Q1.

Problem 1

(a) Using Euclidean distance. We have the Point $(2,-1,-2)$

First, we need to get the Euclidean distances between $(2,-1,-2)$ and other Points. $dis(X,Y) = \sqrt{\sum_{i=1}^{3}(X_i - Y_i)^2}$

$$d(O,C_1) = \sqrt{\sum_{i=1}^{3}(O_i - C_{1i})^2} = \sqrt{(2--3)^2 + (-1+2)^2 + (-2-2)^2}$$
$$= \sqrt{42} \simeq 6.481$$

$$d(O,C_2) = \sqrt{(2+1)^2 + (-1-4)^2 + (-2+1)^2} = \sqrt{35} \simeq 5.9161$$

$$d(O,C_3) = \sqrt{(2-1)^2 + (-1-3)^2 + (-2-0)^2} = \sqrt{21} \simeq 4.5826$$

$$d(O,S_1) = \sqrt{(2+1)^2 + (-1+1)^2 + (-2-3)^2} = \sqrt{34} \simeq 5.831$$

$$d(O,S_2) = \sqrt{(2-1)^2 + (-1-2)^2 + (-2-1)^2} = \sqrt{19} \simeq 4.3589$$

$$d(O,S_3) = \sqrt{(2-2)^2 + (-1+1)^2 + (-2+1)^2} = 1$$

$$d(O,T_1) = \sqrt{(2-0)^2 + (2+1)^2 + (-2+1)^2} = \sqrt{14} \simeq 3.7417$$

$$d(O,T_2) = \sqrt{(2-3)^2 + (-1+2)^2 + (-2-2)^2} = \sqrt{18} \simeq 4.24264$$

→ sort row wise $[1, \sqrt{14}, \sqrt{18}, \sqrt{19}, \sqrt{21}, \sqrt{34}, \sqrt{35}, \sqrt{42}]$

with $k_1 = 1$: the Point S_3 is the closest Point to O, as $d(O,S_3)$ is the min dist compared to the others " then the Prediction is Square

In k=4, Getting mode of (arr[:4]), we can notice that there is a tie, as there are 2 squares and 2 triangles.
→ According to research, there are multiple solutions to avoid tie, such as inc. k or selected the nearest one

$Sum(sq) = 1+\sqrt{19} \approx 5.3589$
$Sum(T) = \sqrt{14}+\sqrt{18} \approx 7.984$] → sq dis < dis(t)

Then, the Predicted class is Square.

With k=7, Now, we need to get mode(arr[:7]). we have 3 square, 2 Triangle samples and 2 circle samples. The square wins according to the methodology of "Getting majority".
If we think about the Σ dist of each class, the square also wins. In this sense, the Square class wins "Predicted"

[b]

Training set = {C_1, C_2, S_1, S_2, T_1}, Validation Set = {C_3, S_3, T_2}

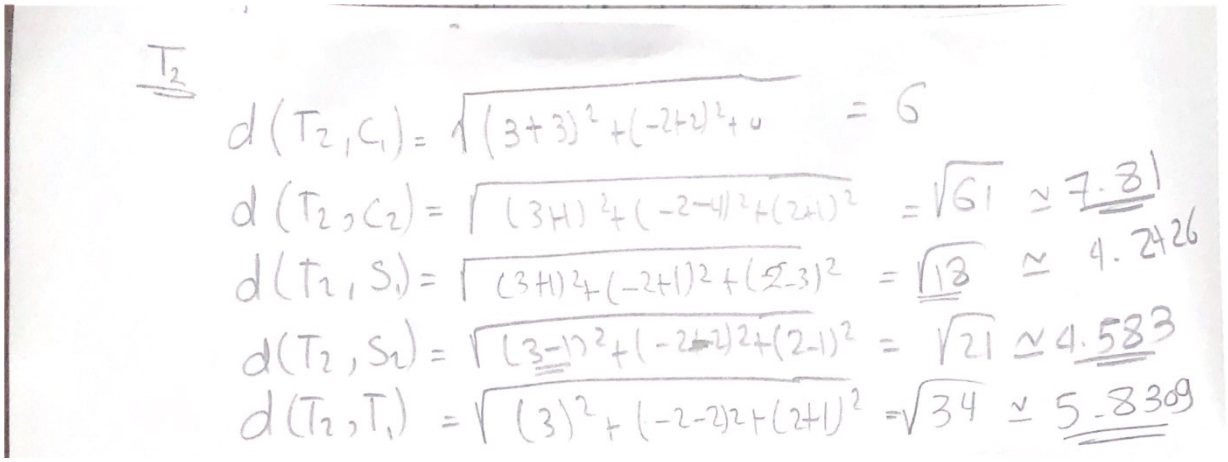
dist diff between validation and training:

C_3

$d(C_3,C_1) = \sqrt{(1+3)^2+(3+2)^2+2^2} = 3\sqrt{5} \approx 6.708$
$d(C_3,C_2) = \sqrt{(1+1)^2+(3-4)^2+(1)^2} = \sqrt{6} \approx 2.4494$
$d(C_3,S_1) = \sqrt{(1+1)^2+(3+1)^2+(3)^2} = \sqrt{29} \approx 5.3852$
$d(C_3,S_2) = \sqrt{(1-1)^2+(3-2)^2+1^2} = \sqrt{2} \approx 1.414$
$d(C_3,T_1) = \sqrt{1^2+1^2+1^2} = \sqrt{3} \approx 1.7321$

S_3

$d(S_3,C_1) = \sqrt{(2+3)^2+(-1+2)^2+(-1-2)^2} = \sqrt{35} \approx 5.916$
$d(S_3,C_2) = \sqrt{(2+1)^2+(-1-4)^2+(-1+1)^2} = \sqrt{34} \approx 5.83$
$d(S_3,S_1) = \sqrt{(2+1)^2+(-1+1)^2+(-1-3)^2} = 5$
$d(S_3,S_2) = \sqrt{(2-1)^2+(-1-2)^2+(-1-1)^2} = \sqrt{14} \approx 3.74$
$d(S_3,T_1) = \sqrt{(2-0)^2+(-1-2)^2+(-1+1)^2} = \sqrt{13} \approx 3.6056$



After getting the euclidean distance between (T2, C3, and S3) and (training dataset), we need to get the curve per k. The possible values for k are from 1 to 5. Unfortunately, we do not have the capability of drawing each k curve. In this sense, we will need to use the concept of threshold. Getting the threshold will help us get some points to construct a full ROC per k. We need to get the points per class for each threshold.

Let's get the three tables for threshold.

K = 1 and Threshold = 0.

Actual in row and Predicted in Column
Circle:

	Predicted Circle	**Predicted non-Circle**
Actual Circle	**1**	**0**
Actual non-circle	**2**	**0**

Square:

	Predicted Square	**Predicted non-Square**
Actual Square	**1**	**0**
Actual Square	**2**	**0**

Triangle:

	Predicted Triangle	**Predicted non-Triangle**
Actual Triangle	**1**	**0**
Actual Triangle	**2**	**0**

$$TPR_{micro} = \frac{TP\ for\ each\ class}{TP\ for\ each\ class + FN\ for\ each\ class} = \frac{1+1+1}{1+1+1+0+0+0} = 1$$

$$FPR_{micro} = \frac{FP\ for\ each\ class}{FP\ for\ each\ class + FN\ for\ each\ class} = \frac{2+2+2}{2+2+2+0+0+0} = 1$$

Threshold = 1

Circle:

	Predicted Circle	**Predicted non-Circle**
Actual Circle	**0**	**1**
Actual non-circle	**0**	**2**

Square:

	Predicted Square	**Predicted non-Square**
Actual Square	**0**	**1**
Actual Non-Square	**2**	**0**

Triangle:

	Predicted Triangle	**Predicted non-Triangle**
Actual Triangle	**0**	**1**
Actual Non-Triangle	**1**	**1**

$$TPR_{micro} = \frac{TP\ for\ each\ class}{TP\ for\ each\ class + FN\ for\ each\ class} = \frac{0}{1+1+1} = 0$$

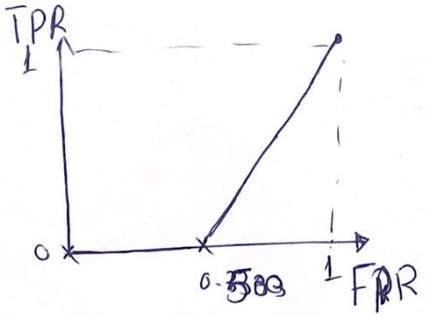
$$FPR_{micro} = \frac{FP\ for\ each\ class}{FP\ for\ each\ class + FN\ for\ each\ class} = \frac{3}{2+4} = \frac{1}{2}$$

Similarly, we will do with k = 2, k = 3 , k = 4 , k = 5 with different thresholds. "I did not mention the other tables as they are a lot, so I did in my notebook the other results only.

K=1

we have two points

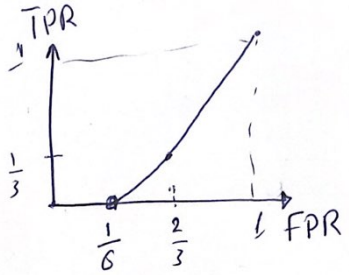
(1,1) and (0, 0)



K=2

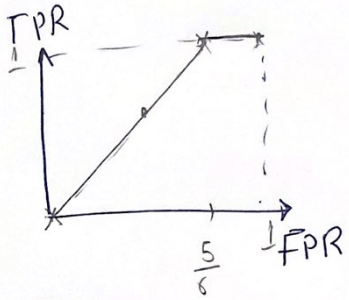
FirstPoint (1,1), 2nd $(\frac{2}{3}, \frac{1}{3})$

thirdPoint $(\frac{1}{6}, 0)$



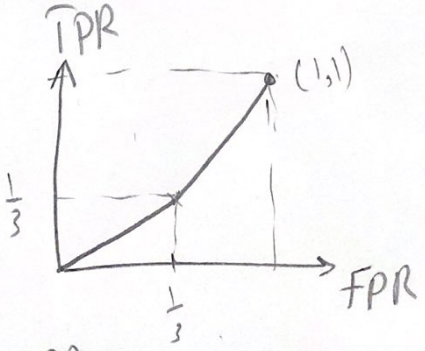
K=3

FirstPoint (1,1), and $(\frac{5}{6}, 1)$



K=4

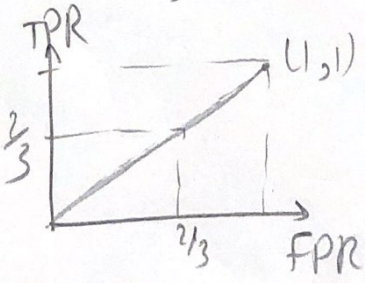
FirstPoint(1,1), $(\frac{1}{3}, \frac{1}{3})$



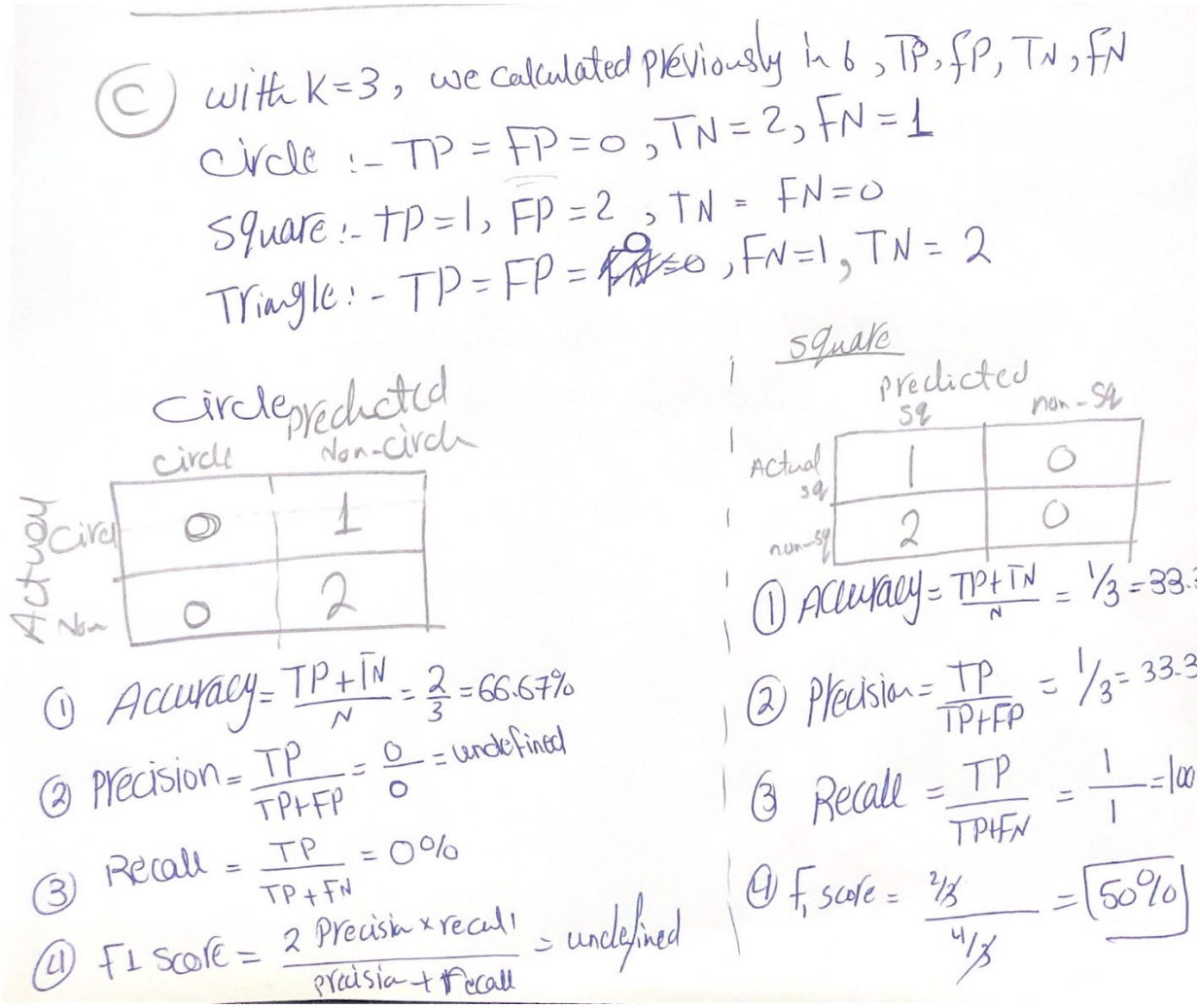
K=5

(1,1), $(\frac{2}{3}, \frac{2}{3})$

(0,0)



From the 5 graphs above, it is easy to notice that the best curve is k = 3, as it has the best AUC.



Triangle (pred)

Actual	T	Not T
T	0	1
Not T	0	2

① Accuracy = $\frac{2}{3}$ = 66.67%

② Precision = $\frac{0}{0}$ (undefined)

③ Recall = 0

④ F1 score = $2\left[\frac{(\frac{0}{0})(0)}{(\frac{0}{0})+(0)}\right]$ = undefined

(d) Use K=3

From the array. we get the majority of $[1, \sqrt{14}, \sqrt{13}]$ (sq)

In this sense, we have 2 Points ⇒ Triangle and only one Point is Square. the Predicted class is triangle.

With the Predicted class with K = 1, 4, 7, we got square as an aswer; while we get K=3 with Predicted class triangle. Also, we had the Problem of tie with K=4 ⇒ Solved using the nearest. With K=3, we did not have this Problem, as it is difficult to have a tie, but not impossible, as we might have [T, S, C].

Q2.

After using the k = sqrt(2267), I got the results required below.

First screenshot shows the precision, recall, f1-score per class.

```
In [10]: from sklearn.metrics import classification_report
         print(classification_report(y_test, y_pred, target_names=['class 0','class 1', 'class 2', 'class 3', 'class 4', 'c
```

	precision	recall	f1-score	support
class 0	0.96	0.99	0.97	816
class 1	0.88	1.00	0.93	909
class 2	0.99	0.90	0.94	846
class 3	0.94	0.93	0.94	937
class 4	0.97	0.93	0.95	839
class 5	0.94	0.94	0.94	702
class 6	0.95	0.98	0.97	785
class 7	0.93	0.94	0.93	893
class 8	0.98	0.87	0.92	835
class 9	0.89	0.94	0.92	838
accuracy			0.94	8400
macro avg	0.94	0.94	0.94	8400
weighted avg	0.94	0.94	0.94	8400

The screenshot below gives **accuracy** per class

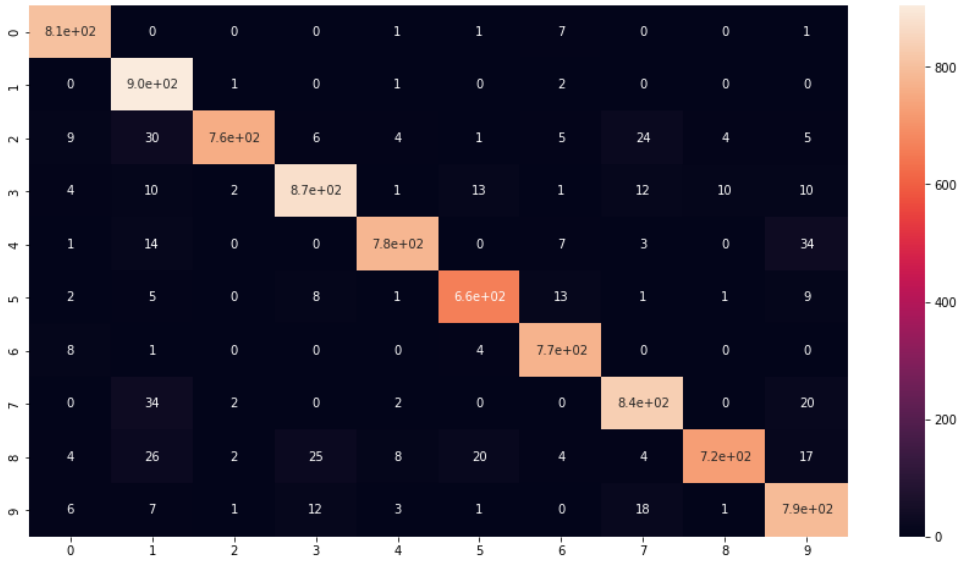
Accuracy per class

```
In [109]: conf_mat.diagonal()/conf_mat.sum(axis=1)
Out[109]: array([0.9877451 , 0.99559956, 0.89598109, 0.93276414, 0.92967819,
                 0.94301994, 0.98343949, 0.93505039, 0.86826347, 0.94152745])
```

The screenshot below is the confusion matrix used to calculate the accuracy above.

In [11]:
```
import seaborn as sns
df_cm = pd.DataFrame(conf_mat)
plt.subplots(figsize=(15,8))
sns.heatmap(df_cm, annot=True)
```

Out[11]: <AxesSubplot:>




```
print('Using Micro average')

print('Micro-averaged precision = {:.2f} (treat instances equally)'
      .format(precision_score(y_test, y_pred, average = 'micro')))
print('Micro-averaged Recall = {:.2f} (treat instances equally)'
      .format(recall_score(y_test, y_pred, average = 'micro')))
print('Micro-averaged F1_Score = {:.2f} (treat instances equally)'
      .format(f1_score(y_test, y_pred, average = 'micro')))

print('Using Macro average')
print('Macro-averaged precision = {:.2f} (treat classes equally)'
      .format(precision_score(y_test, y_pred, average = 'macro')))
print('Macro-averaged Recall = {:.2f} (treat classes equally)'
      .format(recall_score(y_test, y_pred, average = 'macro')))

print('Macro-averaged F1_Score = {:.2f} (treat classes equally)'
      .format(f1_score(y_test, y_pred, average = 'macro')))
```

```
Accuracy = 0.9411904761904762
Using Micro average
Micro-averaged precision = 0.94 (treat instances equally)
Micro-averaged Recall = 0.94 (treat instances equally)
Micro-averaged F1_Score = 0.94 (treat instances equally)
Using Macro average
Macro-averaged precision = 0.94 (treat classes equally)
Macro-averaged Recall = 0.94 (treat classes equally)
Macro-averaged F1_Score = 0.94 (treat classes equally)
```

The AUC per class is

In [26]:
```
# With a help of https://stackoverflow.com/questions/56227246/how-to-calculate-roc-auc-score-hav

from sklearn.metrics import roc_curve
from sklearn.metrics import auc

digits = list(Y.unique().flatten())
digits.sort()

fpr = dict()
tpr = dict()
roc_auc = dict()

i=0
for digit in digits:
    fpr[i], tpr[i], thresholds = roc_curve(yt[:,digit],proba[:,digit])
    auroc = auc(fpr[i], tpr[i])
    print('The Digit: ',digit,'--AUC--->',auroc)
    i=i+1
```

```
The Digit:  0 --AUC---> 0.9991658781128485
The Digit:  1 --AUC---> 0.999214385462041
The Digit:  2 --AUC---> 0.9958222781786737
The Digit:  3 --AUC---> 0.9926905140421669
The Digit:  4 --AUC---> 0.99547462915447
The Digit:  5 --AUC---> 0.9947881160533797
The Digit:  6 --AUC---> 0.9983788282429499
The Digit:  7 --AUC---> 0.9918245024315491
The Digit:  8 --AUC---> 0.9921297655844952
The Digit:  9 --AUC---> 0.9875053574618476
```

b) I used the function GridSearchCV to get the best k, I found it k = 3 is the best k.

```
In [88]: k_range = list(range(1,32))
         dict_grid =dict(n_neighbors = k_range)
         Knn_best_k = KNeighborsClassifier(metric='euclidean')
         #Kfold-cross Validation with a range of k
         total_ks = GridSearchCV(Knn_best_k, dict_grid, cv = 10)
         gridSearch_= total_ks.fit(X_train,y_train)

In [91]: gridSearch_.best_params_

Out[91]: {'n_neighbors': 3}

In [ ]:
```

In the screenshot below, it proves that **k = 3** is much better than k = sqrt(2267) and other ks according to the searchCV function. We got accuracy = 97. Below you can see the accuracy per class.

```
In [100]: #predict with best k
          knn_euc = KNeighborsClassifier(n_neighbors = 3)

          knn_euc.fit(X_train,y_train)

          y_pred_euc = knn_euc.predict(X_test)

In [101]: print(classification_report(y_test, y_pred_euc, target_names=['class 0','class 1', 'class 2', 'class 3', 'class 4',
```

	precision	recall	f1-score	support
class 0	0.98	1.00	0.99	816
class 1	0.95	1.00	0.97	909
class 2	0.98	0.95	0.96	846
class 3	0.97	0.96	0.96	937
class 4	0.98	0.97	0.98	839
class 5	0.95	0.97	0.96	702
class 6	0.97	0.99	0.98	785
class 7	0.96	0.97	0.96	893
class 8	0.98	0.93	0.95	835
class 9	0.95	0.95	0.95	838
accuracy			0.97	8400
macro avg	0.97	0.97	0.97	8400
weighted avg	0.97	0.97	0.97	8400

From the screenshot below, **the accuracy per class is quite good compared to the other ks.**

```
The Digit:  9 --AUC---> 0.9922383081088144

In [110]: # getting the accuracy per class
          conf_mat_besteuc = confusion_matrix(y_test, y_pred_euc,labels = [0,1, 2, 3, 4, 5, 6, 7, 8, 9])
          conf_mat_besteuc
          conf_mat_besteuc.diagonal()/conf_mat_besteuc.sum(axis=1)

Out[110]: array([0.99509804, 0.99779978, 0.94562648, 0.95624333, 0.97258641,
                 0.96581197, 0.98853503, 0.96640538, 0.92694611, 0.9522673 ])
```

Finally, the last requirement in b is AUC per class. You can notice that now AUC is approaching to be perfect.

In [107]:

```
proba_euc = knn_euc.predict_proba(X_test)
digits = list(Y.unique().flatten())
digits.sort()

fpr = dict()
tpr = dict()


roc_auc = dict()

i=0
for digit in digits:
    fpr[i], tpr[i], thresholds = roc_curve(yt[:,digit],proba_euc[:,digit])
    auroc = auc(fpr[i], tpr[i])
    print('The Digit: ',digit,'--AUC--->',auroc)
    i=i+1
```

```
The Digit:  0 --AUC---> 0.9984889822226359
The Digit:  1 --AUC---> 0.9979526587019936
The Digit:  2 --AUC---> 0.9901613035474763
The Digit:  3 --AUC---> 0.9891753425758466
The Digit:  4 --AUC---> 0.9943126851153723
The Digit:  5 --AUC---> 0.9927265860300416
The Digit:  6 --AUC---> 0.9983267185533079
The Digit:  7 --AUC---> 0.9893278404881088
The Digit:  8 --AUC---> 0.988224845748028
The Digit:  9 --AUC---> 0.9922383081088144
```

C)

In the screenshot below, we noticed also that the best k is 3 using manhattan.

C. What is the best k with Manhattan Distance ?

```
In [96]: k_range = list(range(1,8))
         dict_grid_2 =dict(n_neighbors = k_range)
         Knn_best_k_2 = KNeighborsClassifier(metric='manhattan')
         #Kfold-cross Validation with a range of k
         total_ks_2 = GridSearchCV(Knn_best_k_2, dict_grid_2, cv = 10)
         gridSearch_1= total_ks_2.fit(X_train,y_train)

In [97]: gridSearch_1.best_params_

Out[97]: {'n_neighbors': 3}
```

From the screenshot below, it is easy to see that the accuracy is the best using manhattan distance. I will mention below compared with **b.**

```
         y_pred_man = knn_man.predict(X_test)

In [104]: print(classification_report(y_test, y_pred_man, target_names=['class 0','class 1', 'class 2', 'class 3'
```

	precision	recall	f1-score	support
class 0	0.97	1.00	0.98	816
class 1	0.93	1.00	0.96	909
class 2	0.98	0.93	0.96	846
class 3	0.95	0.96	0.95	937
class 4	0.98	0.97	0.97	839
class 5	0.95	0.96	0.95	702
class 6	0.98	0.99	0.98	785
class 7	0.95	0.96	0.96	893
class 8	0.99	0.90	0.94	835
class 9	0.94	0.94	0.94	838
accuracy			0.96	8400
macro avg	0.96	0.96	0.96	8400
weighted avg	0.96	0.96	0.96	8400

```
In [105]: from sklearn.metrics import roc_auc_score

          from sklearn.preprocessing import label_binarize
```

Another important requirement is the accuracy per class. I used the confusion matrix to be able to get the accuracy.

```
In [111]: # getting the accuracy per class
          conf_mat_bestman = confusion_matrix(y_test, y_pred_man,labels = [0,1, 2, 3, 4, 5, 6, 7, 8, 9])
          conf_mat_bestman
          conf_mat_bestman.diagonal()/conf_mat_bestman.sum(axis=1)

Out[111]: array([0.99632353, 0.99889989, 0.93498818, 0.95731057, 0.96543504,
                 0.95726496, 0.98726115, 0.96304591, 0.8994012 , 0.94152745])

In [ ]:
```

Final requirement is the AUC, which is quite high still.

```
from sklearn.preprocessing import label_binarize


from sklearn.metrics import roc_curve
from sklearn.metrics import auc

proba = knn_man.predict_proba(X_test)
digits = list(Y.unique().flatten())
digits.sort()

fpr = dict()
tpr = dict()
roc_auc = dict()

i=0
for digit in digits:
    fpr[i], tpr[i], thresholds = roc_curve(yt[:,digit],proba[:,digit])
    auroc = auc(fpr[i], tpr[i])
    print('The Digit: ',digit,'--AUC--->',auroc)
    i=i+1
```

```
The Digit:  0 --AUC---> 0.9988954590934475
The Digit:  1 --AUC---> 0.9977609361523524
The Digit:  2 --AUC---> 0.9830687920103701
The Digit:  3 --AUC---> 0.988668323315693
The Digit:  4 --AUC---> 0.991850233910007
The Digit:  5 --AUC---> 0.9894504918212375
The Digit:  6 --AUC---> 0.9962196636708474
The Digit:  7 --AUC---> 0.9903635293136633
The Digit:  8 --AUC---> 0.9809863419228959
The Digit:  9 --AUC---> 0.9885429218697431
```

```
In [ ]:
```

Comment: From B and C first, we can see that Using Euclidean distance is better than using the Manhattan distance. The results show so. Also, K = 3 is the best regardless the distance function used.

Q3.

Age	Gender	Country	Signup	Lion King	Hero	StarTrek	Wall-E	Inception
28	M	EG	Feb,2018	2.8	3	4.4	4	4.1
22	F	DE	Mar,2016	4.1	4	3.6	4.1	4
24	F	EG	Jul, 2017	3.9	4.5	3.1	3.7	4.6
20	M	EG	Aug,2016	3.2	3.5	4.8	4.3	4.9
32	M	DE	Dec,2017	3.9	3.8	4.1	3.5	4.2

From the Table above, we need to do pre-processing in Gender, Country, and Sign Up. Age also can be rescaled. What I need to do is to make the Gender, Country categorical, as Gender and country features have just 2 categories for each. We can use One-Hot encoding for both of them for sake of accuracy. Age can be scaled using Max_ Min scaling. Regarding the Signup, we can start with March,2016 and end with Feb 2018. It is now easy to rescale it using Min_Max. It will be easy for me to calculate the rating predictions.

Age will be scaled using the following equation : $\frac{age - Age_{min}}{age_{max} - age_{min}}$

If we think about the Sign Up: Starting with March 2016 and ending with Feb 2018, so we have (9 months in 2016 + 2 months in 2018 + 12 months in 2017) = 23

Age(Scaled)	Male	Female	EG	DE	SignUP(Starting with March 2016)	Lion King	Hero	StarTrek	Well-E	Inception
$\frac{2}{3}$	1	0	1	0	1	2.8	3	4.4	4	4.1
$\frac{1}{6}$	0	1	0	1	0	4.1	4	3.6	4.1	4
$\frac{1}{3}$	0	1	1	0	$\frac{16}{23}$	3.9	4.5	3.1	3.7	4.6
0	1	0	1	0	$\frac{5}{23}$	3.2	3.5	4.8	4.3	4.9
1	1	0	0	1	$\frac{21}{23}$	3.9	3.8	4.1	3.5	4.2

a. We have a new user whose age is 26, Male, EG, Signed up in December 2016. What is his/her rating?

First, **his vector is [26, M, EG, December-2016].** We need to scale this vector to be compatible with the table above.

Age	M	F	EG	DE	SignUp
$\frac{26-20}{32-20} = \frac{1}{2}$	1	0	1	0	$\frac{9}{23}$

First, we need to normalize the whole table and the new user vector.

The Table row number (The user number).	The Normalization $\sqrt{\sum_{i=1}^{n=(number\ of\ features\ with\ rating)} x_i^2}$
1	8.5120176
2	8.9781834
3	9.083779
4	9.490904
5	8.9545323

The Table row number (The user number).	The Normalization,[ageScaled, M,F,EG,DE,Si $\sqrt{\sum_{i=1}^{n=(number\ of\ features\ without\ rating)} x_i^2}$
1	1.855921
2	1.4240
3	1.61091

4	1.43082
5	1.95797
The new user (a)	1.550199

Now, we can get the similarity between each user and user n using the equations

$w_i = sim(u_i, u_n) = \frac{\sum_{i=1}^{n} A_i x B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$, and $r_n = \frac{\sum_{i=1}^{n-1} w_i r_i}{\sum_{i=1}^{n-1} w_i}$

Keep in mind that I will use **3NN only as instructed in the question.**

So the Similarity will be calculated using cosine similarity by using the following equation $w_i = \frac{\sqrt{AgeScaled_i x\ ageScaled_n + M_i xM_n + F_i xF_n + EG_i xEG_n + DE_i xDE_n + SignUp_i xSignUp_n}}{\sqrt{AgeSCaled_i + M_i + F_i + EG_i + DE_i + SignUP_i} X \sqrt{AgeSCaled_n + M_n + F_n + EG_n + DE_n + SignU_n}}$, where n is the number of the new user that we need to get its ratings, and i is a user in the table given in the question.

Using the above equation,

User (i)	1	2	3	4	5
Similarity (w_i)	0.9470313	0.0377504	0.5761895	0.94005	0.611904

We know that Similarity is the 1 - distance, so the higher similarity is the highest three, so we will use 1, 4, 5 for evaluating the ratings (I chose 3, as the question instructed to use 3NN).

The ratings of the new user are

Using the equation $r_n = \frac{\sum_{i=1}^{n-1} w_i r_i}{\sum_{i=1}^{n-1} w_i}$

Movie Name	**Lion King**	**Hero**	**StarTrek**	**Wall-E**	**Inception**
Rating	**3.21978**	**3.38329**	**4.4770108**	**3.99042**	**4.42542**

b. The answer is simply **yes**. **We** will do as we have done above but with ratings.
Vector of new user **[3.7 , 3.9, 4.1, X, 3.8], |V| = 7.75564**

The equation used to calculate the similarity will be

$w_i = \frac{LionKing_i xLionKing_n + Hero_i xHero_n + StarTrek_i xStarTrek_n + Inception_i inception_n}{\sqrt{LionKing_i + Hero_i + StarTrek_i + Inception_i} X \sqrt{LionKing_n + Hero_n + StarTrek_n + Inception_n}}$, where n is the number of the new user that we need to get its ratings, and i is a user in the table given in the question.

User (i)	1	2	3	4	5
Similarity (w_i)	0.98601	0.99631	0.98501	0.986	0.9988

Get the highest three (5,2,1)

Then the rating of Well-E = $\frac{0.9988x3.5 + 0.99631x4.1 + 0.98601x4}{0.9988+0.99631+0.98601} = 3.86589$

c. Given that Alice vector = [27, Female, DE, May 2017] .Here is there a new addition, which says that Rating equal to or above 3.7 indicates True, and below that it is a False.

We need to change the Table to be as follows.

Age(Scaled)	Male	Female	EG	DE	SignUP(Starting with March 2016)	Lion King	Hero	StarTrek	Well-E	Inception
$\frac{2}{3}$	1	0	1	0	1	0	0	1	1	1
$\frac{1}{6}$	0	1	0	1	0	1	1	0	1	1

$\frac{1}{3}$	0	1	1	0	$\frac{16}{23}$	1	1	0	1	1
0	1	0	1	0	$\frac{5}{23}$	0	0	1	1	1
1	1	0	0	1	$\frac{21}{23}$	1	1	1	0	1

So the vector of Alice has becomeV = **[**$\frac{7}{12}$**, 0, 1, 0, 1,** $\frac{14}{23}$**,** 0**, ?, 1, 1, 1]** to be compatible with Table.

Let's try get the magnitude of |V| = 2.3897255.

Now, let's do the similarity with all users to get the three closest ones and check.

Using the following equation:

$$w_i = sim(u_i, u_n) = \frac{\sum_{i=1}^{n} A_i x B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$, where n is the number of features.

User (i)	1	2	3	4	5
Similarity (w_i)	0.65859	0.76532	0.64004	0.583439	0.662525

Get the closest three points: **2, 5, l respectively. If we** take the majority of the three points, we can find that ALICE likes **HERO. SO YES.**

The reason for this:

$$\frac{0.658\ X\ 0 + 0.764x1 + 0.663x1}{0.658+0.764+0.663} = 0.684\ closer\ to\ 1\ so\ it\ is\ yes.$$