2η Εργασία Νευρωνικών Δικτύων

Περιεχόμενα

1. Εισαγωγή

Η εργασία που εκπονήθηκε, πραγματεύεται την υλοποίηση ενός **Support Vector Machine** που θα εκπαιδευτεί για να επιλύει ένα από τα παρακάτω το οποίο εκπαιδεύεται πάνω στην βάση δεδομένων MNIST και σε δεύτερη φάση στην CIFAR-10 και αποσκοπεί στην εύρεση κλάσης παραδειγμάτων που θα το τροφοδοτούμε.

Ο κώδικας υλοποιήθηκε σε γλώσσα προγραμματισμού python σε περιβάλλον VScode και εμπεριέχει συναρτήσεις από το API **keras** τις βιβλιοθήκες **numpy**, **sklearn**, **matplotlib**, **sys**.

2.Κώδικας

```
from keras.datasets import mnist
from keras.datasets import cifar10

from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.utils import resample

from matplotlib import pyplot as plt
```

```
import numpy as np

# mnist dataset
# Loading the Data
(x_train,y_train), (x_test,y_test) = mnist.load_data()
# Shaping the Data
x_train = x_train.reshape(x_train.shape[0], x_train.shape[1]**2)
x_test = x_test.reshape(x_test.shape[0], x_test.shape[1]**2)

x_train, x_test = np.array(x_train, np.float32), np.array(x_test, np.float32)
x_train, x_test = x_train/255., x_test/255.


model = SVC()

model.fit(x_train,y_train)

pred = model.predict(x_test)

a_1 = accuracy_score(y_test, pred)

print(a_1)
#Cifar-10 dataset


#Loading Data
(x_train,y_train), (x_test,y_test) = cifar10.load_data()
n_images,n_rows,n_colimns,n_channels = x_train.shape

x_train = x_train.reshape((x_train.shape[0], -1))
x_test = x_test.reshape((x_test.shape[0], -1))
n_training = 5000
n_testing = 1000

x_train = x_train[0:n_training,:]
x_test = x_test[0:n_testing,:]
y_train = y_train[0:n_training]
y_test = y_test[0:n_testing]
#Shaping Data
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)

# Hyperparameters Arrays
```

```
accuracy_results=[]
kernels = ["rbf", "poly", "linear","sigmoid"]
c_list = [0.01,0.1,0.5,0.8,1]

for k in kernels:
    temp = []
    for temp_c in c_list:
        model2 = SVC(kernel = k, C = temp_c)
        model2.fit(x_train,y_train.ravel())
        pred_2 = model2.predict(x_test)
        a_2 = accuracy_score(y_test, pred_2)
        temp.append(a_2)
        print("Accucacy Score with kernel:", k ,"and C:", temp_c , "----is:",
a_2)
    accuracy_results.append(temp)

plt.plot(c_list, accuracy_results[0],'o')
plt.title="rbf"
plt.xlabel('Hyperparameter "C"')
plt.ylabel('Accuracy Score')
plt.show()
plt.plot(c_list, accuracy_results[1],'o')
plt.title="poly"
plt.xlabel('Hyperparameter "C"')
plt.ylabel('Accuracy Score')
plt.show()
plt.plot(c_list, accuracy_results[2],'o')
plt.title="linear"
plt.xlabel('Hyperparameter "C"')
plt.ylabel('Accuracy Score')
plt.show()
plt.plot(c_list, accuracy_results[3],'o')
plt.title="sigmoid"
plt.xlabel('Hyperparameter "C"')
plt.ylabel('Accuracy Score')
plt.show()
```

3. Ανάλυση Κώδικα

```
(x_train,y_train), (x_test,y_test) = datasets.mnist.load_data()
x_train = x_train.reshape(x_train.shape[0], x_train.shape[1]**2)
x_test = x_test.reshape(x_test.shape[0], x_test.shape[1]**2)
```

Σε αυτό το σημείο του κώδικα κατεβάζουμε το dataset **MNIST**, και εν συνεχεία αλλάζουμε τις διαστάσεις των πινάκων των samples έτσι ώστε να είναι πίνακες από vectors (που είναι τα samples) για να μπορούμε να τα προωθήσουμε στις επόμενες συναρτήσεις μας.

```
x_train, x_test = np.array(x_train, np.float32), np.array(x_test, np.float32)
x_train, x_test = x_train/255., x_test/255.
```

Είναι σημαντικό να κανονικοποιήσουμε τα δεδομένα μας στο πεδίο [0,1] έτσι ώστε να μην έχουμε λανθασμένους υπολογισμούς στο SVC() από τυχόντα outliers. Επειδή ξέρουμε ότι οι gray-scale εικόνες που έχουμε παίρνουν τιμές σε κάθε pixel από 0 έως 255 τότε διαιρούμε όλα τα pixel με των αριθμό 255 κάτι που βγαίνει από τον τύπο:

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$$

```
model = SVC()

model.fit(x_train,y_train)

pred = model.predict(x_test)

a_1 = accuracy_score(y_test, pred)

print(a_1)
```

Σε αυτό το σημείο του κώδικα καλούμε την συνάρτηση SVC() για να αρχικοποιήσουμε το μοντέλο μας και έπειτα καλούμε την συνάρτηση `fit` του μοντέλου για να το εκπαιδεύσουμε πάνω στο MNIST dataset. Κατόπιν δοκιμάζουμε το μοντέλο μας στο υποσύνολο του dataset `x_test` και ελέγχουμε το μοντέλο μας με την μετρική `accuracy` η οποία μας δίνει τιμή `0.9792` κάτι που υποδηλώνει ότι το μοντέλο μας έχει πολύ καλή απόδοση.

```
#Cifar-10 dataset


#Loading Data
```

```
(x_train,y_train), (x_test,y_test) = cifar10.load_data()
n_images,n_rows,n_colimns,n_channels = x_train.shape

x_train = x_train.reshape((x_train.shape[0], -1))
x_test = x_test.reshape((x_test.shape[0], -1))
```

Ομοίως με προηγουμένως στο συγκεκριμένο dataset προσπαθούμε να το φορτώσουμε και να το αναδιαμορφώσουμε έτσι ώστε να μπορεί να μπει σαν input στο μοντέλο που θέλουμε να φτιάξουμε. Επειδή το Cifar-10 έχει από μόνο του shape 60.000 εικόνες με (32, 32, 3) τα οποία είναι pixels κανάλια προσπαθούμε να δημιουργήσουμε 60.000 vectors με την ίδια πληροφορία.

```
n_training = 5000
n_testing = 1000

x_train = x_train[0:n_training,:]
x_test = x_test[0:n_testing,:]
y_train = y_train[0:n_training]
y_test = y_test[0:n_testing]
#Shaping Data
```

Εδώ μικραίνουμε κατά πολύ το dataset γιατί λόγω του σχεδιασμού του SVC() επειδή τα μοντέλα αυτά προσπαθούν να βρουν το υπερεπίπεδο που διαχωρίζει στο μέγιστο τις διαφορετικές classes στον χώρο features και αυτή η διαδικασία τα καθιστά αργά σε μεγάλα datasets όπως το cifar-10.

```
#Shaping Data
scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)

# Hyperparameters Arrays
kernels = ["rbf", "poly", "linear","sigmoid"]
c_list = [0.01,0.1,0.5,0.8,1]
```

Ομοίως εδώ κάνουμε scaling τα δεδομένα μας για να βοηθήσει και στην ταχύτητα της εκπαίδευσης αλλά και στην ποιότητα της.

Από κάτω βρίσκονται οι πίνακες που θα μας βοηθήσουν να κάνουμε κάποιες δοκιμές στις παραμέτρους του μοντέλου μας.

4. Δοκιμές

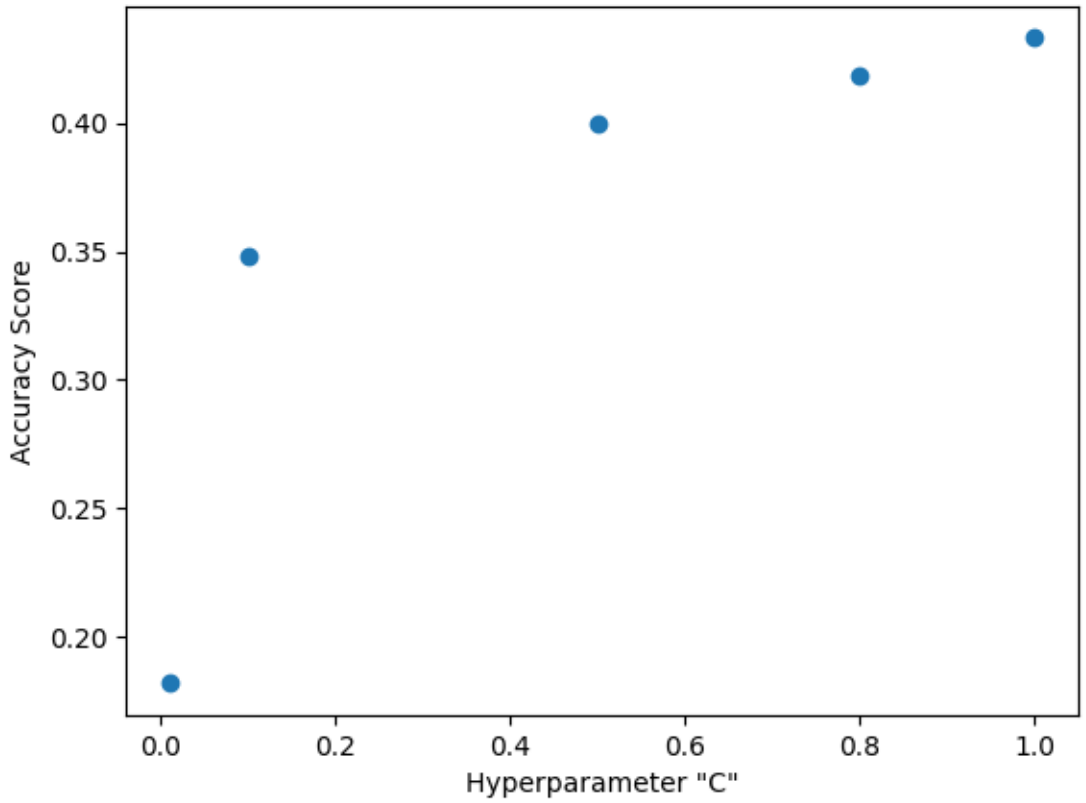
```
for k in kernels:
    temp = []
    for temp_c in c_list:
        model2 = SVC(kernel = k, C = temp_c)
        model2.fit(x_train,y_train.ravel())
        pred_2 = model2.predict(x_test)
        a_2 = accuracy_score(y_test, pred_2)
        temp.append(a_2)
        print("Accucacy Score with kernel:", k ,"and C:", temp_c , "----is:",
a_2)
    accuracy_results.append(temp)
```

Σε μία for loop δοκιμάζουμε διάφορους kernels αλλά και αλλάζουμε την παράμετρο “C” η οποία είναι η regularization parameter και έπειτα σχεδιάζουμε τα plots της εκάστοτε δοκιμής.

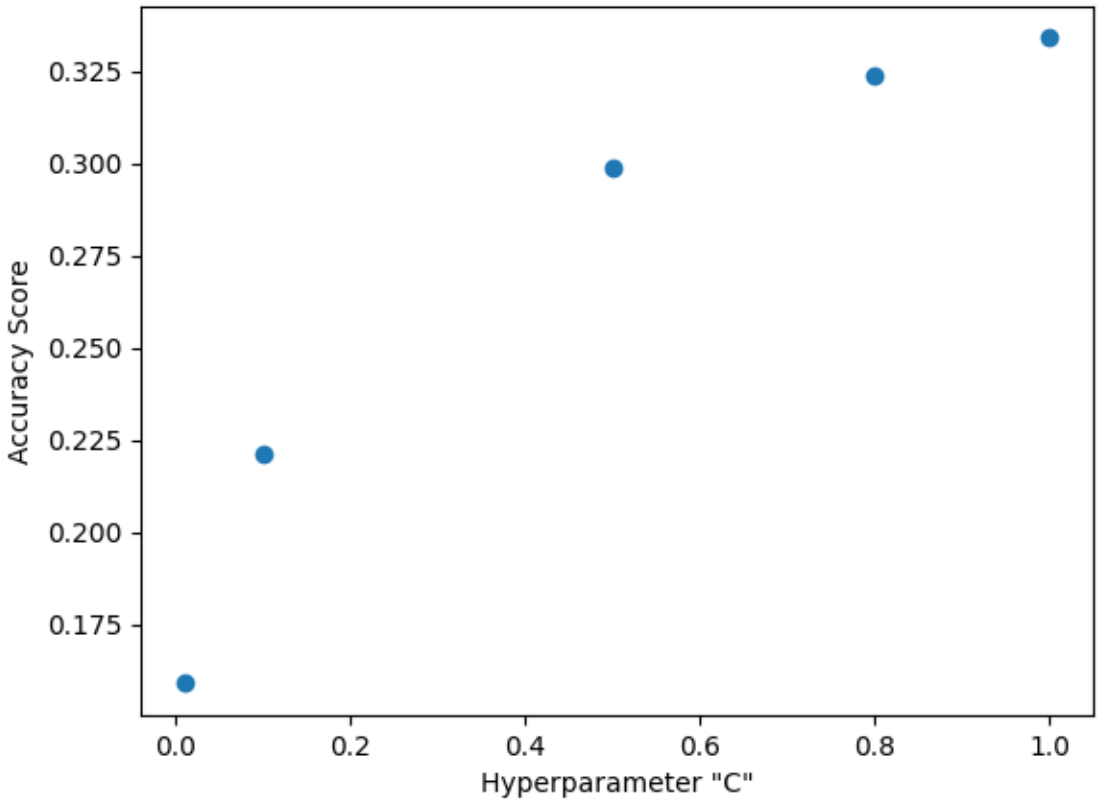
```
plt.plot(c_list, accuracy_results[0],'o')
plt.title="rbf"
plt.xlabel('Hyperparameter "C"')
plt.ylabel('Accuracy Score')
plt.show()
plt.plot(c_list, accuracy_results[1],'o')
plt.title="poly"
plt.xlabel('Hyperparameter "C"')
plt.ylabel('Accuracy Score')
plt.show()
plt.plot(c_list, accuracy_results[2],'o')
plt.title="linear"
plt.xlabel('Hyperparameter "C"')
plt.ylabel('Accuracy Score')
plt.show()
plt.plot(c_list, accuracy_results[3],'o')
plt.title="sigmoid"
plt.xlabel('Hyperparameter "C"')
plt.ylabel('Accuracy Score')
plt.show()
```

Των οποίων τα αποτελέσματα φαίνονται παρακάτω:

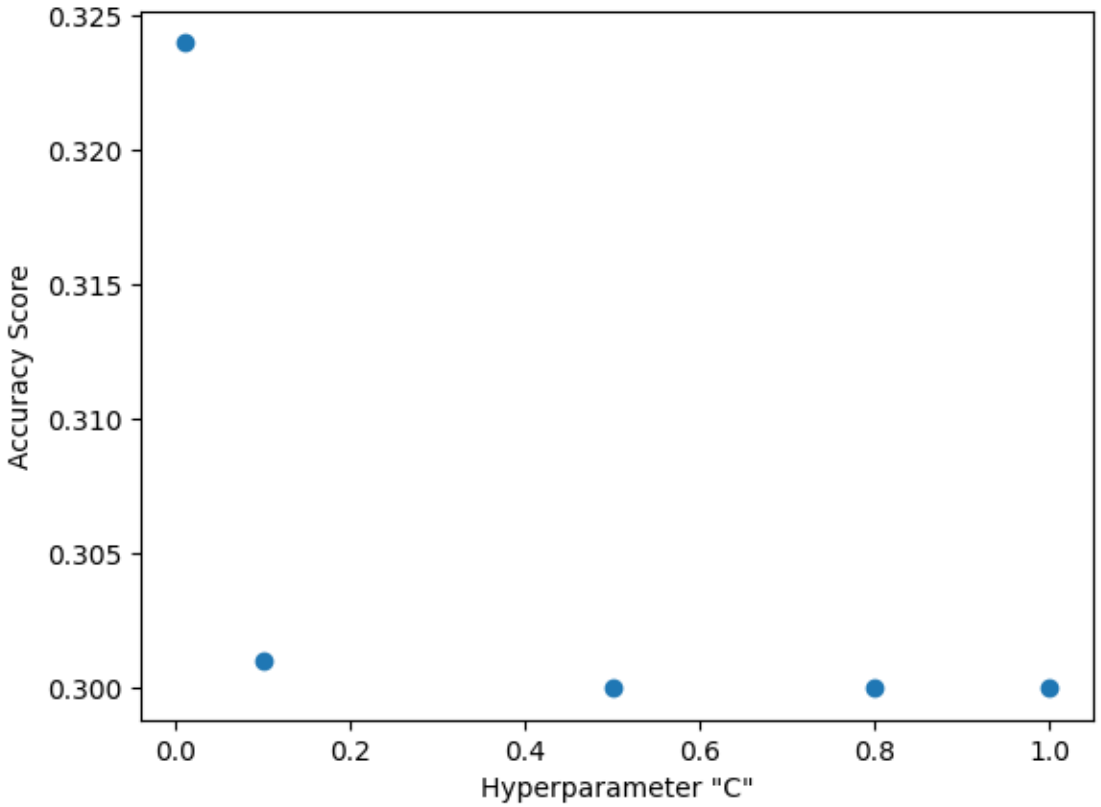
4.1 “rbf”



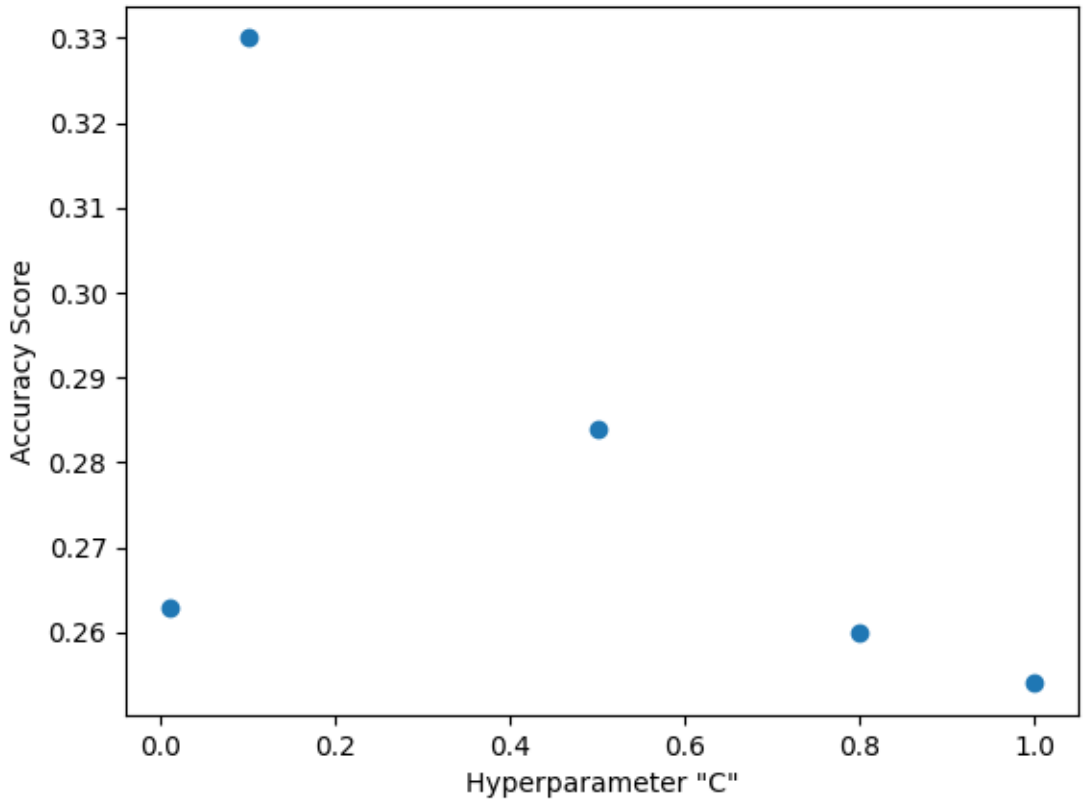
4.2 “poly”


0.325
0.300
0.275
0.250
0.225
0.200
0.175
Accuracy Score
0.0
0.2
0.4
0.6
0.8
1.0
Hyperparameter "C"

4.3 “linear”


0.325
0.320
0.315
0.310
0.305
0.300
Accuracy Score
0.0
0.2
0.4
0.6
0.8
1.0
Hyperparameter "C"

4.4 “sigmoid”



Τα αποτελέσματα δείχνουν ότι επικρατέστερος kernel για το συγκεκριμένο dataset είναι ο «rbf» και ειδικότερα με την παράμετρο "C" να είναι 1, όμως μπορούμε να δούμε ότι καμία από αυτές τις τιμές δεν είναι ικανοποιητικές και δεν μπορούμε να πούμε ότι φτιάξαμε ένα μοντέλο που όταν το τροφοδοτούμε με παραδείγματα θα μας δίνει ασφαλή συμπεράσματα. Μεγάλη είναι η διαφορά από το MNIST Dataset καθώς, η πρόσθετη πολυπλοκότητα της αναγνώρισης χρωμάτων και αντικειμένων μπορεί να κάνει πιο δύσκολο για ένα μοντέλο να εκπαιδευτεί στα δεδομένα CIFAR-10 σε σχέση με τις απλές γκρι εικόνες του MNIST.

Κώδικας ενδιάμεσης εργασίας:

```
from keras.datasets import mnist
from sklearn.neighbors import KNeighborsClassifier, NearestCentroid
from sklearn.metrics import accuracy_score
import time


clf1 = KNeighborsClassifier(1)
```

```
clf2 = KNeighborsClassifier(3)
CentroidClassifier = NearestCentroid()




(x_train,y_train), (x_test,y_test) = mnist.load_data()
x_train = x_train.reshape(x_train.shape[0], x_train.shape[1]**2)
x_test = x_test.reshape(x_test.shape[0], x_test.shape[1]**2)

start_time1 = time.time()
clf1.fit(x_train,y_train)
pred1 = clf1.predict(x_test)
print("Execution Time for kNN-1 Classifier --- %s seconds ---" % (time.time() -
start_time1))
print("Accuracy for kNN1 --- %s ---" % accuracy_score(y_test, pred1))

start_time2 = time.time()
clf2.fit(x_train,y_train)
pred2 = clf2.predict(x_test)
print("Execution for kNN-3 Classifier --- %s seconds ---" % (time.time() -
start_time2))
print("Accuracy for kNN1 --- %s ---" % accuracy_score(y_test, pred2))


start_time3 = time.time()
CentroidClassifier.fit(x_train,y_train)
pred3 = CentroidClassifier.predict(x_test)
print("Execution Time for Centroid Classifier --- %s seconds ---" % (time.time()
- start_time3))
print("Accuracy for kNN1 --- %s ---" % accuracy_score(y_test, pred3))

#Χρησιμοποιήσαμε την μετρική Accuracy για να εκτιμήσουμε την απόδοση των
Classifier μας στο συγκεκριμένο dataset.
#Με την βοήθεια της βιβλιοθήκης time καταφέραμε να μετρήσουμε τον Χρόνο εκτέλεσης
του κάθε ένα από τους Classifiers.
#Αφού δεν μπορούμε να εισάγουμε στο fit 3D data πρέπει να τα ανασχηματίσουμε σε
διαστάσεις που μας βολεύουν.
```

Με output στην κονσόλα:

Execution Time for kNN-1 Classifier --- 21.646045684814453 seconds ---

Accuracy for kNN1 --- 0.9691 ---

Execution for kNN-3 Classifier --- 18.950039625167847 seconds ---

Accuracy for kNN1 --- 0.9705 ---

Execution Time for Centroid Classifier --- 0.12199664115905762 seconds ---

Accuracy for kNN1 --- 0.8203 ---

Παρατηρούμε ότι ο Centroid Classifier δεν τα καταφέρνει καλά στο να ανεβάσει την μετρική σε επιθυμητά σημεία αλλά ο kNN καταφέρνει σε πολύ μικρό χρόνο να σχηματίσει ένα μοντέλο που έχει αρκετά μεγάλη ακρίβεια στο MNIST dataset, εφάμιλλη αυτής που δίνουν τα SVM που δοκιμάσαμε προηγουμένως. Την καλή απόδοση του kNN την αποδίδουμε στην χαμηλή πολυπλοκότητα του MNIST Dataset σε άλλα datasets περιμένουμε μεγαλύτερη απόδοση από το νευρωνικό μοντέλο.

Τα ίδια μοντέλα για το Cifar-10 dataset μας δίνουν τιμές:

```
from keras.datasets import cifar10
from sklearn.neighbors import KNeighborsClassifier, NearestCentroid
from sklearn.metrics import accuracy_score
import time

clf1 = KNeighborsClassifier(1)
clf2 = KNeighborsClassifier(3)
CentroidClassifier = NearestCentroid()



(x_train,y_train), (x_test,y_test) = cifar10.load_data()
n_images,n_rows,n_colimns,n_channels = x_train.shape

x_train = x_train.reshape((x_train.shape[0], -1))
x_test = x_test.reshape((x_test.shape[0], -1))

start_time1 = time.time()
clf1.fit(x_train,y_train)
pred1 = clf1.predict(x_test)
print("Execution Time for kNN-1 Classifier --- %s seconds ---" % (time.time() -
start_time1))
print("Accuracy for kNN1 --- %s ---" % accuracy_score(y_test, pred1))

start_time2 = time.time()
```

```
clf2.fit(x_train,y_train)
pred2 = clf2.predict(x_test)
print("Execution for kNN-3 Classifier --- %s seconds ---" % (time.time() -
start_time2))
print("Accuracy for kNN1 --- %s ---" % accuracy_score(y_test, pred2))

start_time3 = time.time()
CentroidClassifier.fit(x_train,y_train)
pred3 = CentroidClassifier.predict(x_test)
print("Execution Time for Centroid Classifier --- %s seconds ---" % (time.time()
- start_time3))
print("Accuracy for kNN1 --- %s ---" % accuracy_score(y_test, pred3))
```

Execution Time for kNN-1 Classifier --- 96.48795938491821 seconds ---

Accuracy for kNN1 --- 0.3539 ---

Execution for kNN-3 Classifier --- 93.10341382026672 seconds ---

Accuracy for kNN1 --- 0.3303 ---

Execution Time for Centroid Classifier --- 0.3750438690185547 seconds ---

Accuracy for kNN1 --- 0.2774 ---

Κάτι που μας δείχνει ότι στο συγκεκριμένο dataset ούτε αυτά τα μοντέλα είναι αποδοτικά,κάτι το οποίο είναι αναμενόμενο γιατί αυτοί οι classifiers δεν είναι ιδανικοί για complex image classification.

Τα plot files του κάθε run εμπεριέχονται στο zip μαζί με το αρχείο py.