Assignment 2 - Melissa LaHoud

Part 1: Theory

1. What is the role of a *confusion matrix* in the evaluation of a machine trained for a pattern recognition task? In your answer, refer to a concrete example, either from literature or one you created. Anchor you answer in relevant literature.

Confusion matrix has a role of being a measure used while solving classification problems. Pattern recognition or object detection is a little different than numeric. According to Koech, K. (2020), "But calculating of confusion matrix for object detection and instance segmentation tasks is less intuitive. First, it is necessary to understand another supporting metric: Intersection over Union (IoU). A key role in calculating metrics for object detection and instance segmentation tasks is played by Intersection over Union (IoU)." IoU is calculated as the area of overlap/intersection between gt and pd divided by the area of the union between the two, that is,

$$\text{IoU} = \frac{\text{area}(gt \cap pd)}{\text{area}(gt \cup pd)}$$

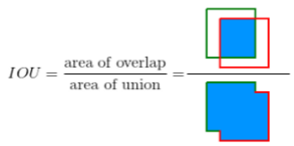



Fig 1 (Source: Author)

"A confusion matrix is made up of 4 components, namely, True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN). To define all the components, we need to define some threshold (say α) based on IoU." (Koech, K., 2020)

- True Positive (TP) — This is an instance in which the classifier predicted positive when the truth is indeed positive, that is, a detection for which IoU ≥ α.
- False Positive (FP) — This is a wrong positive detection, that is, a detection for which IoU < α.
- False Negative (FN) — This is an actual instance that is not detected by the classifier.
- True Negative (TN) — This metric implies a negative detection given that the actual instance is also negative. In object detection, this metric does not apply because there exist many possible predictions that should not be detected in an image. Thus, TN includes all possible wrong detection that were not detected.

These will also help you find Precision, sensitivity, specificity, error rate, and accuracy. Below are the equations to each listed along with a confusion matrix itself to see where each equation is pulling from.

| | | Predicted Category | | |
|---|---|---|---|---|
| | | 0 | 1 | **Total** |
| Actual category | **0** | *TN* | *FP* | **TAN** |
| | **1** | *FN* | *TP* | **TAP** |
| | **Total** | **TPN** | **TPP** | **GT** |

$$Precision = \frac{TP}{TPP}$$

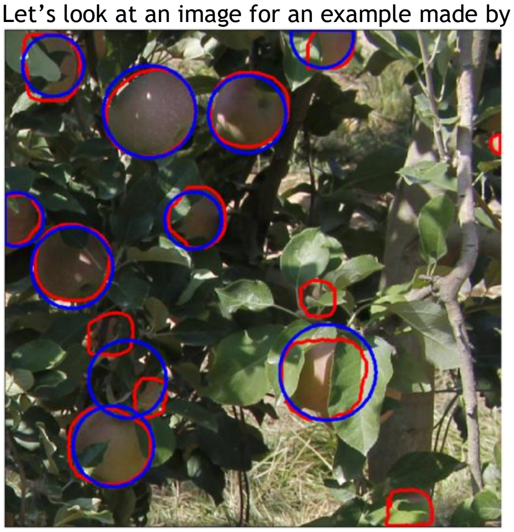
$$Sensitivity = \frac{Number\ of\ true\ positives}{Total\ actually\ positive} = \frac{TP}{TAP} = \frac{TP}{TP+FN}$$

$$Specificity = \frac{Number\ of\ true\ negatives}{Total\ actually\ negative} = \frac{TN}{TAN} = \frac{TN}{FP+TN}$$

$$Error\ Rate = 1 - Accuracy = \frac{FN+FP}{TN+FN+FP+TP} = \frac{FN+FP}{GT}$$

$$Accuracy = \frac{TN+TP}{TN+FN+FP+TP} = \frac{TN+TP}{GT}$$

Let’s look at an image for an example made by Koech, K (2020) for image recoginition:



Below are the parameters:

**Parameters:**

- **ground** — is $n \times m \times 2$ array where $n$ is number of the ground truth instances for the given image, $m$ is the number of $(x,y)$ pairs sampled on the circumference of the mask.
- **pred** is $p \times q \times 2$ array where $p$ is the number of detections, and $q$ is the number of $(x,y)$ points sampled for the prediction mask
- **iou_value** is the IoU threshold

With lots of code in python, he concluded:

For Fig 5 and IoU threshold, α = 0.5, `evaluation(ground,pred,iou_value)` →

```
TP: 9    FP: 5    FN: 0    GT: 10
Precall: 0.643    Recall: 1.0    F1 score: 0.783
```

Another example I did in a previous class which is using the Loans dataset where I used a confusion matrix to come up with each of model evaluation measures.

In [9]:
```
y = training[['Approval']]

x = pd.concat((training[['Debt-to-Income Ratio']], training[['FICO Score']], training[['Request Amount']]), axis=1)

x_names = ["Debt to Income Ratio", "FICO","Request Amt"]

y_names = ["T","F"]

C50_01 = DecisionTreeClassifier(criterion = "entropy", max_leaf_nodes=5).fit(x,y)

training['C1'] = C50_01.predict(x)

C51 = pd.crosstab(training['Approval'], training['C1'])
C51
```

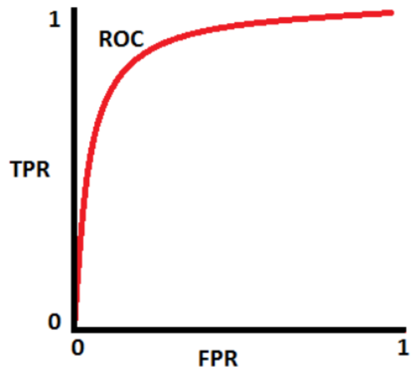
Out[9]:

| C1<br>Approval | F | T |
|---|---|---|
| F | 53589 | 21477 |
| T | 2237 | 72999 |

| Evaluation Measure | Model 1 Formula | Model 1 Value | Model 2 Formula | Model 2 Value |
|---|---|---|---|---|
| Accuracy | 53589+72999/150302 | .842 | 62443+60789/150302 | .820 |
| Error Rate | 1-.842 | .158 | 1-.820 | .180 |
| Sensitivity | 72999/75236 | .970 | 60789/75236 | .808 |
| Specificity/Recall | 53589/75066 | .714 | 62443/75066 | .832 |
| Precision | 72999/94476 | .773 | 60789/73412 | .828 |

2. What is the role of the ROC curve? How would you use it to compare the performance of several classifiers? In your answer, refer to concrete examples of classifiers, either from literature or one you created. Illustrate the ROC curves and anchor your answer in relevant literature.

The role of a ROC curve is to show the performance measurement for the classification problems at various threshold settings, ROC is a probability curve. According to Narkhede, S. (2021), "It is one of the most important evaluation metrics for checking any classification model's performance because we need to check or visualize the performance of the multi-class classification problem." We can plot the ROC curve by knowing the True Positive Rate/Recall/Sensitivity and the False Positive Rate which is the complement of the specificity. "An ROC graph, hence, shows relative trade-offs between advantages (true positives) and costs (false positives)."




Where TPR and FPR can be calculated from the confusion matrix explained above. Here are the equations;

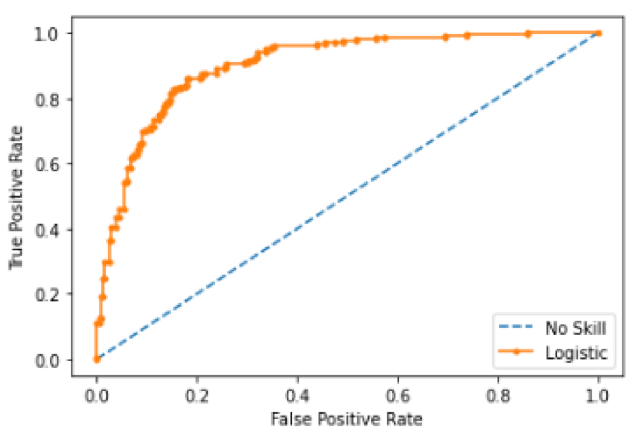
$$\text{Sensitivity} = tp \text{ rate} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

$$1 - \text{specificity} = fp \text{ rate} = \frac{\text{FP}}{\text{FP} + \text{TN}}$$

There was a great example done in python that I tried:

```python
# roc curve and auc
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from matplotlib import pyplot

# generate 2 class dataset
X, y = make_classification(n_samples=1000, n_classes=2, random_state=1)

# split into train/test sets
trainX, testX, trainy, testy = train_test_split(X, y, test_size=0.5, random_state=2)

# generate a no skill prediction (majority class)
ns_probs = [0 for _ in range(len(testy))]

# fit a model
model = LogisticRegression(solver='lbfgs')
model.fit(trainX, trainy)

# predict probabilities
lr_probs = model.predict_proba(testX)

# keep probabilities for the positive outcome only
lr_probs = lr_probs[:, 1]

# calculate scores
ns_auc = roc_auc_score(testy, ns_probs)
lr_auc = roc_auc_score(testy, lr_probs)

# summarize scores
print('No Skill: ROC AUC=%.3f' % (ns_auc))
print('Logistic: ROC AUC=%.3f' % (lr_auc))

# calculate roc curves
ns_fpr, ns_tpr, _ = roc_curve(testy, ns_probs)
lr_fpr, lr_tpr, _ = roc_curve(testy, lr_probs)

# plot the roc curve for the model
pyplot.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
pyplot.plot(lr_fpr, lr_tpr, marker='.', label='Logistic')

# axis labels
pyplot.xlabel('False Positive Rate')
pyplot.ylabel('True Positive Rate')

# show the legend
pyplot.legend()

# show the plot
pyplot.show()
```

```
No Skill: ROC AUC=0.500
Logistic: ROC AUC=0.903
```




(Brownlee, J., 2021)

Part 2: Application

You are tasked to build an image classifier for the MNIST dataset of handwritten numbers, implementing the *k-nearest neighbors (k-NN)* algorithm. You will need the following:

- The MNIST dataset, available on multiple servers on the Internet. For example:
  - http://yann.lecun.com/exdb/mnist/
  - http://www.pymvpa.org/datadb/mnist.html
- The Python package ***neighbors.KNeighborsClassifier:*** https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

The input to your classifier program is an image containing a digit, 0-9. Your program must correctly identify the digit with an accuracy of 95%. Here the outline of your task, but you will have to do a bit of research on your own (and increasingly so throughout the program) to fill in the details:

Familiarize yourself with the MNIST dataset

Familiarize yourself with the k-NN algorithm and its Python implementation in sklearn

Create a Jupyter notebook for this assignment and implement the k-NN algorithm:

- Import the package kNeighborsClassifier.

[2]:
```
import os
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
import numpy as np
from sklearn.datasets import fetch_openml, load_digits
import matplotlib.pyplot as plt
import matplotlib
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_recall_fscore_support
from sklearn.metrics import accuracy_score
from sklearn.metrics.pairwise import euclidean_distances
```

- Be mindful of the train-test split and set the parameters accordingly (justify your choice).

0]:
```
os.chdir("C:\\Users\\melis\\Documents\\DSC-540 Machine Learning")
data = pd.read_csv('mnist_test.csv')
data.head()
```

ut[60]:

| | label | 1x1 | 1x2 | 1x3 | 1x4 | 1x5 | 1x6 | 1x7 | 1x8 | 1x9 | ... | 28x19 | 28x20 | 28x21 | 28x22 | 28x23 | 28x24 | 28x25 | 28x26 | 28x27 | 28x28 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 7 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **1** | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **2** | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **3** | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **4** | 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

5 rows × 785 columns

1]:
```
train_img = np.array(data)
train_target = np.array(data)

X = train_img
y = train_target

X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.5)
```

2]:
```
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)
```

```
(5000, 785) (5000, 785)
(5000, 785) (5000, 785)
```

[6]: 

```
pd.crosstab(index=train['label'], columns='count')
```

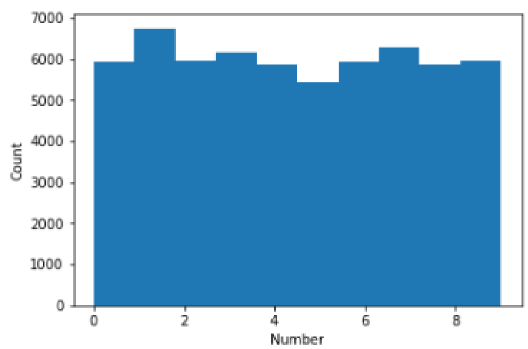
Out[6]:

| col_0 | count |
|---|---|
| **label** | |
| 0 | 5923 |
| 1 | 6742 |
| 2 | 5958 |
| 3 | 6131 |
| 4 | 5842 |
| 5 | 5421 |
| 6 | 5918 |
| 7 | 6265 |
| 8 | 5851 |
| 9 | 5949 |

[7]:

```
plt.hist(train["label"])
plt.xlabel("Number")
plt.ylabel("Count")
```

Out[7]: Text(0, 0.5, 'Count')




I did the first round at half because to find the Euclidean distance between an element in the test set and the training set there need to be an equal number of elements to find them all. Later I split them differently.

- Identify the variables in the dataset and define the Euclidean distance between an element in the test set and the training set.

Variables: "label" is the number that was hand written and the rest of the variables are the pixels.

53]:

```
euclidean_distances(X_train,y_train)
```

)ut[63]:

```
array([[   0.        , 2538.96947599, 2881.96165832, ..., 2660.42252283,
        2644.38045674, 2940.55947058],
       [2538.96947599,    0.        , 3035.68789568, ..., 2278.61975766,
        2709.85977497, 2406.13673759],
       [2881.96165832, 3035.68789568,    0.        , ..., 2797.15408943,
        2404.33046813, 2867.76446034],
       ...,
       [2660.42252283, 2278.61975766, 2797.15408943, ...,    0.        ,
        2365.24544181, 2024.58687144],
       [2644.38045674, 2709.85977497, 2404.33046813, ..., 2365.24544181,
           0.        , 2372.7267015 ],
       [2940.55947058, 2406.13673759, 2867.76446034, ..., 2024.58687144,
        2372.7267015 ,    0.        ]])
```

- <u>Calculate the distance between the test element and each of if its k nearest neighbors.</u>
- <u>Count the occurrence of each digit within the k nearest neighbors and identify the most popular digit.</u>

- Identify the test element as the digit voted as most popular in the set of the k nearest neighbors.
- Classify the test element accordingly (i.e. based on the popular vote).
- Calculate the error.

In [64]:

```
class KNN:
    def __init__(self, K=3):
        self.K = K
```

In [65]:

```
class KNN:
    def __init__(self, K=3):
        self.K = K
    def fit(self, x_train, y_train):
        self.X_train = x_train
        self.Y_train = y_train
```

In [66]:

```
def predict(self, X_test):
    predictions = []
    for i in range(len(X_test)):
        dist = np.array([euc_dist(X_test[i], x_t) for x_t in
        self.X_train])
        dist_sorted = dist.argsort()[:self.K]
        neigh_count = {}
        for idx in dist_sorted:
            if self.Y_train[idx] in neigh_count:
                neigh_count[self.Y_train[idx]] += 1
            else:
                neigh_count[self.Y_train[idx]] = 1
        sorted_neigh_count = sorted(neigh_count.items(),
        key=operator.itemgetter(1), reverse=True)
        predictions.append(sorted_neigh_count[0][0])
    return predictions
```

In [67]:

```
from sklearn.datasets import load_digits
mnist = load_digits()
print(mnist.data.shape)
```

```
(1797, 64)
```

68]:
```
X = mnist.data
y = mnist.target
```

69]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=123)
```

71]:
```
kVals = np.arange(3,60,2)
accuracies = []
for k in kVals:
  model = KNeighborsClassifier(n_neighbors = k)
  model.fit(X_train, y_train)
  pred = model.predict(X_test)
  acc = accuracy_score(y_test, pred)
  accuracies.append(acc)
  print("K = "+str(k)+"; Accuracy: "+str(acc))
```

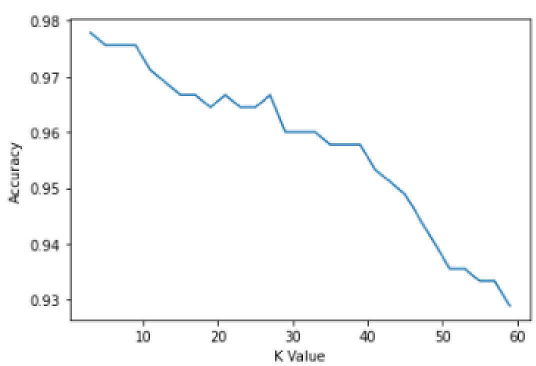
```
K = 3; Accuracy: 0.9777777777777777
K = 5; Accuracy: 0.9755555555555555
K = 7; Accuracy: 0.9755555555555555
K = 9; Accuracy: 0.9755555555555555
K = 11; Accuracy: 0.9711111111111111
K = 13; Accuracy: 0.9688888888888889
K = 15; Accuracy: 0.9666666666666667
K = 17; Accuracy: 0.9666666666666667
K = 19; Accuracy: 0.9644444444444444
K = 21; Accuracy: 0.9666666666666667
K = 23; Accuracy: 0.9644444444444444
K = 25; Accuracy: 0.9644444444444444
K = 27; Accuracy: 0.9666666666666667
K = 29; Accuracy: 0.96
K = 31; Accuracy: 0.96
K = 33; Accuracy: 0.96
K = 35; Accuracy: 0.9577777777777777
K = 37; Accuracy: 0.9577777777777777
K = 39; Accuracy: 0.9577777777777777
K = 41; Accuracy: 0.9533333333333334
K = 43; Accuracy: 0.9511111111111111
K = 45; Accuracy: 0.9488888888888889
K = 47; Accuracy: 0.9444444444444444
K = 49; Accuracy: 0.94
K = 51; Accuracy: 0.9355555555555556
K = 53; Accuracy: 0.9355555555555556
K = 55; Accuracy: 0.9333333333333333
K = 57; Accuracy: 0.9333333333333333
K = 59; Accuracy: 0.9288888888888889
```

72]:
```
max_index = accuracies.index(max(accuracies))
print(max_index)
```

```
0
```

73]:
```
plt.plot(kVals, accuracies)
plt.xlabel("K Value")
plt.ylabel("Accuracy")
```

Out[73]:
```
Text(0, 0.5, 'Accuracy')
```




74]:
```
Classifer = KNeighborsClassifier(n_neighbors=3)
Classifer.fit(X_train,y_train)
```

Out[74]:
```
KNeighborsClassifier(n_neighbors=3)
```

75]:
```
pred = Classifer.predict(X_train)
```

16]:
```
classifier3 = KNeighborsClassifier(n_neighbors=3)
classifier3.fit(X3_train, Y3_train)
```

```
<ipython-input-16-a9197e567ed8>:2: DataConversionWarning: A column-vector y was
change the shape of y to (n_samples, ), for example using ravel().
  classifier3.fit(X3_train, Y3_train)
```

Out[16]:
```
KNeighborsClassifier(n_neighbors=3)
```

17]:
```
Pred3 = classifier3.predict(X3_test)
```

18]:
```
from sklearn.metrics import classification_report, confusion_matrix

print(confusion_matrix(Y3_test, Pred3))
print(classification_report(Y3_test, Pred3))
```

```
[[ 974    1    1    0    0    1    2    1    0    0]
 [    0 1133    2    0    0    0    0    0    0    0]
 [   10    9  996    2    0    0    0   13    2    0]
 [    0    2    4  976    1   13    1    7    3    3]
 [    1    6    0    0  950    0    4    2    0   19]
 [    6    1    0   11    2  859    5    1    3    4]
 [    5    3    0    0    3    3  944    0    0    0]
 [    0   21    5    0    1    0    0  991    0   10]
 [    8    2    4   16    8   11    3    4  914    4]
 [    4    5    2    8    9    2    1    8    2  968]]
              precision    recall  f1-score   support

           0       0.97      0.99      0.98       980
           1       0.96      1.00      0.98      1135
           2       0.98      0.97      0.97      1032
           3       0.96      0.97      0.96      1010
           4       0.98      0.97      0.97       982
           5       0.97      0.96      0.96       892
           6       0.98      0.99      0.98       958
           7       0.96      0.96      0.96      1028
           8       0.99      0.94      0.96       974
           9       0.96      0.96      0.96      1009

    accuracy                           0.97     10000
   macro avg       0.97      0.97      0.97     10000
weighted avg       0.97      0.97      0.97     10000
```

(Penumudy, T., 2021)

[8]:

```
pred_prob2 = classifier3.predict_proba(X3_test)
```

[9]:

```
from sklearn.metrics import roc_curve
```

10]:

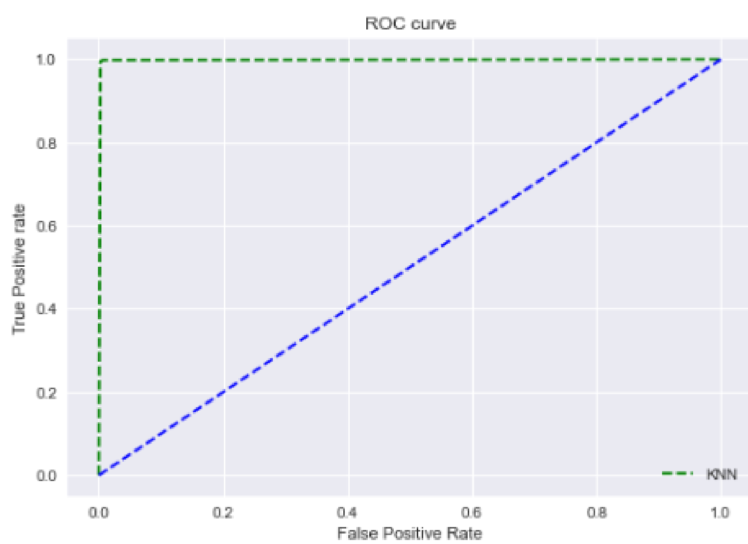
```
fpr2, tpr2, thresh2 = roc_curve(Y3_test, pred_prob2[:,1], pos_label=1)
```

13]:

```
random_probs = [0 for i in range(len(Y3_test))]
p_fpr, p_tpr, _ = roc_curve(Y3_test, random_probs, pos_label=1)
```

16]:

```
plt.style.use('seaborn')

plt.plot(fpr2, tpr2, linestyle='--',color='green', label='KNN')
plt.plot(p_fpr, p_tpr, linestyle='--', color='blue')
# title
plt.title('ROC curve')
# x label
plt.xlabel('False Positive Rate')
# y label
plt.ylabel('True Positive rate')

plt.legend(loc='best')
plt.savefig('ROC',dpi=300)
plt.show();
```




(Aniruddha B., 2020)

Review the article “Handwritten Digit Recognition Using K-Nearest Neighbour Classifier.” Note the algorithms used, but focus on the way the authors:

- Present the findings
- Discuss the findings
- Calculate the accuracy of the results
- Write the article, using professional terminology and content organization

Write a technical report (i.e., not a full-fledged academic paper) to accompany the Jupyter notebook that implements the classifier, using the aforementioned article as a guide on what to address and how to present the mini-project and report the findings.

Address the potential role of a confusion matrix in your report (refer to Part 1).

Address the potential role of ROC curve in your report (refer to Part 1).

I used the MNIST dataset which includes a training set of 60,000 images and a test set of 10,000 Images. "The proposed method uses k-nearest neighbor (knn) classification algorithm for classifying the MNIST digit images in test database using the feature vector of training database. The k-nearest neighbor algorithm (k-NN) is classification technique to classify the objects base on training features space. The functionality of k-NN algorithm is to define the computations until classification is done irrespective of the learning techniques."(Babu, U, etc., 2014) I used Euclidean distance measures to compute the distance between the values of the test sample and the training image. The majority among the k-nearest training samples was also based on Euclidean distance. I calculated the k by seeing which K had the highest accuracy. Below is a graph representation of are my results for the k values:

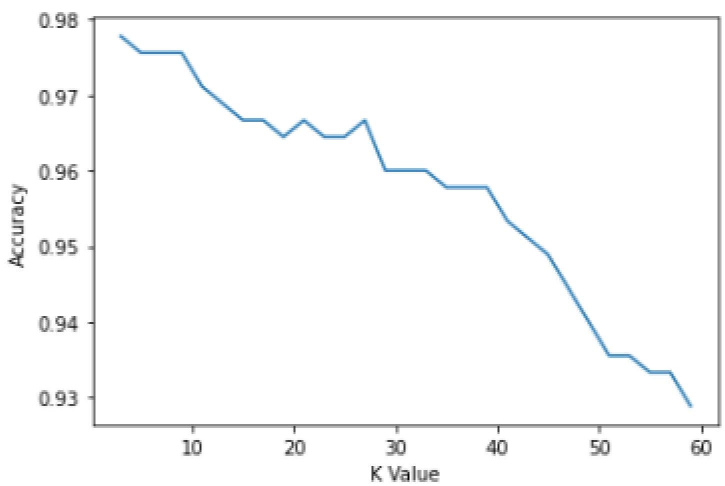



Figure 1. Effect of different testing samples on accuracy by taking different values of k

I executed the algorithm with the value of k is 3. This algorithm produced a high percentage of recognition for the algorithm at 97% as we can see in the table 3. We can also see in table 3 that the precision and recall values are also very large meaning the k-NN algorithm did a good job predicting the handwritten images.

```
[[ 974    1    1    0    0    1    2    1    0    0]
 [    0 1133    2    0    0    0    0    0    0    0]
 [   10    9  996    2    0    0    0   13    2    0]
 [    0    2    4  976    1   13    1    7    3    3]
 [    1    6    0    0  950    0    4    2    0   19]
 [    6    1    0   11    2  859    5    1    3    4]
 [    5    3    0    0    3    3  944    0    0    0]
 [    0   21    5    0    1    0    0  991    0   10]
 [    8    2    4   16    8   11    3    4  914    4]
 [    4    5    2    8    9    2    1    8    2  968]]
```

Table 1. Values of correct recognition for 10,000 test set

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.97 | 0.99 | 0.98 | 980 |
| 1 | 0.96 | 1.00 | 0.98 | 1135 |
| 2 | 0.98 | 0.97 | 0.97 | 1032 |
| 3 | 0.96 | 0.97 | 0.96 | 1010 |
| 4 | 0.98 | 0.97 | 0.97 | 982 |
| 5 | 0.97 | 0.96 | 0.96 | 892 |
| 6 | 0.98 | 0.99 | 0.98 | 958 |
| 7 | 0.96 | 0.96 | 0.96 | 1028 |
| 8 | 0.99 | 0.94 | 0.96 | 974 |
| 9 | 0.96 | 0.96 | 0.96 | 1009 |
| accuracy | | | 0.97 | 10000 |
| macro avg | 0.97 | 0.97 | 0.97 | 10000 |
| weighted avg | 0.97 | 0.97 | 0.97 | 10000 |

Table 2. Conclusion of algorithm executed with k = 3

ROC curves are a nice way to see how any predictive model can distinguish between the true positives and negatives. The ROC curve does this by plotting sensitivity, the probability of predicting a real positive will be a positive, against 1-specificity, the probability of predicting a real negative will be a positive. The further the curve is from the diagonal line, the better the model is at discriminating between positives and negatives in general. When we look at Figure 2, we can see the model is almost always going to be good at discriminating between positives and negative.

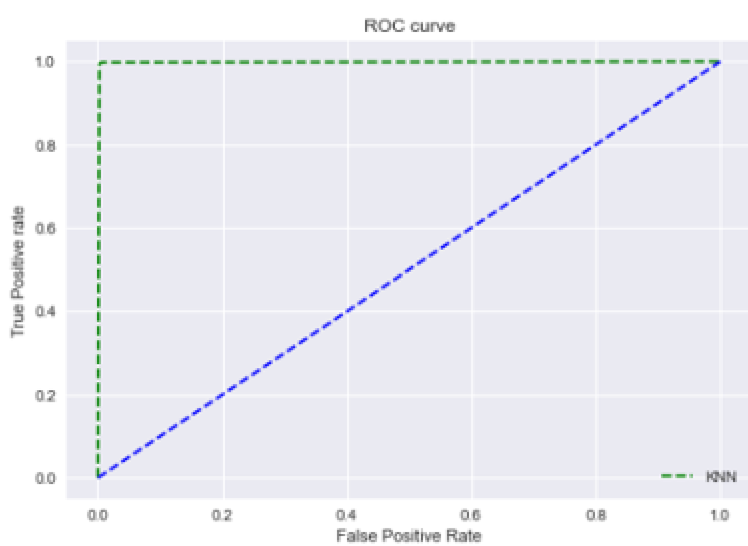



Figure 2. ROC curve

References:

Aniruddha B. (2020, July 20). AUC-ROC Curve in Machine Learning Clearly Explained. Retrieved from https://www.analyticsvidhya.com/blog/2020/06/auc-roc-curve-machine-learning/

Babu, U., Venkateswarlu, Y, & Chintha, A. (2014) Handwritten Digit Recognition Using K-Nearest Neighbour Classifier. Retrieved from https://ieeexplore-ieee-org.lopes.idm.oclc.org/stamp/stamp.jsp?tp=&arnumber=6755106

Brownlee, J. (2021, January 12). How to Use ROC Curves and Precision-Recall Curves for Classification in Python. Retrieved from https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/

Koech, K. E. (2020, July 31). Confusion Matrix and Object Detection. Retrieved from https://towardsdatascience.com/confusion-matrix-and-object-detection-f0cbcb634157

Narkhede, S. (2021, January 14). Understanding AUC - ROC Curve. Retrieved from https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5

Penumudy, T. (2021, January 29). A Beginner's Guide to KNN and MNIST Handwritten Digits Recognition using KNN from Scratch. Retrieved from https://medium.com/analytics-vidhya/a-beginners-guide-to-knn-and-mnist-handwritten-digits-recognition-using-knn-from-scratch-df6fb982748a