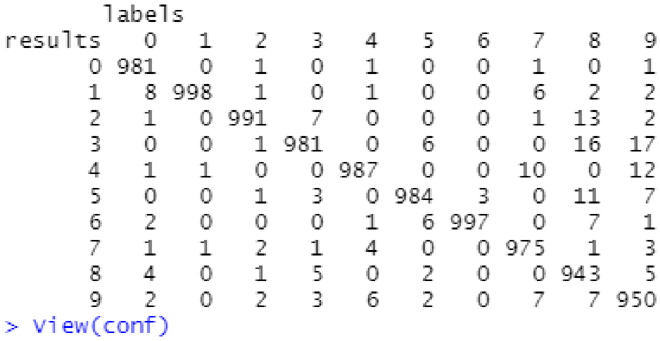
Assignment 1 CA274

In this assignment I try to optimise the KNN algorithm to classify handwritten digits with the highest possible speed and efficiency. Note all that time measurements are taken from differences of elapsed time (proc.time).

Question 1

Note that setting k = j uses j-1 neighbours to classify and not j. The first neighbour is set to the target digit itself. Using k=2, This is the confusion matrix I obtained:

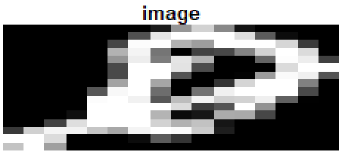


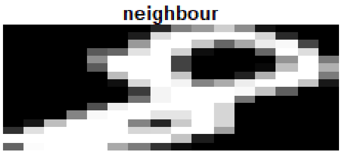
```
       labels
results   0   1   2   3   4   5   6   7   8   9
      0 981   0   1   0   1   0   0   1   0   1
      1   8 998   1   0   1   0   0   6   2   2
      2   1   0 991   7   0   0   0   1  13   2
      3   0   0   1 981   0   6   0   0  16  17
      4   1   1   0   0 987   0   0  10   0  12
      5   0   0   1   3   0 984   3   0  11   7
      6   2   0   0   0   1   6 997   0   7   1
      7   1   1   2   1   4   0   0 975   1   3
      8   4   0   1   5   0   2   0   0 943   5
      9   2   0   2   3   6   2   0   7   7 950
> View(conf)
```

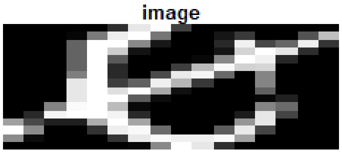
The accuracy for k = 2 is 97.87%

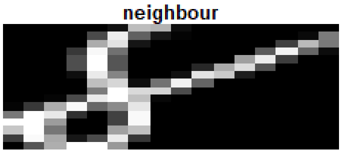
I will now investigate some of the misclassifications. There were 4 zeroes misclassified as eights. let's have a look at these.

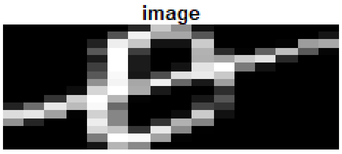
Some of the misclassified zeroes are not drawn clearly. As can be seen in the first pair of images, the 0's share similar stroke shapes to the eights. The slanted of the eights share similarity to the zeroes also. 1 neighbour cannot be enough to correct such errors so increasing the number of neighbours may improve the accuracy.

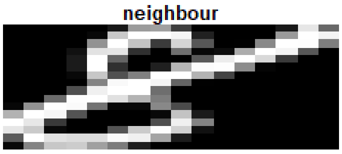


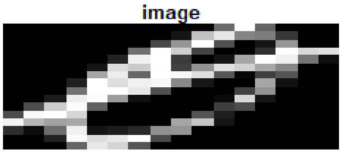


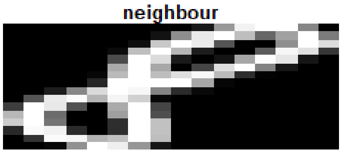






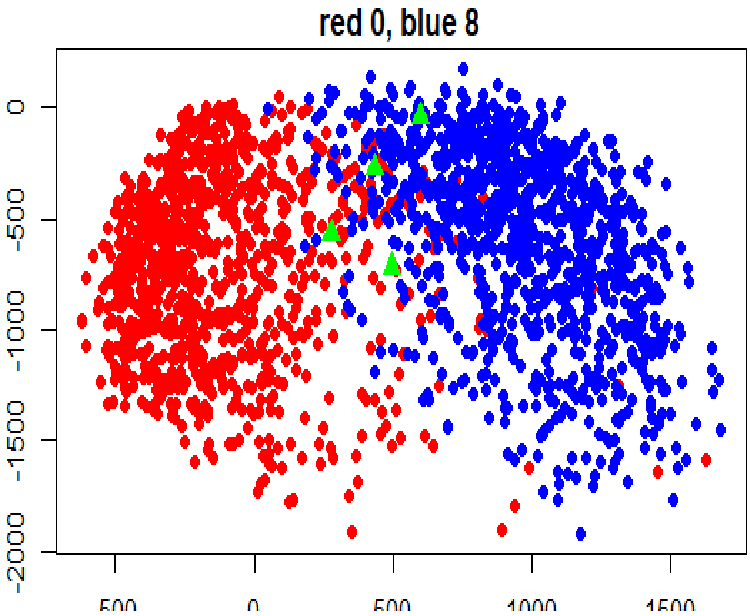




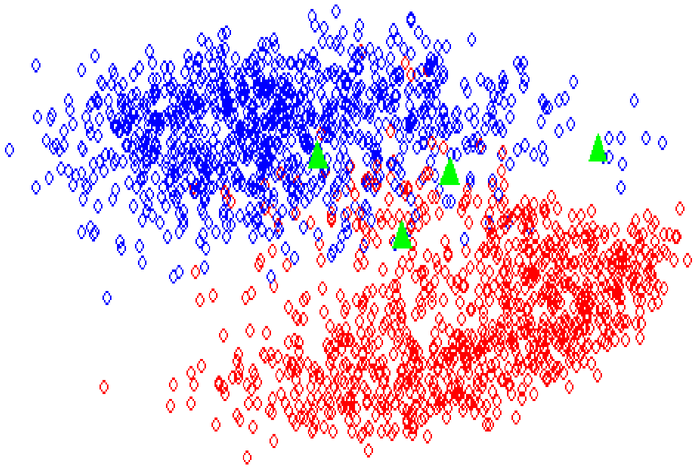


Question 2

I will now display the 0's and 8's in 2D and 3D point cloud. I will use PC1, PC2, and PC3 as the dimensions on which to generate the point cloud. Note that I performed PCA only on 0's and 8's, no other digits. I removed the axis when displaying the 3d point cloud here to make the misclassified points more visible.



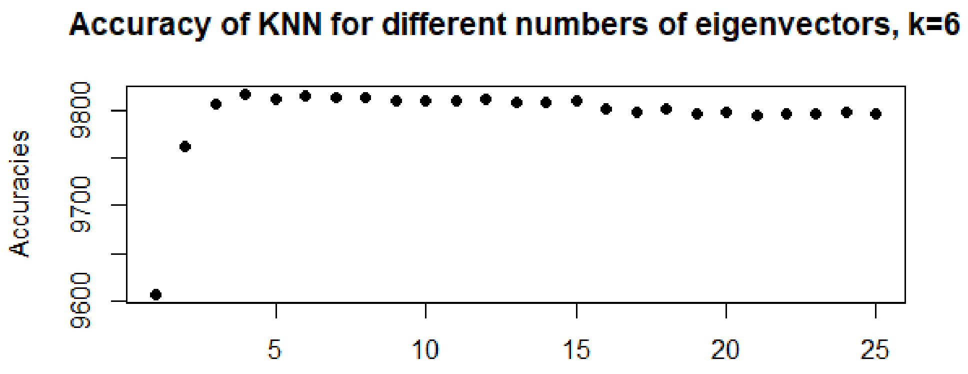
Red indicates the digit **0** while **Blue** indicates the digit **8. Green** indicates 0's misclassified as 8's. No 8's were misclassified as 0. The misclassified digits are fairly spread apart, with most of these appearing in the 8's blue cloud.

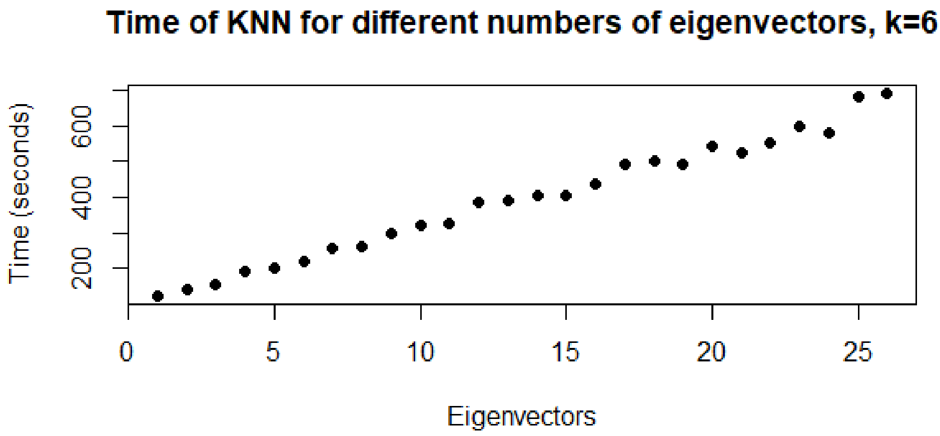


The algorithm is working on k = 2 so increasing k slightly will deal with ~2 of these misclassifications based on their positioning (those in the centre which are near the border between the clusters) Most of the misclassified zeroes are ~equally or closer to the 8's cluster than the 0's cluster so for reasonable values of k these are likely to still be misclassified by KNN.

Question 3

I will fix k = 6 and measure the accuracy of KNN using different numbers of eigenvectors ranging from 2 to 256 in steps of 10. I will hence go from 2 to 252 eigenvectors. For the accuracy plot I removed the first KNN accuracy (For 1 eigenvector) to make the shape of the plot clearer.

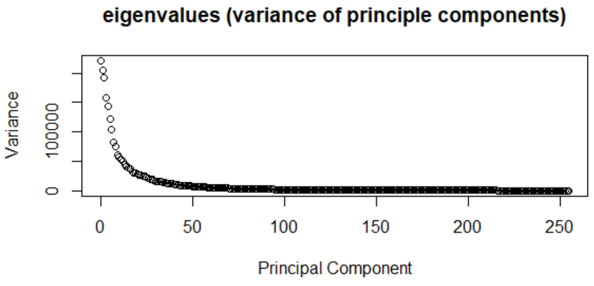




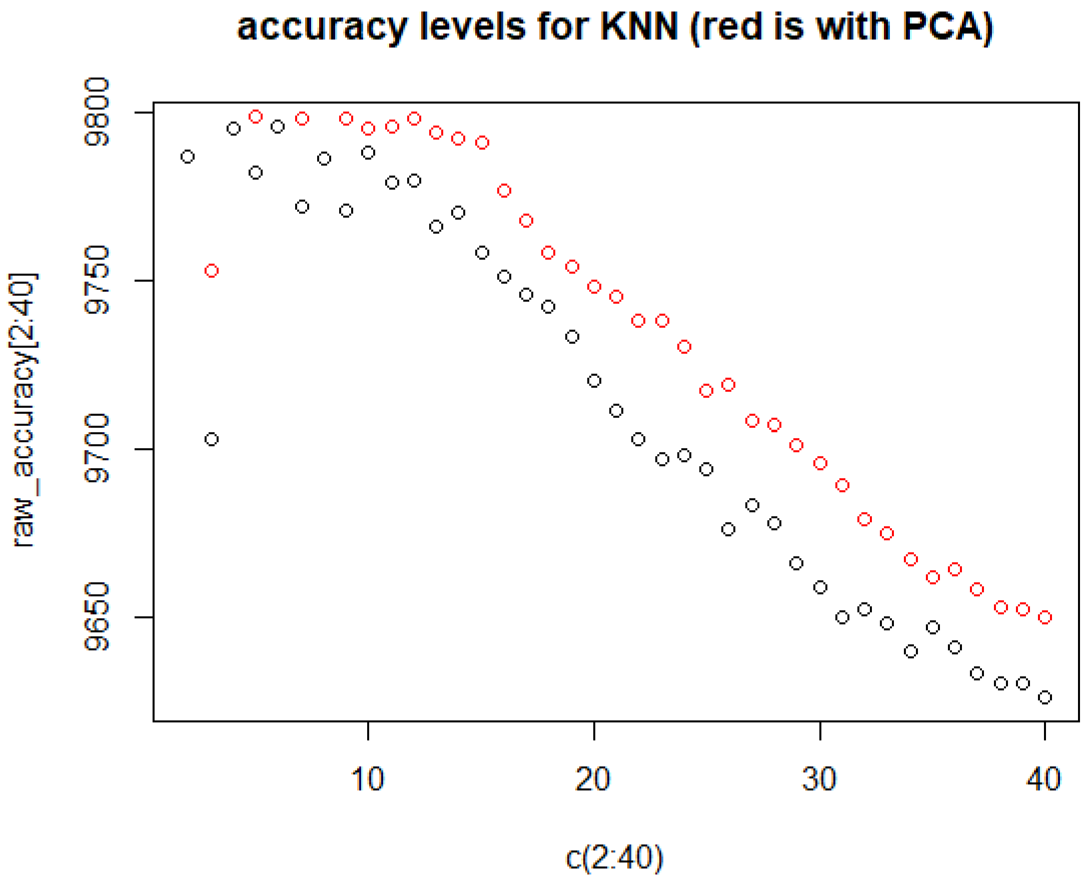
The accuracy curve appears to follow a logarithmic growth pattern but upon reaching ~40 eigenvectors it begins decreasing slowly and then levelling. The highest accuracy was when I used 42 eigenvectors. The following metrics for this were observed:

Accuracy: 98.17%, in comparison to raw data accuracy of 97.96%.

Time: ~202s, in comparison to raw data training time being ~502s. Since I went from 2 to 252 in steps of 10, This gives reason to believe that around 40 eigenvectors are a good number to optimise accuracy. Around 40 eigenvectors almost all the variance is accounted for in the data so adding more components doesn't increase the accuracy of the model. It even slightly decreases the quality of the prediction! The runtimes increased linearly from ~124 to ~689 seconds. The entire job took around 2.5 hours. As of this point, using 40 eigenvectors has the best accuracy and it's time ratio with the raw data approach is very good.



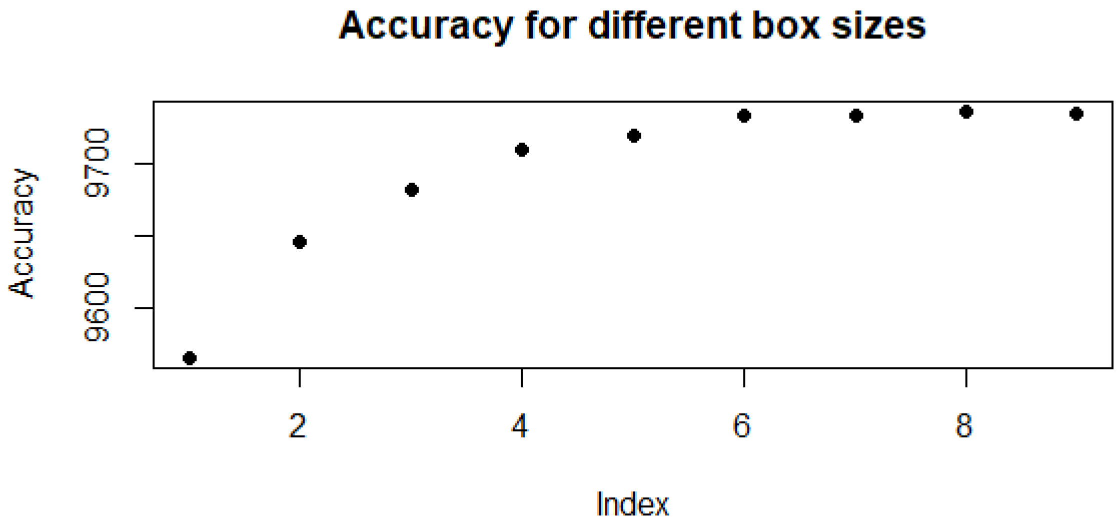
As can be seen here, the accuracy using 40 eigenvectors is higher than that of the raw data at average

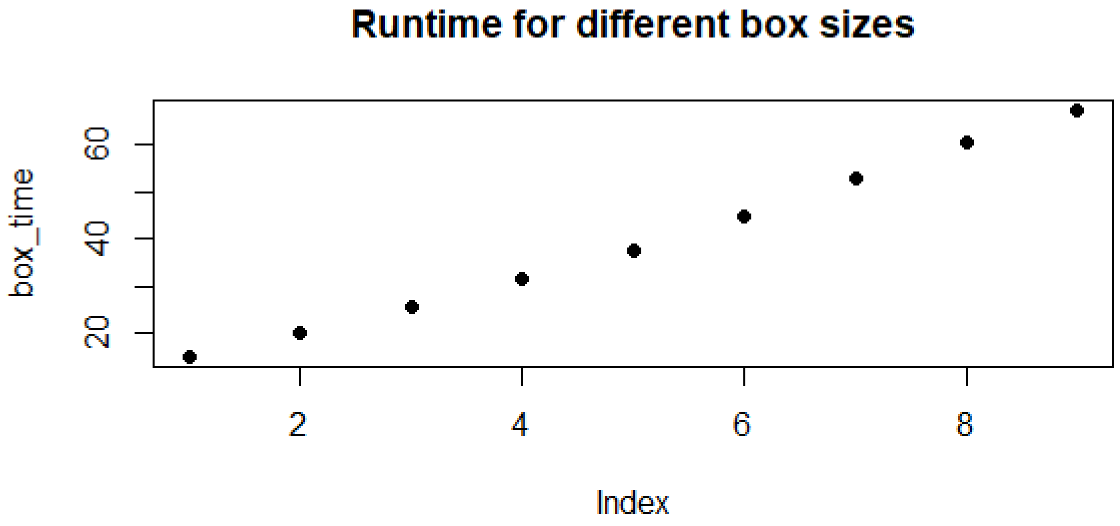


Question 4

I will fix the algorithm for k=6 and 20 eigenvectors (principal components) and run KNN using the box search method. I will only base the boxes on PC1 and PC2. From my analysis, I found that the average distance between point A in PC1 and any other point in PC1 is around 800. For PC2 it is around 300. The boxes will range from (400 x 100) to (800 x 500) increasing both sides by 50 units per run.

The accuracy increased in a logarithmic shape with a maximum of 97.37% for a box of (750x450) which took 60.47s to run. Previously, to find the nearest neighbours using 20 principal components, k=6, and no boxes had an accuracy of 97.38% and took 163.83s. The accuracy to time ratio here is fantastic when comparing these two methods. The times taken run boxed KNN ranged from 15.17s to 67.29s.

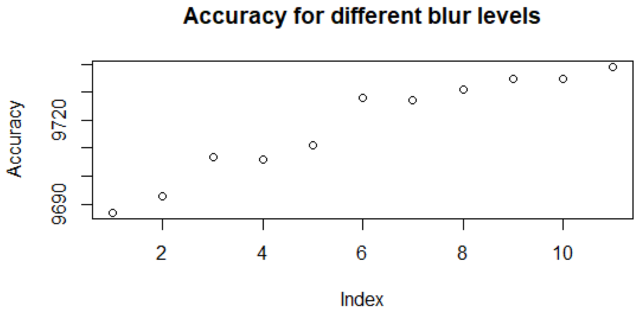


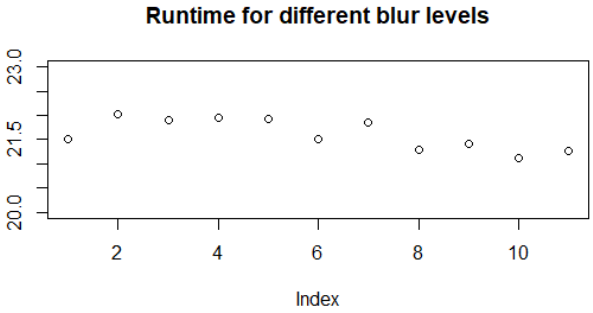


Question 5

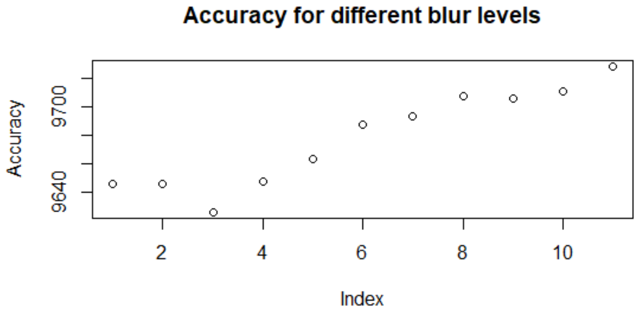
I'll first test accuracies and times for different blurring levels, fixing the box size, and finding the region of high accuracy and medium runtime. I ran this test for widths 5 to 15 increasing by 1 on each iteration. I classified the data using the original principal components with **40 eigenvectors** but found search regions (boxes) using blurred principal components (PC1, PC2)

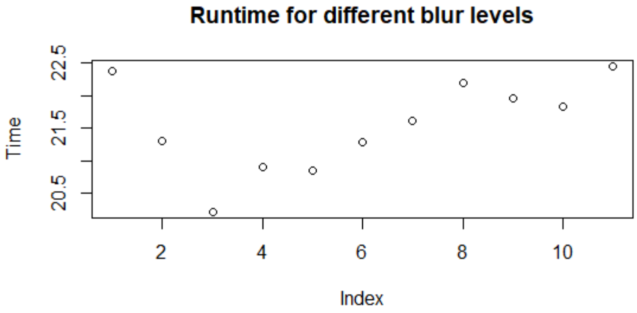
I found here that the runtimes for different blur levels were approximately uniformly distributed with a min of ~21s and a max of ~22s. The accuracies were increasing linearly with the widths. Let's see how high the accuracy can go without a time trade-off.


Accuracy for different blur levels


Runtime for different blur levels

Running KNN for 40 eigenvectors, using blurred PC1 and PC2 to find search regions, and using blur widths from 15 to 30, I obtained the following results.

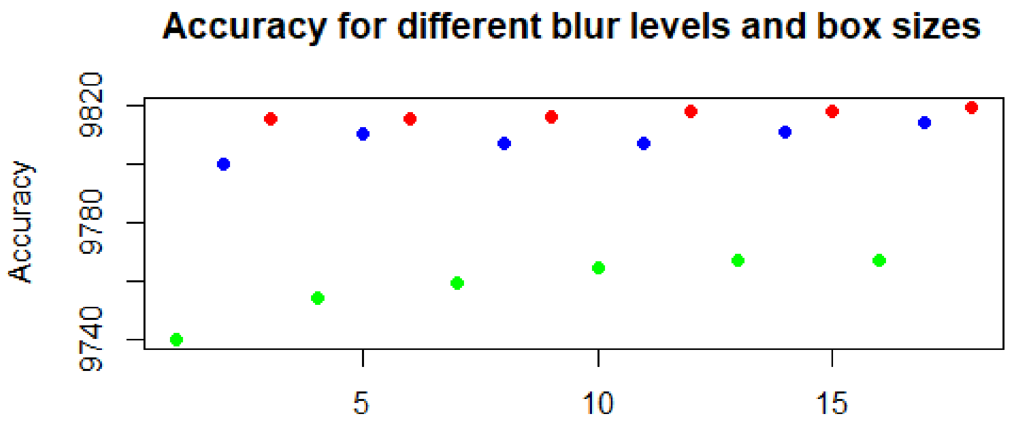

Accuracy for different blur levels


Runtime for different blur levels

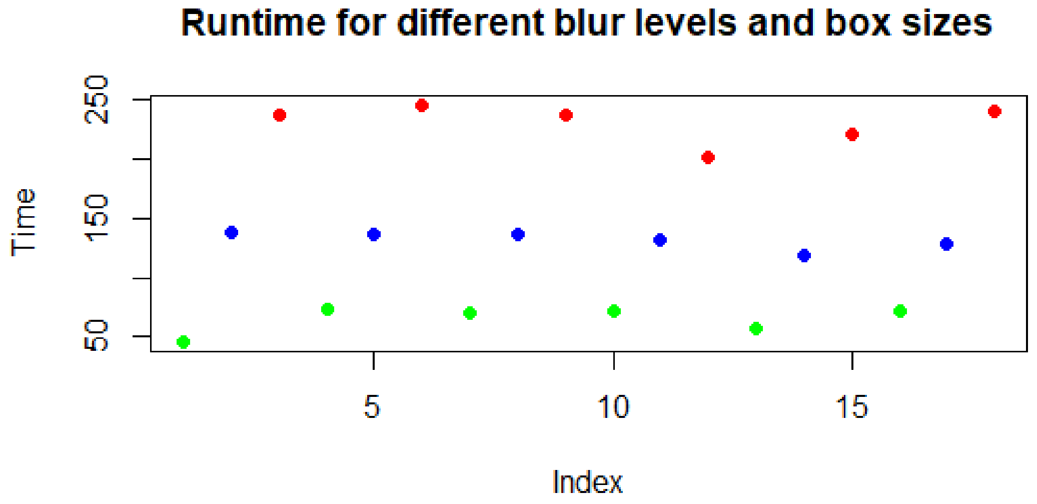
Runtimes moved from range (20s, 21s) to (20.5s, 22.5s) but still, the accuracy increases. With this as I increase the width of the blur, I am slowly reverting to a regular PCA approach with no blurring but using smaller boxes. The highest accuracy I observed was 97.28%. (the upper benchmark is currently 98.17% for 42 eigenvectors) which had a time of 22.45s (202s benchmark) From this testing I believe that the accuracy will only converge when the blurring level is very small (high width). A good box sizing system will be more helpful.

I will now do a multilevel test to try and maximise accuracy in the blur width range [5,10]. For each width from 5 to 10, I will calculate box sizes based on the variance of pc1 and pc2. Using $\frac{2}{3} \times \sigma(PC), \sigma(PC), and\ \ 3/2\ \times\ \sigma(PC)$ where σ(x) is the standard deviation of data vector x. As the width increases, the variance of pc1 and pc2 reduce and hence the search regions are more populated which seems to improve the accuracy while keeping the runtime trade-off at an impressive low.

Each colour represents a different box size. Green is small box, blue is medium, and red is largest box. The first 3 data points relate to the different box sizes with blur width = 5, the next 3 with blur width = 6 and so on until blur width = 10. The runtimes for the different box sizes appear to be distributed approximately uniformly once again. The middle box sizes show a greater jump in accuracy than that of medium to large boxes. Considering time and accuracy, width = 10, and medium box size provides an accuracy of 98.14% with a runtime of ~128s on my machine. This is a higher accuracy and better time than the previous benchmark using PCA without blurring or boxes (98.04%, ~202s)



Accuracy for different blur levels and box sizes



Runtime for different blur levels and box sizes