickn...	Insulin	BMI	Age	label
2	148	72	35	0	33.6	50	1
3	85	66	29	0	26.6	31	0
4	183	64	0	0	23.3	32	1
5	89	66	23	94	28.1	21	0
6	137	40	35	168	43.1	33	1
7	116	74	0	0	25.6	30	0
8	78	50	32	88	31	26	1
9	115	0	0	0	35.3	29	0
10	197	70	45	543	30.5	53	1
11	125	96	0	0	0	54	1
12	110	92	0	0	37.6	30	0
13	168	74	0	0	38	34	1
14	139	80	0	0	27.1	57	0
15	189	60	23	846	30.1	59	1
16	166	72	19	175	25.8	51	1
17	100	0	0	0	30	32	1
18	118	84	47	230	45.8	31	1
19	107	74	0	0	29.6	31	1

تمرین 3_1)

حالا dataset را به اپلیکیشن Classification Learner اضافه می‌کنیم و مدل Linear SVM را روی داده‌ها train می‌کنیم. نتیجه این مدل را در تصاویر زیر مشاهده می‌کنیم.

مدل روی training data دقت 77 درصد دارد.



Model 2: SVM
Status: Trained

Training Results

Accuracy (Validation)	77.3%
Total cost (Validation)	136
Error rate (Validation)	22.7%
Prediction speed	~8300 obs/sec
Training time	2.7895 sec
Model size (Compact)	~25 kB

Model Hyperparameters

Preset: Linear SVM
Kernel function: Linear
Kernel scale: Automatic
Box constraint level: 1
Multiclass coding: One-vs-One
Standardize data: Yes

Feature Selection: 6/6 individual features sele

	Select	Features
1	☑	Glucose
2	☑	BloodPressure
3	☑	SkinThickness
4	☑	Insulin
5	☑	BMI
6	☑	Age

توضیحاتی درباره confusion matrix:

- در مسائل classification با n لیبل مختلف، این ماتریس nxn خواهد بود.
- عدد در سطر iاُم و ستون jاُم نشان دهنده تعداد داده‌هایی است که لیبل i دارند و لیبل j برای‌شان پیش‌بینی شده است. پس به جز خانه‌های روی قطر اصلی ماتریس، بقیه داده‌ها اشتباه پیش‌بینی شده‌اند.
- 350 نفر دیابتی نبودند که مدل هم آن‌ها را دیابتی تشخیص نداده است.

- 114 نفر دیابتی بودند که مدل نیز درست عمل کرده است.
- 42 نفر دیابت نداشته‌اند اما مدل آن‌ها را دیابتی تشخیص داده است.
- 94 نفر که دیابت داشتند، مدل نتوانسته تشخیص دهد.

بخش 3_2)

به ازای هر یک از 6 فیچر داخل dataset یک بار مدل را train می‌کنیم.

- گلوکز: دقت 74.3 درصد. نسبت به حالت اول، TP کمتر و FN بیشتری دارد.

Model 3: SVM
Status: Trained

Training Results

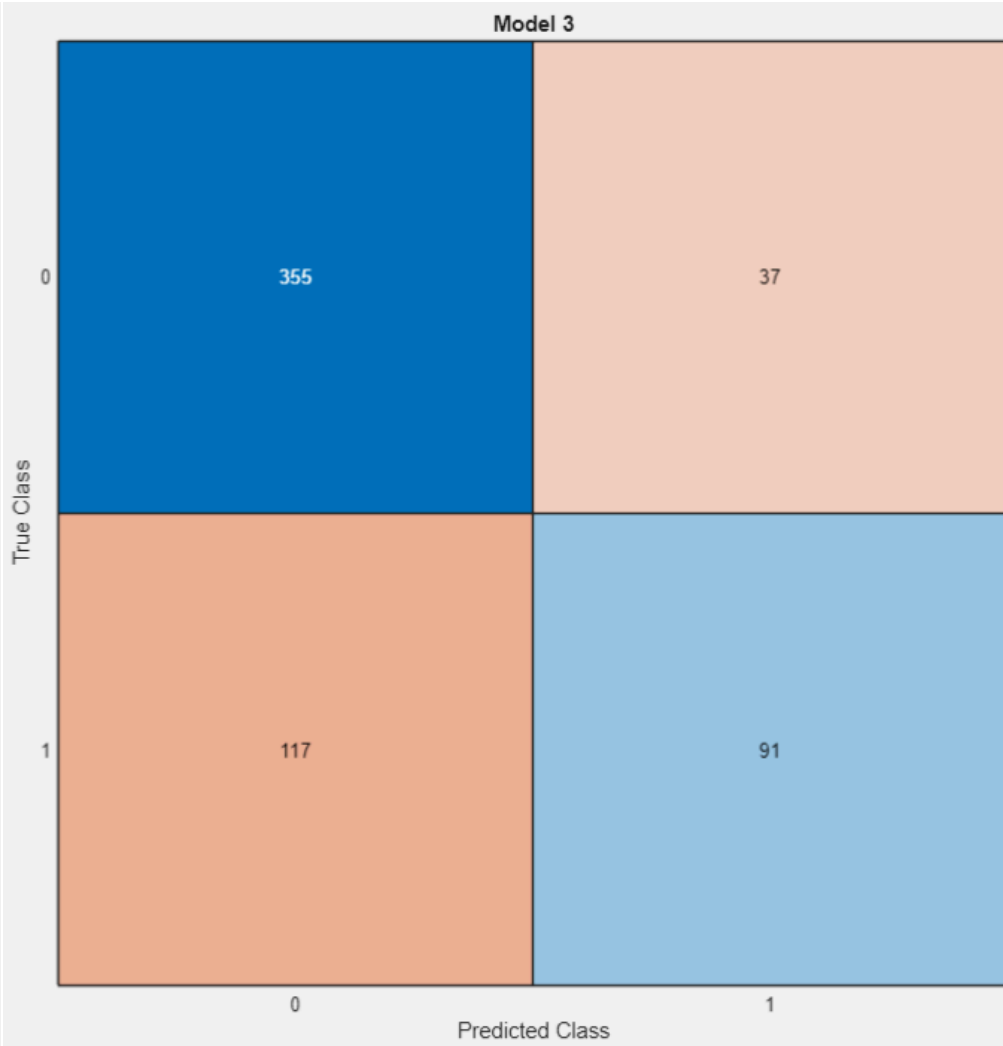
Accuracy (Validation)	74.3%
Total cost (Validation)	154
Error rate (Validation)	25.7%
Prediction speed	~18000 obs/sec
Training time	5.5109 sec
Model size (Compact)	~12 kB

Model Hyperparameters

Preset: Linear SVM
Kernel function: Linear
Kernel scale: Automatic
Box constraint level: 1
Multiclass coding: One-vs-One
Standardize data: Yes

Feature Selection: 1/6 individual features selected

	Select	Features
1	☑	Glucose
2	☐	BloodPressure
3	☐	SkinThickness
4	☐	Insulin
5	☐	BMI
6	☐	Age



- فشار خون: دقت 65.3 درصد. در این حالت مدل تمام مراجعین را سالم پیش‌بینی کرده است! در واقع فقط TP و FP داریم و تعداد TN و FN ها صفر است. این نشان‌دهنده ضعف مدل است.

Model 4: SVM
Status: Trained

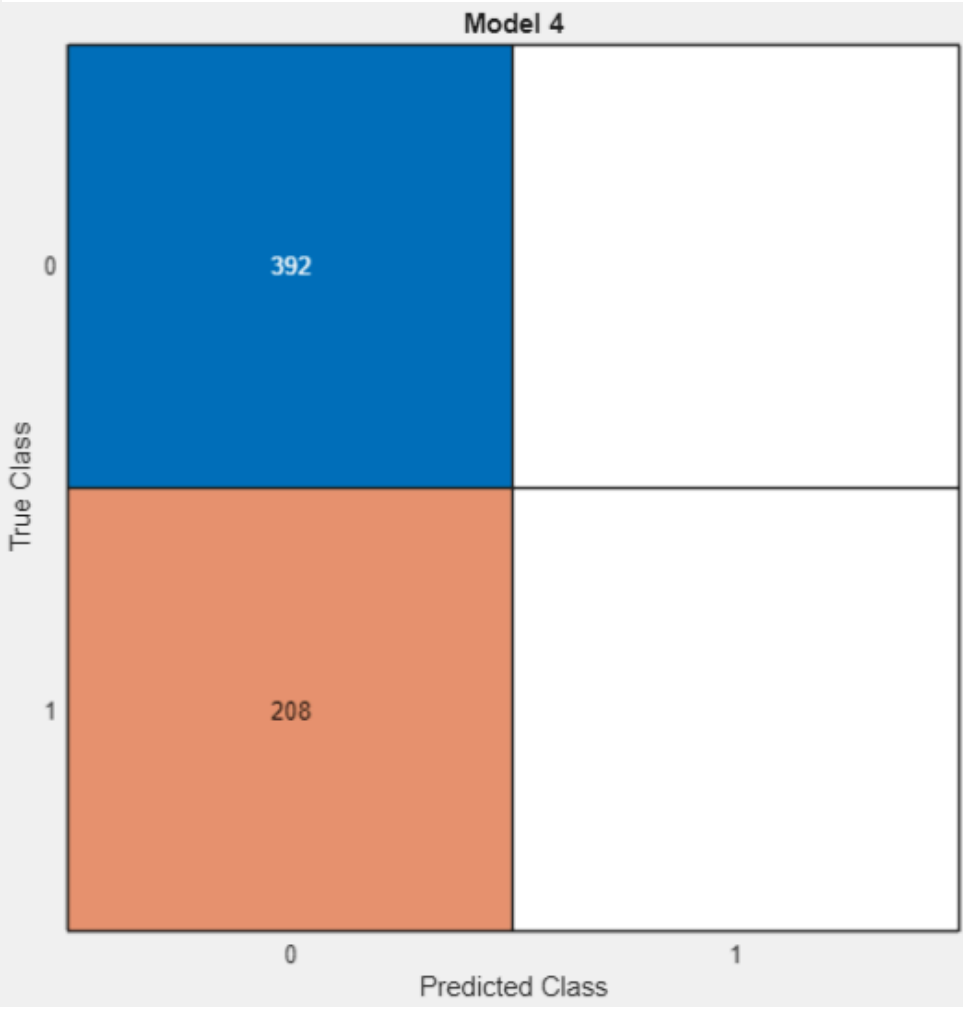
Training Results

Accuracy (Validation)	65.3%
Total cost (Validation)	208
Error rate (Validation)	34.7%
Prediction speed	~81000 obs/sec
Training time	0.53413 sec
Model size (Compact)	~14 kB

Model Hyperparameters

Feature Selection: 1/6 individual features selected

	Select	Features
1	☐	Glucose
2	☑	BloodPressure
3	☐	SkinThickness
4	☐	Insulin
5	☐	BMI
6	☐	Age



- نازکی پوست: دقت 65.3 درصد. همان مشکل مدل train شده روی فشار خون را دارد.

Model 5: SVM
Status: Trained

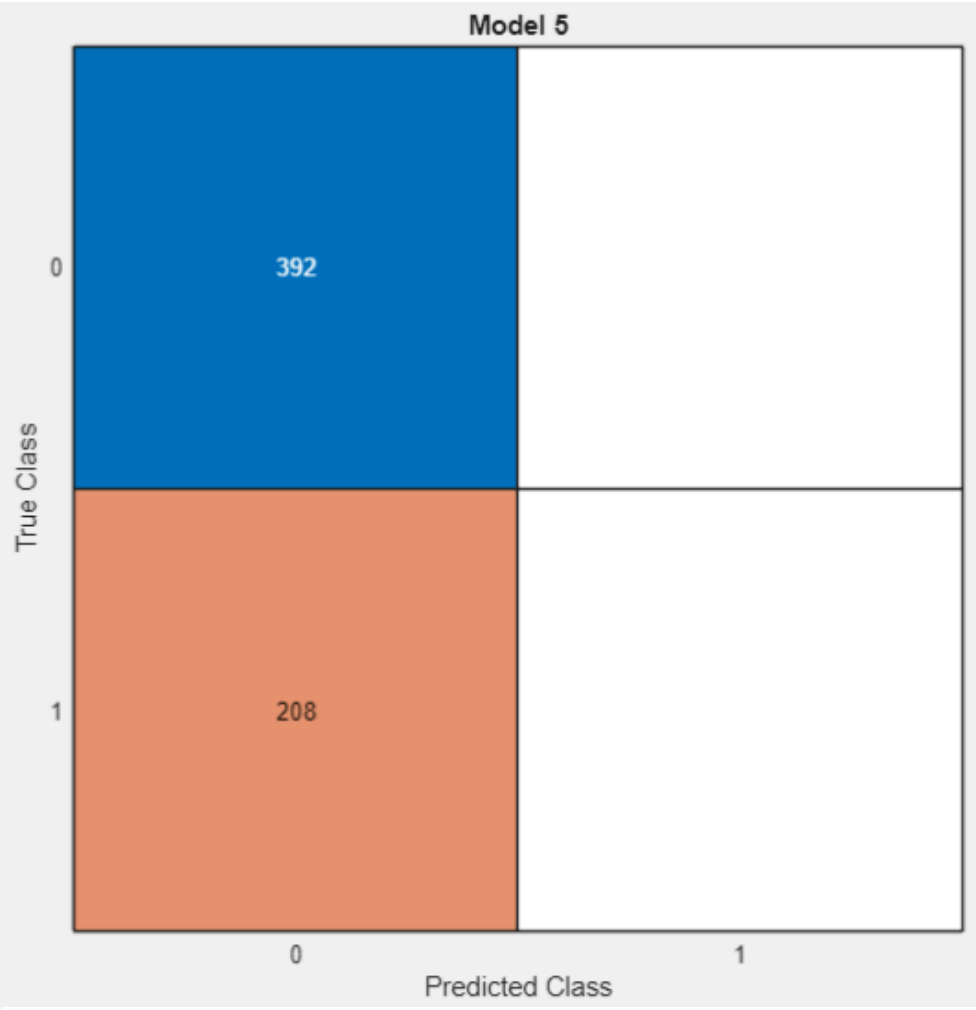
Training Results

Accuracy (Validation)	65.3%
Total cost (Validation)	208
Error rate (Validation)	34.7%
Prediction speed	~68000 obs/sec
Training time	0.63392 sec
Model size (Compact)	~14 kB

Model Hyperparameters

Feature Selection: 1/6 individual features selected

	Select	Features
1	☐	Glucose
2	☐	BloodPressure
3	☑	SkinThickness
4	☐	Insulin
5	☐	BMI
6	☐	Age



- انسولین: دقت 65.3 دارد. همان مشکل دو مدل قبلی را دارد.

Model 7: SVM
Status: Trained

Training Results

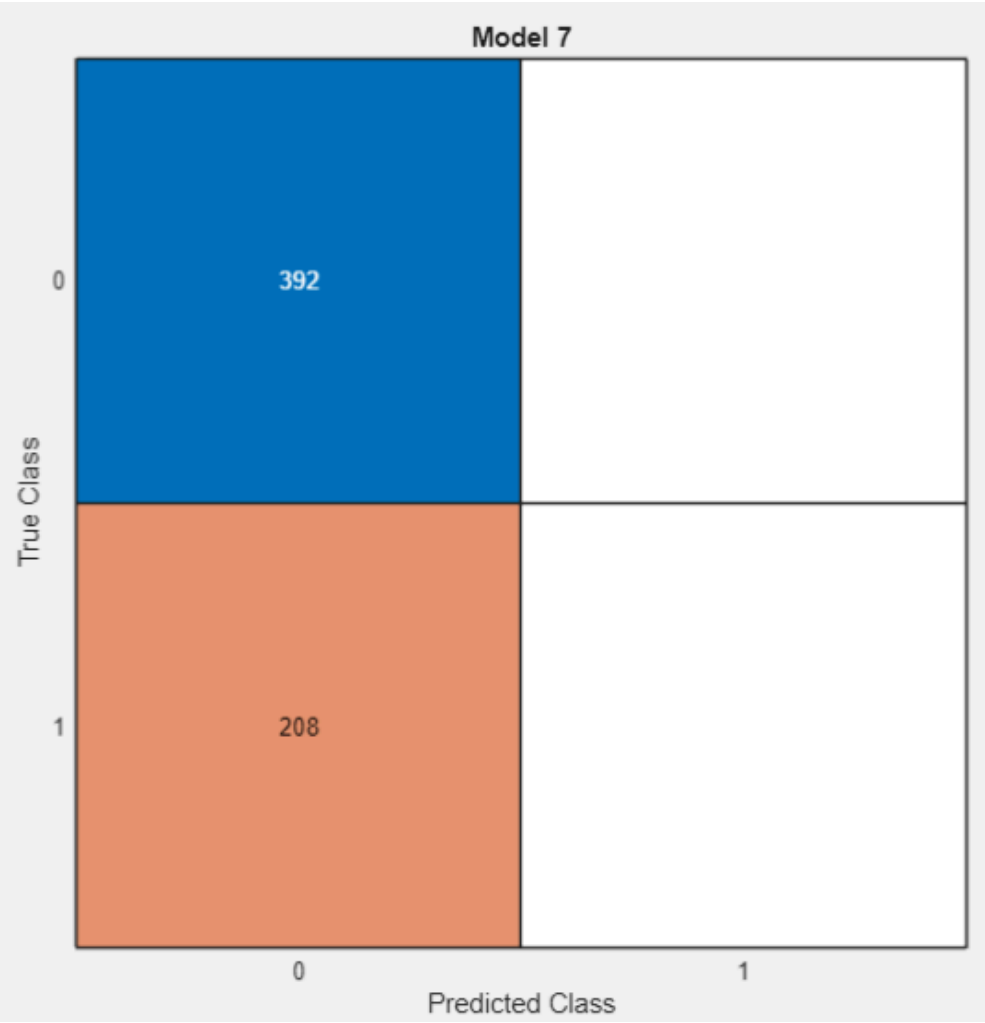
Accuracy (Validation)	65.3%
Total cost (Validation)	208
Error rate (Validation)	34.7%
Prediction speed	~71000 obs/sec
Training time	0.74154 sec
Model size (Compact)	~14 kB

Model Hyperparameters

Preset: Linear SVM
Kernel function: Linear
Kernel scale: Automatic
Box constraint level: 1
Multiclass coding: One-vs-One
Standardize data: Yes

Feature Selection: 1/6 individual features selected

	Select	Features
1	☐	Glucose
2	☐	BloodPressure
3	☐	SkinThickness
4	☑	Insulin
5	☐	BMI
6	☐	Age



- شاخص BMI: دقت مدل 65.5 است که کمی از سه مدل قبلی بهتر شده. زیرا یک داده از FN به TP

Model 8: SVM
Status: Trained

Training Results

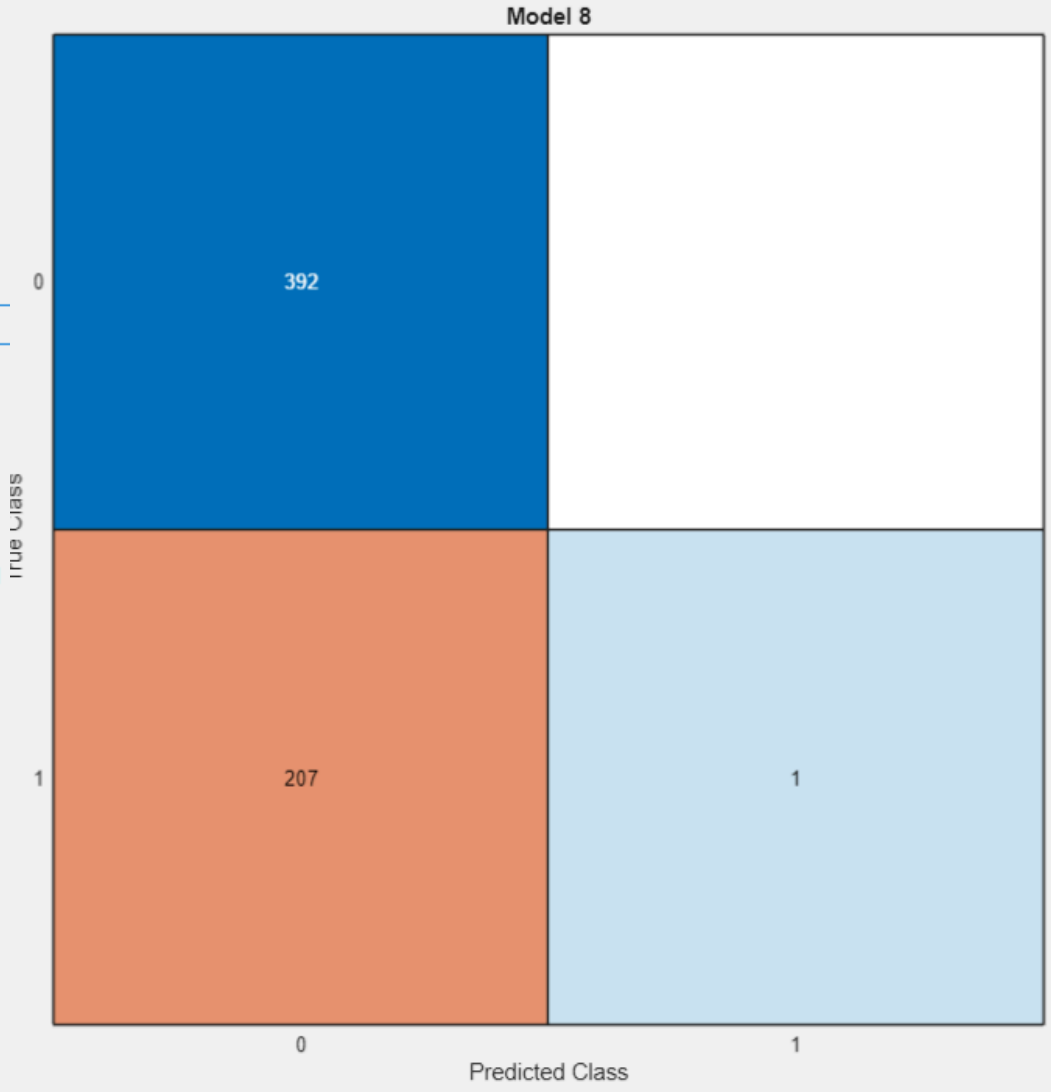
Accuracy (Validation)	65.5%
Total cost (Validation)	207
Error rate (Validation)	34.5%
Prediction speed	~74000 obs/sec
Training time	5.2498 sec
Model size (Compact)	~14 kB

Model Hyperparameters

Preset: Linear SVM
Kernel function: Linear
Kernel scale: Automatic
Box constraint level: 1
Multiclass coding: One-vs-One
Standardize data: Yes

Feature Selection: 1/6 individual features selected

	Select	Features
1	☐	Glucose
2	☐	BloodPressure
3	☐	SkinThickness
4	☐	Insulin
5	☑	BMI
6	☐	Age



- سن: دقت مدل 65.3 است. باز هم همان مشکل تکراری را دارد.

Model 9: SVM
Status: Trained

Training Results

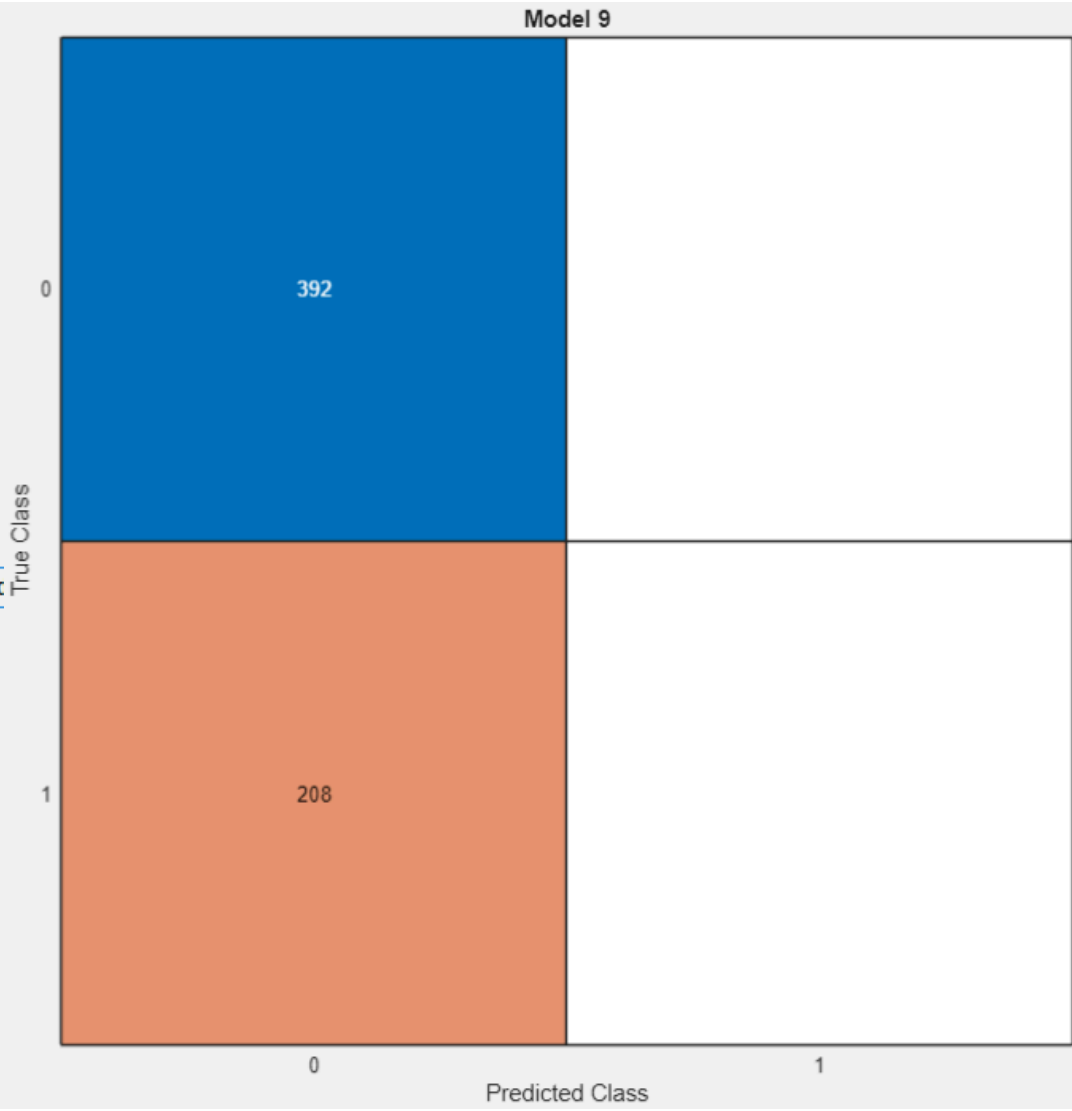
Accuracy (Validation)	65.3%
Total cost (Validation)	208
Error rate (Validation)	34.7%
Prediction speed	~78000 obs/sec
Training time	0.58642 sec
Model size (Compact)	~14 kB

Model Hyperparameters

Preset: Linear SVM
Kernel function: Linear
Kernel scale: Automatic
Box constraint level: 1
Multiclass coding: One-vs-One
Standardize data: Yes

Feature Selection: 1/6 individual features selected

	Select	Features
1	☐	Glucose
2	☐	BloodPressure
3	☐	SkinThickness
4	☐	Insulin
5	☐	BMI
6	☑	Age



پس بهترین فیچر برای پیش‌بینی دیابت داشتن/نداشتن میان فیچرهای در دسترس ما، میزان گلوکز است.

بخش 3_3)

برای این بخش توابعی نوشته شده که در ادامه به توضیح آن‌ها می‌پردازیم.

- تابع پیش‌بینی لیبل‌ها:

```
function predicted_labels=predict_labels(model_file, model_name, dataset_file, dataset_name)
    load(model_file);
    load(dataset_file);
    predicted_labels = model_name.predictFcn(dataset_name);

```

این تابع 4 آرگومان می‌گیرد. اسم فایلی که مدل در آن ذخیره شده است، اسم متغیر مدل در آن فایل و همین دو متغیر برای دیتاست.

سپس لیبل را پیش‌بینی می‌کند و در متغیر predicted_labels بازمی‌گرداند.

- چک کردن پیش‌بینی‌ها:

این تابع لیبل‌های پیش‌بینی شده را با لیبل‌های واقعی داخل دیتاست مقایسه می‌کند و چهار عدد بازمی‌گرداند.

TP: حدس 1، واقعا 1

FP: حدس 1، واقعا 0

TN: حدس 0، واقعا 0

FN: حدس 0، واقعا 1

در واقع همان 4 عدد داخل ماتریس کانفیوژن هستند.

در تابع بعد از این 4 عدد برای محاسبه معیارهای مختلف سنجش مدل طبقه‌بند استفاده می‌کنیم.

```
function [TP, FP, TN, FN]=check_predictions(true, preds)
    n1 = length(true);
    n2 = length(preds);
    if (n1 ~= n2)
        fprintf('size of two arrays do not match!\n');
        return
    end
    n = n1;
    TP = 0;
    FP = 0;
    TN = 0;
    FN = 0;

    for i=1:1:n
        if true(i) == 1 && preds(i) == 1
            TP = TP + 1;
        end

        if true(i) == 1 && preds(i) == 0
            FN = FN + 1;
        end

        if true(i) == 0 && preds(i) == 1
            FP = FP + 1;
        end

        if true(i) == 0 && preds(i) == 0
            TN = TN + 1;
        end
    end
end
```

- ارزیابی مدل binary classifier:

```
function [accuracy, precision, recall, f1]=evaluate_model(TP, FP, TN, FN)
    accuracy = (TP + TN) / (TP + TN + FP + FN);
    precision = (TP) / (TP + FP);
    recall = (TP) / (TP + FN);
    f1 = 2 * (precision * recall) / (precision + recall);
end
```

4 معیار مختلف محاسبه شده‌اند

1. اکیورسی: دقت مدل. تعداد پیش‌بینی‌های صحیح به کل تعداد پیش‌بینی‌ها. مقدار بالای اکیورسی نشان می‌دهد که مدل پیش‌بینی اشتباه کمی می‌کند.
2. پرسیژن: قاطعیت مدل. وقتی مدل به یک داده لیبل 1 می‌زند، چقدر احتمال دارد که آن داده واقعا لیبل 1 داشته باشد؟ مقدار بالای پرسیژن نشان می‌دهد که مدل FP های زیادی تولید نمی‌کند. اگر مدل خیلی محتاط باشد و فقط داده‌هایی که خیلی از 1 بودنشان مطمئن است را 1 لیبل بزند، پرسیژن بالایی خواهد داشت در حالی که مدل خوبی نداریم.
3. ریکال: نشان می‌دهد چند درصد از داده‌های با لیبل 1 واقعا لیبل 1 خورده‌اند. مقدار بالای ریکال به این معناست که مدل اکثر لیبل‌های 1 را تشخیص می‌دهد. اگر مدل هر داده‌ای را که کمی احتمال دهد لیبل 1 دارد لیبل 1 بزند، ریکال بالایی خواهد داشت در حالی که مدل خوبی نداریم.
4. اف وان: میانگین هارمونیک بین پرسیژن و ریکال است. مشکلاتی که برای پرسیژن و ریکال ذکر شد را تا حدی حل می‌کند.

برای اجرای تسک خواسته شده در این بخش اسکریپت زیر را اجرا می‌کنیم.

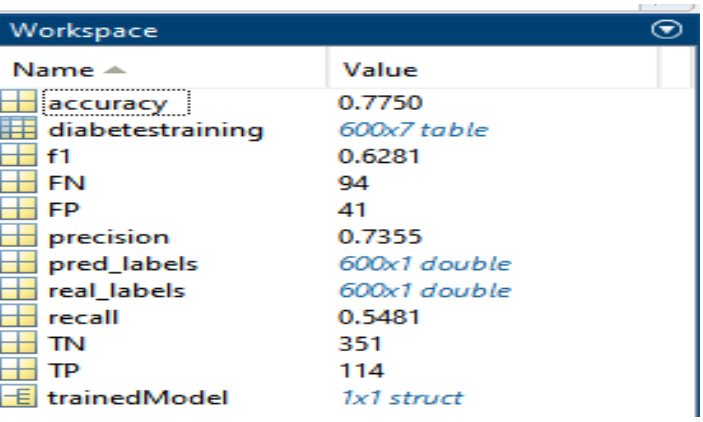
```
clc
clearvars
close all

load('trainSetDiabetes.mat')
load('trainedLinearSvmModel.mat')

pred_labels = predict_labels('trainedLinearSvmModel.mat', trainedModel, 'trainSetDiabetes.mat', diabetestraining);

real_labels = diabetestraining(:,7);
real_labels = table2array(real_labels);

[TP, FP, TN, FN] = check_predictions(real_labels, pred_labels);
[accuracy, precision, recall, f1] = evaluate_model(TP, FP, TN, FN);
```



همانطور که مشخص است دقت مدل و تمام اجزای ماتریس کانفیوژن با مقادیر گزارش شده در بخش 3_1 هم‌خوانی دارند.

بخش 3_4)

در این بخش از توابع 3_3 و اسکریپتی مشابه آن بخش استفاده می‌کنیم.

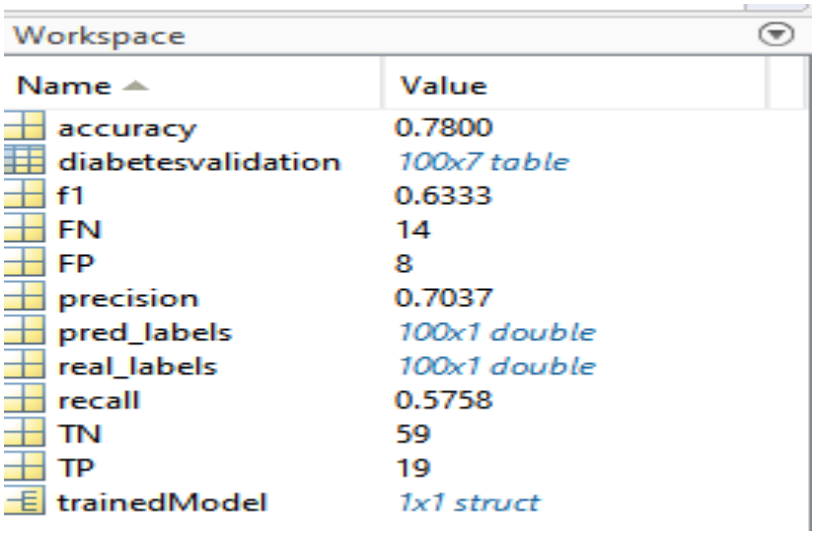
```
clc
clearvars
close all

load('testSetDiabetes.mat')
load('trainedLinearSvmModel.mat')

pred_labels = predict_labels('trainedLinearSvmModel.mat', trainedModel, 'testSetDiabetes.mat', diabetesvalidation);

real_labels = diabetesvalidation(:,7);
real_labels = table2array(real_labels);

[TP, FP, TN, FN] = check_predictions(real_labels, pred_labels);
[accuracy, precision, recall, f1] = evaluate_model(TP, FP, TN, FN);
```



نتایج در تصویر روبه‌رو قابل مشاهده هستند.

دقت در این حالت 78 درصد شد که حتی کمی بهتر از دقت train است.

نزدیکی دو دقت train و test نشان می‌دهد که مدل overfit یا underfit نشده است.

بخش دوم)

در این بخش برای تشخیص IC ها از Normalized correlation coefficient استفاده می‌کنیم.
ابتدا تصاویر را لود کرده و به رنگ خاکستری تبدیل می‌کنیم (کاهش بعد برای جلوگیری از پردازش بیهوده).
بعد از آن در حلقه هر بخش از PCB مطابق با سایز IC را با IC با در فرمول گفته شده در صورت سوال (خط 46 تابع) محاسبه میکنیم و اگر مقدار از threshold بیشتر باشد آن بخش رو به عنوان IC در نظر میگیریم. یک بار دیگر این مرحله را برای IC دوران یافته به اندازه 180 درجه انجام میدهیم.
کد تابع تشخیص IC:

```
1   function ICrecognition (MainBoard,IC)
2       figure;
3       cols = 2;
4       rows = 3;
5       cnt = 1;
6
7       subplot(rows,cols,cnt);
8       cnt = cnt +1;
9       imshow(MainBoard);
10      title('PCB Image');
11
12      subplot(rows,cols,cnt);
13      cnt = cnt +1;
14      imshow(IC);
15      title('IC Image');
16
17      GrayBoard = rgb2gray(MainBoard);
18      GrayIC = rgb2gray(IC);
19
20      subplot(rows,cols,cnt);
21      cnt = cnt +1;
22      imshow(GrayBoard);
23      title('PCB Image Grayscale');
24
25      subplot(rows,cols,cnt);
26      cnt = cnt +1;
27      imshow(GrayIC);
28      title('IC Image Grayscale');
29
30      [row_pcb,col_pcb] = size(GrayBoard);
```

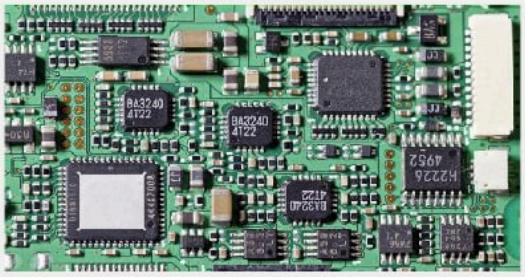
```
31 -      [row_ic,col_ic] = size(GrayIC);
32 -
33 -      subplot(rows,cols,cnt);
34 -      cnt = cnt +1;
35 -      imshow(MainBoard);
36 -      title("Matching Result");
37 -
38 -      threshold = 0.85;
39 -      for rotate = 1 : 2
40 -          GrayIC = imrotate(GrayIC,180);
41 -          for i = 1 : (row_pcb-row_ic)
42 -              for j = 1 : (col_pcb-col_ic)
43 -                  x = double(GrayIC);
44 -                  y = double(GrayBoard(i:(i+row_ic-1),j:(j+col_ic-1)));
45 -                  if sum(sum(x.*x))*sum(sum(y.*y)) ~= 0
46 -                      CorrCoeff = sum(sum(x.*y)) / sqrt(sum(sum(x.*x))*sum(sum(y.*y))) ;
47 -                  else
48 -                         CorrCoeff = 0;
49 -                  end
50 -                  if ( CorrCoeff > threshold )
51 -                      rectangle('Position', [j,i,col_ic,row_ic], 'EdgeColor', 'y', 'LineWidth', 2);
52 -                  end
53 -              end
54 -          end
55 -      end
56 -
57 -      subplot(rows,cols,cnt);
58 -      cnt = cnt +1;
59 -      imshow(imrotate(GrayIC,180));
60 -      title('IC Image Grayscale Rotated');
61 -  end
```

توضیح بخش‌هایی از کد:

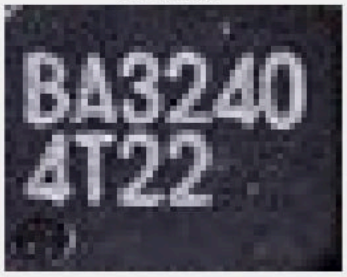
- خط 40: عکس خاکستری IC 180 درجه می‌چرخد. در دور بعدی حلقه 180 درجه دیگر هم می‌چرخد و به جای اصلی خود بازمی‌گردد.
- خطوط 43 و 44: مقدار هر پیکسل عددی صحیح است. آن‌ها را به اعداد دابل تبدیل می‌کنیم که خروجی عملیات‌های ریاضی در هنگام محاسبه correlation قطع نشود.
- خط 45: چک کردن اینکه مخرج فرمول محاسبه correlation صفر می‌شود یا خیر تا به خطای division by zero نخوریم.
- خط 50 و 51: اگر correlation تصویر بیشتر از حد مشخصی بود دور آن مستطیلی کشیده می‌شود.

خروجی کد بالا را در تصویر زیر مشاهده می‌کنیم:

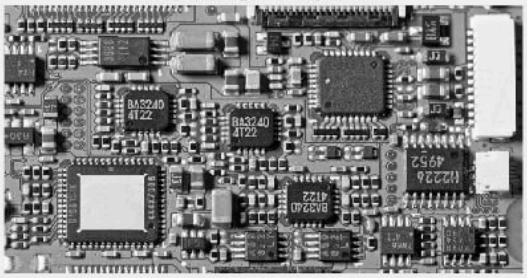
PCB Image



IC Image

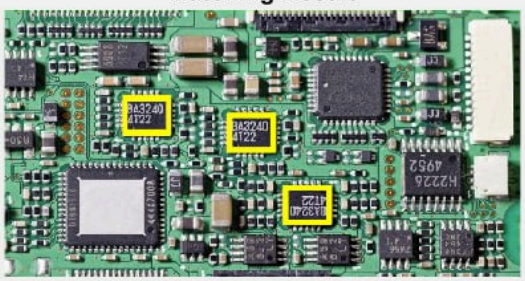


PCB Image Grayscale

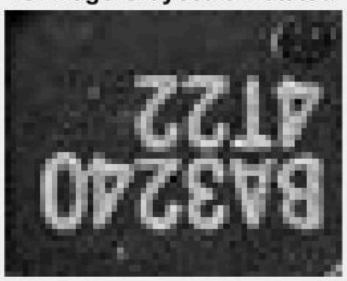


IC Image Grayscale

Matching Result



IC Image Grayscale Rotated



بخش اول)

در این بخش از کدی که استاد سر کلاس نوشتند استفاده کردیم. فقط تصویری که قرار است code/decode شود عوض شده است.

کد را در سه بخش توضیح می‌دهم.

```
1 -    X=imread('gnr.jfif');
2 -    X=rgb2gray(X);
3
4 -    Nch=32;
5 -    mapset=cell(2,Nch);
6 -    Alphabet ='abcdefghijklmnopqrstuvwxyz ,~".!';
7
8 -    for i=1:Nch
9 -        mapset{1,i}=Alphabet(i);
10 -       mapset{2,i}=dec2bin(i-1,5);
11 -   end
12
13 -   message='i love saeed akhavan bahabadi~';
14 -   message_len=length(message);
15 -   message_bin=cell(1,message_len);
16 -   for i=1:message_len
17 -       ch=message(i);
18 -       index=strcmp(ch,mapset(1,:));
19 -       message_bin{i}=mapset{2,index};
20 -   end
21
22 -   binarymessage=cell2mat(message_bin);
23 -   binarymessage_len=length(binarymessage);
24 -   Y=X;
25
```

- خط 1 و 2: لود کردن عکس در ماتریس سه بعدی و کاهش ابعاد آن به 2 (خاکستری)
- خط 4 تا 6: الفبای دیتاست تعریف شده و متلب سل به آن سایز ساخته می‌شود.
- خط 8 تا 11: هر حرف در الفبا، به یک عدد باینری 5 بیتی مپ می‌شود.
- خط 13 تا 15: مسیج ورودی تعریف می‌شود و بر اساس سایزش یک متلب سل تعریف می‌شود.

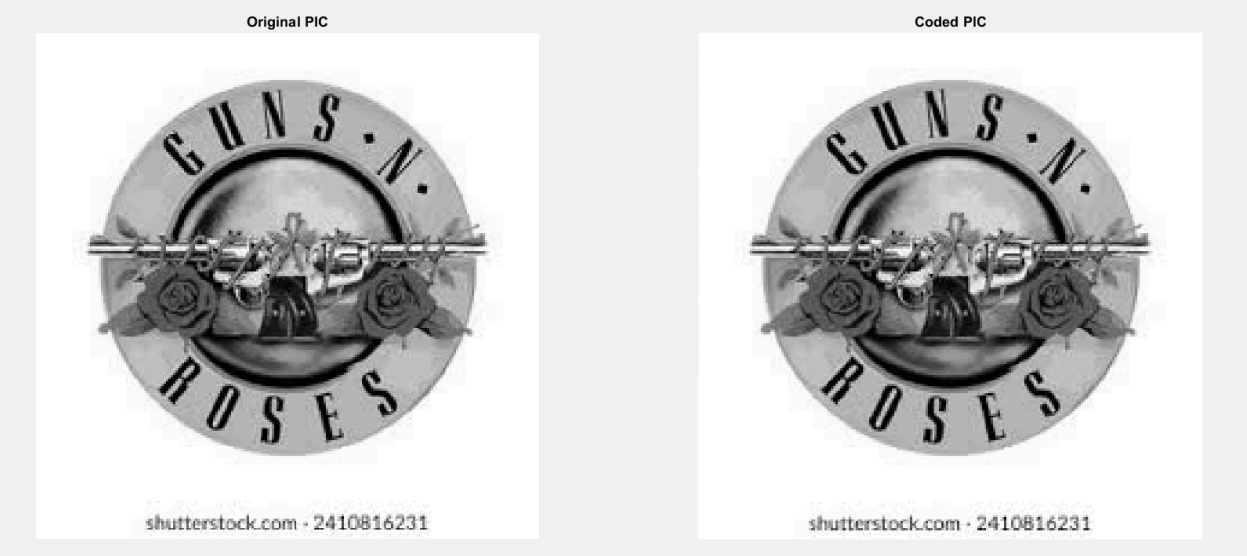
- خط 16 تا 20: هر کاراکتر داخل مسیج به معادل باینری 5 بیتی آن در مپست تبدیل شده و در message_bin ذخیره می‌شود.
- خط 22 تا 24: پیام باینری شده از متلب سل به استرینگ ریخته می‌شود و عکس اولیه در متغیر Y ذخیره می‌شوذ.

```
26 -  for i=1:binarymessage_len
27 -      vals=X(i);
28 -      valsbin=dec2bin(vals);
29 -      valsbin1=valsbin;
30 -      valsbin1(end)=binarymessage(i);
31 -      Y(i)=bin2dec(valsbin1);
32 -  end
33 -  subplot(1,2,1)
34 -  imshow(X)
35 -  title('Original PIC')
36 -  subplot(1,2,2)
37 -  imshow(Y)
38 -  title('Coded PIC')
39
40
41    % Decoding
42 -  DcodedMessageBin=[];
43 -  flag=1;
44 -  ind=1;
45
```

- خط 26 تا 32: هر بیت از رشته باینری در یک پیکسل از عکس ذخیره می‌شود. در واقع در بیت کم اهمیت (LSB) ذخیره می‌شود. به این دلیل که تغییرات زیادی در عکس ایجاد نکند و قابل مشاهده با چشم نباشد. پس اگر مسیج ورودی سایز n داشته باشد، به 5n بیت باینری تبدیل می‌شود و در 5n بیت اول
- خط 33 تا 38: پلات کردن عکس اولیه (قبل از code کردن) و عکس نهایی (بعد از code کردن).
- خط 42 تا 44: تعریف متغیرهای مورد نیاز برای decode کردن عکس.

```
46 -  while flag
47 -      characterbin=zeros(1,5);
48 -      for cont=1:5
49 -          vals=Y(ind);
50 -          vals1=dec2bin(vals);
51 -          characterbin(cont)=str2double(vals1(end));
52 -          ind=ind+1;
53 -      end
54 -      num=sum(characterbin.*(2.^(4:-1:0)))+1;
55 -      if strcmp(Alphabet(num),'~')
56 -          flag=0;
57 -      else
58 -          DcodedMessageBin=[DcodedMessageBin Alphabet(num)];
59 -      end
60 -  end
61
62 -    sprintf(DcodedMessageBin)
```

- خط 47 تا 53: تعریف آرایه characterbin برای نگه داشتن 5 بیت یک حرف. سپس آن را با 5 بیت پیش رو پر می‌کنیم.
- خط 54: آرایه charavterbin را به عدد دسیمال تبدیل کرده و به علاوه یک می‌کنیم. زیرا اندیس مپ‌ست base 1 است.
- خط 55 تا 59: اگر حرف تشخیص داده شده برابر کاراکتر قراردادی انتهای پیام (برای ما ~) بود، آن را ثبت نکن و از حلقه بیا بیرون. اگر برابر آن نبود، ثبتش کن و سراغ حرف بعد برو.



خروجی کد توضیح‌داده شده به صورت بالا است. همانطور که توضیح داده شد پیام داده شد در اولین پیکسل‌های عکس ذخیره می‌شوند (که اینجا همگی سفید هستند). اما چون LSB ها تغییر داده شدند حتی آن پیکسل‌های سفید هم تغییر داده نشدند. پس کد به خوبی کار می‌کند.