

# Computer Assignment 3

مصطفی کرمانی‌نیا 810101575

امیر نداف فهمیده 810101540

## بخش اول)

تمرین 1-1) ابتدا یک آرایه شامل کاراکترهای خواسته شده می‌سازیم همچنین طبق گفته صورت پروژه یک سلول  $2 * 32$  می‌سازیم به عنوان Mapset. سپس در یک حلقه for، برای هر کاراکتر در سطر اول خود آن کاراکتر و در سطر دوم بر اساس ایندکس آن (۱ - آ برای اینکه از ۰ تا ۳۱ کد گذاری شود)، به صورت باینری ۵ بیتی که به آن مرتبط می‌گردد را قرار می‌دهیم.

```

1 %% Part 1
2 % Define our char array which contains alphabet and additional characters
3 chars = ['a':'z', ' ', '.', ',', '!', "'", ";"];
4
5 % Creating a mapset with 2 rows and columns Row 1 is the characters Row 2
6 % is the binary coded value of each character
7 Mapset = cell(2, 32);
8
9 % Set the values into Mapset
10 for i = 1:32
11     % Store the character in the first row of Mapset
12     Mapset{1, i} = chars(i);
13
14     % Add the binary coded value for each character (i-1 in binary)
15     Mapset{2, i} = dec2bin(i - 1, 5);
16 end
17

```

**تمرین 1-2)** همانطور که خواسته شده یکتابع coding نوشته شده که سه ورودی پیام، عکس خاکستری (Gray picture) و mapset را می‌گیرد. سپس در تیکه اول کد، یک ترشلد (مثلا 1000 در کد ما) است می‌کند، عکس را می‌خواند و تعداد سطر و ستون های آن را در می‌آورد. بعد از این مراحل باید طبق دستور پروژه عکس را به بلوک های  $25 \times 25$  تقسیم کنیم و همچنین بلوک هایی را پیدا کنیم که مقدار واریانس آن ها از مقدار threshold بیشتر باشد. این کار به این منظور است که بلوک هایی که تغییرات در آنها کمتر دیده می‌شود (مثلا لبه های ساختمان) را پیدا کنیم تا به راحتی تغییر در عکس مشاهده نشود. مختصات پیکسل بالا دست چپ این بلوک ها را در آرایه selectedBlocks ذخیره می‌کنیم. بعد از این مرحله نیز تعداد بلوک های انتخاب شده را نمایش می‌دهیم.

```

52 %% part 2
53 function coded_pic = coding(message, gpic, mapset)
54
55 -     threshold = 1000;
56 -     % Load the grayscale image
57 -     imgGray = imread(gpic);
58 -     [rows, cols] = size(imgGray); % Image dimensions
59
60 -     % Define a list to store the selected boxes with variance above the threshold
61 -     selectedBlocks = [];
62
63 -     % Loop through the image with a sliding 25x25 window
64 -     for i = 1:25:(rows - 24)
65 -         for j = 1:25:(cols - 24)
66 -             % Extract 25x25 window and calculate its variance
67 -             window = imgGray(i:i+24, j:j+24);
68 -             variance = var(double(window(:)));
69
70 -             % Select the box if its variance is above the threshold
71 -             if variance > threshold
72 -                 selectedBlocks = [selectedBlocks; i, j];
73 -             end
74 -         end
75 -     end
76
77 -     % Display number of selected 25x25 boxes
78 -     disp(['Selected 25x25 boxes: ', num2str(size(selectedBlocks, 1))]);

```

بعد از این که بلوک ها را انتخاب کردیم نوبت به encode کردن پیام می‌رسد. یک رشته message\_bin داریم که وظیفه ذخیره کردن پیام باینری را به عهده دارد. بعد روی کاراکتر های پیام اصلی یک حلقه داریم. برای هر کاراکتر با استفاده از تابع find در سطر اول دنبال کاراکتر مورد نظر می‌گردد و ایندکس آن را در char\_index ذخیره می‌کنیم. اگر کاراکتری غیر از کاراکتر های قرارداد شده وارد شود، ارور می‌دهیم در غیر این صورت کد باینری متناظر آن عدد را به کمک char\_bin در mapset ذخیره کرده و به ادامه message\_bin اضافه می‌کنیم. سپس چک می‌کنیم اگر تعداد کل بیت هایی که می‌توانیم تغییر بدیم (بر اساس تعداد بلوک های انتخاب شده) کمتر از طول پیام باینری ما بود ارور می‌دهیم.

```
80 % Convert message to binary sequence using mapset
81 message_bin = '';
82 for k = 1:length(message)
83     char_index = find(strcmp(mapset(1,:), message(k))); % Find the character index in mapset
84     if isempty(char_index)
85         error(['Character ', message(k), ' not found in mapset.']);
86     end
87     char_bin = mapset{2, char_index}; % Get binary representation of each character
88     message_bin = [message_bin, char_bin]; % Append to the binary message sequence
89 end
90
91 % Check if there is enough space in selected boxes to embed the message
92 total_bits_available = numel(selectedBlocks) * 25 * 25; % Total bits in all selected 25x25 boxes
93 if length(message_bin) > total_bits_available
94     error('Message is too long to fit in the selected areas.');
95 end
```

در آخر این تابع هم عکس رمزنگاری شده خود را می‌سازیم. به این صورت که ابتدا coded\_pic را برابر با تصویر اولیه می‌گذاریم سپس یک پوینتر برای باینری مسیج داریم که به یک ست می‌شود. بعد از آن روی بلوک‌ها حلقه می‌زنیم. زمانی که پوینتر message\_bin از طول msg\_idx بیشتر شد یعنی کل پیام در تصویر جاسازی شد و از حلقه بیرون می‌آییم. اما هر بیت به این صورت در تصویر جاسازی می‌شود که ابتدا پیکسل بالا سمت چپ هر بلوک را در  $i$  و  $j$  ذخیره می‌کنیم. بعد دو حلقه تو در تو برای  $m$  و  $n$  داریم از 0 تا 24. در ابتدای حلقه داخلی چک می‌کنیم که آیا کل پیام در تصویر جاسازی شده یا خیر که اگر شده بود از حلقه‌ها بیرون می‌آییم. در غیر این صورت مقدار پیکسل در مختصات  $i+m, j+n$  را در می‌آوریم و آن را به یک عدد 8 بیتی تبدیل می‌کنیم. (چون هر پیکسل مقداری بین 0 تا 255 اختیار می‌کند و با 8 بیت نمایش داده می‌شود). سپس در کم ارزش ترین بیت آن، یک بیت از پیام encode شده را قرار می‌دهیم و دوباره آن پیکسل را به عدد دسیمال برمی‌گردانیم و در تصویر اصلی قرار می‌دهیم سپس یکی به پوینتر msg\_idx پیکسل را به عدد بعدی اشاره کند.

```

96
97 % Initialize the coded picture as a copy of the original image
98 coded_pic = imgGray;
99 msg_idx = 1; % Pointer for binary message bits
100
101 % Embed binary message in selected boxes
102 for block = 1:size(selectedBlocks, 1)
103     if msg_idx > length(message_bin)
104         break; % Stop if all message bits are embedded
105     end
106
107     % Get top-left corner of the 25x25 box
108     i = selectedBlocks(block, 1);
109     j = selectedBlocks(block, 2);
110
111     % Embed message bits in the least significant bit of each pixel in the box
112     for m = 0:24
113         for n = 0:24
114             if msg_idx > length(message_bin)
115                 break; % Stop if all message bits are embedded
116             end
117             pixel_val = coded_pic(i + m, j + n); % Get pixel value
118             pixel_bin = dec2bin(pixel_val, 8); % Convert to 8-bit binary
119
120             % Replace least significant bit with message bit
121             pixel_bin(end) = message_bin(msg_idx);
122             coded_pic(i + m, j + n) = bin2dec(pixel_bin); % Update pixel in coded image
123
124             msg_idx = msg_idx + 1; % Move to the next bit in the message
125         end
126     end
127 end
128

```

**تمرین 1-3)** در این بخش من از تصویر Amsterdam که در کلاس هم استفاده شد، استفاده کردم. در ابتدا پیام; signal را ست کردم بعد عکس مورد نظر را خوندم بعد آن را تبدیل به عکس خاکستری کردم و ذخیره کردم. سپس تابع coding را صدای زدم و پارامتر های مربوطه را به آن پاس دادم تصویر جدید را ذخیره کردم و تصویر اولیه و تصویر حاوی پیام را در کنار هم نمایش دادم.

```
19 %% part 3
20
21 % Write a message
22 MESSAGE = 'signal';
23
24 % Read the picture
25 FORMAT = ".png";
26 FILE_TO_READ = 'Amsterdam';
27 pic = imread(FILE_TO_READ + FORMAT);
28
29 % Make the picture gray
30 pic_gray = rgb2gray(pic);
31 GRAY_FILE = FILE_TO_READ + "_gray" + FORMAT;
32 imwrite(pic_gray,GRAY_FILE);
33
34 % Encode message in picture
35 coded_pic = coding(MESSAGE, GRAY_FILE, Mapset);
36 ENCODED_FILE = FILE_TO_READ + "_encoded" + FORMAT;
37 imwrite(coded_pic,ENCODED_FILE);
38
39 % Show both original and coded picture
40 figure;
41 subplot(1,2,1);
42 imshow(pic_gray);
43 title('Original PIC');
44 subplot(1,2,2);
45 imshow(coded_pic);
46 title('Coded PIC');
```

---

همانطور که دیده می‌شود تغییرات تصویر با چشم قابل تشخیص نیست. این به دو دلیل است. یکی اینکه ما بلوک هایی را انتخاب کردیم که واریانس آن ها بیشتر از یک مقدار مشخصی بود و دلیل دیگر این که ما بی ارزش ترین بیت هر پیکسل را تغییر می‌دهیم این یعنی مقدار عددی پیکسل نهایتاً یکی کم یا یکی زیاد می‌شود که با چشم قابل مشاهده نیست.



**تمرین 4-1**) تابع decoding سه پارامتر عکس حاوی مسیح، mapset و مقدار threshold را می‌گیرد.  
در ابتدا مانند بخش قبل عمل می‌کنیم و با استفاده از threshold که به عنوان پارامتر وارد شده، بلوک‌های 25 در 25 که واریانس آن‌ها بیشتر از مقدار threshold است را استخراج می‌کنیم و سپس تعداد آن‌ها را گزارش می‌کنیم تا ببینیم به چه تعداد هر دو تابع، بلوک‌ها را تشخیص داده‌اند یا خیر.

```
131
132 %% part 4
133 function decoded_msg = decoding(coded_pic, mapset, threshold)
134
135     % Read the encoded grayscale image
136     imgGray = imread(coded_pic);
137     [rows, cols] = size(imgGray); % Image dimensions
138
139     % Define a list to store selected boxes with variance above the threshold
140     selectedBlocks = [];
141
142     % Loop over the image using a sliding 25x25 window to find areas with variance above threshold
143     for i = 1:25:(rows - 24)
144         for j = 1:25:(cols - 24)
145             % Extract a 25x25 window and calculate its variance
146             window = imgGray(i:i+24, j:j+24);
147             variance = var(double(window(:)));
148
149             % Select the box if its variance is above the threshold
150             if variance > threshold
151                 selectedBlocks = [selectedBlocks; i, j];
152             end
153         end
154     end
155
156     % Display the number of selected 25x25 boxes
157     disp(['Selected 25x25 boxes for decoding: ', num2str(size(selectedBlocks, 1))]);
158
```

در ادامه باید پیام را از این بلوک ها استخراج کنیم. یک رشته داریم که قرار است مسیج باینری در آن قرار بگیرد و یک فلگ داریم که نشان دهنده این است که هنوز بیتی برای استخراج مانده یا خیر. سپس یک حلقه for روی بلوک ها داریم که در ابتدای این حلقه چک می کنیم که آیا تمام بیت های پیام را استخراج کردیم یا خیر. در ادامه مانند تابع coding روی پیکسل های بلوک حلقه می زنیم و کم ارزش ترین بیت هر پیکسل را در انتهای binary\_msg قرار می دهیم. سپس چک می کنیم که اگر بیتی که خواندیم 5 امین بیت کاراکتر بود، 5 بیت آخر binary\_message را برداریم و اگر آن را به علاوه یک کنیم ایندکس متغیر کاراکتر آن در mapset به دست می آید و آن کاراکتر را به decoded\_msg\_chars اضافه می کنیم. در صورتی هم که کاراکتر خوانده شده سمیکالان باشد، flag را به false تغییر می دهیم که از حلقه ها بیرون بیاییم چون

پیام تمام شده است.

```

159 % Initialize an array to store decoded binary message
160 binary_message = '';
161 flag = true; % flag for knowing if still we have to decode or not
162
163 % Start decoding the embedded message from each selected box
164 for block = 1:size(selectedBlocks, 1)
165     if ~flag
166         break; % Stop if the end of the message is reached
167     end

```

```

165 - if ~flag
166 -     break; % Stop if the end of the message is reached
167 - end
168 % Get top-left corner of the current 25x25 box
169 i = selectedBlocks(block, 1);
170 j = selectedBlocks(block, 2);
171 % Extract bits from the least significant bits of each pixel in the 25x25 box
172 for m = 0:24
173     for n = 0:24
174         if ~flag
175             break; % Stop if the end of the message is reached
176         end
177         % Get pixel value and extract the least significant bit
178         pixel_val = imgGray(i + m, j + n);
179         pixel_bin = dec2bin(pixel_val, 8); % Convert to 8-bit binary
180         binary_message = [binary_message, pixel_bin(end)]; % Append LSB to message
181
182         % Check if we've completed a 5-bit character
183         if mod(length(binary_message), 5) == 0
184             % Convert 5 bits to a character based on mapset
185             char_bin = binary_message(end-4:end); % Get the last 5 bits
186             char_idx = bin2dec(char_bin) + 1; % Convert binary to decimal index
187             % Check if the character is the terminator ':'
188             if strcmp(mapset{1, char_idx}, ':')
189                 flag = false; % End of message
190                 decoded_msg_chars{(length(binary_message) / 5)} = mapset{1, char_idx};
191             else
192                 % Append character to the decoded message
193                 decoded_msg_chars{(length(binary_message) / 5)} = mapset{1, char_idx};
194             end
195         end
196     end

```

در آخر هم کاراکتر ها را به هم می‌چسبانیم و پیام decode شده را نمایش می‌دهیم.

```
199 % Convert cell array to string and display the decoded message
200 decoded_msg = strjoin(decoded_msg_chars, '');
201 disp(['Decoded Message: ', decoded_msg]);
202
203 end
204
```

با صدا کردن این تابع، از درستی کد خود مطمئن می‌شویم.

```
48
49 %% test part 4
50 decoded_msg = decoding(ENCODED_FILE,Mapset, 1000);|
51
52
```

خروجی ها به صورت زیر خواهد بود. همانطور که انتظار می‌رفت هر دو تابع 97 بلوک را با threshold مورد نظر یافتند و با توجه به این که پیام به درستی decode شده، هر دو تابع درست کار می‌کنند.

```
>> p1
Selected 25x25 blocks: 97
Selected 25x25 blocks for decoding: 97
Decoded Message: signal;
fr .. |
```

تمرین ۱-۵) با اضافه شدن نویز به تصویر رمزگاری شده، احتمالاً نمی‌توانیم پیام را رمزگشایی کنیم. دلیل آن این است که ما با تغییر دادن یک بیت از بعضی پیکسل ها، پیام را در تصویر جا گذاری کردیم و این نویز ممکن است دوباره این بیت را تغییر دهد و این باعث می‌شود که رمز نگاری ما خراب شود و مثلاً به جای کاراکتر a، کاراکتر b را رمزگشایی کنیم. اگر نویز روی پیکسل های رمزگاری شده کاراکتر سمیکalan بیفتد، ما انتهای پیام هم نمی‌توانیم تشخیص دهیم و عملاً پیام غیر قابل دسترس می‌شود. پس اگر نویز به تصویر رمزگاری شده اضافه شود، امکان رمزگشایی درست، به شدت پایین می‌آید.

---

## بخش دوم

در این بخش مستقیما وارد توضیح کدها می‌شویم و توضیحات تئوری را در قالب توضیح کدها بیان می‌کنیم:

در ابتدا تصویر مدار و تصویر IC مدنظر را خوانده و به تابعی که نوشتمی دهیم.

```
close all;
clc; clear;

% Read images
A = imread('PCB.jpg');
B = imread('IC.png');

ICrecognition(A, B);
```

حالا به توضیح تابع ICrecognition می‌پردازیم:

- در ابتدا تصویر PCB و IC را که توسط آرگومان ها به آن دادیم را بدون درنگ می‌کشد و در subplot یک جای خالی برای تصویر PCB با IC های یافت شده، می‌گذارد.
- در نهایت هم تابع hold on صدای زده شده که یعنی همین تصویرهای کشیده شده را نگه دار تا مربع های تطبیقی هم پیدا شده و رسم شوند.

```

function ICrecognition(A, B)

    figure;

    % Display PCB image
    subplot(2, 2, 1);
    imshow(A);
    title('PCB Image');

    % Display IC template image
    subplot(2, 2, 2);
    imshow(B);
    title('IC Image');

    drawnow; % Display changes immediately

    % Display matching result
    subplot(2, 2, [3, 4]);
    imshow(A);
    title('Matching Result');
    hold on;

```

- حالا توسط تابع `recognizeAndGetRectangles` که در ادامه توضیحش می‌دهیم، مربع‌های نشان دهنده IC‌ها را در `rectangles_original` ریخته، سپس تصویر IC را 180 درجه می‌چرخانیم با تابع `imrotate` و دوباره همان تابع `recognizeAndGetRectangles` را جهت تطبیق دادن و دادن اطلاعات مربع‌های نهایی روی عکس چرخیده شده صدای زنیم.

- حالا لیست مربع‌های تطبیق یافته روی تصویر اصلی و تصویر چرخیده را مرج می‌کنیم، سپس آنها را توسط تابع `no_overlapping` فیلتر می‌کنیم تا از بین مربع‌هایی که از یک حدی بیشتر تداخل دارند، تنها مربعی که بیشترین مقدار کورلیشن و مچ بودن را دارد، باقی بماند. مگرنه دور هر IC که یافت می‌شد، چندین مربع با فاصله‌ی کم مشاهده می‌شد چون مقدار کورلیشن آنها خیلی تفاوتی نداشت با هم.

- نهایتاً هم مربع‌های یافت شده، روی تصویر `pcb` کشیده می‌شوند. نهایتاً هم منظور از این است که چون هر مربع را در اصل با 4 ویژگی ایندکس کف و ایندکس سمت چپو طول و عرضشان نمایش می‌دهیم، پس هر مربع یک لیست است و اینجا هم لیست

---

مربوط به مربع  $a \times b$  به تابع داده می شود و آن هم به رنگ آبی و ضخامت داده شده نمودار را می کشد.

```
% Detect ICs and draw rectangles
rectangles_original = recognizeAndGetRectangles(A, B);

% Rotate the template by 180 degrees and detect again
ic_rotated = imrotate(B, 180);
rectangles_rotated = recognizeAndGetRectangles(A, ic_rotated);

% Combine rectangles from both original and rotated templates
all_rectangles = [rectangles_original; rectangles_rotated];

% Use Non-Maximum Suppression to eliminate overlapping rectangles
final_rectangles = no_overlapping(all_rectangles);

% Draw final rectangles on the matching result image
for k = 1:size(final_rectangles, 1)
    rectangle('Position', final_rectangles(k, :), 'EdgeColor', 'b', 'LineWidth', 2);
end

hold off;
end
```

- حالا به توضیح تابع `recognizeAndGetRectangles` می پردازیم.
- در ابتدای این تابع، طول سه بعد عکس `pcb` و `ic` را می باییم و سپس `ic` را از نوع داده `uint8` (که `double` است) به نوع `double` تبدیل می کنیم. دلیل اصلی این کار این است که محاسبات ریاضی مانند ضرب، جمع، یا محاسبه همبستگی به دقت و محدوده عددی بیشتری نیاز دارند که نوع داده `double` می تواند ارائه دهد.
- سپس یک لیست می سازیم که قرار است ویژگی های مستطیل های پیدا شده را در آن بریزیم و توسط دو حلقه `i` تو در تو، `sub-image` هایی به اندازه `ic` از درون `pcb` جدا کرده و تا اینجای کار، `a` همان `ic` است و `b` همان `sub-image` جدا شده از درون `pcb` به اندازه `ic` است.
- حالا برای تک تک کanal های سه گانه `i` رنگی `a` و `b` (اگر هر دو سیاه و سفید و دارای یک کanal باشند هم مشکلی نیست) یک حلقه زده و برای هر کanal رنگی از کanal های

R,G,B یک بار کورلیشن آن کانال را بین a و b حساب میکنیم با اعمال فرمول داده شده در پروژه. ( نکته اینکه تابع sum(sum)() بمعنى جمع زدن تمام درایه های یک ماتریس دو بعدی است که همان ماتریس کانال انتخاب شده در a یا b است. نهایتا هم بین سه تا کورلیشنی که برای سه کانال محاسبه کردیم، میانگین گرفته و بعنوان avg\_coff محاسبه میکنیم.

حالا اگر مقدار avg\_coff از threshold انتخابی (که مثلا ما 0.9 گذاشتیم) بیشتر بود، آن بخش را بعنوان بخشی که |c در آنجا وجود دارد در نظر گرفته و ویژگی های ان مریع را به لیست rectangles اضافه میکنیم.

```
% Detect ICs and draw rectangles
rectangles_original = recognizeAndGetRectangles(A, B);

% Rotate the template by 180 degrees and detect again
ic_rotated = imrotate(B, 180);
rectangles_rotated = recognizeAndGetRectangles(A, ic_rotated);

% Combine rectangles from both original and rotated templates
all_rectangles = [rectangles_original; rectangles_rotated];

% Use Non-Maximum Suppression to eliminate overlapping rectangles
final_rectangles = no_overlapping(all_rectangles);

% Draw final rectangles on the matching result image
for k = 1:size(final_rectangles, 1)
    rectangle('Position', final_rectangles(k, :), 'EdgeColor', 'b', 'LineWidth', 2);
end

hold off;
end
```

- حالا به توضیح تابع recognizeAndGetRectangles می پردازیم.

```

% Function to calculate correlation and return similar regions without drawing rectangles
function rectangles = recognizeAndGetRectangles(pcb, ic)
    [rowpcb, colpcb, chnls_pcb] = size(pcb);
    [rowic, colic, ~] = size(ic);

    % Convert the template to double for correlation calculation
    a = double(ic);

    % Variable to store rectangles and their correlation coefficients
    rectangles = [];

    % Loop to identify similar parts in the PCB image
    for i = 1:(rowpcb - rowic)
        for j = 1:(colpcb - colic)
            % Extract sub-image from the PCB
            b = double(pcb(i:(i + rowic - 1), j:(j + colic - 1), :));

            % Calculate correlation coefficient for each channel
            total_coff = 0;
            for c = 1:chnls_pcb
                a_channel = a(:, :, c);
                b_channel = b(:, :, c);

                % Calculate correlation coefficient for channel c
                q = sqrt(sum(sum(a_channel .* a_channel)) * sum(sum(b_channel .* b_channel)));
                p = sum(sum(a_channel .* b_channel));
                coff = p / q;

                % Sum correlation coefficients of all channels
                total_coff = total_coff + coff;
            end

            % Calculate average correlation coefficient
            avg_coff = total_coff / chnls_pcb;

            % Store rectangle if the average correlation coefficient is high enough
            if avg_coff >= 0.9
                rectangles = [rectangles; j, i, colic, rowic, avg_coff];
            end
        end
    end
end

```

- حالا به سراغ توضیحات تابع no\_overlapping می رویم که قرار است تمام مربع های تشخیص داده شده بعنوان تصویر ic یا تصویر بر عکس شده ic در تصویر pcb را

گرفته و اگر چندین مربع از یک حدی بیشتر تداخل داشتند، فقط یکی از آنها را که دارای بیشترین کورلیشن است در لیست نهایی بگذارد

- در این تابع ابتدا لیستی برای ریختن مشخصات نهایی مربع های برگزیده میسازیم و در یک حلقه، ابتدا مربعی با بیشترین کورلیشن را بعنوان اولین مربعی که حتماً انتخاب میشود در لیست نهایی گذاشته و سپس آنرا از لیست ورودی حذف میکند و در ادامه در یک حلقه تمام مربع های باقی مانده در لیست ورودی را با مربع برگزیده شده کورلیشن میگیرد و اگر مربعی دارای بیشتر از 0.3 تداخل بود، ایندکس آنرا در `overlapIdx` ذخیره میکند تا بعداً تمام این دسته مربع های تداخل دارنده را حذف کند و فقط مربع اول که دارای بیشترین میزان کورلیشن بود را نگه دارد.

- نهایتاً حلقه‌ی خارجی `while` تا وقتی لیست ورودی خالی شود و از بین تمام دسته مربع های متداخل، مربعی با بیشترین کورلیشن انتخاب شده و در لیست خروجی قرار بگیرد، ادامه می‌یابد

```
function final_rects = no_overlapping(rects)
    % Initialize an empty array for the final rectangles
    final_rects = [];

    while ~isempty(rects)
        % Select the rectangle with the highest correlation coefficient
        [~, max_idx] = max(rects(:, 5));
        current_rect = rects(max_idx, :);
        final_rects = [final_rects; current_rect];

        % Remove the selected rectangle
        rects(max_idx, :) = [];
        overlapIdx = [];

        % Identify rectangles that have significant overlap with the selected one
        for i = 1:size(rects, 1)
            if overlap(current_rect, rects(i, :)) > 0.3 % Threshold for overlap (30%)
                overlapIdx = [overlapIdx; i];
            end
        end

        % Remove overlapping rectangles
        rects(overlapIdx, :) = [];
    end
end
```

- در آخرین مرحله به نحوه‌ی محاسبه‌ی درصد overlap میان دو مربع میپردازیم که توسط تابع overlap انجام میشود و از آن در تابع قبلی و برای شناسایی مربع‌های متداخل استفاده کردیم
- در این تابع هر بار دو مربع دریافت میشوند و x\_overlap نمایانگر میزان تداخل عرضی این دو مربع و y\_overlap نشانگر تداخل ارتفاعی این دو مربع است که با توجه به لیست ویژگی‌های هر مربع و پارامتر‌های موجود در آن به راحتی محاسبه می‌شود (شکل مرتبط با آن را در ادامه میگذارم)
- سپس مساحت ناحیه‌ی overlap محاسبه میشود با ضرب میزان تداخل طولی و عرضی، و با تقسیم این مقدار بر مساحت مربع کوچکتر، در اصل به پارامتری که میتواند گویای میزان تداخل باشد، میرسیم و همین را بعنوان درصد تداخل اعلام کرده و همانطور که دیدیم، در تابع no\_overlapping از آن بعنوان threshold برای یافتن و حذف مربع‌های متداخل استفاده میکنیم.

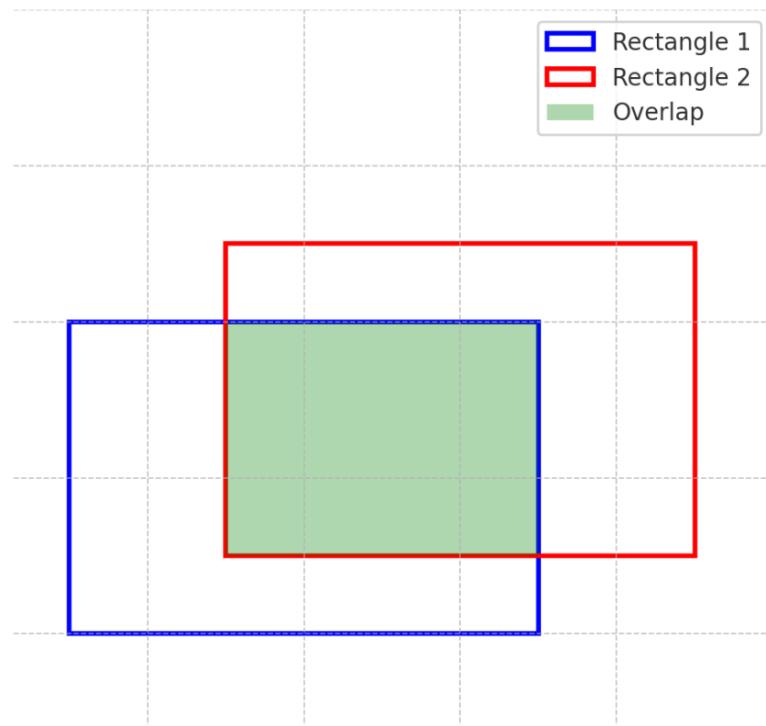
```
% Function to calculate the overlap ratio between two rectangles
function ratio = overlap(rect1, rect2)
    % Calculate overlap coordinates
    x_overlap = max(0, min(rect1(1) + rect1(3), rect2(1) + rect2(3)) - max(rect1(1), rect2(1)));
    y_overlap = max(0, min(rect1(2) + rect1(4), rect2(2) + rect2(4)) - max(rect1(2), rect2(2)));

    % Calculate overlap area
    overlap_area = x_overlap * y_overlap;

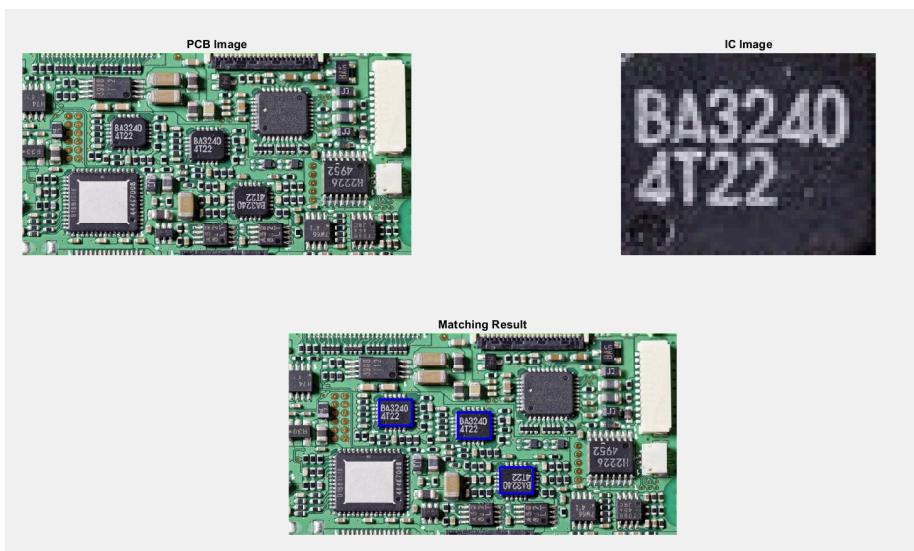
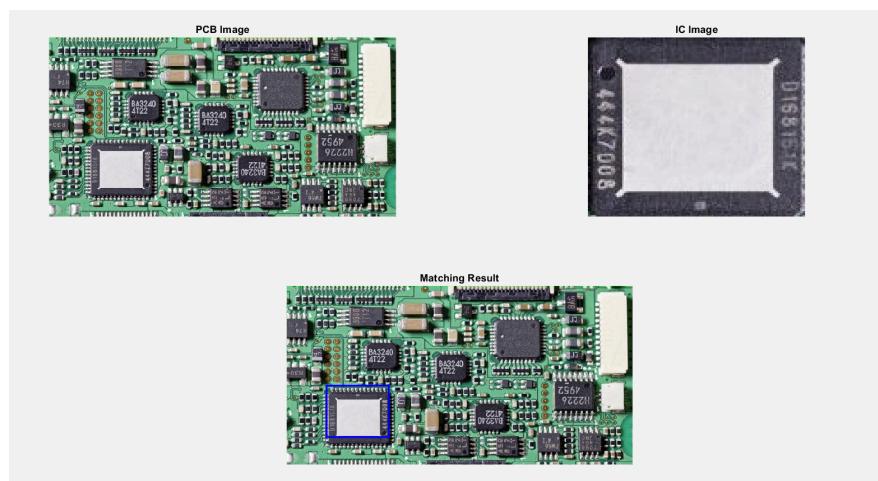
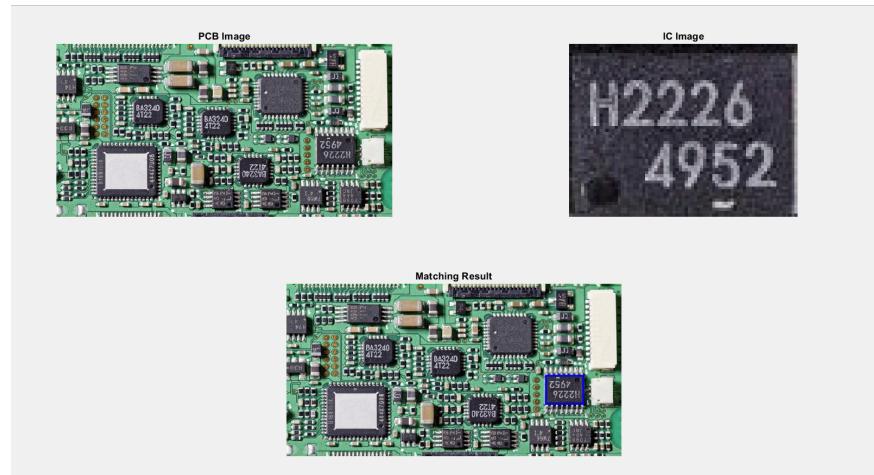
    % Calculate areas of both rectangles
    area1 = rect1(3) * rect1(4);
    area2 = rect2(3) * rect2(4);

    % Calculate the overlap ratio relative to the smaller rectangle area
    ratio = overlap_area / min(area1, area2);
end
```

منظور از overlap همچین چیزی است:

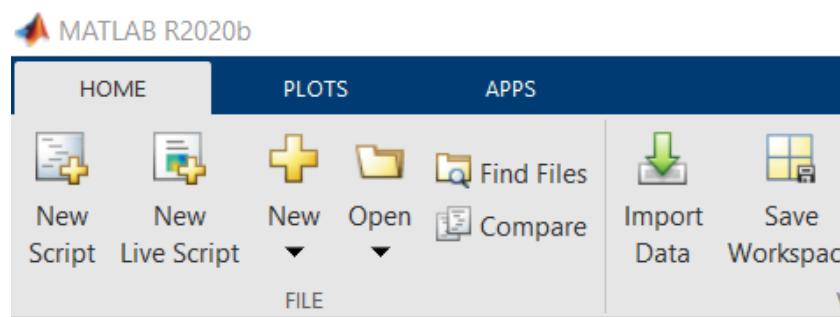


- نهایتا هم برنامه را روی چند IC مختلف روی تصویر pcb تست میکنیم:



### بخش سوم)

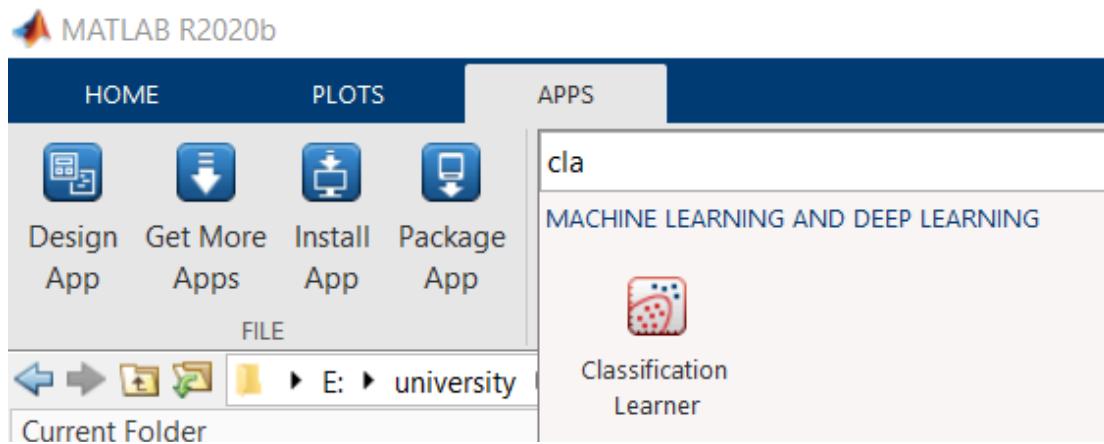
تمرین 3-1) ابتدا از نوار بالا بخش Home گزینه import data را انتخاب می‌کنیم.



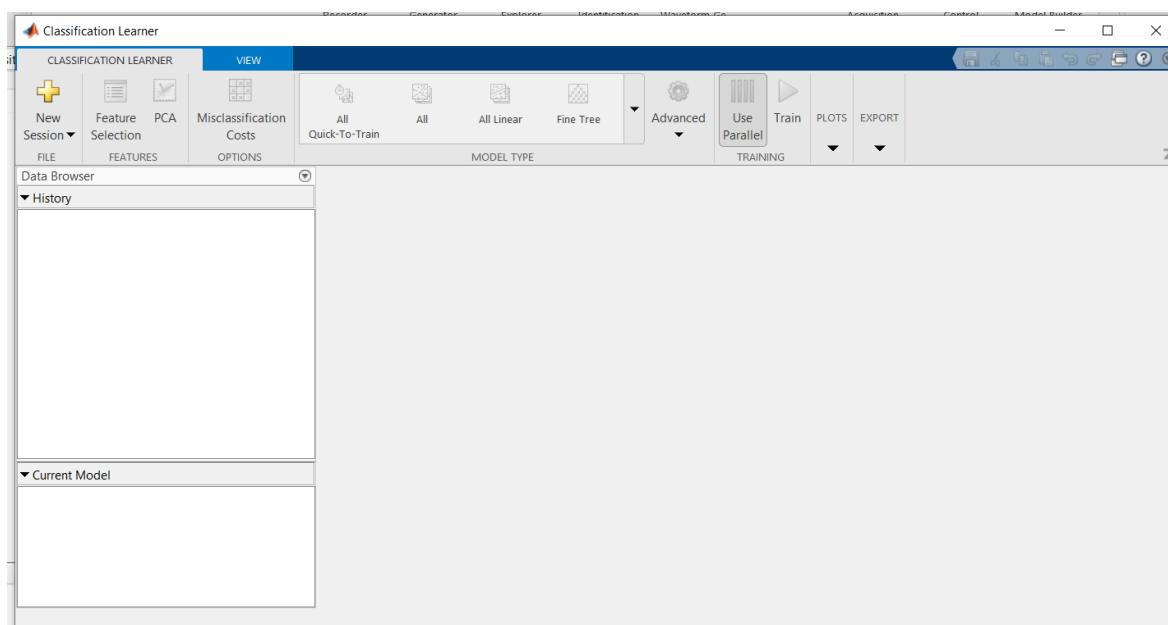
سپس صفحه‌ای باز می‌شود که دیتا در آن لود شده است.

The image shows the 'Import' dialog box for the file 'diabetes-training.csv'. The 'IMPORT' tab is selected. In the 'DELIMITERS' section, 'Comma' is chosen as the column delimiter. The 'SELECTION' section shows the range 'A2:G601'. The 'IMPORTED DATA' section displays a preview of the data in a table format. The table has columns labeled 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'Age', and 'label'. The data consists of 601 rows of numerical values. The 'Import Selection' button is highlighted with a green checkmark.

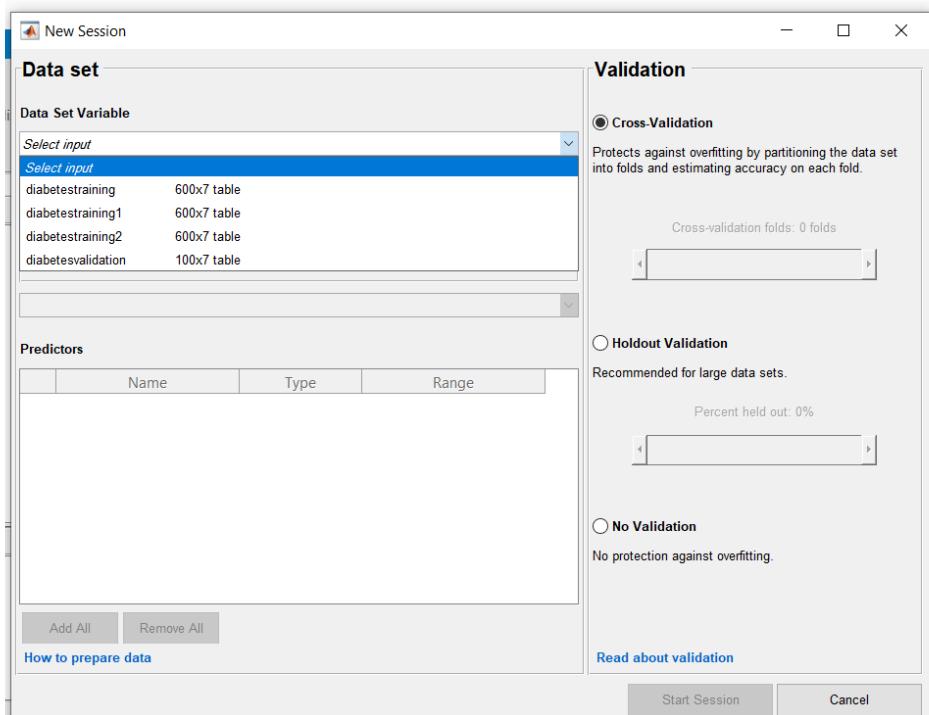
سپس در نوار بالا بخش APPS اپلیکیشن classification learner را انتخاب می‌کنیم.



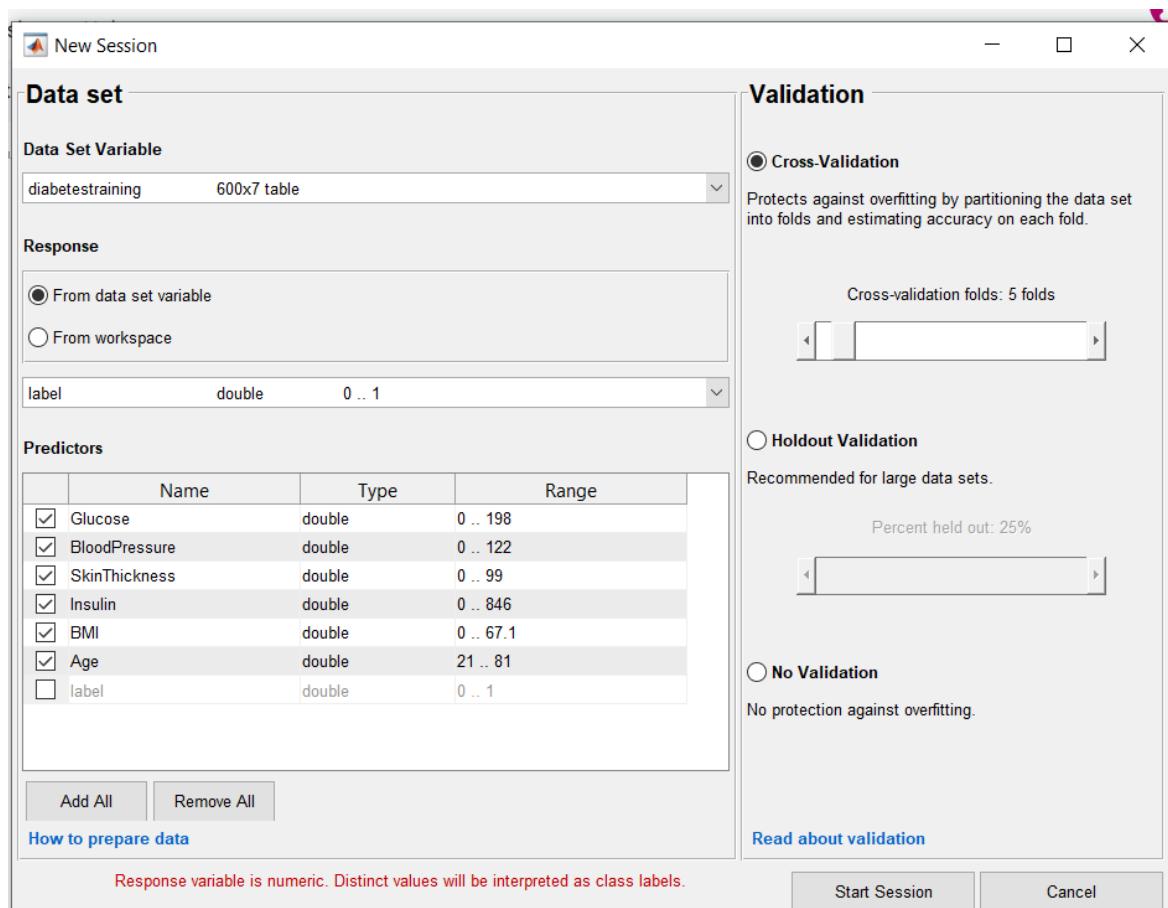
پنجره زیر باز می‌شود



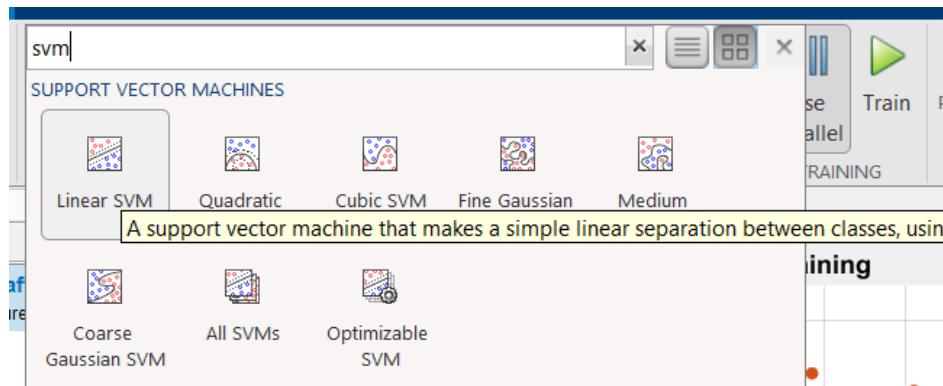
و با زدن دکمه new section پنجره دیتای مورد نظر را انتخاب می کنیم. با زدن روی لیست select لیست اینپوت هایی که وارد workspace کرده ایم نمایش داده می شود (من چندبار داده train را وارد کرده ام)



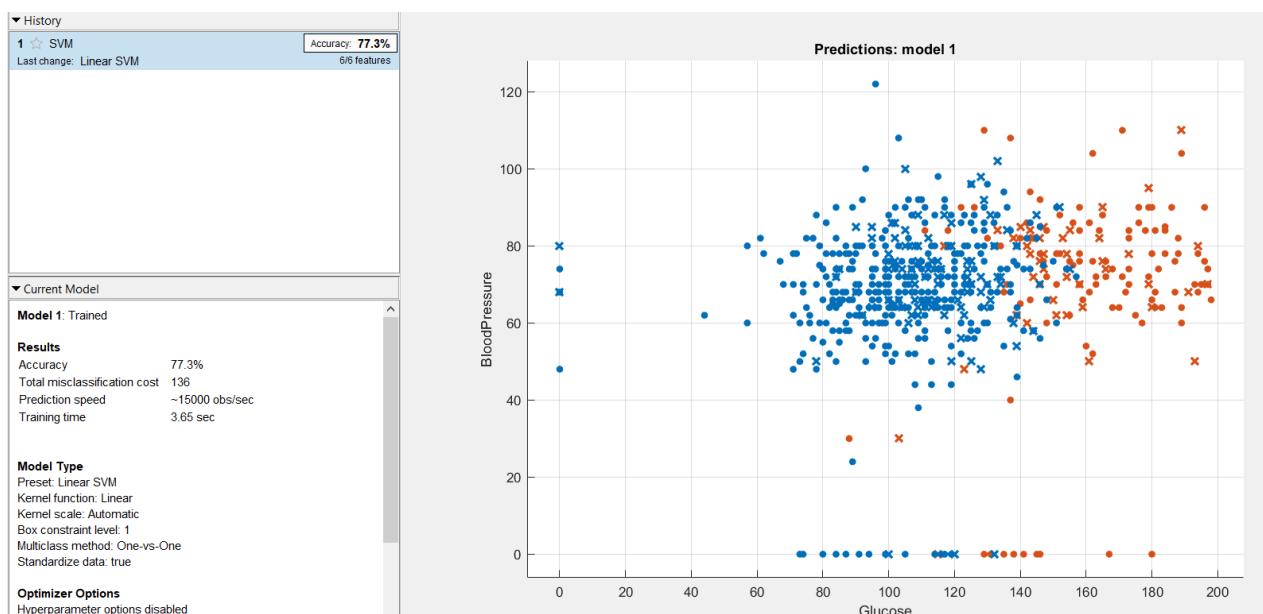
با زدن دکمه Import Section که در بالا دست راست تصویر قبل مشاهده می شود، دیتا را وارد اپلیکیشن می کنیم و start session را می زنیم.



سپس از منوی بالا linear SVM را انتخاب می کنیم و گزینه train را می زنیم.



همانطور که مشاهده می‌شود دقت مدل 77.3 درصد است. در نمودار هم داده‌ها نمایش داده شده اند که هر رنگ نماد یک کلاس است (نارنجی = 1 یا دیابت دار، آبی = 0 یا بدون دیابت) است و داده‌هایی که با دایره نشان داده شده اند به صورت درست پیش‌بینی شده اند و داده‌هایی که با ضربدر نمایش داده شده اند یعنی مدل درست آن‌ها را پیش‌بینی نکرده. (این نمودار به صورت دیفالت بر روی فشار خون بر حسب گلوکز نمایش داده شده است.)

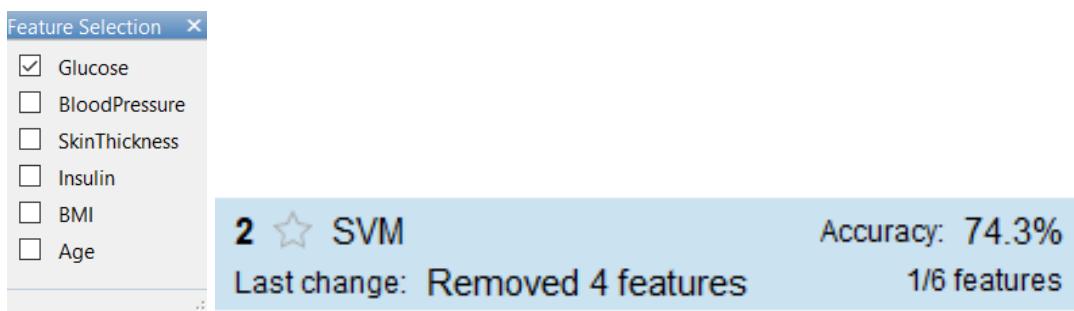


---

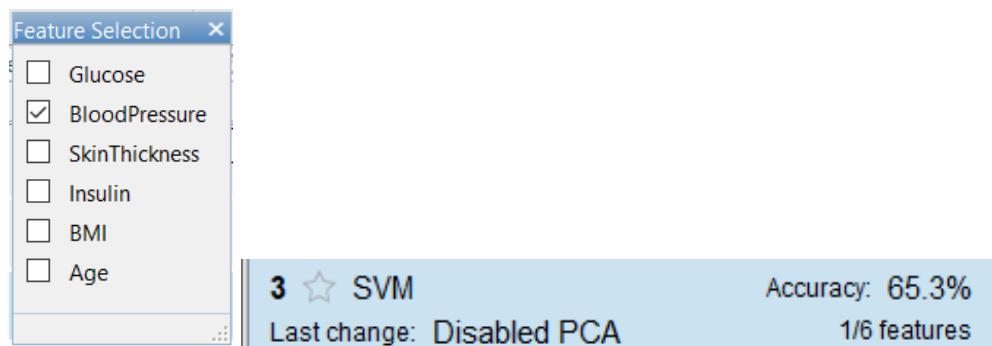
### (2-3) تمرین

مشاهده شد که بیشترین ربط به دیابتی بودن را فیچر Glucose دارد. و استفاده از آن به تنهایی 74.3% دقت به ما می دهد.

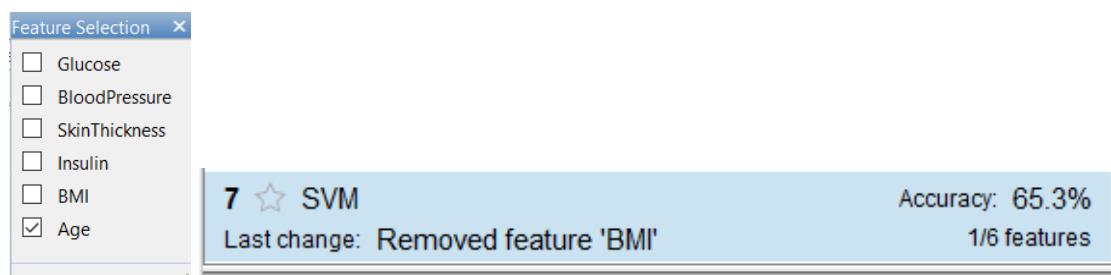
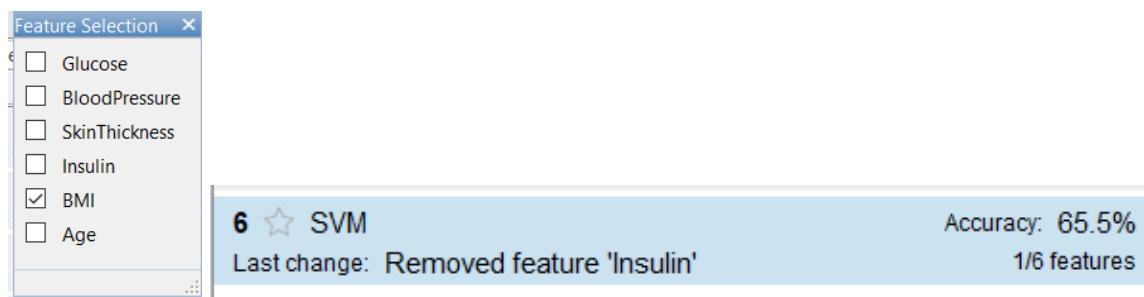
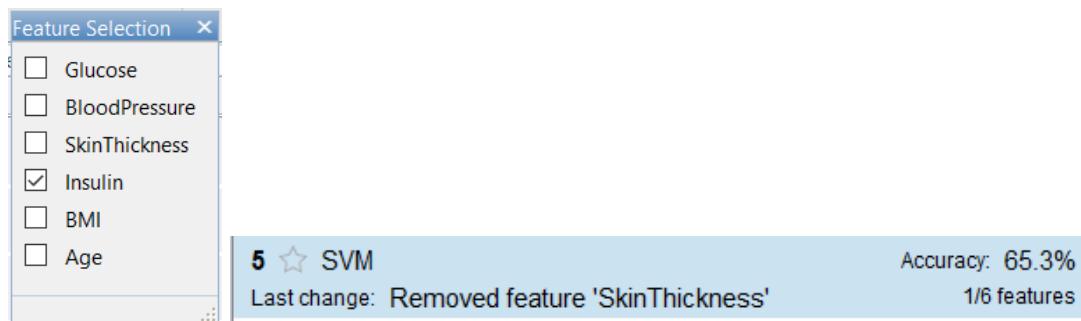
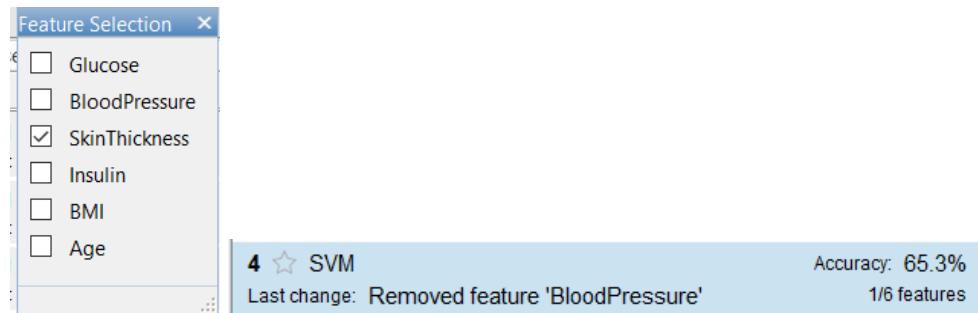
(1)



(2)

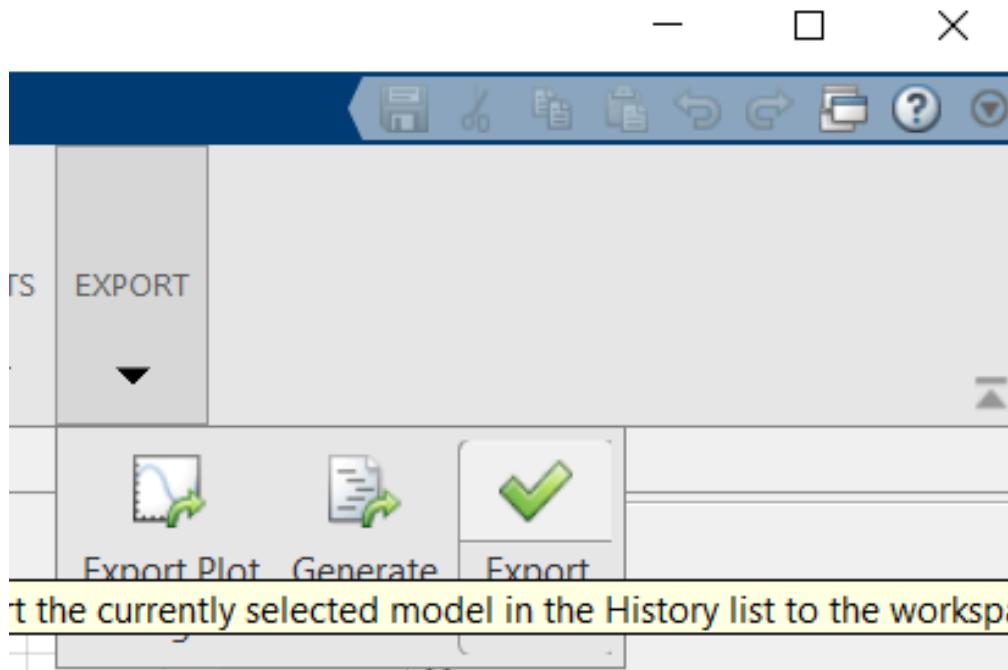


(3)

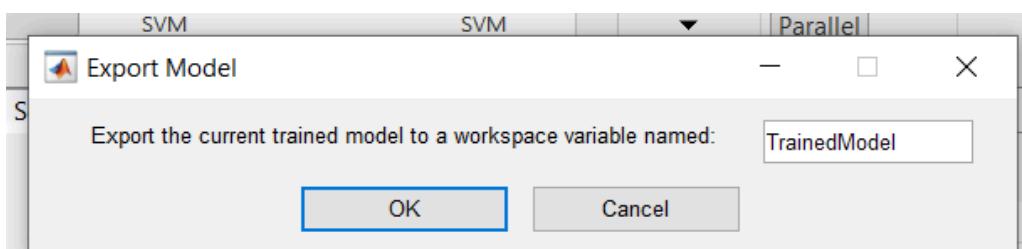


### (3-3) تمرین 3

از نوار بالا گزینه Export model را انتخاب می‌کنیم.



و با نام TrainedModel ذخیره می‌کنیم.



حال در فایل p3.m با استفاده از مدلی که export کردیم دوباره داده train را می‌دهیم تا پیش‌بینی کند.  
ستون های یک تا 6 فیچر ها و ستون 7 لیبل است) سپس یک آرایه pred\_right داریم که اگر لیبل ۱ است  
درست پیش‌بینی شده باشد، در ایندکس ۱ یک و در غیر این صورت صفر قرار می‌دهیم. سپس تعداد درست ها  
که برابر با مجموع یک های موجود در این آرایه است را تقسیم بر تعداد کل دیتا می‌کنیم تا accuracy را به  
دست آوریم.

```
1 %% part 3
2 - pred_y = TrainedModel.predictFcn(diabetestraining(:,1:6));
3 - org_y = table2array(diabetestraining) ;
4 - org_y = org_y(:,7) ;
5 - pred_right = pred_y==org_y ;
6 - accuracy = (sum(pred_right)/numel(pred_right))*100
7
8
```

نتیجه هم به صورت زیر است که بسیار به نتیجه 3-1 نزدیک است.

```
>> p3

accuracy =

77.5000
```

*fx* >>

تمرین 4-3) در آخر هم با همان روش 1-3 دیتای validation را import می‌کنیم و با استفاده از همان قطعه کد بخش قبل این بار برای دیتای diabetesvalidation، دقت یا همان accuracy را محاسبه می‌کنیم.

```
9 %% part 4
10
11 - pred_y2 = TrainedModel.predictFcn(diabetesvalidation(:,1:6));
12 - org_y2 = table2array(diabetesvalidation) ;
13 - org_y2 = org_y2(:,7) ;
14 - pred_right2 = pred_y2==org_y2 ;
15 - accuracy2 = (sum(pred_right2)/numel(pred_right2))*100
16
```

خروجی این بخش هم به صورت زیر است.

```
accuracy2 =
78
```