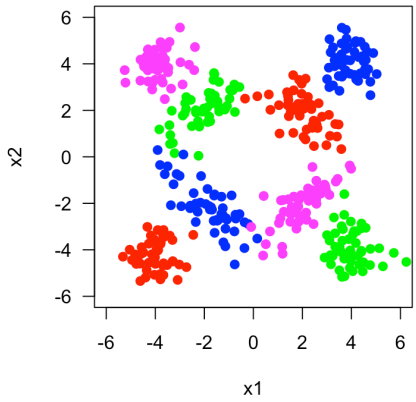
Elif Küçük
0040851
COMP 421
HW 03 Report.

COMP/INDR 421/521 HW 03: Multiclass Multilayer Perceptron – Report

This file briefly explains each function and variable defined in the project. The defined function names and variables are written in bold. The code also has the comment lines to make understating easier.

class_means and **class_covs** matrices store the class means and covariances associated with all four classes. Using these mean and covariance matrices we generate the data points using **mvrnorm** function and store the data in a list called **pointList**.

You can see the generated data using the seed 221 below.



y_binary matrix stores the corresponding true class values for each data point in binary fashion. Therefore it's 2 dimensional, N*K matrix.

We learned the multilayer perceptron using the following functions.

sigmoid(o) function returns the sigmoid transformation of given input.

$$Z = \frac{1}{(1 + \exp(c(1, X) * W))}$$

X is matrix of size (N*D) and W is matrix of size ((D+1)*H) in return Z is matrix of size (N*H).

We define **softmax** function which takes a column vector input and returns the softmax of each input stored in a column vector. So each element in the returning element was calculated in the following fashion.

$$y_k = \frac{\exp(z_{th} * v_{hk})}{1 + \sum_i^K (z_{th} * v_{hi})}$$

Elif Küçük
0040851
COMP 421
HW 03 Report.

$$where\ k = 1,2,3,4\ (class\ type), K = 4, t = 1 \ldots .400\ (sample\ point)\ h = 1 \ldots .20\ (H) hidden\ node$$

softmax function returns softmax of each data point one by one. **softmax_ALL** function in return calculates softmax of all data points.

gradient_v function for each given single input y_predicted, y_truth and z evaluates how much change should be applied to v weights.

$$\boldsymbol{delta\ v} = eta * \ c(1, z_t) * (ytruth_t - ypredicted_t)^T\ where:$$
$$\boldsymbol{v} = (H+1) * K\ matrix$$
$$\boldsymbol{z} = H * 1$$
$$\boldsymbol{ytruth}\ \&\ \boldsymbol{ypredicted} = (1 * K)$$

gradient_w function for each given single input y_predicted, y_truth, z and x evaluates how much change should be applied to w weights.

$$\boldsymbol{delta\ w} = eta * \sum_{h=1}^{H}((v_h)^T\ * (ytruth_t - ypredicted_t)^T) * (z^t * (1 - z^t)) *\ x^t\ \ where:$$
$$\boldsymbol{w} = (D+1) * H\ matrix$$
$$\boldsymbol{z} = H * 1$$
$$\boldsymbol{ytruth}\ \&\ \boldsymbol{ypredicted} = (1 * K)$$
$$\boldsymbol{x} = D * 1$$

In order the detect how much error we are getting I define error function calculator called as **objective_val**(y_truth, y_predicted).

$$Error = \ -\sum_{t=1}^{N=400}\sum_{k=1}^{K=4} y\_predicted_{tk} * \log\left(y\_truth_{tk}\right)$$

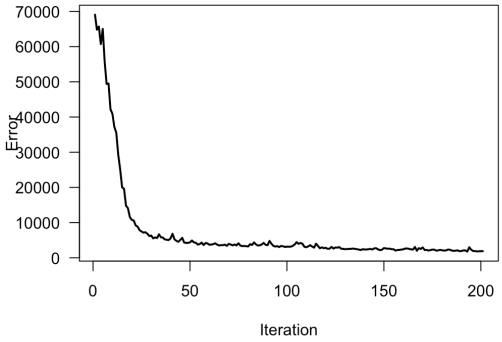
We initialize w and w0 matrices using runif function around 0. And calculate first values of Z and y_pred_binary. We start iteration by 1 and calculate the first error using the **objective_val** function.

In order to find the correct weights for each attribute of each class type

1. We start a while loop in which we use online learning and find w and v for corresponding iteration.
2. In online learning one by one we give each input x to update v and w.
3. After completing giving the inputs and updating the weights. We calculate the objective value.
4. And check if change in objective value is less than predefined epsilon.
5. If it's we return from the loop and we terminate the learning of w and v weights.
6. If not we increase the iteration and return to the beginning of loop.

Elif Küçük
0040851
COMP 421
HW 03 Report.

In the end plotting our objective function gives the following diagram.



In order to see how well we did in our estimation we create a confusion table. I create the **confusion_table** after transforming y_pred_binary and y_binary to y_predicted and y column vectors which has values of 1 : 4.

```
> print(confusion_table)
            y
y_predicted   1   2   3   4
          1 100   0   0   0
          2   0 100   1   0
          3   0   0  99   2
          4   0   0   0  98
>
```

In order to draw decision boundaries I define an interval and set of x1 and x2 points. To find the class values corresponding to each x1 and x2 points I define a function called **predict_plot**(x1, x2) which returns a value between 1 and 4 (the class type) after estimating the y_predicted using the learned parameters w and v.

Here is thre graph with the decision boundaries.