Project Phase 4 Report

Data Mining

CSE 572

Fall 2017

Submitted to:

Professor Ayan Banerjee

Ira A. Fulton School of Engineering

Arizona State University

Submitted by:
Jigar Domadia (jdomadia@asu.edu)
Joseph Campbell (jacampb1@asu.edu)
Rajrshi Raj Shrestha (rrshrest@asu.edu)
Sai Pramod Kolli (skolli6@asu.edu)
November 28, 2017

Table of Contents

1 Introduction

This project is a part of course requirement for Data Mining (CSE572) Class of Fall 2017. The project is to provide solution for estimating the food intake by using the wristband sensor. The information gathered from the wristbands were analyzed to obtain a logical conclusion regarding when the person is eating the food and when he/she is not eating any food.

2 Team Members

Following are the group member of this project

Jigar Domadia (jdomadia@asu.edu)

Joseph Campbell (jacampb1@asu.edu)

Rajrshi Raj Shrestha (rrshrest@asu.edu)

Sai Pramod Kolli (skolli6@asu.edu)

3 Project Phase 1

The first phase of the project was to collect data. Each person from a group were to go to IMPACT lab and take video of eating some food from a designated plate. There were 4 individual portions around four corners of the plate and the student had to pick food from each portion with total 10 cycles, each cycle containing four eating action. The process is repeated for both spoon and fork. So there were total 40 eating action. Unique gesture such as snapping was done at the beginning so that the beginning of the eating action can be easily identified in wristband sensors.

4 Project Phase 2

The second phase of the project was to annotate the video that was taken in phase 1. The video file was provided by TA with CSV files which had data for all eating actions. The video files were loaded into a MATLAB tool (*Labeling.m* and *Labeling.fig*) provided by the TA/Professor. For each eating action two frames were recorded: one when the person starts picking the food from the plate and other when he/she completely puts it in his/her mouth. Also each eating action from each portion was numbered as 1 for food on lower right corner, 2 for lower left corner, 3 for upper left corner and 4 for upper right corner of the plate. The eating action numbering was repeated for each

cycles so there were 40 rows of data for each eating action of spoon and fork. Following are the snippet directly from the annotation file for eating action spoon data.

266, 287, 1

460, 484, 2

718, 745, 3

965, 986, 4

1265, 1299, 1

1497, 1533, 2

.....

.....

5 Project Phase 3

In Phase 3 of the project each group were to perform feature extraction and feature selection task. The raw sensor data from phase 1 and video annotation data from phase 2 were used to perform feature extraction and feature selection. Following were the tasks performed in phase 3 of project

5.1 Task 1: Synchronization

A MATLAB code was written to extract raw data created in phase 1 and save them into two separate classes of data: eating action class and non-eating action class. First the raw data for sensors were synchronized with the frame data created in phase 2. The synchronization was done by matching the last frame of video to the end time of the sensor data.

Each class of eating and non-eating action were saved into two different csv files: *eat_data.csv and noneat_data.csv*. The data inside csv files were arranged in such a way that each row contained all the data for single axis of a sensor for that individual eating or non-eating action.

Following data are directly from the eating and non-eating csv files.

Eating Action 1 Ori X -0.862 -0.861 -0.861 -0.861 -0.862 -0.864 -0.866 -0.866 -0.867 ……..

Eating Action 1 Ori Y -0.504 -0.506 -0.507 -0.507 -0.505 -0.502 -0.499 -0.498 -0.497……..

Eating Action 1 Ori Z -0.045 -0.045 -0.044 -0.042 -0.04 -0.036 -0.033 -0.03 -0.027 ……..

Eating Action 1 Acc X 0.853 0.887 0.875 0.929 0.987 0.905 0.902 0.867 0.88 0.901 ……..

…….. …….. …….. ……..

…….. …….. …….. ……..

5.2 Task 2: Feature Extraction

The exported data from Task 1 is initially organized into two three-dimensional matrices with dimensions NxMxP; one matrix contains sensor data corresponding to eating actions and the other matrix non-eating actions. Each matrix contains N unique observations/actions (2568 in our case), M data streams for each sensor (18), and a time series of length P which represents the raw sensor data. The actual length of the sensor data time series would vary between observations and sensors depending on the length of the action and the sampling frequency of the device (50 Hz for IMU and 100 Hz for EMG). Furthermore, the length of non-eating actions was typically longer than eating actions. For this task, following feature extraction methods were selected. Please see Phase 2 report for detailed explanation of each extraction methods.

1) Variance
2) Discrete Wavelet Transform
3) Fast Fourier Transform
4) Temporal Location of Max Value
5) Windowed Mean

5.3 Task 3: Feature Selection

In this Task we analyzed the feature extraction methods we picked in Task 2 and analyzed which features were responsible the most variance in the data set, according to Principle Component Analysis. PCA was certainly helpful in this scenario. We were able to reduce our feature matrix size from Nx33 to Nx1 and still able to capture 72.5% of the variance in the combined data set. This is important, since features with variance are required to distinguish classes. Intuitively, if a feature has little variance (i.e. it doesn't change), then it can't be used to distinguish observations because it doesn't tell us anything. But if a feature does change and thus exhibits variance then it can. When we used the top two principal components, we were able to capture 89.0% of the data set with an Nx2 feature matrix

6 Project Phase 4

The new features sets obtained from Principal Component Analysis from Phase 3 is used to train and test three different types of classifiers in Phase 4: Decision Trees, Support Vector Machines and Neural Networks. There are three different MATLAB functions with *DataAnalysis.m*; one for each classifier. MATLAB codes are submitted along with this report. There are an additional two phases (or tasks) for Phase 4: User dependent analysis and User independent analysis. Each MATLAB function for the corresponding classifier does both user dependent and user independent task. The two tasks are further discussed in detail below. To proceed with the Phase 4 analysis, eating and non eating actions for each group were mapped to a single CSV file. The csv file that contains the eating and non eating actions for each group is saved into *"mapping.csv"*. MATLAB code to generate *"mapping.csv"* file is inside *"DataParser.m"* file. Once the mapping for the eating and non eating actions are done, PCA components are exported to the following CSV files: *"eat_pca.csv"* and *"noneat_pca.csv"*. Data from each CSV files from PCA analysis is used to group the data for training and testing each machine. Please refer to ReadMe.txt for the exact commands to execute our code.

6.1 Task 1: User Dependent Analysis

In this task the data obtained from Phase 3 is divided into two sets: train and test. We randomly sampled 60% of the individual eat/non-eat actions from each group and compiled them into a single training matrix for a total size of 3077x33; the 33 columns represent our PCA-transformed extracted features from Phase 3. The random sampling is to ensure that we don't bias our train data by only selecting actions from the beginning or ending of the data collection session. The remaining actions from each group (40%) were compiled into a test matrix of size 2061x33. Labels were constructed for each action indicating whether it is an eat action (1) or non-eat action (0). In this way we model our problem as binary classification.

We would also like to point out that our group indices as reported in this section do not align with the actual group indices in the class. This is because we developed our data partitioning strategy before the group mappings were sent out, under the assumption that 2 consecutive files belonged to the same group (this was confirmed after the mappings were released). Thus the mapping between our reported group indices and the actual group numbers is given below.

Group index (ours)	**Group number (actual)**
1	27
2	2
3	33
4	21
5	31
6	32
7	11
8	1
9	10
10	28

11	16
12	24
13	5
14	23
15	7
16	13
17	3
18	29
19	25
20	15
21	9
22	19
23	30
24	14
25	18
26	26
27	4
28	20
29	17
30	6
31	22
32	8
33	12

Note that not all groups have exactly 40 actions as expected. Some annotation files contained incorrect entries and so were filtered out, and in other cases the EMG/IMU recordings were incomplete. For example, in at least one case the recordings were missing data from either the start or the end (it is impossible for us to know which without extensive analysis) and so we fail to find actions at the beginning of the annotation file (because we work backwards from the end). This can be seen for annotation file *73741. According to the summary metadata, it is 373 seconds long. By comparison, *63307 is 268 seconds long. As such we would expect *73741 to contain more data rows than *63307 but this is not the case. *73741's IMU file contains 11365 rows and *63307's contains 13399 rows. Thus we have lost data. We print out the names of the files for which we have failed to collect all annotations during parsing.

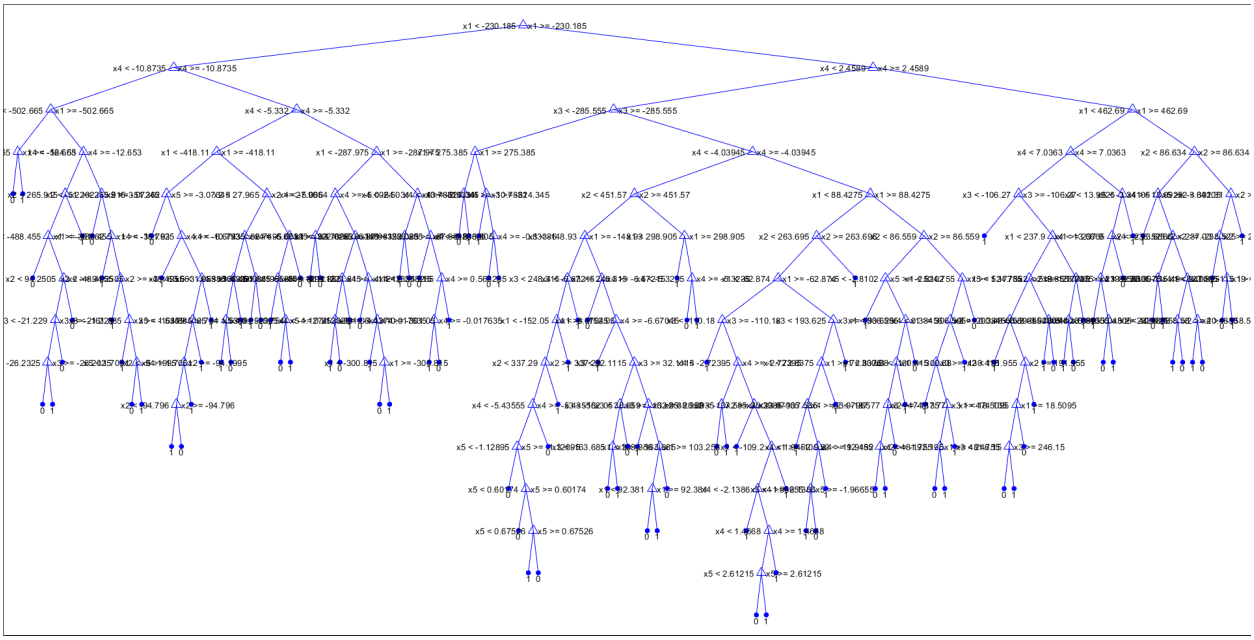
The remainder of this section will detail each of our three classification methods (Decision Tree, Support Vector Machine, and Neural Network) and four result metrics (Precision, Recall, F1 score, and Area Under Curve for ROC). We also provide the ROC plot for the full test data set (not broken into groups) as we find it interesting.

6.1.1 Accuracy metrics for Decision Tree

For our decision tree classifier we iteratively trained and tested with a variable number of principle components to determine which yielded the best accuracy. We opted for the first 5 PCA components, as using more than 5 did not show any significant improvement in test accuracy. The

principal component matrix along with the training data were fed into the MATLAB decision tree module *"fitctree"*. *"fitctree"* returns a fitted binary classification decision tree based on the PCA component and labels contained in training data. MATLAB code *"predict"* was used to get the predicted labels and which is further used to compute the true positive, true negative, false positive and false negative. These values were further used to compute the precision, recall and F1 score for each user. MATLAB code *"perfcurve"* was used to get the area under the curve.

The decision tree contained 249 nodes and split based on the Gini diversity index. We utilized the default values for most of the MATLAB parameters, such as the minimum parent size of 10 and minimum leaf size of 1. The full decision tree is shown below.



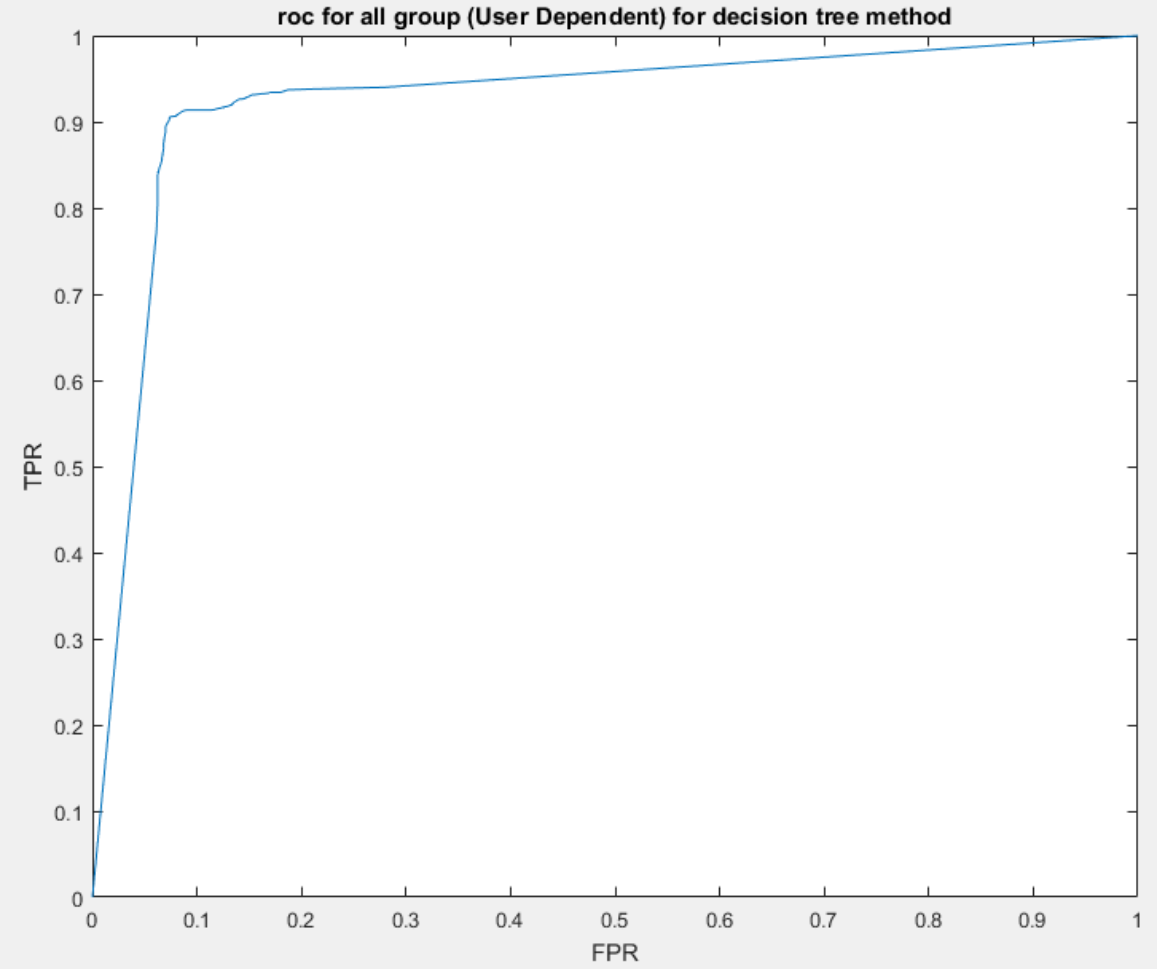
The MATLAB code for our decision tree analysis can be found in the *analyze_dt* function of *DataAnalysis.m* and can be executed with "*analyzer.analyze_dt();*". The following table provides our result metrics.

```
**********************************
*  TASK 1 OUTPUT – DECISION TREE  *
**********************************
```

Group index	Precision	Recall	F1 score	AUC (ROC)
1	0.966667	0.935484	0.95082	0.962366
2	0.980769	0.927273	0.953271	0.979167
3	0.952941	0.94186	0.947368	0.963343
4	0.957627	0.957627	0.957627	0.981934
5	0.901316	0.925676	0.913333	0.5875
6	0.874346	0.938202	0.905149	0.92402
7	0.888393	0.947619	0.917051	1

8	0.868726	0.93361	0.9	0.882698
9	0.857143	0.926471	0.890459	0.858871
10	0.866667	0.934641	0.899371	0.982843
11	0.852691	0.887906	0.869942	0.640762
12	0.85974	0.892183	0.875661	0.961426
13	0.8487	0.8975	0.872418	0.885714
14	0.856522	0.901602	0.878484	0.981481
15	0.847082	0.901499	0.873444	0.913235
16	0.850288	0.894949	0.872047	0.837798
17	0.855311	0.891221	0.872897	0.900985
18	0.864028	0.896429	0.87993	0.985119
19	0.861564	0.89966	0.8802	0.929563
20	0.867079	0.903382	0.884858	0.965787
21	0.871111	0.904615	0.887547	0.97931
22	0.874465	0.908148	0.890988	0.984615
23	0.865229	0.911932	0.887967	0.866518
24	0.868661	0.910082	0.888889	0.936765
25	0.87234	0.911111	0.891304	0.963343
26	0.862605	0.910467	0.88589	0.822917
27	0.863953	0.902795	0.882947	0.865686
28	0.867039	0.905484	0.885845	0.966667
29	0.868958	0.908989	0.888523	0.978495
30	0.872539	0.912243	0.891949	0.981916
31	0.875125	0.913452	0.893878	0.946839
32	0.876819	0.91498	0.895493	0.97931
33	0.875351	0.917485	0.895923	0.905172

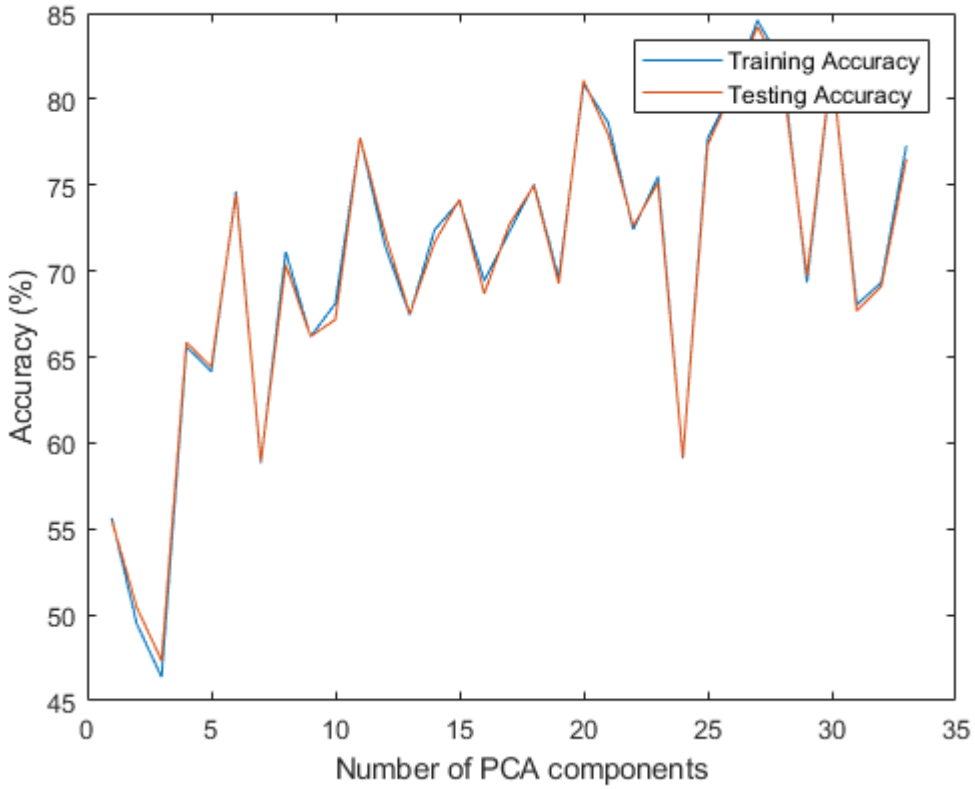
Below is the ROC plot for the test data from all groups.



6.1.2 Accuracy metrics for Support Vector Machine

For our SVM classifier we utilized a linear kernel such that our eating and non-eating actions were separated by a hyperplane. Non-linear kernels were tested, such as a Gaussian and Radial Basis Function kernel, however, they did not provide better results than the simple linear kernel and so we utilized that.

Furthermore, we iteratively trained and tested with a variable number of principle components, from 1 to 33 (all of them) and determined that using 27 components yields the maximum accuracy over the user-dependent data set (Phase 1). Thus, all training and testing utilized 27 of our PCA components.



The trained classifier utilized 375 support vectors; 151 were for eating actions and 224 for non-eating. We do not provide the support vectors here as they are difficult to visualize in 27 dimensions and the raw data does not provide any significant benefit.

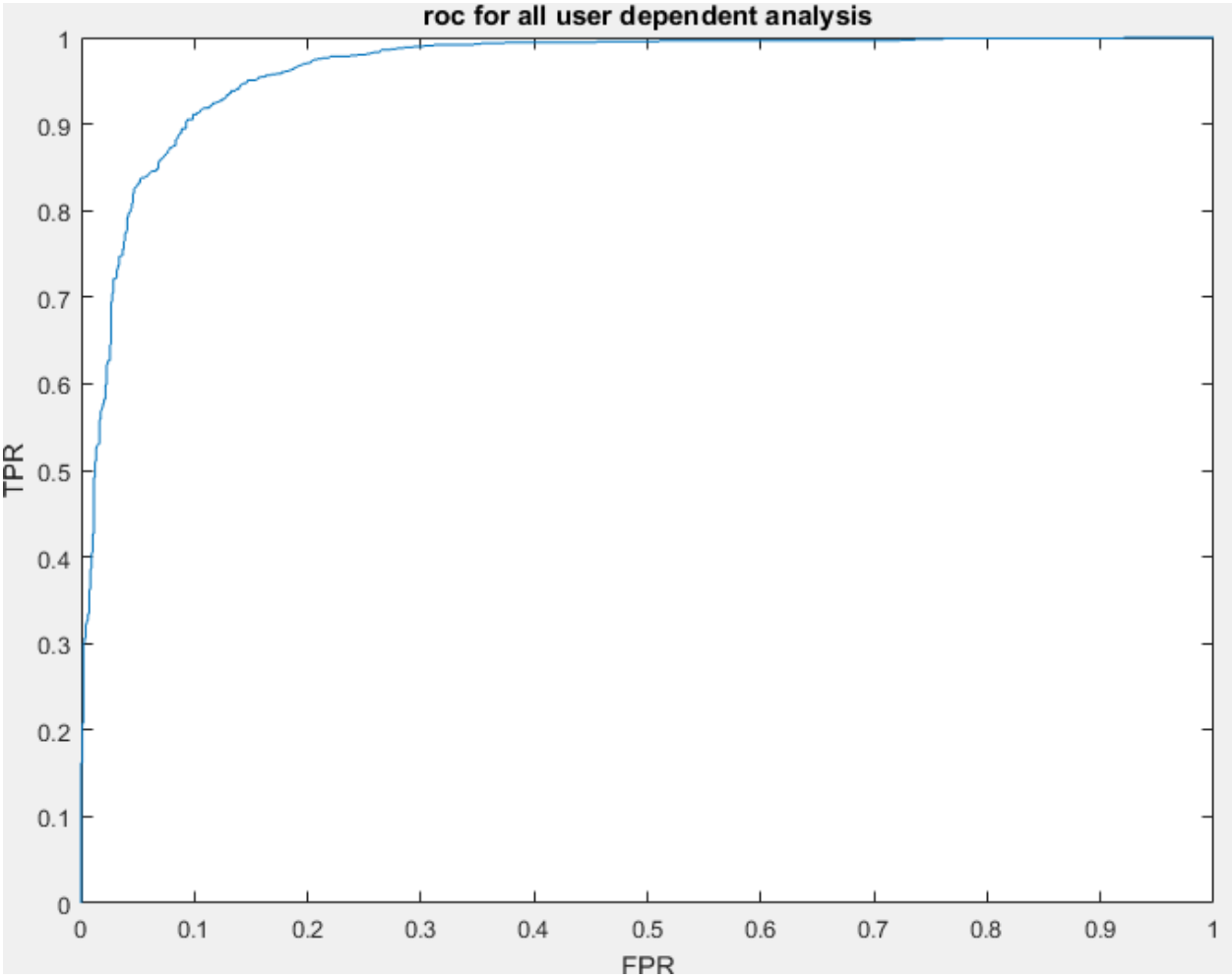
The MATLAB code for our SVM analysis can be found in the *analyze_svm* function of *DataAnalysis.m* and can be executed with "*analyzer.analyze_svm();*". The following table provides our result metrics.

```
*****************************************
* TASK 1 OUTPUT – SUPPORT VECTOR MACHINE *
*****************************************
```

Group index	Precision	Recall	F1 score	AUC (ROC)
1	1	1	1	1
2	1	0.606061	0.754717	1
3	1	1	1	1
4	0.961538	0.757576	0.847458	0.969697
5	0.869565	0.8	0.833333	0.913684
6	0.904762	0.974359	0.938272	0.985641
7	1	0.903226	0.949153	0.999022
8	1	0.321429	0.486486	0.998848
9	0.846154	0.88	0.862745	0.9375
10	0.931034	1	0.964286	0.996997

11	0.913043	0.636364	0.75	0.84965
12	1	0.481481	0.65	0.979424
13	0.965517	0.903226	0.933333	0.986895
14	1	0.837838	0.911765	0.993994
15	0.708333	0.485714	0.576271	0.628571
16	1	0.807692	0.893617	1
17	1	0.83871	0.912281	0.975562
18	0.956522	0.647059	0.77193	0.97549
19	1	0.933333	0.965517	0.994118
20	1	0.909091	0.952381	0.997067
21	1	0.857143	0.923077	1
22	1	0.5	0.666667	0.992063
23	0.842105	0.969697	0.901408	0.974186
24	0.958333	0.676471	0.793103	0.97451
25	0.90625	0.90625	0.90625	0.948242
26	0.860465	1	0.925	0.941942
27	0.941176	0.470588	0.627451	0.965686
28	0.945946	1	0.972222	1
29	0.96875	0.911765	0.939394	0.984314
30	1	1	1	1
31	1	0.857143	0.923077	0.998095
32	1	0.8	0.888889	0.992857
33	1	0.88	0.93617	0.992941

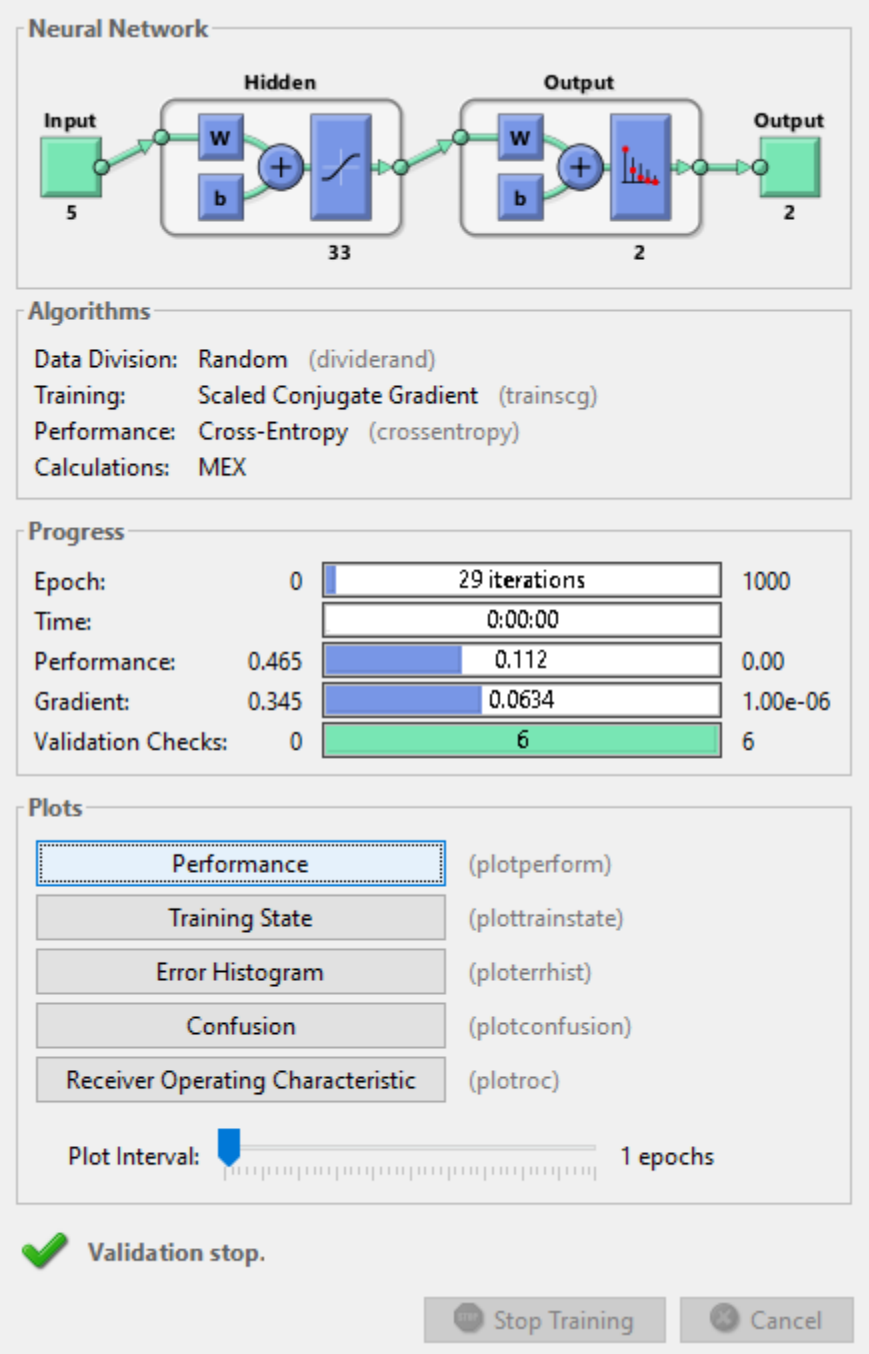
Below is the ROC plot for the test data from all groups.


roc for all user dependent analysis
TPR
FPR
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1

6.1.3 Accuracy metrics for Neural Net Machine

Our neural network classifier utilized a simple Feed Forward neural network for binary classification. We selected only 1 hidden layer with 33 neurons due to results from testing. The hidden layer activation function is the hyperbolic tangent sigmoid function while our output layer is the softmax. As-is typical in classification tasks, we encoded our data with 1-hot encoding and utilized a cross-entropy loss for training via the scaled conjugate gradient method. This means the output layer has two nodes: one for eat-actions and one for non-eat and a probability is generated for both. Based on limited testing, we also chose to utilize 5 PCA components for training and testing. Therefore the number of input nodes is 5.

The MATLAB neural network toolbox was employed to train our network and the results can be seen below.

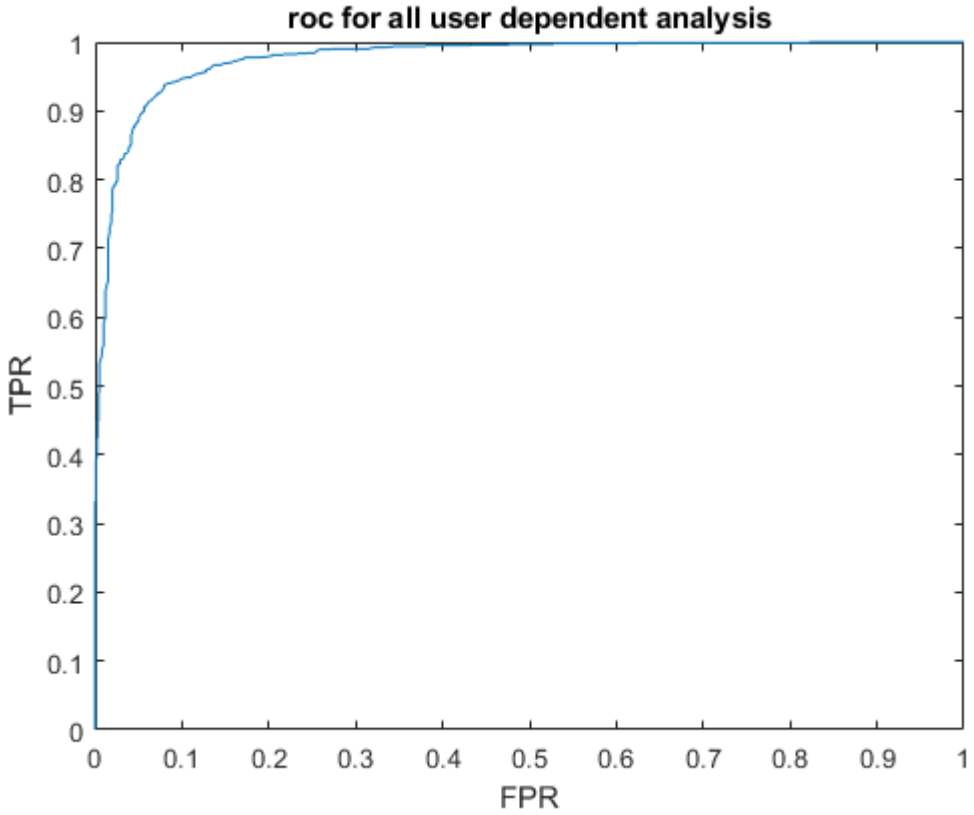


The MATLAB code for our NN analysis can be found in the *analyze_nn* function of *DataAnalysis.m* and can be executed with "*analyzer.analyze_nn(),*". The following table provides our result metrics.

****
* TASK 1 OUTPUT – NEURAL NETWORK MACHINE *
****

Group index	Precision	Recall	F1 score	AUC (ROC)
1	1	0.96875	0.984127	0.990489
2	1	0.9375	0.967742	1
3	1	1	1	1
4	0.967742	0.967742	0.967742	0.999022
5	0.666667	0.916667	0.77193	0.791667
6	0.911765	1	0.953846	0.991935
7	0.942857	0.970588	0.956522	0.997059
8	1	0.694444	0.819672	0.998932
9	0.9	0.931034	0.915254	0.982759
10	0.941176	1	0.969697	1
11	0.75	0.545455	0.631579	0.729437
12	1	0.903226	0.949153	0.999022
13	0.75	0.964286	0.84375	0.961224
14	0.970588	1	0.985075	0.999022
15	0.896552	0.722222	0.8	0.905556
16	1	0.956522	0.977778	1
17	1	0.933333	0.965517	1
18	1	1	1	1
19	0.967742	0.967742	0.967742	0.987903
20	0.966667	1	0.983051	1
21	0.941176	0.969697	0.955224	0.978495
22	1	0.884615	0.938776	0.998988
23	0.775	0.96875	0.861111	0.952009
24	0.958333	0.766667	0.851852	0.986275
25	1	0.966667	0.983051	1
26	0.789474	0.9375	0.857143	0.938477
27	1	0.794118	0.885246	1
28	0.918919	1	0.957746	0.99902
29	0.972222	0.972222	0.972222	0.994048
30	1	1	1	1
31	1	0.967742	0.983607	1
32	0.970588	0.970588	0.970588	0.99902
33	0.90625	1	0.95082	1

Below is the ROC plot for the test data from all groups.

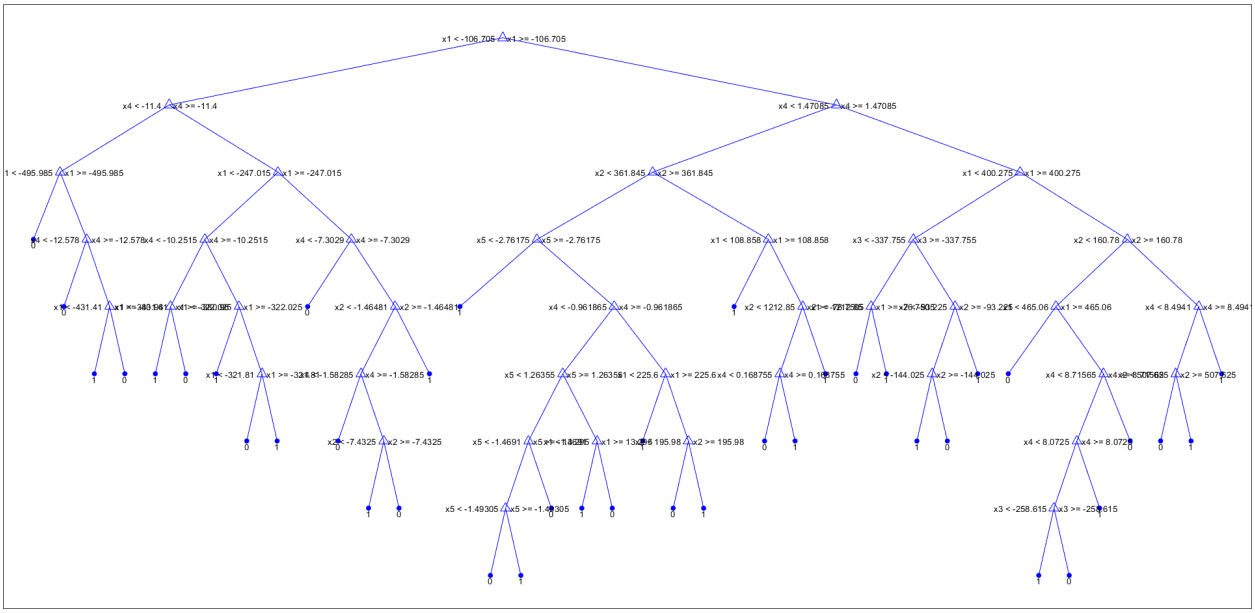


6.2 Task 2: User Independent Analysis

In this task 10 groups were uniformly randomly sampled and their feature points were used for training. The remaining 23 groups were used for testing. Note that because the groups were randomly chosen, the group indices don't necessarily correspond to the actual group indices. E.g., group index 1 does not mean group index 1 in the table given earlier. We failed to capture which specific groups were sampled. The training matrix has a size of 1576x33 and the test matrix 3562x33. These sizes can fluctuate between runs depending on which groups are sampled (not all groups have the same number of actions).

6.2.1 Accuracy metrics for Decision Tree

The decision tree contained 145 nodes and split based on the Gini diversity index. We utilized the default values for most of the MATLAB parameters, such as the minimum parent size of 10 and minimum leaf size of 1. The full decision tree is shown below.



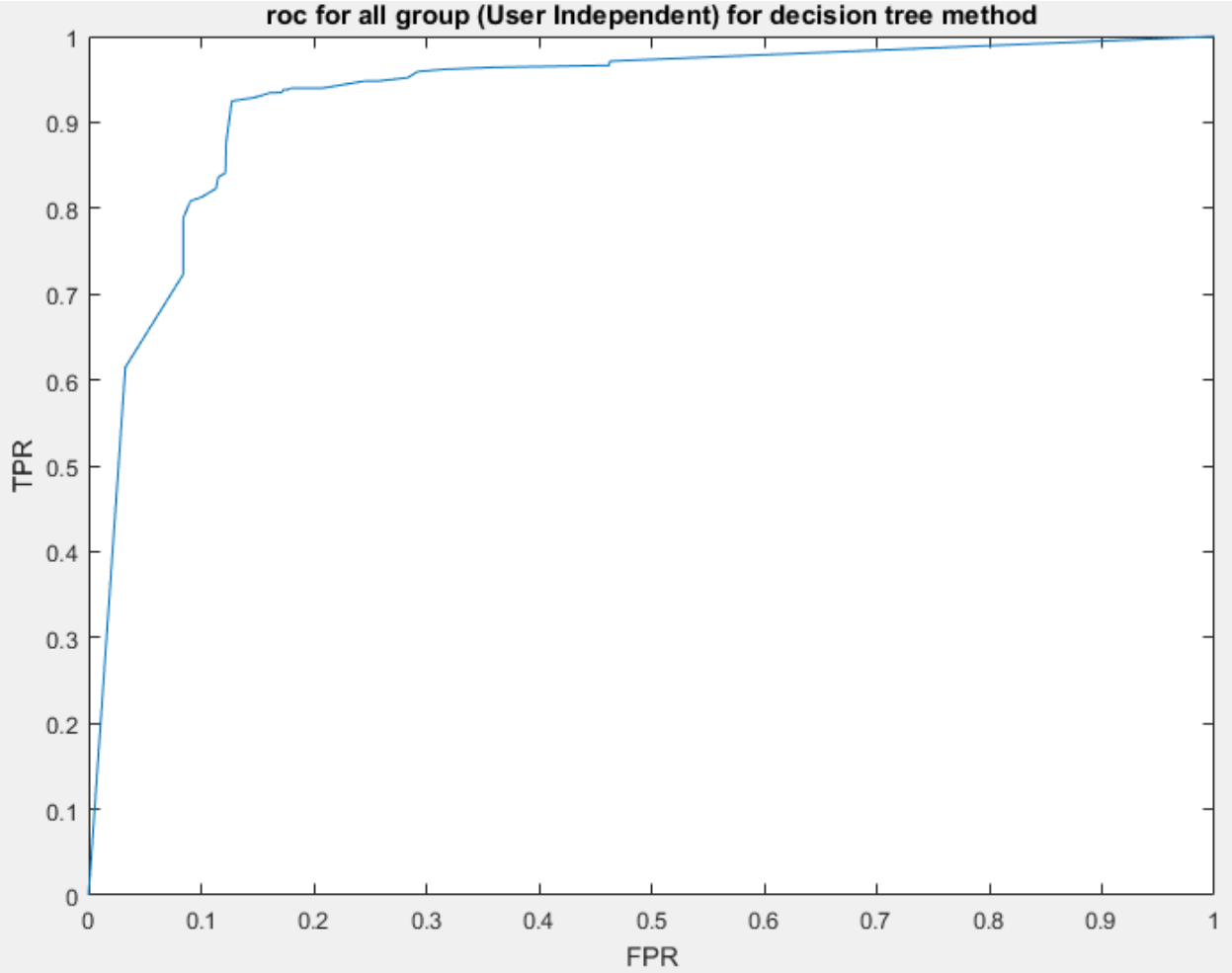
The MATLAB code for our decision tree analysis can be found in the *analyze_dt* function of *DataAnalysis.m* and can be executed with "*analyzer.analyze_dt();*". The following table provides our result metrics.

****
* TASK 2 OUTPUT – DECISION TREE *
****

Group index	Precision	Recall	F1 score	AUC (ROC)
1	0.987179	0.950617	0.968553	0.981786
2	0.95679	0.962733	0.959752	0.985547
3	0.95082	0.966667	0.958678	0.97901
4	0.867692	0.949495	0.906752	0.778086
5	0.800443	0.95756	0.871981	0.918594
6	0.813653	0.964989	0.882883	0.970469
7	0.809295	0.94041	0.86994	0.833906
8	0.786999	0.932787	0.853713	0.804185
9	0.796319	0.94058	0.862458	0.965938
10	0.803769	0.941558	0.867225	0.961719
11	0.786065	0.942353	0.857143	0.906641
12	0.775893	0.934409	0.847805	0.886406
13	0.7875	0.935644	0.855204	0.951094
14	0.787316	0.934803	0.854744	0.911873
15	0.797671	0.937553	0.861974	0.980859
16	0.801234	0.935949	0.863368	0.955625
17	0.808442	0.936795	0.867898	0.954375
18	0.797814	0.932576	0.859948	0.866094
19	0.8	0.935484	0.862454	0.972841

20	0.802402	0.9375	0.864706	0.931563
21	0.808977	0.940534	0.869809	0.997656
22	0.815168	0.938693	0.872581	0.971651
23	0.819672	0.939746	0.875612	0.974375

Below is the ROC plot for the test data from all groups.



6.2.2 Accuracy metrics for Support Vector Machine

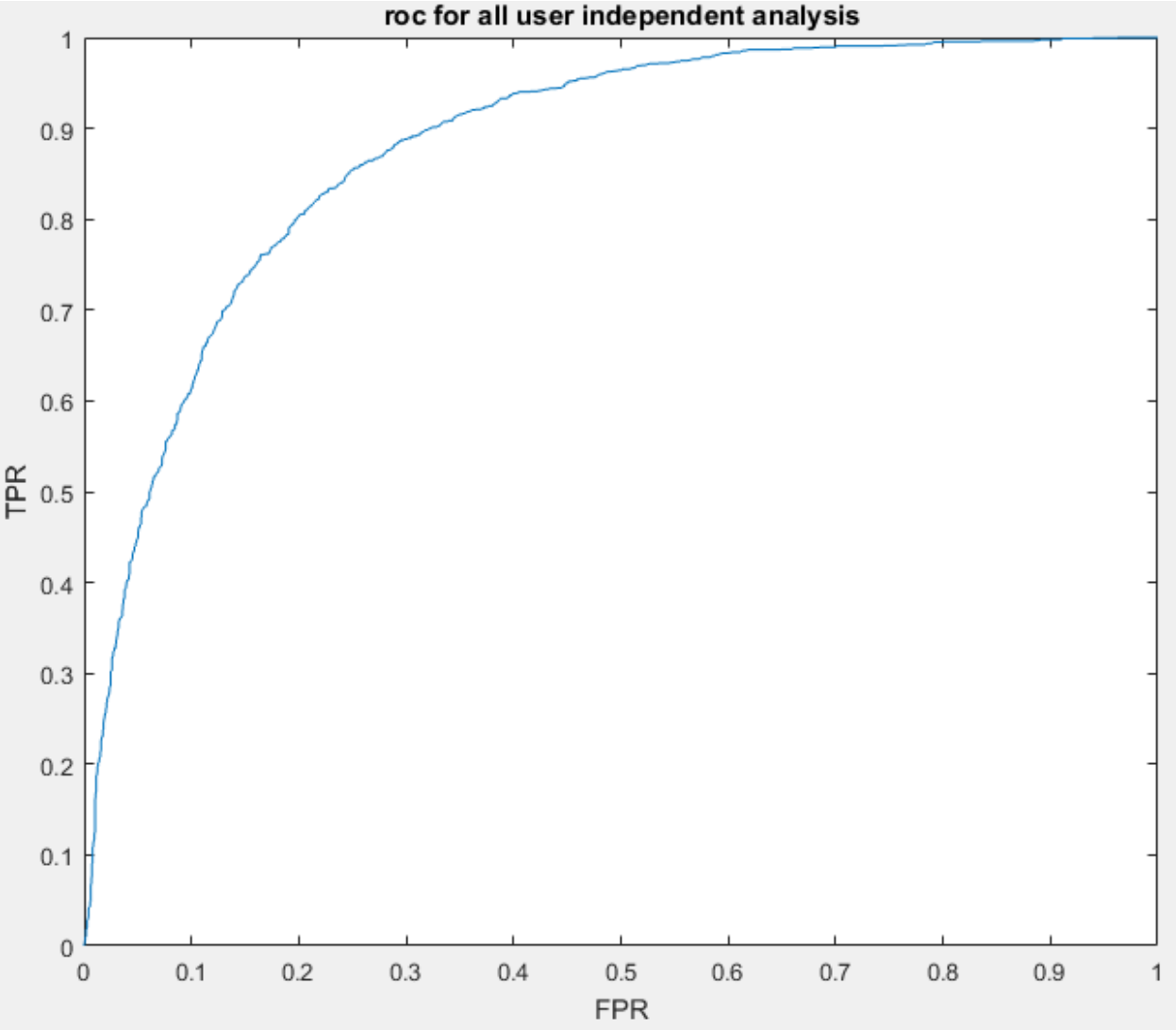
The trained classifier utilized 205 support vectors; 81 were for eating actions and 124 for non-eating. We do not provide the support vectors here as they are difficult to visualize in 27 dimensions and the raw data does not provide any significant benefit.

The MATLAB code for our SVM analysis can be found in the *analyze_svm* function of *DataAnalysis.m* and can be executed with "*analyzer.analyze_svm();*". The following table provides our result metrics.

****
* TASK 2 OUTPUT – SUPPORT VECTOR MACHINE *
****

Group index	Precision	Recall	F1 score	AUC (ROC)
1	0.969697	0.941176	0.955224	0.988106
2	0.945946	0.432099	0.59322	0.943454
3	0.761364	0.8375	0.797619	0.857656
4	0.862069	0.632911	0.729927	0.816055
5	0.765306	0.9375	0.842697	0.876603
6	0.52	0.1625	0.247619	0.914312
7	0.708333	0.6375	0.671053	0.814493
8	0.659091	0.3625	0.467742	0.681579
9	0.978261	0.5625	0.714286	0.965938
10	0.910714	0.784615	0.842975	0.865089
11	0.906667	0.85	0.877419	0.943281
12	0.90625	0.3625	0.517857	0.958438
13	0.953488	0.5125	0.666667	0.978906
14	1	0.1875	0.315789	0.970156
15	0.73913	0.918919	0.819277	0.897335
16	0.953488	0.5125	0.666667	0.945312
17	0.762887	0.925	0.836158	0.893125
18	0.657895	0.9375	0.773196	0.835
19	0.948718	0.4625	0.621849	0.977969
20	0.929412	1	0.963415	0.984458
21	0.942857	0.825	0.88	0.982595
22	0.974026	0.9375	0.955414	0.98125
23	0.921053	0.4375	0.59322	0.96788

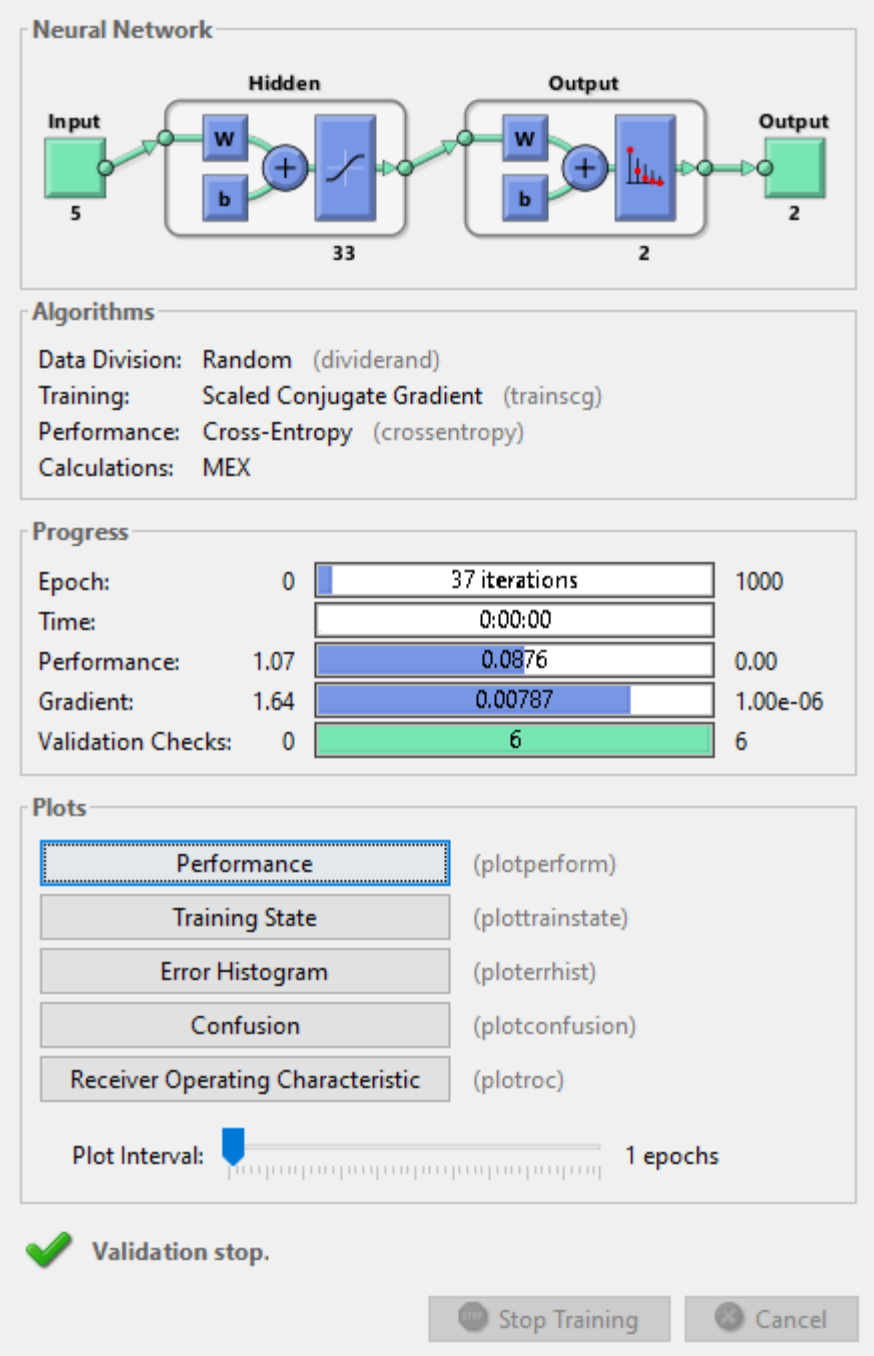
Below is the ROC plot for the test data from all groups.



6.2.3 Accuracy metrics for Neural Net Machine

The architecture for our network did not change in this task. As-is typical in classification tasks, we encoded our data with 1-hot encoding and utilized a cross-entropy loss for training via the scaled conjugate gradient method. This means the output layer has two nodes: one for eat-actions and one for non-eat and a probability is generated for both.

The MATLAB neural network toolbox was employed to train our network and the results can be seen below.



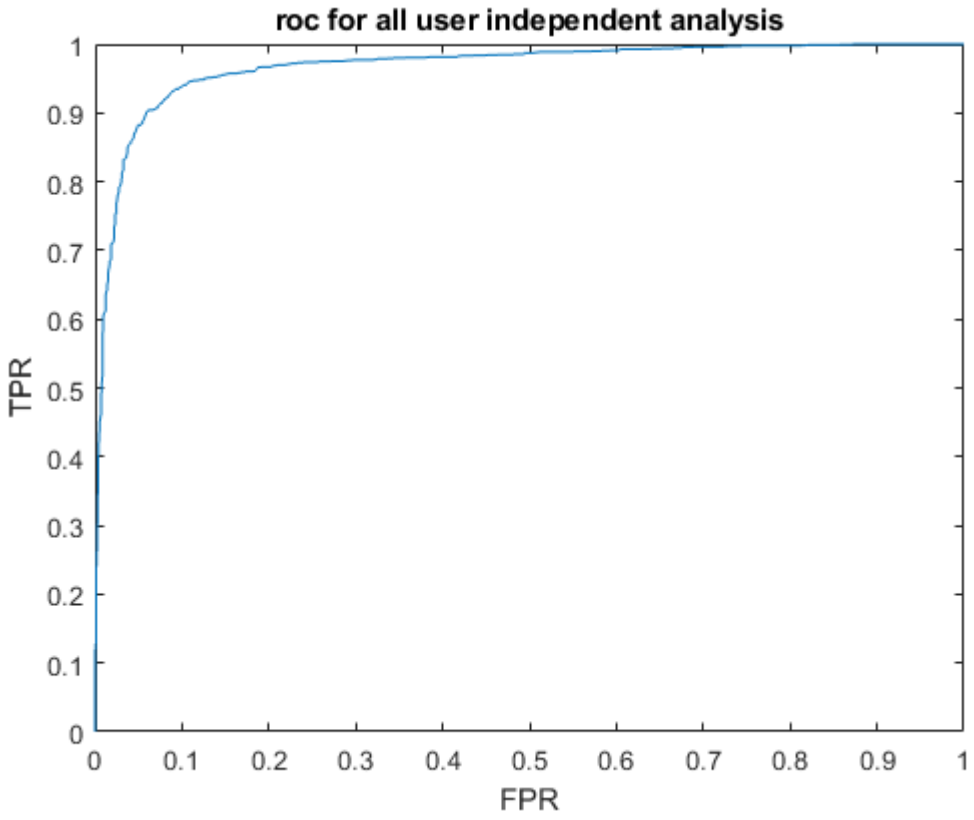
The MATLAB code for our NN analysis can be found in the *analyze_nn* function of *DataAnalysis.m* and can be executed with "*analyzer.analyze_nn();*". The following table provides our result metrics.

```
*****************************************
* TASK 2 OUTPUT – NEURAL NETWORK MACHINE *
*****************************************
```

Group index	Precision	Recall	F1 score	AUC (ROC)

1	0.985507	1	0.992701	0.995675
2	0.97561	1	0.987654	0.998437
3	0.671642	0.789474	0.725806	0.804038
4	0.738318	0.9875	0.84492	0.907532
5	0.91954	1	0.958084	0.99625
6	0.927536	0.8	0.85906	0.955435
7	0.831325	0.945205	0.884615	0.961562
8	0.97561	1	0.987654	0.999688
9	0.761905	0.4	0.52459	0.73913
10	0.737864	0.95	0.830601	0.912025
11	0.9875	0.9875	0.9875	0.999688
12	0.930556	0.8375	0.881579	0.956762
13	1	0.923077	0.96	0.976805
14	0.96	0.911392	0.935065	0.97728
15	0.963855	1	0.981595	0.999844
16	0.9875	0.9875	0.9875	0.99
17	1	0.975	0.987342	1
18	0.962025	0.95	0.955975	0.985
19	0.877778	1	0.934911	0.991828
20	0.963415	0.9875	0.975309	0.990348
21	1	1	1	1
22	1	0.901235	0.948052	0.998933
23	0.975309	0.9875	0.981366	0.996519

Below is the ROC plot for the test data from all groups.



7 Conclusion

With the completion of this project we gained a much more detailed insight regarding the machine learning process as a whole. The project started with data collection phase where we took the video of eating some food from a plate. Then we annotated the video frames using MATLAB tool provided by the professor. Once the annotation was completed, we analyzed the data using various feature extraction tools. The extracted features were further analyzed which included reduction of the feature space and keeping only the features that showed significant separation between two classes: eating and non-eating. After the selection of relevant feature sets, the data set was divided into training and testing sets which were then analyzed using three different machines: Decision Tree, Support Vector Machine and Neural Network. While all classifiers were able to fairly accurately classify eating and non-eating actions in both phases, the neural network's performance in particular is quite good. There were no significant differences observed between the results for support vector machine and decision tree. We honestly did not expect classification to perform as well as it did, particularly in Phase 2 where we are classifying actions for users that have not been seen in training. This speaks well to the generalization capabilities for neural networks (not surprising given 1 hidden layer is enough to act as a universal approximator).