1 Artificial Neural Network

1.1 Introduction

Artificial neurons, which are a set of interconnected units or nodes that loosely resemble the neurons in a biological brain, are the foundation of an ANN. Like the synapses in a biological brain, each connection has the ability to send a signal to neighboring neurons. An artificial neuron can signal neurons that are connected to it after processing signals that are sent to it. The output of each neuron is calculated by some non-linear function of the sum of its inputs, and the "signal" at a connection is a real number. Edges refer to the connections. The weight of neurons and edges typically changes as learning progresses. The weight alters the signal strength at a connection by increasing or decreasing it.

1.2 Model representation

Our neural network is shown in the following figure.

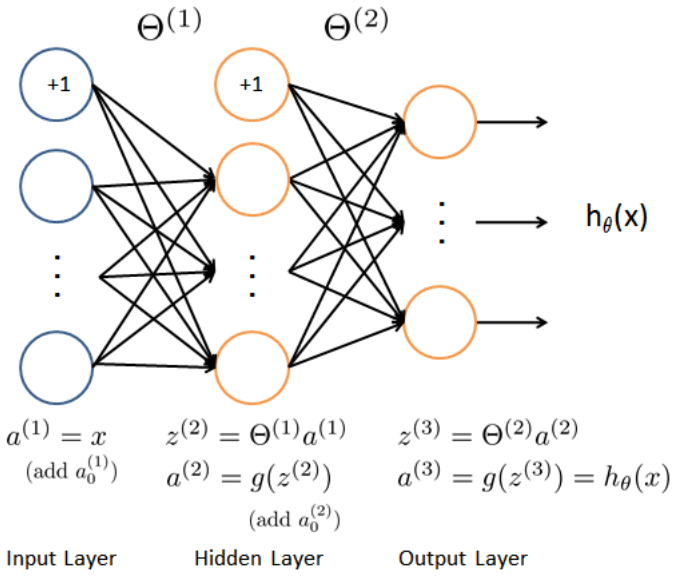

Fig. 1. ANN model representation

It has 3 layers: an input layer, a hidden layer and an output layer. Recall that our inputs is the symptoms of diseases and Severity column is the sum of all values in a case. The total number of symptoms are 142, corresponding to 142 input layer units (not counting the extra bias unit).

At first, we set an initial value for the units of hidden layer is 10, which is neither too large nor too small and 42 output units (corresponding to 42 diseases).

Activation function: An activation function in a neural network describes how a node or nodes in a layer of the network transform the weighted sum of the input into an output.

A "transfer function" is another name for the activation function. Numerous activation functions have nonlinear behavior, which is referred to as "nonlinearity" in network or layer design.

In this project, we use sigmoid function or logistic function as the activation function to compute the information from layer to the following layers.

$$\text{sigmoid}(z) = g(z) = \frac{1}{1 + e^{-z}}$$

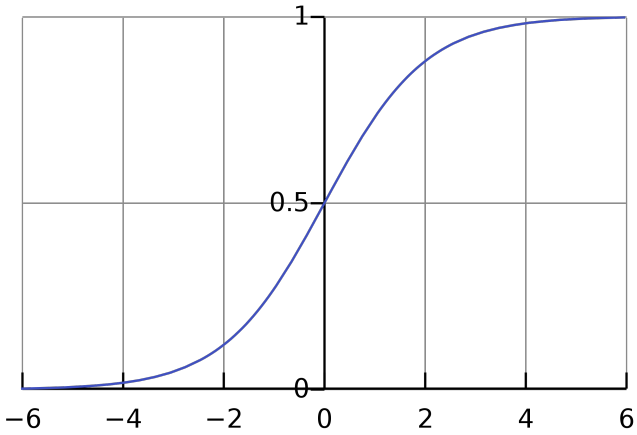

Fig. 2. Illustration of sigmoid function

Encoding Labels: As mentioned above, the original labels were the name of the disease, for the purpose of training a neural network, we need to encode the labels as vectors of dimension 42 containing only values 0 or 1, so that

$$y = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad \cdots \quad \text{or} \quad \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}.$$

1.3 Randomly Initialization

When training neural networks, it is important to randomly initialize the parameters for symmetry breaking. One effective strategy for random initialization is to randomly select values for $\Theta^{(l)}$ uniformly in the range $[-\epsilon_{init}, \epsilon_{init}]$.

Normalized Xavier Weight Initialization: The normalized xavier initialization method is calculated as a random number with a uniform probability distribution by choosing ϵ_{init} is $\epsilon_{init} = \frac{\sqrt{6}}{\sqrt{L_{in}+L_{out}}}$ where $L_{in} = s_l$ and $L_{out} = s_{l+1}$ are the number of units in the layers adjacent to Θ^l.

1.4 Feed forward process in MLPs and Cost function

The vector representation of x and $z^{(j)}$ is:

$$x = \begin{bmatrix} x_0 x_1 \cdots x_n \end{bmatrix} z^{(j)} = \begin{bmatrix} z_1^{(j)} z_2^{(j)} \cdots z_n^{(j)} \end{bmatrix}$$

Setting x = $a^{(1)}$ we obtain the general formula $z^{(i+1)} = \theta^{(i)} a^{(i)}$ and $a^{(i)} = g(z^{(i)})$

The cost function: The cost function for the neural network (without regularization) is:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K} \left[-y_k^{(i)} \log \left(\left(h_\theta \left(x^{(i)} \right) \right)_k \right) - \left(1 - y_k^{(i)} \right) \log \left(1 - \left(h_\theta \left(x^{(i)} \right) \right)_k \right) \right]$$

where $h_\theta \left(x^{(i)} \right)$ is computed as shown in the neural network figure above, and K = 42 is the total number of possible labels. Note that $h_\theta(x^{(i)})_k = a_k^{(3)}$ is the activation (output value) of the k^{th} output unit. Also, recall that whereas the original labels (in the variable y) were the String that describe the name of the diseases but for the purpose of training a neural network, we need to encode the labels as vectors containing only values 0 or 1 by one hot encoder (a built-in encoder in sklearn lib)

$$y = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad \cdots \quad \text{or} \quad \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}.$$

Also, the cost function for neural networks with regularization is given by:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \sum_{k=1}^{K} \left[-y_k^{(i)} \log \left(\left(h_\theta \left(x^{(i)} \right) \right)_k \right) - \left(1 - y_k^{(i)} \right) \log \left(1 - \left(h_\theta \left(x^{(i)} \right) \right)_k \right) \right]$$

$$+\frac{\lambda}{2m}\left[\sum_{j=1}^{10}\sum_{k=1}^{142}\left(\Theta_{j,k}^{(1)}\right)^2+\sum_{j=1}^{42}\sum_{k=1}^{10}\left(\Theta_{j,k}^{(2)}\right)^2\right]$$

1.5 Back propagation

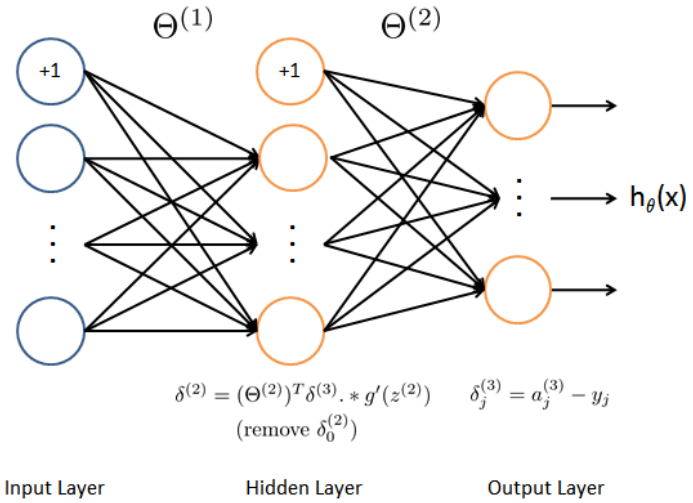

Fig. 3. Back propagation process

Given a training example $(x^{(t)}, y^{(t)})$, we will first run a "forward pass" to compute all the activations throughout the network, including the output value of the hypothesis $h_\theta(x)$. Then, for each node j in layer l, we would like to compute an "error term" $\delta_j^{(l)}$ that measures how much that node was "responsible" for any errors in our output.

For an output node, we can directly measure the difference between the network's activation and the true target value, and use that to define $\delta_j^{(3)}$ (since layer 3 is the output layer).

Step 1: Set the input layer's values $(a^{(1)})$ to the t^{th}training example $x^{(t)}$. Perform a feedforward pass, computing the activations $(z^{(2)}, a^{(2)}, z^{(3)}, a^{(3)})$ for layers 2 and 3. Note that you need to add a '+1' term to ensure that the vectors of activations for layers $a^{(1)}$ and $a^{(2)}$ also include the bias unit.

Step 2: For each output unit k in layer 3 (the output layer), set

$$\delta_k^{(3)} = \left(a_k^{(3)} - y_k\right)$$

where $y_k \in \{0, 1\}$ indicates whether the current training example belongs to class k ($y_k = 1$), or if it belongs to a different class ($y_k = 0$).

Step 3: For the hidden layer $l = 2$, set

$$\delta^{(2)} = \left(\Theta^{(2)}\right)^T \delta^{(3)} * g'\left(z^{(2)}\right)$$

Step 4: Accumulate the gradient from this example using the following formula. Note that you should skip or remove $\delta_0^{(2)}$.

$$\Delta^{(l)} = \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^{(T)}$$

Step 5: Obtain the (unregularized) gradient for the neural network cost function by dividing the accumulated gradients by $\frac{1}{m}$:

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)} = \frac{1}{m} \Delta_{ij}^{(l)}$$

1.6 Learning parameters using scipy.optimize.minimize

After having successfully implemented the neural network cost function and gradient computation, the next step we will use scipy's minimization to learn a good set parameters.

In [80]:
```
pred = utils.predict(Theta1, Theta2, X)
print('Training Set Accuracy: %f' % (np.mean(pred == y) * 100))
```

```
Training Set Accuracy: 97.257657
```

In [81]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

In [82]:
```
options= {'maxiter': 200}
lambda_ = 1
costFunction = lambda p: nnCostFunction(p, input_layer_size,
                                        hidden_layer_size,
                                        num_labels, X_train, y_train, lambda_)

res = optimize.minimize(costFunction,
                        initial_nn_params,
                        jac=True,
                        method='TNC',
                        options=options)

nn_params = res.x

Theta1 = np.reshape(nn_params[:hidden_layer_size * (input_layer_size + 1)],
                    (hidden_layer_size, (input_layer_size + 1)))

Theta2 = np.reshape(nn_params[(hidden_layer_size * (input_layer_size + 1)):],
                    (num_labels, (hidden_layer_size + 1)))
```

In [83]:
```
pred = utils.predict(Theta1, Theta2, X_test)
print('Test Set Accuracy: %f' % (np.mean(pred == y_test) * 100))
```

```
Test Set Accuracy: 97.214154
```

Fig. 4. Scipy.optimize.minimize set up and result

After having successfully implemented the neural network cost function and gradient computation, the next step we will use scipy's minimization to learn a good set parameters. The result below give information about the accuracy of the network on both training set and test set.

For the very first moment, we can see that the accuracy was pretty high, which indicates that the network learnt well on the dataset. But accuracy was just one of the criterion to evaluate the efficiency of the model, besides, we should investigate further about other metrics. We also need to divide the dataset again (by Cross-validation resampling technique) to tune the parameters to obtain the best result and reduce computational workloads.

1.7 Tuning parameters in Neural Network

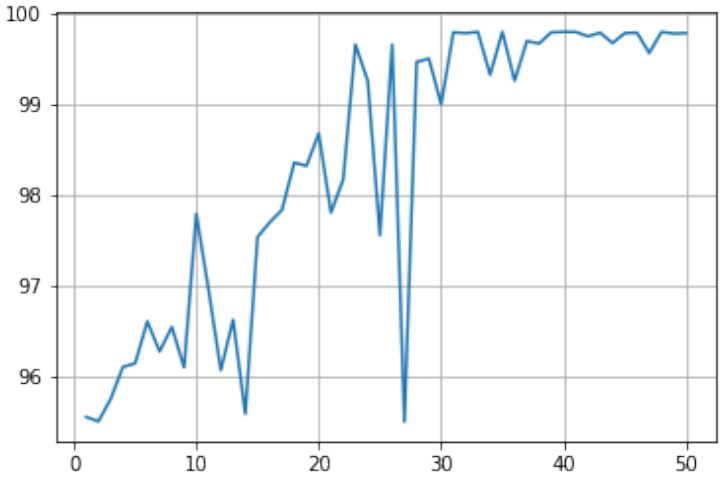

Fig. 5. Relation between number of nodes in hidden layer and accuracy

Number of neurons in hidden layers: As described from above figure, when the nodes pass 30, the line started to flatten and less fluctuate. Relatively, the accuracy is nearly 100 percent on cross-validation set.

We set the neurons in the hidden layer to a large enough number - 29 to explore more about the network demonstration. Consequently, the accuracy reach 99,77 percent. But accuracy alone is not enough to evaluate our model. Additionally,, we try to find the precision, recall and F1-score.

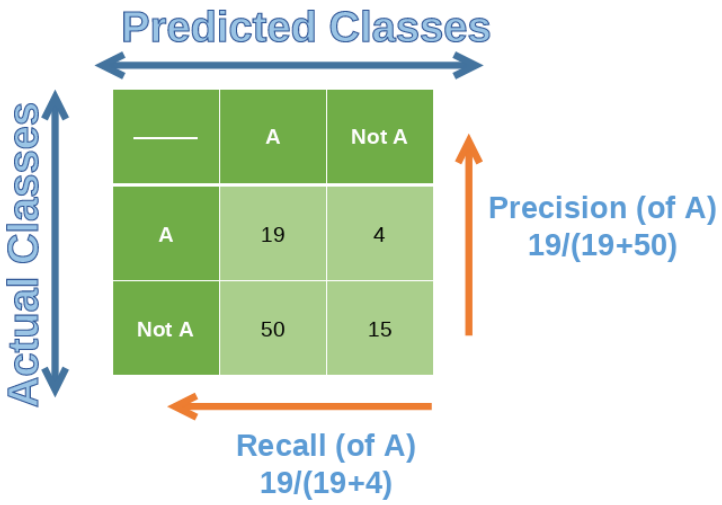

Fig. 6. Example of confusion matrix

Precision Precision is defined as the ratio of True Positives count to total True Positive count made by the model. Precision = TP/(TP+FP)

Recall Recall is defined as the ratio of True Positives count to the total Actual Positive count. Recall = TP/(TP+FN) Recall is also called "True Positive Rate" or "sensitivity".

F1 score The F1-score of a classification model is calculated as follows: 2*(P*R)/(P+R) with P is the precision and R is the recall.

```
In [100]: 1 print(classification_report(y_test, pred))
```

	precision	recall	f1-score	support
0	0.96	1.00	0.98	27
1	0.96	0.96	0.96	27
2	1.00	0.92	0.96	26
3	0.91	1.00	0.95	29
4	1.00	0.81	0.90	27
5	1.00	0.95	0.98	22
6	0.95	1.00	0.98	20
7	0.96	1.00	0.98	24
8	0.86	0.86	0.86	22
9	0.96	1.00	0.98	26
10	0.96	1.00	0.98	23
11	1.00	1.00	1.00	57
12	0.93	0.93	0.93	29
13	1.00	0.88	0.94	26
14	1.00	0.96	0.98	27
15	0.96	0.96	0.96	25
16	0.86	1.00	0.93	19
17	0.88	1.00	0.94	22
18	1.00	1.00	1.00	23
19	0.96	0.96	0.96	23
20	0.96	0.92	0.94	24
21	0.94	0.89	0.92	19
22	0.78	0.95	0.86	22
23	1.00	0.96	0.98	23
24	1.00	0.96	0.98	26
25	0.96	0.96	0.96	26
26	0.97	0.94	0.96	35
27	0.90	0.96	0.93	27
28	0.85	0.94	0.89	18
29	0.91	0.88	0.89	24
30	1.00	0.97	0.98	31
31	1.00	0.91	0.95	34
32	0.96	1.00	0.98	24
33	0.95	0.88	0.91	24
34	1.00	1.00	1.00	21
35	0.93	0.96	0.95	28
36	0.95	0.95	0.95	21
37	0.90	0.97	0.93	29
38	1.00	0.97	0.98	29
39	1.00	0.90	0.95	20
40	0.96	1.00	0.98	24
41	0.89	0.86	0.88	29
micro avg	0.95	0.95	0.95	1082
macro avg	0.95	0.95	0.95	1082
weighted avg	0.95	0.95	0.95	1082
samples avg	0.95	0.95	0.95	1082

Fig. 7. Report of recall, precision and F1 score

We all see that the precision, recall and F1-score is so good. This is partially because we add the regularization term to the cost function. But we want to explore more about the data to see the black magic behind this phenomenon.

1.8 Reduce computational workload

First, i use the PCA to reduce our data from 142 features to 3 features. And it turns out that the accuracy was still rather high, so i tried to visualize the state of the whole data in 3D and this is the result.

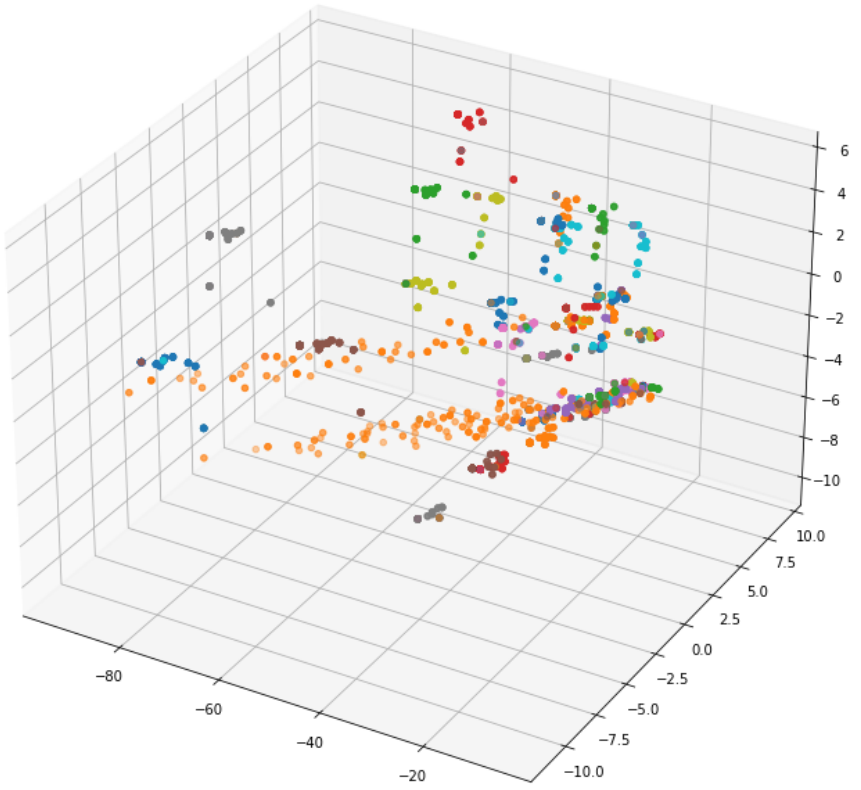

Fig. 8. Projection of the dataset to 3 dimensional space

As you can see about the figure above, the data is separately distributed and the point with the same color (or in the same class) were put together though a lot of information lost after we project our data from 142 features to only 3 main features. The pattern of the dataset mainly depends on some of the bias features.

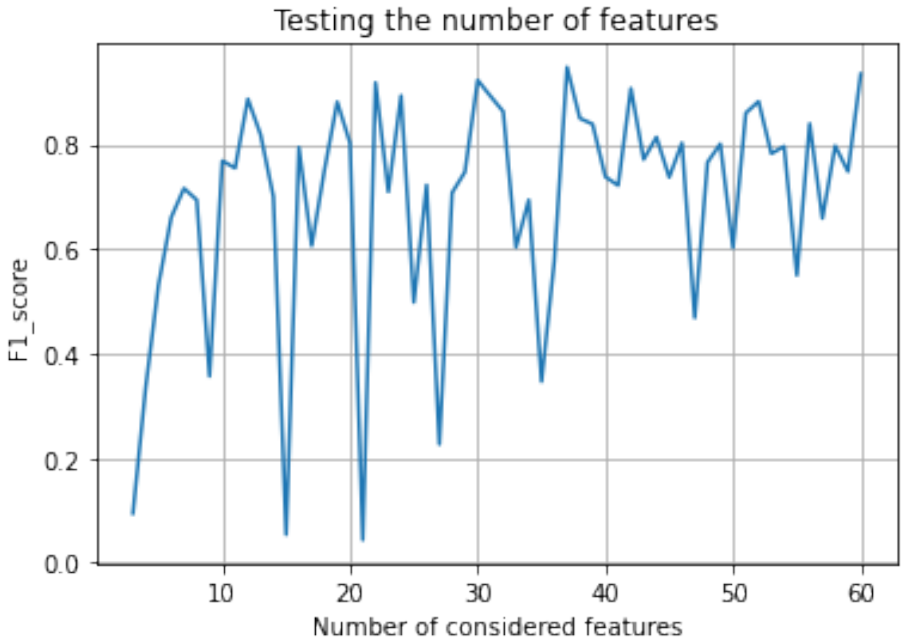

Fig. 9. Dependence of F1 score on number of features

In [113]:

```
input_layer_size = 12

X_train_cv, X_test, y_train_cv, y_test = train_test_split(projectData(X, pca(X)[0], input_layer_size), y, test_size
X_train, X_cross_val, y_train, y_cross_val = train_test_split(X_train_cv, y_train_cv, test_size=0.25)

Theta1 = randInitializeWeights(input_layer_size, hidden_layer_size)
Theta2 = randInitializeWeights(hidden_layer_size, num_labels)

initial_nn_params = np.concatenate([Theta1.ravel(), Theta2.ravel()], axis=0)

costFunction = lambda p: nnCostFunction(p, input_layer_size,
                                        hidden_layer_size,
                                        num_labels, X_train_cv, y_train_cv, lambda_)

res = optimize.minimize(costFunction,
                        initial_nn_params,
                        jac=True,
                        method='TNC',
                        options=options)

nn_params = res.x

Theta1 = np.reshape(nn_params[:hidden_layer_size * (input_layer_size + 1)],
                    (hidden_layer_size, (input_layer_size + 1)))

Theta2 = np.reshape(nn_params[(hidden_layer_size * (input_layer_size + 1)):],
                    (num_labels, (hidden_layer_size + 1)))
pred = utils.predict(Theta1, Theta2, X_test)
accuracy = np.mean(pred == y_test) * 100
print(accuracy)
```

```
99.6171111697914
```

Fig. 10. Accuracy in test set with 12 feature

From the figure, we can conclude that increase the number of feature will not help. A large enough features such as 12 can not only reduce a lot of computational workload but also keep the F1 score high enough.

```
In [114]: 1 print(classification_report(y_test, pred))
```

	precision	recall	f1-score	support
0	1.00	0.90	0.95	21
1	0.95	0.95	0.95	22
2	0.74	1.00	0.85	29
3	0.91	0.95	0.93	21
4	0.82	0.88	0.85	32
5	0.77	0.96	0.86	28
6	1.00	0.94	0.97	18
7	0.86	1.00	0.92	18
8	0.89	0.93	0.91	27
9	0.65	0.95	0.77	21
10	0.97	1.00	0.98	28
11	0.95	0.93	0.94	44
12	1.00	1.00	1.00	21
13	1.00	0.88	0.94	34
14	0.96	1.00	0.98	27
15	0.96	0.93	0.94	27
16	0.82	0.74	0.78	19
17	0.97	0.97	0.97	32
18	0.91	0.94	0.92	32
19	0.97	0.97	0.97	32
20	1.00	1.00	1.00	30
21	0.92	0.92	0.92	25
22	0.79	0.65	0.71	23
23	0.95	0.91	0.93	22
24	0.89	0.84	0.86	19
25	1.00	0.95	0.97	20
26	0.82	0.93	0.87	15
27	0.93	0.93	0.93	27
28	1.00	0.81	0.90	37
29	1.00	0.86	0.93	29
30	0.95	0.86	0.90	22
31	1.00	0.87	0.93	23
32	1.00	0.92	0.96	24
33	0.92	0.82	0.87	28
34	0.96	0.93	0.95	29
35	0.87	0.90	0.88	29
36	0.96	0.81	0.88	27
37	0.96	0.96	0.96	27
38	1.00	1.00	1.00	25
39	0.87	0.95	0.91	21
40	1.00	1.00	1.00	28
41	0.86	0.95	0.90	19
micro avg	0.92	0.92	0.92	1082
macro avg	0.92	0.92	0.92	1082
weighted avg	0.93	0.92	0.92	1082
samples avg	0.92	0.92	0.92	1082

Fig. 11. Classification report of 12 features