**ENGR421 – HW3 - Discrimination by Regression**

Melis Oktayoğlu / 0064388

I first, set the class means, covariances and sizes as was stated on the description of the homework. Then, using my school id as a seed I randomly generated the datapoints of each class using the *multivariate\_normal* function of the numpy library. For easy use I also concatenated all points as X, then I plotted these points with class 1, 2 and 3 respectively being red, green, and blue.

Then I created the labels of these classes by the help of np.repeat. Since class\_sizes array is of size 3 I iterated over it to create the labels using the class\_sizes array for repetition numbers. I also turned the labels into one-of-K encoding which creates a matrix of dimensions *(number of datapoints, number of classes)* and each column represents which datapoint (aka which row) belongs to which class (represented by columns). Thereby this matrix only has 1s and 0s, with 0s representing that datapoint not belonging to that class, and each row by-definition only has one instance of 1.

After this, I defined the sigmoid function. That is 1/(1+exp[-WTX + w0]). Since this is a multiclass problem and each row in W represent the values for each class matrix multiplication of X and W corresponds to multiplying each datapoint with the W values and summing and storing them at the respective class column. The resulting WTX equivalent matrix is of size *(number of datapoints, number of classes).* After I also added the result with the w0 array that is of size (*1* *, number of classes).* By the addition of numpy arrays this also adds the w0 array to all rows. Further, since the denominator of the sigmoid function contains an exponential expression, I found the max value and replicated that by each column (so that it is subtracted from each value in a row) and subtracted from the WTX + w0 value. This prevents overflow in the exponential in the case that these terms may go up to thousands. Finally, I returned the expression 1/(1+exp[-WTX + w0]).

Then, since the sigmoid outcome would be the probabilities of each class with each column holding these values, I needed to get the maximum value from each sigmoid score row. To this end I first wrote the *get\_max\_class(score)* function which gets a row of score that corresponds to a distinct datapoint and obtained the maximum score of the row by *np.max*. Then I searched this max value in the score row, then I saved the index it appeared which then I used to set that index to one in an array of zeros (length 3 for each class, to replicate one-of-K encoding). This function is called for each datapoint in a separate function called *generate\_y\_hats(scores)* which simply calls the *get\_max\_class(score)* function for each datapoint and concatenates the results.

Following this step, I wrote the gradient functions. To derive the terms of the ΔW and Δw0 I used the chain rule for taking the derivative of 0.5 ∑ ∑ (𝑦 – 𝑦\_hat)2.

*(See next page for derivation)*

Text, letter

Description automatically generated

Most left-hand term is dError/dw0c

To do these calculations for the W and w0 gradients, I did these calculations respectively for each class and returned them as (2,3) and (1,3) sized numpy arrays respectively.

Then, to do the discrimination by regression I used the gradient descent and set the W and w0 to some small values in uniform distribution and set the learning parameters as said. Then with trail and error I found that 100 is a good iteration number, if I would let it to a while loop ending with the Euclidian distance smaller than epsilon, I couldn’t end the loop, so I choose to opt for this for loop. In each iteration the class predictions are repreated with the updated W and w0 and the new error is stored in *objective\_values* than the updates are done to W and w0 using their gradient functions and the learning rate eta.

Then to confirm that indeed the error is minimized I plotted the errors across each iteration and indeed they converged to a negligible number.

To construct the confusion matrix, I first converted the one-k-encoded class predictions to their actual [1,3] labels by iterating through each datapoint. After, with the pd.crosstab I constructed the confusion matrix.

Finally, I plotted the given datapoints again on a plot. Then to draw the discriminant lines on the plot, I some linear values between the x1 and x2 range and created a matrix by the lengths of these two ranges and the third dimension as the class numbers. The first two dimensions indicate the datapoints on the space, and the third dimension the class label. In a for loop, for each class I multiplied the final W values with the X1 and X2 values and added the final w0 value of the class, this basically results in continuous values that make up the linear discriminant line. After adding the discriminant lines on the plot, I circled the wrong predictions.