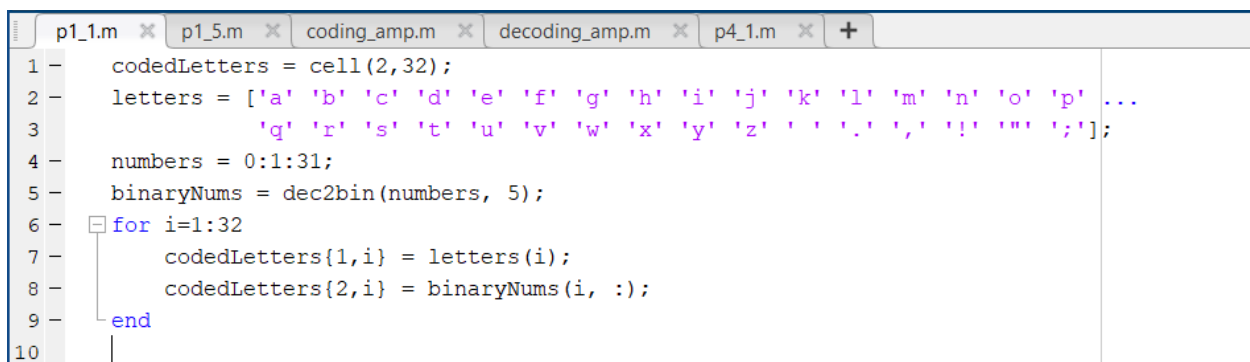


## پروژه ی چهارم سیگنال و سیستم ها

### بخش اول)

(1-1)

برای این قسمت، مشابه پروژه قبل عمل میکنیم. که کد آن به شکل زیر میباشد:



```

1 codedLetters = cell(2,32);
2 letters = ['a' 'b' 'c' 'd' 'e' 'f' 'g' 'h' 'i' 'j' 'k' 'l' 'm' 'n' 'o' 'p' ...
3           'q' 'r' 's' 't' 'u' 'v' 'w' 'x' 'y' 'z' ' ' '!' ' " ' ; '];
4 numbers = 0:1:31;
5 binaryNums = dec2bin(numbers, 5);
6 for i=1:32
7     codedLetters{1,i} = letters(i);
8     codedLetters{2,i} = binaryNums(i, :);
9 end
10

```

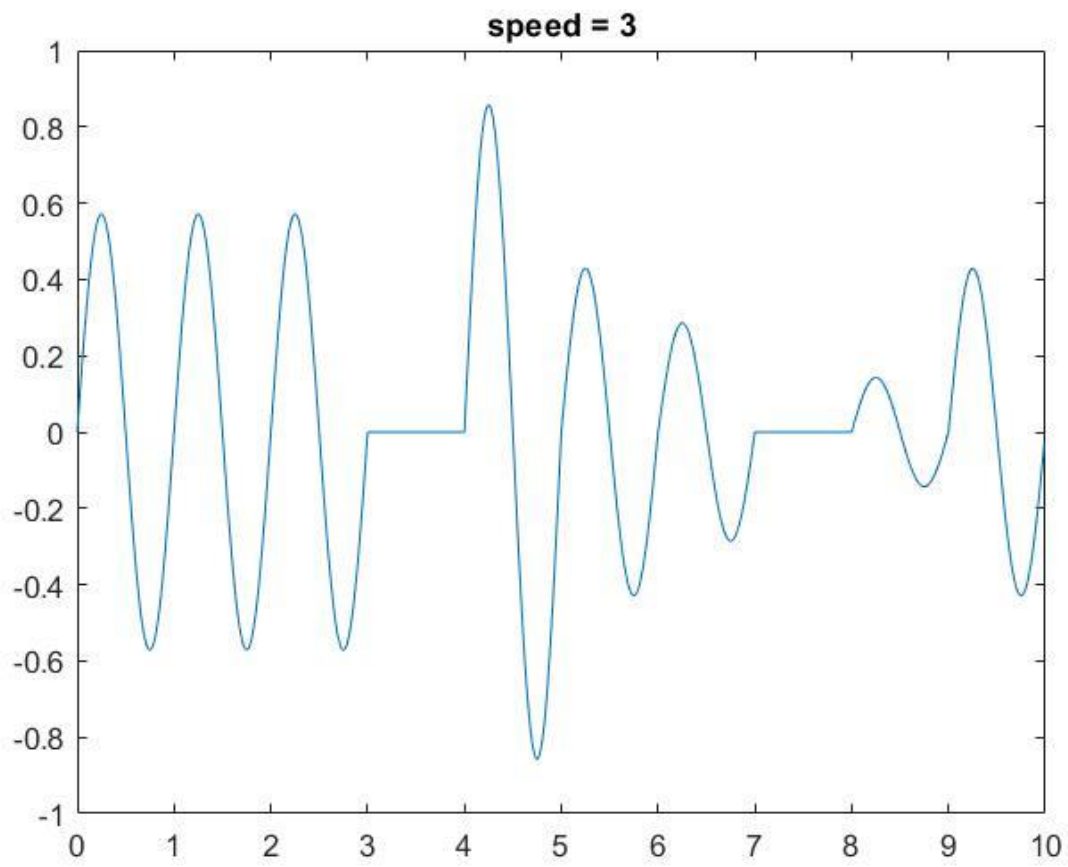
(2-1)

در این قسمت، روش پیاده سازی این است که ابتدا پیام داده شده را به باینری تبدیل کنیم. سپس به ازای هر speed حرف این باینری مسیج، باید یک سیگنال که ضربی از  $\frac{1}{2^{speed}-1} \sin(2\pi t)$  است، جنریت شود. ضریب گفته شده، برابر است با مقدار آن عدد باینری ای که این speed حرف نمایش دهنده آن است. برای مثال، اگر speed برابر 3 باشد، و بخواهیم پیام 011110 را ارسال کنیم، در یک ثانیه اول، سیگنال را توسط  $\frac{3}{2^3-1} \sin(2\pi t)$  و ثانیه دوم را توسط  $\frac{6}{2^3-1} \sin(2\pi t)$  جنریت میکنیم. و در نهایت این دو سیگنال را به یکدیگر میچسبانیم. پیاده سازی این توضیحات در تصویر زیر آمده است.

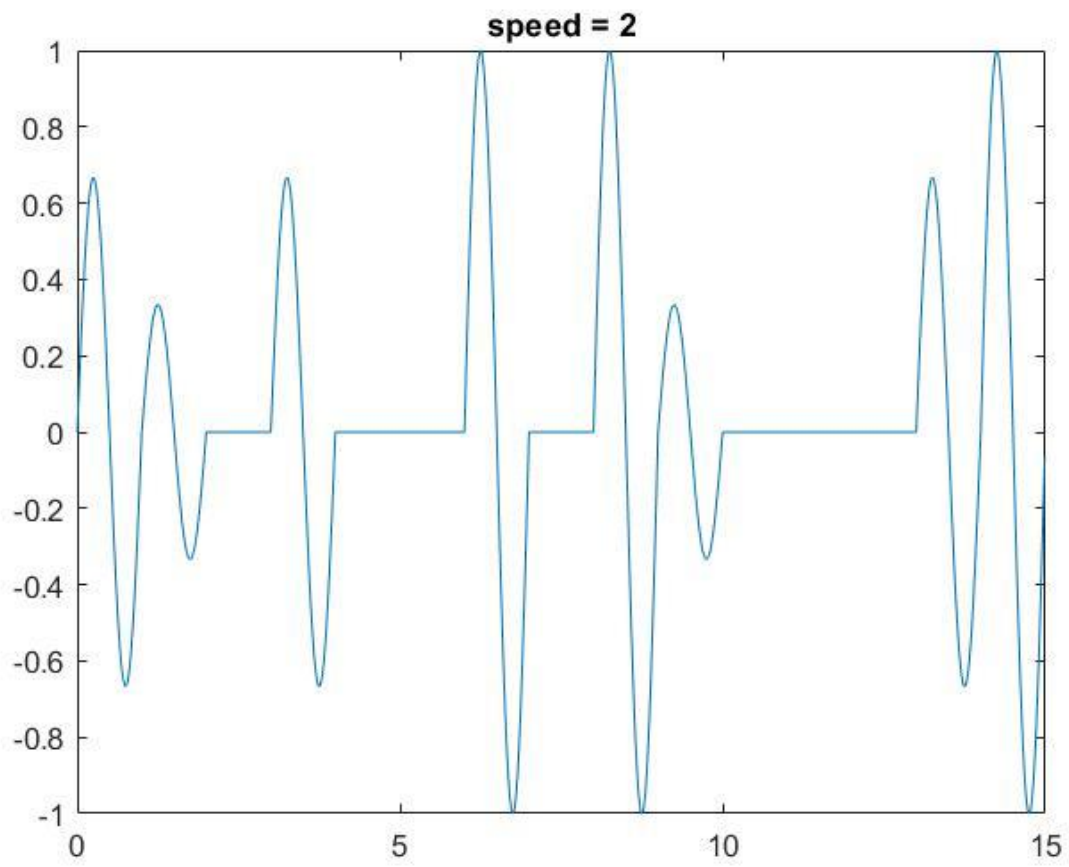
```
p1_1.m x p1_5.m x coding_amp.m x decoding_amp.m x p4_1.m x +
1 function signal = coding_amp(mapSet, message, speed)
2     binarizedMessage = [];
3     for i=1:size(message,2)
4         for j=1:size(mapSet,2)
5             if(message(1,i)==mapSet{1,j})
6                 binarizedMessage = [binarizedMessage mapSet{2,j}];
7             end
8         end
9     end
10    signal = [];
11    fs = 100;
12    t = 0:1/fs:0.99;
13    for i=1:speed:size(binarizedMessage, 2)
14        this = binarizedMessage(i:i+speed-1);
15        appendSignal = 1/(2^speed-1)*sin(2*pi*t) * bin2dec(this);
16        signal = [signal appendSignal];
17    end
18 end
```

(3-1

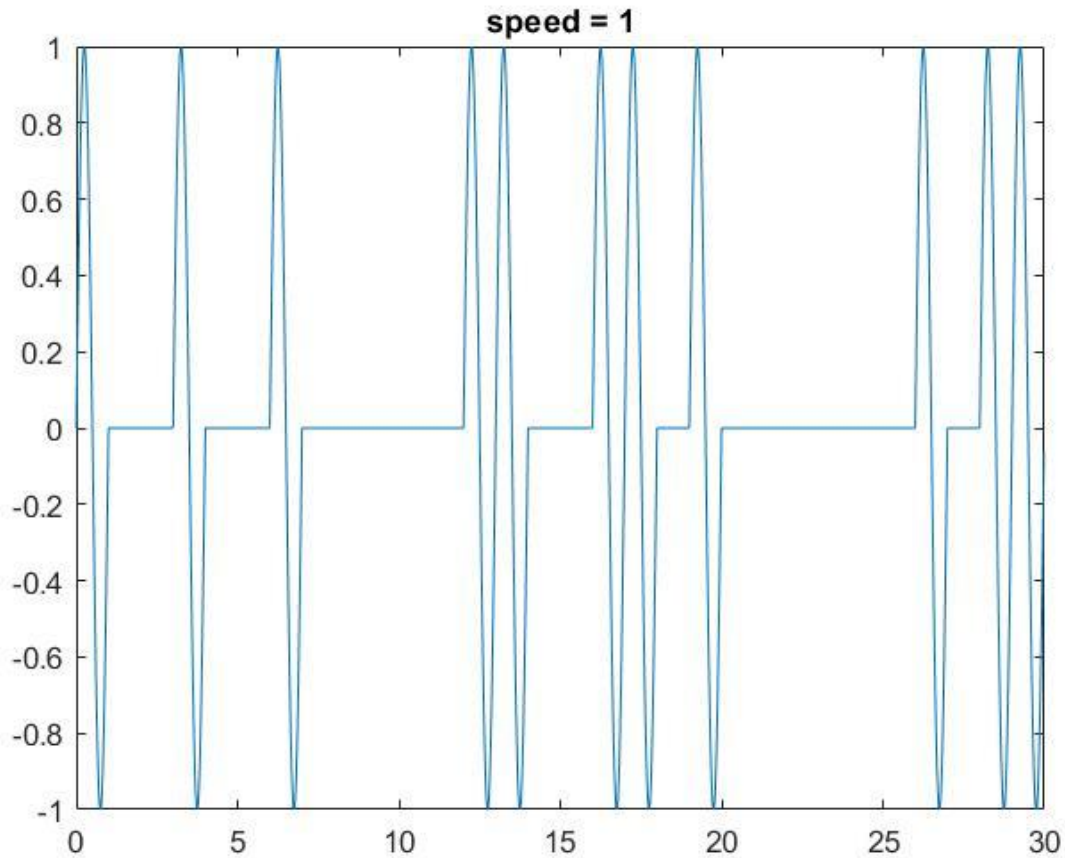
خروجی به ازای سرعت 3:



خروجی به ازای سرعت 2:



خروجی به ازای سرعت 1:



(4-1)

ایده ی حل این قسمت، این است که ابتدا یک قسمت از سیگنال به طول یک ثانیه را جدا می کنیم و با  $\sin(2\pi t)$  ضرب داخلی می کنیم. حال آن را ضربدر  $2/100$  می کنیم که مقدار آن نرمالایز شود و به ضریب خواسته شده برسیم. این ضریب در واقع همان  $\frac{k}{2^{speed}-1}$  می باشد که برای رسیدن به مقدار  $k$ ، آن را در مخرج آن ضرب می کنیم. با تبدیل  $k$  به باینری، آن  $speed$  حرف را بدست می آوریم. حال اگر این حرف ها را به ازای تمامی سیگنال های به طول 1، به یکدیگر بچسبانیم، به پیام باینری مورد نظر میرسیم. ازینجای ماجرا به بعد، مانند پروژه قبل می باشد که آن را در فایل `decoding_amp.m` مشاهده می کنیم.

```
p1_1.m x p1_5.m x coding_amp.m x decoding_amp.m x p4_1.m x +
1 function message = decoding_amp(mapSet, signal, speed)
2     fs = 100;
3     t = 0:1/fs:0.99;
4     genSin = sin(2*pi*t);
5     corrValues = [];
6     for i=1:fs:size(signal, 2)
7         partSignal = signal(i:i+fs-1);
8         corr = dot(genSin, partSignal);
9         corrValues = [corrValues 2/100*corr];
10
11     end
12
13     binarizedMessage = [];
14     for i=1:size(corrValues, 2)
15         corrVal = round(corrValues(i) * (2^speed - 1));
16         binarizedMessage = [binarizedMessage dec2bin(corrVal, speed)];
17     end
```

با استفاده از کد زیر، تابع ساخته شده را به ازای سرعت های مختلف امتحان میکنیم که همانطور که مشخص است، برای همه ی سرعت ها، به درستی کار می کند:

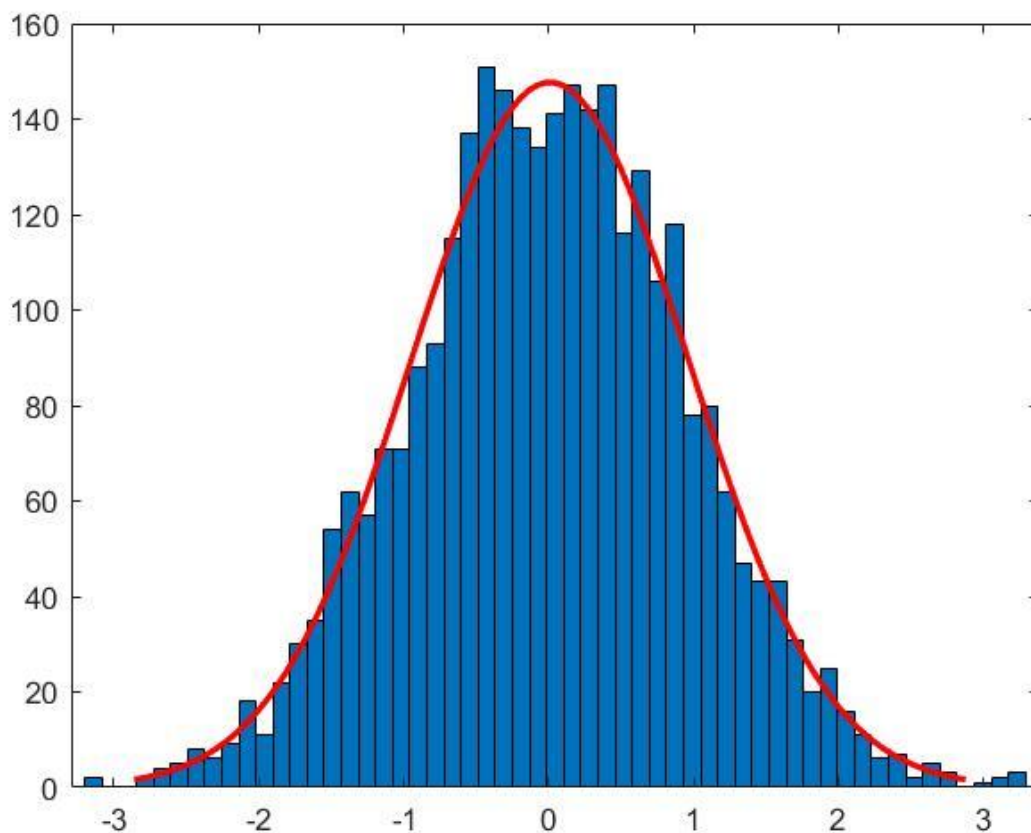
```
28 message1 = decoding_amp(codedLetters, signal3x, 3);
29 message2 = decoding_amp(codedLetters, signal2x, 2);
30 message3 = decoding_amp(codedLetters, signal1x, 1);
31 fprintf('decoded message for speed 3: %s \n', message1);
32 fprintf('decoded message for speed 2: %s \n', message2);
33 fprintf('decoded message for speed 1: %s \n', message3);
34
```

---

```
Command Window
>> p1_1
decoded message for speed 3: signal
decoded message for speed 2: signal
decoded message for speed 1: signal
fx >>
```

(5-1)

برای نشان دادن اینکه خروجی تابع randn، گوسی میباشد، لازم است که histogram آن را رسم کنیم، که خروجی آن برای randn(1, 3000) به شکل زیر می باشد:



که مشاهده می شود توزیع نرمال داریم.

برای نشان دادن میانگین و واریانس، توسط کد زیر میانگین و واریانس را محاسبه می کنیم:

```
p1_1.m x p1_5.m x coding_amp.m x decoding_amp.m x p4_1.m x +
1 - x = 1:3000;
2 - y = randn(1,3000);
3 - histfit(y);
4 - mean = sum(y)/length(y);
5 - variance = sum(y.*y)/length(y) - mean^2;
6 - fprintf('calculated mean on randn function is: %f \n', mean);
7 - fprintf('calculated variance on randn function is: %f \n', variance);

Command Window
>> p1_5
calculated mean on randn function is: 0.008023
calculated variance on randn function is: 0.913867
fx >>
```

مشاهده میشود که میانگین برابر 0.008 و واریانس برابر 0.913 بدست آمده که باید به ترتیب 0 و 1 بدست می آمد. علت این ماجرا این است که randn، تابعی تصادفی است و لزوماً به این مقادیر دقیق نمیرسیم. اما با تقریب خوبی، توانستیم این ماجرا را نمایش دهیم.

## (6-1)

در این قسمت، ابتدا کد مربوط به decoding\_amp را به گونه ای آپدیت میکنیم که برای حالت های نویزی، به درستی کار کند. به این صورت که همانگونه که در ابتدای پروژه ذکر شده بود، آستانه های مختلف تعریف میکنیم و چک می کنیم correlation به دست آمده در کدام بازه قرار دارد و عدد مربوط به آن را ازینجا پیدا می کنیم. قسمت های تغییر یافته در decoding\_amp، به شرح زیر است:



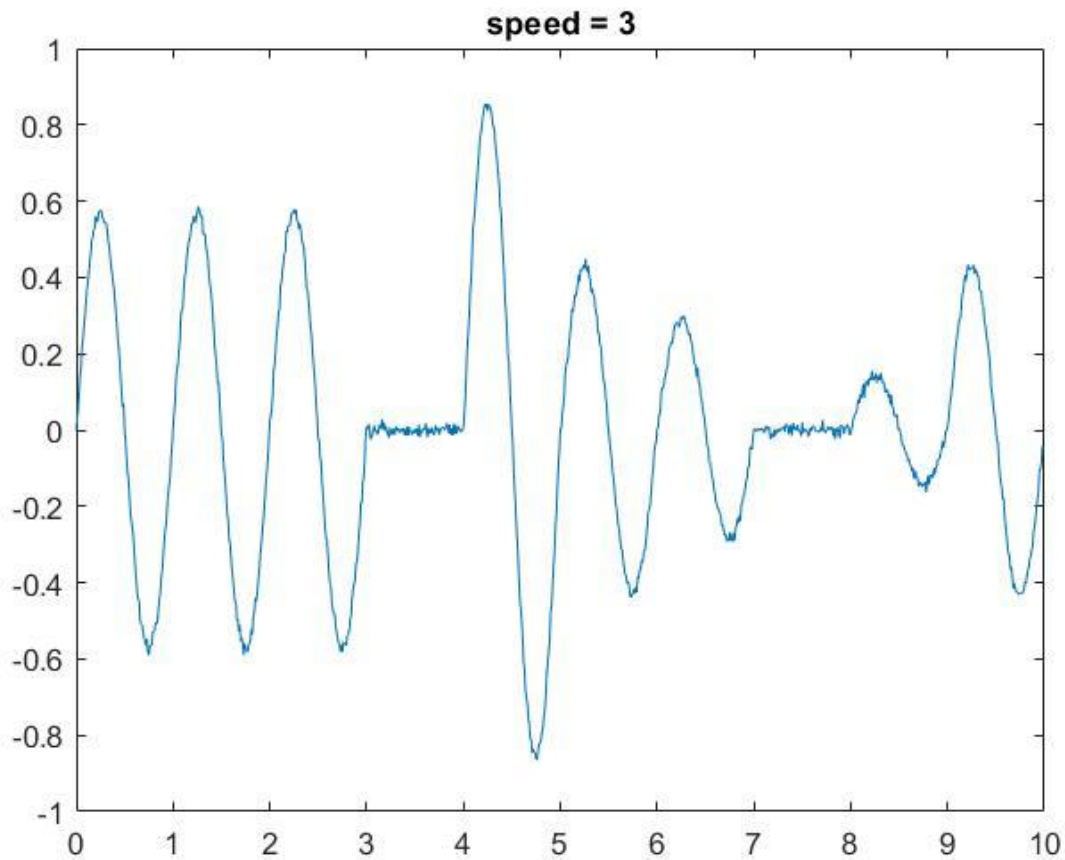
```

13 - edges = zeros(1, 2^speed-1);
14 - for i=0:2^speed-2
15 -     edges(i+1) = i + 1/2;
16 - end
17
18 - binarizedMessage = [];
19 - for i=1:size(corrValues, 2)
20 -     corrVal = corrValues(i) * (2^speed - 1);
21 -     findEx = 2^speed-1;
22 -     for j=1:size(edges, 2)
23 -         if(corrVal < edges(j))
24 -             findEx = j-1;
25 -             break;
26 -         end
27 -     end
28 -     binarizedMessage = [binarizedMessage dec2bin(findEx, speed)];
29 - end

```

که در آن، آستانه ها را در edge تعریف کرده ایم. و به در حلقه، به جای صرفا گرد کردن مقدار corrVal، آن را با استفاده از روش گفته شده در بالا بدست می آوریم.

حال سیگنال هایی که نویز به آن ها اضافه شده است را می سازیم:(در فایل p1\_6.m)  
سیگنال نویزی شده به ازای انحراف معیار 0.01 برای سرعت 3:



حال تابع خود را بر روی آن ها تست می کنیم و خروجی به شکل زیر می باشد:

```
decoded message for speed 3: signal
decoded message for speed 2: signal
decoded message for speed 1: signal
```

(7-1)

در اینجا مرحله به مرحله نویز را زیاد کرده و قسمت 1-6 را برای آن تکرار می کنیم:

```
noise power is: 0.100000
decoded message for speed 3: signal
decoded message for speed 2: signal
decoded message for speed 1: signal
```

```
noise power is: 0.150000  
decoded message for speed 3: signal  
decoded message for speed 2: signal  
decoded message for speed 1: signal  
>>
```

```
noise power is: 0.200000  
decoded message for speed 3: signal  
decoded message for speed 2: signal  
decoded message for speed 1: signal  
>>
```

```
noise power is: 0.230000  
decoded message for speed 3: signal |  
decoded message for speed 2: signal  
decoded message for speed 1: signal
```

```
noise power is: 0.250000  
decoded message for speed 3: sygnad  
decoded message for speed 2: signal |  
decoded message for speed 1: signal  
>> |
```

همانطور که مشاهده میشود، برای انحراف معیار 0.25، مسیج دیکود شده برای سرعت 3، اشتباه است.

حال دوباره پله پله آن را زیاد می کنیم:

```
noise power is: 0.350000  
decoded message for speed 3: signal  
decoded message for speed 2: signal  
decoded message for speed 1: signal  
>> |
```

```
noise power is: 0.450000
decoded message for speed 3: sihnal
decoded message for speed 2: signal
decoded message for speed 1: signal
>>

noise power is: 0.800000
decoded message for speed 3: oyhnad
decoded message for speed 2: signal
decoded message for speed 1: signal
>>

noise power is: 0.900000
decoded message for speed 3: sigjak
decoded message for speed 2: signil
decoded message for speed 1: signal
```

همانطور که مشاهده میشود، برای انحراف معیار 0.9، مسیج دیکود شده برای سرعت 2 هم اشتباه می شود.

حال دوباره پله پله آن را زیاد می کنیم:

```
noise power is: 1.200000
decoded message for speed 3: okmvql
decoded message for speed 2: smgnck
decoded message for speed 1: signal
>>

noise power is: 1.300000
decoded message for speed 3: nihnqt
decoded message for speed 2: kio!ap
decoded message for speed 1: signal
>>

noise power is: 1.500000
decoded message for speed 3: seerbd
decoded message for speed 2: qyemal
decoded message for speed 1: smgnil
>>
```

---

همانطور که مشاهده میشود، برای انحراف معیار 1.5، مسیج دیکود شده برای سرعت 1 هم اشتباه می شود.

در نتیجه، به همان چیزی که در ابتدا در سوال گفته شده بود، رسیدیم. یعنی با افزایش سرعت انتقال، مقاوم بودن پیام نسبت به نویز کاهش می یابد و نتایج کاملاً همخوانی دارند.

## (8-1)

همانطور که در تصاویر قسمت قبل مشخص است، بیشترین واریانسی که سیگنال با سرعت 3 به آن مقاوم است، تقریباً برابر 0.23، بیشترین انحراف معیاری که سیگنال با سرعت 2 به آن مقاوم است، تقریباً برابر 0.8، بیشترین واریانسی که سیگنال با سرعت 1 به آن مقاوم است، تقریباً برابر 1.3 می باشد.

## (9-1)

راهکار، دقیقاً در صورت پروژه ارائه شده است که به طور خلاصه این است که باید دامنه ی صدا را افزایش دهیم که آن مقادیر threshold، فاصله بیشتری از یکدیگر گرفته و در نتیجه مقاومت بیشتری نسبت به نویز ها خواهیم داشت. Trade off ای که در اینجا داریم، این است که power بیشتری برای تولید سیگنال نیاز است.

## (10-1)

به صورت تئوری، هر چقدر که بخواهیم، می توانیم بیت ریت را افزایش دهیم. منتها مشکلی که وجود دارد این است که تا جایی میتوانیم پیش برویم که اعداد، قابل تمایز در کامپیوتر باشند. چرا که برای مثال برای ذخیره ی یک عدد اعشاری به صورت floating point، تنها 32 بیت در اختیار داریم. اگر بیت ریت، خیلی خیلی زیاد شود، از آنجایی که دو سیگنال مختلف ممکن است بر یکدیگر منطبق بشوند، در حالی که با یکدیگر برابر نیستند، باعث میشود خطا رخ بدهد و ما نتوانیم تمایز دو سیگنال را تشخیص دهیم. بنابراین تا جایی میتوانیم پیش برویم که این مشکل به دلیل محدودیت های کامپیوتری، ظاهر نشود.

(11-2)

اگر دامنه را تغییر ندهیم و صرفاً سیگنال دریافتی را در مقدار 5 ضرب کنیم، هیچ تفاوتی ظاهر نمیشود چرا که با اینکار، درست است که دامنه را به 5 رسانده ایم اما قدرت نویز ها هم 5 برابر شده است. و در قدرت تشخیص ما هیچ تاثیری نمیگذارد.

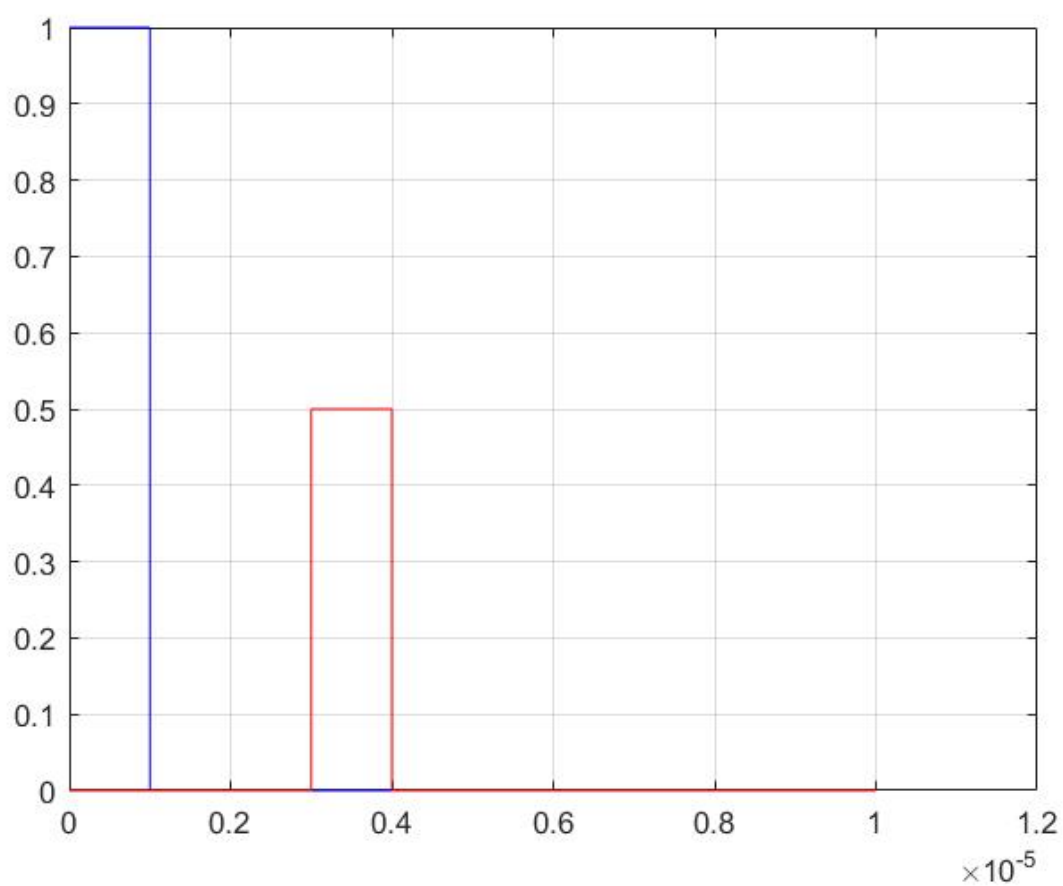
(12-2)

سرعت اینترنت خانگی به طور متوسط، برابر 8 مگابیت بر ثانیه است که برابر  $2^{23}$  بیت بر ثانیه می باشد. ما در این تمرین، اطلاعات را حداکثر با سرعت 3 بیت بر ثانیه ارسال کردیم که اختلاف بسیار زیادی است.

## بخش دوم)

(1-2)

این بخش دقیقاً مطابق پروژه قبل است و هیچ نکته خاصی ندارد. سیگنال ها ارسال و دریافت شده به شکل زیر هستند: (کد این بخش در p2.m نوشته شده است.)



(2-2)

در این بخش، بجای کورولیشن گرفتن بین سیگنال دریافت شده و شیفت یافته های سیگنال ارسال شده، از ایده کانولوشن گرفتن بین سیگنال ارسالی و دریافتی استفاده میکنیم. که پیاده سازی آن در تابع p2func نوشته شده است.

```

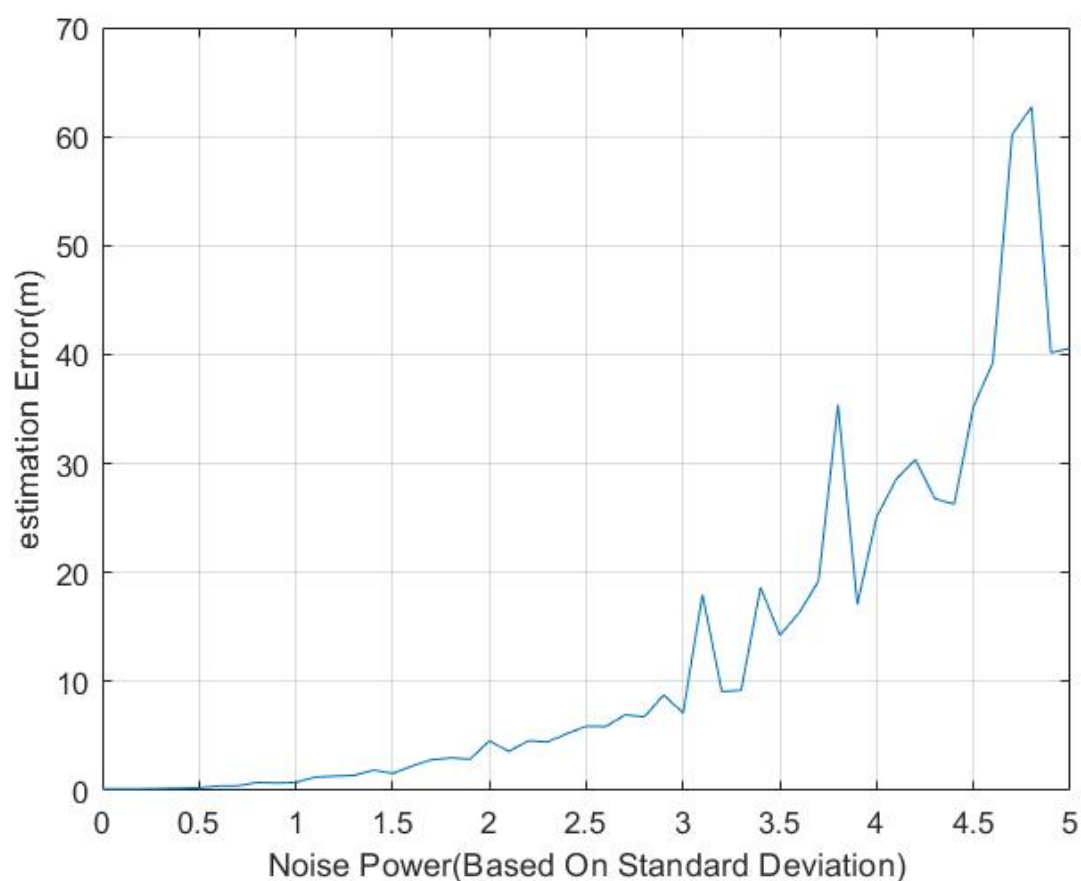
1  function [findR,corrVal]=p2func(sentSignal, recievedSignal)
2  -     ts=1e-9;
3  -     T=1e-5;
4  -     tau=1e-6;
5  -     t=0:ts:T;
6  -     c=3e8;
7
8  -     yConv = conv(sentSignal, recievedSignal);
9  -     [findMaxCorr,findMaxIndex]=max(yConv);
10 -     if(findMaxIndex - 1000 < 1)
11 -         findMaxIndex = 1;
12 -     else
13 -         findMaxIndex = findMaxIndex - 1000;
14 -     end
15 -     findTd=t(findMaxIndex);
16 -     findR=findTd*c/2;
17 - end

```

(3-2)

در این بخش، دقیقا مشابه پروژه ی اول، با پله هایی به گام 0.1، نویز را زیاد می کنیم و خطای فاصله سنجی را می یابیم، همانگونه که در شکل مشاهده میشود، آخرین جایی که فاصله یابی دقیقی داشته ایم(خطای کمتر از 10 متر)، در انحراف معیار 3 بوده است.(کد این بخش در p2.m نوشته شده است که هم قسمت 1 و هم قسمت 3 سوال 2 در این فایل است.)





## بخش سوم)

برای این قسمت، توضیح خاصی وجود ندارد و صرفاً باید یک mapSet تهیه کرد و آن را لود کرد. سپس از آنها در جای مورد نیاز استفاده کرد. که در customer\_calling.m نوشته شده است.

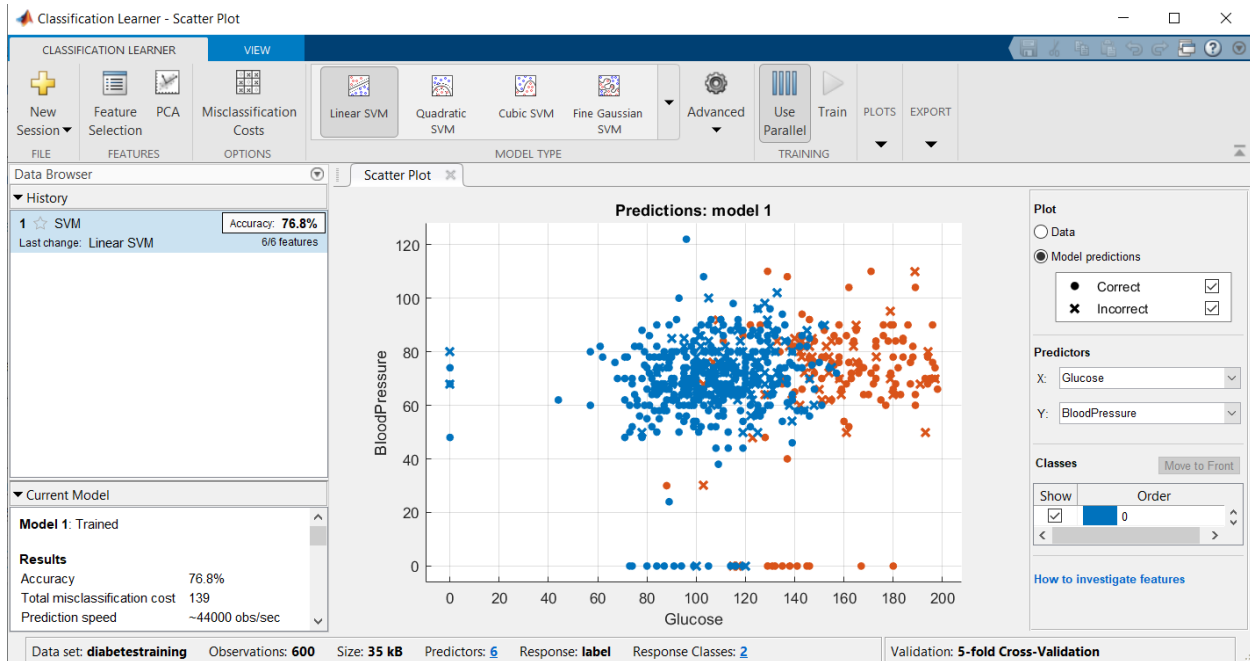
که ویس ها باید به ترتیب شماره ی، دهگان، اُ، یکان، به باجه ی، باجه باشد.

تنها نکته ای که باید به آن دقت کرد این است که زمانی که شماره ی مشتری، تک رقمی است، نباید دهگان و بخش اُ را خواند و فقط باید بخش یکان آن را خواند. به همین جهت باید اعداد دو رقمی و تک رقمی را جدا همدل کنیم. (تست آن در p3.m انجام شده است).

## بخش چهارم)

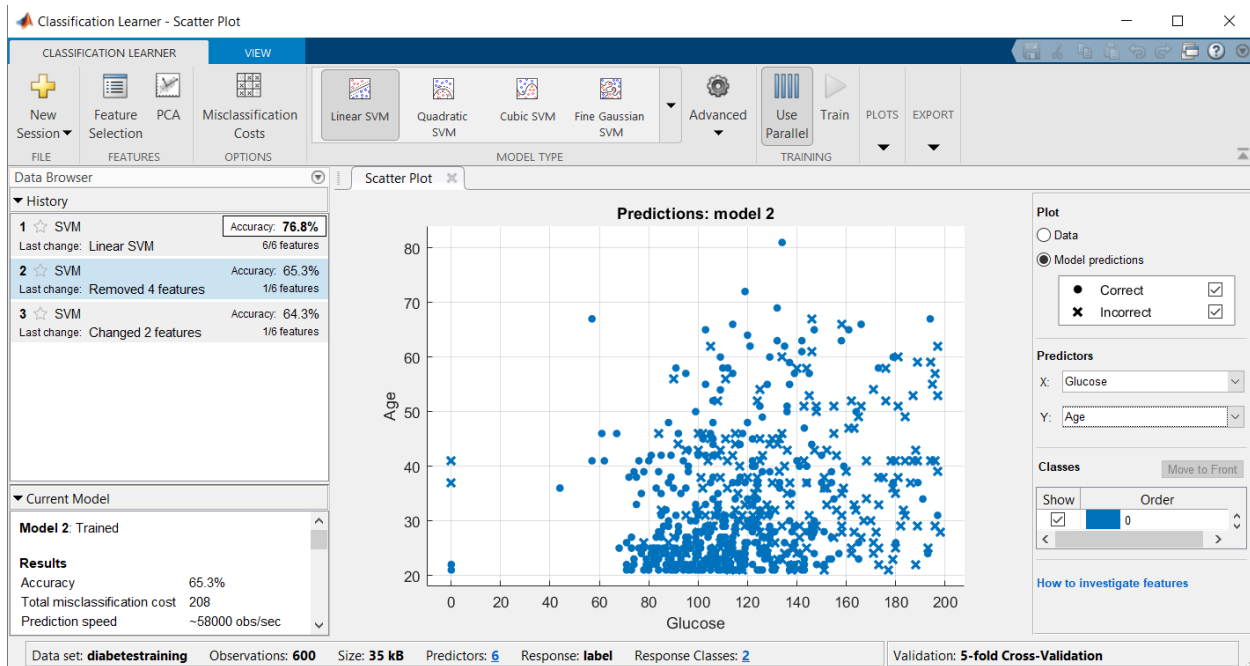
(1-4

بعد از لود کردن diabetes\_training در برنامه، و قرار دادن model type بر روی Linear SVM و TRAIN کردن آن، به نتیجه زیر رسیدیم. همانگونه که در عکس مشخص است، دقت گزارش شده برابر 76.8 درصد می باشد.

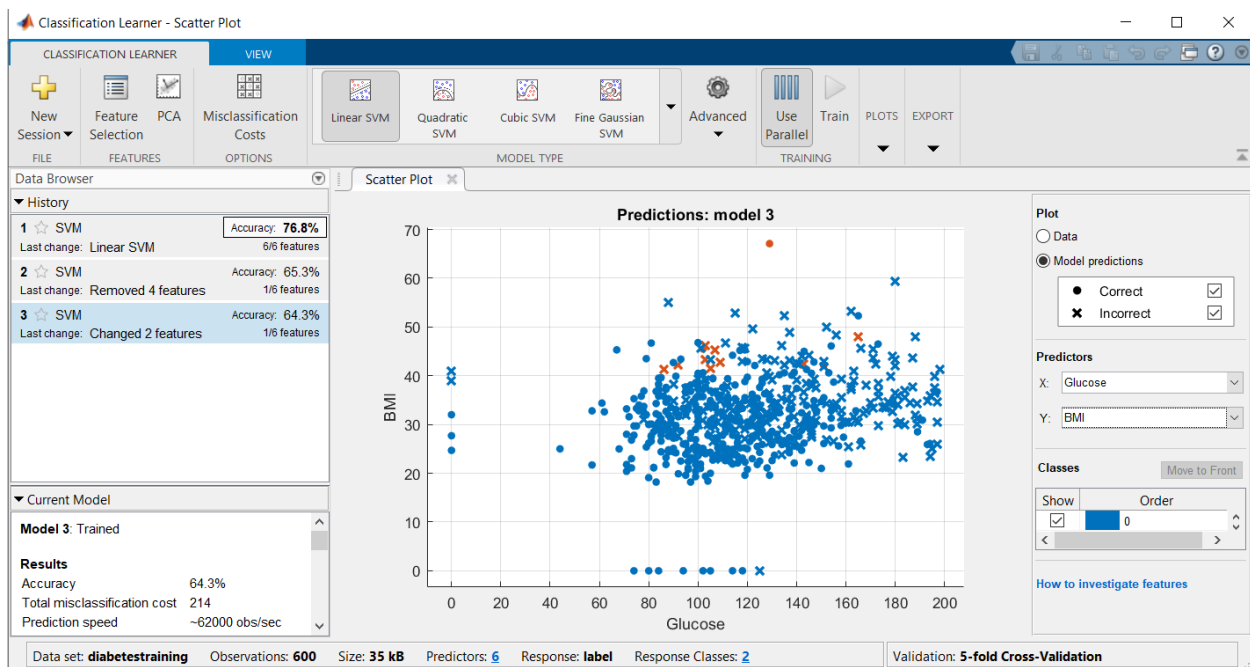


(2-4

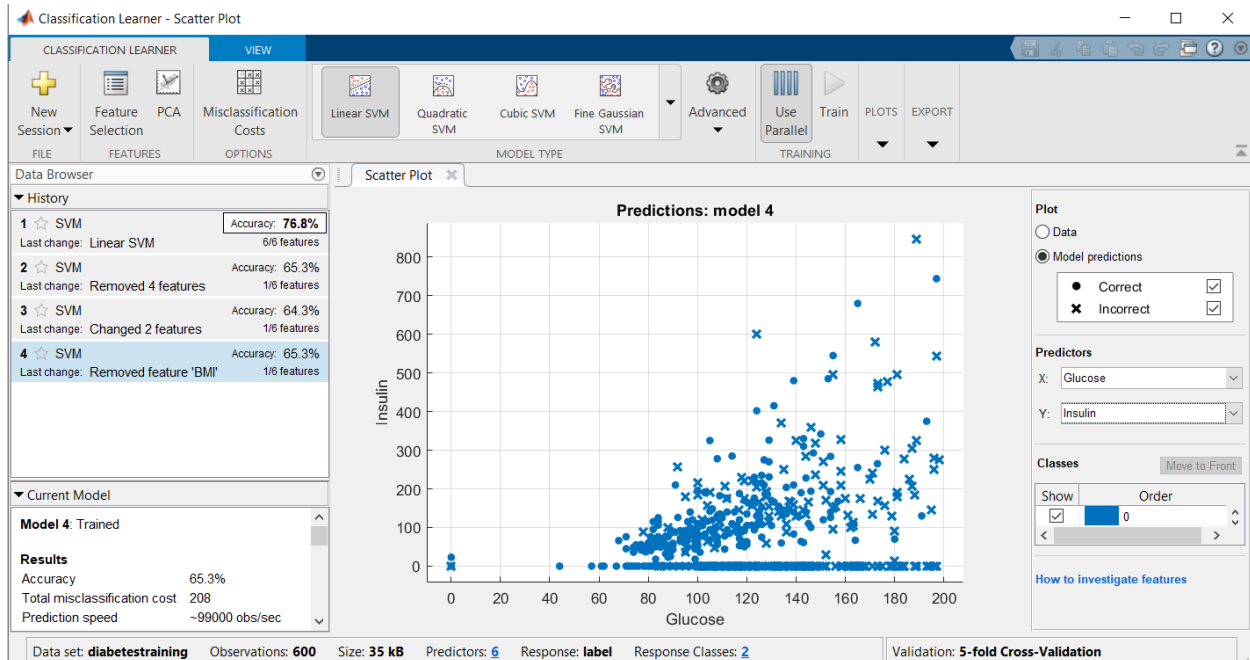
برای شاخص age، دقت به دست آمده برابر 65.3 درصد می باشد:



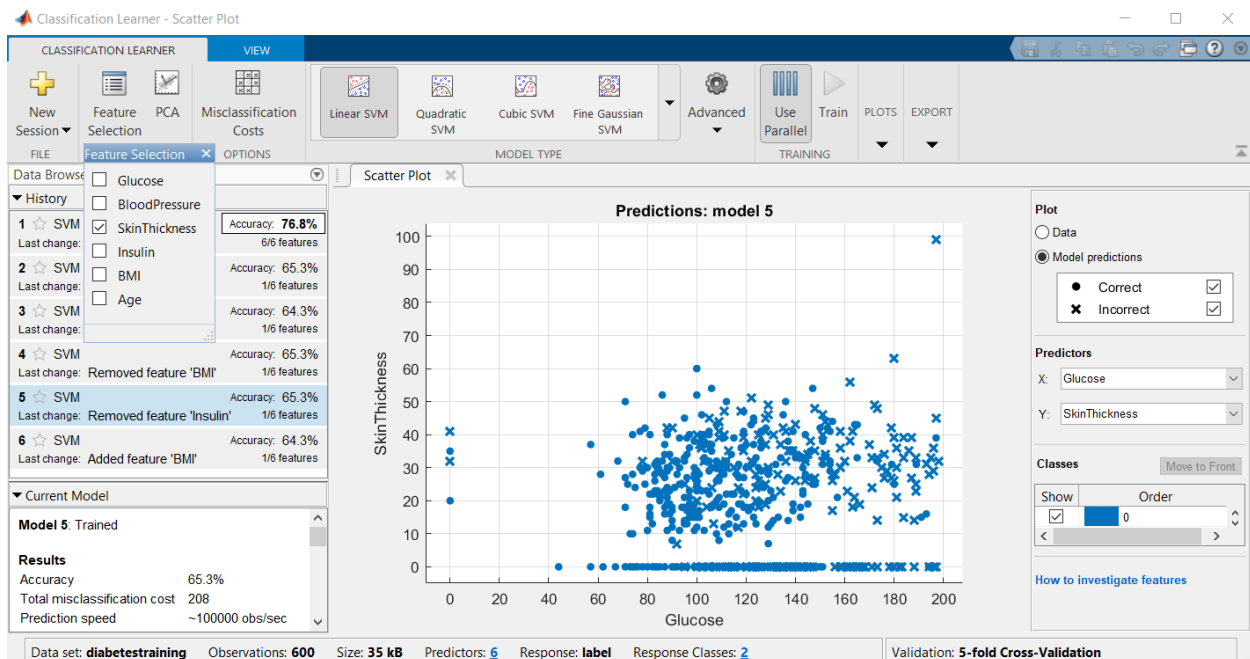
برای شاخص BMI، دقت به دست آمده برابر 64.3 درصد می باشد:



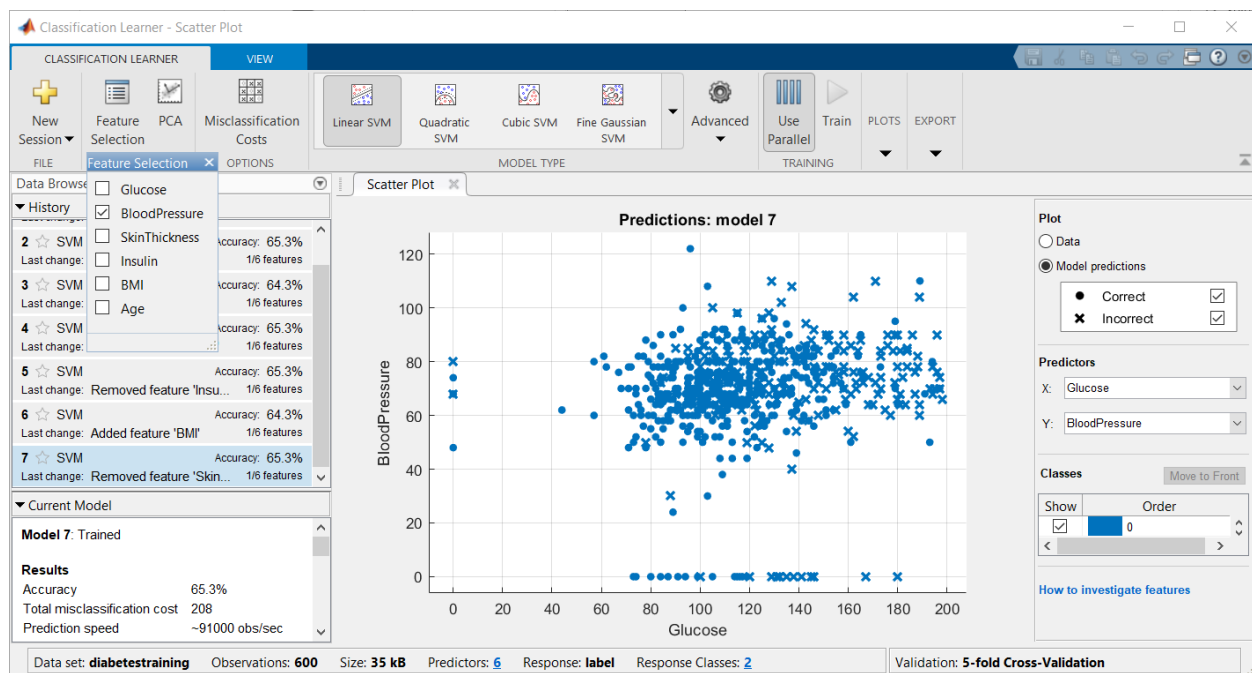
برای شاخص Insulin، دقت به دست آمده برابر 65.3 درصد می باشد:



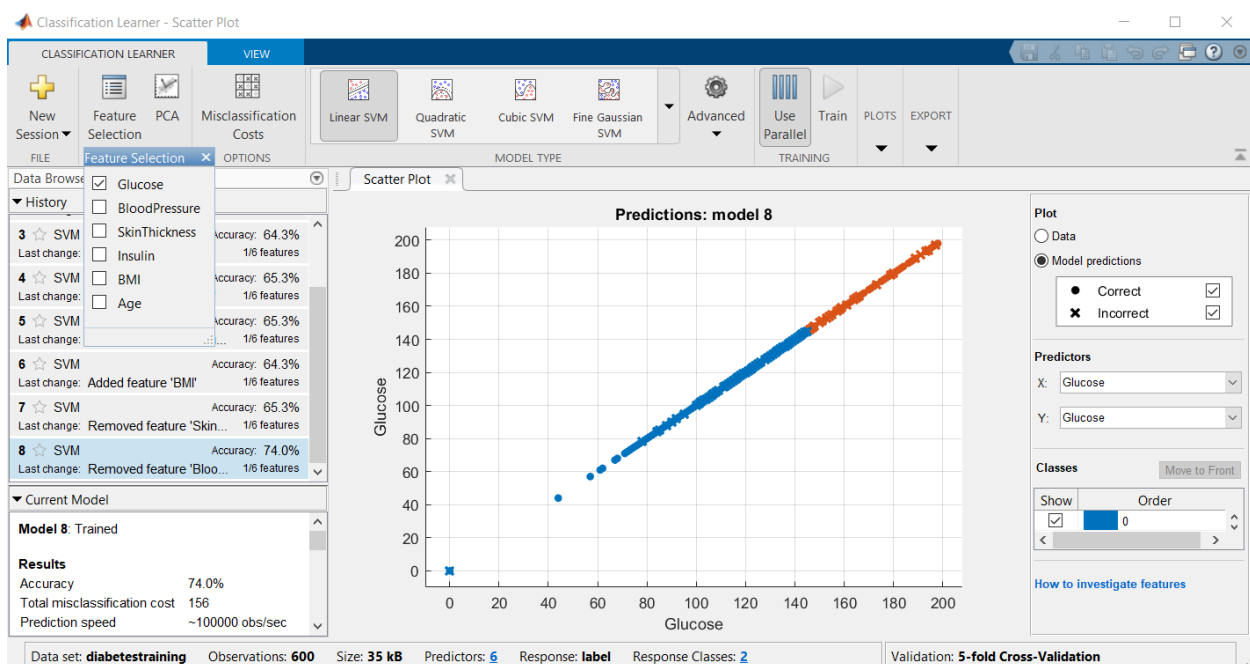
برای شاخص  $skinThickness$ ، (که در تصویر مدل شماره 5 است)، دقت به دست آمده برابر 65.3 درصد است:



برای شاخص bloodPressure، (مادل شماره 7) دقت به دست آمده برابر 65.3 درصد است:



برای شاخص Glucose، (مادل شماره 8) دقت به دست آمده برابر 74 درصد است:



با توجه به دقت های بدست آمده، به نظر میرسد شاخص Glucose بیشتر به دیابتی بودن فرد ارتباط دارد.

(3-4

ابتدا با استفاده از predictFcn، یک مجموعه دیتا به مدل آموزش دیده می‌دهیم و خروجی که predictedLabels است، نشان می‌دهد به ازای هر دیتا، خروجی این مدل بر اساس آموزشی که دیده، 0 است یا 1. حال با مقایسه آن با ستون آخر، متوجه میشویم که چند تا از آنها را درست حدس زده ایم.

با اجرای برنامه ی زیر، به خروجی 77.5 درصد میرسیم. که با نتیجه به دست آمده در قسمت اول، که 76.8 درصد بود، تفاوت کمی دارد و تقریباً برابر است.

```
Untitled.m x coding_amp.m x decoding_amp.m x p4_1.m x +
1 - predictedLabels = TrainedModel.predictFcn(diabetestraining);
2 - lastColoumn = table2array(diabetestraining(:, 7));
3
4 - sameNumber = 0;
5 - for i=1:size(lastColoumn, 1)
6 -     if(lastColoumn(i, 1) == predictedLabels(i, 1))
7 -         sameNumber = sameNumber + 1;
8 -     end
9 - end
10
11 - successPercentage = sameNumber / size(lastColoumn, 1);
12 - fprintf("success percentage is : %f %%", 100*successPercentage);
13 |
14

Command Window
>> p4_1
fx success percentage is : 77.500000 %>>
```

(4-4

همان کد قسمت قبل را ایندفعه برای dataSet جدید اجرا می کنیم. دقت به دست آمده برای دیتاست جدید، مطابق تصویر برابر 78 درصد می باشد:

```

14 - predictedLabels = TrainedModel.predictFcn(diabetesvalidation);
15 - lastColoumn = table2array(diabetesvalidation(:, 7));
16
17 - sameNumber = 0;
18 - for i=1:size(lastColoumn, 1)
19 -     if(lastColoumn(i, 1) == predictedLabels(i, 1))
20 -         sameNumber = sameNumber + 1;
21 -     end
22 - end
23
24 - successPercentage = sameNumber / size(lastColoumn, 1);
25 - fprintf("success percentage for the validation dataSet is : %f %%", 100*successPerce
< >

```

Command Window

```

>> p4_1
success percentage is : 77.500000 %
fx success percentage for the validation dataSet is : 78.000000 %>>

```