CSC3060 AIDA – Assignment 1

Fionn Mc Conville

40156103

16/11/18

Introduction

In this report I will attempt to demonstrate and describe the differences in handwritten symbols, these symbols are in three different sub-groups; digits, letters and mathematical symbols. In my report below I will detail how I created the dataset of images (section 1), the feature engineering I performed on the dataset, (section 2) and finally the statistical analysis I performed on the feature data which enabled me to single out features which are statistically significant in discriminating between the subset groups and within the subset groups (section 3).

Section 1: Creating the dataset

I created the handwritten symbol images using the touchscreen computers in the CSB using the application software GIMP. These images were saved as PGM files, type ASCII, with the appropriate file labelling as requested in the assignment instructions. In the code for section 1 I then converted these files to binary matrices in a csv file format using python. To do this I started by creating a function called convertToBinary() which took a PGM file as an argument and converted it to binary – (with 0s being white pixels/whitespace and 1s being black pixels) then outside that function I created a for loop that cycled through the directory containing each csv file and performed the convertToBinary() function on it. The code within the loop then converted these files to a numpy array, split the array into a 20x20 matrix then converted it to a dataframe where it was then converted to a csv file and placed in a local directory as specified by the assignment instructions. I was going to convert the files straight from numpy array to csv however this proved unreliable and so by converting the numpy arrays to dataframes first I found the results more agreeable.

Section 2: Feature Engineering

Again, for this section, I used python to engineer the features for this assignment. I began by loading in the csv files I had created earlier in section 1 as numpy arrays so they would be easily processed by the methods I created to calculate the features of each image. The first two methods of label and index are simply done by reading the filenames and splitting the strings where appropriate.

Section 2.1: Features counting black pixels

This section of the report is aptly named “counting black pixels because that is largely what the next several features do, just in a different variety of ways. The methods are largely all similar though with just small variations within them according to their desired functionality.

The first method **nr_pix** – which simply counted black pixels –totalled the black pixels using a counter. I achieved this by using a nested for loop – (for row in array: for item in row) that cycled through each element/pixel in the numpy array and incrementing the counter each time a 1 occurred.

A likewise method was used for the **height** feature – (which counted the number of rows containing at least one black pixel) however it just tested if 1 was in row. This worked well because in the nested for loop, when going through "for row **in** array", takes the row as a separate array and python has an easy condition "in" that can return true if it identifies the element you are seeking in the for condition.

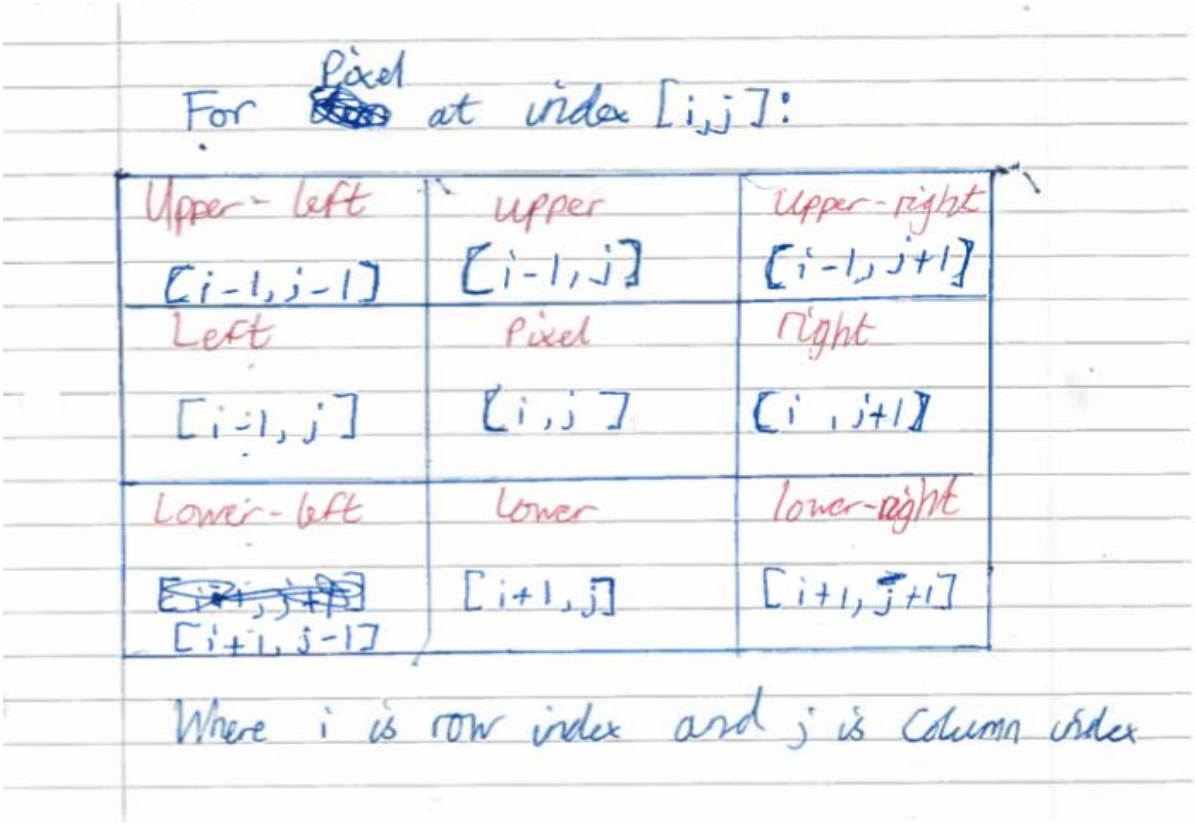
The **width** feature – (counting the number of columns containing at least 1 black pixel) was just a repeat of the height function, but it transposes the array so that columns are rows and rows are columns, in other words an element at say (5, 4) in a matrix is now at (4, 5). The width feature is just an inverse of the height feature and so by inversing/transposing the numpy array I can use the methods which are successful in calculating row-based features and use them on similar column-based features. This will come in very useful later in the assignment. The code to calculate **tallness** – (height/width) is self-explanatory.

The **rows_with_1** feature is calculated largely with a rehash of the code in the nr_pix method but it has two counters, one for black_pixels and another for black_pixels_in_row – this counter is reinitialised to 0 every iteration of the first for loop – reset to 0 for each row. Within each iteration of the first for loop it checks with an if statement whether black_pixels_in_row is equal to 1, if it is equal to 1 it increments the black_pixels counter which is the eventual return value. The **cols_with_1** feature has code exactly like this, but its difference is that the matrix is transposed – which works as described earlier.

The **rows_with_5** feature and **cols_with_5** feature are calculated pretty similarly to their counterparts described above, however the if statement checks whether the black_pixels_in_row or black_pixels_in_col counters are greater than or equal to 5 before they increment the return counter.

Section 2.2: Features counting neighbouring pixels

This section focuses on how I created methods to obtain values for features dealing with neighbouring pixels. I started by trying first to determine what exactly constitutes a neighbouring pixel and how do I calculate it within the confines of a for loop. I drew the following diagram to make sense of it:



Therefore, from the outset I knew what constituted a neighbouring pixel in my dataset, I began by using an enumerated for loop, (for rowi, row in enumerate(array)) and nested within that a likewise enumerated column for loop that would allow me to log not only the row/column – but also their indexes (rowi, coli) respective to the overall array. Within the **one_neighbour** function, I began by logging each neighbouring pixels using the code below within each iteration:

```
# neighbours = np.array([array[rowi - 1][index - 1], array[rowi][index - 1],array[row + 1][index - 1],\
# array[row - 1][index], array[row + 1][index], array[row - 1][index + 1],   \
# array[row][index + 1], array[row + 1][index + 1]])
```

However this proved problematic as it threw up an IndexOutOfBounds exception every time the function was attempted, as the for loop obviously starts iterating at [0,0] in the numpy array we are passing as an argument and so indexes like [-1,-1] do not exist and throw up an exception. I attempted to resolve this by using a try catch block over the above code however this again proved problematic as it would skip over the entire pixel if that exception was thrown for any one of the neighbouring pixels and so any black pixel which had an index near the edges of the image would not be counted in the feature.

My overall solution to this obstacle was to create a function that would return a numpy array containing the values of each neighbouring index in the array. As an argument, it took a row index and a column index as well as the overall array., below is the function alongside descriptive comments:

```
#The function below returns all of the neighbours of an element of index rowi, coli in arr array and returns it as a numpy array
def get_neighbours(rowi, coli, array):
    neighbours = []
    for i in range(-1, 2):
        for j in range(-1, 2):
            if i == 0 and j == 0: # Make sure to skip over the actual item itself
                continue
            try:
                neighbours.append(array[rowi + i][coli + j])
# We iterate through all elements in array arr surrounding arr[rowi][coli] here, the for loop starts at arr[rowi - 1][coli - 1]
# (upper-left) and continues on to arr[rowi - 1][coli + 0] and so on...
# We add these elements to the empty list of neighbours which we created at the start of the function
            except IndexError:
                pass
# We make sure to catch instances of index errors because at the edges of our image matrix IndexError exceptions will be thrown
    return np.array(neighbours)
```

As we can see, the function iterates over the rows directly before and after as well as the columns directly before and after the index passed as an argument. The nested for loop also makes sure that it passes over the index itself with a continue statement for when the index is equal to array[rowi[[coli]. Finally, the function resolves the earlier issue of an IndexOutOfBounds exception by executing a pass on that index should an exception arise.

The lengthy description of this function is necessary because the code within it is vital to the workings of the next few functions calculating features. For instance, it is used to simplify the method of **one_neighbour** which is calculated like so: within each nested for loop if the sum of the neighbours numpy array returned by the get_neighbours function is equal to 1 then the counter - which is the eventual return value - is incremented by 1. The **three_or_more_neighbours** function is almost identical however it checks if the sum of the neighbours numpy array is greater than or equal to 3.

In both of these functions and the following functions within this section, at the start of each for loop the index item must equal to 1 (a black pixel) otherwise none of the functionality described above is performed on it as a continue statement makes us go to the next iteration of the for loop if the condition of the index item being equal to zero is met

The **no_neighbours_below** function takes the code from the get_neighbours function and alters it slightly within it's nested for loop by making it only iterate through the rows below it, on the following page is a snippet of the code to demonstrate:

```
def no_neighbours_below(array):
    no_neighbours_below = 0
    for rowi, row in enumerate(array):
        for coli, item in enumerate(row):
            if item == 0:
                continue
#This below block of code is basically a copy of the get_neghbours function however we only count rows in range 1, 2 (which means
#only count the rows + 1 in our for loop). The rows + 1 in relation to our pixel are the rows below it in the nparray matrix
            neighbours = []
            for i in range(1, 2): #rows are i
                for j in range(-1, 2): # columns are j
                    try:
                        neighbours.append(array[rowi + i][coli + j])
                    except IndexError:
                        pass
            neighbours = np.array(neighbours)

            if np.sum(neighbours) == 0:
                no_neighbours_below += 1
    return no_neighbours_below
```

Underlined here is the area of interest, we only iterate through the row +1 of our neighbouring pixels, the columnar iterations stay the same. Again, a neighbours array is calculated for each iteration of the for loop and if it's sum is equal to zero then the counter - which is the eventual return value – is incremented.

The next few functions are almost identical in functionality, but they all vary slightly on what rows/columns they iterate through depending on their desired purpose; **no_neighbours_above** only checks through rows -1 (i + 1) the index, **no_neighbours_after** only checks through columns + 1 (j + 1) the index, and **no_neighbours_before** only checks through columns -1 (j – 1) the index.

Section 2.2: Features searching neighbouring pixels

For the calculation of nr_regions in this section I employed a recursive search to cycle through the neighbours of each connected black pixel and with each iteration of the search mark the pixel visited so that it was not mistakenly searched again. I used the Boolean visited in my code to mark if the index in question was visited, below is my code for searching the neighbours of each black pixel:

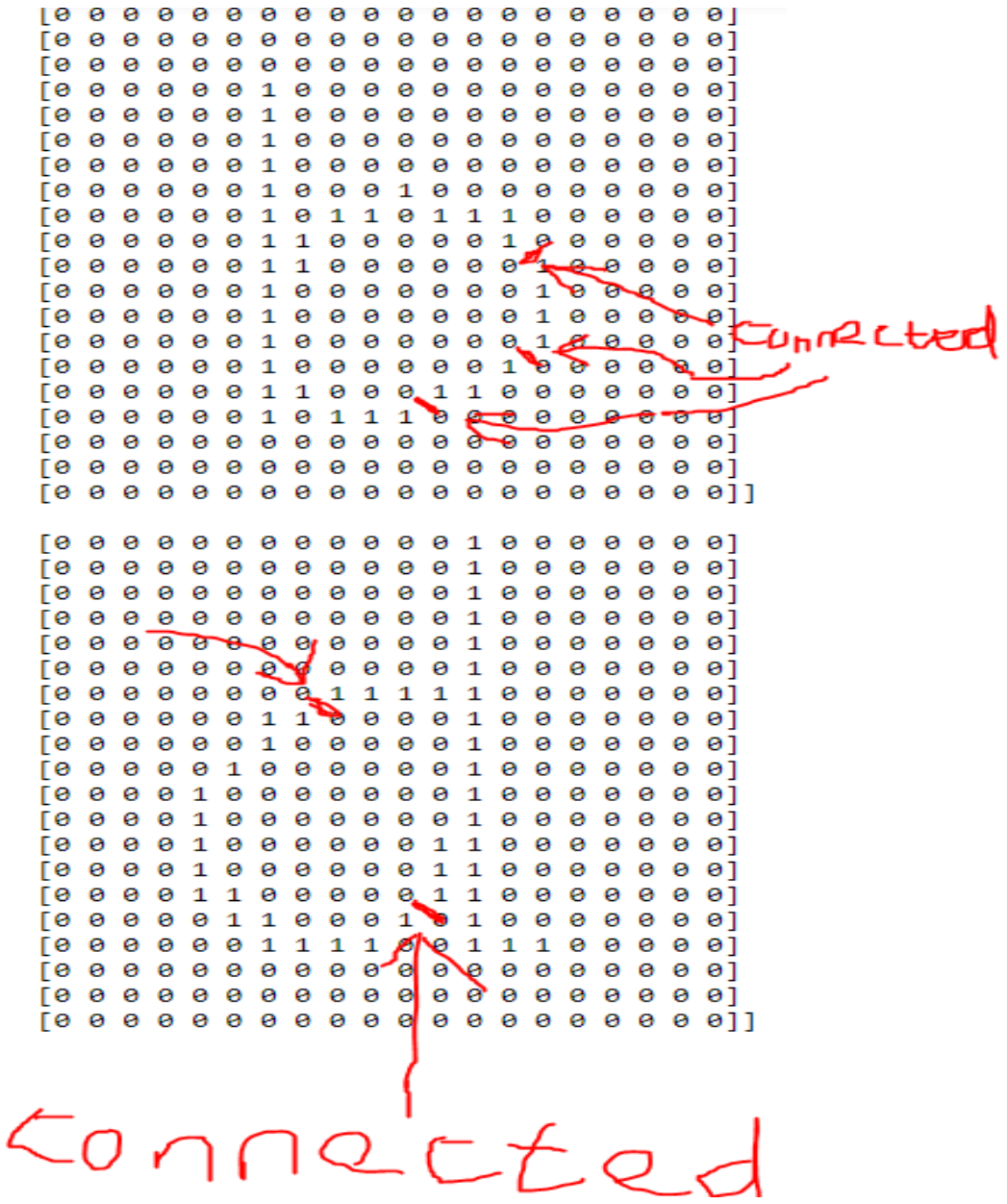
```
def searchNeighbours(array, i, j, visited):

        visited[i][j] = True
        for x in range(-1, 2):
            for y in range(-1, 2):
                if x == 0 and y == 0: # Make sure to skip over the actual item itself
                    continue
                if -1 < i + x < 20 and -1 < j + y < 20: #keep within index range of the pixel array
                    if not visited[i + x][j + y] and array[i + x][j + y] == 1:
            #Recursively searches each connected black pixel until they've all been visited
                        searchNeighbours(array, i + x, j + y, visited)
                        #print(array[i + x][j + y])

#visited is included in params so it's visited status can be kept track of over recursions

#visited =  [[False for j in range(20)]for i in range(20)]
#searchNeighbours(approx, 6, 10, visited)
```

This function was used in the **nr_regions** code, as the recursive nature of the function meant that it only broke out of it's recursion when all connected black pixels were searched – which is exactly the functionality I required for the nr_regions feature. When this function broke a counter – the eventual return value – was incremented. Below is the function of nr_regions with the important area in question highlighted:

```
def nr_regions(array):
    visited =  [[False for i in range(20)]for j in range(20)]
#Above I create a 2D boolean array equal in size to the pixel array I'm measuring
    count = 0
    for rowi, row in enumerate(array):
        for coli, item in enumerate(row):
            if array[rowi][coli] == 1 and visited[rowi][coli] == False:
                searchNeighbours(array, rowi, coli, visited)
                count +=1
#Every time a new region is found the searchNeighbours function wll trigger,
#this count keeps track of it
    return count
```

The purpose of the **nr_eyes** feature was quite similar to the nr_regions code in that it too was calculating a region – just that the region was not a maximally connected set of black pixels but rather it was a connected region of whitespace. I assumed that the function would transfer over easily, and that I could just substitute the 1's that nr_regions was searching through with 0's.

However, this proved not to be the case, what I found was that the nr_eyes function, if implemented in this way, would never identify a region of whitespace surrounded by black pixels as it would exit the region by sometimes going to a **diagonally connected** white pixel, alongside is an illustration to demonstrate.



With this problem the solution was then easy once the issue was identified, I prevented the function from searching diagonally connected pixels, by slightly altering the searchNeighbours function with one line of code; after the nested for loop I inserted an if statement to check if absolute x was equal to absolute y – this meant that all instances of diagonally connected indexes were excluded from the search as the diagonally connected indexes are at (-1, -1 – upper left, -1, +1 – upper right, +1, -1 – lower left and +1, +1 – lower right) . This was made into a function; searchNeighboursNoDiagonal and it worked seamlessly from there

Section 2.3: Other features and exporting the results to a csv file

Late on in this assignment I decided to try to attempt to create the bd feature function as well as the two improvisational feature functions. I had success with the b_or_d function, with an uncomplicated method of identifying a column which had 7 or more pixels in it, then later on in the function determining if that long vertical line was on the left or right side of the image by seeing which side of 10 (the midway index in a 20x20 matrix) that long vertical line was on. This simple method proved effective when only having to distinguish between a b or a d but I'm not sure how useful it would be other than that.

The other two features I devised late on in the assignment were the identify_isolated_horizontal function and its vertical counterpart. I came up with these functions late, so I was not able to test if they were good for distinguishing symbols statistically, or if they were error prone on a large scale, still I think the idea is a good one and it could have been built upon further given time.

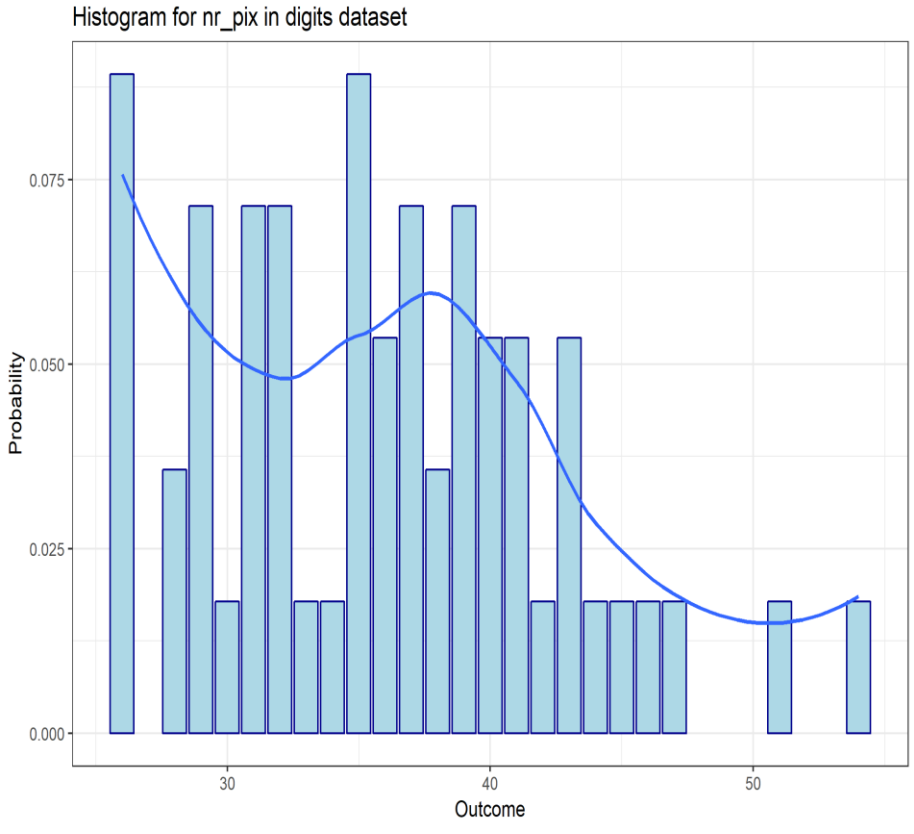
Finally, to put the features in a csv file so that it could be processed later by my R code I created the function calculate_features() which did an Os.walk to iterate through each csv file in my earlier created ..section1_images directory and created a stacked numpy list for each of the 168 images in that directory which it then returned. This returned list was converted to csv and exported to the local section2_features directory as per instructions in the assignment.

Section 3: Statistical analysis of the feature data

For this section, as requested, I used R to statistically analyse the feature data of each handwritten image. A significance value of 0.05 was used, therefore for most of my findings I had a 90% confidence level – this is the case unless specified otherwise. This was used on most occasions, except when doing multiple comparisons, in which a correction was made.

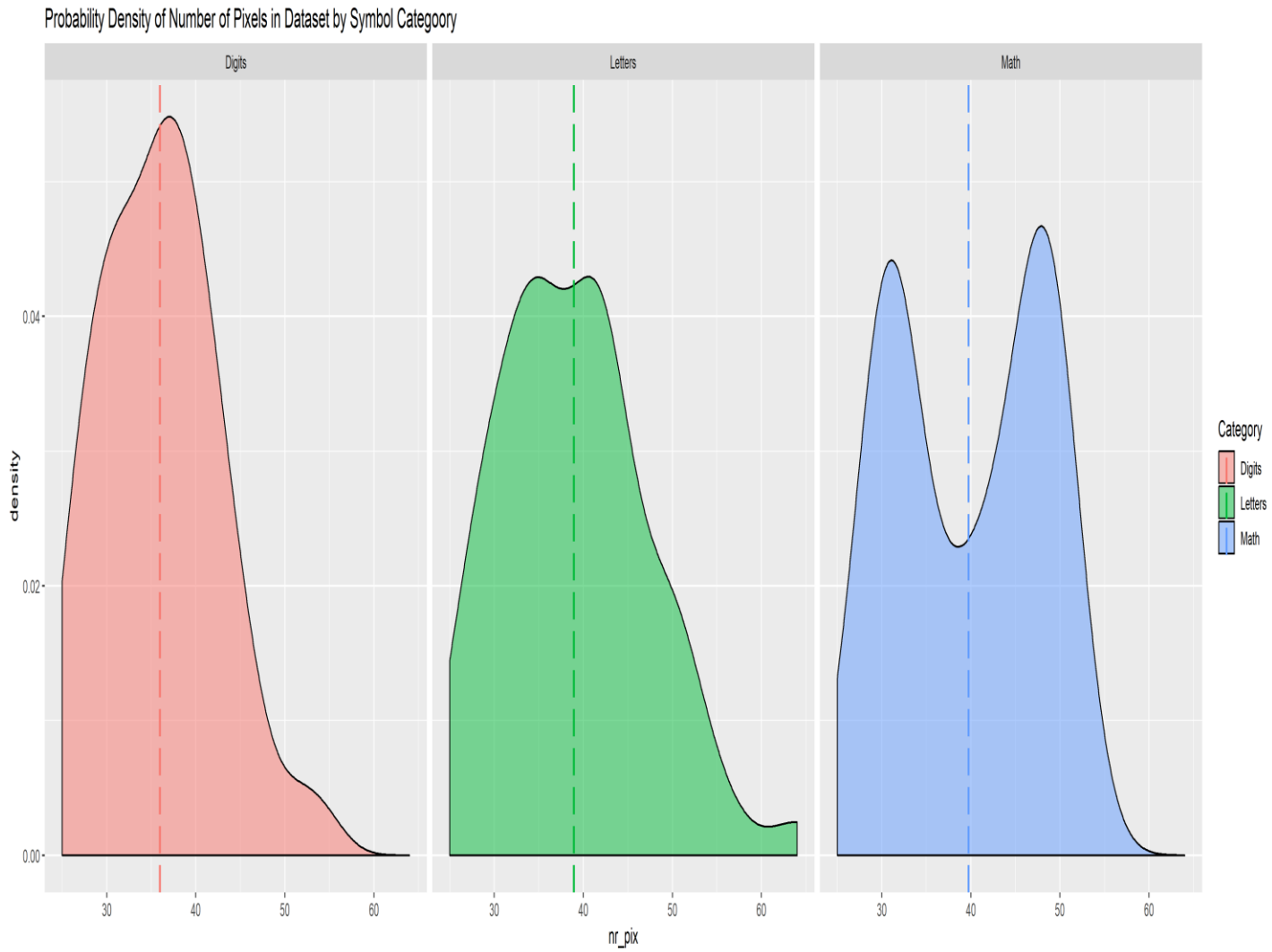
Section 3.1: Probability distribution for the nr_pix feature

Below I will show the probability distribution for each of the three symbol groups and briefly discuss the shape of the distribution for each group, as well having the accompanying probability distribution table below it. Originally, I set out to display the probabilities like so:



Outcome	Probability
26	0.089
28	0.036
29	0.071
30	0.018
31	0.071
32	0.071
33	0.018
34	0.018
35	0.089
36	0.054
37	0.071
38	0.036
39	0.071
40	0.054
41	0.054
42	0.018
43	0.054
44	0.018
45	0.018
46	0.018
47	0.018
51	0.018
54	0.018

However, this looked unkempt and I only was required to give a rough estimation of the probability distribution. With time, my skill in R improved throughout the assignment, so I created the graph below to more cleanly showcase the probability distribution of each symbol group:

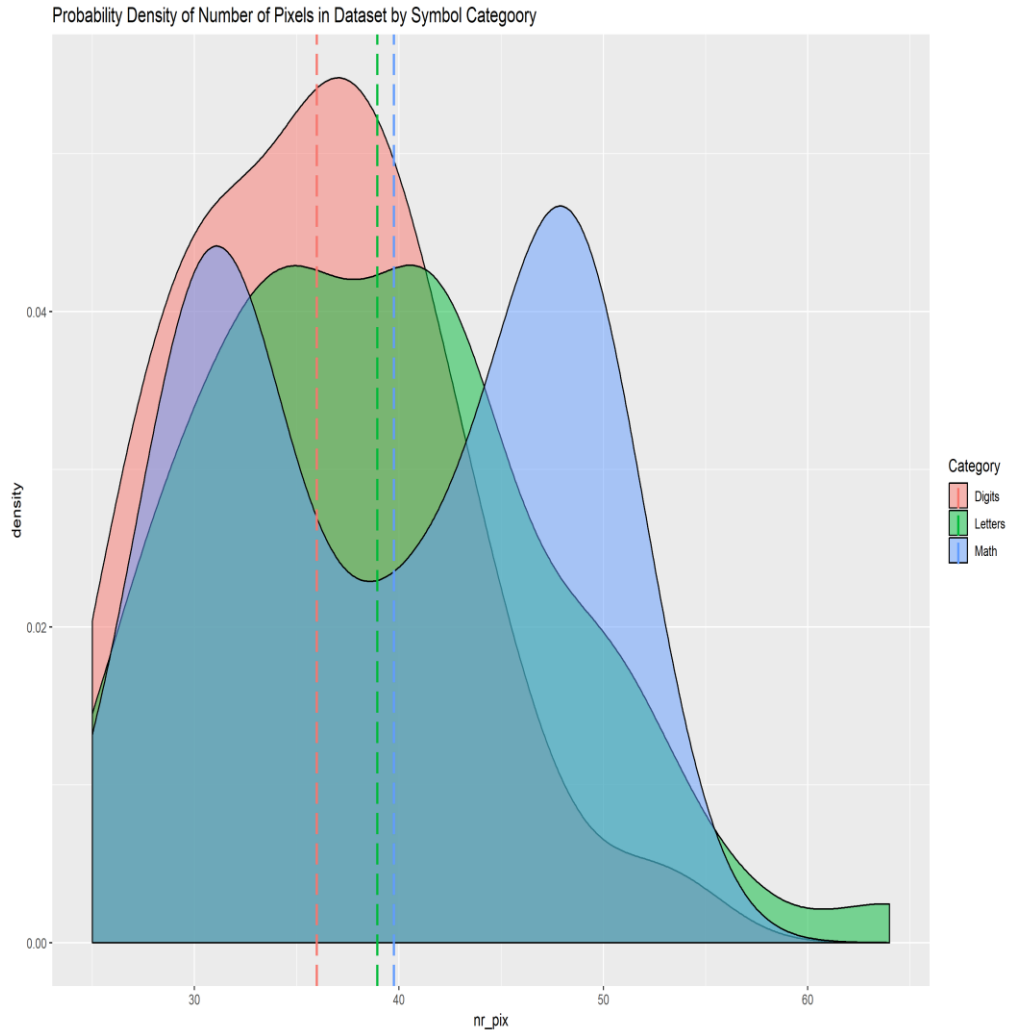


For the digits dataset (in red), the distribution seems right skewed with a mostly normal distribution. It seems pretty unimodal with its peak at the start of the graph. It has a few outliers beyond the 50-pixel mark, this might come from the same symbol, and it could prove useful later when I need to distinguish a digit from the rest of the group. Despite these outliers the distribution seems relatively normal

The letters dataset is unmistakably unimodal, with again, a right skew. Like the digits dataset it has a few outliers, but this time they seem to be beyond the 60 mark, which could prove useful in distinguishing from within that sub-group later.

The math symbols dataset has a bimodal distribution and is symmetric in its distribution, as a result of this it does not seem to have many outliers.

The probability density graph above demonstrate that within my dataset, the feature values for nr_pix are safe to assume normality on and within each of the datasets for the nr_pix feature the data collected is not extremely skewed, which will be useful for later when hypothesis testing is used. Below is the probability distribution for nr_pix on each group within the same axis.



The dashed lines in this graph are the respective means of each dataset and as we can see, the mean number of pixels for the digits dataset is the least, followed by the letters dataset, then closely followed by the math symbols dataset. This graph helps us to compare the difference in feature values for each group and I may use it more throughout this report to display these differences for feature values.

Section 3.2: Probability that a random digit image has greater than 20 pixels

The probability that an image with greater than 20 pixels is 1, or certain. Each image within my digits group of symbols has more than 20 pixels. Alongside is the probability distribution of number of pixels within the digits dataset, (shown before) to demonstrate:

Outcome	Probability
26	0.089
28	0.036
29	0.071
30	0.018
31	0.071
32	0.071
33	0.018
34	0.018
35	0.089
36	0.054
37	0.071
38	0.036
39	0.071
40	0.054
41	0.054
42	0.018
43	0.054
44	0.018
45	0.018
46	0.018
47	0.018
51	0.018
54	0.018

Section 3.3: Summary statistics for all the features

(a): Summary table for all of the features in overall dataset:

	nr_pix	height	width	tallness	rows_with_1	cols_with_1	rows_with_5_plus	cols_with_5_plus	one_neighbour	three_or_more_neighbours	no_neighbours_below	no_neighbours_above	no_neighbours_before	no_neighbours_after	nr_regions	nr_eyes	r5_c5
Mean	38.23	13.95	13.4	1.1	5.22	1.99	1.86	1.17	2.23	12.67	17.07	16.79	7.73	7.88	1.18	0.38	0.7
Stdev	7.83	3.27	2.63	0.37	4.35	1.82	1.04	1.09	1.35	7.95	7.77	8.08	4.64	4.96	0.39	0.59	1.23
Variance	61.39	10.69	6.92	0.13	18.89	3.31	1.09	1.19	1.82	63.18	60.36	65.32	21.55	24.61	0.15	0.35	1.51
Skewness	0.38	-2.21	-0.05	-0.47	0.65	1.13	0.27	0.6	0.7	1.38	1.03	0.97	0.38	0.52	1.61	1.26	0.43

(b): Summary table for all of the features in digits sub-group:

	nr_pix	height	width	tallness	rows_with_1	cols_with_1	rows_with_5_plus	cols_with_5_plus	one_neighbour	three_or_more_neighbours	no_neighbours_below	no_neighbours_above	no_neighbours_before	no_neighbours_after	nr_regions	nr_eyes	r5_c5
Mean	35.98	15.04	12.02	1.28	7.34	2.54	1.95	1.34	1.82	11.86	15.05	14.29	8.09	8.2	1	0.39	0.61
Stdev	6.48	1.57	2.08	0.25	3.18	1.68	0.84	0.88	0.72	5.88	4.45	4.87	3.44	4.1	0	0.65	1.14
Variance	42.02	2.47	4.31	0.06	10.12	2.84	0.71	0.77	0.51	34.56	19.83	23.7	11.83	16.78	0	0.42	1.3
Skewness	0.42	0.6	0.05	1.23	0.47	-0.18	0.28	0.09	-0.03	1.12	0.85	0.63	0.22	0.79	NA	1.36	0.14

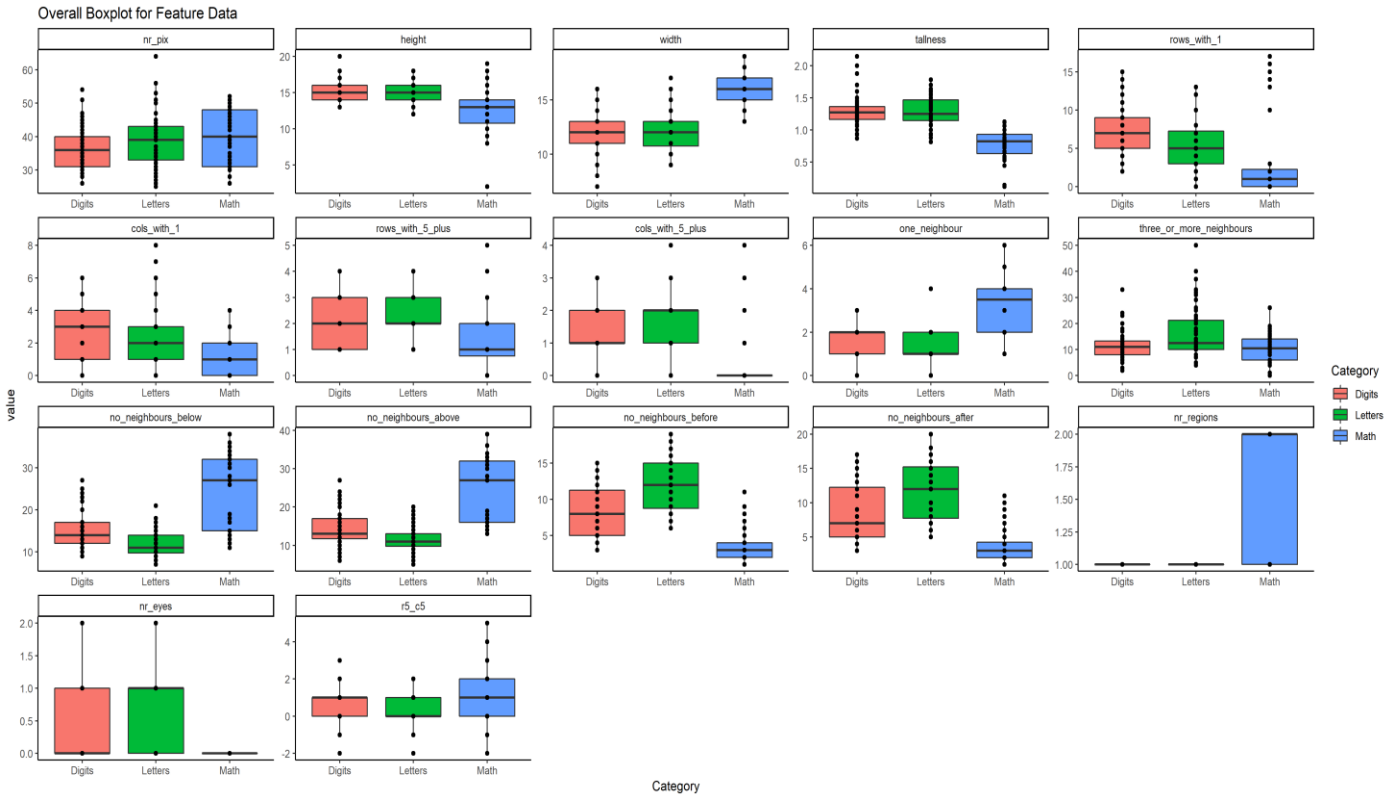
(c): Summary table for all of the features in letters sub-group:

	nr_pix	height	width	tallness	rows_with_1	cols_with_1	rows_with_5_plus	cols_with_5_plus	one_neighbour	three_or_more_neighbours	no_neighbours_below	no_neighbours_above	no_neighbours_before	no_neighbours_after	nr_regions	nr_eyes	r5_c5
Mean	38.95	15.04	12.04	1.28	5.48	2.32	2.2	1.89	1.59	16.45	11.93	11.48	11.62	11.8	1	0.75	0.3
Stdev	8.28	1.33	1.93	0.24	3.7	2.26	0.88	0.93	1.17	9.89	3.17	3.35	3.75	4.18	0	0.58	1.06
Variance	68.49	1.78	3.74	0.06	13.71	5.09	0.78	0.86	1.37	97.85	10.03	11.24	14.09	17.43	0	0.34	1.12
Skewness	0.54	0.21	0.35	0.22	0.5	1.2	0.4	0.61	0.89	1.17	0.65	0.47	0.04	-0.11	NA	0.07	-0.16

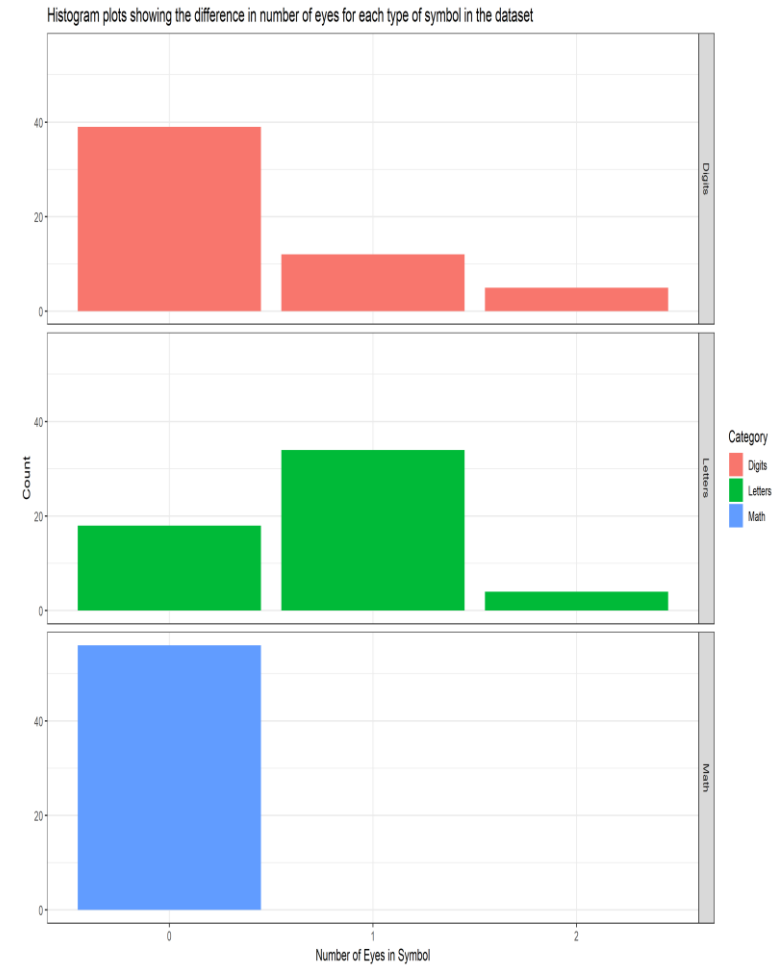
(d) Summary table for all of the features in math symbols sub-group

	nr_pix	height	width	tallness	rows_with_1	cols_with_1	rows_with_5_plus	cols_with_5_plus	one_neighbour	three_or_more_neighbours	no_neighbours_below	no_neighbours_above	no_neighbours_before	no_neighbours_after	nr_regions	nr_eyes	r5_c5
Mean	39.75	11.79	16.14	0.73	2.84	1.11	1.45	0.27	3.27	9.7	24.21	24.59	3.48	3.64	1.55	0	1.18
Stdev	8.24	4.59	1.21	0.29	4.81	0.95	1.23	0.77	1.39	5.94	8.39	8.15	2.36	2.54	0.5	0	1.32
Variance	67.86	21.04	1.47	0.08	23.12	0.9	1.52	0.6	1.95	35.23	70.39	66.39	5.56	6.45	0.25	0	1.75
Skewness	-0.07	-1.1	-0.21	-1.02	1.92	0.8	0.78	3.21	0.04	0	-0.13	-0.09	1.34	1.38	-0.21	NA	0.65

Above are the summary tables showing the mean, standard deviation, variance and skewness for all the features in my dataset. None of the skew exceeds in any of my tables the extreme value of 3 or -3 so the assumption of relative normality can be inferred for each feature. However, these tables are difficult to use to distinguish between the different groups, as the reading of numbers between each table can be tiresome. Below is a boxplot for each feature which better summarises and distinguishes the differences between the groups in my dataset:

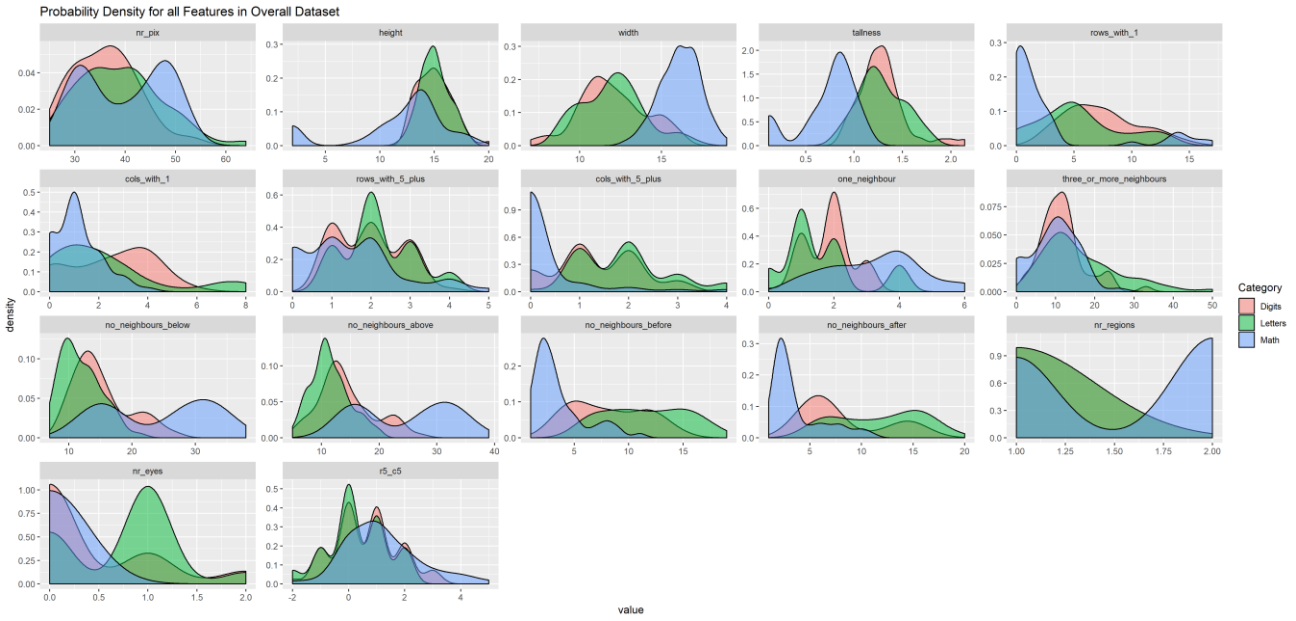


We can already see here some noteworthy differences, namely, that math dataset is the only group that has symbols with two regions and also that no math symbol in my dataset has an eye in it. Below are some histograms which corroborate my findings:



Also, there seems to be no math symbols which have a column with more than 5 black pixels in it which seems to suggest there are no symbols in my math symbols data that have a straight vertical line in it. This is further confirmed by the difference in the no_neighbours_before and no_neighbours_after feature that the math dataset has with the other groups. These feature values are calculated by analysing adjacent columns – symbols that have a vertical line in them like a 1 or an f have large values for these features, this may be a plausible explanation for the difference we see.

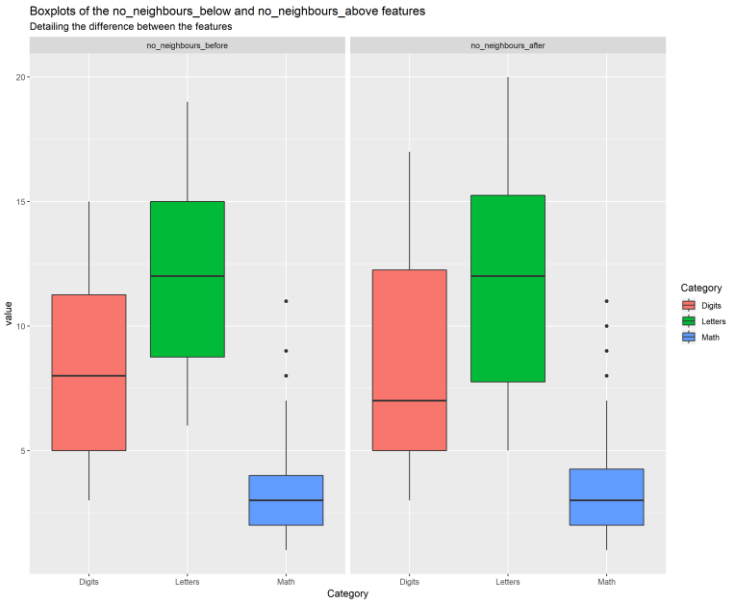
However, these boxplots do not as much to demonstrate as clear a difference between the letters and digits dataset, perhaps a probability density curve may show a clearer contrast:

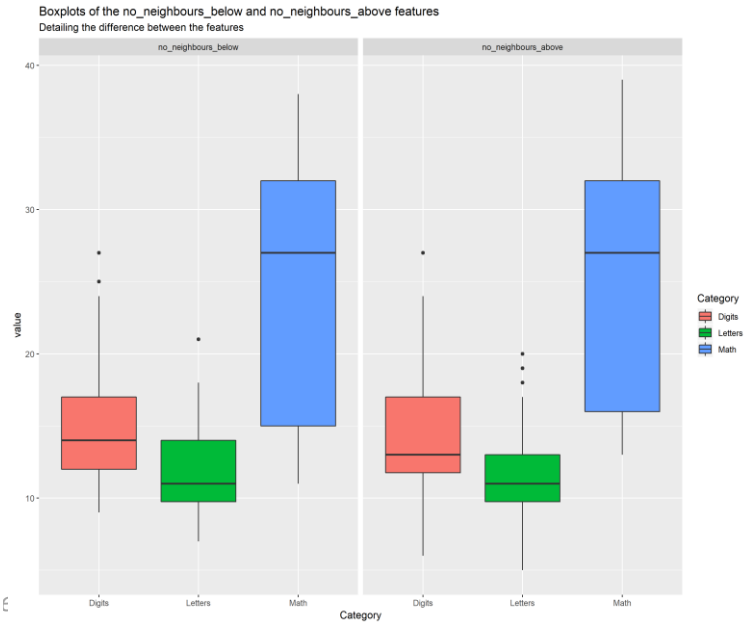


The curves of digits and letters are largely in line with each other whereas often the math symbols curve is much different. A place where the letters dataset is clearly different than the digits dataset is in the nr_regions probability density graph, however I suspect that this is an error in the way one of the handwritten letters was drawn rather than anything significant as the probability of a letter in my data having 2 regions seems very small going by the graph.

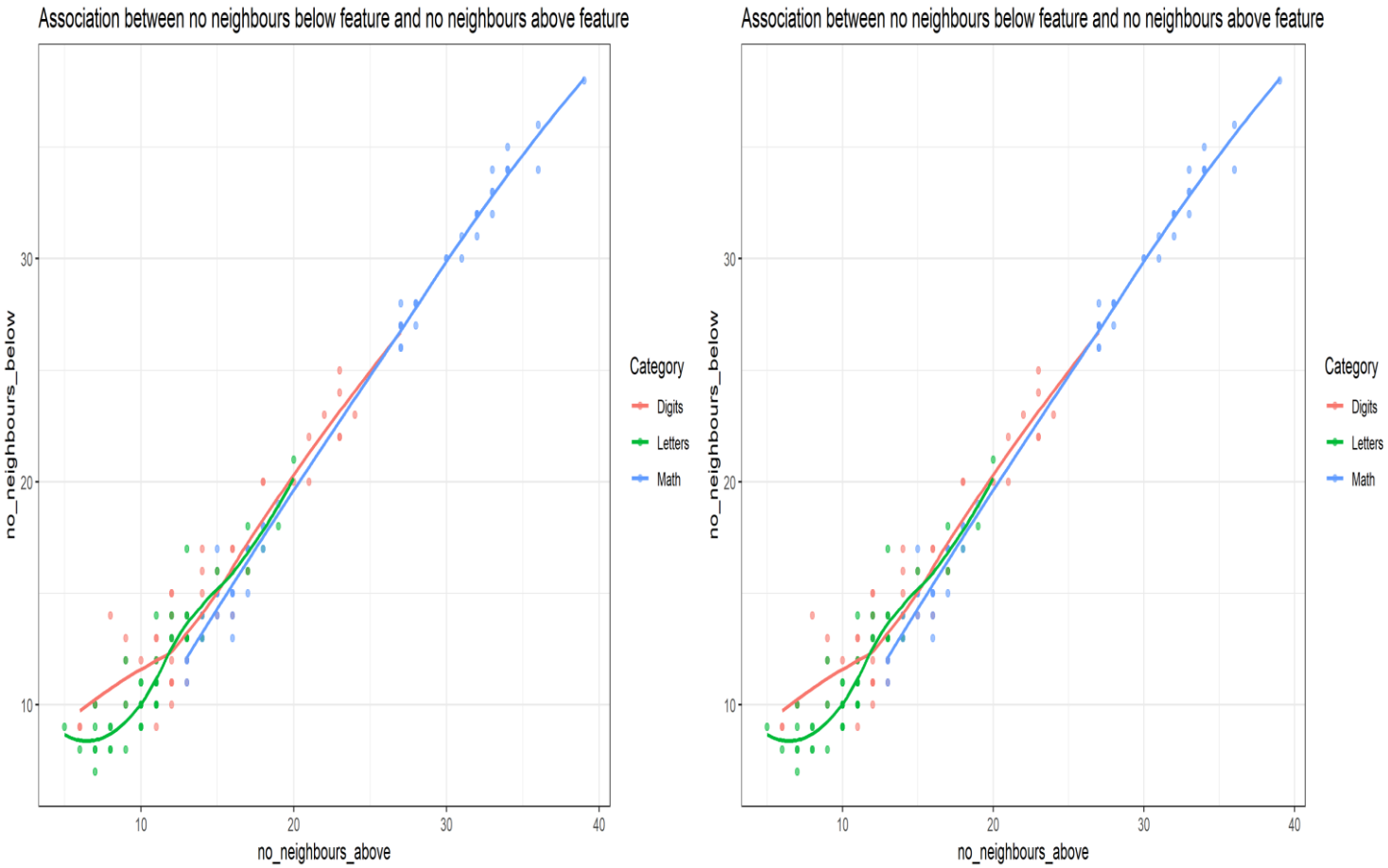
Section 3.4: Highly associated features

Using the boxplots I created earlier I was able to identify features that were highly associated with each other. Below they are demonstrated:





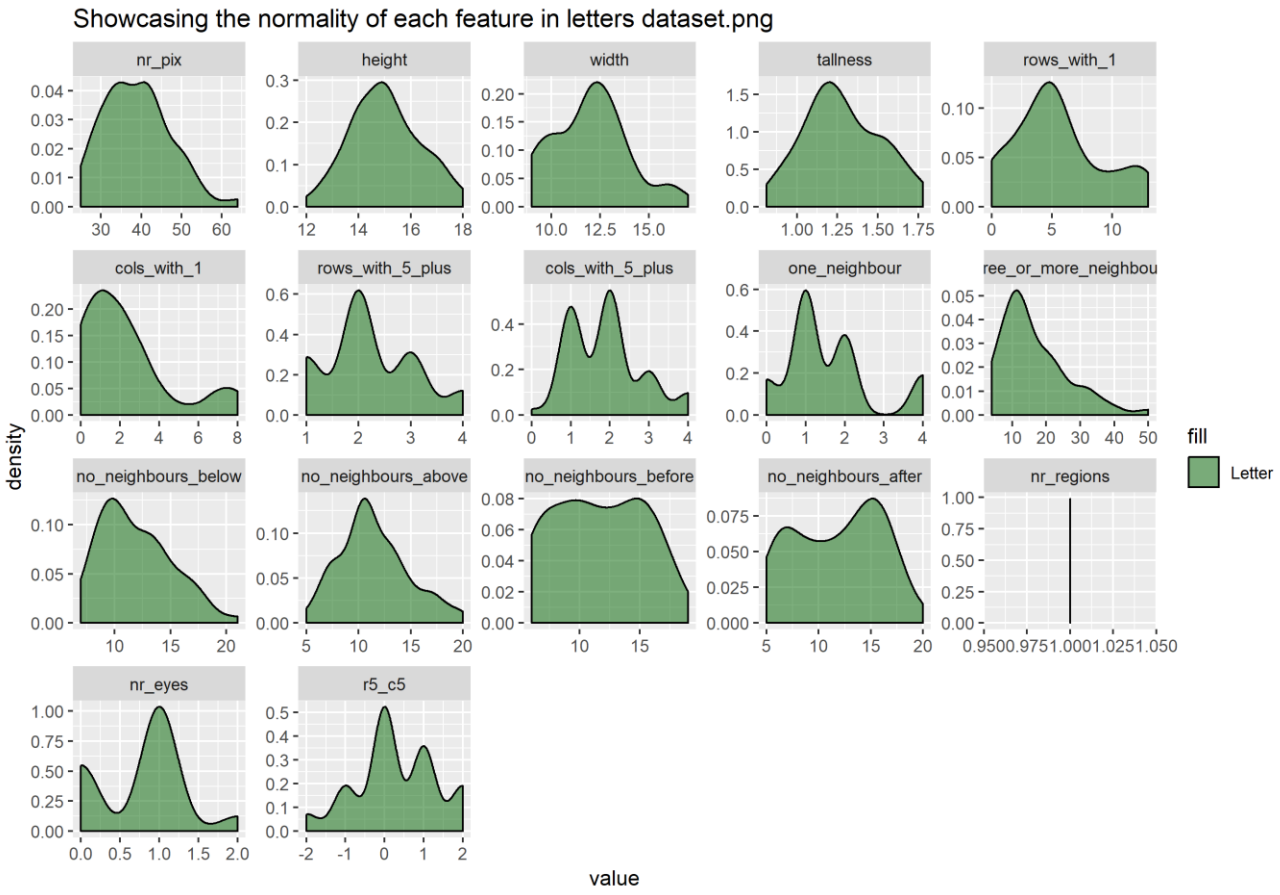
As we can see from above the no_neighbours_before and no_neighbours_after features are nearly the same in value each time and thus are highly associated, this is likewise the case for the no_neighbours_below and the no_neighbours_above features. This can be further validated by the graphs below also:



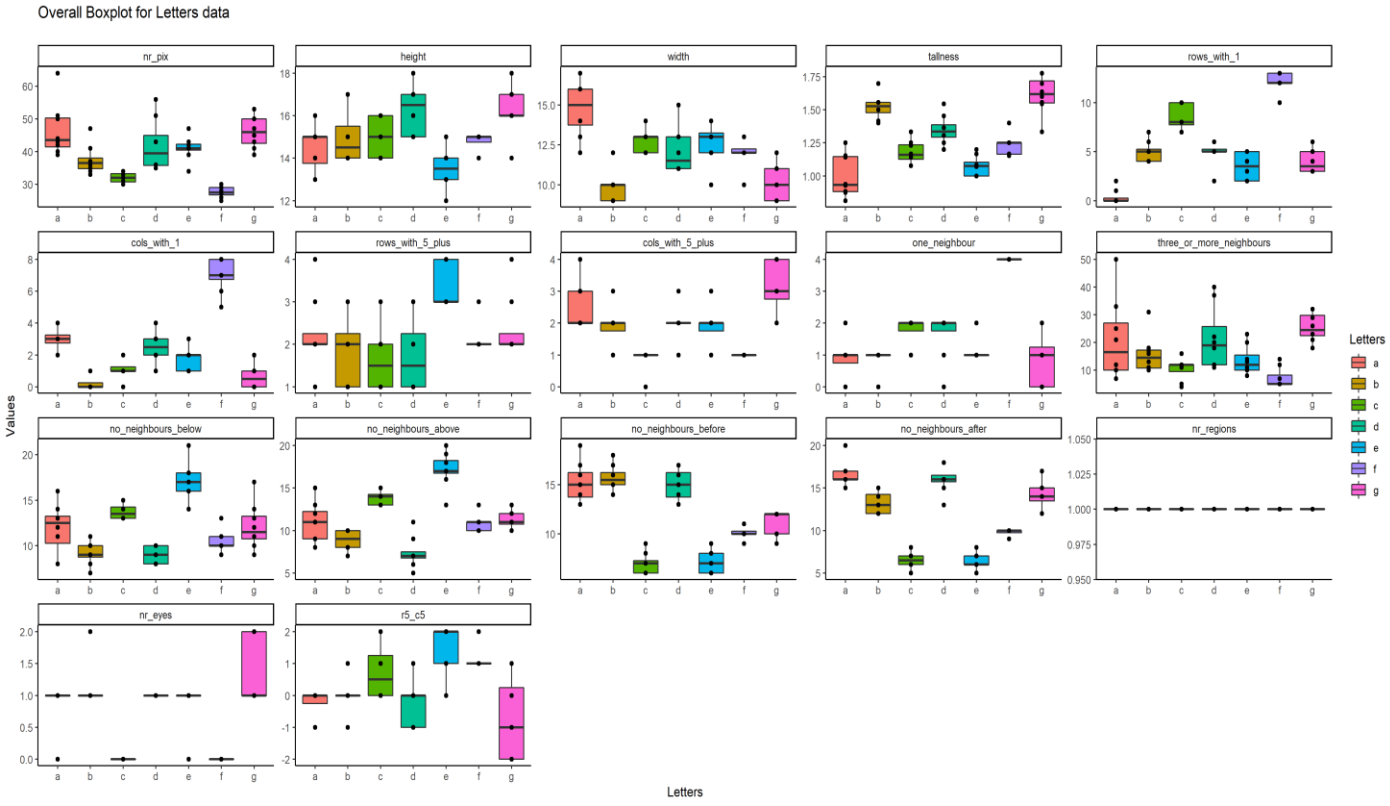
These graphs demonstrate a near identical association. Perhaps the reason for the similarities in these features can be explained by what I mentioned earlier, that being that many of the black pixels that are counted by these features, are usually either straight horizontal or vertical lines, and so are isolated on both fronts – isolated above and below in the case of horizontal lines, and isolated before and after for vertical lines. I'd be hesitant though to discard of the features from each of these associated groups as thy could be of use later possibly. A better solution would be that the features could be combined – e.g no_neighbours_before_or_after and no_neighbours_below_or_above for new features. These features could help identify symbols that have straight horizontal lines and straight vertical lines later.

Section 3.5: Features that are useful in discriminating between the different letters

For the next few questions – (section 3.5 – 3.7) I required a statistical method that tested for statistically significant differences between multiple groups as there are 7 different groups of symbols within the letters and digits dataset, and 3 different sub-groups in the entire features dataset. Therefore, I used a two-way ANOVA test – (not a one way test as I only had to discover if there were differences in the sample means). Before I did this, for each question, 3 conditions had to be met, these are described below. **Nearly normal observations:** As mentioned before the assumption of normality within each feature is a founded assumption. Below is a probability density graph for each feature in the letters dataset to further justify this:



Also necessary is the **independence of the samples within and between each group**, these handwritten symbols are less than 10% of the population of all handwritten symbols for each of their groups so this condition is met. To demonstrate **equal variability along groups**, as well as handily display the differences between letters I created a boxplot:



For each of the following questions I will take the steps above to demonstrate that the conditions for an ANOVA test have been met.

Feature	P Value
nr_pix	2.34E-09
height	1.85E-05
width	1.59E-09
tallness	1.49E-15
rows_with_1	2.99E-24
cols_with_1	3.73E-22
rows_with_5_plus	0.000390445
cols_with_5_plus	3.59E-09
one_neighbour	1.13E-15
three_or_more_neighbours	0.000431641
no_neighbours_below	1.38E-11
no_neighbours_above	1.44E-15
no_neighbours_before	2.96E-21
no_neighbours_after	5.14E-25
nr_eyes	1.80E-13
r5_c5	8.11E-07

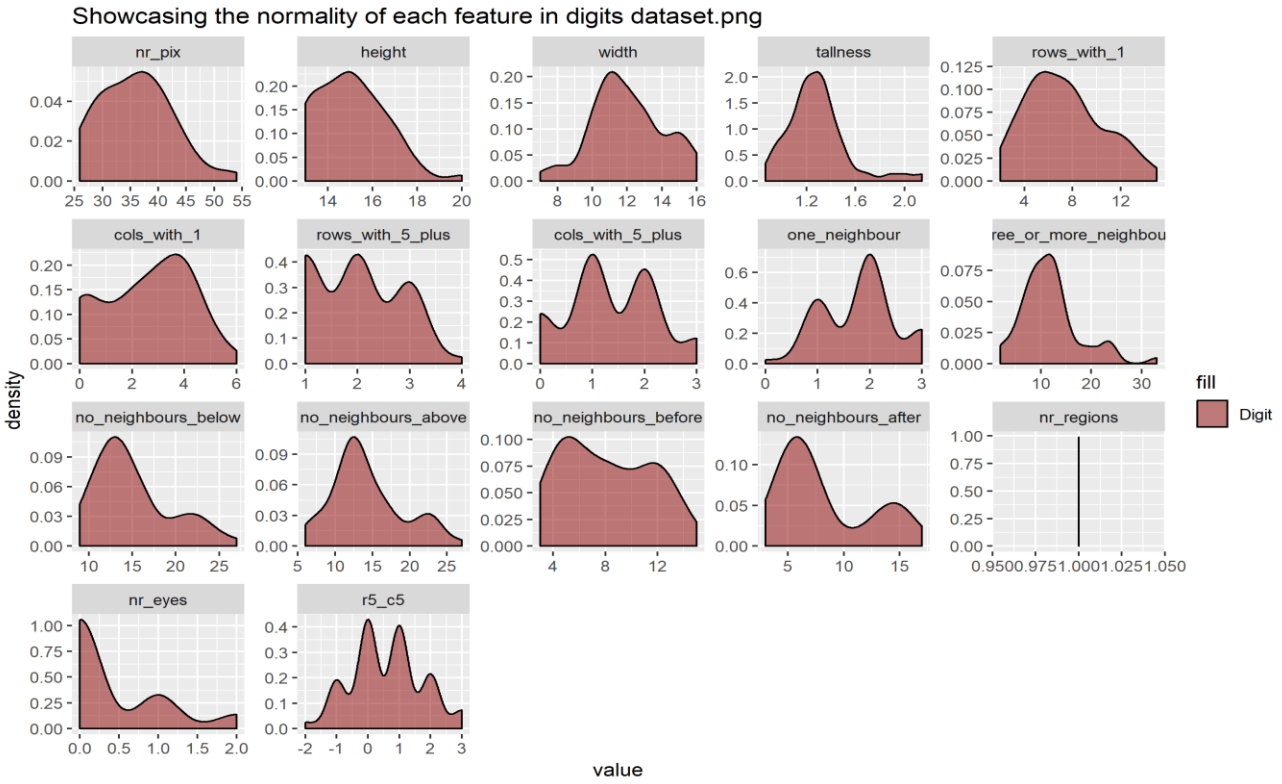
The results of my ANOVA test, with a significance value of 0.05 returned these features as being useful to discriminate between the 7 different letters, if one looks at the boxplot above, you can see that these calculations are largely accurate. The only feature that does not return statistically significant is the nr_regions feature as it is equal to 1 for each of the letters. This ANOVA test is merely a preliminary test If the feature returns a p value that is less than our alpha, it does not mean that the feature is useful for distinguishing for all groups of letters all the time but rather it can be significant in distinguishing at least one letter from the others. The features which seem to have the smallest p-value – i.e the most useful in discriminating between letters is the no_neighbours_before and the no_neighbours_after features – this is probably because they isolate letters that have a long vertical line in them – letters like a, b, d, f and g are separated from c and e which are more horizontal in their orientation. The rows_with_1 feature is marked as especially significant for the same reason. I would have expected that the nr_eyes feature would have been the largest discriminator for letters but I believe that the large variance and error within the g symbol threw off the ANOVA test. The hypothesis test for each feature is as follows:

H_0 = The mean value across each letter is the same

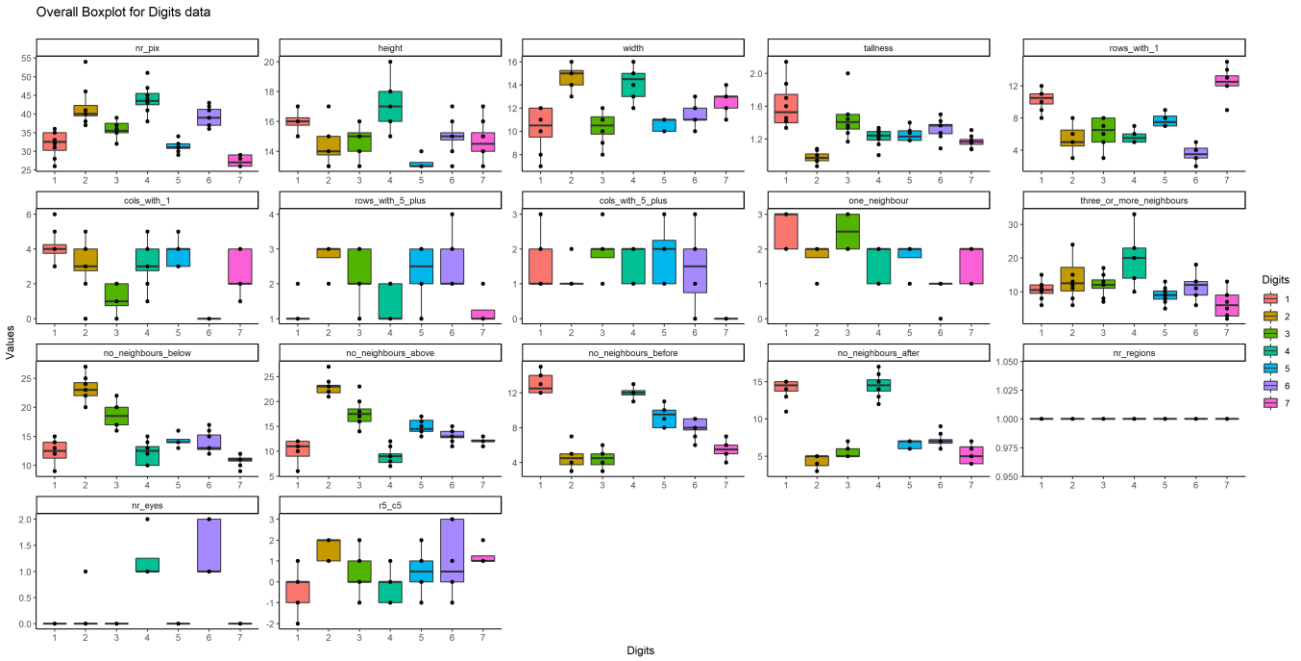
H_A = The mean value for at least one pair of letters is different

Section 3.6: Features that are useful in discriminating between the different digits

The relative normality of the feature data for the digits dataset is demonstrated below:



The notable non-normal distribution in this data is the nr_eyes feature, a reason for this is highlighted in the following boxplot – which also reassures us that the variability along groups is nearly equal too.



The nr_eyes feature singles out the digits 4 and 6 from the rest of the group quite clearly, again there must be some error in the way that some 6's were originally drawn as it's suggesting that some of the images have two eyes

Alongside are the results of my ANOVA test, the hypothesis test for each feature in this ANOVA test is as follows:

H_0 = The mean value across each digit is the same

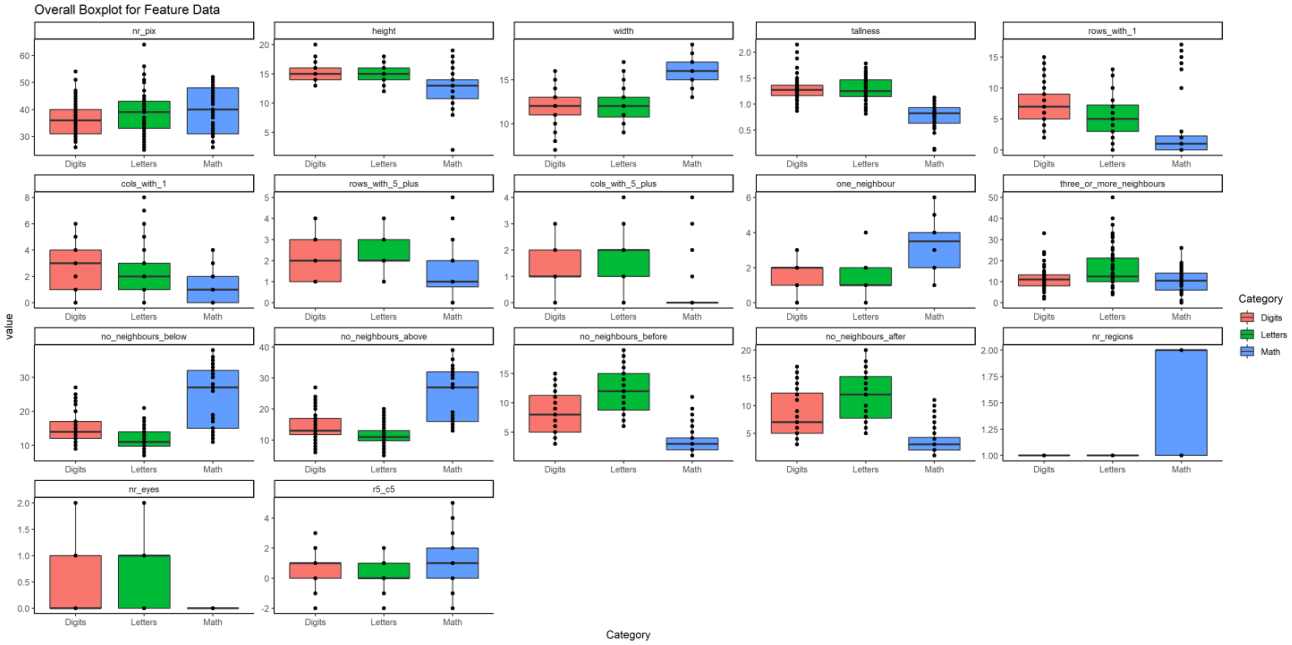
H_A = The mean value for at least one pair of digits is different

Again, the only feature that does not have any statistical significance is the nr_regions feature - as each digit has only one region. In a manner identical to the letters dataset, the most significant feature are the no_neighbours_before and no_neighbours_after features, with the no_neighbours_below and no_neighbours_above features following closely behind. The reason for this, I assume, is again identical to the reason discussed above.

Feature	P Value
nr_pix	3.19E-14
height	7.49E-07
width	5.59E-11
tallness	2.90E-08
rows_with_1	1.30E-16
cols_with_1	3.23E-10
rows_with_5_plus	2.17E-07
cols_with_5_plus	1.20E-05
one_neighbour	3.53E-08
three_or_more_neighbours	3.19E-05
no_neighbours_below	2.09E-18
no_neighbours_above	1.51E-19
no_neighbours_before	4.12E-24
no_neighbours_after	5.76E-27
nr_eyes	1.70E-16
r5_c5	0.000315125

Section 3.7: Features that are useful for discriminating between the three groups

I can use a two-way ANOVA test again to distinguish differences between the three groups, with this ANOVA test the results we obtain should be more concise and we can be more confident of their findings as we are comparing only between three groups rather than seven like before. The assumptions are like before except we are more confident in the normality of our feature data as a larger sample size usually leads to more normality in our data. The variability is roughly equal within and between the feature data for the majority of features as shown below:



The hypothesis test for this ANOVA for each feature is as follows:

H_0 = The mean value across each symbol group is the same

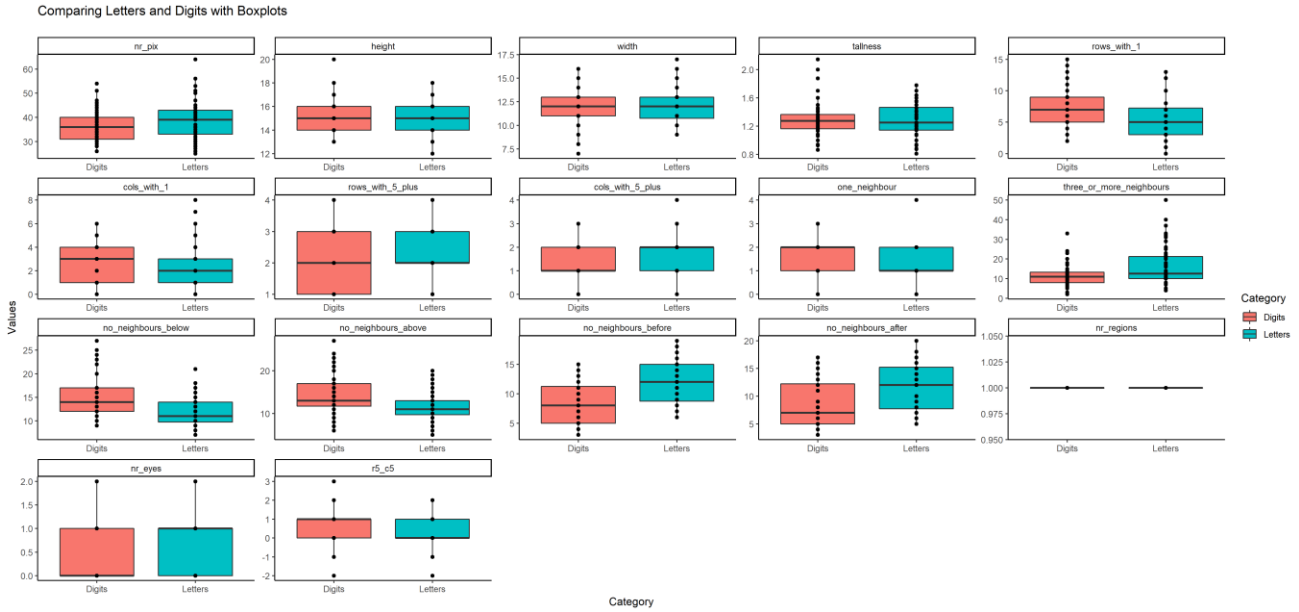
H_A = The mean value for at least one pair of symbol groups is different

By this analysis all the features seem to be statistically significant in discriminating at least one group from the others for each feature. I suspect this is mostly to do with the difference of the Math dataset from the other two groups. The most significant values are again for the neighbour features, which was already discussed in section 3.3, to further what was said there, in the case of the symbols I believe that the math symbols are more horizontal in their orientation and less vertical than the other two sub groups of symbols and this may explain the difference. The findings of the ANOVA when it comes to the statistical significance of the r5_c5 feature is worrying though, it does not seem by glance of the boxplot distribution to be the case. I imagine this is because the wide variance of the Math dataset has thrown the ANOVA test off.

Features	P Value
nr_pix	0.026560613
height	1.15E-09
width	4.23E-29
tallness	6.50E-26
rows_with_1	6.64E-08
cols_with_1	2.52E-05
rows_with_5_plus	0.000424735
cols_with_5_plus	4.68E-18
one_neighbour	8.79E-14
three_or_more_neighbours	1.48E-05
no_neighbours_below	2.42E-22
no_neighbours_above	8.65E-25
no_neighbours_before	6.15E-27
no_neighbours_after	1.59E-22
nr_regions	2.59E-22
nr_eyes	3.54E-12
r5_c5	0.000505478

Section 3.8: Features that are useful for discriminating between the set of digits and the set of letters

As I was only comparing two groups this time, I found that a t-test was more suited to my needs, below is a boxplot that can help as a guide to comparing differences in the two groups alongside the p value table I got from my t-test:



The result of the t-test is below. As we can see, far less features were returned as statistically significant, this is as a result of less groups to compare and the digits and letters datasets are relatively similar for most

Features	Confidence interval for Difference in Means (if positive, difference is on digits side, otherwise it's on the letters side)	Mean Difference	P Value
rows_with_1	0.19 : 3.52	1.857142857	0.029215262
cols_with_5_plus	-0.96 : -0.15	-0.553571429	0.008786281
three_or_more_neighbours	-7.62 : -1.56	-4.589285714	0.003640935
no_neighbours_below	1.56 : 4.69	3.125	0.000196315
no_neighbours_above	1.3 : 4.31	2.803571429	0.000446728
no_neighbours_before	-4.66 : -2.41	-3.535714286	5.11E-08
no_neighbours_after	-4.63 : -2.58	-3.607142857	2.88E-09
nr_eyes	-0.62 : -0.09	-0.357142857	0.008576641

features. Yet again the most statistically significant feature seems to be the pairing of the no_neighbours_before and the no_neighbours_after features – my estimation for why this is has already been explained before. The inclusion of the nr_eyes being statistically significant is likely because there are possibly a couple of outlier in the letters dataset that have been erroneously drawn and couted as having two eyes and so skew our p value.

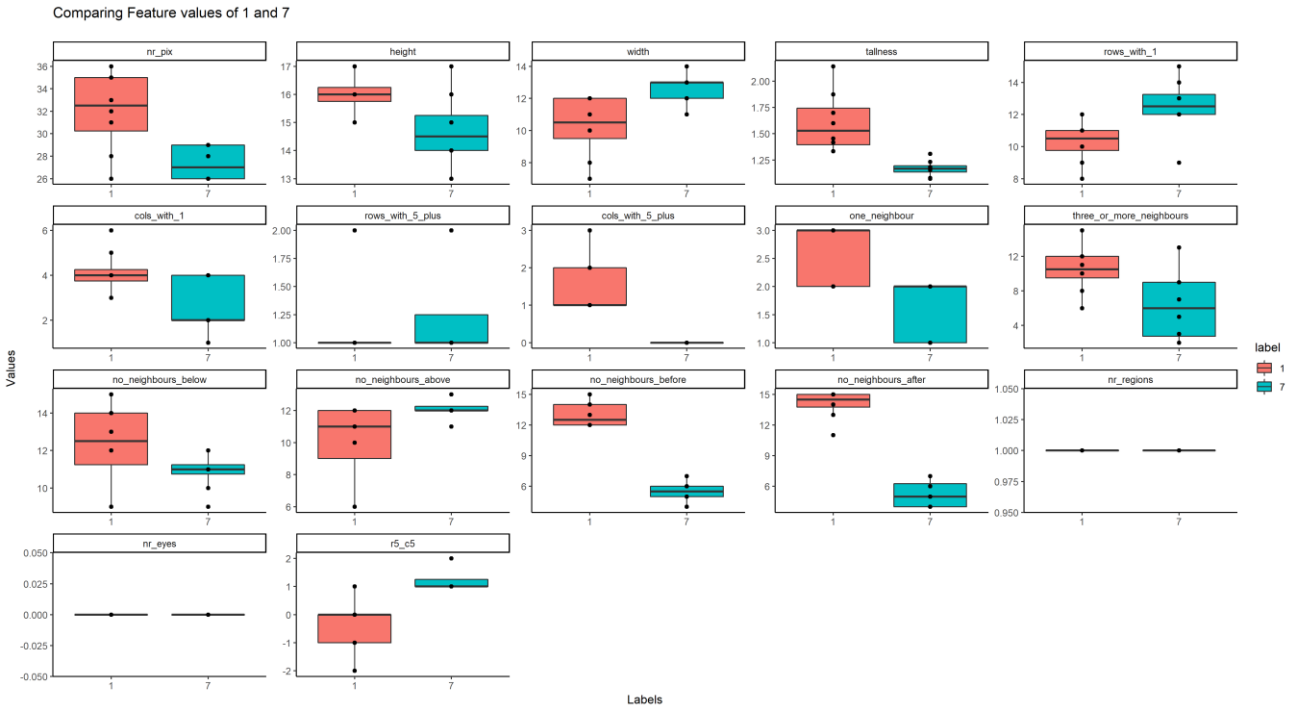
The hypothesis test for this t-test for each feature was as follows:

H_0 = The mean value is the same for the Digits and Letters dataset

H_A = The mean value is different for the Digit and letters dataset

Section 3.9: Features that are useful for discriminating between digit ‘1’ and the digit’7’

For this test, as well as the next three questions – I used a two-way t-test, as the questions only asked me to show a difference and not whether one symbol is greater or lesser in a feature than it’s counterpart. In this test as well as the next few t-tests I assumed no extreme skew in either group and an independence within and between groups. Unlike an ANOVA equal variability along groups is not necessary, however I still feel like a boxplot may be beneficial to have to corroborate the findings of my t-test:

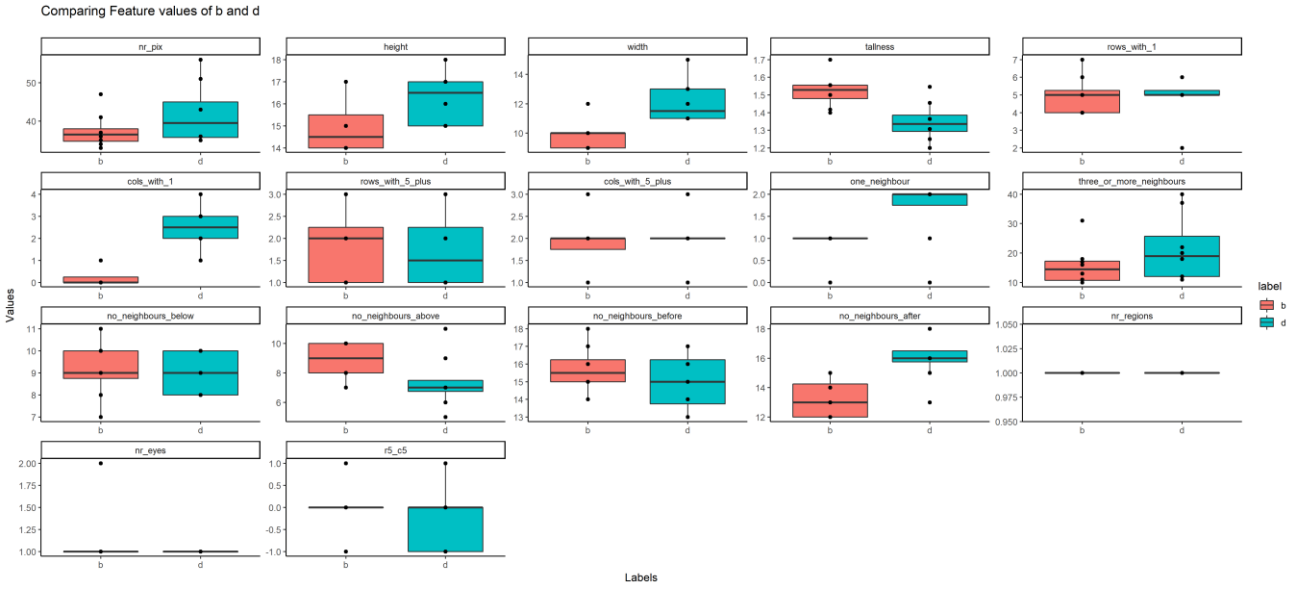


There seem to be a wide range of notable differences to choose from going by the boxplot above, although

Feature	Confidence Interval for Difference in Means (if positive difference is on the side of '1', if negative difference is on the side of '7'	Mean Difference	P_Value
nr_pix	1.63 : 7.62	4.625	0.008169
width	-3.55 : -1.2	-2.375	0.002033
tallness	0.2 : 0.67	0.437490634	0.003174
cols_with_1	0.87 : 2.13	1.5	0.000805
cols_with_5_plus	0.87 : 2.13	1.5	0.000805
one_neighbour	0.37 : 1.63	1	0.007247
no_neighbours_above	-3.99 : -0.26	-2.125	0.030962
no_neighbours_before	6.09 : 8.91	7.5	4.70E-06
no_neighbours_after	6.67 : 10.83	8.75	2.24E-05
r5_c5	-2.51 : -0.74	-1.625	0.003424

the significant features returned here are much less than before with the comparison of two groups – despite the differences seeming more notable by box-plot. Likely, this is a result of a smaller sample size having less degrees of freedom and so the t-distribution has a fatter tail. It treats outliers with more leniency than the previous tests we used. The digit 7 as a symbol, using the stats above as a guide, seems to be wider and have a more horizontal orientation – with the significance of the no_neighbours_before and no_neighbours_after features being prevalent. This is an expected result. What perhaps is not expected is the mean number of black pixels being more on the side of ‘1’ digit than the ‘7’ digit. Perhaps this is because of the way I draw my 1’s.

Section 3.10: Features that are useful for discriminating between symbol ‘b’ and ‘d’



The differences between the symbols of b and d at a glance are far less pronounced than comparisons before. Below are the results of the t-test I performed on them

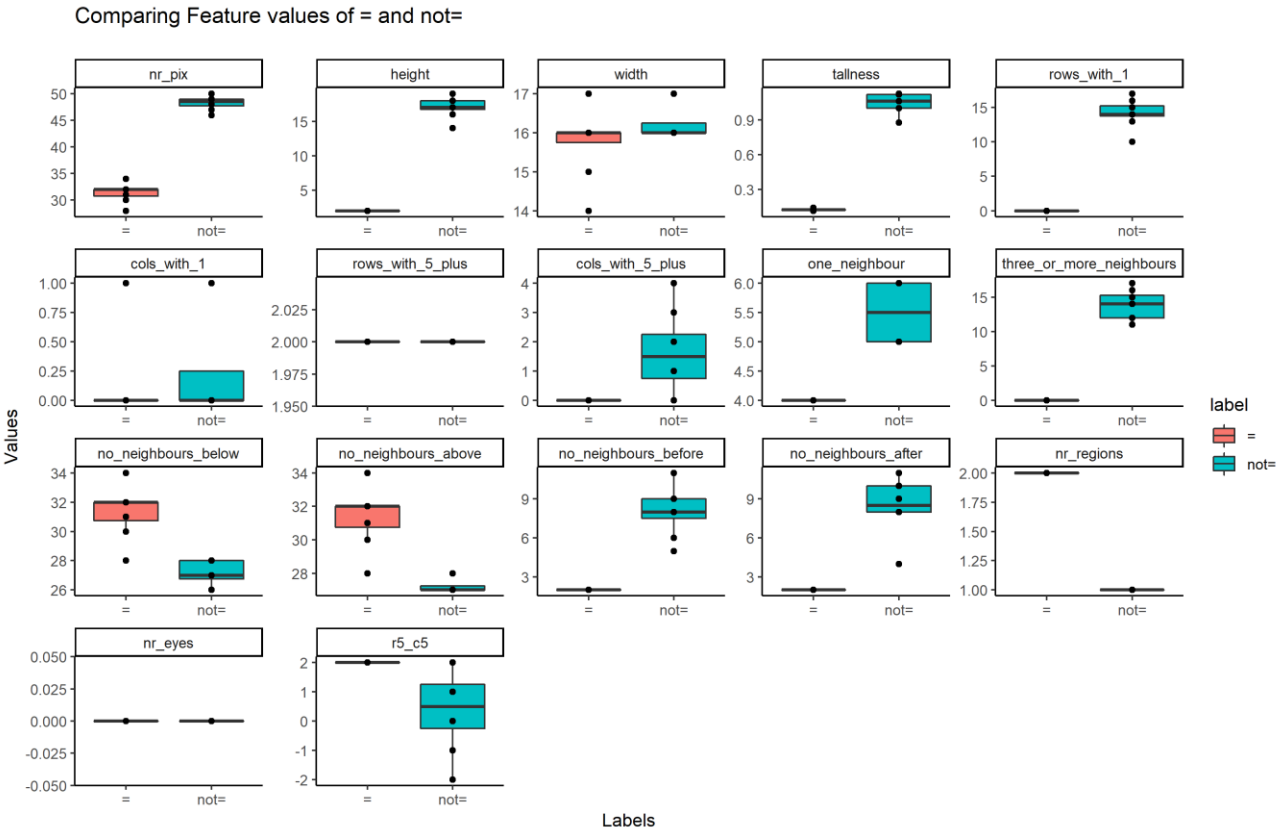
Feature	Confidence Interval for Difference in Means (if positive difference is on the side of 'b', if negative difference is on the side of ‘d’	Mean Difference	P_Value
nr_pix	-8.73 : -0.02	-4.375	0.049174
height	-2.32 : -0.18	-1.25	0.028161
width	-3.22 : -1.28	-2.25	0.000943
tallness	0.11 : 0.24	0.173834499	0.000328
cols_with_1	-3.12 : -1.38	-2.25	0.000468
one_neighbour	-1.49 : -0.01	-0.75	0.047945
no_neighbours_above	0.32 : 2.68	1.5	0.019942
no_neighbours_after	-3.91 : -1.59	-2.75	0.000815

The symbol of ‘d’ seems to have a greater mean number of pixels than the b symbol. This may be related to the height difference too – I possibly draw my d’s on average taller than my b’s – this is further confirmed by the large difference the two symbols seem to have in the tallness features. - Another notable difference is the no_neighbours_after feature – which ‘d’ has more occurrences of – this is most likely related to the fact that the symbol ‘d’ has it’s vertical line on the right of the symbol and for each pixel in that vertical line it should increment the no_neighbours_after feature.

H_0 = The mean value is the same for the ‘b’ sample and the ‘d’ sample

H_A = The mean value is different for the ‘b’ sample and the ‘d’ sample

Section 3.11: Features that are useful for discriminating between **symbol '=' and '≠'**



Unlike the 'b' and 'd' symbol, at a glance the '=' and '≠' symbols have pronounced differences. An interesting observation that can be made from the boxplot above is that the features that the samples in the '='symbol group have possessed very little variance – they seem to be quite the same most of the time. Unfortunately, the standard t-test function would not work on these data and I could not diagnose the problem, so I had to use the pairwise.t.test function. Consequently, the information here is less than the information before. Anyways, below are the p-values from my dataset to illuminate further differences in the two symbols:

The major discriminator between the pair of symbols is the nr_regions feature – where an = sign is always 2 regions, a ≠ is always one region. With this feature alone you could reliably distinguish one symbol from the other. Also prevalent is the difference in height, nr_pix and tallness features which are as simply a result of ≠ being more massive shape on average. The rows_with_1 is also a telling feature as there are no rows_with_1 features being recorded on any of the handwritten = symbols in my sample. The hypothesis test for this two-way t-test for each feature was as follows:

H_0 = The mean value is the same for the '=' sample and the '≠' sample

H_A = The mean value is different for the '=' sample and the '≠' sample

Features	P Value
nr_pix	3.21E-12
height	1.05E-13
tallness	3.92E-14
rows_with_1	2.12E-11
cols_with_5_plus	0.005646215
one_neighbour	1.50E-06
three_or_more_neighbours	2.70E-11
no_neighbours_below	2.52E-05
no_neighbours_above	1.70E-05
no_neighbours_before	2.72E-07
no_neighbours_after	5.85E-07
nr_regions	0
r5_c5	0.005646215

Section 3.12: For each feature, find the pairs of digits that have a statistically significant difference for that feature

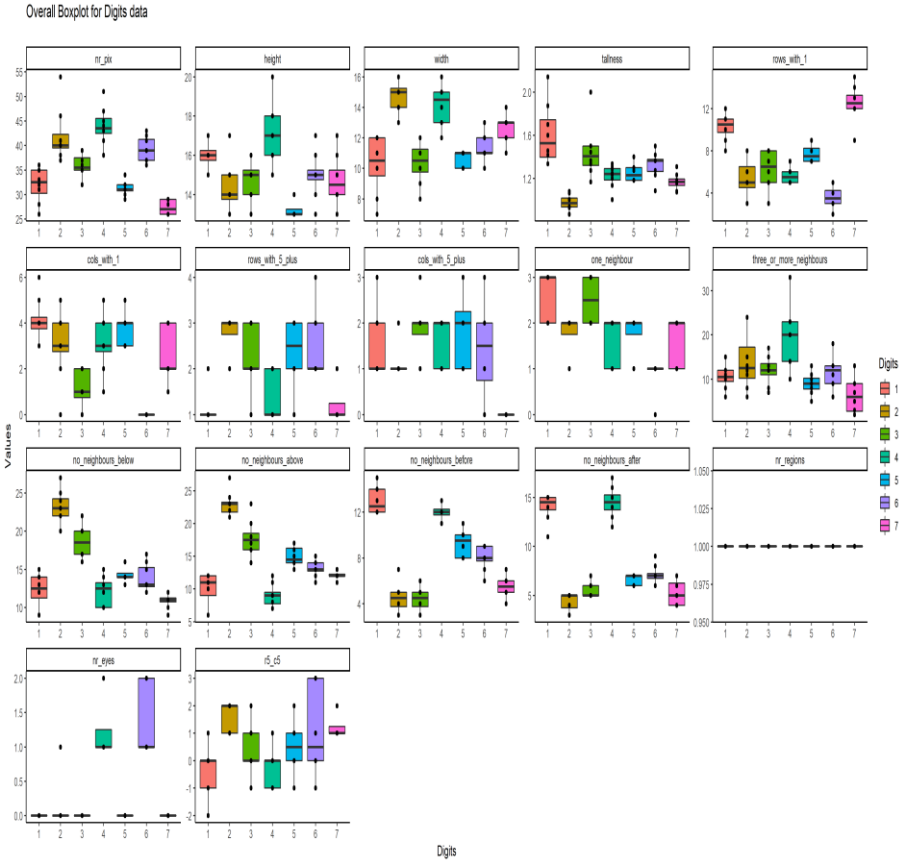
For the next three questions of this assignment, I was required to do multiple comparisons within each group of symbols in order to pinpoint where exactly the differences in means are for each feature. I had to be careful here as with multiple comparisons comes an increased probability in a type 1 error, so multiple comparison correction had to be made. For this reason, I chose to use a TukeyHSD post-hoc test, as through research it seemed more suitable to use when having a sample size with a substantial number of multiple comparisons, rather than the pairwise.t.test with bonferonni correction suggested in the lectures. Within the Tukey HSD function I made sure to specify a 98% family-wise confidence level to account for errors. Below are the results I acquired for each group of symbols along with a boxplot to verify the results.

Feature	Comparison	P_Value
nr_pix	2-1.	3.06E-06
nr_pix	4-1.	3.97E-08
nr_pix	6-1.	0.000954124
nr_pix	3-2.	0.009929681
nr_pix	5-2.	6.02E-07
nr_pix	7-2.	1.40E-10
nr_pix	4-3.	0.00021109
nr_pix	7-3.	5.77E-05
nr_pix	5-4.	7.82E-09
nr_pix	7-4.	0
nr_pix	6-5.	0.00021109
nr_pix	7-6.	5.21E-08
height	5-1.	0.000255421
height	4-2.	0.000255421
height	4-3.	0.002160843
height	5-4.	2.57E-07
height	6-4.	0.008171518
height	7-4.	0.002160843
height	6-5.	0.049187675
width	2-1.	3.86E-08
width	4-1.	1.45E-06
width	7-1.	0.005390136
width	3-2.	7.97E-08
width	5-2.	3.41E-07
width	6-2.	2.56E-05
width	7-2.	0.017487414
width	4-3.	2.99E-06
width	7-3.	0.009819837
width	5-4.	1.26E-05
width	6-4.	0.00079718
width	7-5.	0.030355475
tallness	2-1.	1.39E-08
tallness	4-1.	0.000389309
tallness	5-1.	0.001387011
tallness	6-1.	0.02011356
tallness	7-1.	5.46E-05
tallness	3-2.	1.29E-05
tallness	5-2.	0.026429497
tallness	6-2.	0.001908219
tallness	7-3.	0.023352317
rows_with_1	2-1.	2.57E-07
rows_with_1	3-1.	1.94E-05
rows_with_1	4-1.	8.92E-07
rows_with_1	5-1.	0.016327898
rows_with_1	6-1.	4.51E-11
rows_with_1	7-1.	0.041497387
rows_with_1	5-2.	0.026296689
rows_with_1	7-2.	8.07E-13
rows_with_1	6-3.	0.00995251
rows_with_1	7-3.	2.91E-10
rows_with_1	7-4.	1.11E-11
rows_with_1	6-5.	1.05E-05
rows_with_1	7-5.	4.79E-07
rows_with_1	7-6.	0

Feature	Comparison	P_Value
cols_with_1	3-1.	8.74E-06
cols_with_1	6-1.	3.83E-09
cols_with_1	3-2.	0.010925435
cols_with_1	6-2.	8.74E-06
cols_with_1	4-3.	0.005370196
cols_with_1	5-3.	0.000108173
cols_with_1	6-4.	3.72E-06
cols_with_1	6-5.	5.03E-08
cols_with_1	7-6.	0.000108173
rows_with_5_plus	2-1.	2.73E-05
rows_with_5_plus	3-1.	0.00674831
rows_with_5_plus	5-1.	0.001859292
rows_with_5_plus	6-1.	0.000477678
rows_with_5_plus	4-2.	0.000477678
rows_with_5_plus	7-2.	0.000116426
rows_with_5_plus	7-3.	0.022369787
rows_with_5_plus	5-4.	0.022369787
rows_with_5_plus	6-4.	0.00674831
rows_with_5_plus	7-5.	0.00674831
rows_with_5_plus	7-6.	0.001859292
cols_with_5_plus	7-1.	0.000951911
cols_with_5_plus	7-2.	0.025828268
cols_with_5_plus	7-3.	2.28E-05
cols_with_5_plus	7-4.	0.00028362
cols_with_5_plus	7-5.	2.28E-05
cols_with_5_plus	7-6.	0.003049482
one_neighbour	2-1.	0.011830805
one_neighbour	4-1.	0.002543492
one_neighbour	5-1.	0.011830805
one_neighbour	6-1.	6.13E-08
one_neighbour	7-1.	0.002543492
one_neighbour	3-2.	0.047295653
one_neighbour	6-2.	0.011830805
one_neighbour	4-3.	0.011830805
one_neighbour	5-3.	0.047295653
one_neighbour	6-3.	3.83E-07
one_neighbour	7-3.	0.011830805
one_neighbour	6-4.	0.047295653
one_neighbour	6-5.	0.011830805
one_neighbour	7-6.	0.047295653
three_or_more_neighbours	4-1.	0.004346458
three_or_more_neighbours	7-2.	0.020693901
three_or_more_neighbours	4-3.	0.027697041
three_or_more_neighbours	5-4.	0.000560079
three_or_more_neighbours	6-4.	0.015346921
three_or_more_neighbours	7-4.	9.70E-06

Feature	Comparison	P_Value
no_neighbours_below	2-1.	0
no_neighbours_below	3-1.	1.68E-07
no_neighbours_below	3-2.	0.000133107
no_neighbours_below	4-2.	0
no_neighbours_below	5-2.	9.13E-12
no_neighbours_below	6-2.	0
no_neighbours_below	7-2.	0
no_neighbours_below	4-3.	1.04E-07
no_neighbours_below	5-3.	0.000518352
no_neighbours_below	6-3.	8.39E-05
no_neighbours_below	7-3.	8.53E-10
no_neighbours_below	7-5.	0.006662267
no_neighbours_below	7-6.	0.030578373
no_neighbours_above	2-1.	0
no_neighbours_above	3-1.	2.34E-09
no_neighbours_above	5-1.	6.24E-05
no_neighbours_above	6-1.	0.028996397
no_neighbours_above	3-2.	2.52E-05
no_neighbours_above	4-2.	0
no_neighbours_above	5-2.	9.30E-10
no_neighbours_above	6-2.	0
no_neighbours_above	7-2.	0
no_neighbours_above	4-3.	5.78E-11
no_neighbours_above	6-3.	0.000236926
no_neighbours_above	7-3.	6.36E-06
no_neighbours_above	5-4.	1.58E-06
no_neighbours_above	6-4.	0.001318319
no_neighbours_above	7-4.	0.028996397
no_neighbours_before	2-1.	0
no_neighbours_before	3-1.	0
no_neighbours_before	5-1.	1.74E-07
no_neighbours_before	6-1.	4.99E-11
no_neighbours_before	7-1.	0
no_neighbours_before	4-2.	0
no_neighbours_before	5-2.	2.58E-10
no_neighbours_before	6-2.	8.98E-07
no_neighbours_before	4-3.	0
no_neighbours_before	5-3.	1.15E-10
no_neighbours_before	6-3.	3.95E-07
no_neighbours_before	5-4.	0.000113261
no_neighbours_before	6-4.	3.37E-08
no_neighbours_before	7-4.	0
no_neighbours_before	7-5.	1.74E-07
no_neighbours_before	7-6.	0.000528852
no_neighbours_after	2-1.	0
no_neighbours_after	3-1.	0
no_neighbours_after	5-1.	0
no_neighbours_after	6-1.	0
no_neighbours_after	7-1.	0
no_neighbours_after	4-2.	0
no_neighbours_after	5-2.	0.003813136
no_neighbours_after	6-2.	0.000220786
no_neighbours_after	4-3.	0
no_neighbours_after	5-4.	0
no_neighbours_after	6-4.	0
no_neighbours_after	7-4.	0
no_neighbours_after	7-6.	0.025776928

Feature	Comparison	P_Value
nr_eyes	4-1.	7.13E-10
nr_eyes	6-1.	3.49E-11
nr_eyes	4-2.	1.41E-08
nr_eyes	6-2.	7.13E-10
nr_eyes	4-3.	7.13E-10
nr_eyes	6-3.	3.49E-11
nr_eyes	5-4.	7.13E-10
nr_eyes	7-4.	7.13E-10
nr_eyes	6-5.	3.49E-11
nr_eyes	7-6.	3.49E-11
r5_c5	2-1.	0.001765398
r5_c5	6-1.	0.03784677
r5_c5	7-1.	0.01866927
r5_c5	4-2.	0.004008646
r5_c5	7-4.	0.03784677



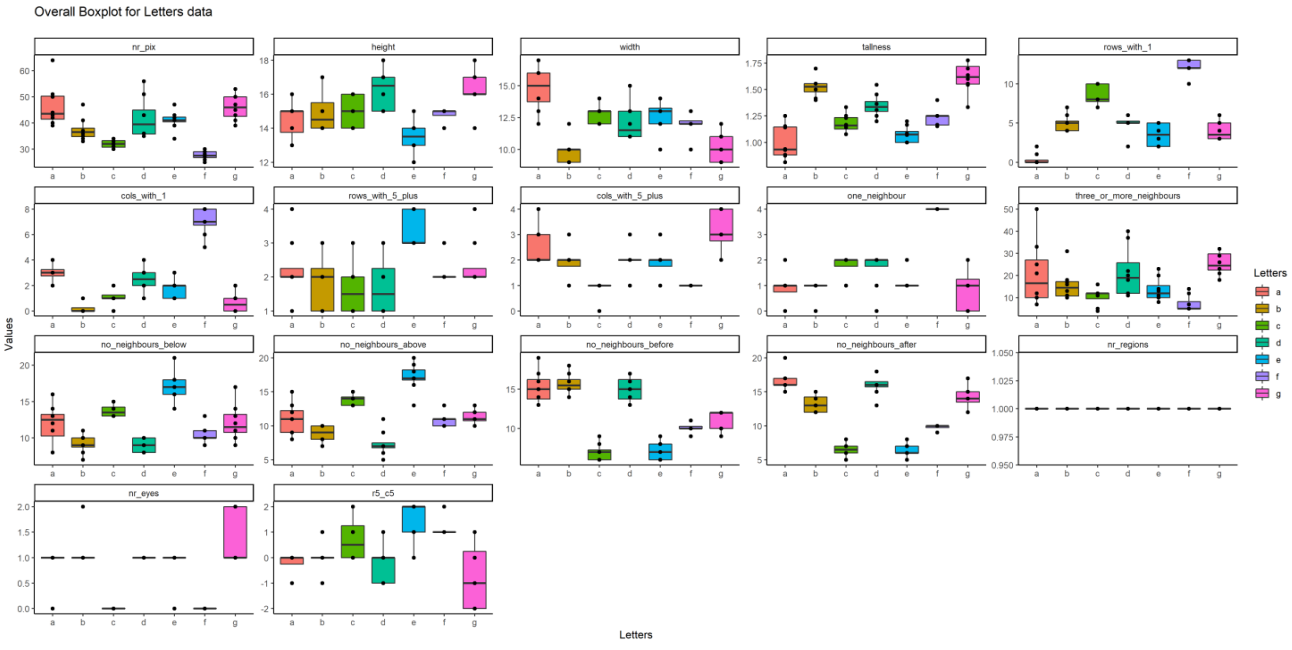
Section 3.13: For each feature, find the pairs of letters that have a statistically significant difference for that feature

Feature	Comparison	P_Value
nr_pix	b-a	0.0183456
nr_pix	c-a	2.37E-05
nr_pix	f-a	6.80E-08
nr_pix	f-b	0.0081954
nr_pix	g-b	0.0345732
nr_pix	d-c	0.0081954
nr_pix	e-c	0.0208854
nr_pix	g-c	5.36E-05
nr_pix	f-d	3.87E-05
nr_pix	f-e	0.0001202
nr_pix	g-f	1.58E-07
height	d-a	0.0224768
height	g-a	0.0224768
height	e-d	5.15E-05
height	g-e	5.15E-05
width	b-a	4.53E-09
width	c-a	0.0305784
width	d-a	0.0015501
width	e-a	0.0176308
width	f-a	0.0008071
width	g-a	1.92E-08
width	c-b	0.0004145
width	d-b	0.0099088
width	e-b	0.0008071
width	f-b	0.0176308
width	g-c	0.0015501
width	g-d	0.0305784
width	g-e	0.0029308
tallness	b-a	3.30E-11
tallness	c-a	0.0339928
tallness	d-a	1.91E-06
tallness	f-a	0.0019605
tallness	g-a	0
tallness	c-b	3.01E-06
tallness	d-b	0.0472873
tallness	e-b	4.21E-09
tallness	f-b	9.29E-05
tallness	g-c	8.53E-09
tallness	e-d	0.00022
tallness	g-d	0.000345
tallness	g-e	1.03E-11
tallness	g-f	2.82E-07

Feature	Comparison	P_Value
rows_with_1	b-a	4.05E-09
rows_with_1	c-a	0
rows_with_1	d-a	8.67E-09
rows_with_1	e-a	3.82E-05
rows_with_1	f-a	0
rows_with_1	g-a	1.85E-06
rows_with_1	c-b	3.97E-06
rows_with_1	f-b	0
rows_with_1	d-c	1.85E-06
rows_with_1	e-c	4.19E-10
rows_with_1	f-c	1.85E-06
rows_with_1	g-c	8.67E-09
rows_with_1	f-d	0
rows_with_1	f-e	0
rows_with_1	g-f	0
cols_with_1	b-a	1.11E-07
cols_with_1	c-a	0.000281
cols_with_1	e-a	0.035534
cols_with_1	f-a	0
cols_with_1	g-a	3.36E-06
cols_with_1	d-b	1.04E-05
cols_with_1	e-b	0.005921
cols_with_1	f-b	0
cols_with_1	d-c	0.014956
cols_with_1	f-c	0
cols_with_1	f-d	0
cols_with_1	g-d	0.000281
cols_with_1	f-e	0
cols_with_1	g-f	0
rows_with_5_plus	e-b	0.002872
rows_with_5_plus	e-c	0.000323
rows_with_5_plus	e-d	0.000982
rows_with_5_plus	f-e	0.020932
cols_with_5_plus	c-a	3.35E-05
cols_with_5_plus	f-a	0.00014
cols_with_5_plus	c-b	0.024679
cols_with_5_plus	g-b	0.002144
cols_with_5_plus	d-c	0.007613
cols_with_5_plus	e-c	0.024679
cols_with_5_plus	g-c	2.08E-08
cols_with_5_plus	f-d	0.024679
cols_with_5_plus	g-d	0.007613
cols_with_5_plus	g-e	0.002144
cols_with_5_plus	g-f	9.15E-08
one_neighbour	c-a	0.038614
one_neighbour	f-a	0
one_neighbour	c-b	0.038614
one_neighbour	f-b	0
one_neighbour	f-c	2.19E-09
one_neighbour	g-c	0.038614
one_neighbour	f-d	4.49E-10
one_neighbour	f-e	0
one_neighbour	g-f	0

Feature	Comparison	P_Value
three_or_more_neighbours	f-a	0.025364
three_or_more_neighbours	g-c	0.01094
three_or_more_neighbours	f-d	0.018242
three_or_more_neighbours	g-f	0.000987
no_neighbours_below	d-a	0.039018
no_neighbours_below	e-a	9.77E-06
no_neighbours_below	c-b	0.000106
no_neighbours_below	e-b	2.18E-10
no_neighbours_below	g-b	0.027213
no_neighbours_below	d-c	6.61E-05
no_neighbours_below	e-c	0.008584
no_neighbours_below	f-c	0.012753
no_neighbours_below	e-d	1.35E-10
no_neighbours_below	g-d	0.018739
no_neighbours_below	f-e	4.53E-08
no_neighbours_below	g-e	2.55E-05
no_neighbours_above	c-a	0.011215
no_neighbours_above	d-a	0.000607
no_neighbours_above	e-a	1.00E-08
no_neighbours_above	c-b	1.55E-06
no_neighbours_above	e-b	0
no_neighbours_above	d-c	1.89E-09
no_neighbours_above	e-c	0.002737
no_neighbours_above	f-c	0.007096
no_neighbours_above	g-c	0.026835
no_neighbours_above	e-d	0
no_neighbours_above	f-d	0.001012
no_neighbours_above	g-d	0.000214
no_neighbours_above	f-e	5.75E-09
no_neighbours_above	g-e	3.06E-08
no_neighbours_before	c-a	0
no_neighbours_before	e-a	0
no_neighbours_before	f-a	1.75E-08
no_neighbours_before	g-a	3.31E-06
no_neighbours_before	c-b	0
no_neighbours_before	e-b	0
no_neighbours_before	f-b	1.29E-09
no_neighbours_before	g-b	2.41E-07
no_neighbours_before	d-c	0
no_neighbours_before	f-c	0.000528
no_neighbours_before	g-c	3.31E-06
no_neighbours_before	e-d	0
no_neighbours_before	f-d	6.47E-08
no_neighbours_before	g-d	1.21E-05
no_neighbours_before	f-e	0.000961
no_neighbours_before	g-e	6.35E-06

Feature	Comparison	P_Value
no_neighbours_after	b-a	4.35E-05
no_neighbours_after	c-a	0
no_neighbours_after	e-a	0
no_neighbours_after	f-a	0
no_neighbours_after	g-a	0.004101
no_neighbours_after	c-b	0
no_neighbours_after	d-b	0.00119
no_neighbours_after	e-b	0
no_neighbours_after	f-b	2.19E-05
no_neighbours_after	d-c	0
no_neighbours_after	f-c	8.58E-05
no_neighbours_after	g-c	0
no_neighbours_after	e-d	0
no_neighbours_after	f-d	1.42E-12
no_neighbours_after	f-e	4.35E-05
no_neighbours_after	g-e	0
no_neighbours_after	g-f	1.61E-07
nr_eyes	c-a	1.04E-05
nr_eyes	f-a	1.04E-05
nr_eyes	g-a	0.027963
nr_eyes	c-b	3.01E-08
nr_eyes	f-b	3.01E-08
nr_eyes	d-c	5.65E-07
nr_eyes	e-c	1.04E-05
nr_eyes	g-c	9.12E-11
nr_eyes	f-d	5.65E-07
nr_eyes	f-e	1.04E-05
nr_eyes	g-e	0.027963
nr_eyes	g-f	9.12E-11
r5_c5	e-a	0.000646
r5_c5	f-a	0.012881
r5_c5	e-b	0.004981
r5_c5	g-c	0.004981
r5_c5	e-d	0.000646
r5_c5	f-d	0.012881
r5_c5	g-e	7.66E-06
r5_c5	g-f	0.00022



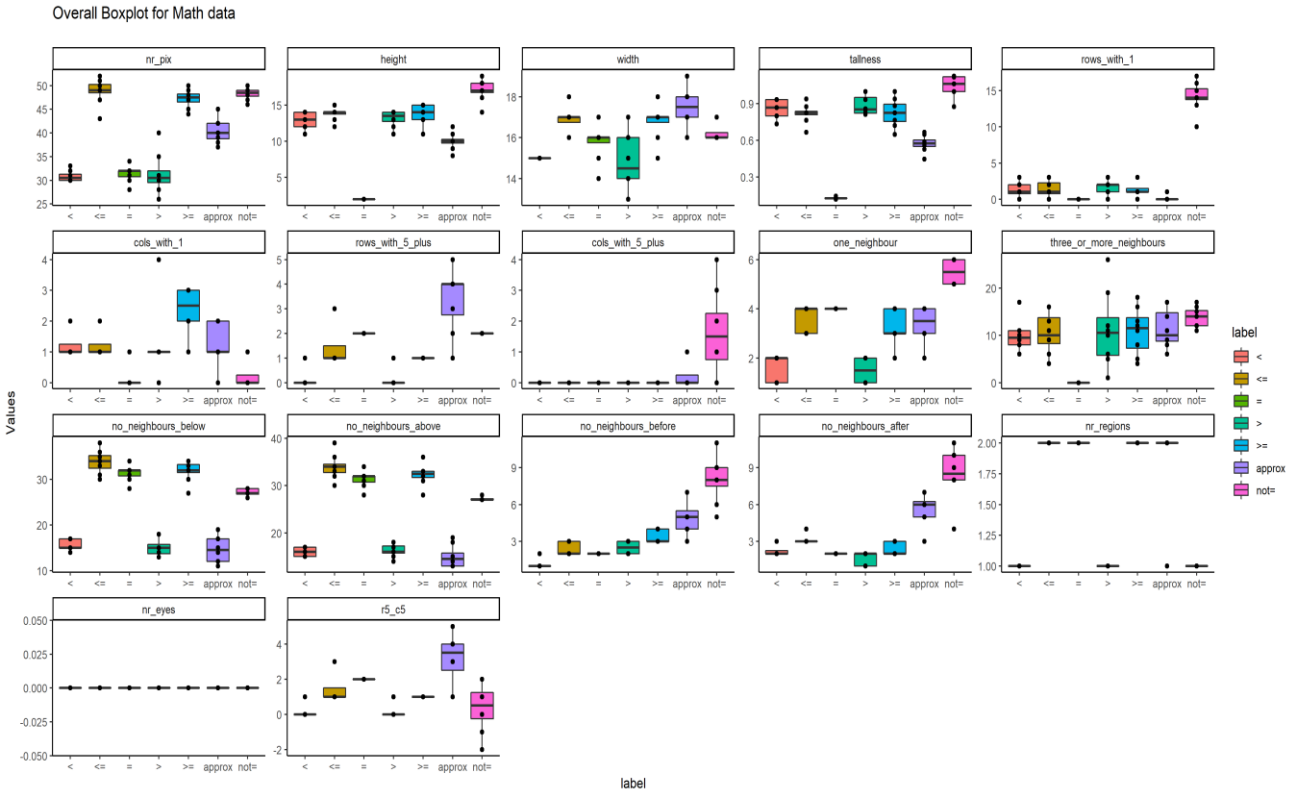
Section 3.14: For each feature, find the pairs of Math symbols that have a statistically significant difference for that feature

Feature	Comparison	P_Value
nr_pix	<=-<	0
nr_pix	>=-<	0
nr_pix	approx-<	1.28E-08
nr_pix	not=-<	0
nr_pix	=-<=	0
nr_pix	>-<=	0
nr_pix	approx-<=	3.21E-07
nr_pix	>=-=	0
nr_pix	approx-=	5.36E-08
nr_pix	not=-=	0
nr_pix	>=->	0
nr_pix	approx->	5.36E-08
nr_pix	not=->	0
nr_pix	approx->=	2.29E-05
nr_pix	not=-approx	1.35E-06
height	=-<	0
height	approx-<	0.000114
height	not=-<	4.85E-08
height	=-<=	0
height	approx-<=	5.11E-07
height	not=-<=	1.16E-05
height	>-=	0
height	>=-=	0
height	approx-=	0
height	not=-=	0
height	approx->	2.50E-05
height	not=->	2.33E-07
height	approx->=	5.11E-07
height	not=->=	1.16E-05
height	not=-approx	0
width	<=-<	0.000821
width	>=-<	0.002124
width	approx-<	5.05E-06
width	>-<=	0.000308
width	approx-=	0.002124
width	>=->	0.000821
width	approx->	1.76E-06
width	not=->	0.028756
tallness	=-<	0
tallness	approx-<	5.98E-08
tallness	not=-<	0.000352
tallness	=-<=	0
tallness	approx-<=	2.39E-06
tallness	not=-<=	1.01E-05
tallness	>-=	0
tallness	>=-=	0
tallness	approx-=	0
tallness	not=-=	0
tallness	approx->	5.20E-09
tallness	not=->	0.0032
tallness	approx->=	1.18E-06
tallness	not=->=	2.02E-05
tallness	not=-approx	0

Feature	Comparison	P_Value
rows_with_1	not=-<	0
rows_with_1	not=-<=	0
rows_with_1	not=-=	0
rows_with_1	not=->	0
rows_with_1	not=->=	0
rows_with_1	not=-approx	0
cols_with_1	=-<	0.025191
cols_with_1	>=-<	0.025191
cols_with_1	=-<=	0.025191
cols_with_1	>=-<=	0.025191
cols_with_1	>-=	0.025191
cols_with_1	>=-=	4.27E-07
cols_with_1	approx-=	0.025191
cols_with_1	>=->	0.025191
cols_with_1	approx->=	0.025191
cols_with_1	not=->=	1.59E-06
rows_with_5_plus	<=-<	0.001287
rows_with_5_plus	=-<	6.13E-06
rows_with_5_plus	approx-<	0
rows_with_5_plus	not=-<	6.13E-06
rows_with_5_plus	>-<=	0.001287
rows_with_5_plus	approx-<=	6.13E-06
rows_with_5_plus	>-=	6.13E-06
rows_with_5_plus	>=-=	0.040223
rows_with_5_plus	approx-=	0.001287
rows_with_5_plus	approx->	0
rows_with_5_plus	not=->	6.13E-06
rows_with_5_plus	approx->=	2.26E-08
rows_with_5_plus	not=->=	0.040223
rows_with_5_plus	not=-approx	0.001287
cols_with_5_plus	not=-<	9.45E-06
cols_with_5_plus	not=-<=	9.45E-06
cols_with_5_plus	not=-=	9.45E-06
cols_with_5_plus	not=->	9.45E-06
cols_with_5_plus	not=->=	9.45E-06
cols_with_5_plus	not=-approx	0.000203
one_neighbour	<=-<	6.70E-08
one_neighbour	=-<	5.76E-10
one_neighbour	>=-<	8.00E-06
one_neighbour	approx-<	1.64E-06
one_neighbour	not=-<	0
one_neighbour	>-<=	1.36E-08
one_neighbour	not=-<=	3.31E-07
one_neighbour	>-=	1.19E-10
one_neighbour	not=-=	3.83E-05
one_neighbour	>=->	1.64E-06
one_neighbour	approx->	3.31E-07
one_neighbour	not=->	0
one_neighbour	not=->=	2.78E-09
one_neighbour	not=-approx	1.36E-08

Feature	Comparison	P_Value
three_or_more_neighbours	=-<	0.000903
three_or_more_neighbours	=-<=	0.000437
three_or_more_neighbours	>-=	0.000143
three_or_more_neighbours	>=-=	0.000251
three_or_more_neighbours	approx-=	0.000119
three_or_more_neighbours	not=-=	2.48E-06
no_neighbours_below	<=-<	0
no_neighbours_below	=-<	0
no_neighbours_below	>=-<	0
no_neighbours_below	not=-<	0
no_neighbours_below	>-<=	0
no_neighbours_below	approx-<=	0
no_neighbours_below	not=-<=	5.88E-07
no_neighbours_below	>-=	0
no_neighbours_below	approx-=	0
no_neighbours_below	not=-=	0.002393
no_neighbours_below	>=->	0
no_neighbours_below	not=->	0
no_neighbours_below	approx->=	0
no_neighbours_below	not=->=	0.000748
no_neighbours_below	not=-approx	0
no_neighbours_above	<=-<	0
no_neighbours_above	=-<	0
no_neighbours_above	>=-<	0
no_neighbours_above	not=-<	0
no_neighbours_above	>-<=	0
no_neighbours_above	approx-<=	0
no_neighbours_above	not=-<=	4.27E-08
no_neighbours_above	>-=	0
no_neighbours_above	approx-=	0
no_neighbours_above	not=-=	0.000843
no_neighbours_above	>=->	0
no_neighbours_above	not=->	0
no_neighbours_above	approx->=	0
no_neighbours_above	not=->=	3.49E-05
no_neighbours_above	not=-approx	0
no_neighbours_before	>=-<	0.000387
no_neighbours_before	approx-<	2.58E-09
no_neighbours_before	not=-<	0
no_neighbours_before	approx-<=	2.70E-05
no_neighbours_before	not=-<=	0
no_neighbours_before	approx-=	1.71E-06
no_neighbours_before	not=-=	0
no_neighbours_before	approx->	6.64E-05
no_neighbours_before	not=->	0
no_neighbours_before	approx->=	0.021044
no_neighbours_before	not=->=	7.85E-12
no_neighbours_before	not=-approx	1.71E-06

Feature	Comparison	P_Value
no_neighbours_after	approx-<	4.21E-07
no_neighbours_after	not=-<	0
no_neighbours_after	approx-<=	0.000177
no_neighbours_after	not=-<=	0
no_neighbours_after	approx-=	7.25E-08
no_neighbours_after	not=-=	0
no_neighbours_after	approx->	5.24E-09
no_neighbours_after	not=->	0
no_neighbours_after	approx->=	1.01E-06
no_neighbours_after	not=->=	0
no_neighbours_after	not=-approx	1.39E-05
nr_regions	<=-<	0
nr_regions	=-<	0
nr_regions	>=-<	0
nr_regions	approx-<	0
nr_regions	>-<=	0
nr_regions	not=-<=	0
nr_regions	>-=	0
nr_regions	not=-=	0
nr_regions	>=->	0
nr_regions	approx->	0
nr_regions	not=->=	0
nr_regions	not=-approx	0
r5_c5	<=-<	0.037626
r5_c5	=-<	0.001288
r5_c5	approx-<	1.61E-07
r5_c5	>-<=	0.037626
r5_c5	approx-<=	0.007598
r5_c5	>-=	0.001288
r5_c5	not=-=	0.007598
r5_c5	approx->	1.61E-07
r5_c5	approx->=	0.000193
r5_c5	not=-approx	1.26E-06



Conclusion:

This investigation surprised me in that I assumed that nr_eyes and nr_regions would be by far the biggest distinguishers between symbols as they are categorical factors – a symbol either has an eye, or it doesn't, a symbol either has 1 or 2 regions. However, this assumption was proved wrong, possibly because these features are so low in their variance that one outlier/error can throw of the entire variance of the feature data. The features that proved the most consistently important were the no neighbours before/after/above/below features as they seem the best at identifying the vertical/horizontal orientation of a symbol. If I had more time to do this project then perhaps I would attempt to create features based on identifying horizontal and vertical lines in a shape, using the code in these features as inspiration. Also, another regret I have is that I should have not assumed normality for each feature in each dataset – some of the feature data was extremely skewed in these smaller samples and I could have maybe more accurately measured the data if I had done a randomisation test for the F- scores and decided the statistical significance of the features that had skewed distributions from there. Another issue I have is with my code in section 3; I could have possibly used functions to simplify the code as much of the code to create the visualisations was often recycled, unfortunately I wasn't familiar enough with R to be confident of going down such a route.