Παραδοτέο ενδιάμεσης εργασίας Νευρωνικά

Χρυσικός Χρήστος AEM:9432 THMMY

Παρακάτω είναι τα απαραίτητα modules που χρειάζονται για να τρέξουν οι αλγόριθμοι. Χρησιμοποιμε την βιβλιοθήκη sklearn για τον KNeighborsClassifier και NearestCentroid

In [ ]:
```
from sklearn import metrics
import matplotlib.pyplot as plt
from keras.datasets import mnist
from sklearn.neighbors import KNeighborsClassifier, NearestCentroid
from time import time
```

Τα δεδομένα τα φορτώνουμε απο την Database MNIST χρησιμοποιώντας το module keras. Όμως επειδή η fit δέχεται είσοδο 2D Array και τα μοντέλα είναι σε μορφή (SAMPLES_NUM, 28, 28) τα μετασχηματίζουμε σε (SAMPLES_NUM, 784)

Data Transform

In [ ]:
```
(train_X, train_y), (test_X, test_y) = mnist.load_data()

train_X = train_X.reshape(train_X.shape[0], train_X.shape[1] ** 2)
test_X = test_X.reshape(test_X.shape[0], test_X.shape[1] ** 2)
```

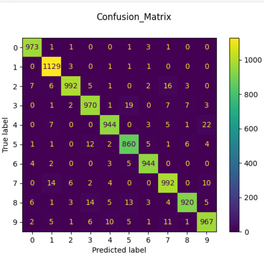
KNN Classifier

Με μια συνάρτηση επανάληψης τρέχουμε 2 φορες τον classifier για K = 1 και K = 3

In [ ]:
```
neighbors = [1, 3]
for k in neighbors:
    #Fitting the KNearest Neighbor Classifier
    knn = KNeighborsClassifier(k)
    # Starting timer
    t = time()
    # Fitting the training data to the model
    knn.fit(train_X, train_y)
    # Creating first time stamp for time elapsed to fit the data
    time_fit = time() - t
    t = time()
    # Using testing data to evaluate the model
    predicted = knn.predict(test_X)
    # Creating second time stamp for time elapsed to predict
    time_predict = time() - t
    print(f"Analysis for K Nearest Neighbor = {k}")
    # Displaying of the time it took to fit, predict the data and the total runtime of the process.
    print(f"Time to fit: {time_fit:.2f}s - Time to predict: {time_predict:.2f}s\nTotal time: "
          f"{(time_fit + time_predict):.2f}s")
    # Displaying the scores for each classification of the model.
    print(
        f"Classification report for classifier {knn}:\n"
        f"{metrics.classification_report(test_y, predicted)}\n"
    )
    # Using Confusion Matrix to display the Model's performance
    display = metrics.ConfusionMatrixDisplay.from_predictions(test_y, predicted)
    display.figure_.suptitle("Confusion_Matrix")
    plt.show()
```

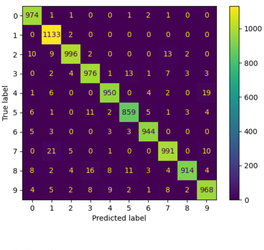
KNN (K = 1) Classifier Scores

In [ ]:
```
Analysis for K Nearest Neighbor = 1
Time to fit: 0.01s - Time to predict: 17.34s
Total time: 17.35s
Classification report for classifier KNeighborsClassifier(n_neighbors=1):
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       980
           1       0.97      0.99      0.98      1135
           2       0.98      0.96      0.97      1032
           3       0.96      0.96      0.96      1010
           4       0.97      0.96      0.97       982
           5       0.95      0.96      0.96       892
           6       0.98      0.99      0.98       958
           7       0.96      0.96      0.96      1028
           8       0.98      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
```



KNN (K = 3) Classifier Scores

In [ ]:
```
Analysis for K Nearest Neighbor = 3
Time to fit: 0.00s - Time to predict: 18.88s
Total time: 18.89s
Classification report for classifier KNeighborsClassifier(n_neighbors=3):
              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
```

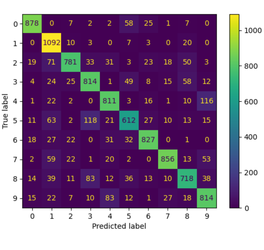


Nearest Centroid

Αλλάζουμε τον Classifier και ακολουθούμε ακριβώς την ίδια διαδικασία

In [ ]:
```
#Fitting the Nearest Centroid Classifier
NCentroid = NearestCentroid()
# Starting timer
t = time()
NCentroid.fit(train_X, train_y)
# Creating first time stamp for time elapsed to fit the data
time_fit = time() - t
t = time()
# Using testing data to evaluate the model
predicted = NCentroid.predict(test_X)
# Creating second time stamp for time elapsed to predict
predict_time = time() - t

print(f"Analysis for Nearest Centroid")
# Displaying of the time it took to fit, predict the data and the total runtime of the process.
print(f"Time to fit: {time_fit:.2f}s - Time to predict: {time_predict:.2f}s\nTotal time: "
      f"{(time_fit + time_predict):.2f}s")

# Displaying the scores for each classification of the model.
print(
    f"Classification report for classifier {NCentroid}:\n"
    f"{metrics.classification_report(test_y, predicted)}\n"
)
# Using Confusion Matrix to display the Model's performance
display = metrics.ConfusionMatrixDisplay.from_predictions(test_y, predicted)
display.figure_.suptitle("Confusion Matrix")
plt.show()
```

In [ ]:
```
Analysis for Nearest Centroid
Time to fit: 0.05s - Time to predict: 18.88s
Total time: 18.93s
Classification report for classifier NearestCentroid():
              precision    recall  f1-score   support

           0       0.91      0.90      0.90       980
           1       0.77      0.96      0.86      1135
           2       0.88      0.76      0.81      1032
           3       0.77      0.81      0.78      1010
           4       0.80      0.83      0.81       982
           5       0.75      0.69      0.72       892
           6       0.88      0.86      0.87       958
           7       0.91      0.83      0.87      1028
           8       0.79      0.74      0.76       974
           9       0.77      0.81      0.79      1009

    accuracy                           0.82     10000
   macro avg       0.82      0.82      0.82     10000
weighted avg       0.82      0.82      0.82     10000
```



Conclusion

Είναι πλέον προφάνες οτι για το MNIST DataSet ο KNN classifier είναι ιδανικότερος αφού πετυχαίνουμε μεγαλύτερη ακρίβεια και K = 1 και K = 3. Πιο συγκεκριμένα για K = 1 πετυχαίνουμε την ίδια ακρίβεια και σε χαμηλότερο χρόνο.

In [ ]:
```
```

In [ ]:
```
```

In [ ]:
```
```

In [ ]:
```
```

In [ ]:
```
```

In [ ]:
```
```

In [ ]:
```
```

In [ ]:
```
```