UCLA CS 260 - Project Report (Yao-Jen Chang, 704405423)

In this project, I have several python programs and a readme file, please see readme to know how the programs work. I used 4 learning methods including **KNN** (KNN.py), **single-layer perceptron** (Perceptron.py), **multi-layer perceptron in sequential training** (Perceptron.py), and **multi-layer perceptron in batch training** (MLPbatch.py). For the MLP in batch training, I wrote my own MLP based on book (use numpy). Otherwise, I implemented all the program and implemented them by myself.

I draw every point and mean value to observe data. (Fig.1 and Fig.2 in Appendix) From the graphs, data may could be divided into several groups, but they aren't linear separable. So KNN may work better than single-layer perceptron. Each user have 2 time series data points (26 points). To extract features, I applied statistical methods and combine different features. I describes my uniques features in **Table 1**.

Statical Method for Extracting Features		Detailed Description for 2 Time Series Data	# of Features
1	none	Don't do anything.	52
2	weight1	I give more weight to the most recent terms in the time series and less weight to older data. I have time series data with n points. (In our data set, n= 26) There's a variable $sum = \sum_{i=1}^{n} i$, $weights_i = x_i * i / sum$ i for 1 to n and x as time series data.	2
3	mean	Mean of 2 time series data	2
4	addMean	Sum of 2 means time series data. (Sum of **method 3**.)	1
5	miusMean	Absolute value of subtraction of 2 means time series data. (Absolute value of subtraction of **method 3**.)	1
6	addMean_miusMean	method "addMean" and method "miusMean"	2
7	rms	Root mean square of time series data	2
8	std	Standard deviation of time series data	2
9	median	Median of time series data	2
10	mean_median	Mean and median of time series data (**Method 3 and method 9**)	4
11	mean_std	Mean and standard deviation of data (**Method 3 and method 8**)	4
12	mean_rms	Mean and root mean square of data (**Method 3 and method 7**)	4

Table 1. Feature explanation in "calStatic" function (ML_project.py)

K value	Used Features	feature. #	precission	recall	sensitivity	specificity	F-measure	KNN Accuracy
5	"mean"	2	0.579	0.688	0.688	0.652	0.629	**66.67%**
5	"weight1"	2	0.579	0.647	0.647	0.636	0.611	64.1%
5	"addMean"	1	0.632	0.667	0.667	0.667	0.649	**66.67%**
5	"addMean_miusMean"	2	0.579	0.688	0.688	0.652	0.629	**66.67%**
5	"rms"	2	0.579	0.688	0.688	0.652	0.629	**66.67%**
21	"median"	2	0.737	0.609	0.609	0.688	0.667	64.1%
7	"mean_median"	4	0.632	0.667	0.667	0.667	0.649	**66.67%**
5	"mean_rms"	4	0.579	0.688	0.688	0.652	0.629	**66.67%**

Table 2. KNN for first dataset with LOOCV

1. Experiments for first dataset (Leave-one-out cross-validation, LOOCV)

I used LOOCV to evaluate my 4 prediction models and unique features in Table 1. I have listed well-defined parameters and features in different following Tables. In each Table, you can see precision, recall, sensitivity, specificity, F-measure and accuracy. Except for the ROC curves, all other mentioned graphs are in Appendix section. Additionally, you can see more analysis graphs in "First Data Img" folder and "Second Data Img" folder. You can easily figure it out the graph meaning from every files' name.

I test **KNN** with odd K value (like 1, 3, 5… 37) **for every feature set**. After finding the K nearest neighbors, I count the majority class of these neighbors as predicted class. I recorded some of the best results in **Table. 2**. We can observe the accuracy trend from fig. 3 and fig 4. Fig. 5 and fig.6 show ROC curves when I tested different K values. Also, because the position of every data are fixed, accuracy are always same if I give same k value and same feature sets. The final accuracy can be high as 66.67%!

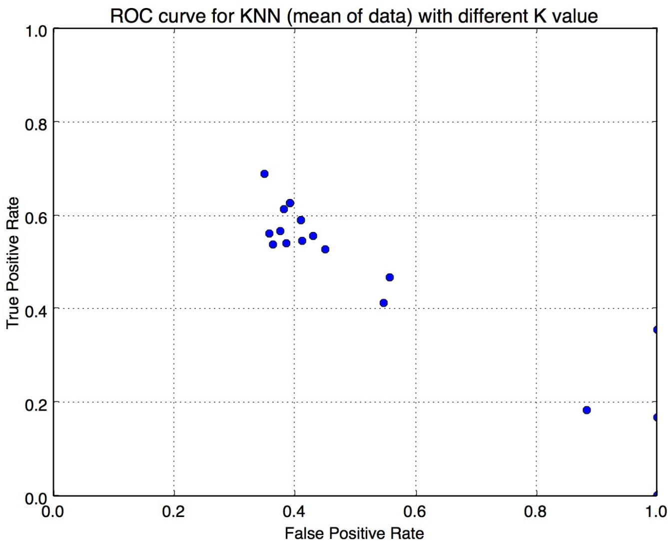


Fig. 5 - ROC curve of KNN for mean

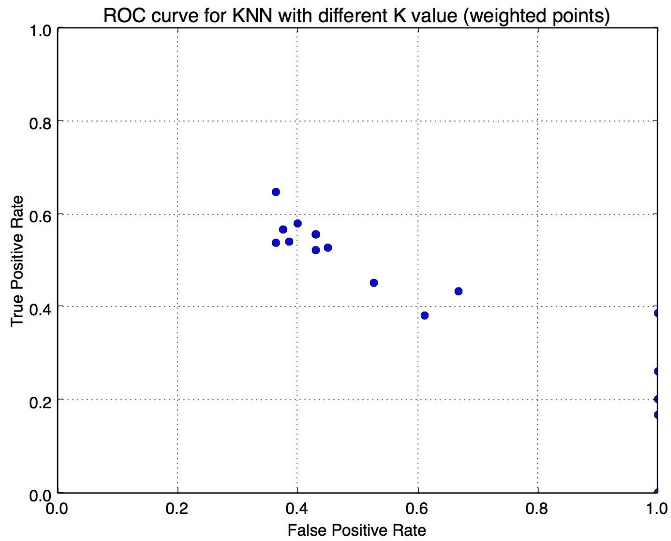
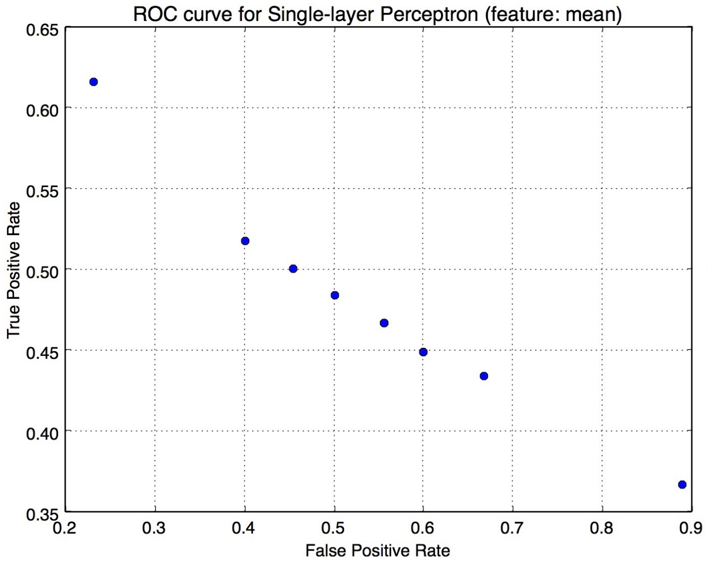


Fig. 6 - ROC curve of KNN for weighted points

Then I tested **single-layer perceptron** with or without **bias node**. Additionally, the data isn't linear separable so I tried to **apply one kernel function** to see if data will become dividable. I derive 2 dimension features (x_1, x_2) into 6 dimension $(1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2)$. Also, I **normalized** the dataset into 0 to 1. I tested some uniques features with different parameters and recorded some better results in **Table. 3**. However, even though I used normalization and kernel function, the weights still don't converge. So the results will changes since every run has random initial weights. Sometimes, it might have high accuracy like 66.67%, but it happened rarely. I showed 2 ROC curves in Fig. 7 and Fig. 8.

bias	kernel fun.	thres hold	Used Features	precission	recall	sensitivity	specificity	F-measure	Accuracy
N	N	0.1	"mean"	0.579	0.647	0.647	0.636	0.611	**64.1%**
N	Y	0.1	"mean"	0.895	0.548	0.548	0.750	0.680	58.97%
Y	N	0.5	"mean"	1.000	0.487	0.487	0.000	0.655	48.71%
Y	Y	0.3	"mean"	0.947	0.474	0.474	0.000	0.632	46.15%
N	N	0.2	"weight1"	0.263	0.625	0.625	0.548	0.370	56.41%
N	Y	0.1	"weight1"	0.789	0.577	0.577	0.692	0.667	**61.53%**
N	Y	0.3	"rms"	0.579	0.524	0.524	0.556	0.550	53.84%
N	Y	0.2	"mean_median"	0.737	0.519	0.519	0.583	0.609	53.84%
N	Y	0.6	"mean_std"	0.947	0.486	0.486	0.500	0.643	48.71%
N	Y	0.2	"mean_rms"	0.526	0.714	0.714	0.640	0.606	**66.67%**

Table 3. Perceptron for first dataset with LOOCV (1000 training iteration, 0.25 learning rate)



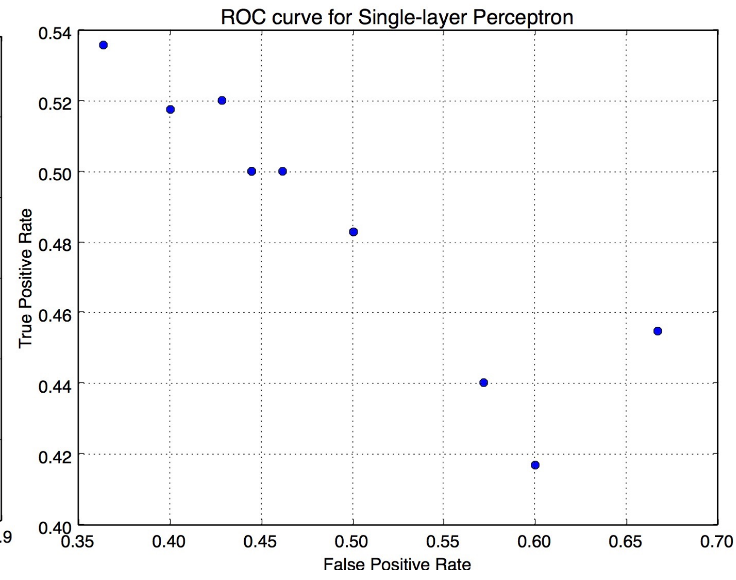


Fig. 7 - ROC (perceptron, no bias, kernel, mean)

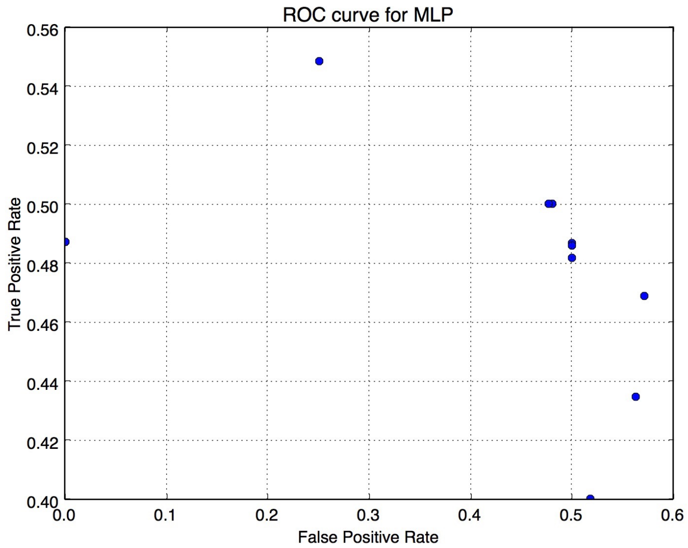
Fig. 8 - ROC (perceptron, no bias, kernel, weighted)

hidden layer nodes	types of output neurons	# of iteration	thre-shold	Used feature	precission	recall	sensitivity	specificity	F-measure	accuracy
2	'logistic'	1000	0.6	“mean"	0.368	0.636	0.636	0.571	0.467	58.97%
3	'logistic'	1000	0.5	“mean"	0.526	0.625	0.625	0.609	0.571	**61.53%**
2	'logistic'	10000	0.5	“mean"	0.526	0.526	0.526	0.550	0.526	53.84%
3	'logistic'	10000	0.5	“mean"	0.579	0.647	0.647	0.636	0.611	**64.1%**
4	'logistic'	10000	0.5	“mean"	0.474	0.600	0.600	0.583	0.529	58.97%
3	'logistic'	10000	0.4	“mean"	0.789	0.652	0.652	0.750	0.714	**69.23%**
3	'logistic'	10000	0.5	“mean"	0.632	0.706	0.706	0.682	0.667	**69.23%**

Table 4. MLP(sequential) for first data with LOOCV (0.25 learning rate, has Bias node, no kernel)

hidden layer nodes	types of output neurons	# of iteration	thre-shold	Used feature	precission	recall	sensitivity	specificity	F-measure	accuracy
2	“linear”	1000	0.5	“mean"	0.789	0.556	0.556	0.667	0.652	58.97%
3	“linear”	1000	0.5	“mean"	0.842	0.533	0.533	0.667	0.653	56.41%
4	“linear”	1000	0.5	“mean"	0.789	0.517	0.517	0.600	0.625	53.84%
2	'logistic'	1000	0.5	“mean"	0.789	0.556	0.556	0.667	0.652	58.97%
3	'logistic'	1000	0.5	“mean"	0.579	0.611	0.611	0.619	0.595	**61.53%**
4	'logistic'	1000	0.5	“mean"	0.526	0.588	0.588	0.591	0.556	58.97%
2	“linear”	10000	0.5	“mean"	0.842	0.500	0.500	0.571	0.627	51.28%
3	“linear”	10000	0.5	“mean"	0.895	0.567	0.567	0.778	0.694	**61.53%**
2	'logistic'	10000	0.5	“mean"	0.474	0.692	0.692	0.615	0.562	**64.1%**
3	'logistic'	10000	0.5	“mean"	0.474	0.692	0.692	0.615	0.562	**64.1%**
4	'logistic'	10000	0.5	“mean"	0.632	0.600	0.600	0.632	0.615	**61.53%**

Table 5. MLP(batch) for first data with LOOCV (0.3 learning rate)



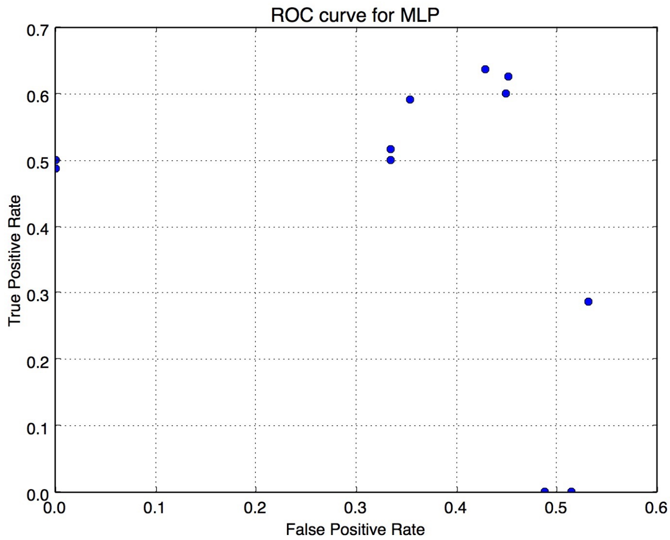


Fig. 9 - MLP(batch) ROC (2 hidden node, “linear”)   Fig. 10 - MLP(batch) ROC (2 hidden node, “logistic”)

Later, I tested **multi-layer perceptron(MLP)** for this project. From the book , there’re sequential and batch training for MLP. I implemented MLP(sequential) by myself and I use the book’s sample code as referenced to complete MLP(batch). MLP(sequential) updated and computes weights after every input. MLP(batch) used matrix method for inputs and weights (used NumPy), which is more efficient.

Both have two types of output neurons, linear and logistic. I stop the MLP training process if the error suddenly raise too much or the error stop changing. For both methods, their results are varied during different run with same parameter like single-layer perceptron. Also, due to other features’ similar performance, I decided to keep using “mean” feature. I used 0.25 learning rate for MLP(sequential) and 0.3 for MLP(batch) because 0.3 is too much for MLP(sequential), which made it worse.

From Table.4 and Table.5, I tested MLP with different number of hidden layer nodes and different number of training iteration. Sometimes, I have higher accuracy than single-layer perceptron, and I can get as high as 69.23% in MLP(sequential) and 64.1% in MLP(batch). Finally, I show ROC curves in Fig.9 and Fig.10 of MLP(batch) when I have different thresholds (for activation function).

2. Experiments to test first dataset through LOOCV

K value	Used Features	feature. #	precission	recall	sensitivity	specificity	F-measure	KNN Accuracy
7	“none"	52	0.700	0.583	0.583	0.600	0.636	**58.97%**
5	“mean"	2	0.650	0.520	0.520	0.500	0.578	51.28%
11	“weight1”	2	0.800	0.552	0.552	0.600	0.653	56.41%
7	"addMean"	1	0.700	0.583	0.583	0.600	0.636	**58.97%**
21	“miusMean"	1	0.800	0.552	0.552	0.600	0.653	56.41%
7	"addMean_miusMean"	2	0.650	0.520	0.520	0.500	0.578	51.28%
7	“rms"	2	0.650	0.520	0.520	0.500	0.578	51.28%
3	“std”	2	0.750	0.600	0.600	0.643	0.667	**61.53%**
7	“median”	2	0.750	0.556	0.556	0.583	0.638	56.41%
7	“mean_median”	4	0.700	0.538	0.538	0.538	0.609	53.84%
11	“mean_std”	4	0.700	0.538	0.538	0.538	0.609	53.84%
7	“mean_rms”	4	0.650	0.520	0.520	0.500	0.578	51.28%

Table 6. KNN for second dataset with LOOCV

bias	kernel fun.	threshold	Used Features	precission	recall	sensitivity	specificity	F-measure	Accuracy
N	Y	0.3	“mean"	0.500	0.500	0.500	0.474	0.500	48.71%
Y	Y	0.2	“weight1”	0.750	0.536	0.536	0.545	0.625	53.84%
N	Y	0.3	“rms"	0.500	0.588	0.588	0.545	0.541	**56.41%**
N	Y	0.2	“mean_median”	0.650	0.591	0.591	0.588	0.619	**58.97%**
N	Y	0	“mean_std”	0.750	0.500	0.500	0.444	0.600	48.71%
N	Y	0.5	“mean_rms”	0.300	0.545	0.545	0.500	0.387	51.28%

Table 7. Perceptron for second dataset with LOOCV (1000 training iteration, 0.25 learning rate)

hidden layer node	types of output neurons	# of iteration	thre-shold	Used feature	precission	recall	sensitivity	specificity	F-measure	accuracy
2	'logistic'	1000	0.5	“mean"	0.650	0.591	0.591	0.588	0.619	**58.97%**
3	'logistic'	1000	0.5	“mean"	0.650	0.520	0.520	0.500	0.578	51.28%
2	'logistic'	10000	0.5	“mean"	0.600	0.462	0.462	0.385	0.522	43.58%
3	'logistic'	10000	0.5	“mean"	0.600	0.429	0.429	0.273	0.500	38.46%

Table 8. MLP(sequential) for first data with LOOCV (0.25 learning rate, has Bias node, no kernel)

hidden layer nodes	types of output neurons	# of iteration	thre-shold	Used feature	precission	recall	sensitivity	specificity	F-measure	accuracy
2	“linear”	1000	0.5	“mean"	0.850	0.486	0.486	0.250	0.618	46.15%
3	“linear”	1000	0.5	“mean"	0.950	0.543	0.543	0.750	0.691	56.41%
4	“linear”	1000	0.5	“mean"	0.900	0.529	0.529	0.600	0.667	53.84%
2	“linear”	10000	0.5	“mean"	0.900	0.562	0.562	0.714	0.692	**58.97%**
3	“linear”	10000	0.5	“mean"	0.750	0.469	0.469	0.286	0.577	43.58%

Table 9. MLP(batch) for first data with LOOCV (0.3 learning rate)

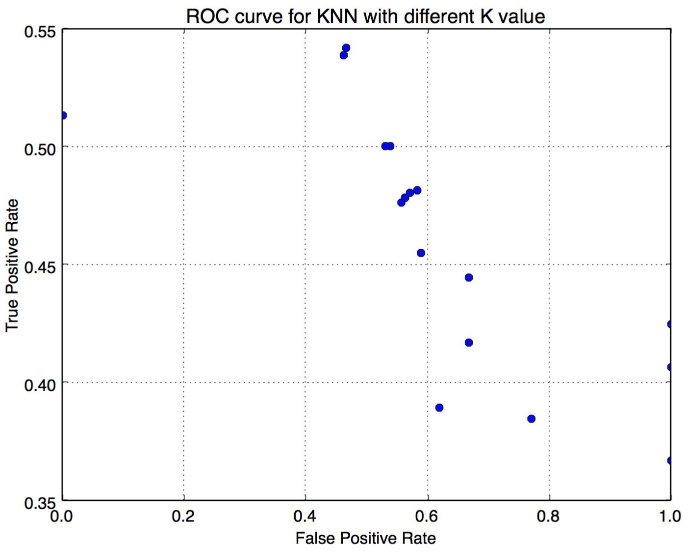


Fig.11 - ROC curve of KNN for “mean_std”

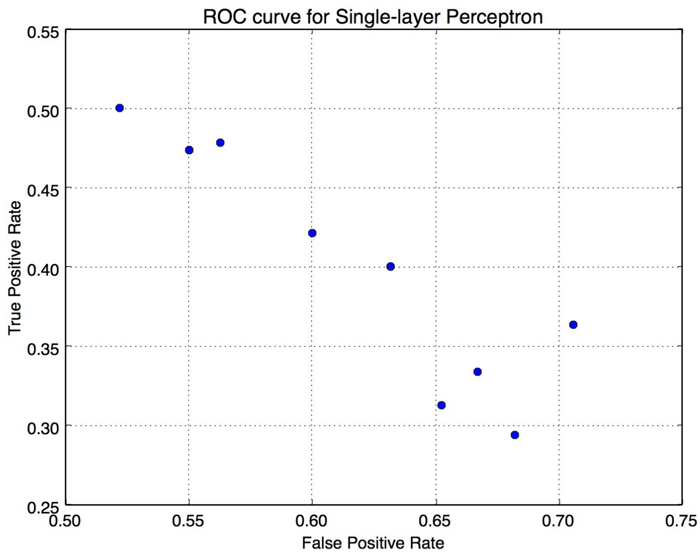
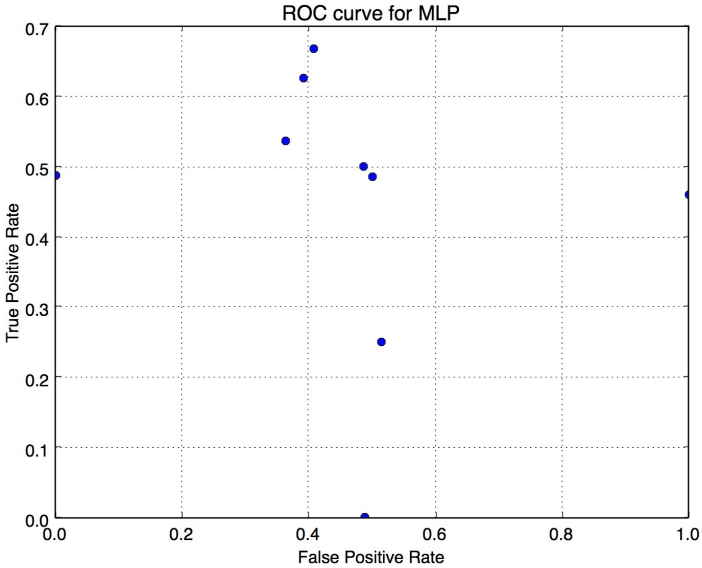


Fig.12-ROC(perceptron,no bias,has kernel,weight1)(2nd data)



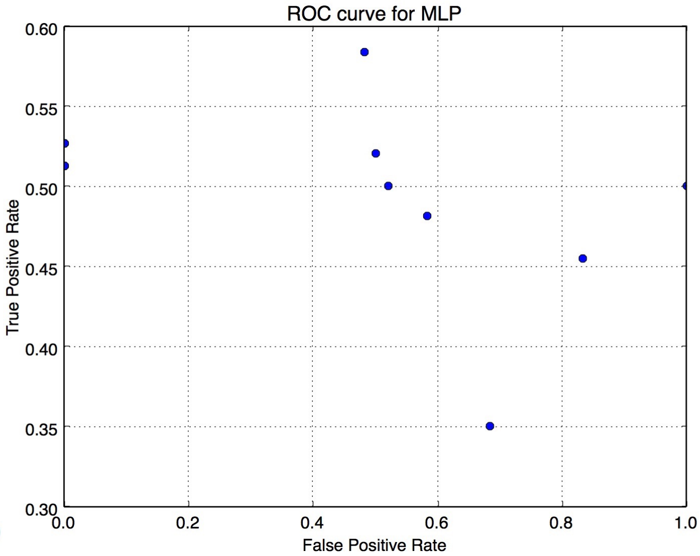


Fig.13-MLP(sequential) ROC(2 hidden, “logistic”) Fig.14-MLP(batch),ROC (3 hidden, “linear”) (2nd data)

For the new second dataset, I do the same analysis as first dataset. I ran exact 4 learning methods like section 1. It’s good to test different parameter like K values, thresholds or number of hidden nodes, because the second dataset has different distribution compared first dataset. Therefore, it’s predictable that the best parameters and methods for first dataset may not fit second dataset.

Table 6 shows the KNN method, and I can get **61.53%** accuracy with standard deviation feature. Tables 7, 8, and 9 shows perceptron, MLP(sequential) and MLP(batch). I have **58.97%** accuracy from these methods, which is not as high as they do in the first dataset. Due to the limited time, I didn’t try more feature set, more hidden-layer nodes, and more training iteration. It might be better with different parameters.

3. Conclusion and Future improvement

I think it’s good to do this project and I can get around as high as 66% for first dataset and 60% for the second dataset. There are still more way worth to try. In addition to trying more feature set and different parameter for perceptron and MLP, I may change the label 0 and 1 into -1 and 1 for perceptron and MLP, which may can give more different results. Besides, I can try to blending some of these models to mix and then get new prediction results.

4. Uniqueness

Here are my uniqueness for this project:

1. **12 different feature** sets described in Table. 1
2. Applied **normalization** to input features into 0 to 1 for perceptron and MLP
3. Applied **Bias node** to single-layer perceptron
4. Used **kernel function** for perceptron to try make data linear separable:
   - make 2 dimension features (x_1, x_2) into 6 dimension . $(1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2)$
5. Implemented **multi-layer perceptron** (MLP)
   - **sequential training version** and **batch training version**
   - ‘logistic’ and ‘linear’ error calculation for both version

5. Program and Project Output

You see the README.txt file and comments in programs to see how my program work. You run the “ML_project.py” and simply get many sample experiments for first dataset and second dataset. If you want to run the new dataset, you just change the folder’s name in “ML_project.py” at line 504.

After you run “ML_project.py” (may take around 8 to 9 minutes), it will generate 2 output files (already in submission folder). 'Output_first_dataset.txt' is for first dataset and 'Output_second_dataset.txt' is for the second dataset. In these 2 files, you can see the sample experiments’ results including **accuracy,**

precision, recall, sensitivity, specificity, F-measure, and confusion matrices (TP, FP, TN, FN). Here are some sample results in the output files. You can see the whole experiments results in these 2 files.

```
Exp 1.  5-NN LOOCV (feature: means of data):  Accuracy: 0.666666666667

Precission:     recall:         sensitivity:    specificity:
0.5789          0.6875          0.6875          0.6522

F - measure: 0.628571428571

              Output
          class 1, class 0
 class 1     11         8
 class 0     5         15
```

Confusion matrices

```
-----------------------------------------------------------------------------------

Exp 2.  5-NN LOOCV (feature: addMean):  Accuracy: 0.666666666667

Precission:     recall:         sensitivity:    specificity:
0.6316          0.6667          0.6667          0.6667

F - measure: 0.648648648649

              Output
          class 1, class 0
 class 1     12         7
 class 0     6         14
-----------------------------------------------------------------------------------

Exp 13.  MLP LOOCV with 3 hidden layer nodes (feature: "mean") has Bias, no Kernel:  Accuracy: 0.58974

Precission:     recall:         sensitivity:    specificity:
0.4737          0.6000          0.6000          0.5833

F - measure: 0.529411764706

              Output
          class 1, class 0
 class 1     9          10
 class 0     6          14
-----------------------------------------------------------------------------------

Exp 14.  MLP(batch) (linear) LOOCV with 2 hidden layer nodes (feature: "mean"):  Accuracy: 0.512820512

Precission:     recall:         sensitivity:    specificity:
0.7368          0.5000          0.5000          0.5455

F - measure: 0.595744680851

              Output
          class 1, class 0
 class 1     14         5
 class 0     14         6
-----------------------------------------------------------------------------------
```

The ROC curves are showed in previous paragraphs and some sample graphs are in flowing Appendix section. Additionally, I strongly recommend that you can see more analysis graphs (like distribution of data, ROC curve and accuracy) in “**First Data Img**” folder and “**Second Data Img**” folder. You can easily figure it out the graph meaning from every files’ name.

Appendix

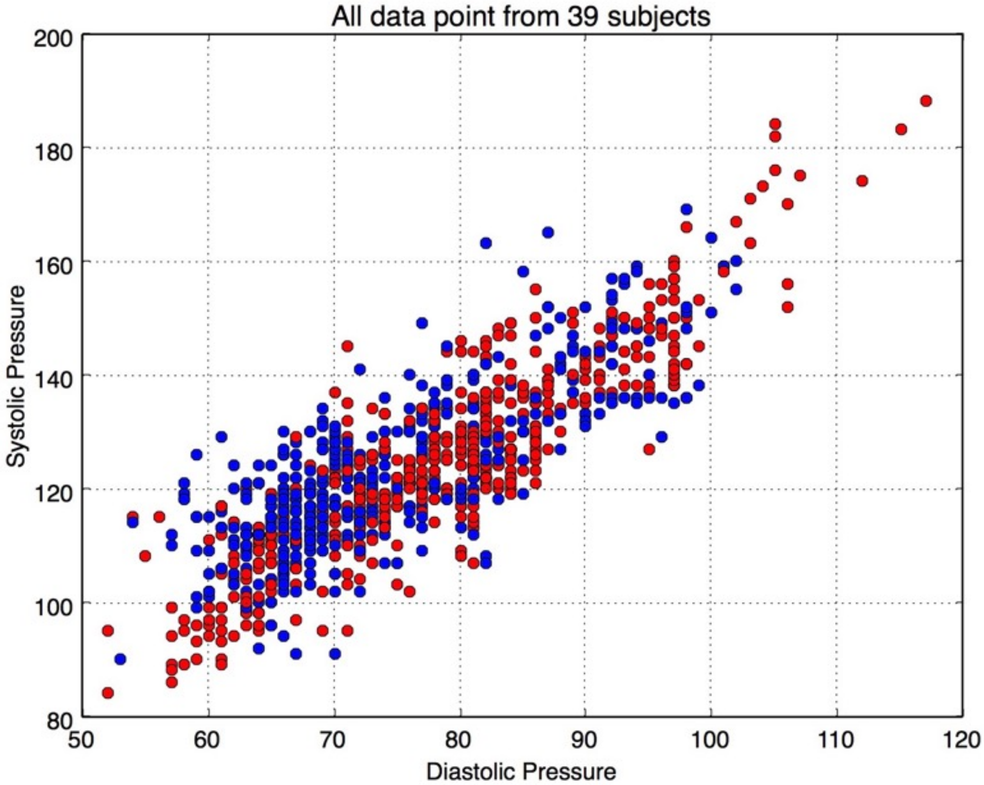


Fig. 1 - every data point of each subject (first dataset)

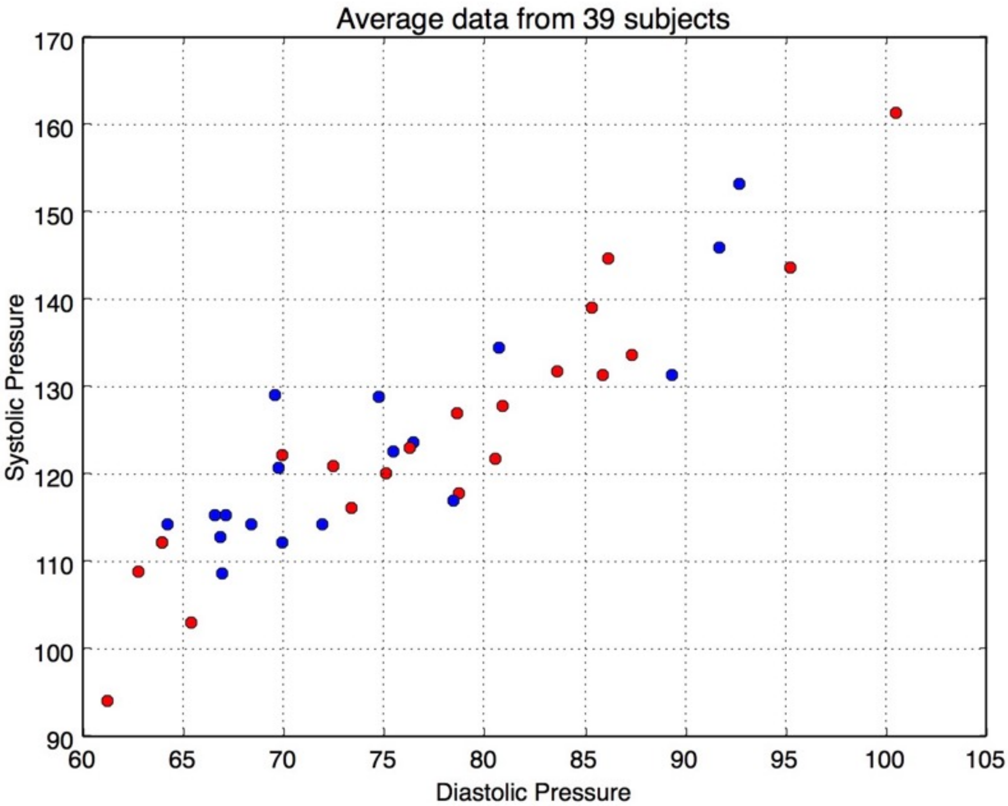


Fig. 2 - mean data point of each subject (first dataset)

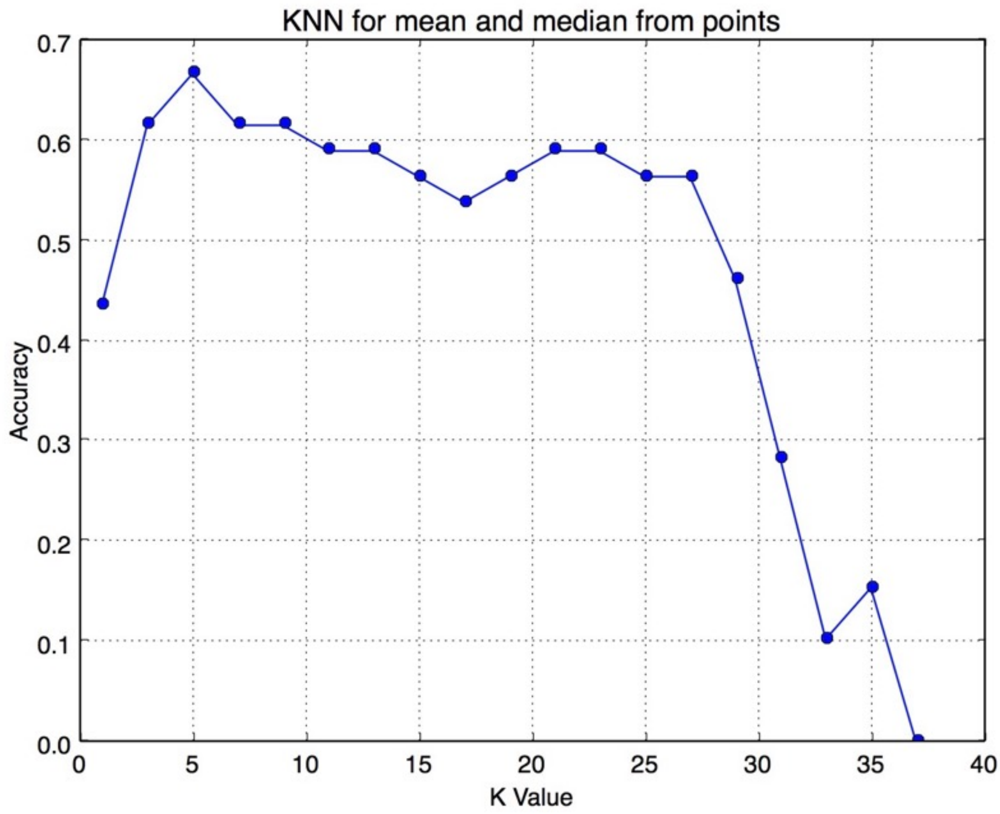


Fig. 3 - KNN for mean data of each subject (first dataset)

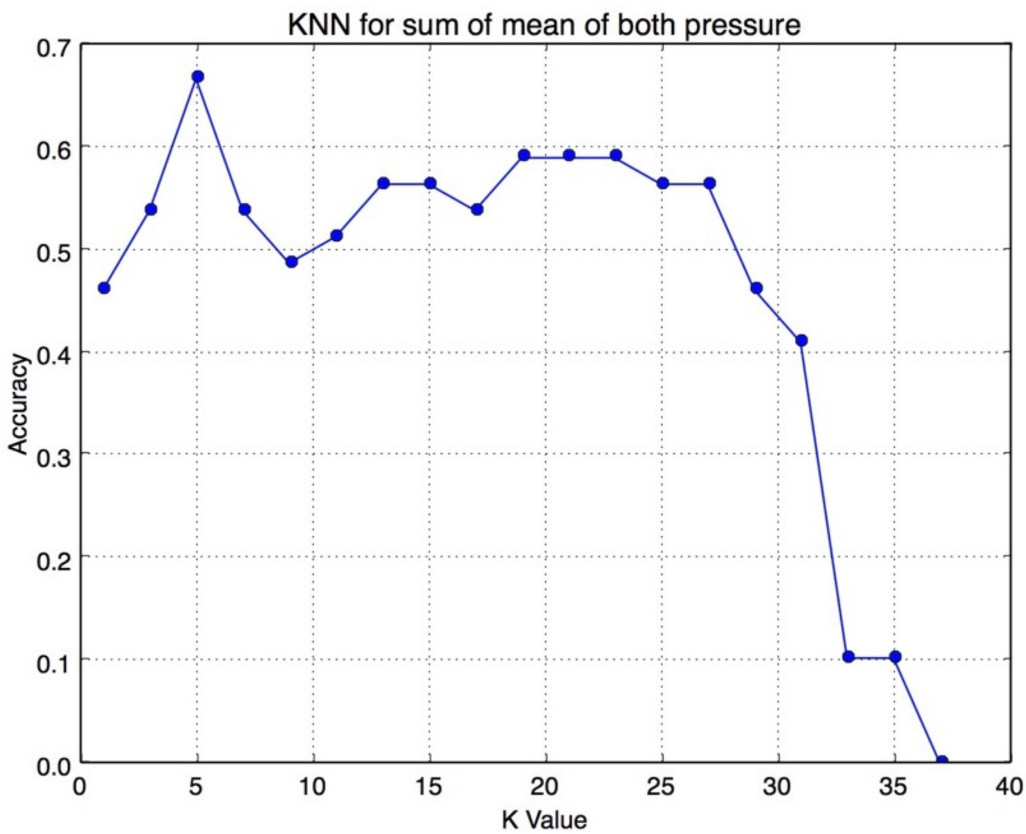


Fig. 4 - KNN for sum of mean data (first dataset)

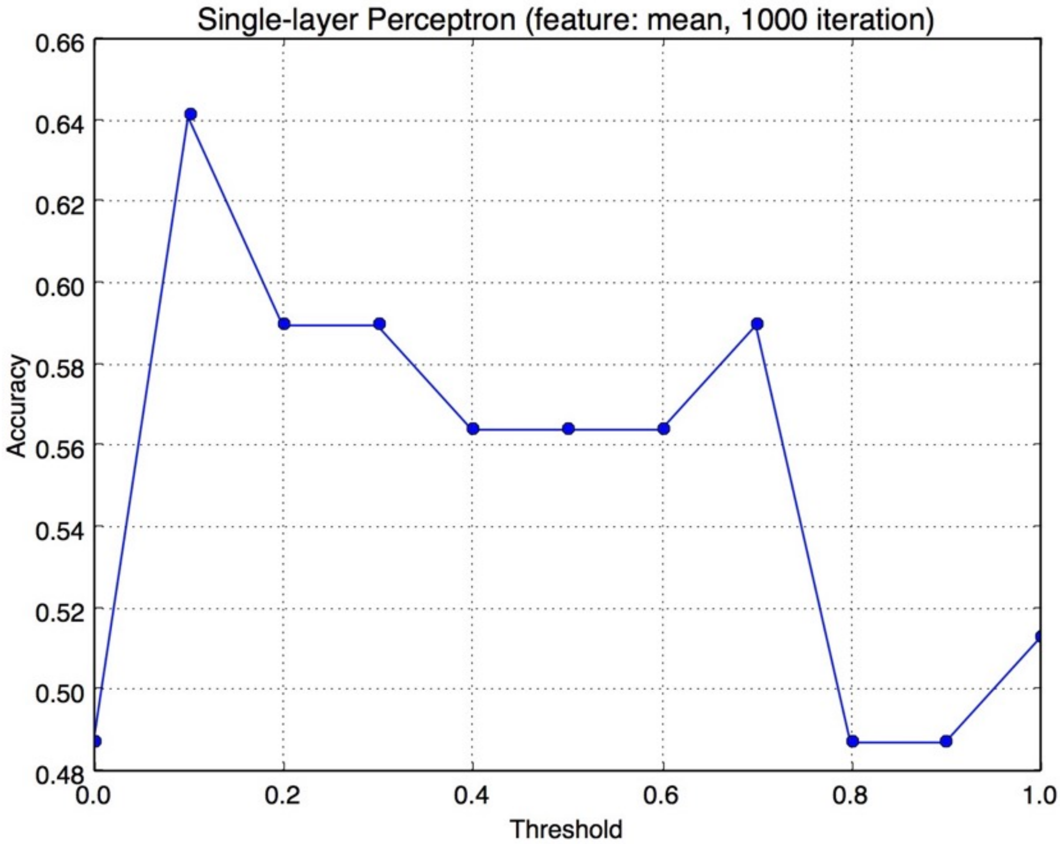


Fig.5 - Perceptron for mean of data for threshold (no bias, no kernel) (first dataset)

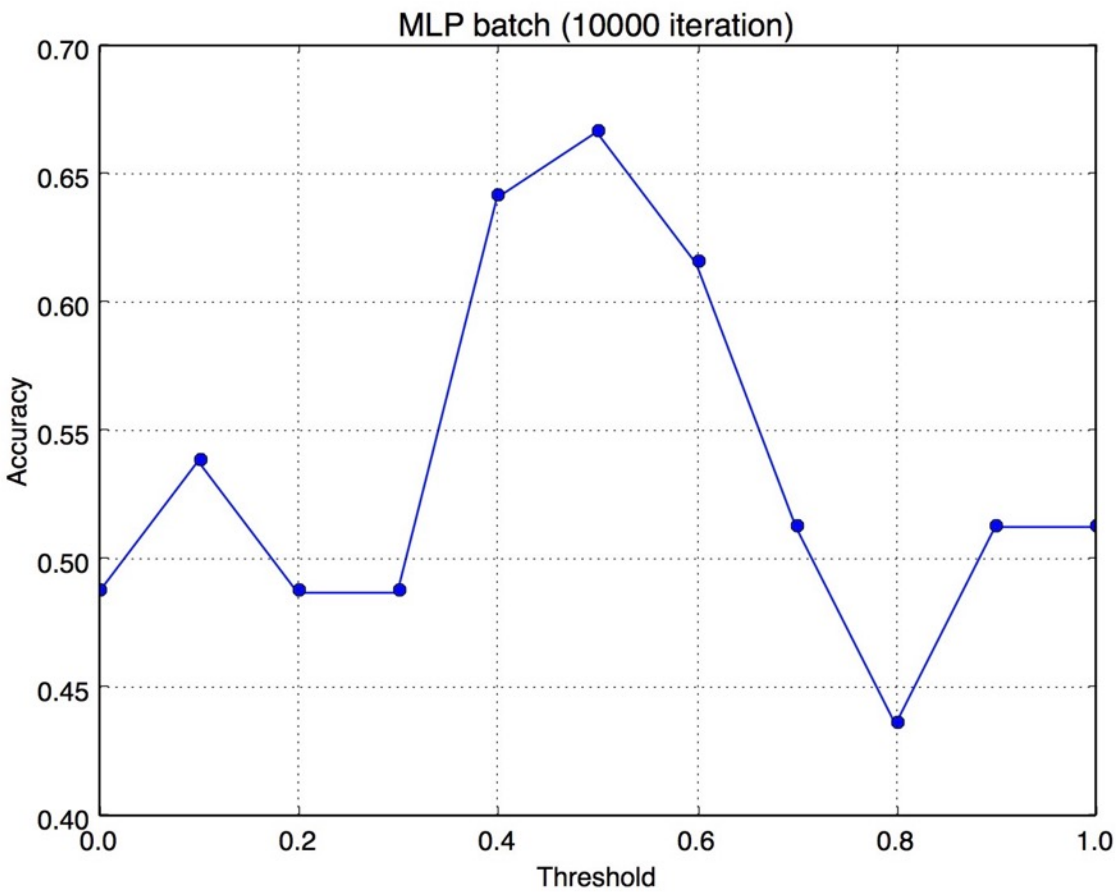


Fig.6 - MLP (batch) for 2 hidden layer node (‘logistic’)

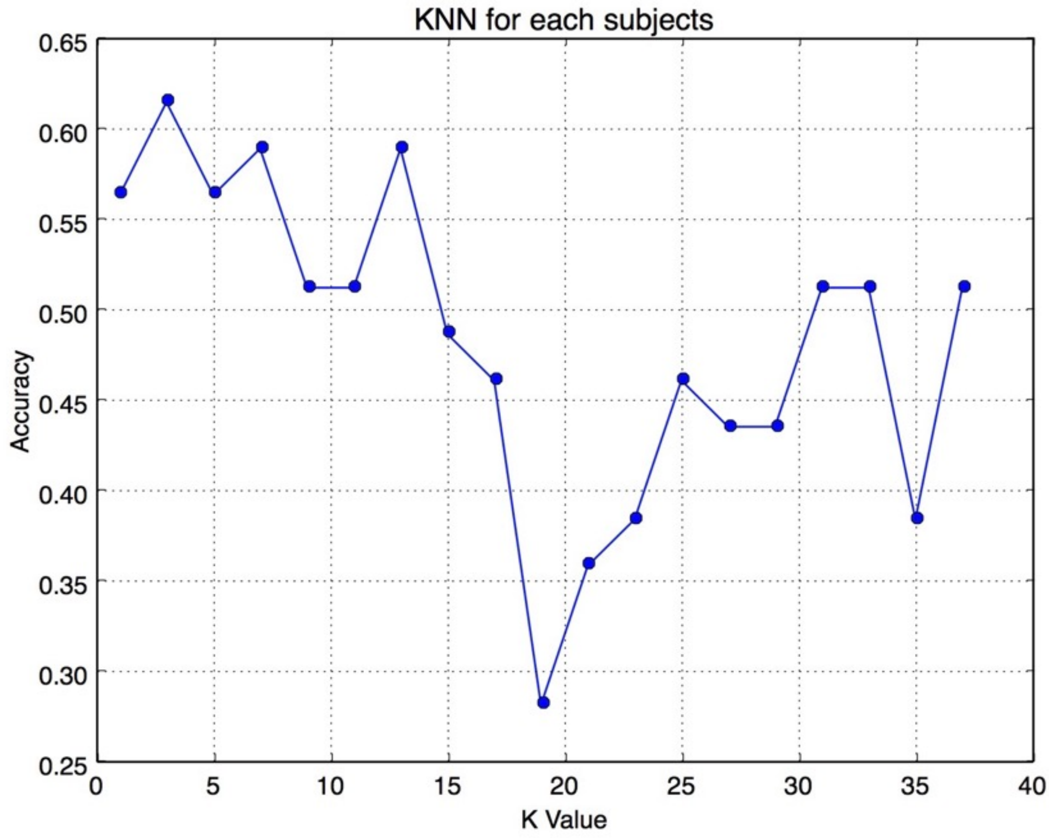


Fig. 7 - KNN for standard deviation data (second dataset)

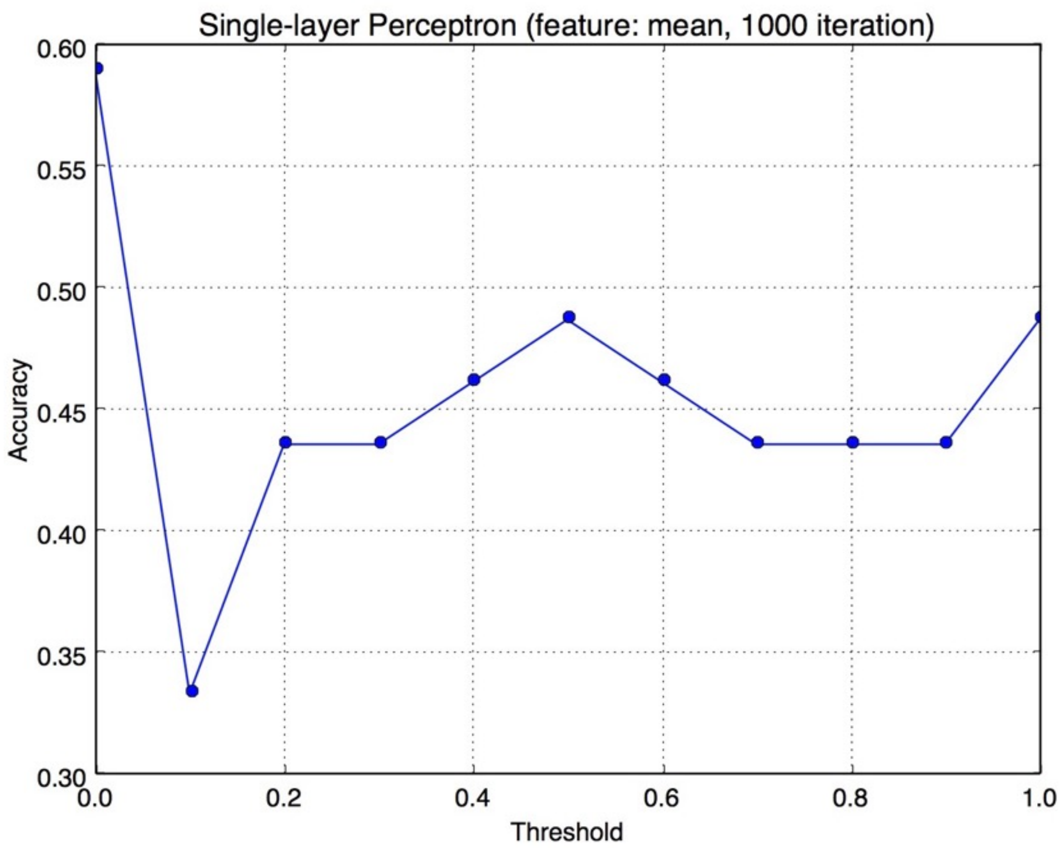


Fig.8 - Perceptron, mean and median of data for threshold (no bias, has kernel) (second dataset)

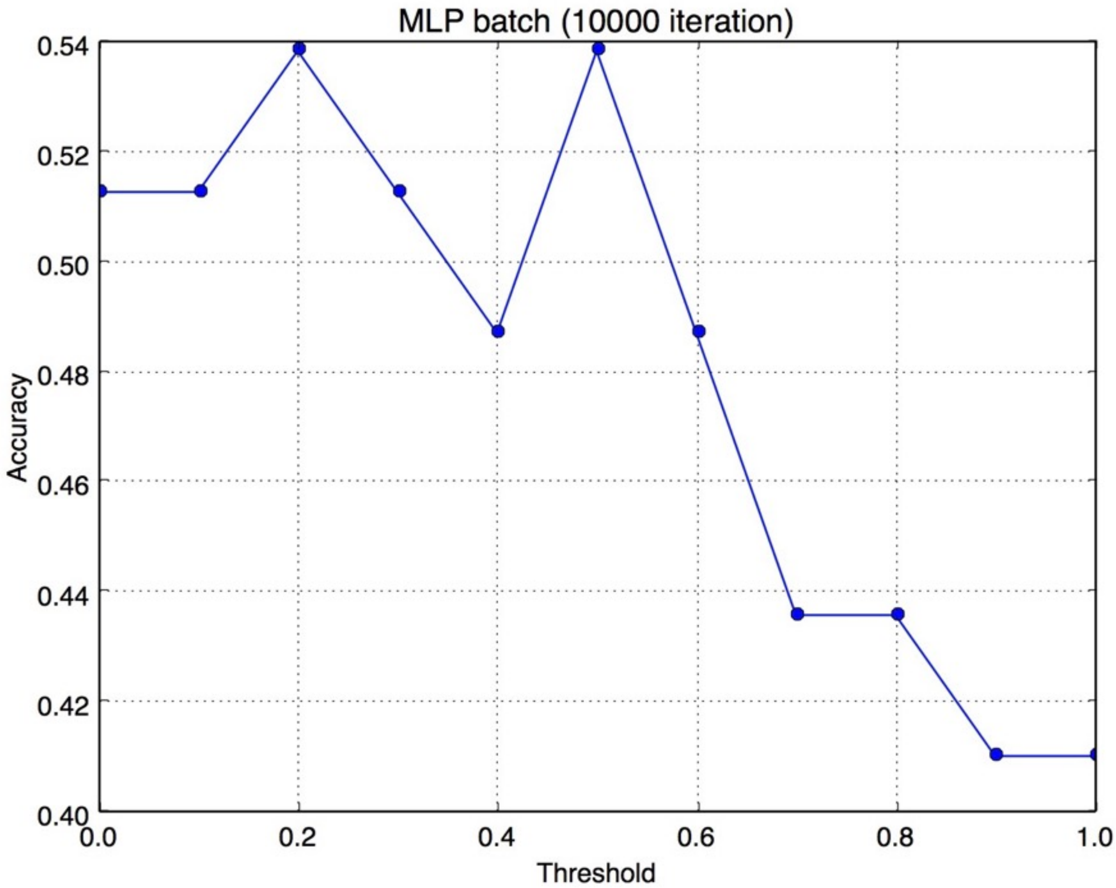


Fig.9 - MLP (sequential) for 3 hidden layer node (‘linear’)