AI HW5

Joshua Wiseman

October 2024

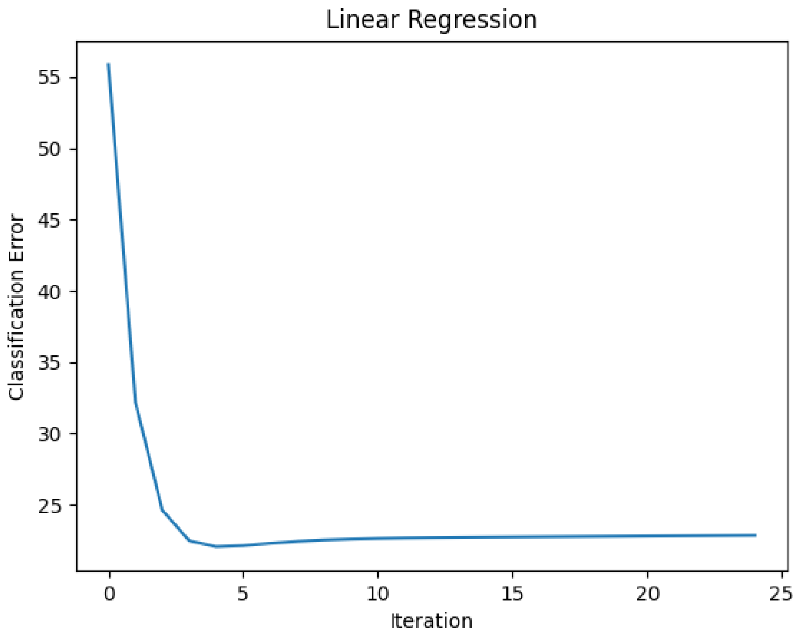

Figure 1: Linear Regression MSE Over Time

For the update function used the sum of the errors times x times some small number alpha plus w. I did use dot product instead of for loops, but it's the same function as used in the video. That was the main method for updating the w values and then the MSE for evaluating the results. Of which we can see the results decreasing the error term with each given iteration starting from an MSE of 130 to an MSE of around 20 in around 5 iterations.

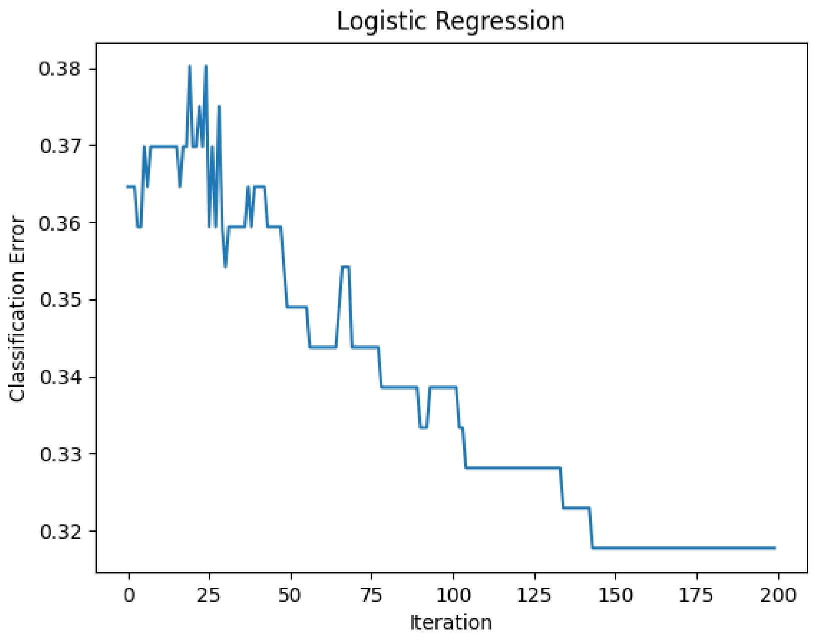

Figure 2: Logistic Regression Error Rate Over Time

For the update function, I used the method given in class. The only thing that could be different here is that i'm using numpy dot product instead of for loops to update the weights. We can see that the error of the approximated function drops from around 38% to 32% & then takes a around 150 iterations to finally hit the lowest error rate (32%) without any other methods (like normalization).

For this algorithm implementation of K-Means I used numpy for the computation of euclidean distance. In this case, being in 100D, which saves time without a python for loop. Along with other additions to simply code output and time as well (like mp.mean). There is also a threshold to ignore super small changes to not go through all iterations, so that is more of an optimization change that doesn't affect the core algorithm.

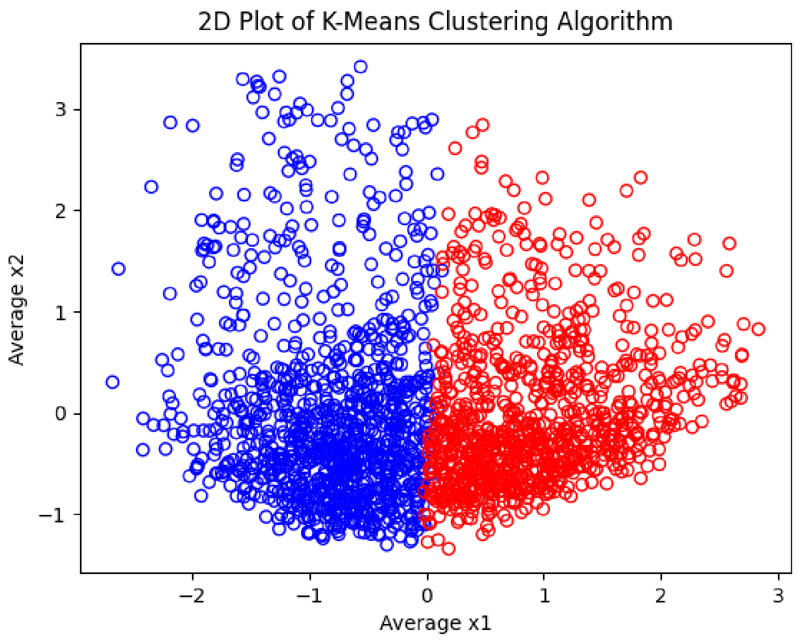

Figure 3: k=2 2D Scatter Plot

For k=2 in 2D, we can see a divide between the two groups from the middle and this sort of grouping seems to be fine, but seems like a higher k value could be more beneficial.

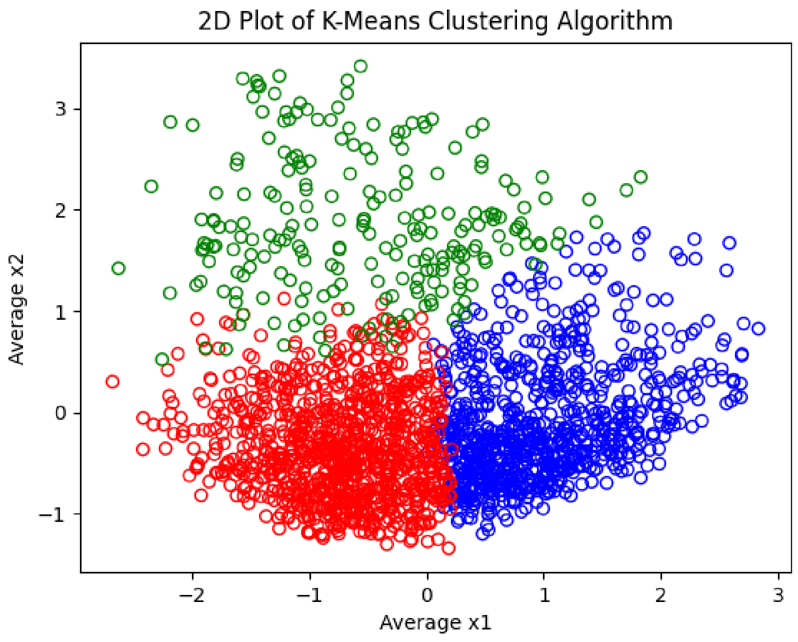

Figure 4: k=3 2D Scatter Plot

For k=3 in 2D, we can see a divide between the three groups, one in the middle like before but also one for the top section which is more scattered out and seems that it does a better job at describing the feature distribution than k=2.

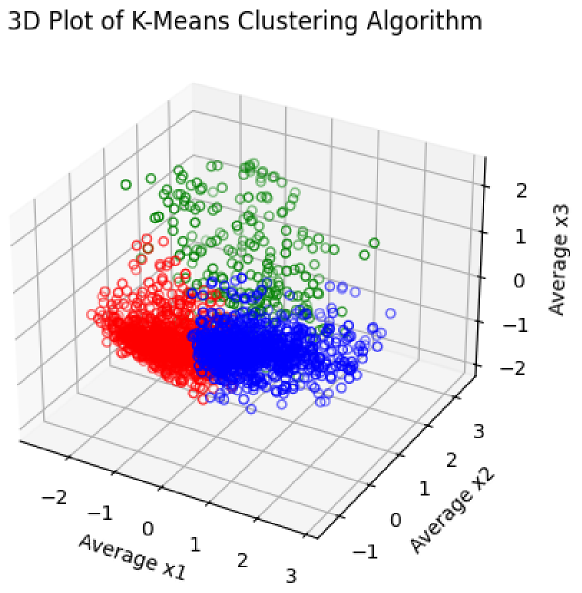

Figure 5: k=3 3D Scatter Plot

For k=3 in 3D, we can see that the green values also have a higher x3 axis value than we could previously see in the 2D plot. So we can further back our assumption that k=3 is more informative than a k=2 plot would be because we wouldn't have gotten much more information from that extra axis in the k=2 case.

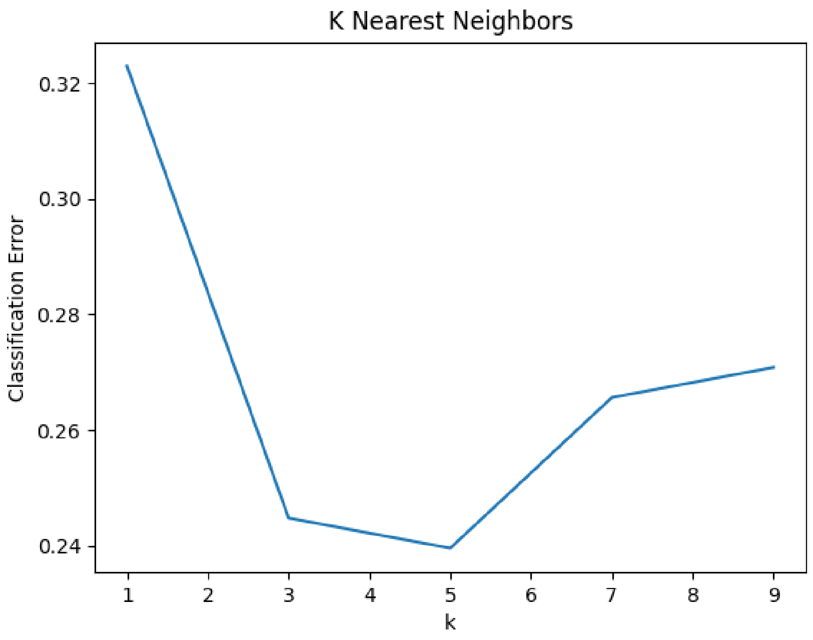

Figure 6: K Nearest Neighbors Error Rate Over Time

The algorithm above uses numpy for finding the closest values, along with a distance library to easily calculate distance. Overall this k nearest neighbors was way easier to implement in comparison to k means mostly due to the lack of centroids.

At k=1 and k=2 we can tell that the error is to large and likely an overfit because we are finding the immediate values closest to the point. By k=3 this improves and by k=5 we get our lowest error. When k>5 the error seems to increase again lead me to assume that the model is now more on the underfitting side (average to the whole data set is not useful for example).

Decision Tree THEORY TASK

Student ID	F1	F2	F3	F4	F5	Grade (A/B)
01	1	1	0	0	1	B
02	1	0	0	1	1	B
03	1	0	1	0	0	B
04	0	0	1	1	0	A
05	1	1	0	1	0	B
06	0	1	0	1	1	A
07	0	1	1	1	1	B
08	1	1	0	0	1	B
09	1	1	1	0	1	A
10	1	0	1	0	0	A

I(p(v_1), p(v_2), ...) $= \sum_{i=1}^{k} -p(v_i) * log_2 p(v_i)$

Rem(A) $= \sum_{i=1}^{d} \frac{p_i+n_i}{p+n} I(\frac{p_i}{p_i+n_i}, \frac{n_i}{p_i+n_i})$

Where p_i and n_i are number of positive/negative examples in E_d and p, n are the number of positive negative before the split.

We must find the lowest remainder to get the highest gain (Information Gain method).

Rem(F1) $= \frac{7}{10} I(\frac{5}{7}, \frac{2}{7}) + \frac{3}{10} I(\frac{2}{3}, \frac{1}{3})$
$= \frac{7}{10}(-\frac{5}{7} * log_2 \frac{5}{7} - \frac{2}{7} * log_2 \frac{2}{7}) + \frac{3}{10}(-\frac{2}{3} * log_2 \frac{2}{3} - \frac{1}{3} * log_2 \frac{1}{3}) = 0.87967$
Rem(F2) $= \frac{6}{10} I(\frac{4}{6}, \frac{2}{6}) + \frac{4}{10} I(\frac{2}{4}, \frac{2}{4})$
$= \frac{6}{10}(-\frac{4}{6} * log_2 \frac{4}{6} - \frac{2}{6} * log_2 \frac{2}{6}) + \frac{4}{10}(-\frac{2}{4} * log_2 \frac{2}{4} - \frac{2}{4} * log_2 \frac{2}{4}) = 0.95097$
Rem(F3) $= \frac{5}{10} I(\frac{3}{5}, \frac{2}{5}) + \frac{5}{10} I(\frac{4}{5}, \frac{1}{5})$
$= \frac{5}{10}(-\frac{3}{5} * log_2 \frac{3}{5} - \frac{2}{5} * log_2 \frac{2}{5}) + \frac{5}{10}(-\frac{4}{5} * log_2 \frac{4}{5} - \frac{1}{5} * log_2 \frac{1}{5}) = 0.84643$
Rem(F4) $= \frac{5}{10} I(\frac{3}{5}, \frac{2}{5}) + \frac{5}{10} I(\frac{3}{5}, \frac{2}{5})$
$= \frac{5}{10}(-\frac{3}{5} * log_2 \frac{3}{5} - \frac{2}{5} * log_2 \frac{2}{5}) + \frac{5}{10}(-\frac{3}{5} * log_2 \frac{3}{5} - \frac{2}{5} * log_2 \frac{2}{5}) = 0.97095$
Rem(F5) $= \frac{6}{10} I(\frac{4}{6}, \frac{2}{6}) + \frac{4}{10} I(\frac{2}{4}, \frac{2}{4})$
$= \frac{6}{10}(-\frac{4}{6} * log_2 \frac{4}{6} - \frac{2}{6} * log_2 \frac{2}{6}) + \frac{4}{10}(-\frac{2}{4} * log_2 \frac{2}{4} - \frac{2}{4} * log_2 \frac{2}{4}) = 0.95097$

The lowest number here is 0.84643, so we will now split from the **F3 feature**.

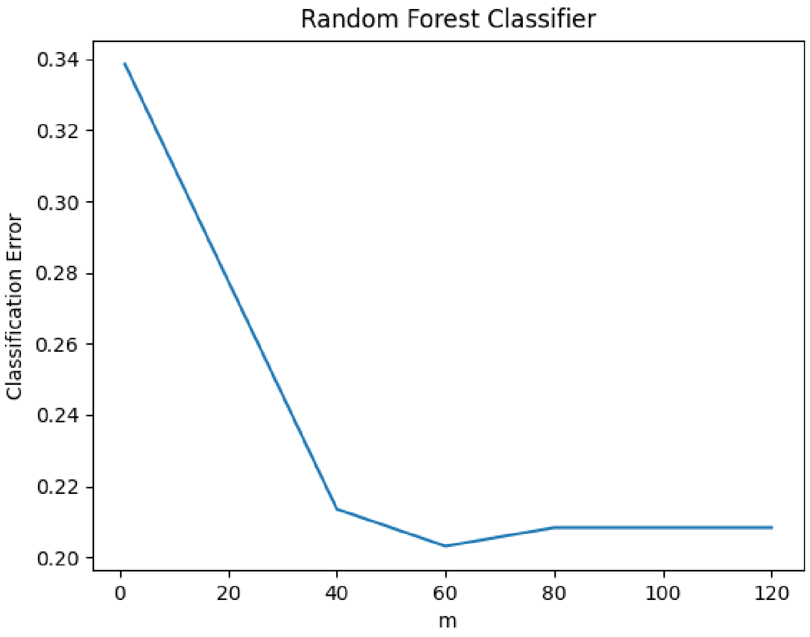

Figure 7: Random Forest Classifier Error Rate Over Time

At the beginning ($m < 10$), the error is very larger. However, it seems that the error goes down from m=1 to around m=60 and then from there it seems to increase a little bit when m>60 (bounce back). So it seems that the idea value of m here is around m=60 for the lowest error term.

	Basic Logistic Regression	Smote Method
Group 1 Error	38.8%	27.7%
Group 2 Error	29%	27.4%
Error Gap	9.8%	0.3%

Table 1: Class Error & Method Implementation

Here we can see that there is around a 10% gap between the male error (Group 1, 38% error) vs female error (Group 2, 29% error). In the basic logistic regression. However in my method (Implement SMOTE with logistic regression) the error of both classes drops to 27% and the gap error shrinks to 0.3% which gives us not just an equal prediction for both classes, but an equal chance that the prediction is wrong too, which will reduce any unfairness going on compared to the previous implementation.