CA274 – Creating model for random 2's and 4's – Adam Tegart

In this assignment I will be applying what we have covered about generating random digits for 1's and 7's in CA274 to complete similar work on models for 2's and 4's.

Section 1 – Creating the initial models

To get a general idea of what the real digits looked like I flicked through some of them, these can be found below for the 2's.

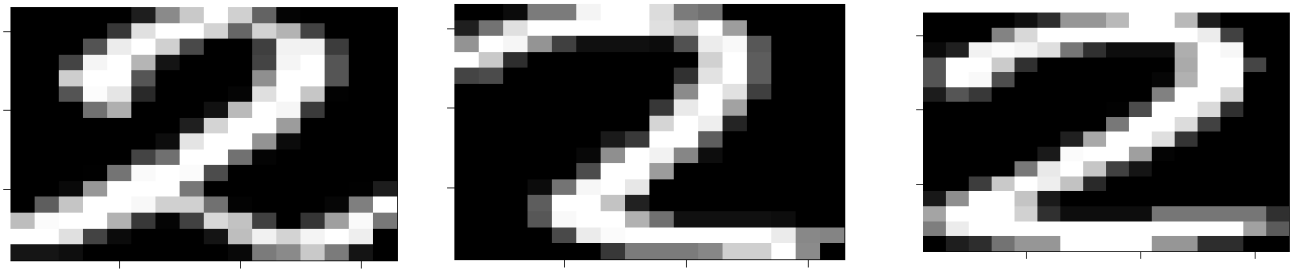


Fig 1.1 2's from real numbers (2447, 2452, 2461 from left to right)

There were some very strange looking 2's but I thought if I get a general model working it could be tuned later and would still work reasonably well with outliers.

I also decided it might be worth looking at the most general 2, an average of all of the real 2's. This can be seen below. At the time it seemed like a good idea, but when it came to finding nearest neighbours, I found this was not the case.

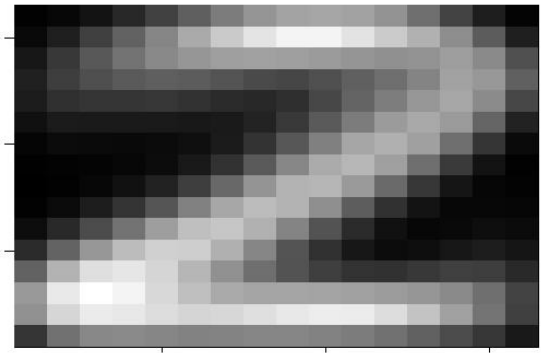


Fig 1.2 The average 2 which I will base my model around.

The model itself was started with the upper stroke, I started in the top left and kept a reasonably tight angle. I made use of an if statement so that if the x value was low and the y value was below 12 or so the chances of getting a steeper angle was higher, to match that seen above in the example images.

The rest of the model made use of a connecting diagonal stroke that took the end point of the upper stroke into account when finding the start point. The lower stroke was not given too wide of an angle. I also made use of trigonometry to ensure the connecting stroke did not overlap with the upper stroke or come to close, or that the connecting stroke started or ended outside of the 16x16 as no 2's I had seen had that shape.

In my design I tried not to implement too much variation, which was the downfall of the model, I generalised too much, and the model did not match well with outlier and was not even amazing with those it matched well with.

Making changes to the angles of the first and connecting stroke can have a large impact on the model and is I believe the reason I need to add more variation, they need to cover many of the different possibilities that are seen in the real digits. I settled on the model that I have as I felt it consistently had a rough 2 shape (excluding the upper strokes curve as I could not implement that with a line) and could fit the real digits reasonably well as it made use of the average of them all.

In a similar way to the 2's, I looked at several real 4's to get an idea of how they were shaped, got the average 4 and used both of those to create my general model which again probably could have had more variation.

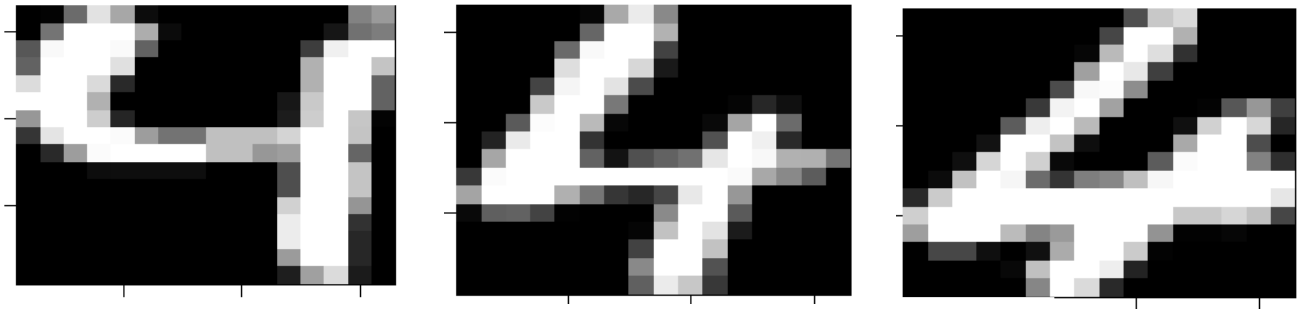


Fig 1.3 4's from the real digits (4443, 4468, 4498 from left to right)

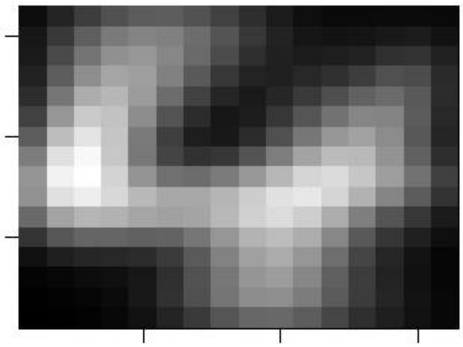


Fig 1.4 The average 4 which I will base my model around.

The model for the 4 worked out better than I had expected. I started with the right stroke and set the start of the stroke in the middle to bottom right. The stroke horizontally took the middle of the right stroke into account when deciding the y and x. I set it so that it was more likely to move to the right on the x so that it crossed on both side of this first stroke. The final stroke started in a box around the end of the cross stroke. There seems to be a relationship between the angle of the left and right stroke, so I gave them a little bit of dependence so that the angles were reasonably similar in the model.

I only had to make sure that the final stroke did not start outside of the 16x16 grid. In the same way as the last model changes to the angles on any of the first 2 strokes can greatly affect the general model, so I kept them somewhat tight. Other than that, the rest of the parameter can be altered relatively freely without making any big changes to the model. I intend to adjust these later when I can see how the model fails to fit certain real digits.

Section 2 – Fitting the model to the real digits

After calculating the distances between the 10,000 2's generated from my model and the real 2's I created a histogram to view the distances. This can be seen below, and I must admit, I was a little disappointed. It seems that very few were real 2's that were matches very well by the model. Most of the matches lie in the 1.5 – 2.5 million range.

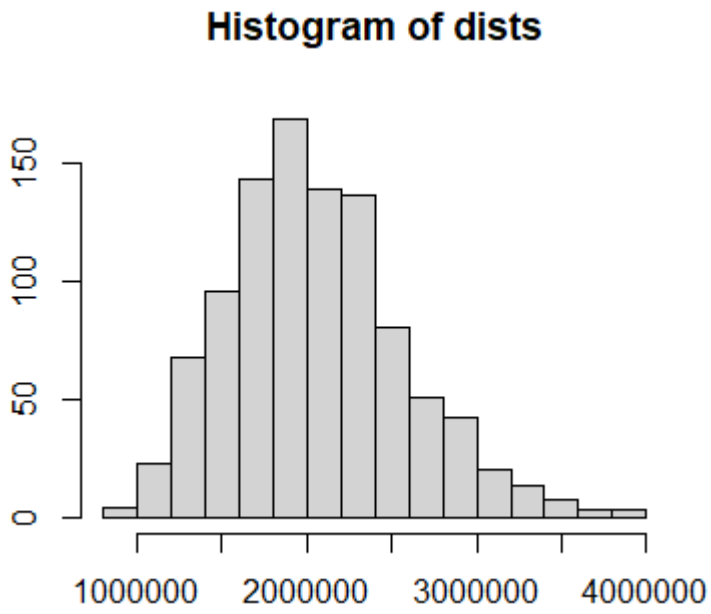


Fig 2.1 A histogram showing distances from real 2's to nearest generated 2.

Having a look at the good fits I can see that these are the types of digits that my model was based upon. The upper stroke of my model seems to be quite stiff most of the time, so maybe I should make changes there.

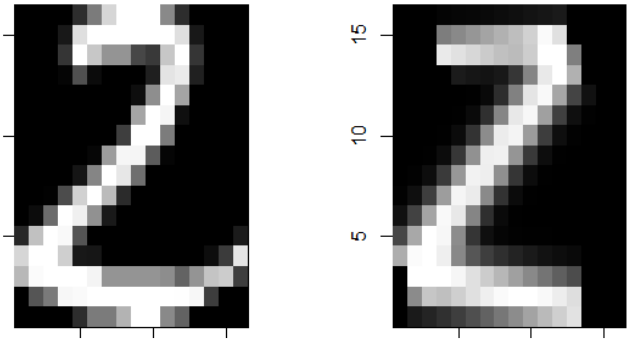


Fig 2.2 Real 2 and the nearest generated 2 to it for a good match (Real is on the left).

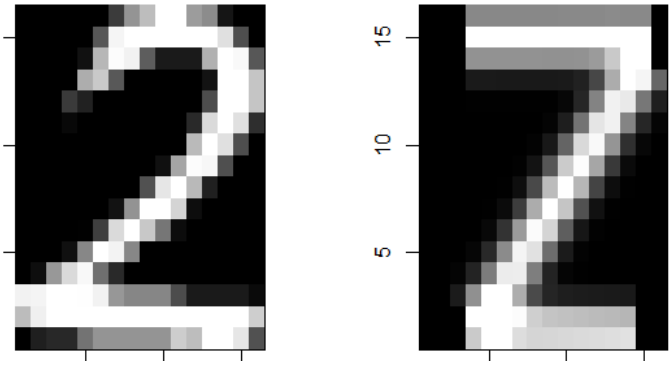


Fig 2.3 Real 2 and the nearest generated 2 to it for very bad matches (Real is on the left for both).

Regarding bad matches, it seems that there are multiple places the model could be improved.

It can be seen from the above bad matches that some of the 2's do not follow the shape of the average, which is to be expected I suppose when you work with an average of 1,000 images. So maybe I could add a 4th stroke in to create that little loop at the bottom? Or maybe I just adjust the parameters to fit the shape of these better and leave more variation in the model so we can get more reasonable fits.

I would like to look at the bulk in the model that are modelled okay, specifically in the 1.5-2.5 million range, as these should ideally be being matched better, so maybe there is something I missed in my model that could improve the fit.

What can be seen above is that most of the variation between the two of these is in the upper stroke. This is

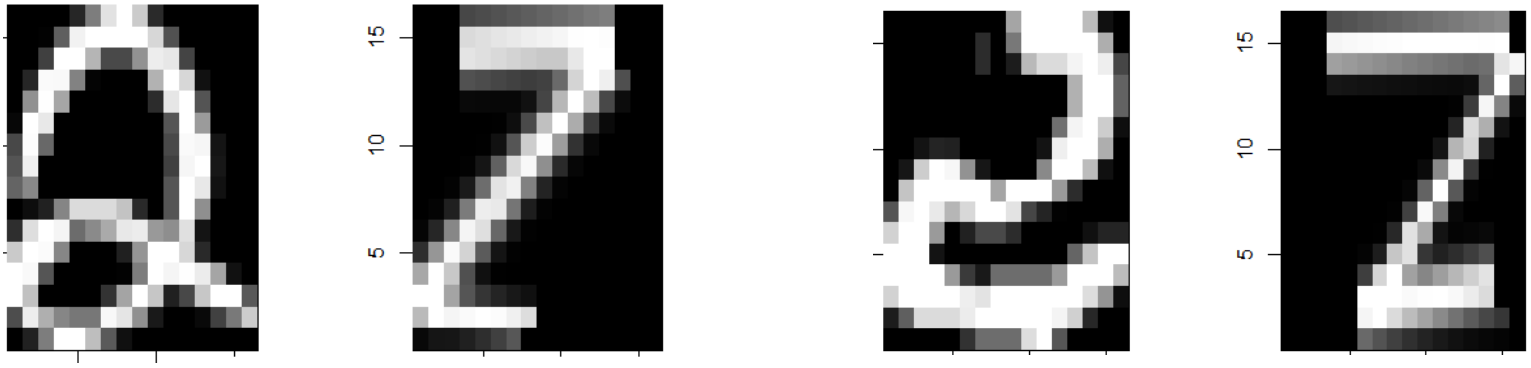


Fig 2.4 Real 2 and the nearest generated 2 to it for average matches (Real is on the left).

a common trend across many of these 2's. What I think may help is altering the starting position of the upper stroke and incorporating a 4th stroke connecting the upper stroke and diagonal so that I can have more of a loop in the 2. I know that this will add variation, but I hope that it can still improve the model by providing a better fit to a large majority of these 2's.

Having changed the model and again looked at the distances between real 2's and the nearest generated digit I must say I am quite pleased with the results. Having added another stroke and altered some of the other parameters I have created a model that has a much better accuracy than that of the old one (below on the left).

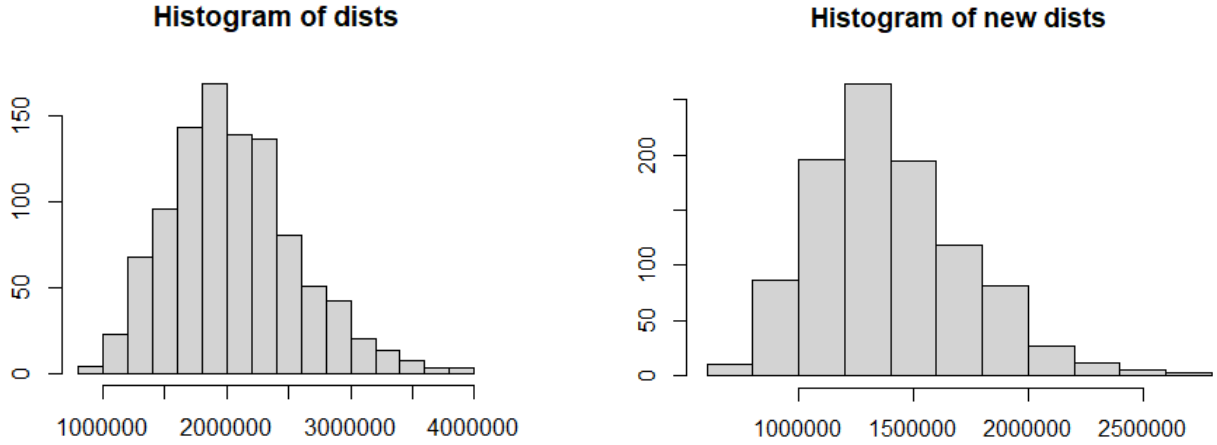


Fig 2.5 Comparison of histograms of distances for old(left) and new(right) 2 models.

As you can see above, the new model has about 94% of the real digits with a match < 2,000,000 which was closer to the 50% mark in the old model as can be seen above. This is a major improvement.

Below see some of the best and worst matches from the new model. The real digit is on the left in all of them.

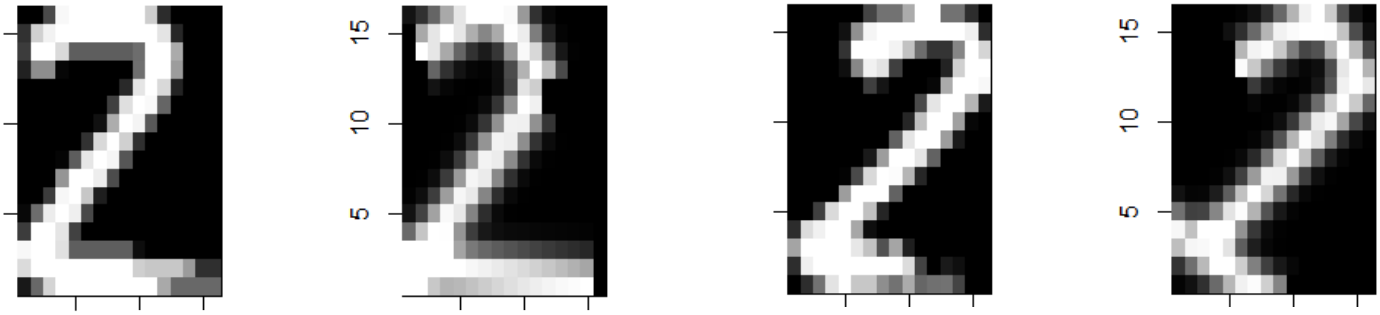


Fig 2.6 Best matches for new 2 model, distances of <700,000

Now let us look at the bad matches within the new model, these can be found below.

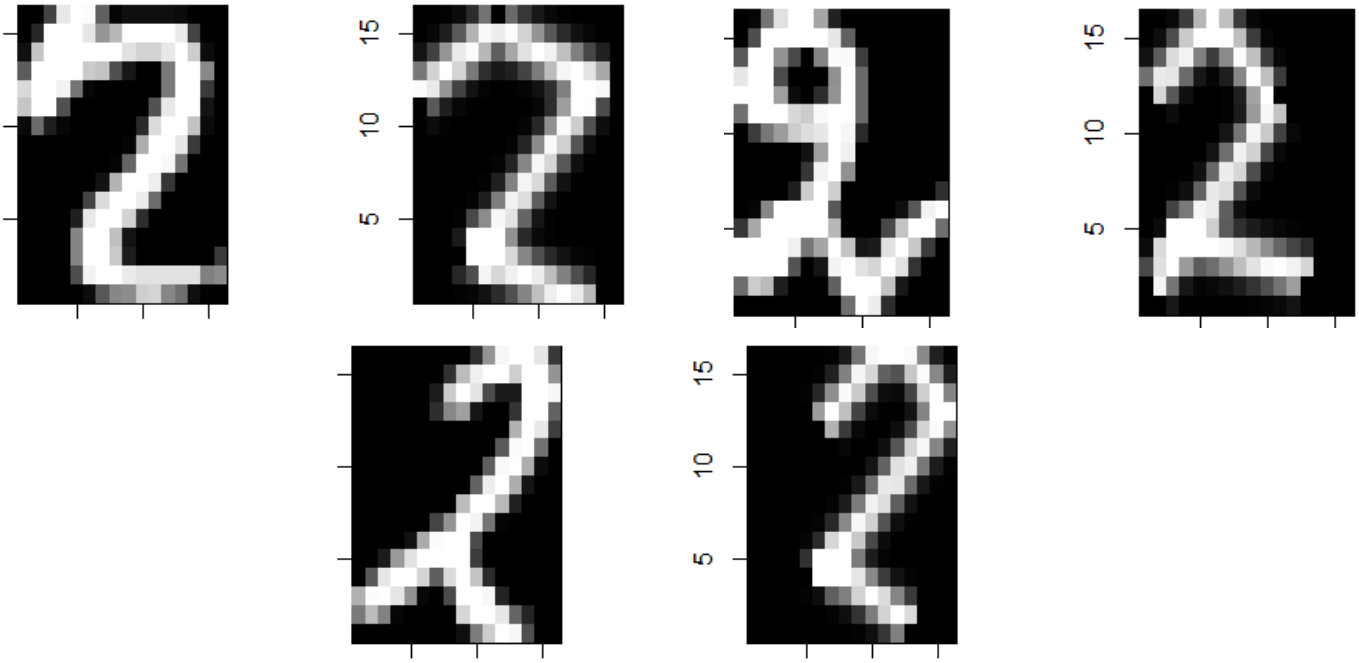


Fig 2.7 Worst matches for new 2 model, distances of > 2,500,000

In all 3 of the cases the model did a reasonable job of getting the rough shape. The first real 2 has an extra upper stroke that makes the top stroke very thick and hard to match. The second has a closed loop at the top which is very unlike a 2 and not something my model can accurately match. The last example the model has done a great job of getting the shape but has missed the last stroke, this could be solved possibly via adding another stroke or changing the parameters so the last stroke starts further up the diagonal stoke and the diagonal stroke goes further. Although I believe this would be pointless as it would add a lot of variation for a very specific test case which I believe to be an outlier and not a typical 2 one would encounter.

Overall, I am very pleased with the accuracy of my new model for generating random 2's.

In the same way as the 2's. I generated a histogram of the distances from the real 4's to the nearest generated 4, this can be found below.

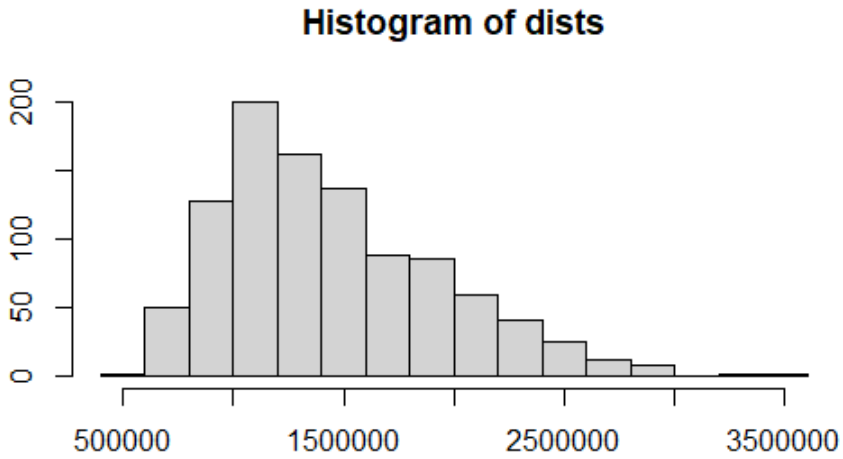


Fig 2.8 A histogram showing distances from real 4's to nearest generated 4.

As can be seen, the model is reasonably accurate. A large amount of the distances are < 1,500,000 which is very good. However, I do believe that I can change the model to get a better accuracy. Below see some of the good matches, the model seems to be doing a pretty great job!

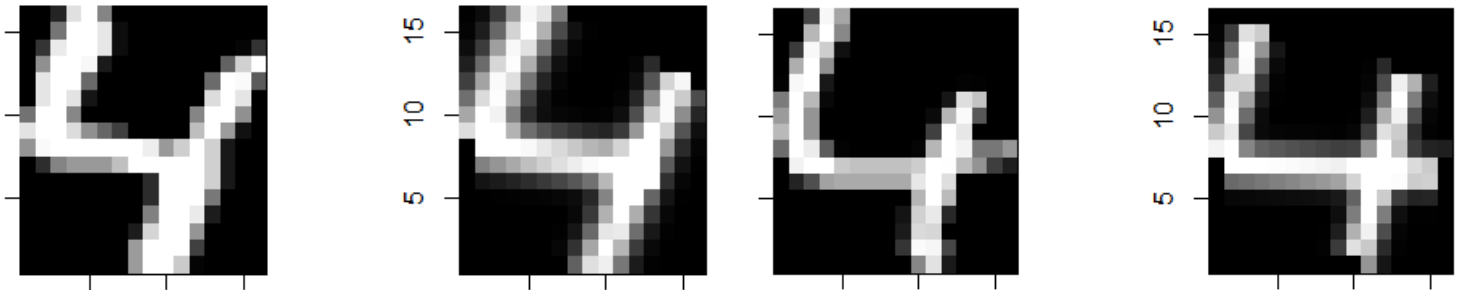


Fig 2.9 Real 4's and the nearest generated 4's to them (great matches, <750,000. Real on left).

Now we will look at the matches that were not so good and decipher why exactly that might be.

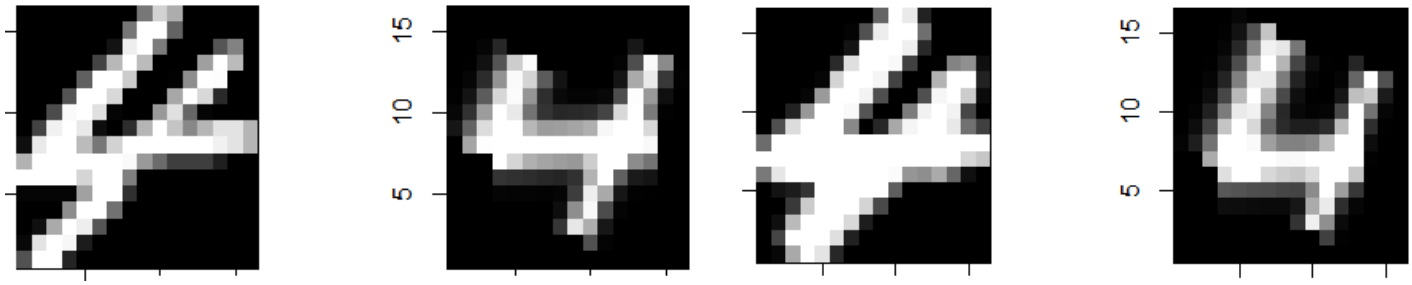


Fig 2.10 Real 4's and the nearest generated 4's to them (bad matches, >3,000,000. Real on left).

From looking at the above it can be said that the left stroke could be longer, and the right stroke could start more leftward as currently it can only start in the lower right-hand corner of the 16x16 as is clear from the left example. In addition, some of the stroke widths are bigger than what the model can currently do, so maybe this is a part of the model to improve upon. Flicking through other groupings of dists this seems to be the case in many of the example with a reasonably high dist.

Looking at the histogram for the new model it is clear there is a good improvement, for comparison the old histogram is below on the left. As you can see the model has gotten significantly more accurate.

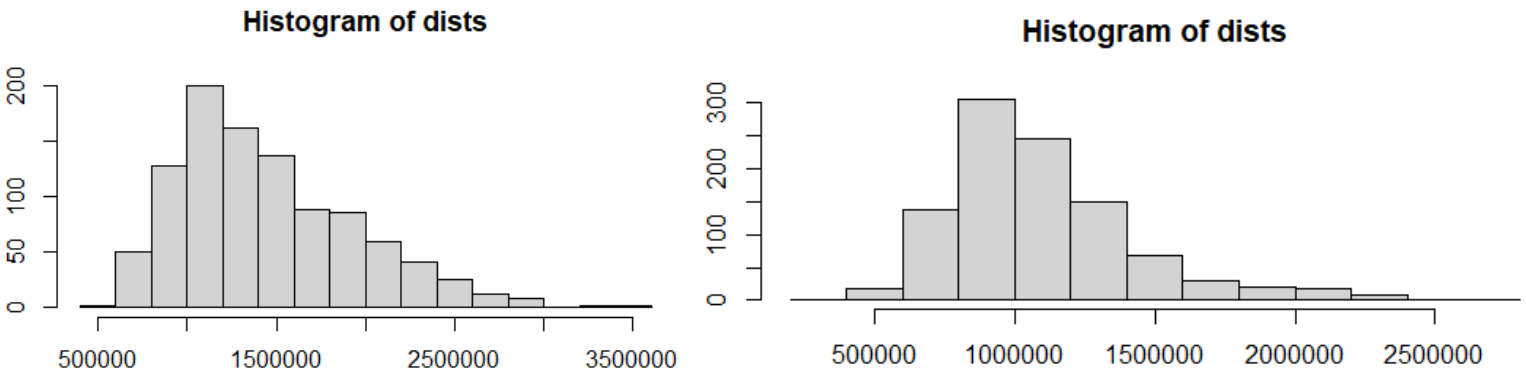


Fig 2.11 Comparison of histograms of distances for old(left) and new(right) 4 models.

The new model has only 29 distances >2,000,000, meaning that 97.1% of the real digits have a generated digit within 2 million of it, which is very good compared to the previous model which had about ~80-85% by the looks of the histogram. Below are some of the goof matches from the new model.

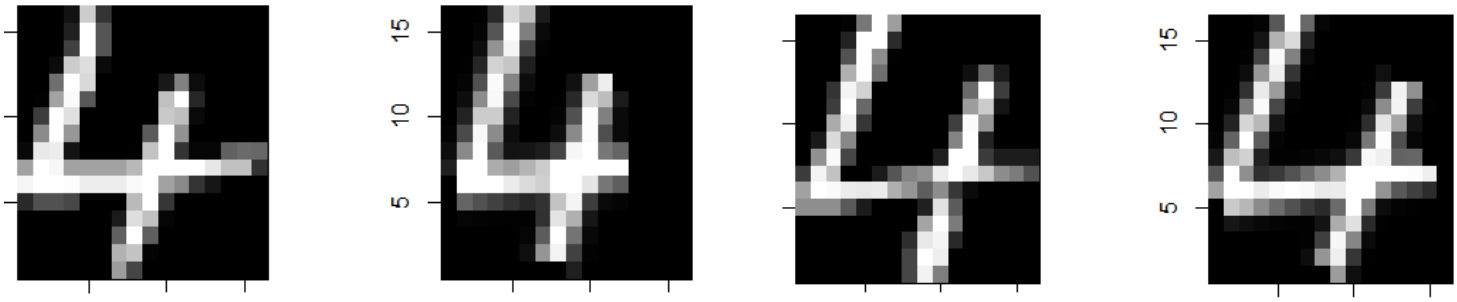


Fig 2.12 Best matches for new 4 model, distances of <500,000

As you can see, the new model has generated some random digits that are very similar to that of the real digits. It has of course reduced the severity of the bad matches, however there were certain real 4's in the dataset that I don't believe I would like my model matching well. These can be seen below.

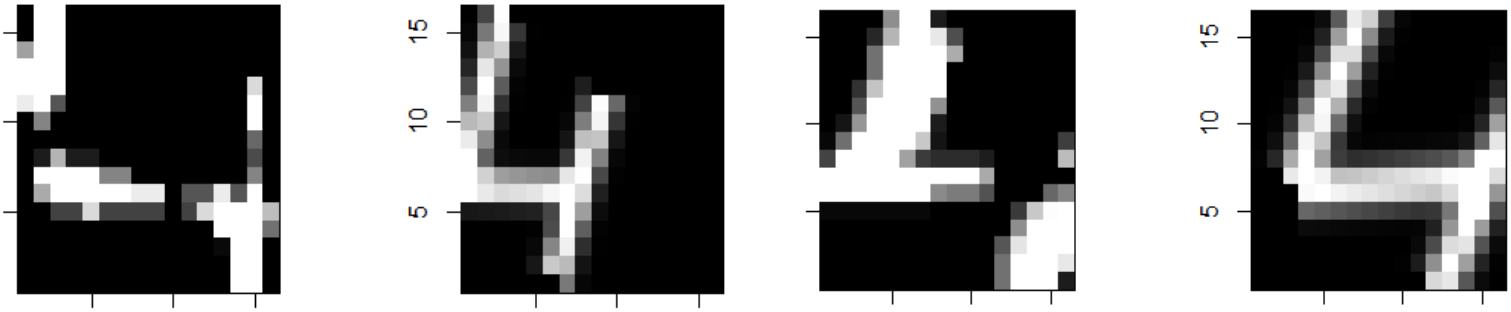


Fig 2.13 Bad matches for new 4 model (outliers), distances of >2.250,000

As I previously said, these outliers are not the type of real digits that I want to be generating close digits for. So, excluding these there are only a handful of matches that are not ideal, and these are mainly because of weirdly shaped strokes or very thick strokes. Both of these things are extra variation that I do not want to be including in my model to satisfy a certain few select real digits.

Overall, I am happy with the accuracy of both models, and I believe the changes I made including adding a stroke into the model for 2's and choosing not to for the 4's has made both models noticeably better.

Section 3 – Fitting the models to the other digits and then simultaneously

Fitting the random 2's to the real 4's produced the distances histogram below. The smallest distance is 1,500,000 and the largest is ~4,000,000 by the look of the histogram. The model for 2's fits the real 4's much worse than the model for 4's. Below you can also find a table that has a count of how many times a random 2 fit a real 4 better than a random 4 and the indexes of these digits.

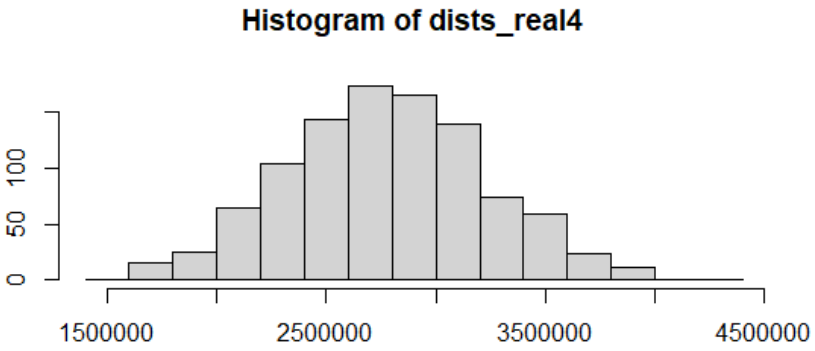


Fig 3.1 Histogram of distances for real 4's and random 2's.

```
FALSE  TRUE
  987    13
```

Fig 3.2 Table of 4's that would be misclassified.

```
114 127 139 272 326 416 430
556 587 596 633 646 934
```

Fig 3.3 Indexes of 4's that would be misclassified,

As you can see, some of the 4's would be matched better with a random 2, why exactly is this the case. Well, let us look at these digits all side by side. So first we have the real 4 on the left, the nearest random 4 and then the nearest random 2.

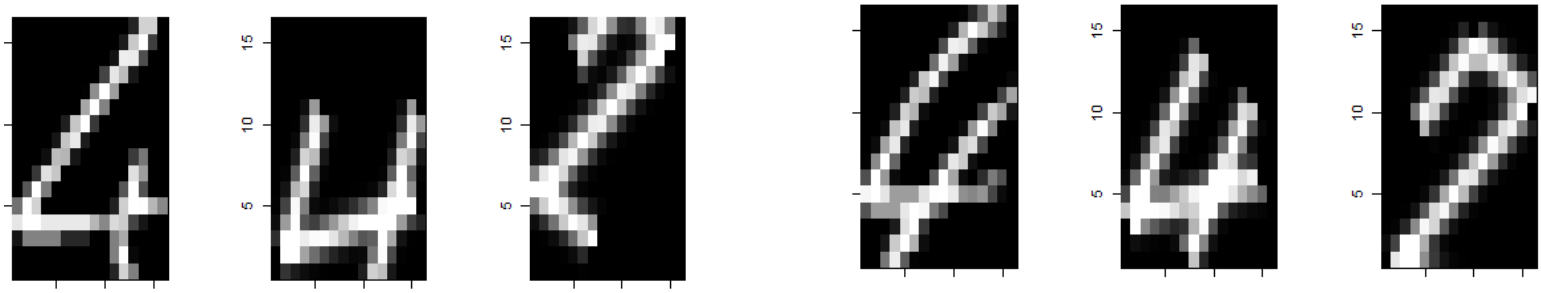


Fig 3.4 Examples of why 4's were better matched with random 2's.

Looking at the above examples and the other 11 cases where a real 4 was better matched with a random 2 highlights these 2 issues. The first issue is in the left example, the diagonal stroke of the 2 matches the left stroke of the 4 better than a random 4. The solutions to this would be to make sure the diagonal stroke was not as high or as steep maybe. Another option would be to alter the angle and length of the left stroke on the random 4's.

The second example shows that the random 4's don't get far enough into the bottom left corner with the right stroke of the 4. In addition, the left stroke would need to be longer and a little more clockwise. The diagonal stroke of the 2 matches perfectly with the right stroke of the real 4. Even with the increase in distance because of the connector on the top of the random 2 the fact the right stroke is matched makes it more accurate than the random 4. So maybe the model for random 4's needs to be adjusted slightly.

With regard to the model for 4's being fitted to the real 2's, the following histogram was produced. As can be seen, the smallest value is greater than 2,000,000 and the largest is ~5,200,000. It can also be seen that only two real 2's were matched better with a random 4 than a random 2, which is great to see.

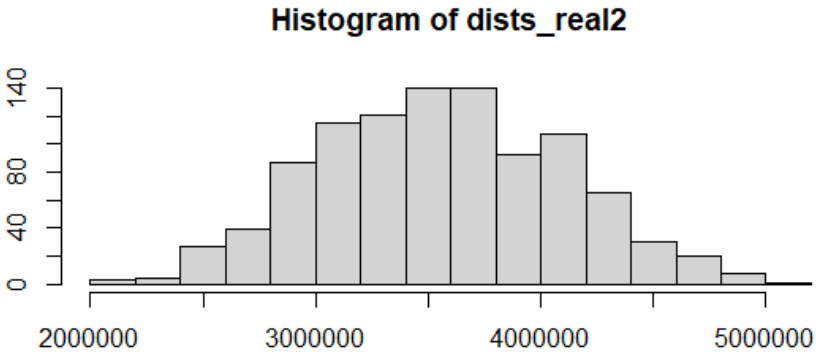


Fig 3.5 Histogram of distances for real 2's and random 4's.

```
FALSE  TRUE
  998     2
```

Fig 3.6 Table of 2's that would be misclassified.

```
701 980
```

Fig 3.7 Indexes of 2's that would be misclassified, outliers.

Having a look at these two real 2's like before, with the real on the left, nearest random 2 in the middle and nearest random 4 on the right, may shed some light on these potential misclassifications.

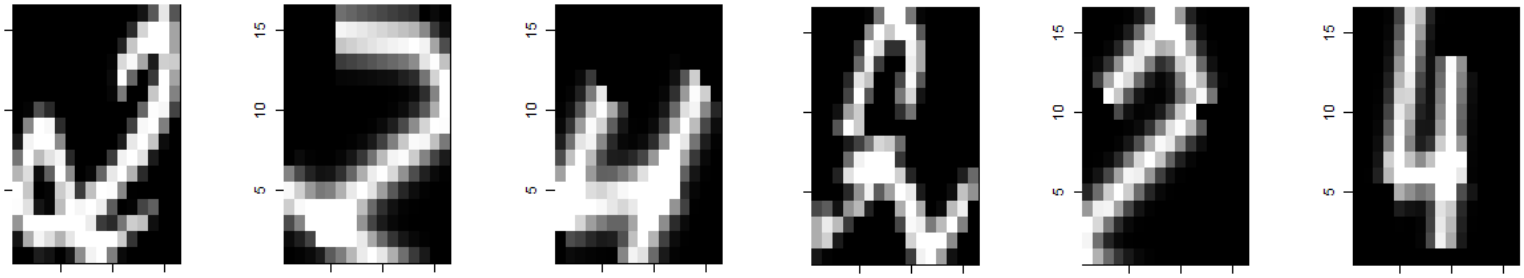


Fig 3.8 Examples of why 2's were better matched with random 4's.

It can be seen above that the model for random 4's makes little better job at being very similar to the real 2 than the model for random 2's, in fact the difference between the distances of the above is ~150,000 in both. These real twos have strange bottom curves. The random 4's do a good job of covering this area and not being too wrong. The strange starting position of the left real 2 is difficult for the random 2 to match and the curve at the bottom is not something I designed the model to work well with as it was an extreme case. The solution to this I believe would only involve making changes to the model for 2's, possibly moving the start position of the upper stroke more to the right and allowing it to have a steeper angle. In an extreme case where you wanted the model to be very accurate another stroke could be added in and the number of generated random digits could be increased to compensate for the increase in variation.

Following these comparisons of switching the model the join confusion matrix for classifying the real digits using 10k of each generated digit is shown below. As can be seen, it is identical to that of the previously shown tables where the other model would fit a set of real numbers.

	labels	
results	2	4
2	998	13
4	2	987

Fig 3.9 Confusion matrix for classifying the 2's and 4's.

As previosuly mentioned, there are changes I may make to the model for 4's, but the cases that are misclassified for the 2's is due to poor real digits that I would be happy enough with my model not accurately genreating digits for. In the case of the 4's though I intend to change the starting position of the right stroke to be more to the left and increase the length of the left stroke and its angle also. Although, even if these changes make little difference, I am happy with an accuracy of 99.25% for these models at the moment, and as they say, "Perfect is the enemy of good", I don't want to be tailoring the models to very rare scenarios and losing accuracy in common ones.

So to sum up the changes, I widened the width of the interval for the start of the rigth stroke of a 4, so they can reach into that bottom corner well. I moved the interval for the length of the left stroke up slightly, and allowed the angle to lean a bit more in the clockwise direction.

It seems that these changes made it less likely for the 2's to match with the 4's, so now there is only one error classifying 2's and it is shown below. It is due to the input being noise and missing parts of a stroke. The histogram of dists for the new 4 model is shown below too. The model has become more general and has lost some really good matches as a result, but has reduced the really bad matches.

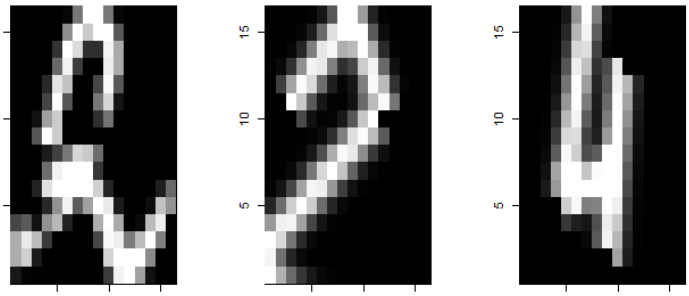


Fig 3.10 2 that is still being misclassified.

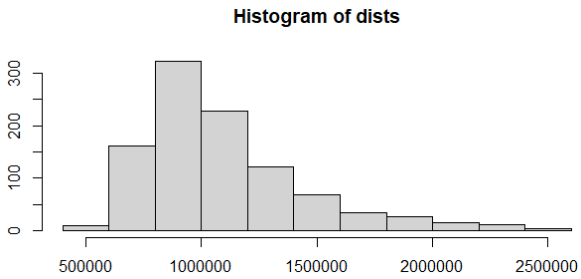


Fig 3.11 Histogram of dists for new model of 4's

The new confusion matrix can be seen below. It has increased in accuracy slightly, the indexes that were incorrectly classified the last time for the 4's have changed slightly, several are now being classified correctly and some that were previously correct are now incorrect. The number has stayed the same however and the accuracy has gone from 99.25% to 99.3% which is negligible but was worth the attempt to make the model more accurate. So, I will stick with this new model for 4's for now. One possible solution would be to make the 2's generated less like a 4 by changing the angle of the diagonal stroke, but for now 99.3% is perfect.

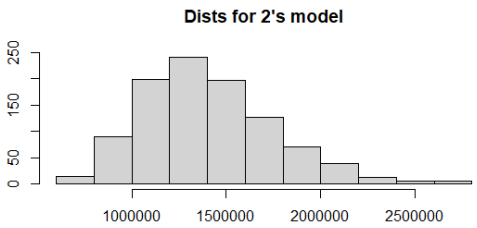
	labels	
results	2	4
2	999	13
4	1	987

Fig 3.12 Confusion matrix for classifying digits with new model.

Section 4 – Visualizing the distributions of the parameter values

To record the parameter values I had to generate all the digits again and same them as they were being created. So, to ensure the model was still accurate I ran the findnns and classify again with the whole set to ensure similar results. As can be seen below in that confusion matrix, the results are identical and the histograms of distances for both are also shown which can be seen to be like the previous ones. One of the very poorly classified 4's has become slightly worse but the classification % is still the same.

results	labels 2	4
2	999	13
4	1	987



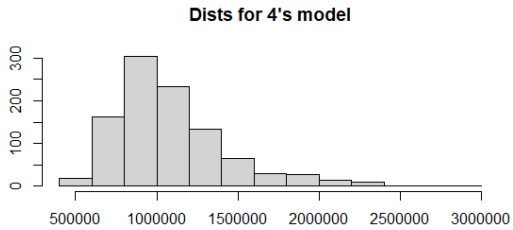
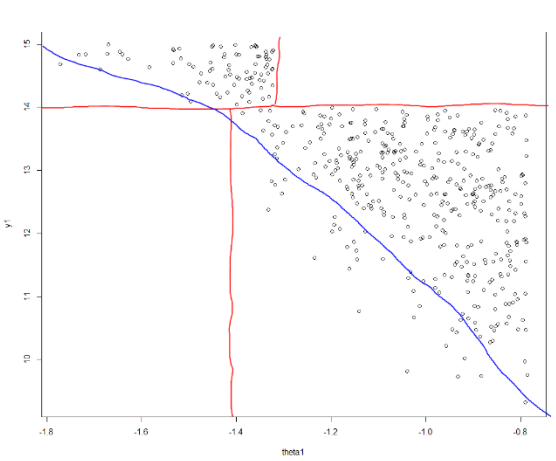


Fig 4.1 Confusion matrix for newly generated digits and histograms of dists for 2's and 4's, identical to last.

Starting with the parameters of the 2's, if we take a good fit to be <2,000,000 roughly, we can see how these parameters are distributed to each other. Looking through distribution I found that some parameters could do with changing, these can be seen below.



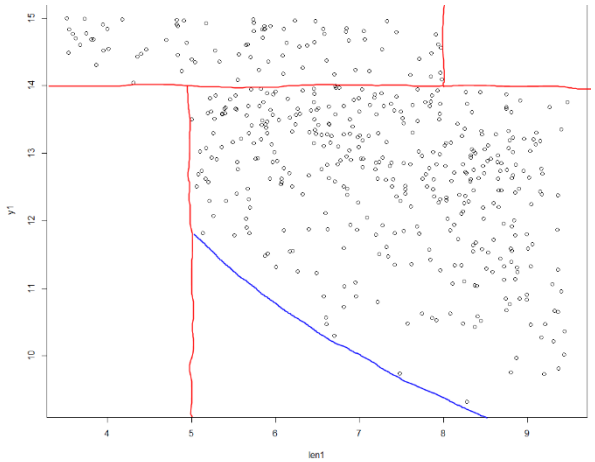


Fig 4.2 Scatter plots for Theta 1 against y1 on the left and len1 against y1 on the right.

What stood out to me about these plots was the groupings of points and empty space. Both distributions look strange because some of the parameters were dependent on others. The y-value for the first stroke determined the ranges for the angles and length. The thought process behind this was to allow for a steeper angle when at a lower y and for the length to be longer as a diagonal distance from a lower y-value must be longer than a horizontal one at a higher y to reach the same point. This is why there are 2 boxes outlined in red where points can exist. As can be seen in the left distribution, a roughly linear line can be drawn into the distribution and majority of good fits lie above it. This I believe is because at higher y-values I had an angle that was heading down to the right or just horizontal to the right, it seems that there were not many good fits when the angle went down to the right so I may increase this value. As the y-value decreased the angles that caused good fits dropped, so that means they moved anti-clockwise and became steeper. This is not really something I can alter using runif generation I do not think, but things like this will be picked up when using PCA on parameters in the next step. Although, I can lower the range of theta when y > 14 as I believe there are still some good fits to be found to the right of the upper box.

For the left distribution, there are no good fits below the blue line. That means that at low y-values low lengths do not produce good fits, which is what I expected to be the case, but it is nice to have evidence now. This is not something I can change within the parameters easily, so I will leave it to PCA.

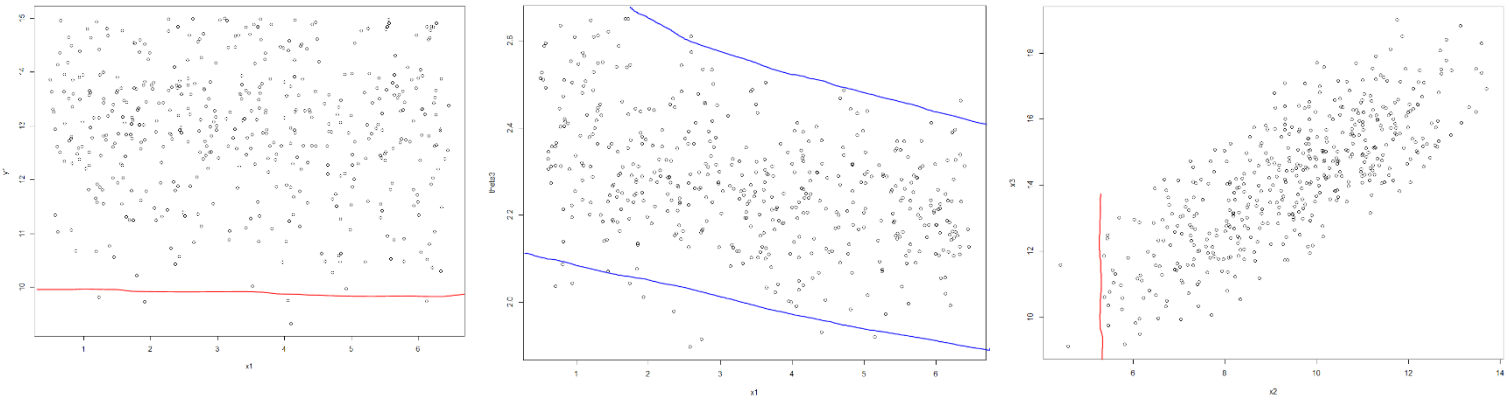


Fig 4.3 Scatter plots for x1 against y1, x1 against theta3 and x2 against x3.

Looking at the distribution of x1 against y1 above, values of y below 10 rarely give a good fit. So maybe I can bring the lower bound up to 10 to get some better fits. Looking at the other distribution of x1 against theta3 I found that for small value of x an angle for the diagonal stroke closer to a vertical line works well, as it keeps the 2 over to the left side and has adequate room for the bottom stroke. In the situation where x is bigger, and the beginning of the diagonal is over to the right of the 16x16 a vertical stroke would mean the bottom stroke begins in the lower right of the 16x16 and so does not mimic the shape of a typical 2. In the last distribution of x2 against x3 values of x2 lower than about 5.5 rarely are good fits. So, I will increase the bottom of my range for x2 slightly to get better fits.

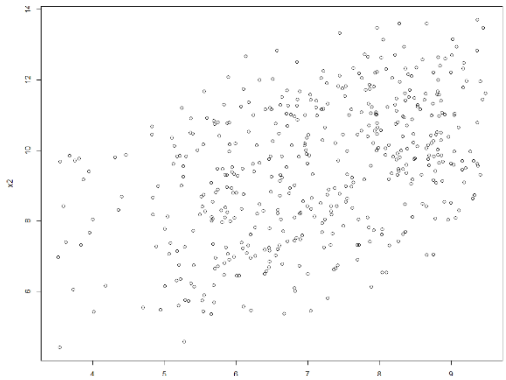


Fig 4.4 Scatter plot for len1 against x2.

Above the distribution of len1 and x2 can be seen. There is a clear dependency, but the issue here is that through my parameter selection I have added some of these dependencies in earlier parts of the digit generation. All the x and y values after the first take the end point of the last into consideration in the range of the next. In this way the length of the first stroke causes a natural increase in the x value of the start position of the second stroke. It is for this reason I must consider the relationships carefully. In this scenario though, under the distribution we have an instance where the length of the first stroke was large enough, but the start of the second stroke is small. So, the first stroke has a small value of x, a small angle (steep, possibly as far vertical as my range allowed) and a large length. This itself does not mimic the shape of a normal 2 and so it is understandable that it does not cause good fits. Above the line we have high x-values and a low length, so the line would want to be essentially horizontal with a high x-value for the start of the first stroke. In the same way as before, this would not be a shape that many of the 2's share and so I believe these dependencies are justified and need not be changed. This is something that I cannot make less common in my parameter generation, but in the next step involving PCA these dependencies will be used to improve the generation of parameters.

Regarding relationships, because of these dependencies I introduced into my model to make the 2's seem more natural with regard to stroke start and end points, all the x and y values are related and have if looking at two of the same variables, i.e., x2 and x3, they have a linear relationship as an increase in one causes an increase in the other. When looking at an x and a y, i.e., x3 and y3, they have a roughly bi-normal distribution as can all be seen below.

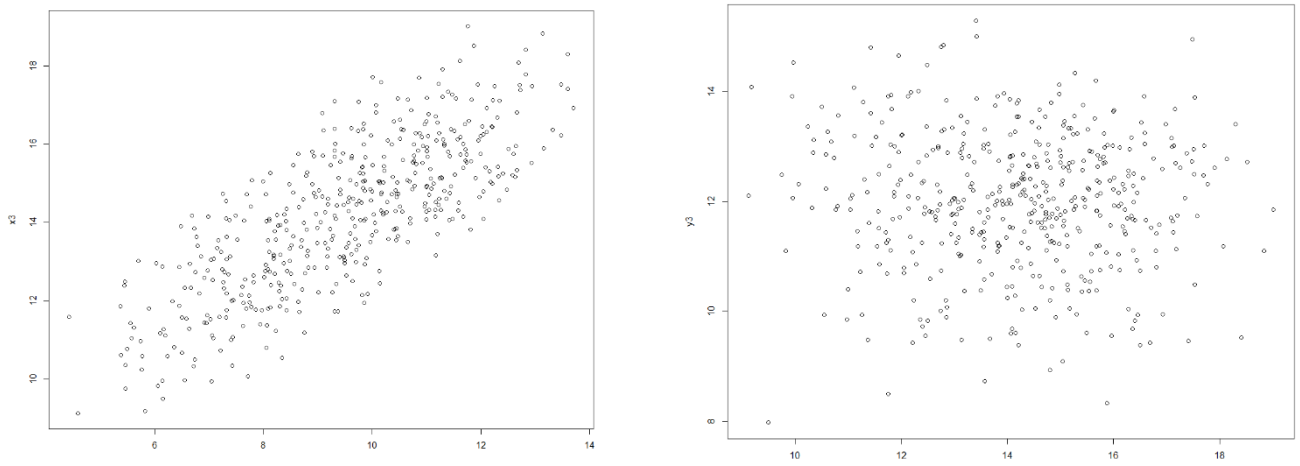


Fig 4.5 Scatter plots for x2 against x3 and x3 against y3.

After taking all the small changes into account to y1, theta1 and x2 I generated 10k new digits and got the distances. Below are the hists of the old and new dists, I will look at the classification after I have made changes to the 4's if any are necessary.

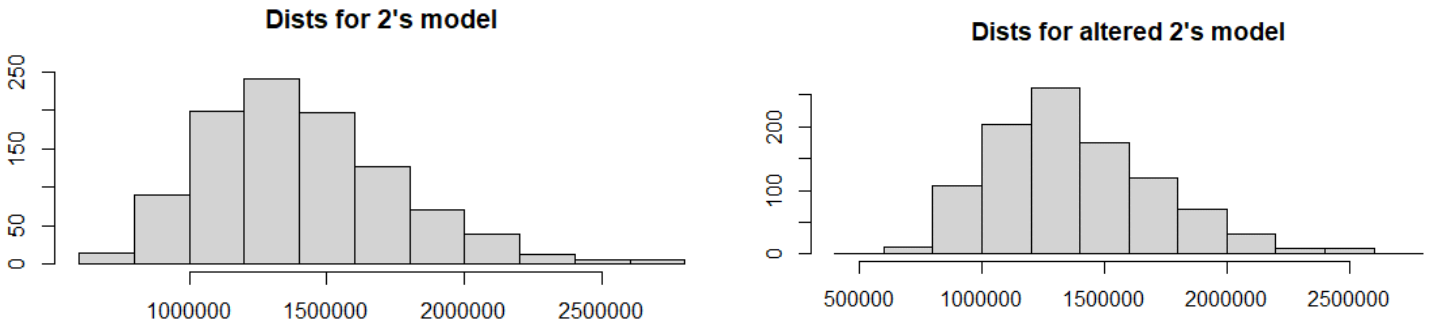


Fig 4.6 Histograms of distances for old (left) and new model from changes (right).

Comparing these histograms shows to me that the model has become a bit less general and has a really good match < 500,000 now. On the other end, it seems that we have lost a really bad match and there is only 1 now in that final bin. The 2,500,000 bin looks identical and the next bin to the left of that has decreased in size also. At first glance this model seems marginally better than the last, but we will be sure when we classify after analysing the distributions of parameters for the random 4's.

Now having a look at the parameters for the 4's, the following are distributions that I think are of interest and may mean that there are some changes that need to be made.

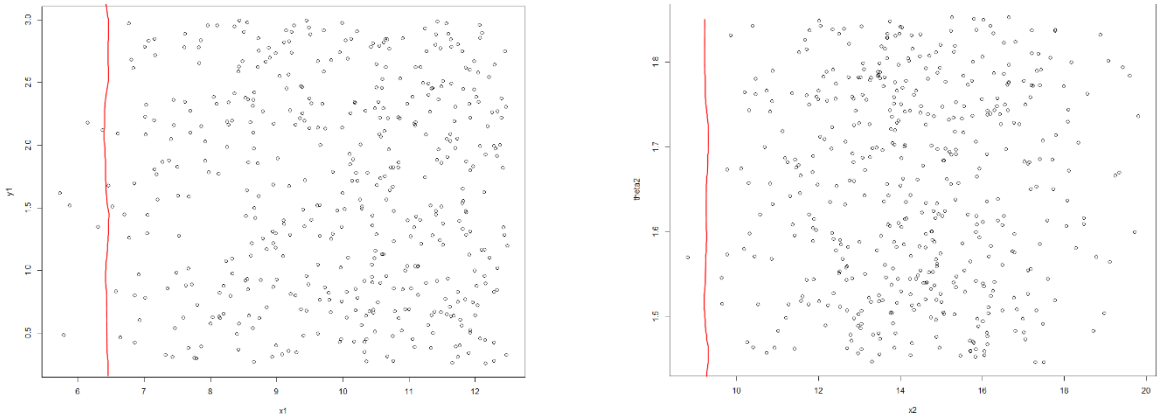


Fig 4.7 Scatter plots of x1 against y1 and x2 against theta2.

In the above it can be seen that increasing the value of x1 to about 6.5 would make a good impact on the good fits and provide less variation that is not making much difference to overall accuracy anyway. If this change is made to x1 it means that this will also fix the issue in the 2nd distribution which has x2 along the x-axis. These two parameters are dependent, specifically, x2 takes x1 into account. So, this change should make the model more accurate.

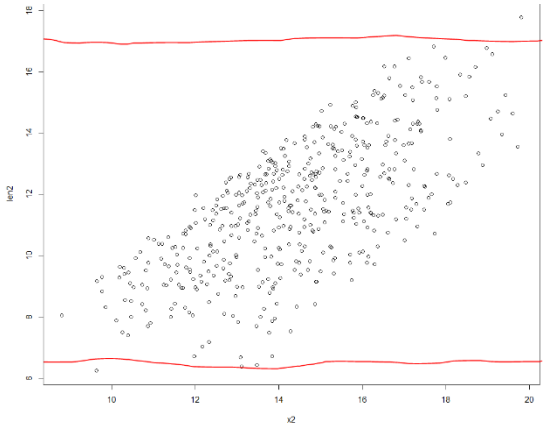


Fig 4.8 Scatter plot of x2 against len2.

In the above plot it can be seen that there is a relationship between x2 and len2. This is because I introdced it so that if x2 is to the right the distance is made up for in the length to create a cross stroke similar to that found on some 4's. In this way I can get the stroke to both sit on and to the side of the right stroke of the 4 and create this cross stroke with just three strokes. It can be seen though that the width of the interval for the length needs to be altered slightly as there are parts that don't get very good fits. I will decrease the range of len2 in an effort to fix this.

In the same way as the model for 2's, the model for 4's has some introduced dependencies that cause realationships. I believe that these are dependencies that exist when creating a typical digit in both of these cases and is true for about 98-99% of the real digits. The end of one stroke influences the start of the next as when drawign a 2 you never lift the pen (from my experience and the real digits presented here) and so the shape is a complete entity. In the same way, the angles in a typical 4 are consistent and the only stroke not dependent on the other is the right stroke.

Below are some of these such relationships.

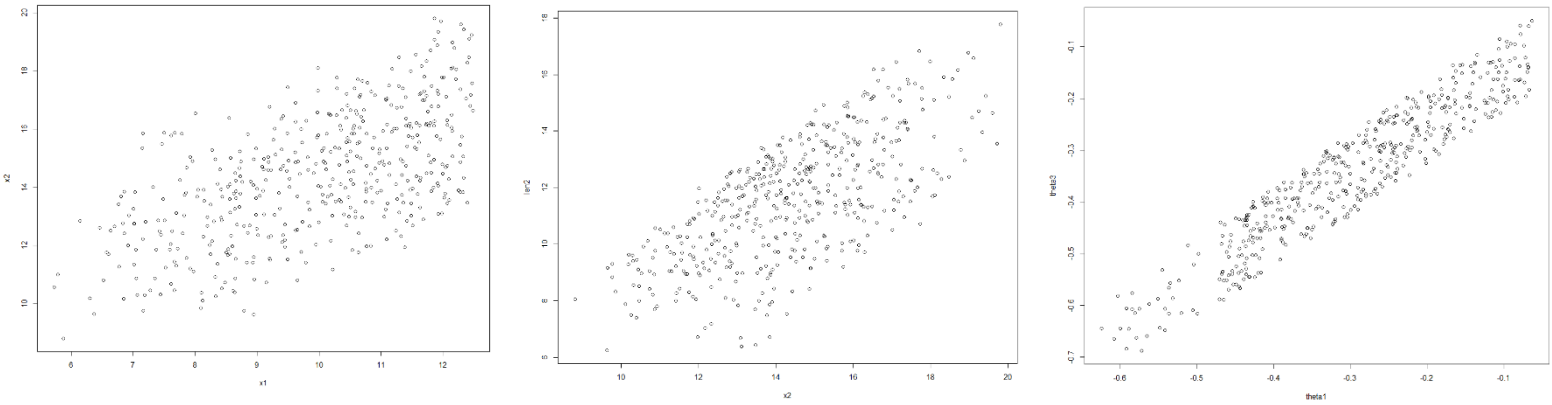


Fig 4.9 Examples of dependencies among model 4's parameters.

Now I will look at the dists for the new digits generated after these changes to x1 and len2.

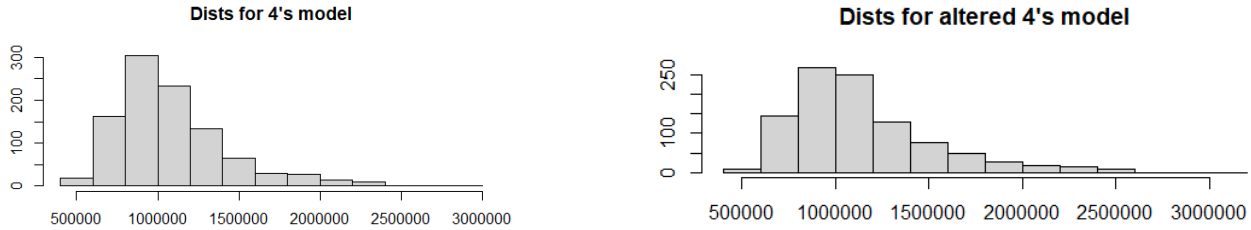


Fig 4.10 Histograms of distances for old (left) and new altered model of 4's (right).

Comparing the above above histograms makes me believe that the altered model here is less accurate. We lose some great matches, and gain some terrible matches. The distribution looks as though the values have increased across the board, so maybe it would be best to revert these changes as they have not improved the model. The confusion matrix below backs this up further as we have one more misclassified digit than before.

	labels	
results	2	4
2	999	14
4	1	986

Fig 4.11 Confusion matrix for new models.

So if I revert the changes to the model for 4's and use the original 4's and the altered 2's we find that there is still one more misclassified than previosuly found. So we will stick with the previous models for now.

	labels	
results	2	4
2	999	14
4	1	986

Fig 4.12 Confusion matrix for original 4's and altered 2's.

Section 5 – Applying PCA to the parameter values

Using the method from the lectures I completed PCA on both models parameters, in particular the good fits. This worked well and I found that the digits produced were very like the real digits and those produced by my original model. Some of these digits for both can be found below.

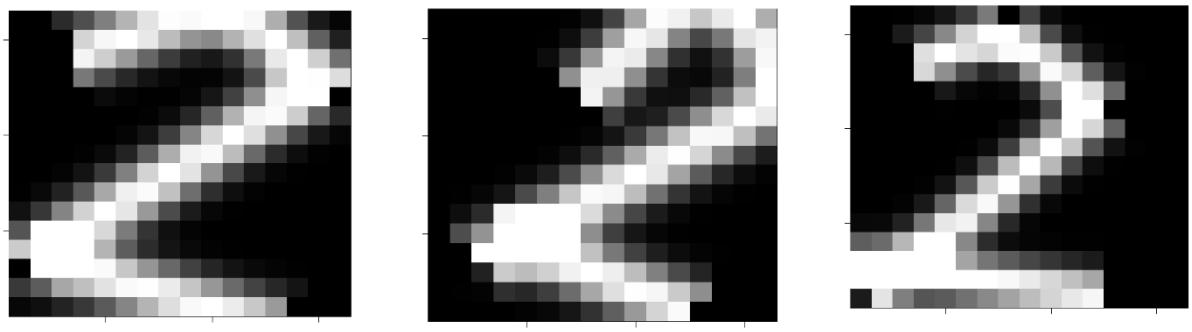


Fig 5.1 Some examples of 2's produced from using PCA on the parameters.

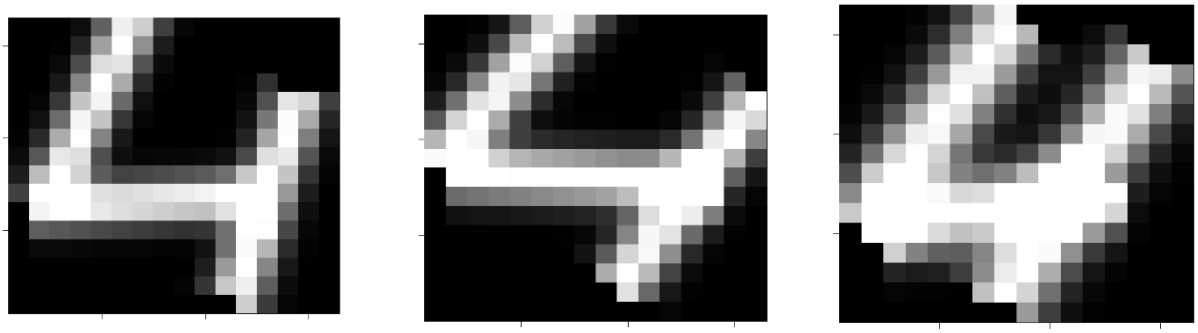


Fig 5.2 Some examples of 4's produced from using PCA on the parameters.

After running classification on the real digits using these randomly generated digits using PCA I got a confusion matrix that didn't look so confusing. The values for different numbers of k are shown beside the confusion matrix. It can be seen that the lowest was an accuracy of 99.85% at many values of k. With k = 3 we get our max of 2,000 correct out of 2,000 which give a 100% accuracy for these real digits using these random digits.

```
         labels
results    2    4
      2 1000    0
      4    0 1000
```

```
> acc
 [1] 1999 1999 2000 1999 1999 1999 1998 1998 1998 1999
[11] 1998 1998 1998 1998 1998 1998 1998 1998 1998 1998
[21] 1998 1997 1998 1997 1997 1997 1997 1997 1998 1997
[31] 1997 1997 1997 1997 1997 1997 1997 1997 1997 1997
```

Fig 5.3 Confusion matrix for PCA random digits and the corresponding entries in acc.

I wanted to find out how much of an improvement this new model was over the old one. So I decide to get the histogram of distances for these random digits and compare them to the previous best that I had.

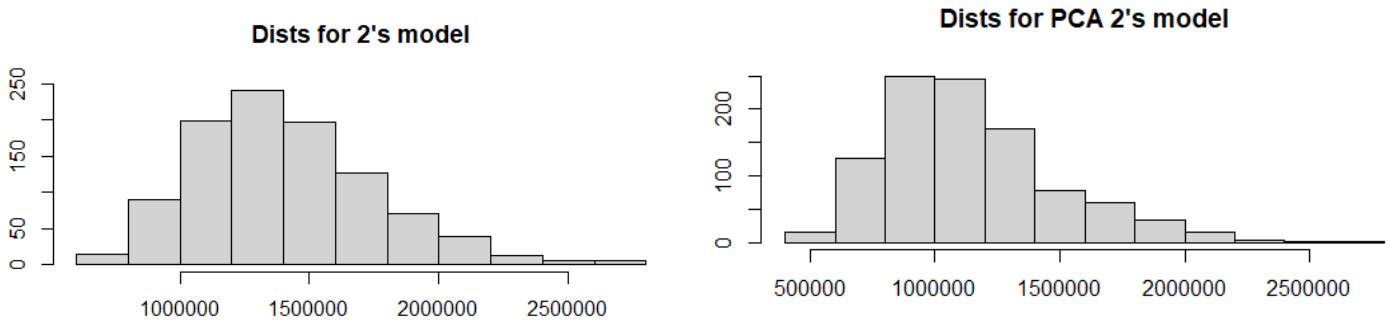


Fig 5.4 Histograms of dists for previous best (left) and PCA (right) 2's.

It can be seen from the above histograms that there is a large improvement. There are a lot of matches now under 1,000,000 where this spike was previously closer to 1,500,000. There are also a lot more really good matches and even some excellent matches. In addition, there are less really bad matches up the far end past 2,000,000 so it is undeniable that this model is an improvement upon the last.

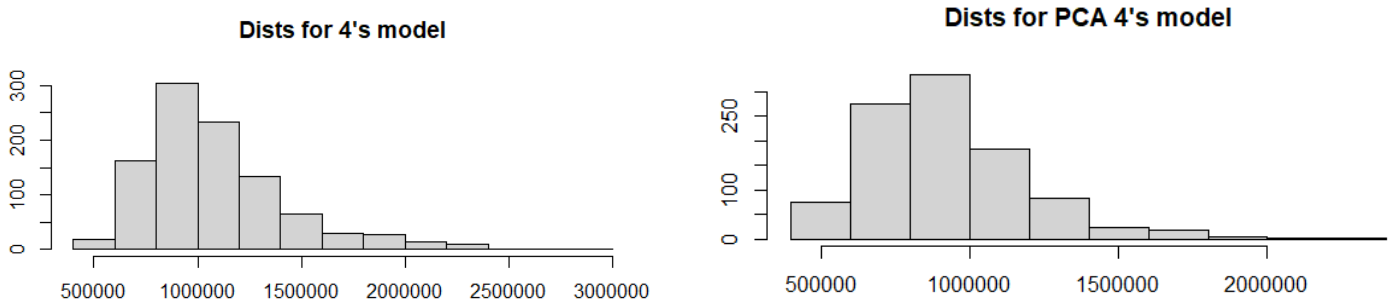


Fig 5.5 Histograms of dists for previous best (left) and PCA (right) 4's.

Looking at these histograms there is again an obvious improvement. There are more excellent matches, there are very few matches past 1,500,000 in comparison to the old model. This I believe explains how the PCA model has achieved a 100% accuracy with these real digits. The issue could be however that in the long run when applied to other sets of real digits that are not similar to these real digits there will be many errors. I believe that this however will not occur because these real digits provided contain many examples of typical 2's and 4's and I don't believe that digits straying too far from these could be considered a very good 2 or 4.

As there are no errors I unfortunately cannot provide cases of mismatches and understand where the model went wrong, but I know that there would definitely be digits not contained in this set that the model's may struggle to accurately match.

Section 6 - Using different numbers of image-eigenvectors.

Making use of eigenvectors can drastically improve the speed at which nearest neighbours are found with the knn algorithm. In a simialr way to the last assignment I will look at the accuracy and the times taken for different numbers of eigenvectors. Below the plot of no. of eigenvectors against accuracy can be found.

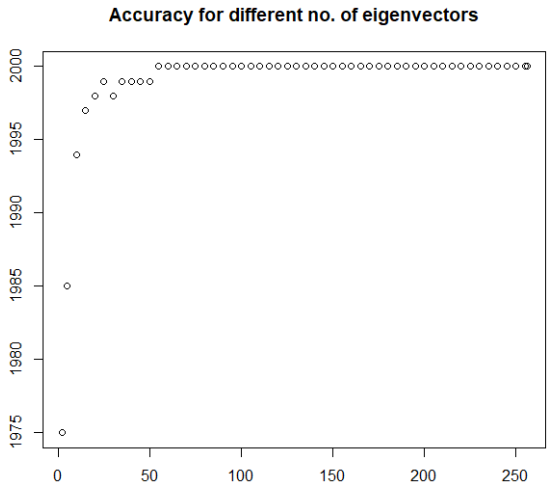


Fig 6.1 Plot of no. of eigenvectors against accuracy.

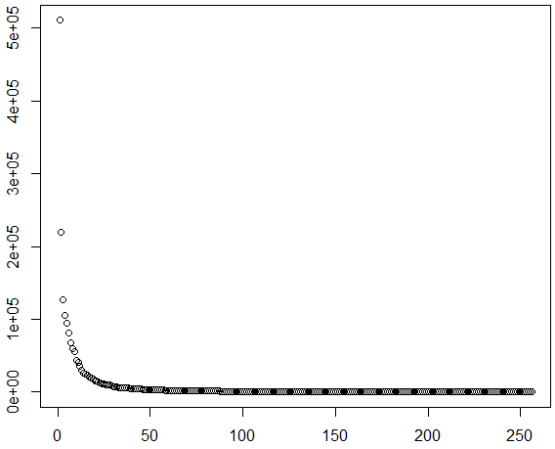


Fig 6.2 Plot of eigenvalues for eigenvectors.

It can be seen that the above plot is very similar to that of the last assignment. Making use of only a few eigenvectors means that you lose some information, as can be seen in the plot of eigenvalues after 50 eigenvectors essentially all of the information is preserved. This is clear as in the plot of accuracies you can see 100% accuracy is achieved at 55 eigenvectors. The curve of the plot can be seen as that of a typical log function, lots of change in small values and then it levels out as the values get bigger. This is of course because eigenvectors added after about 55 have relatively little information about the digits and so only add extra computations, in fact the same amount of computations as if you add the second eigenvector to the first. The only difference is that the 56^{th} is much less informatiove.

Following on from the plot of accuracies I looked at the time taken to compute the nearest neighbours with different numbers of eigenvectors, this plot can be seen below. As you can see, there is a linear relationship between the number of eigenvectors and the time taken for the computation. This means that adding more eigenvectors increases the time by the same amout whether you start with 10 or 100 and add more. The linear relationship here is 0.856 + (number of eigenvectors * 0.013) = time in seconds.

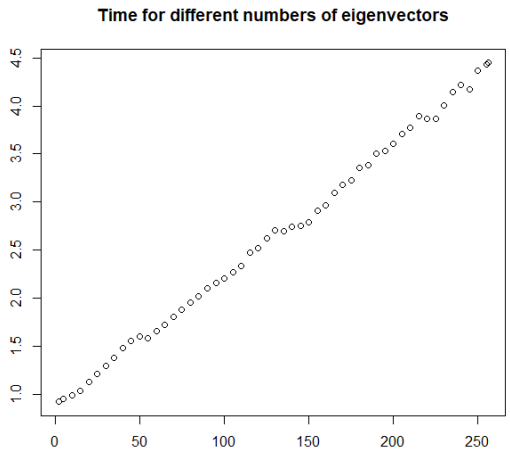


Fig 6.3 Plot of time for different numbers of eigenvectors.

So this then means, if after 55 eigenvectors adding more eigenvectors is useless and adding more just increases the time then isn't this the max? Well, for 100% accuracy which happens at 55 eigenvectors it takes 1.58 minutes. For 99.95% accuracy it only takes 25 eigenvectors which takes 1.21 minutes. These can also be computed from the given formula, 0.856 + (55)*0.013 = 1.57 minutes and 0.856 + (25)*0.013 = 1.181 minutes. So for the sake of 20 seconds you can get 100% accuracy and essentially all of the information from the digits using eigenvectors!

Section 7 - Classifying the ones, twos, fours and sevens together.

To begin I decided to just check the accuracy with the basic model for the 1's and 7's from the lectures, not making use of PCA and then repeating with PCA afterwards if I felt it was appropriate. The histograms of dists for both the 1's and 7's can be found below.

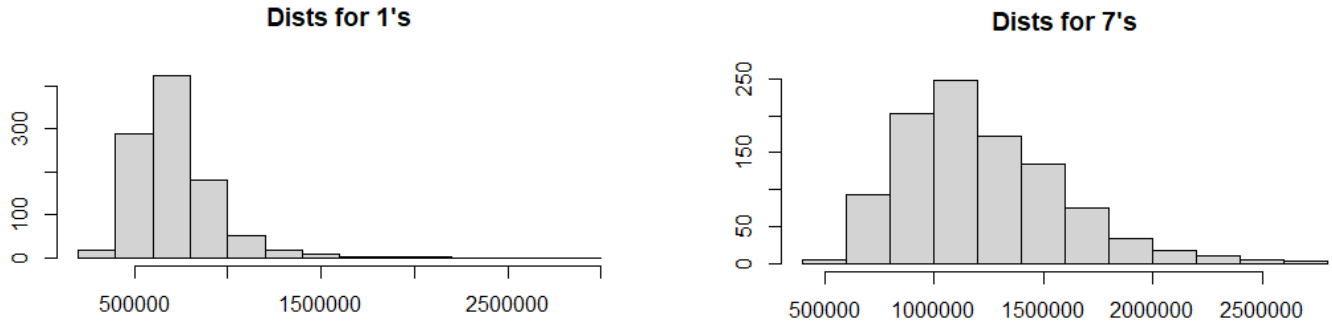


Fig 7.1 Histogram of dists for 1's and 7's

It can be seen that both models are pretty good at fitting the real 1's and 7's. In the same way as with the 2's and 4's, there are some outliers that would not be considered a typical 1 or 7 and you would not want to base the model arounf getting these very accurate if there's a trade-off in accuracy for more common 1's or 7's. So following on from here, I computed the nearest neighbours for the real 1's, 2's, 4's and 7's together and found the confusion matrix which can be seen below. I got the best result where k was 15.

results	labels 1	2	4	7
1	990	6	9	123
2	0	993	1	45
4	1	1	989	28
7	9	0	1	804

Fig 7.2 Confusion matrix for 1's,2's,4's and 7's together.

As can be seen, the accuracy has dropped from before, which is to be expected when integrating model that have never been compared against one another. They have only been tested in pairs, so it can be seen that there is one 2 classified as a 4 and one 4 classified as a 2. The overall accuracy of the classification is 94.4%. The main issue lies within the model for 7's as it is not as close to the real 7's as the other model in 20% of cases roughly. So I will have a quick look at some of these digits that are misclassified and why that may be.

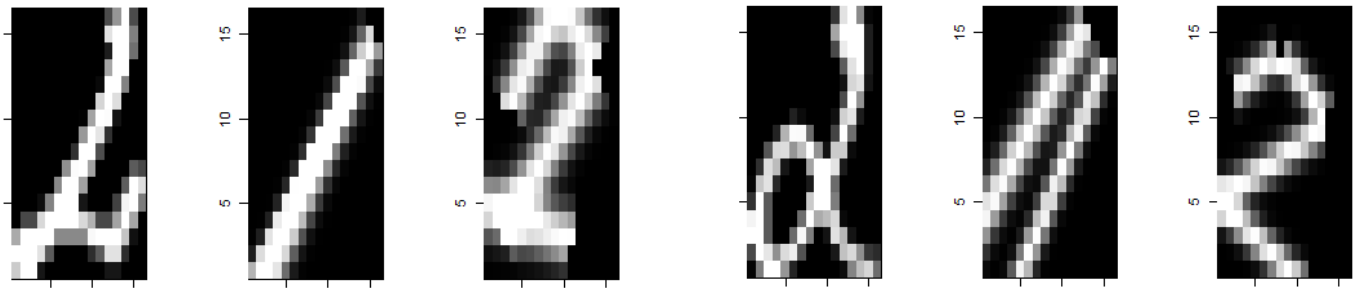


Fig 7.3 Some examples of real 2's misclassified as 1's and nearest random 2.

It is clear that some of these 1's fit better when they join strokes as some of these real 2's have strange shapes that the model finds hard to match. I hope that using PCA will elimnate these more unpredicatble 1's and improve accuracy.

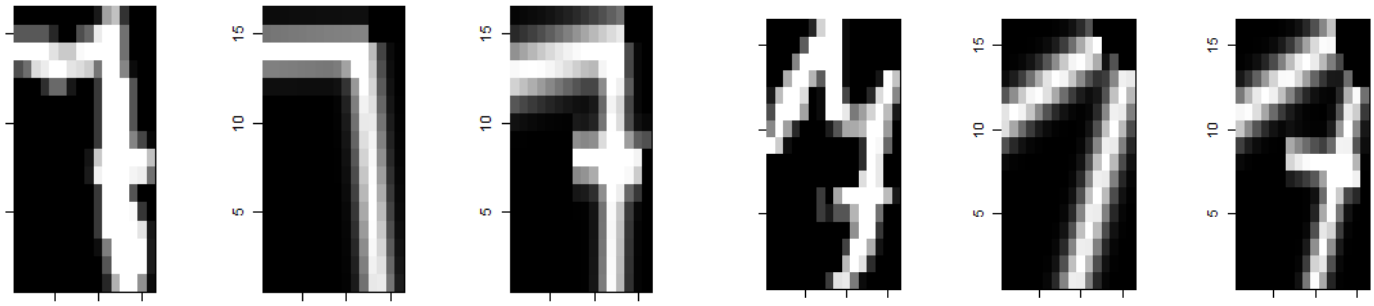


Fig 7.4 Some examples of real 7's misclassified as 1's and nearest random 7.

It can be seen above the excluding the cross stroke the 1 and 7 are very similar. Placing that cross stroke in the wrong place can lead to random 7's further from the real 7's than random 1's. So I hope that using PCA will aid in getting a better spread of random digits.

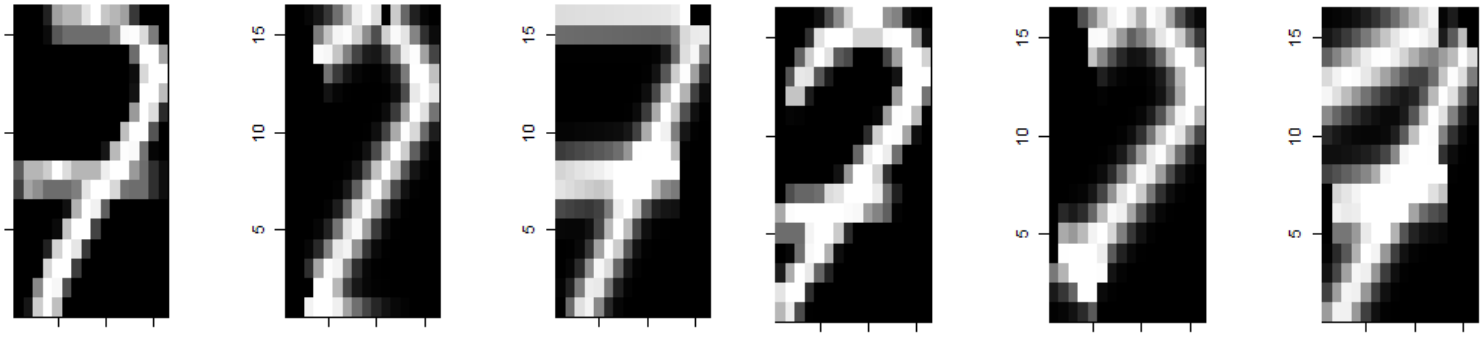


Fig 7.5 Some examples of real 7's misclassified as 2's and nearest random 7.

Again, it is clear how these 7's have been misclassified. The extra stroke I added for the top of the 2's has matched with some hooked upper strokes on 7's better than a random 7 with just one stroke can match. In addition, when the bottom stroke goes off the both of the 16x16 of the 2 it does not get penalised for it and is only missing a cross stroke to be a 7. I hope that PCA for the 7's will increase the overall accuracy and reduce the number of these random 2's that are nearer the the real 7 than the random 7's are.

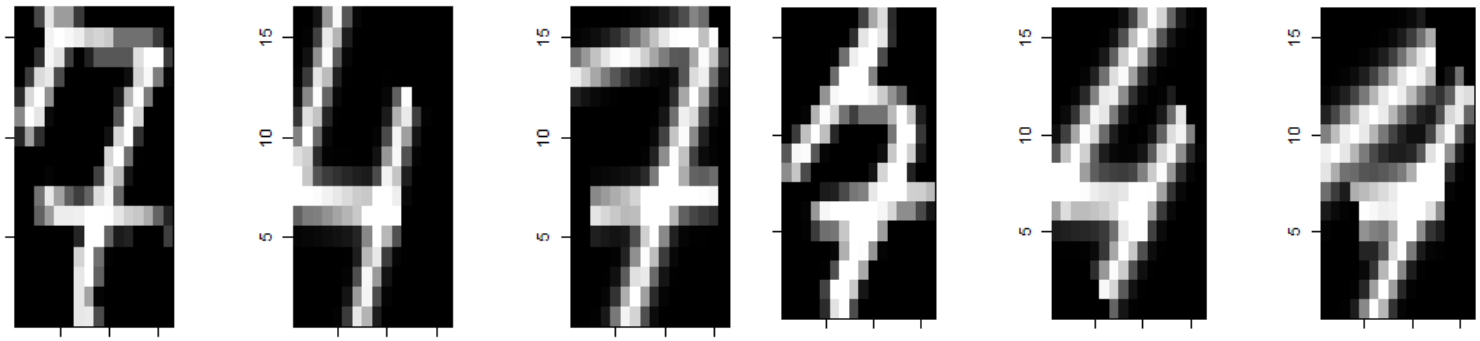


Fig 7.6 Some examples of real 7's misclassified as 4's and nearest random 7.

In the above real 7's the hook makes it difficult for the 7's to match well with them. This is not something that PCA can fix, but I hope it will mean the rest of the 7's shape is a better fit and the random 7's become closer than the random 4's. These 7's however are not the most common. It may end up though that adding another stroke to the upper stroke of the 7 will increase the accuracy, which if the accuracy does not improve from PCA would be my suggestion for improvement.

Having looked at some mismatches now I will use PCA on the 1's and 7's in the same way as with the 2's and 4's.

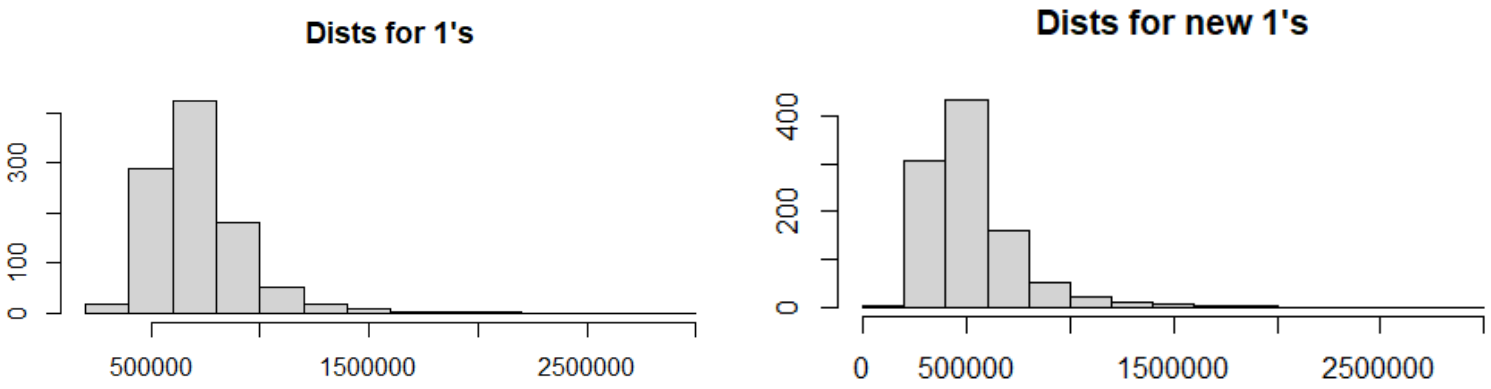


Fig 7.7 Histograms of dists for old 1's (left) and new 1's (right).

It seems that PCA has had a large impact on the dists between real 1's and its nearest random 1. It can be seen that the large peak is now at the 500,000 bin and many of the other bins have shifted down slightly too. So hopefully this will help to decrease the errors in classifying.

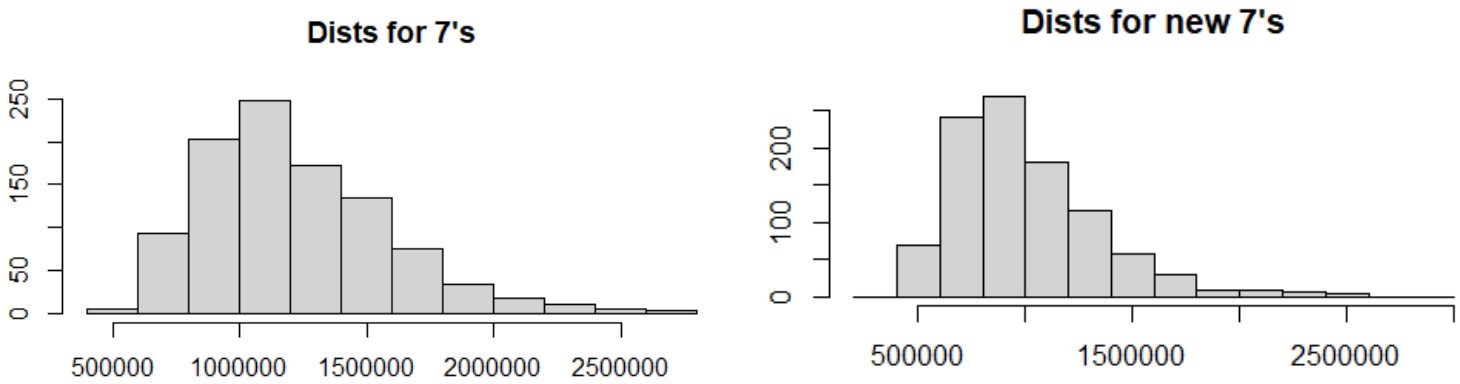


Fig 7.8 Histograms of dists for old 7's (left) and new 7's (right).

In a similar way to the 1's, the dists between real 7's and the nearest random 7 has dropped noticeably. There are only a couple past 2,500,000 and all other bins have shifted to the left giving a more built up peak at under 1,000,000. This will defintely help with classification as the distance between real 7's and the random ones are much smaller than previously.

Having run the classification again with the new random digits for the 1's and 7's I must say I am pleased. There has been an overll increase of about 4.08%, so the new accuracy is 98.5%. The confusion matrix is shown below with the old one for comparison.

results	labels 1	2	4	7
1	990	6	9	123
2	0	993	1	45
4	1	1	989	28
7	9	0	1	804

results	labels 1	2	4	7
1	998	3	7	14
2	1	997	1	21
4	1	0	992	13
7	0	0	0	952

Fig 7.9 Confusion matrix for previous (left) and classification with final random digits(right).

Comparing the new with the old shows an increase across the board with 8 more 1's, 4 more 2's, 3 more 4's and 148 more 7's being classified correctly, so that is a total of 163 more correctly classified digits than last time. The only real issue now seem to be alone the first row and the last column. Looking at some misclassified digits may help to highlight where the change is needed.

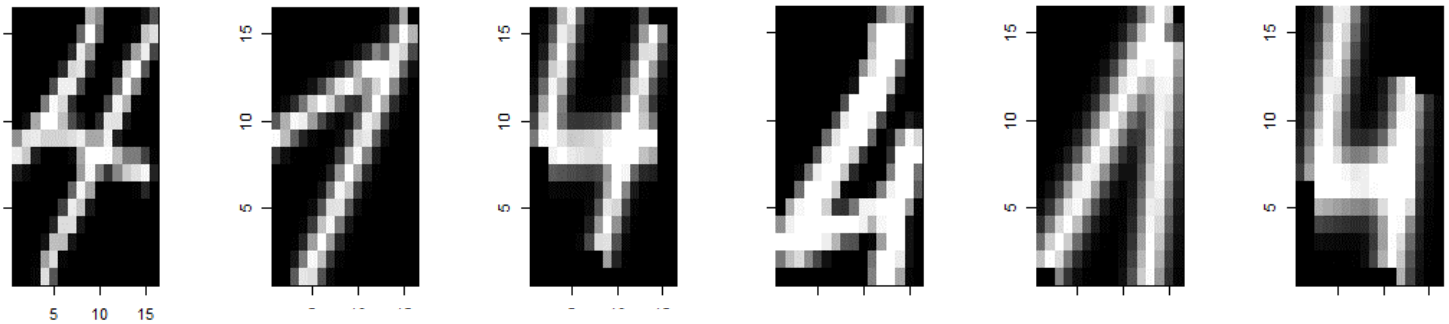


Fig 7.10 Some examples of real 4's misclassified as 1's and nearest random 4.

Looking at the misclassified 4's we find that the 1's can fit the shape of some of the real 4's better. Some of the real 4's start their right stroke over to the left and the model just isn't going over far enough. In the scenario on the left the right stroke also goes all the way to the top of the 16x16 which the 1 picks up very well. So maybe a change to the start point and length of the right stroke may help. Looking at the other example we have a very loe cross-stroke and low start to the left stroke which is very uncommon. The real 1 is able to match that stroke very well, so maybe the length of the initial right stroke could be lowered and the length and angle of the left stroke changed to be closer to the one seen above to the right.

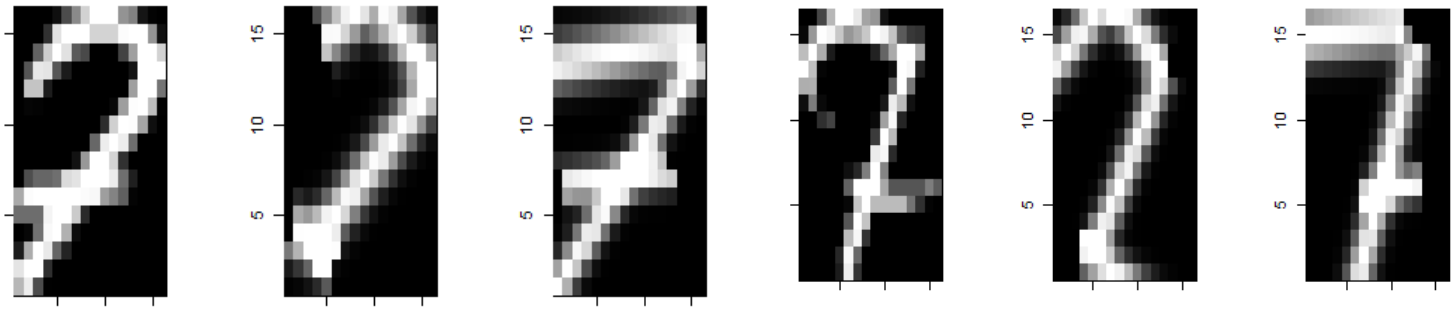


Fig 7.11 Some examples of real 7's misclassified as 2's and nearest random 7.

I believe that the best way to improve this further would be to add another stroke at the top of the 7. This stroke has the possibiliy of being tiny and horizontal or a bit longer and diagonal down to the bottom left. This would help the 7's match more with these 7's that are matching with 2's as a result of a similar stroke in the 2's. About 80% of the remaining errors lie with classifying 7's, so I believe this is the best approach.

Overall I have found this assignement extremely interesting and am very pleased with the final results.