

## *Logbook for Machine Learning*

Your Name Stefan M Ahmed

Your Student Number 21359035

01/2021

## Week #

Task 1. ....

Your answer provided here.

Task 2. ...

Your answer provided here.

....

## Week #

Task 1. ....

Your answer provided here.

Task 2. ...

Your answer provided here.

....

## Week 3

Task 1. ....

Your answer provided here.

Task 2. ...

Your answer provided here.

....

## Week 3

### Task 1

Say you have data that has been measured and it plotted as points. You then want to look for a pattern to find facts from the plotted points by dividing them into clusters. You could divide them into different colours to show they are being clustered. There are issues you end up with when doing so and options to fix these problems.

#### Issues with k:

The k-means algorithm has several issues. One issue is putting the data into the incorrect number of clusters. Doing so will produce strange results. If you run k-means algorithm with k being the incorrect number of clusters you will get the clusters that are not right. You don't see groups of data being clustered. It is a random operation. You start by coming up with a number of k clusters you think is suitable enough. You then try to divide the data into clusters by randomly picking a data point as a centroid. Being random explains running k-means algorithm with a wrong number.

You cannot tell if there is a cluster in the data as you see the same number of groups as the k no. of clusters you have chosen. It is rather confusing. This brings on the other issue of the number of centroids which is k. The number of centroids are the same as the clusters. This means you are likely to end up repeating the steps in the process which could be a lot of steps to repeat. End up repeating the steps that pick the next centroid as the middle of the data points again and again. These are the no. of iterations becoming too much to find the average of the points with a centroid. The final cluster of the data is likely to not appear to be the cluster you expect to see. You don't see the cluster form as you keep picking the middle of the points and group the points that are nearer to it. If the algorithm is run fast i.e. is greedy it could end up showing you results that you are not trying to get.

#### Options to deal with the issues:

The first option is to look for a fair number of groups to cluster. Since this algorithm runs on as many as millions of data you want to take a chunk of that data and run the hierarchical clustering algorithm. This basically puts the clusters into two as pairs to get an idea of the data being clustered that makes sense with a number of clusters k that makes sense. So the clustered data is being divided as it goes along the hierarchy. You can also put your knowledge of examples of data that have been measured. Examples such as the number of colours of the balls you play snooker with. You could put a cluster groups of that number.

One option is to run the algorithm with different  $k$  numbers of cluster in the data. So try different numbers to see the different results you get. This is to see which numbers of clusters  $k$  give results that seem to make sense. This is by following the Elbow method. This method is worked on to measure how good the results are. See if you can make sense out of it.

The Elbow method is shown on a graph that looks just like an elbow. Before I explain that I have to explain the sums you have to do. It starts with working out the 'within-cluster sum of squares' (WCSS). This goes on the y-axis of the graph. This is finding the distance of each data point to its centroid in the first cluster. Find each distance add them all up and square the answer. That gives the variability of the first cluster. You do the same for each cluster, however many you have, to get the variability of them. The variabilities are then added together to get WCSS. We prefer to have smaller variabilities of each cluster. This means the points are less spread out i.e. closer to the centroid. If you put  $k$  number of clusters as 1 you're grouping the data into one large cluster and so the variability will be large. So  $k$  being 1 seems not a right no. of clusters.

So the Elbow plot on the graph is an easier way to find the right  $k$ . We have the WCSS on the y-axis that you have worked out for every number of clusters. The  $k$  clusters are on the x-axis. Plotting it on the graph as a curve for every  $k$  should show a curve looking like an elbow.

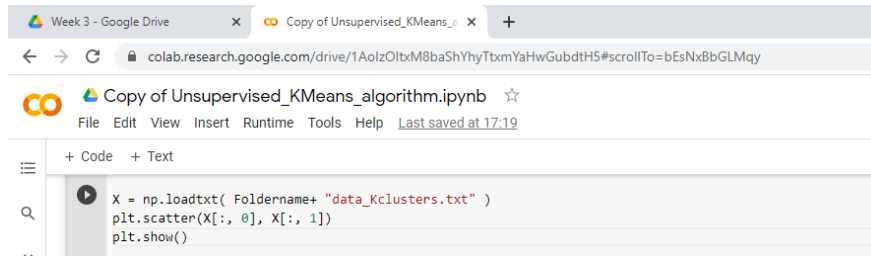
The elbow curve is likely to go flat as you increase the  $k$  and the WCSS decreases. So bringing  $k$  up does not quite help. From  $k$  being 1 to 3 the curve gradually goes down to form the elbow. Say a rapid decrease from 8000 to 3000 if  $k$  goes from 1 to 2. And then a gradual decrease from 3000 to 1000 when it goes from 2 to 3. At this point is the elbow. So 3 at the elbow is the best number of clusters so far. Increasing  $k$  upto a million brings the WCSS down to 0. This result is not worth having. It is about trying different values of  $k$  to see which is the best result that you get.

### Dealing with unfair centroids

If the centroids are chosen in the first place in an awkward position like two centroids being really close to each other the algorithm continues to put each of them into a group. The centroids stay in the same place and the cluster is put into groups. Although the grouping seems to be awkward with one centroid outside the groups. So far this is not a very good one. This can happen because you randomly pick the centroids in the first place. The way to solve this problem is to try different random pickings of centroids and see which one gives you the best result. Or the result you think is better. You get different results by running the algorithm with different pickings of centroids in the first place. So it is matter of choosing the one that seems to make sense. Checking the WCSS of the number of clusters lets you find out which variability of each cluster is smaller to get the cluster that makes sense. Also you don't know if the centroids you have chosen first are correct or not. This is

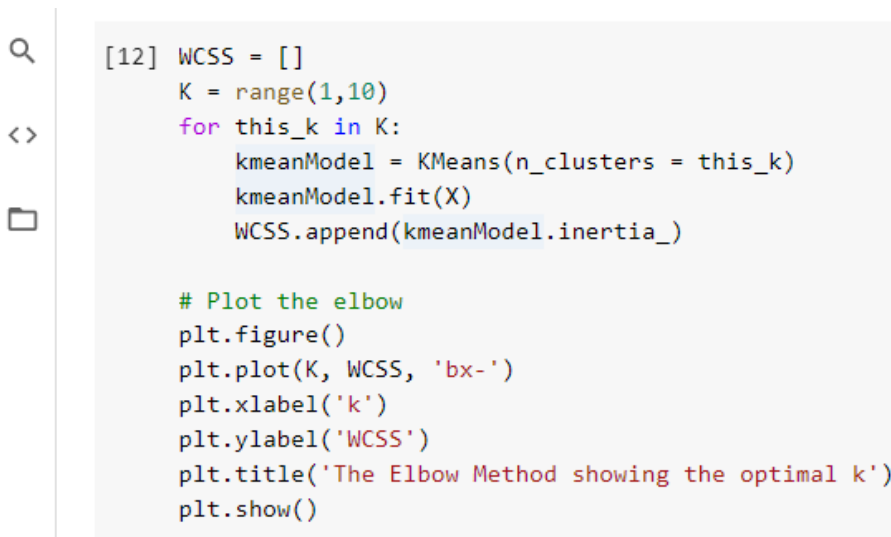
because there could be millions of data to look at when you're working on 1000s out of that to put into a cluster.

## Task 2

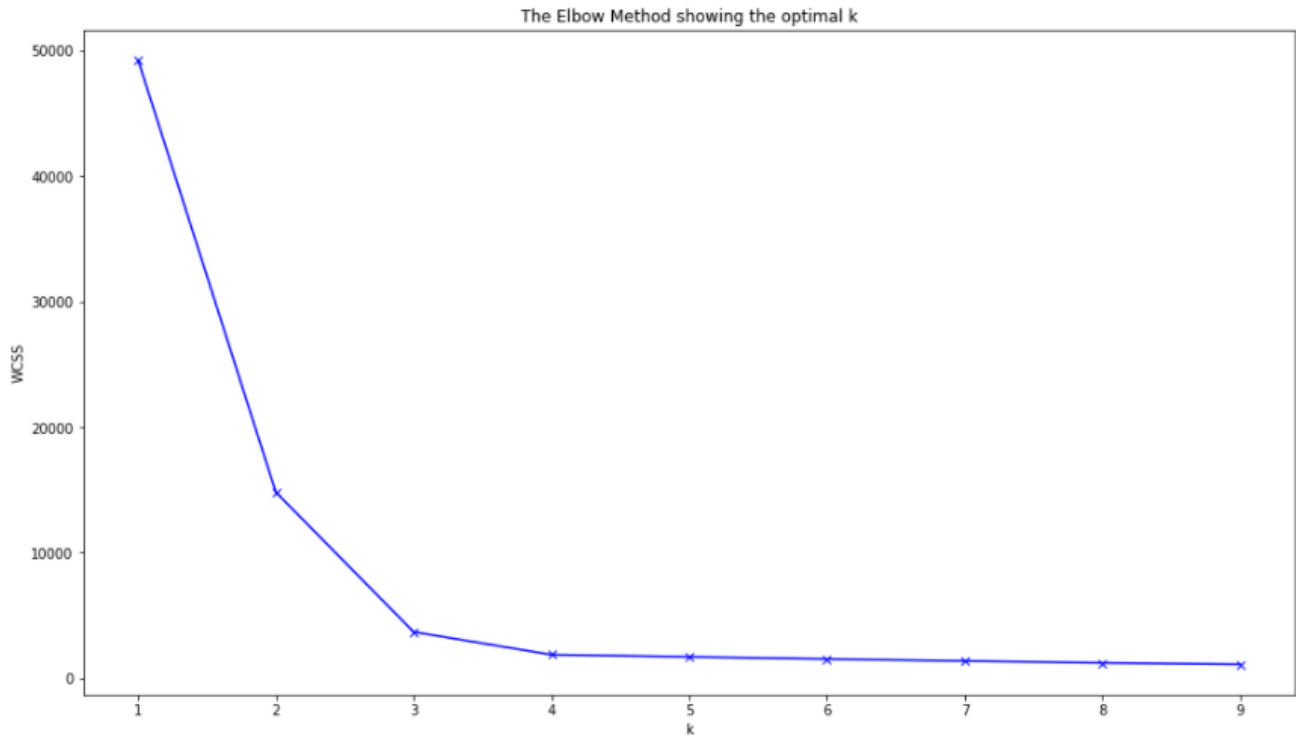


This task is on the set of data I have been given which has 1000 values. The no. of clusters is not yet known. I load file having the set of data into the Python script to make the code give the plot of the data ready to be clustered into k no. of groups.

Running the Elbow method in code written in Python on my Colab Notebook to find the better value of k number of clusters.



Starting with the values of k being from 1 to 10 on the x-axis. This is the range(1,10). Followed by the for loop for all the worked out values for k number of clusters. Then the lines of each with the dot notation to plot the graph of the elbow curve. There is also the labels for the plot in quotation marks. We need the dot notation to show the plot.



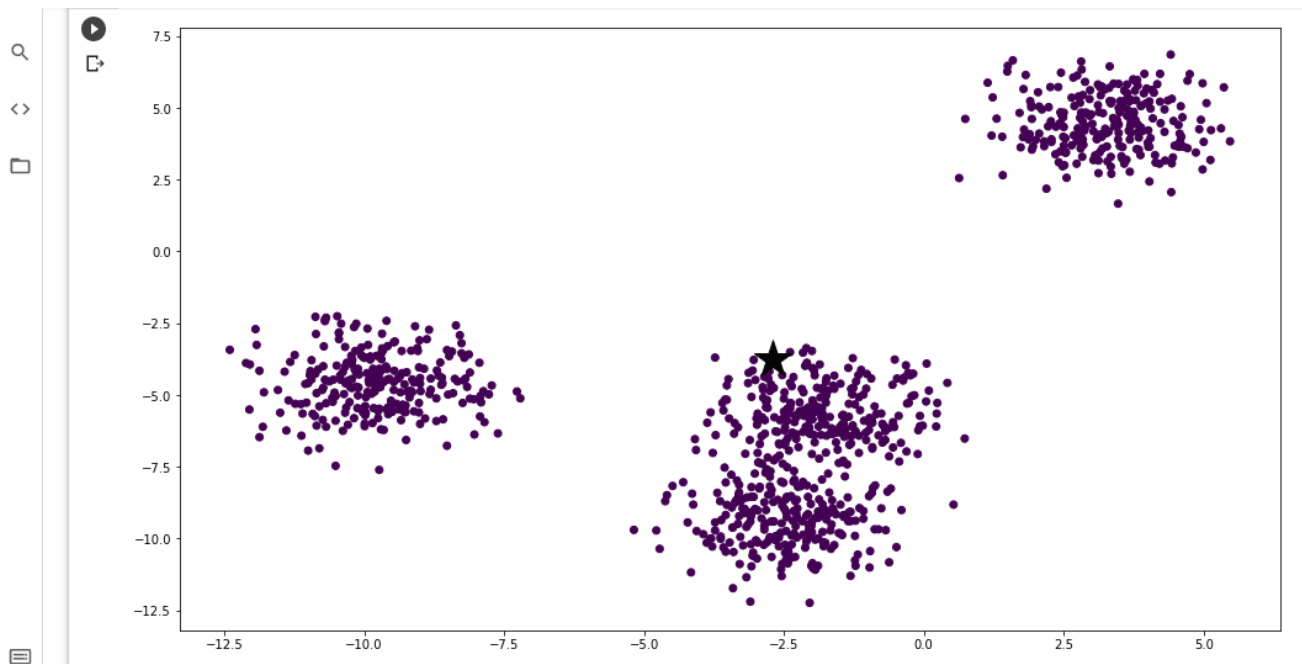
So here is the plot of the elbow. Like I described in task 1 the values of k up to 4 seem to help forming the elbow. The point at which it bends is 3. So up to 4 the WCSS is short enough. This means the variability of their cluster groups is short enough to give a result that makes sense. We can tell the best value of k here is between 2 and 4. It seems 3 is the best value.

```
# Initializing KMeans
K_elbow = 1 # replace this value by what you found in elbow method
kmeans = KMeans(n_clusters = K_elbow)
# Fitting with inputs
kmeans = kmeans.fit(X)
# Predicting the clusters
labels = kmeans.predict(X)
# Getting the cluster centers
C = kmeans.cluster_centers_

plt.figure()
plt.scatter(X[:, 0], X[:, 1], c=labels)
# plt.scatter(X[:, 0], X[:, 1], c=labels)
plt.scatter(C[:, 0], C[:, 1], marker='*', c='#050505', s=1000)
plt.show()
```



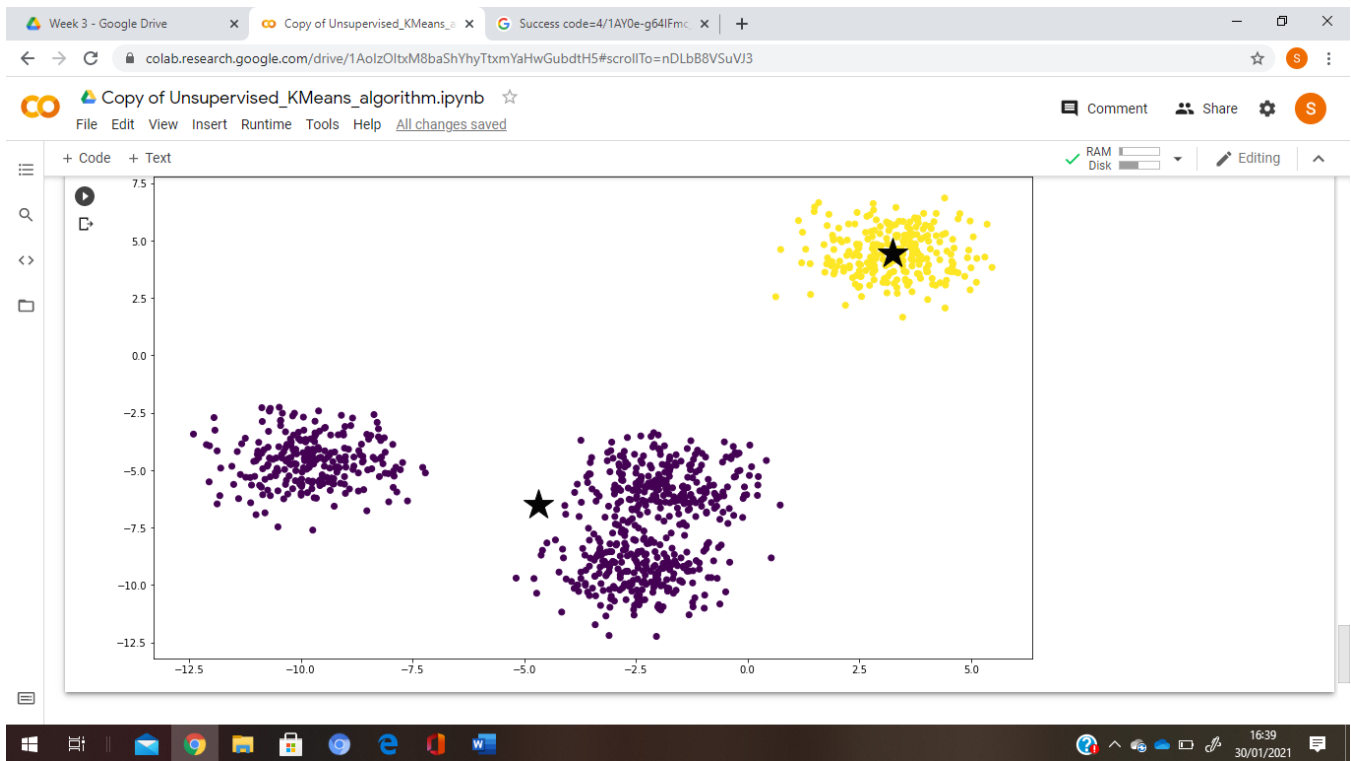
Running the k-means algorithm to see which k gives the best number of clusters starting with 1. I Put 1 in the k\_elbow variable – the kMeans algorithm states in the code above that the no.of clusters is the number put in the k\_elbow.



One group of clusters is produced on the plot. It has one centroid. This tells us that putting the no,of clusters as 1 does not give the best result as it is just 1 group. Plus the distance of the points being calculated to the only centroid for this cluster.

# Logbook for Machine Learning

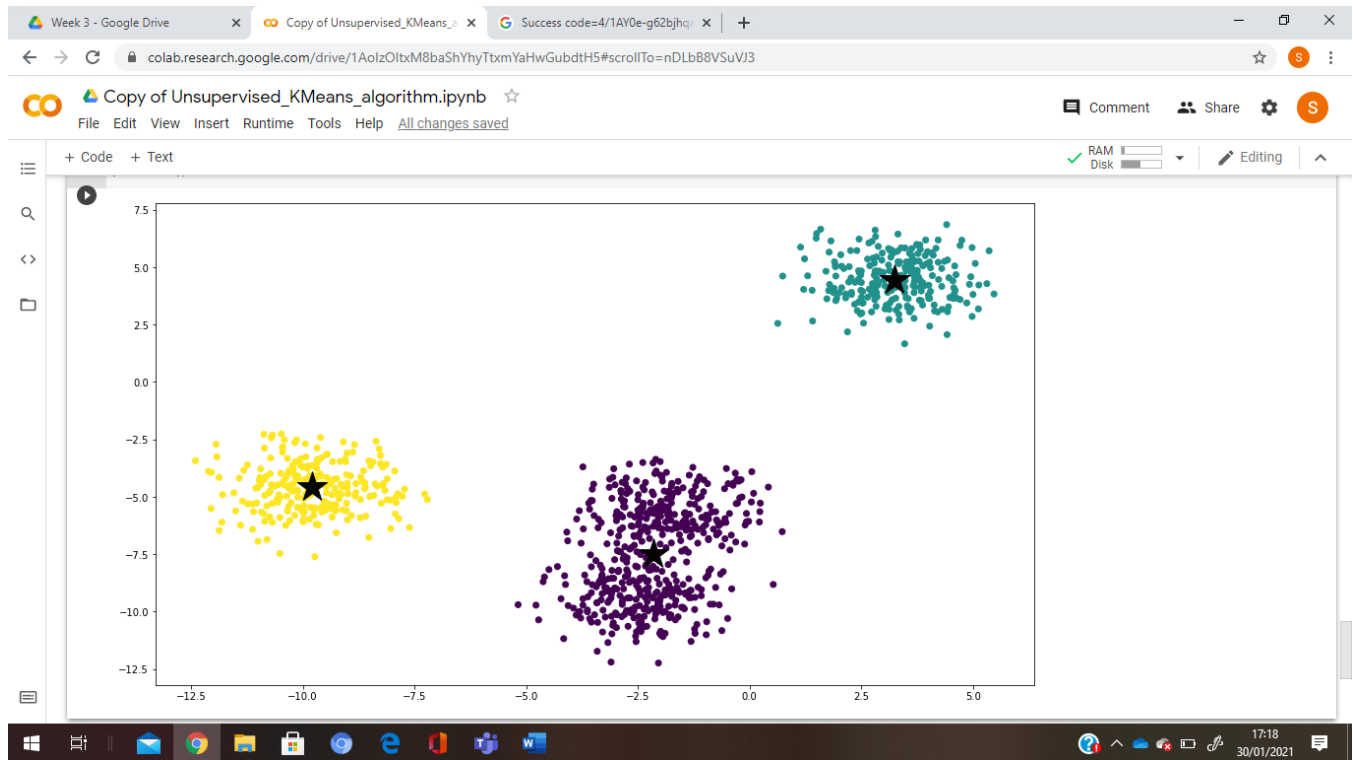
...



Running kmeans with 2 as no.of clusters gives a better result. Compared to 1 we have the data put into 1 big group and 1 small group of clusters. The points in the bigger group are more spread out from the centroid meaning a bigger variability adding up to a bigger WCSS than with 3 clusters. Having points closer to the centroid in the smaller group does not make this look the best result yet.

## Logbook for Machine Learning

...



Running kmeans as 3 seems to produce a better looking result. Each group of clusters have a centroid and are not too close too each other. It is how far the points are to their centroid than the distance between the clusters. Infact the data points seem to be close enough to their centroid per cluster kaming them fairly compact. So according to the elbow where it bends at 3 and the plot above it seems  $k = 3$  is the best value. Running kmeans as 4, 5 or up to 9 shows clusters that are next to each other. Therefore I choose  $k = 3$  as the best value of  $k$ .

Week 5

## Week 9

### Task 1

A Multi-layer perceptron can address the limitation of a single-layer Perceptron. A single-layer has a line of best fit to separate the perceptron training. It's just a line going through the points that best fits through them. It does not exactly separate the different types of points. The multi-layer has a line that separates the different points by moving around them. The line is shaped rather than being straight separating the different points.

## Week 4

## Task 1

Having a model to predict the no.of people who will be put in hospital the following week to be dialysed. The model is a confusion matrix having True Positive = 33, False Negative = 17, True Negative = 40 and False Positive = 10

The performance metrics I have calculated:

$$\text{Recall (Sensitivity)} = TP / TP + FN$$

$$= 33 / (33 + 17)$$

$$= 33 / 50$$

$$= 0.66$$

$$\text{Specificity} = TN / FP + TN$$

$$= 40 / (10 + 40)$$

$$= 40 / 50$$

$$= 0.8$$

$$\text{Accuracy} = TP + TN / TP + TN + FP + FN$$

$$= 33 + 40 / (40 + 17 + 10 + 33)$$

$$= 73 / 100$$

$$= 0.73$$

$$\text{Precision} = TP / TP + FP$$

$$= 33 / (33 + 10)$$

$$= 33 / 43$$

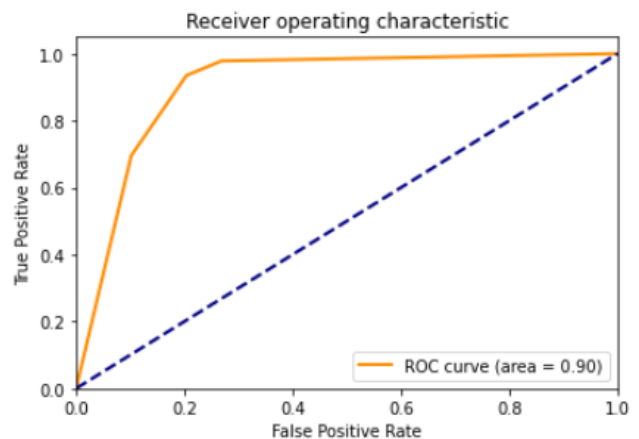
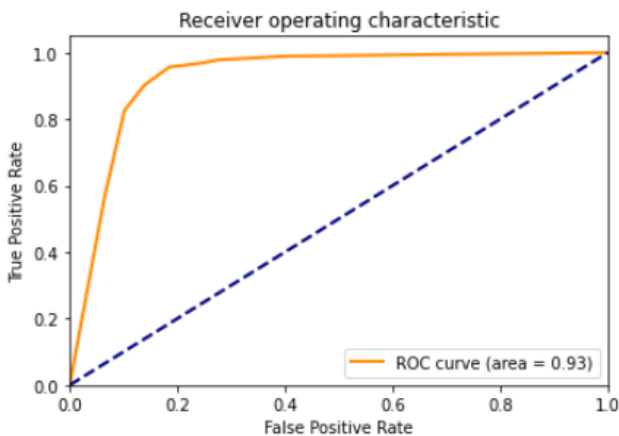
$$= 0.76$$

## Task 2

```
[191] ## split into train/test sets
      from sklearn.model_selection import train_test_split
      trainX, testX, trainy, testy = train_test_split(X, y, test_size=0.2, random_state=2)

# train a classifier
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors=3)
model.fit(trainX, trainy)

KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=None, n_neighbors=3, p=2,
                    weights='uniform')
```



The curve of number of neighbours  $k = 7$  and the curve of no. of neighbours  $k = 3$ .

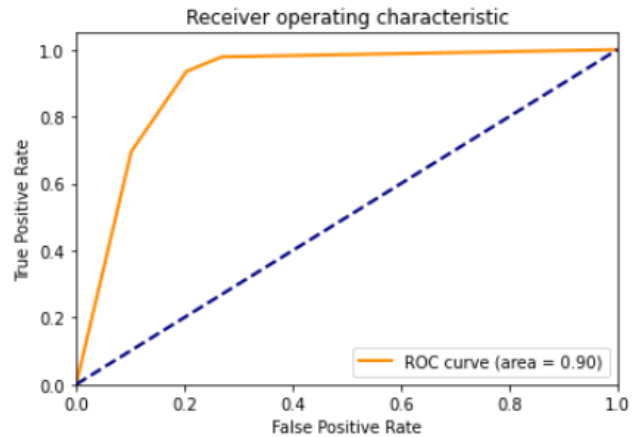
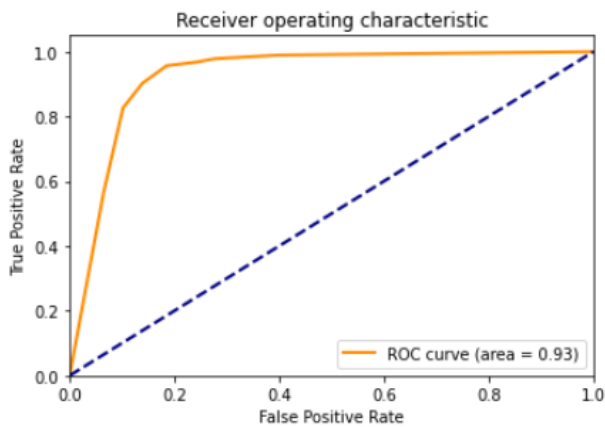
Looking at the two different curves the curve of the higher number of neighbours  $k$  being 7 is the better result. It curves in the top left corner higher than the one for 3 neighbours. The sensitivity is higher for 7. The anti-specificity becomes higher too.

Both ROC curves tell us the item being tested is put in the class very well. It is being classified very well. The curves are not of a classifier that is false. They both start at 0 and move up towards the right.

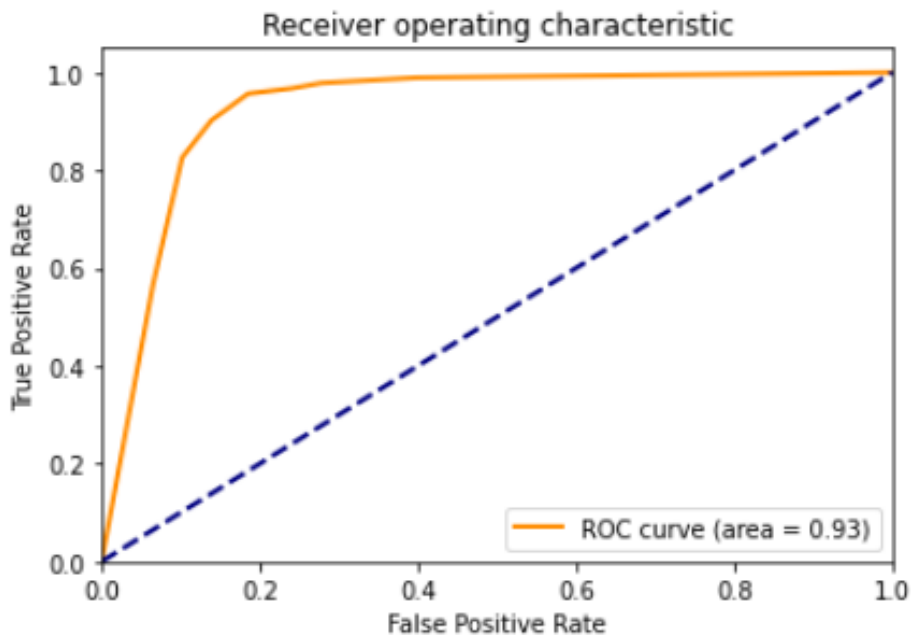
Like a positive correlation of a scatter graph. A curve of the item not being put in the class very well i.e. in the wrong class starts at 0 and moves towards the right and then goes up. That is a classifier that is false.

Area Under The Curve: AUC = 0.926

Area Under The Curve: AUC = 0.904



The area under the ROC curve of k being 7 is calculated to be 0.926. That is 0.93 to 2 decimal places. This is a big area close to 1. It is bigger than the area of k being 3 which is 0.904. That is 0.90 to two decimal places. The two areas are very close to each other. They are very close to 1.





I have to point out that the x-axis gives the anti- specificity. This is basically 1 minus the specificity. If the specificity is 0.8 the anti of that is 0.2 given by the x-axis. So 1 minus 0.8. Basically the specificity is being read backwards as it increases to draw the ROC curve. It starts at 1 instead of 0 and goes up to 0 instead of 1.

```
from sklearn.metrics import confusion_matrix
cnf_matrix = confusion_matrix(testy, model.predict(testX))
TP = cnf_matrix[0][0]
FP = cnf_matrix[0][1]
FN = cnf_matrix[1][0]
TN = cnf_matrix[1][1]
print('TP = ', TP)
print('FP = ', FP)
print('FN = ', FN)
print('TN = ', TN)

print('\nPerformance Metrics')
#Sensitivity, hit rate, recall, or true positive rate
print('TPR = Sensitivity = recal = '+'{:.2f}'.format(TP/(TP+FN)))
# Specificity or true negative rate
print('TNR = Specificity = '+'{:.2f}'.format(TN/(TN+FP)))
# Precision or positive predictive value
print('PPV = Precision = '+'{:.2f}'.format(TP/(TP+FP)))
# Overall accuracy
print('ACC = Accuracy = '+'{:.2f}'.format((TP+TN)/(TP+FP+FN+TN)))
```

```
TP = 86
FP = 22
FN = 6
TN = 86
```

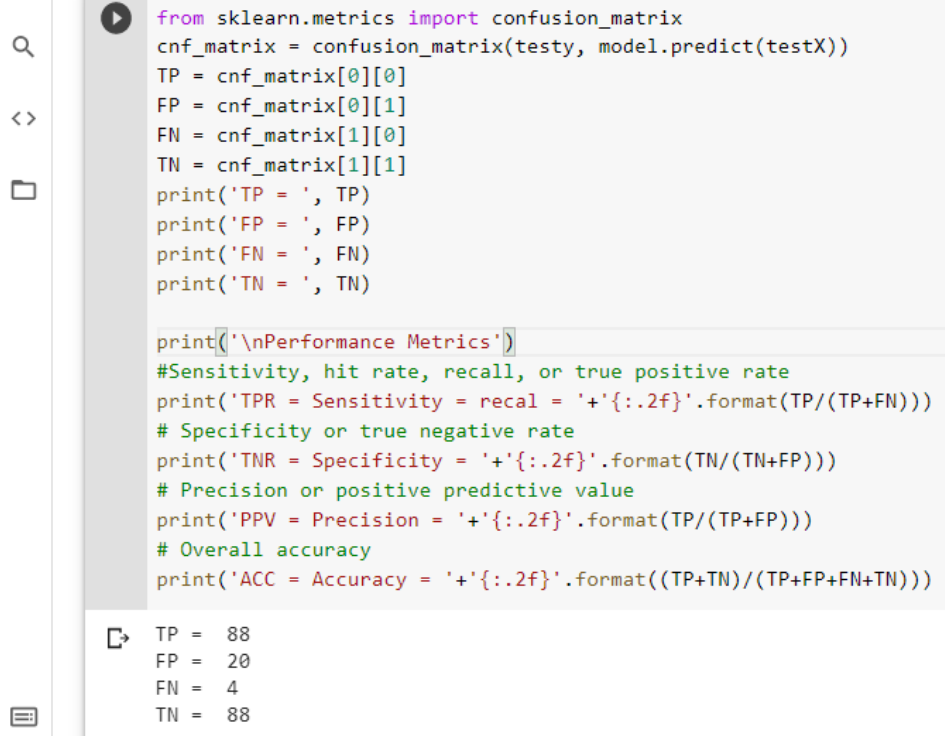
The breakdown of the data into true and false positives and true and false negatives before being shown in the confusion matrix. This being for  $k = 3$  neighbours. The code for the confusion matrix is in binary 0 and 1. There are 0 or 1s for each data that is true positive, false positive, false negative and true negative.

```
print('\nPerformance Metrics')
#Sensitivity, hit rate, recall, or true positive rate
print('TPR = Sensitivity = recal = '+'{:.2f}'.format(TP/(TP+FN)))
# Specificity or true negative rate
print('TNR = Specificity = '+'{:.2f}'.format(TN/(TN+FP)))
# Precision or positive predictive value
print('PPV = Precision = '+'{:.2f}'.format(TP/(TP+FP)))
# Overall accuracy
print('ACC = Accuracy = '+'{:.2f}'.format((TP+TN)/(TP+FP+FN+TN)))
```

```
TP = 86
FP = 22
FN = 6
TN = 86
```

```
Performance Metrics
TPR = Sensitivity = recal = 0.93
TNR = Specificity = 0.80
PPV = Precision = 0.80
ACC = Accuracy = 0.86
```

The same formulas I used in task 1 to calculate the sensitivity, precision, accuracy and specificity are in each line of code above the breakdown of the data. Brackets are needed to work them out correctly. These metrics of performance are given.



```
from sklearn.metrics import confusion_matrix
cnf_matrix = confusion_matrix(testy, model.predict(testX))
TP = cnf_matrix[0][0]
FP = cnf_matrix[0][1]
FN = cnf_matrix[1][0]
TN = cnf_matrix[1][1]
print('TP = ', TP)
print('FP = ', FP)
print('FN = ', FN)
print('TN = ', TN)

print('\nPerformance Metrics')
#Sensitivity, hit rate, recall, or true positive rate
print('TPR = Sensitivity = recal = '+ '{:.2f}'.format(TP/(TP+FN)))
# Specificity or true negative rate
print('TNR = Specificity = '+ '{:.2f}'.format(TN/(TN+FP)))
# Precision or positive predictive value
print('PPV = Precision = '+ '{:.2f}'.format(TP/(TP+FP)))
# Overall accuracy
print('ACC = Accuracy = '+ '{:.2f}'.format((TP+TN)/(TP+FP+FN+TN)))
```

TP = 88  
FP = 20  
FN = 4  
TN = 88

The breakdown of the data into true and false positives(TP & FP) and true and false negatives(TN & FN) before being shown in the confusion matrix. This being for  $k = 7$  neighbours. The same code for the confusion matrix is above that in 0s and 1s.

```
print('\nPerformance Metrics')
#Sensitivity, hit rate, recall, or true positive rate
print('TPR = Sensitivity = recal = '+'{:.2f}'.format(TP/(TP+FN)))
# Specificity or true negative rate
print('TNR = Specificity = '+'{:.2f}'.format(TN/(TN+FP)))
# Precision or positive predictive value
print('PPV = Precision = '+'{:.2f}'.format(TP/(TP+FP)))
# Overall accuracy
print('ACC = Accuracy = '+'{:.2f}'.format((TP+TN)/(TP+FP+FN+TN)))
```

```
TP = 88
FP = 20
FN = 4
TN = 88
```

```
Performance Metrics
TPR = Sensitivity = recal = 0.96
TNR = Specificity = 0.81
PPV = Precision = 0.81
ACC = Accuracy = 0.88
```

---

The same for this k number of neighbours. With the same formulas to work out each performance metric. Each of them including the Sensitivity and Specificity are higher than those for 3 neighbours. Therefore the k number of neighbours being 7 gives the better result.