Αναγνώριση Προτύπων
1η Εργαστηριακή Άσκηση

Ομάδα εργασίας :
- Συμπέθερος Αριστοτέλης-Γεώργιος (ΑΜ :031 16005)
- Σιφναίος Σάββας (ΑΜ : 031 16080)

9ο Εξάμηνο
ΣΗΜΜΥ

Εισαγωγή:

Σκοπός της πρώτης εργαστηριακής άσκησης είναι η υλοποίηση ενός συστήματος οπτικής αναγνώρισης χειρόγραφων ψηφίων. Συγκεκριμένα, διαθέτουμε δεδομένα τα οποία προέρχεται από το US Postal Service και πρόκειται για ένα σύνολο χειρόγραφων αριθμών από το 0 έως και το 9, οι οποίοι έχουν ψηφιοποιηθεί και κωδικοποιηθεί υπό την μορφή ενός ανύσματος 256 διαστάσεων. Στα πλαίσια της συγκεκριμένης εργασίας, θα μελετήσουμε διάφορους τύπους ταξινομητών προκείμενου να καταφέρουμε να αναπτύξουμε τελικά ένα σύστημα με όσο το δυνατόν μεγαλύτερη ακρίβεια στις προβλέψεις του.

Βήμα 1:

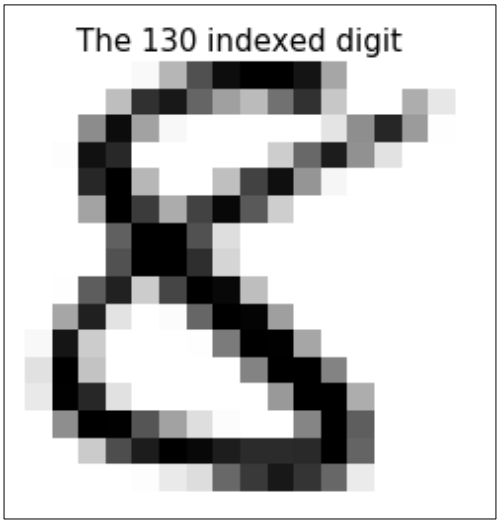
Διαβάζουμε αρχικά από τα 2 αρχεία ‘train.txt’ και ‘test.txt’ τα δεδομένα μας. Τα δεδομένα από κάθε αρχείο τα χωρίζουμε σε 2 πίνακες X και y, όπου ο X περιέχει σε κάθε γραμμή ένα δείγμα που αποτελείται από 256 feutures. Επιπλέον,γνωρίζουμε πως η πρώτη στήλη των αρχείων με τα δεδομένα αντιστοιχεί στο label της κλάσης. Για τον λόγο αυτό, απομονώνουμε την πρώτη στήλη ,καθενός από τα δύο αρχεία, και την αντιγράφουμε στον πίνακα y. Συνεπώς καταλήγουμε τελικά στους 4 πίνακες X_train,y_train,X_test,y_test. Στον κώδικα μας πραγματοποιούμε το παραπάνω και τυπώνουμε τα μεγέθη των πινάκων και παίρνουμε το παρακάτω στιγμιότυπο:

```
Train set shape is (2007, 256) and train labels shape is (2007,)
Test set shape is (2007, 256) and test labels shape is (2007,)
```

Το στιγμιότυπο αυτό μας δείχνει ότι έχουμε 2007 δείγματα και στα 2 set μας (Train & Test) τα οποία αποτελούνται από 256 feutures. Αντίστοιχα οι πίνακες y με τα labels είναι προφανώς ίδιοι σε μέγεθος, αλλά είναι μονοδιάστατοί καθώς περιέχουν μόνο την κλάση που ανήκει το κάθε δείγμα.

Βήμα 2:

Στο βήμα αυτό θέλουμε να σχεδιάσουμε το υπ’ αριθμόν 131 ψηφίο , δεδομένου οτι η αρίθμηση στον πίνακα μας αρχίζει από το 0 , θέλουμε να σχεδιάζουμε το στοιχείο με index=130. Καλούμε την show_sample με παράμετρο το train set και index=130. Η συνάρτηση με την σειρά της παίρνει το δείγμα από τον πίνακα με βάση το index και στην συνέχεια αναδιατάσσει ,μέσω του np.reshape ,τα 256 feutures σε έναν πίνακα 16x16 που στην συνέχεια μέσω της matplotlib εμφανίζει, όπως φαίνεται παρακάτω:



Βλέπουμε ότι πρόκειται για τον αριθμό 8, το οποίο μπορούμε να επιβεβαιώσουμε από τον πίνακα y_test:

```
1 print(f"The label of the 130 indexed digit is {y_train[130]}")
```

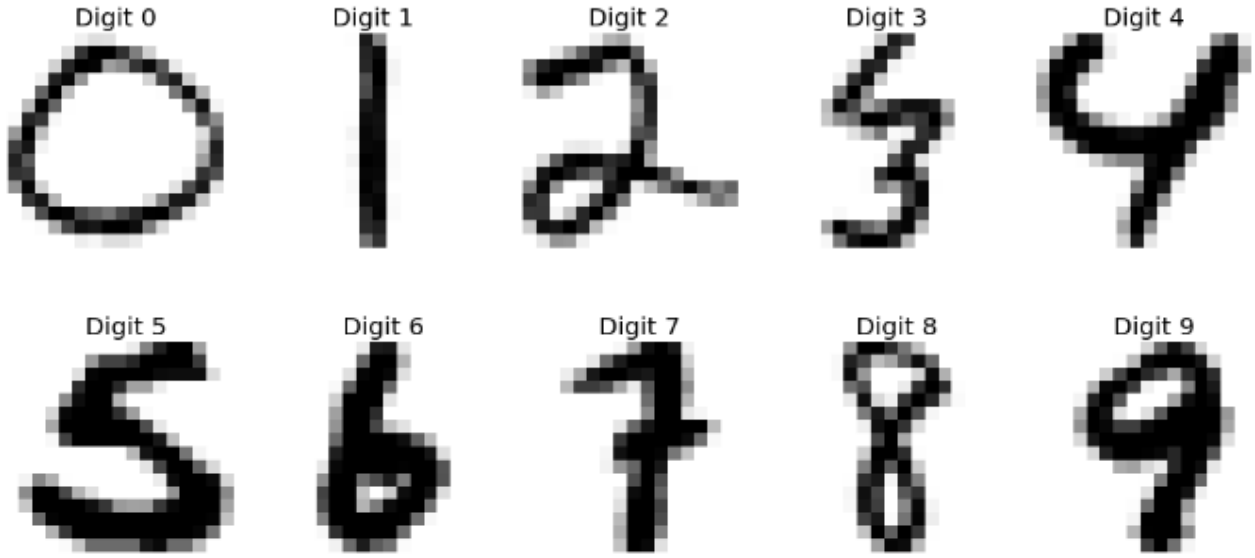
```
The label of the 130 indexed digit is 8
```

Επίσης κάναμε χρήση της παραμέτρου “gray_r” κατά την κλήση της plt.imshow() , ώστε οι τιμές του (16 x 16) πίνακα να αναπαρασταθούν ως κάποια από τις 256 αποχρώσεις του γκρι. Ωστόσο, η ‘gray_r’ κάνει reverse τις τιμές για τις οποίες ένα πίξελ γίνεται άσπρο ή μαύρο , έτσι ώστε οπτικοποιώντας με μαύρο το ψηφίο και με άσπρο το backround να είναι πιο οικείο στο μάτι.

Βήμα 3:

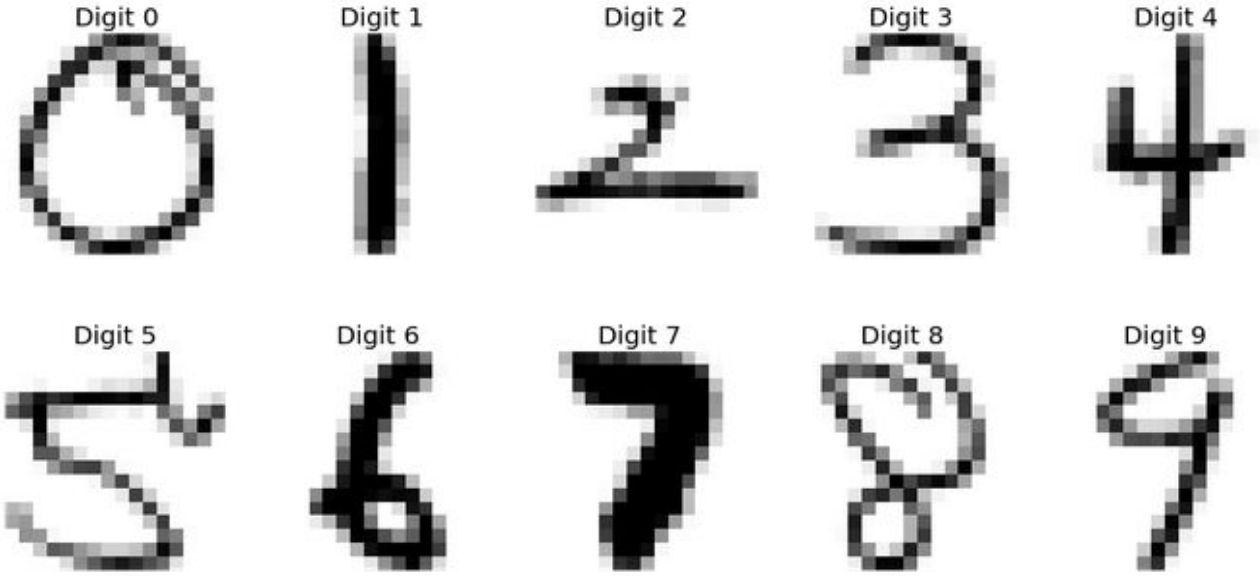
Στην συνέχεια επιλέγουμε 1 τυχαίο δείγμα για κάθε label ,δηλαδή για κάθε αριθμό από το 0 έως το 9. Επιλέγουμε τυχαία ένα δείγμα από το σύνολο εκπαίδευσης, αποθηκεύουμε το στοιχείο καθώς και ότι έχουμε δείγμα από το label αυτού του στοιχείου. Συνεχίζουμε την ίδια διαδικασία με προηγουμένως αυξάνοντας κατά 1 το index , μέχρι να έχουμε 1 δείγμα και από τους 10 αριθμούς. Σε περίπτωση που ξεπεράσει ο δείκτης μας το μέγεθος του πίνακα των δεδομένων επιλέγουμε ξανά ένα τυχαίο index. Παραθέτουμε παρακάτω 2 στιγμιότυπα από 2 εκτελέσεις της παραπάνω συνάρτησης (plot_digits_sample()) :

1)



---

2)



Βήμα 4-5:

Στα train δεδομένα υπολογίζουμε για όλες τις “εμφανίσεις” του ψηφίου ‘0’ το μέσο όρο (βήμα 4) και την διασπορά του (βήμα 5) στοιχείου/pixel (10,10) ,όπου όμοια με προηγουμένως δεδομένου ότι η αρίθμηση ξεκινά από το (0,0) το στοιχείο που εξετάζουμε πρόκειται για αυτό βρίσκεται στην ενδέκατη γραμμή και ενδέκατη στήλη.

Ο Μέσος όρος και η διασπορά που υπολογίζουμε φαίνεται παρακάτω:

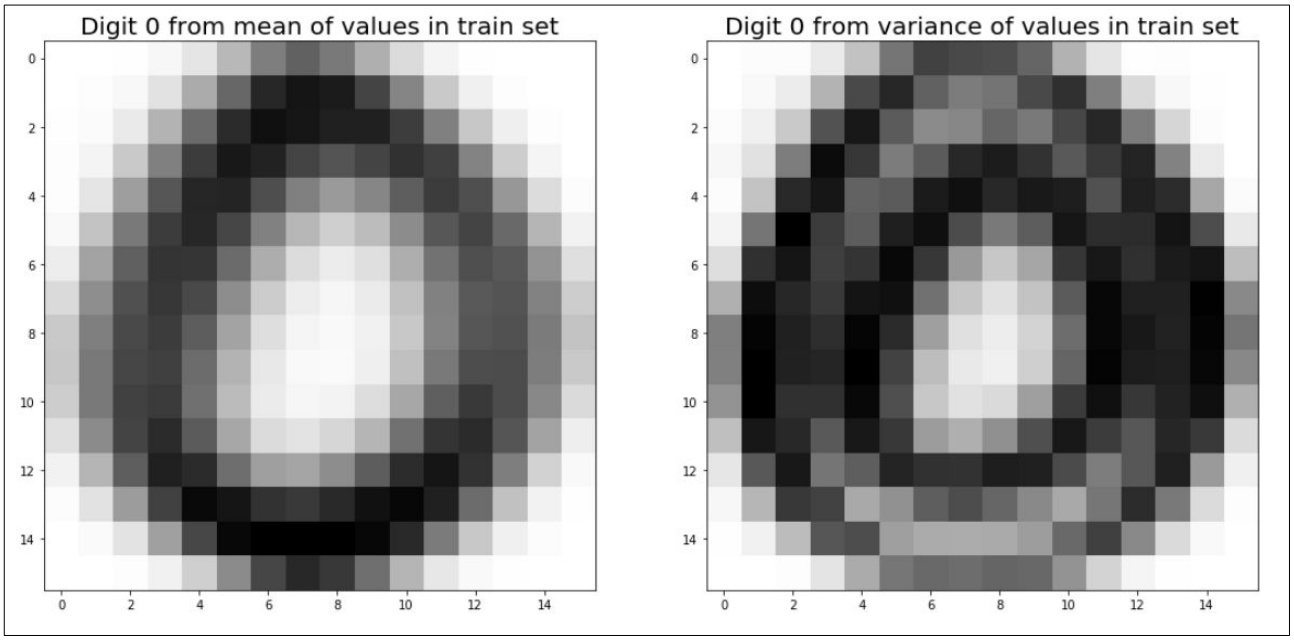
```
Mean Value of pixel (10, 10) for digit zero in train Set is -0.38300000000000006
```

```
Variance of pixel (10, 10) for digit zero in train Set is 0.6082944066852368
```

Λαμβάνοντας υπόψη ότι οι τιμές των pixel κυμαίνονται από το -1 έως το 1 , μπορούμε να αντιστοιχίσουμε το -1 στο λευκό και το 1 με τα μαύρο (με βάση τον την οπτικοποίηση προηγουμένως) .Βλέπουμε,λοιπόν, από την μέση τιμή ότι το στοιχείο (10,10) είναι πιο κοντά στο -1. Από την διασπορά επιπλέον θα μπορούσαμε να συμπεράνουμε πως ,λόγω της σχετικά υψηλής τιμής της, πρόκειται για στοιχείο που βρίσκεται σε θέση του “περιγράμματος” του γενικού σχήματος του ψηφίου αυτού και συνεπώς οδηγείται σε μεγαλύτερες τιμές διασποράς. Το θέμα αυτό θα συζητηθεί περαιτέρω σε επόμενο ερώτημα.

Βήμα 6-7-8:

Όμοια με τα προηγούμενα 2 ερωτήματα υπολογίζουμε την μέση τιμή και την διασπορά με βάση όλα τα στοιχεία του train set για το ψηφίο 0 ,για όλα του τα pixel, με τις συναρτήσεις digit_mean() και digit_variance() (βήμα 6). Στην συνέχεια σχεδιάζουμε αντικριστά το ψηφίο ‘0’ κάνοντας χρήση μόνο των μέσων τιμών στην μια περίπτωση (βήμα 7) και στην άλλη κάνοντας χρήση μόνο των τιμών της διασποράς (βήμα 8). Παρακάτω φαίνεται το αποτέλεσμα που λαμβάνουμε:



Αρχικά παρατηρούμε ότι και στις 2 περιπτώσεις μπορούμε με ευκολία να διακρίνουμε ότι πρόκειται για το ψηφίο 0. Για την αναπαράσταση με τις τιμές των μέσων όρων αρχικά έχουμε ότι κάθε τιμή pixel αναπαριστά της μέση τιμή που έχει μεταξύ όλων των εικόνων εκπαίδευσης. Παρατηρούμε πιο έντονα μαύρα pixel (πιο μεγάλες τιμές μέσου όρου) στις πάνω και κάτω “πλευρές” του ψηφίου ‘0’ ,ενώ στις πλαϊνές “πλευρές” έχουμε τιμές που οδηγούν σε γκρι pixel αλλά λιγότερο έντονα από ότι στις πάνω και κάτω πλευρές. Επίσης είναι προφανές ότι στα άκρα έχουμε λευκά τελείως pixel , αυτό συμβαίνει καθώς

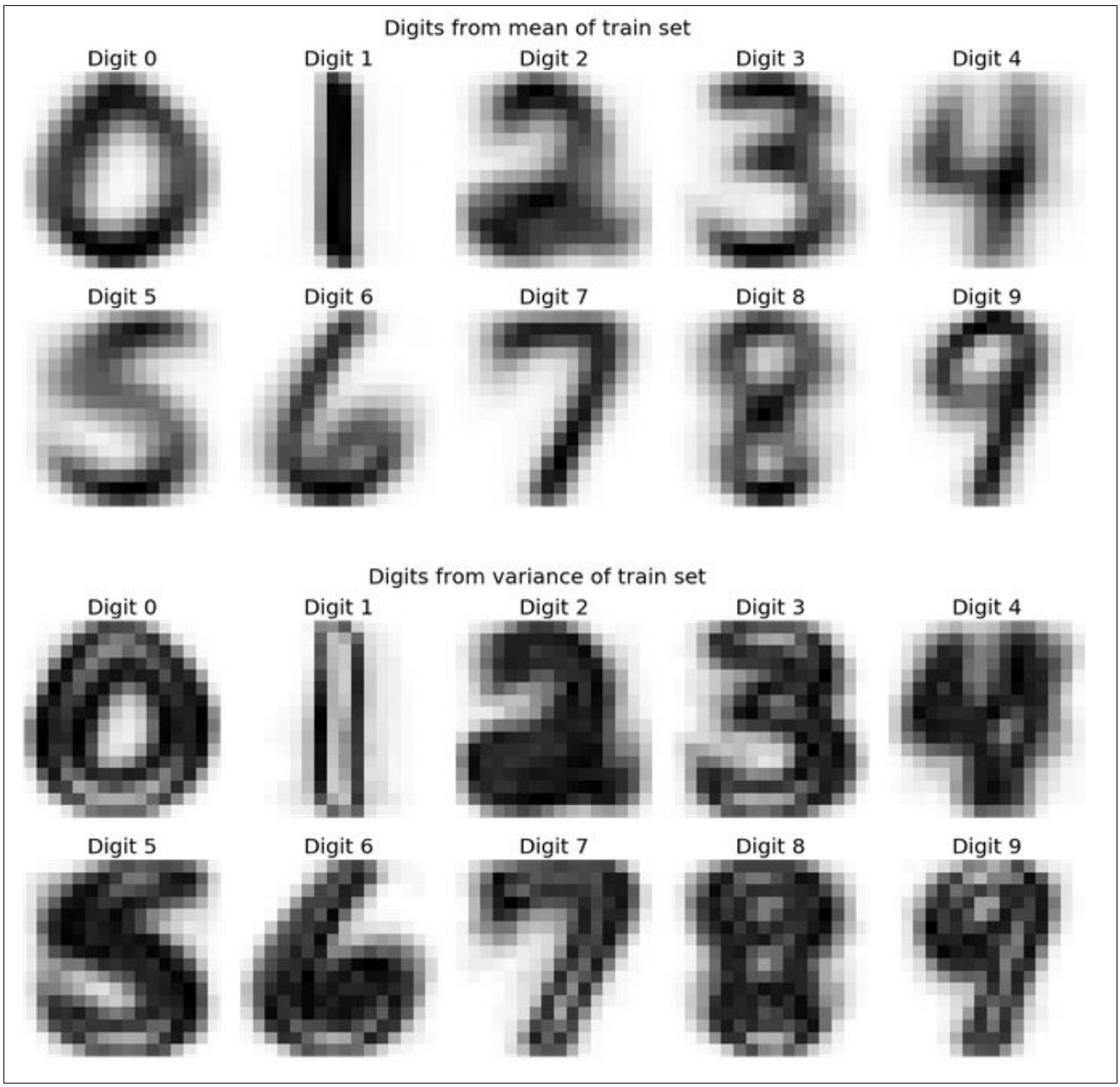
μεταξύ όλων των εμφανίσεων του δείγματος ‘0’ τα άκρα της εικόνας του είναι λευκά και συνεπώς ο μέσος όρους τους είναι και αυτός. Όμοια βλέπουμε ότι στο κέντρο της εικόνας έχουμε πάλι την εμφάνιση λευκών και ανοιχτών γκρι στοιχείων το οποίο συμβαίνει λόγω του “σχήματος” του ψηφίου 0 και συνεπώς στα περισσότερα δείγματα , στο κέντρο τους είναι λευκά.

Στην συνέχεια για την παρατήρηση που κάναμε προηγουμένως ,πως δηλαδή οι πάνω και κάτω πλευρές έχουν μεγαλύτερες τιμές (πιο μαύρες) από ότι οι πλαϊνές πλευρές, θεωρούμε ότι οφείλεται στο γεγονός ότι (δεδομένου ότι στο σύνολο δεδομένων που μας δίνεται τα ψηφία φαίνεται να έχουνε κεντραριστεί και προσαρμοστεί με βάση το “ύψος” των ψηφίων) στην πλειοψηφία των δειγμάτων του ψηφίου ‘0’ η πάνω και κάτω “πλευρά” βρίσκεται αντίστοιχα στην μέση των πρώτων γραμμών και στην μέση των τελευταίων αντίστοιχα. Ωστόσο η δεξιά και αριστερά πλευρές μπορούν άλλοτε να είναι πιο κοντά στα άκρα και άλλοτε λιγότερο κοντά (οδηγώντας σε πιο στρογγυλό 0 ή πιο οβάλ σχηματικά 0). Για τον παραπάνω λόγο θεωρούμε ότι οι τιμές έχουν μια μεγαλύτερη διαφοροποίηση μεταξύ τους και για τον λόγο αυτό οδηγούν σε σχετικά μικρότερους μέσους όρους και άρα λιγότερο σκούρα γκρι τιμές. Αυτή η τελευταία παρατήρηση μας οδηγεί στο δεξιά σχήμα όπου έχουμε σχεδιάσει την διασπορά του κάθε pixel στο σύνολο εκπαίδευσης. Η μεγαλύτερη διαφοροποίηση που μεταξύ ίδιων pixel σε άλλα δείγματα που αναφέραμε πρόκειται για την διασπορά. Συνεπώς στα στοιχεία αυτά περιμένουμε μεγαλύτερες τιμές διασποράς όπου και πράγματι η δεξιά και αριστερή πλευρά του ψηφίου ‘0’ στην γραφική αυτή (και ιδιαίτερα στο περίγραμμα) έχει πιο σκούρες τιμές γκρι και μαύρα πίξελ καθώς οι τιμές αυτές σε διαφορετικές δείγματα φαίνεται να έχουνε συχνά διαφορετικές τιμές. Εννοώ αντίθετα στις γωνίες του σχήματος και στο κέντρο που οι τιμές δεν αλλάζουν σχεδόν καθόλου βλέπουμε ότι έχουμε σχεδόν μηδενική διασπορά , όμοια στην κάτω και πάνω πλευρά έχουμε χαμηλές τιμές διασποράς καθώς μπορούμε να πούμε με απλά λόγια ότι περισσότερα παραδείγματα ‘0’ έχουνε παρόμοιες τιμές στα pixel αυτά.

Περαιτέρω παρατηρήσεις πάνω στην ποιοτική ερμηνεία των γραφικών αυτών θα γίνουν και στο επόμενο βήμα.

Βήμα 9:

Επαναλαμβάνουμε την παραπάνω διαδικασία για όλα τα ψηφία (πάνω στο σύνολο εκπαίδευσης πάντα) και στην συνέχεια τα σχεδιάζουμε όλα τα ψηφία με βάση την μέση τιμή αλλά επιπρόσθετα σχεδιάζουμε και όλα τα ψηφία χρησιμοποιώντας τις τιμές διασποράς τους. Τα αποτελέσματα που παίρνουμε είναι τα παρακάτω:



Τα αποτελέσματα που βλέπουμε συνάδουν με τα σχόλια που κάναμε στο προηγούμενο βήμα .Ένα πολύ καλό παράδειγμα αποτελεί το ψηφίο ‘1’ το οποίο από την σχεδίαση με μέσες τιμές βλέπουμε πως στην πλειοψηφία των δειγμάτων (μαύρα pixel) του έχουμε μια σχεδίαση που αποτελείται από μια ευθεία κάθετη γραμμή πάχους 2 pixel και έχουμε κάποιες περιπτώσεις (γκρι pixel) όπου έχουμε πιο φαρδιά γραμμή (η τιμές αυτές θα μπορούσαν να προέρχονται από ‘1’ όμοιου πάχους αλλά μετατοπισμένο λίγο αριστερά ή δεξιά). Αντίστοιχα βλέπουμε στην γραφική της διασποράς μικρές τιμές γύρω στο κέντρο του ‘1’ , καθώς αυτές είναι πολύ κοινές για όλα τα δείγματα του ψηφίου αυτού και μεγαλύτερες τιμές στο περίγραμμα του , καθώς στις τιμές αυτές βλέπουμε μεγαλύτερη διαφοροποίηση (πιο “χοντρά” ή πιο “λεπτά” σχεδιασμένες ευθείες). Μπορούμε να παρατηρήσουμε από όλα τα σχήματα με βάση την διασπορά ότι το γενικό περίγραμμα του ψηφίου έχει πιο μεγάλες τιμές (πιο σκούρες) .

Βήμα 10:

Στην συνέχεια κατασκευάζουμε έναν ευκλείδειο ταξινομητή, ο οποίος χρησιμοποιεί τους μέσους όρους κάθε κλάσης (βήμα 9) και για κάθε δείγμα που του δίνεται υπολογίζει τις ευκλείδειες αποστάσεις από όλα τα class means. Στην συνέχεια ταξινομεί τα δείγματα στην κλάση εκείνη της οποίας ο μέσος όρος απείχε λιγότερο. Ο συγκεκριμένος ταξινομητής αποτελεί μια υποπερίπτωση του Bayes Classifier με κανονικές παραμετρικές κατανομές , όπου οι πίνακες συνδιακύμανσης ανάμεσα σε όλες τις κατηγορίες ταυτίζονται με το μοναδιαίο ενώ οι a-priori πιθανότητες κάθε κλάσης θεωρούνται ίσες.
Η συνάρτηση euclidean_distance() επιστρέφει την ευκλείδεια απόσταση μεταξύ των 2 δοθέντων διανυσμάτων (που δίνονται ως παράμετροι). Γίνεται χρήση της έτοιμης συνάρτησης linalg.norm της βιβλιοθήκης numpy. Η ευκλείδεια απόσταση υπολογίζεται ως εξής , έστω *p* το διάνυσμα (feature vector) ενός δείγματος που θέλουμε να εξετάσουμε και *q* το διάνυσμα μέσων όρων της κλάσης που εξετάζουμε στην επανάληψη αυτή. Η απόσταση υπολογίζεται με τον παρακάτω τύπο (όπου *n*=256 στην περίπτωση μας):

$$d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_i - q_i)^2 + \cdots + (p_n - q_n)^2}$$

Κατασκευάζουμε την συνάρτηση euclidean_distance_classifier() η οποία δέχεται σαν παραμέτρους έναν πίνακα διαστάσεων X , n_samples x n_feutures όπου στην περίπτωση μας τα n_feutures=256 καθώς και έναν πίνακα X_mean ο οποίος έχει διαστάσεις n_classes x n_features όπου η n_classes=10 και n_features=256 όπως αναφέραμε.
Στην συνέχεια για κάθε δείγμα (γραμμή) του X υπολογίζουμε την ευκλείδεια απόσταση του από κάθε γραμμή του πίνακα X_means , δηλαδή την απόσταση του από τα class means όπως αναφέραμε στην αρχή. Η πρόβλεψη που δίνει ο ταξινομητής είναι η κλάση της οποίας το διάνυσμα μέσων όρων στον πίνακα πίνακας X_mean απείχε το λιγότερο κατά ευκλείδεια απόσταση από το υπό εξέταση δείγμα.
Στην συνέχεια ταξινομούμε το υπ’αριθμόν 101 στοιχείο του test set μας, δηλαδή (εφόσον η αρίθμηση ξεκινά από το 0) το στοιχείο με δείκτη 100. Η ταξινομητής εκτιμά ότι ανήκει στην κλάση 0 και εξετάζοντας τον πίνακα y_test βλέπουμε ότι πράγματι ανήκει στην κλάση αυτή. Παρακάτω παραθέτουμε στιγμιότυπο από την έξοδο :

```
Digit with in 101 position is classified as 0
Digit with in 101 position is actually a 0
```

Βήμα 11:

Στην συνέχεια ταξινομούμε όλα τα στοιχεία του test set και εκτιμούμε το ποσοστό επιτυχίας με χρήση της accuracy_score() (της βιβλιοθήκης sklearn.metrics) στην οποία δίνουμε ως παραμέτρους τα αποτελέσματα του εκτιμητή μας , καθώς και των πίνακα y_test που περιέχει τα πραγματικά labels κάθε δείγματος. Τα αποτελέσματα είναι τα παρακάτω:

```
print(f'The accuracy of the Euclidean Distance Classifier is {accuracy_score(y_pred,y_test)}')
```
```
The accuracy of the Euclidean Distance Classifier is 0.8196312904833084
```

Όπου παρατηρούμε ότι ο ταξινομητής μας είχε επιτυχία σχεδόν 82%.

Βήμα 12:

Υλοποιούμε στην συνέχεια μια κλάση για τον παραπάνω ταξινομητή συμβατή με τον scikit-learn estimator. Η υλοποίηση της , όπως και όλα από τα παραπάνω ,βρίσκεται στο αρχείο lib.py. Η ίδια είναι με το όνομα EuclideanDistanceClassifier(). Στην συνέχεια φτιάχνουμε ένα στιγμιότυπο της παραπάνω κλάσης ,το οποίο κάνουμε fit στο σύνολο εκπαίδευσης και στην συνέχεια (με χρήση της predict) υπολογίζουμε το ποσοστό επιτυχίας της, που όπως ήταν αναμενόμενο είναι ίδιο με προηγουμένως. Τα παραπάνω φαίνονται παρακάτω:

```
clf = lib.EuclideanDistanceClassifier()
clf.fit(X_train,y_train)
clf.predict(X_test)
print(f'Once again, the accuracy of Euclidean Distance Classifier is :')
print(clf.score(X_test,y_test))
```
```
Once again, the accuracy of Euclidean Distance Classifier is :
0.8196312904833084
```

Βήμα 13:

(α) Κατασκευάζουμε την evaluate_classifier() η οποία δέχεται ως ορίσματα έναν classifier (sklearn.base.BaseEstimator) ,το σύνολο των δεδομένων των ψηφίων (πίνακα) και έναν πίνακα με τα labels των τελευταίων. Με την σειρά της κάνει χρήση της συνάρτησης cross_val_score() της βιβλιοθήκης sklearn.model_selection ,η οποία με την σειρά της δέχεται ως παραμέτρους έναν classifier, το σύνολο X και τα labels y πάνω στο οποίο θα κάνει το cross validation ,τον αριθμό των folds τα οποία θα κάνει (Στην περίπτωση μας 5) καθώς και τη μετρική την οποία θα χρησιμοποιήσει για την αξιολόγηση ,που στην περίπτωση μας είναι το ποσοστό επιτυχίας. Στην συνέχεια επιστρέφουμε τον μέσο όρο και των 5 folds. Τα αποτελέσματα μας φαίνονται παρακάτω:

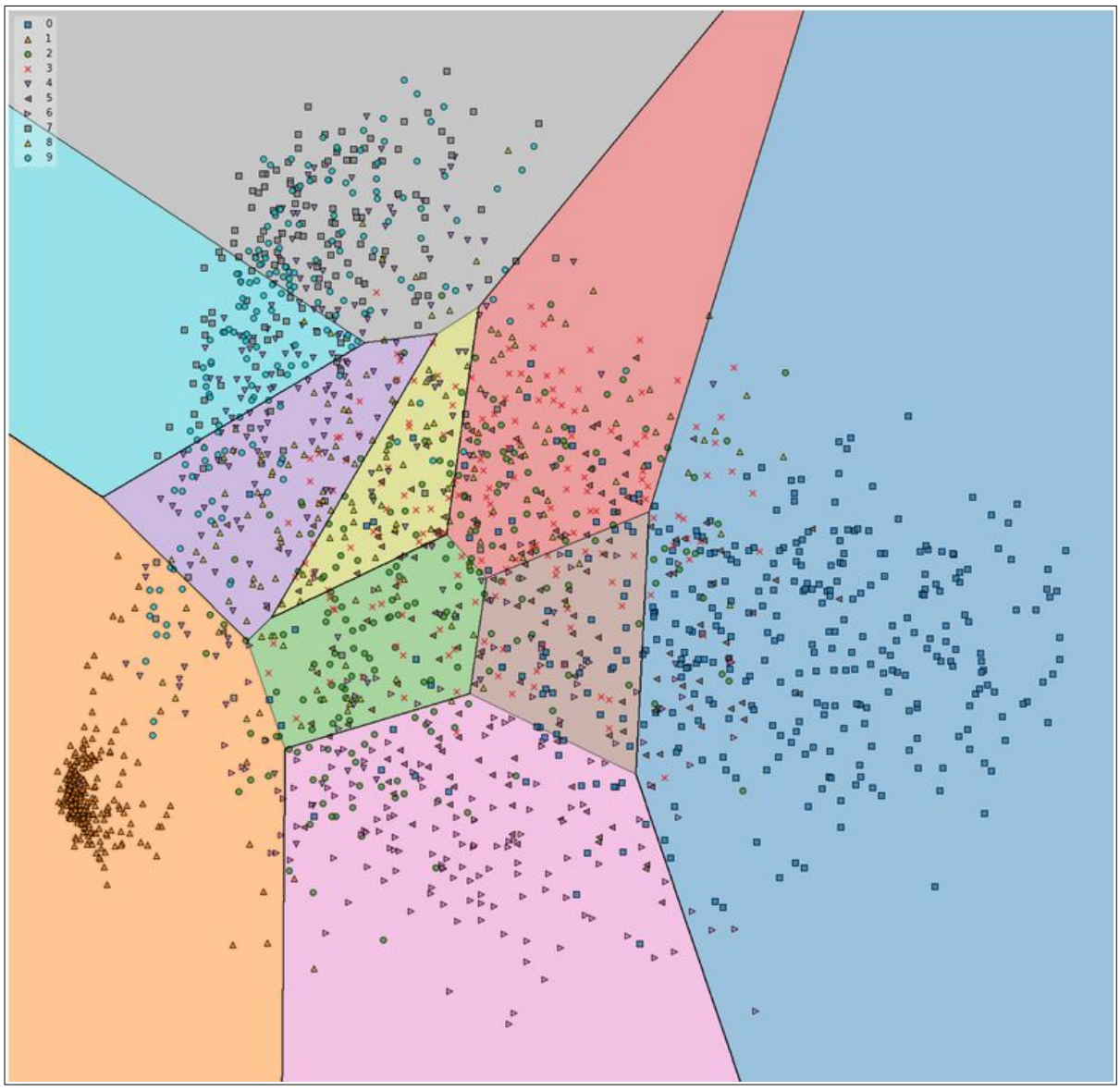
```
print (f'The accuracy of Euclidean Distance Classifier when using 5-fold-cross-validation is :')
print(lib.evaluate_classifier(clf,X_train,y_train))
```
```
The accuracy of Euclidean Distance Classifier when using 5-fold-cross-validation is :
0.8071736082678875
```

Το αποτέλεσμα όπως βλέπουμε είναι λίγο χαμηλότερο από αυτό που υπολογίσαμε νωρίτερα το οποίο αποτελεί πιο αντικειμενική τιμή καθώς εξετάζει διαφορετικούς χωρισμούς των δεδομένων και συνεπώς επηρεάζεται λιγότερο από το πως έτυχε να διαχωριστούν εξαρχής.

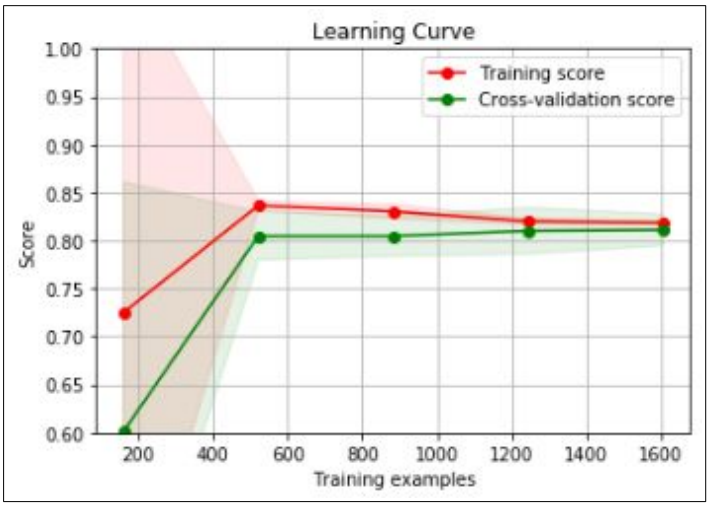
Στο σημείο αυτό αξίζει να σημειωθεί πως εφόσον δεν πραγματοποιούμε κάποιο fine tuning των υπερπαραμέτρων του classifier μας, το cross validation θα μπορούσε να πραγματοποιηθεί και πάνω στην συνένωση των δύο συνόλων (train και test set) όπως συμβαίνει παρακάτω.

(β)

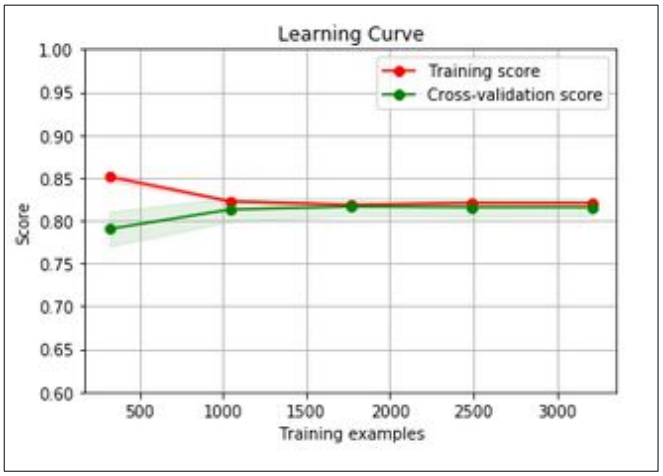
Στην συνέχεια θέλουμε να σχεδιάσουμε την περιοχή απόφασης του ευκλείδειου ταξινομητή. Προφανώς δεν μπορούμε να το σχεδιάσουμε ως έχει καθώς το κάθε δείγμα αποτελείται από 256 διαστάσεις. Ωστόσο μπορούμε να εφαρμόσουμε ανάλυση σε κύριες συνιστώσες (Principal component analysis) για μείωσης του dimensionality σε 2, ώστε να μπορέσουμε να αναπαραστήσουμε γραφικά το ζητούμενο. Εφόσον οι διαστάσεις μας γίνουν πλέον 2 με χρήση της plot_decision_regions της βιβλιοθήκης mlxtend.plotting οπτικοποιούμε τις περιοχές απόφασης για τον ταξινομητή μας, όπου φαίνεται ο διαχωρισμός των 10 κλάσεων. Είναι προφανές ότι λόγω μείωσης των διαστάσεων δεν αντικατοπτρίζονται ορθά οι περιοχές απόφασης του ταξινομητή μας.
Η γραφική παράσταση είναι η παρακάτω:



(γ) Στο σημείο αυτό σχεδιάζουμε την καμπύλη εκμάθησης του ευκλείδειου ταξινομητή, όπου γίνεται χρήση της συνάρτησης learning_curve() της βιβλιοθήκης sklearn.model_selection. Η οποία δέχεται ως ορίσματα τον classifier μας , το σύνολο εκπαίδευσης , τα label του συνόλου εκπαίδευσης, των αριθμό των folds που θα εκτελέσει για το cross validation και ένα ποσοστό του συνόλου δειγμάτων που θα χρησιμοποιήσει για να δημιουργήσει την καμπύλη εκμάθησης. Η συνάρτηση αυτή μας επιστρέφει τα score , πάντα ως την μετρική accuracy, τόσο για το training set όσο και για το test καθώς επίσης και το πλήθος των δειγμάτων από το σύνολο εκπαίδευσης που χρησιμοποιήθηκαν. Στην συνέχεια , καλούμε την συνάρτηση plot_learning_curve με παραμέτρους τις εξόδους της learning_curve προκειμένου να οπτικοποιήσουμε την καμπύλη εκμάθησης του ταξινομητή μας. Η τελική καμπύλη που προκύπτει παρατίθεται παρακάτω .



Στην οποία παρατηρούμε το προφανές, πως όσο αυξάνονται τα δείγματα εκπαίδευσης αυξάνεται και το ποσοστό επιτυχίας ιδιαίτερα στις πρώτες τιμές (200-500) . Στην συνέχεια με περαιτέρω αύξηση βλέπουμε πως σταδιακά βελτιώνεται το Cross-validation score και αρχίζει να σταθεροποιείται το training score. Όσο αυξάνονται τα δείγματα η διαφορά μεταξύ των 2 μειώνεται όλο ένα και παραπάνω το οποίο είναι αναμενόμενο καθώς το πλέον υπάρχουν αρκετά περισσότερα δείγματα μέσω των οποίων μπορεί να κατατάξει καλύτερα και με μεγαλύτερη σταθερότητα σε αντίθεση με τα λίγα δείγματα όπου ο τυχαίος διαχωρισμός μπορεί να επηρεάσει αρκετά τα αποτελέσματα, καθώς είναι αρκετά πιο πιθανό να έχει δει κανένα ή πολύ λίγα δείγματα από μία κλάση. Παρατηρούμε επίσης από την σταθερότητα και των 2 μετρικών στις τελευταίες τιμές των δειγμάτων πως πιθανώς η αύξηση του συνόλου δειγμάτων δεν θα οδηγήσει κάποια περαιτέρω βελτίωση. Το παραπάνω επιβεβαιώνεται αν κατασκευάσουμε την learning curve αλλά ως δεδομένα δώσω και τα δύο set ,training και testing. Άρα συνολικά 2007 +2007=4014 δείγματα . Η γραφική που παίρνουμε είναι η παρακάτω:



Όπου πράγματι μετά τα 1600-1700 περίπου δείγματα βλέπουμε ότι και οι 2 μετρικές έχουν σταθεροποιηθεί και μάλιστα ταυτίζονται σχεδόν μεταξύ τους, στην τιμή περίπου 82%.

Βήμα 14 :

Το πρώτο βήμα κατά την υλοποίηση ενός Naive Bayes ταξινομητή είναι ο υπολογισμός των apriori πιθανοτήτων κάθε κλάσης. Για τον υπολογισμό των πιθανοτήτων αυτών ορίσαμε την συνάρτηση calculate_priors η οποία λαμβάνοντας ως ορίσματά το σύνολο δεδομένων εκπαίδευσης και τα labels που αντιστοιχούν σε κάθε δείγμα , μετρά το πλήθος των δειγμάτων από κάθε κλάση και διαιρώντας με το συνολικό αριθμό δειγμάτων επιστρέφει , τελικά , ένα μονοδιάστατο np.array δέκα θέσεων που σε κάθε θέση περιέχει την apriori πιθανότητα της αντίστοιχης κλάσης.

Συγκεκριμένα, για τον υπολογισμό των apriori πιθανοτήτων κάθε κλάσης χρησιμοποιήθηκε η σχέση:

$$Pr[\omega_i] = \frac{number\ of\ instances\ i}{total\ number\ of\ instances}$$

Βήμα 15 :

(α) Συνεχίζοντας την υλοποίηση του Naive Bayes Classifier , υποθέτουμε πως όλα τα features στα δείγματά μας είναι μεταξύ τους ασυσχέτιστα και επομένως αυτά μπορούν να περιγραφούν από μία και μόνο Gaussian κατανομή. Επιπλέον, λαμβάνοντας υπόψη την θεωρία απόφασης του Bayes έχουμε :

$$p(x_j|\omega_i) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x_j-\mu_{ji})^2}{2\sigma^2}}$$

και λόγω της ανεξαρτησίας των features ,που έχουμε υποθέσει, καταλήγουμε στον υπολογισμό των posteriori πιθανοτήτων σύμφωνα με την σχέση :

$$p(x|\omega_i)=\prod_{j=1}^{256} p(x_j|\omega_i)=\prod_{j=1}^{256} \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x_j-\mu_{ji})^2}{2\sigma^2}}$$

Στην συνέχεια, ο ταξινομητής μας υπολογίζει την πιθανότητα μιας κλάσης δεδομένου του συγκεκριμένου δείγματος λαμβάνοντας υπόψη τόσο τις posteriori όσο και τις apriori πιθανότητες που έχουμε υπολογίσει , ακολουθώντας τον παρακάτω τύπο :

$$p(\omega_i|x)\propto p(x|\omega_i)*P(\omega_i)$$

Στο σημείο αυτό, αξίζει να αναφέρουμε πως για την αποφυγή του υπολογισμού γινομένων με πολλαπλούς όρους καθώς και του εκθετικού όρου που εμπεριέχεται στον τύπο της Gaussian κατανομής, επιλέξαμε να λάβουμε τον λογάριθμο των παραπάνω ποσοτήτων και έτσι τελικά καταλήγουμε στην εξής σχέση :

$$p(\omega_i|x)\propto p(x|\omega_i)*P(\omega_i)\Rightarrow \ln(p(\omega_i|x))\propto \ln(p(x|\omega_i)*P(\omega_i))\Rightarrow$$
$$p(\omega_i|x)\propto \ln(P(\omega_i))+\sum_{j=1}^{256}\frac{x_j-\mu_{ji}}{2\sigma^2}-\frac{1}{2}\ln(2\pi\sigma^2)$$

Τέλος , ο Naive Bayes Classifier ταξινομεί το εκάστοτε δείγμα στην κλάση εκείνη για την οποία η παραπάνω πιθανότητα μεγιστοποιείται.

Ακολουθώντας, λοιπόν , την παραπάνω θεωρητική ανάλυση, επιχειρήσαμε να κατασκευάσουμε μια δική μας υλοποίηση του συγκεκριμένου ταξινομητή η οποία θα είναι συμβατή με το πακέτο scikit-learn. Αρχικά, ορίσαμε μία python class CustomNBClassifier που κληρονομεί τους BaseEstimator και ClassifierMixin του πακέτου scikit-learn. Στην κλάση αυτή, ορίσαμε τρεις μεθόδους την fit , την predict καθώς και την score. Η πρώτη μέθοδος, δέχεται ως ορίσματα ένα σύνολο δεδομένων εκπαίδευσης καθώς και τα labels που αντιστοιχούν στο εκάστοτε δείγμα του συνόλου και υπολογίζει τη μέση τιμή των features για κάθε ψηφίο στο σύνολο εκπαίδευσης , καθώς και την διακύμανσή τους και τα αποθηκεύει τελικά στα np.arrays X_mean και X_var αντίστοιχα. Τέλος, η συγκεκριμένη μέθοδος υπολογίζει και τις apriori πιθανότητες των ψηφίων. Η μέθοδος predict δέχεται ως όρισμα ένα σύνολο δεδομένων διαφορετικό του συνόλου εκπαίδευσης , το test set , και πραγματοποιεί τον υπολογισμό των πιθανοτήτων όπως ακριβώς αναλύσαμε παραπάνω και στο τέλος επιστρέφει ένα μονοδιάστατο np.array που σε κάθε θέση του έχει την πρόβλεψη της κλάσης στην οποία ανήκει το αντίστοιχο δείγμα. Ο υπολογισμός των posteriori πιθανοτήτων γίνεται με την βοήθεια της συνάρτησης gaus_prob , η οποία με ορίσματα κάποιο feature και την αντίστοιχη μέση τιμή την διακύμανση και επιστρέφει τον λογάριθμο της πιθανότητας . Σε περίπτωση που η διακύμανση είναι μηδενική , προσθέτει μια πολύ μικρή ποσότητα (1e-9) για την αποφυγή απειρισμού της ποσότητας $\frac{x_j-\mu_{ji}}{2\sigma^2}$. Τέλος, η μέθοδος score δέχεται ως ορίσματά της ένα test set και τα αντίστοιχα labels του και καλώντας την predict επιστρέφει το accuracy του ταξινομητή μας.

(β) Το accuracy score του ταξινομητή μας, όπως αναφέραμε και προηγουμένως μπορεί να βρεθεί καλώντας την μέθοδο score του CustomNBClassifier και είναι ίσο με 66,52% , όπως φαίνεται και παρακάτω :

```
time0 = time()
print(clf.score(X_test,y_test))
print("\nTotal Time (in minutes) =",(time()-time0)/60)
```

```
0.6651718983557549

Total Time (in minutes) = 0.40926579634348553
```

(γ) Στο σημείο αυτό θα χρησιμοποιήσουμε την έτοιμη υλοποίηση του ίδιου ταξινομητή από το πακέτο scikit-learn, προκειμένου να συγκρίνουμε τις επιδόσεις των δύο ταξινομητών. Ο ταξινομητής αυτός εκπαιδεύτηκε πάνω στο ίδιο σύνολο δεδομένων και αξιολογήθηκε επίσης με βάση το ίδιο test set. Τα αποτελέσματα του συγκεκριμένου ταξινομητή παρατίθενται παρακάτω.

Παρατηρούμε πως και η υλοποίηση του scikit-learn έχει ακριβώς το ίδιο accuracy με την δική μας. Ωστόσο, κρίνεται αναγκαίο να αναφέρουμε πως η έτοιμη υλοποίηση του συγκεκριμένου ταξινομητή είναι αρκετά ταχύτερη από την δική μας. Το πανομοιότυπο των αποτελεσμάτων των δύο υλοποιήσεων θα μπορούσε πιθανώς να αποδοθεί στο γεγονός πως και οι δύο χρησιμοποιούν ως smoothing factor για τις μηδενικές διακυμάνσεις την ποσότητα 1e-9.

Βήμα 16

Συνεχίζοντας την μελέτη μας, επιλέγουμε να δοκιμάσουμε να ταξινομήσουμε τα δείγματα του test set χρησιμοποιώντας έναν Naive Bayes Classifier, ο οποίος υποθέτει μοναδιαία διακύμανση για όλα τα features του συνόλου εκπαίδευσης. Για να το πετύχουμε αυτό, τροποποιούμε ελαφρώς την python class που υλοποιήσαμε στο προηγούμενα βήμα , ώστε κατά την αρχικοποίησή της να λαμβάνει μία boolean μεταβλητή , την use_unit_variance , η οποία από προεπιλογή θα είναι False. Όταν, λοιπόν, η μεταβλητή αυτή λάβει την τιμή True, κατά την κλήση της μεθόδου fit αντί να υπολογίσουμε την διακύμανση των features για τα δείγματα κάθε κλάσης , θέτουμε το np.array X_var_ ίσο με ένα δισδιάστατο np.array μεγέθους (10,256) που σε κάθε του θέση περιέχει την τιμή 1.

Ακολουθώντας, την ίδια λογική με αυτή στο προηγούμενο βήμα , υπολογίζουμε το accuracy score αυτού του ταξινομητή , το οποίο και παρουσιάζεται παρακάτω .

```
clf=lib.CustomNBClassifier(use_unit_variance=True)
clf.fit(X_train,y_train)
clf.score(X_test,y_test)
```

```
0.8186347782760339
```

Παρατηρούμε πως το accuracy score του συγκεκριμένου ταξινομητή είναι αρκετά μεγαλύτερο συγκριτικά με αυτό του "κλασσικού" Naive Bayes . Η αύξηση στην ακρίβεια των προβλέψεων του ταξινομητή μπορεί ενδεχομένως να οφείλεται στο γεγονός πως ο ταξινομητής δίνει ,πλέον, περισσότερη έμφαση στην μέση τιμή των features με αποτέλεσμα να προσεγγίζει περισσότερο την συμπεριφορά του EuclideanDistanceClassifier που υλοποιήσαμε παραπάνω.

Βήμα 17 :

Για λόγους πληρότητας, στο σημείο αυτό θα χρησιμοποιήσουμε και κάποιους άλλους ταξινομητές προκειμένου να μπορέσουμε να καταλήξουμε ,τελικά, στον βέλτιστο ταξινομητή για την συγκεκριμένη εργασία.

Στα πλαίσια ,λοιπόν, της μελέτης αυτής θα χρησιμοποιήσουμε τους εξής ταξινομητές :

1. Linear SVM Classifier
2. Rbf SVM Classifier
3. Polynomial SVM Classifier
4. Sigmoid SVM Classifier
5. 5 Nearest Neighbors Classifier
6. scikit-learn Naive Bayes Classifier
7. Custom Naive Bayes Classifier
8. Euclidean Classifier

Επιπλέον, αξίζει να αναφέρουμε πως για την εξαγωγή του accuracy καθενός από τους παραπάνω ταξινομητές θα κάνουμε χρήση της μεθόδου του cross validation με 5 folds , όπου το σύνολο δεδομένων θα είναι η συνένωση των test και train set που είχαμε μέχρι τώρα.

Για την εξαγωγή των accuracy κάθε ταξινομητή, ορίσαμε της συναρτήσεις evaluate_{classifier} , όπου {classifier} το όνομα καθενός από τους παραπάνω ταξινομητές. Όλες οι συναρτήσεις παίρνουν ως ορίσματα ένα σύνολο δεδομένων , τα labels που αντιστοιχούν στο συγκεκριμένο σύνολο καθώς και το πλήθος των folds που θα πραγματοποιηθούν κατά το cross validation ,στο οποίο πλήθος από προεπιλογή του έχει ανατεθεί η τιμή 5. Στην συνέχεια, η κάθε συνάρτηση αρχικοποιεί τον αντίστοιχο ταξινομητή και έπειτα καλεί την συνάρτηση evaluate_classifier με ορίσματα τον classifier που μόλις ορίσθηκε , το σύνολο δεδομένων , τα αντίστοιχα labels και το πλήθος των folds. Τέλος, οι συναρτήσεις αυτές επιστρέφουν τον μέσο όρο των accuracies από τα k-folds που έχουν κάνει πάνω στο σύνολο δεδομένων, το cross validation δηλαδή accuracy.

Παρακάτω παρατίθενται τα αποτελέσματα από τις κλήσεις των συναρτήσεων που παρουσιάσαμε προηγουμένως.

```
X=np.append(X_train,X_test,axis=0)
y=np.append(y_train,y_test,axis=0)
```

```
print(f"SVM Linear classifier accuracy is {lib.evaluate_linear_svm_classifier(X,y)}")
print(f"SVM Rbf classifier accuracy is {lib.evaluate_rbf_svm_classifier(X,y)}")
print(f"SVM Poly classifier accuracy is {lib.evaluate_poly_svm_classifier(X,y)}")
print(f"SVM Sigmoid classifier accuracy is {lib.evaluate_sigmoid_svm_classifier(X,y)}")

```

```
SVM Linear classifier accuracy is 1.0
SVM Rbf classifier accuracy is 0.9810663254690175
SVM Poly classifier accuracy is 0.9967618314115085
SVM Sigmoid classifier accuracy is 0.8654720608193092
```

```
print(f"5NN classifier accuracy is {lib.evaluate_knn_classifier(X,y)}")
```

```
5NN classifier accuracy is 0.9367210864495051
```

```
print(f"Sklearn Naive Bayes classifier accuracy is {lib.evaluate_sklearn_nb_classifier(X,y)}")
```

```
Sklearn Naive Bayes classifier accuracy is 0.6556997916168482
```

```
print(f"Custom Naive Bayes classifier accuracy is {lib.evaluate_custom_nb_classifier(X,y)}")
```

```
Custom Naive Bayes classifier accuracy is 0.6561979236218296
```

```
print(f"Euclidean distance classifier accuracy is {lib.evaluate_euclidean_classifier(X,y)}")
```

```
Euclidean distance classifier accuracy is 0.8146507951789269
```

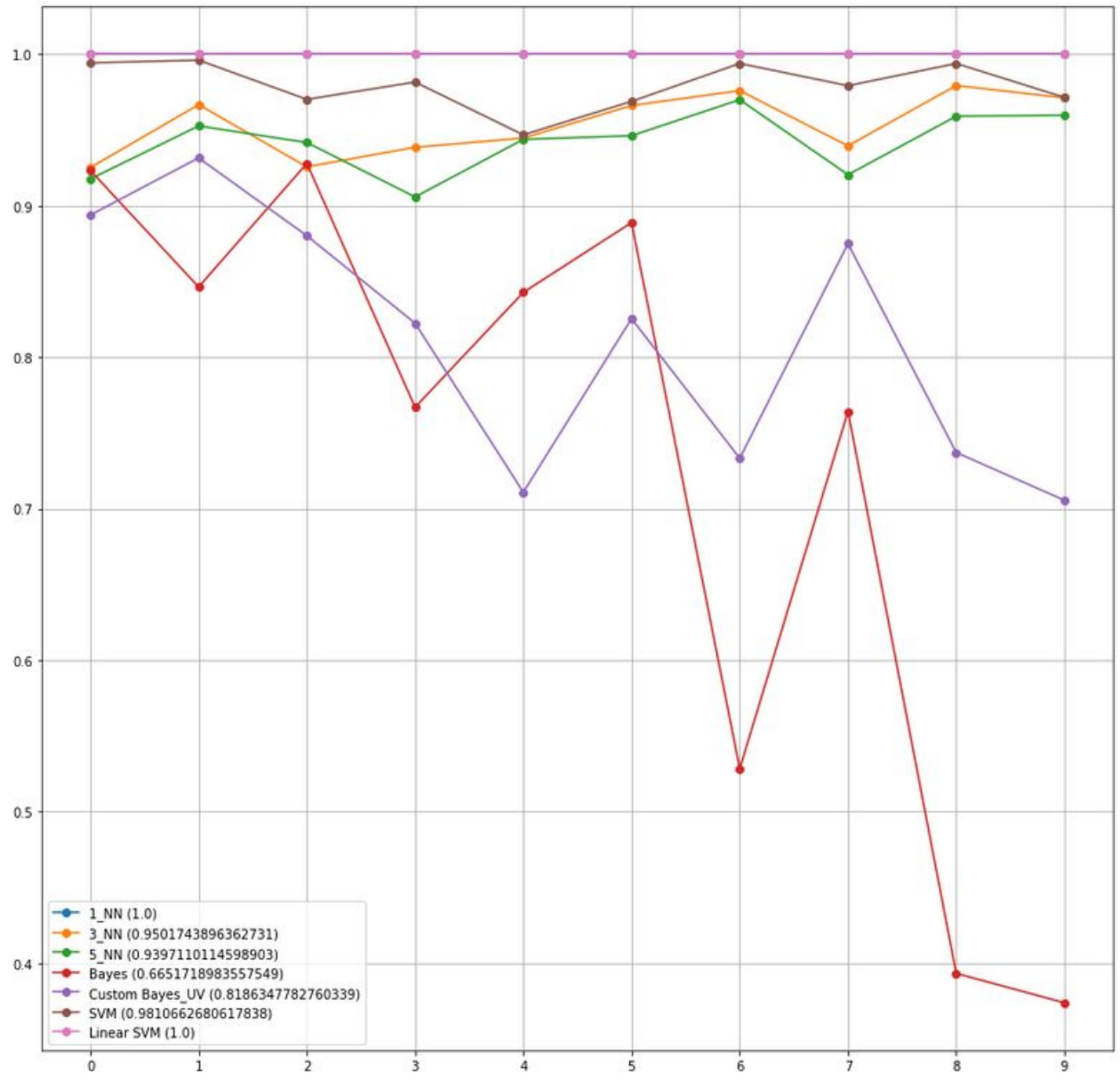
Αξιοσημείωτη φαίνεται η επίδοση του γραμμικού SVM ταξινομητή , ο οποίος όπως βλέπουμε πετυχαίνει 100% ακρίβεια στις προβλέψεις του. Επιπλέον , παρατηρούμε πως τόσο ο Rbf SVM Classifier , όσο και οι Polynomial SVM Classifier και 5 Nearest Neighbors πετυχαίνουν ιδιαίτερα υψηλά ποσοστά ακρίβειας στις προβλέψεις τους. Τέλος, για ακόμη μία φορά φαίνεται πως η δική μας υλοποίηση του Naive Bayes Classifier και αυτή του scikit-learn έχουν πολύ κοντινά ποσοστά ακρίβειας στις προβλέψεις τους.

Βήμα 18 :

Στο συγκεκριμένο βήμα θα επιχειρήσουμε να συνδυάσουμε ορισμένους από τους προμελετηθέντες ταξινομητές προκειμένου να λάβουμε προβλέψεις που θα χαρακτηρίζονται από μεγαλύτερη ακρίβεια. Η διαδικασία αυτή ονομάζεται ensembling και προκειμένου να επιφέρει προβλέψεις μεγαλύτερης ακρίβειας είναι αναγκαίο οι ταξινομητές που θα επιλεχθούν αφενός να έχουν σχετικά καλές επιδόσεις από μόνοι τους και αφετέρου να τείνουν να κάνουν λάθος προβλέψεις σε διαφορετικά ψηφία, δηλαδή ο καθένας να αδυνατεί να προβλέψει καλά μια συγκεκριμένη κλάση.

Συνεπώς, για να μπορέσουμε να επιλέξουμε τους καταλληλότερους προς συνδυασμό ταξινομητές πραγματοποιούμε παραθέτουμε αρχικά μια οπτική αναπαράσταση της ακρίβειας πρόβλεψης κάθε ταξινομητή ανά κλάση/ ψηφίο.

Στο παρακάτω γράφημα, ο οριζόντιος άξονας αντιστοιχεί στα ψηφία που υπάρχουν στο σύνολο των δεδομένων μας, ενώ ο κατακόρυφος άξονας αντιστοιχεί στην ακρίβεια πρόβλεψης του εκάστοτε ταξινομητή για το συγκεκριμένο ψηφίο .



Από το παραπάνω διάγραμμα παρατηρούμε πως ο RBF SVM Classifier, καθώς και οι 3_NN και 1_NN Classifier επιδεικνύουν ικανοποιητικά υψηλές επιδόσεις τόσο ως προς το γενικό accuracy , το οποίο φαίνεται στο legend, όσο και ως προς κάθε ψηφίο ανεξάρτητα. Επιπλέον, οι ταξινομητές αυτοί φαίνεται να κάνουν λάθος σε σχετικά διαφορετικές κλάσεις , επομένως επιλέγοντάς τους ικανοποιούμε και τις δύο προϋποθέσεις που θέσαμε παραπάνω.

(α) Έχοντας , λοιπόν, επιλέξει τους ταξινομητές που θα συνδυάσουμε στον voting classifier μας , καλούμε την συνάρτηση evaluate_voting_classifier που με ορίσματα το σύνολο δεδομένων , τα labels που αντιστοιχούν στα δείγματα του συνόλου αυτού και το πλήθος των folds για το cross validation , υπολογίζει και επιστρέφει το accuracy του Hard Voting Classifier και του Soft Voting Classifier.

Τα αποτελέσματα φαίνονται παρακάτω :

```
means  = lib.evaluate_voting_classifier(X,y)
print (f'The accuracy of Soft Voting Classifier when using 5-fold-cross-validation is :\n {means[1]}')
print('--------------------------------------------------------------------------------')
print (f'The accuracy of Hard Voting Classifier when using 5-fold-cross-validation is :\n{means[0]}')
```

```
The accuracy of Soft Voting Classifier when using 5-fold-cross-validation is :
 0.998007161423962
--------------------------------------------------------------------------------
The accuracy of Hard Voting Classifier when using 5-fold-cross-validation is :
0.9860488877432818
```

(β) Στην συνέχεια, επιλέγουμε τον Rbf SVM Classifier για να δοθεί στον Bagging Classifier και τα αποτελέσματα του ταξινομητή που προκύπτει είναι τα εξής :

```
mean = lib.evaluate_bagging_classifier(X,y)
print (f'The accuracy of Bagging Classifier when using 5-fold-cross-validation is :\n{mean}')
```

```
The accuracy of Bagging Classifier when using 5-fold-cross-validation is :
0.983807293720866
```

(γ) Αρχικά , αναφορικά με τους Voting Classifiers παρατηρούμε πως και οι δύο τους κατάφεραν να πραγματοποιήσουν προβλέψεις αρκετά μεγάλης ακρίβειας, τα οποία είναι μεγαλύτερα τόσο από αυτά του 3 Nearest Neighbors Classifier όσο και από του Rbf SVM Classifier. Το αποτέλεσμα αυτό είναι σχετικά αναμενόμενο, καθώς στα ψηφία που οι παραπάνω δύο ταξινομητές κάνουν λάθος βρίσκεται ο 1 Nearest Neighbor Classifier να τους "διορθώσει" το λάθος τους.

Επιπλέον, συγκρίνοντας τους δύο αυτούς ταξινομητές μεταξύ τους παρατηρούμε πως ο Soft Voting Classifier πραγματοποιεί προβλέψεις ελαφρώς μεγαλύτερης ακριβείας. Το αποτέλεσμα αυτό είναι πιθανό να οφείλεται στον τρόπο με τον οποίο λαμβάνουν τις αποφάσεις του οι δύο ταξινομητές. Συγκεκριμένα, ο Hard Voting ταξινομητής λαμβάνει υπόψη του τις προβλέψεις και των τριών ταξινομητών ισοβαρώς και η τελική του απόφαση είναι στην ουσία η πλειοψηφούσα. Αντίθετα, στον Soft Voting Classifier κάθε ταξινομητής δίνει μια πιθανότητα κλάσης με βάση το συγκεκριμένο δείγμα , η οποία σταθμίζεται με βάση την σημαντικότητα του εκάστοτε ταξινομητή προκειμένου τελικά να αθροιστούν. Η απόφαση στην περίπτωση του Soft Voting ταξινομητή είναι η κλάση με το μεγαλύτερο σταθμισμένο άθροισμα πιθανοτήτων. Συνεπώς, είναι λογικό ο Soft Voting Classifier να τα πηγαίνει καλύτερα καθώς δίνει περισσότερη έμφαση στις αποφάσεις του 1_NN ταξινομητή.

Στην συνέχεια, ο Bagging Classifier πετυχαίνει να βελτιώσει ελάχιστα την ακρίβεια των προβλέψεων του Rbf SVM Classifier. Αυτό συμβαίνει διότι ο συγκεκριμένος ταξινομητής χωρίζει τυχαία το training set και εκπαιδεύει τον ταξινομητή, ο οποίος στην συνέχεια προβλέπει την κλάση των δεδομένων του test set. Η διαδικασία αυτή επαναλαμβάνεται ορισμένες φορές και η τελική απόφαση του Bagging Classifier προκύπτει ως ο μέσος όρος των επιμέρους προβλέψεων για το κάθε δείγμα.

Βήμα 19 :

Στο σημείο αυτό της εργαστηριακής άσκησης θα επιχειρήσουμε να ταξινομήσουμε τα δείγματα του test set σε μία από τις 10 κλάσεις χρησιμοποιώντας ως ταξινομητή ένα νευρωνικό δίκτυο.

(α) Για να μπορέσουμε να εκπαιδεύσουμε οποιοδήποτε νευρωνικό δίκτυο είναι αναγκαία η ύπαρξη ενός dataloader μέσω του οποίου πραγματοποιείται η διοχέτευση των δεδομένων στο νευρωνικό δίκτυο σε mini batches συγκεκριμένου μεγέθους.

Κατασκευάζουμε , λοιπόν, την κλάση DigitsDataset μέσω της οποίας θα υλοποιηθούν οι μέθοδοι __len__ και __getitem__ οι οποίες χρησιμοποιούνται στην συνέχεια από το Dataloader της βιβλιοθήκης pytorch.

(β) Στην συνέχεια, ορίζουμε ορισμένα νευρωνικά δίκτυα τα οποία διαφέρουν τόσο ως προς το πλήθος των κρυφών layers που διαθέτουν όσο και ως προς τον αριθμό των νευρώνων που υπάρχουν ανά layer. Τα δίκτυα που ορίσαμε παρουσιάζονται παρακάτω :

```
The first Neural Network is : NN1Hidden(
  (linear1): Linear(in_features=256, out_features=64, bias=True)
  (output): Linear(in_features=64, out_features=10, bias=True)
  (logsoftmax): LogSoftmax(dim=1)
)

-----------------------------------------------------------------------------------------------------------

The second Neural Network is : Mynn(
  (hidden1): Linear(in_features=256, out_features=64, bias=True)
  (hidden2): Linear(in_features=64, out_features=64, bias=True)
  (output): Linear(in_features=64, out_features=10, bias=True)
  (logsoftmax): LogSoftmax(dim=1)
)

-----------------------------------------------------------------------------------------------------------

The third Neural Network is : NN3Hidden(
  (linear1): Linear(in_features=256, out_features=4, bias=True)
  (linear2): Linear(in_features=4, out_features=2, bias=True)
  (linear3): Linear(in_features=2, out_features=1, bias=True)
  (output): Linear(in_features=1, out_features=10, bias=True)
  (logsoftmax): LogSoftmax(dim=1)
)
```

(γ) Συνεχίζοντας , κατασκευάσαμε την κλάση PytorchNNModel η οποία πρόκειται να λειτουργήσει ως wrapper των νευρωνικών δικτύων μας ώστε αυτά να παραμένουν συμβατά με τις υλοποιήσεις του scikit-learn. Συγκεκριμένα, η παραπάνω κλάση κατά την αρχικοποίησή της δέχεται ως όρισμα το νευρωνικό δίκτυο το οποίο θα το "μετατρέψει" σε μορφή συμβατή με το scikit-learn και στην συνέχεια ορίζει της μεθόδους fit , predict και score κατά τρόπο όμοιο με αυτόν των προηγούμενων δύο ταξινομητών που υλοποιήσαμε. Η παράμετρος model που δίνεται κατά την αρχικοποίηση του PytorchNNModel έχει default τιμή το νευρωνικό δίκτυο 2 layers ,ώστε να μην υπάρχουν προβλήματα κατά τον ορισμό instances της κλάσης χωρίς ορίσματα. Στην μέθοδο fit γίνεται ο διαχωρισμός του συνόλου δεδομένων εκπαίδευσης σε train set και validation set κατά αναλογία 80/20. Έπειτα το νευρωνικό δίκτυο εκπαιδεύεται πάνω στα δεδομένα του νέου train set και ανά 10 εποχές υπολογίζεται κάποιο accuracy με βάση τα δεδομένα του validation set. Η διαδικασία αυτή επαναλαμβάνεται για 150 εποχές , όπου και ολοκληρώνεται η διαδικασία της εκπαίδευσης του νευρωνικού δικτύου μας.

Παρακάτω παρατίθενται στιγμιότυπα που απεικονίζουν το accuracy του εκάστοτε δικτύου κατά την διαδικασία της εκπαίδευσης.

```
Evaluation of Neural Network NN1Hidden(
  (linear1): Linear(in_features=256, out_features=64, bias=True)
  (output): Linear(in_features=64, out_features=10, bias=True)
  (logsoftmax): LogSoftmax(dim=1)
) on Train and Validation data ...

Epoch 0 (on Val set) -loss: 2.1898243610675516 -accuracy:0.3192019950124688
Epoch 10 (on Val set) -loss: 0.38868478055183703 -accuracy:0.8753117206982544
Epoch 20 (on Val set) -loss: 0.26012963686998075 -accuracy:0.885286783042394
Epoch 30 (on Val set) -loss: 0.2088789212015959 -accuracy:0.8827930174563591
Epoch 40 (on Val set) -loss: 0.2087514030818756 -accuracy:0.8778054862842892
Epoch 50 (on Val set) -loss: 0.15002429026823777 -accuracy:0.885286783042394
Epoch 60 (on Val set) -loss: 0.11326232065375035 -accuracy:0.885286783042394
Epoch 70 (on Val set) -loss: 0.09704640736946693 -accuracy:0.885286783042394
Epoch 80 (on Val set) -loss: 0.07770821803177778 -accuracy:0.8877805486284289
Epoch 90 (on Val set) -loss: 0.0664223087951541 -accuracy:0.8902743142144638
Epoch 100 (on Val set) -loss: 0.05587168517880715 -accuracy:0.8877805486284289
Epoch 110 (on Val set) -loss: 0.047893622531913795 -accuracy:0.8827930174563591
Epoch 120 (on Val set) -loss: 0.041762713629465834 -accuracy:0.8877805486284289
Epoch 130 (on Val set) -loss: 0.03719274026270096 -accuracy:0.885286783042394
Epoch 140 (on Val set) -loss: 0.03205213280251393 -accuracy:0.8827930174563591

Training Time (in minutes) = 0.06005187034606933
--------------------------------------------------------------------------------------------------------------------

Evaluation of Neural Network Mynn(
  (hidden1): Linear(in_features=256, out_features=64, bias=True)
  (hidden2): Linear(in_features=64, out_features=64, bias=True)
  (output): Linear(in_features=64, out_features=10, bias=True)
  (logsoftmax): LogSoftmax(dim=1)
) on Train and Validation data ...

Epoch 0 (on Val set) -loss: 2.269939688535837 -accuracy:0.3940149625935162
Epoch 10 (on Val set) -loss: 0.4956636761243527 -accuracy:0.8703241895261845
Epoch 20 (on Val set) -loss: 0.3130024465230795 -accuracy:0.9052369077306733
Epoch 30 (on Val set) -loss: 0.22470357741873997 -accuracy:0.9002493765586035
Epoch 40 (on Val set) -loss: 0.18936004117131233 -accuracy:0.9027431421446384
Epoch 50 (on Val set) -loss: 0.13864000757726339 -accuracy:0.9002493765586035
Epoch 60 (on Val set) -loss: 0.10789780848874496 -accuracy:0.9152119700748129
Epoch 70 (on Val set) -loss: 0.08680808601471093 -accuracy:0.9077306733167082
Epoch 80 (on Val set) -loss: 0.06539816287561105 -accuracy:0.9077306733167082
Epoch 90 (on Val set) -loss: 0.054442789405584335 -accuracy:0.9077306733167082
Epoch 100 (on Val set) -loss: 0.04193269941382683 -accuracy:0.912718204488778
Epoch 110 (on Val set) -loss: 0.034456385765224695 -accuracy:0.9152119700748129
Epoch 120 (on Val set) -loss: 0.025633802243436758 -accuracy:0.912718204488778
Epoch 130 (on Val set) -loss: 0.020454589696153283 -accuracy:0.9152119700748129
Epoch 140 (on Val set) -loss: 0.017233307112921745 -accuracy:0.9152119700748129

Training Time (in minutes) = 0.06759688456853231
--------------------------------------------------------------------------------------------------------------------
Evaluation of Neural Network NN3Hidden(
  (linear1): Linear(in_features=256, out_features=4, bias=True)
  (linear2): Linear(in_features=4, out_features=2, bias=True)
  (linear3): Linear(in_features=2, out_features=1, bias=True)
  (output): Linear(in_features=1, out_features=10, bias=True)
  (logsoftmax): LogSoftmax(dim=1)
) on Train and Validation data ...

Epoch 0 (on Val set) -loss: 2.3361038244687595 -accuracy:0.09226932668329177
Epoch 10 (on Val set) -loss: 2.274600808437054 -accuracy:0.14713216957605985
Epoch 20 (on Val set) -loss: 2.262384433012742 -accuracy:0.14713216957605985
Epoch 30 (on Val set) -loss: 2.2549948508922872 -accuracy:0.14713216957605985
Epoch 40 (on Val set) -loss: 2.2526922959547777 -accuracy:0.14713216957605985
Epoch 50 (on Val set) -loss: 2.248238591047434 -accuracy:0.14713216957605985
Epoch 60 (on Val set) -loss: 2.261486832912152 -accuracy:0.14713216957605985
Epoch 70 (on Val set) -loss: 2.247564334135789 -accuracy:0.14713216957605985
Epoch 80 (on Val set) -loss: 2.2552351584801307 -accuracy:0.14713216957605985
Epoch 90 (on Val set) -loss: 2.2563984302374034 -accuracy:0.14713216957605985
Epoch 100 (on Val set) -loss: 2.253119679597707 8 -accuracy:0.14713216957605985
Epoch 110 (on Val set) -loss: 2.260801150248601 -accuracy:0.14713216957605985
Epoch 120 (on Val set) -loss: 2.246759818150447 -accuracy:0.14713216957605985
Epoch 130 (on Val set) -loss: 2.2481628197890062 -accuracy:0.14713216957605985
Epoch 140 (on Val set) -loss: 2.2594116009198704 -accuracy:0.14713216957605985

Training Time (in minutes) = 0.0666622519493103
```

(δ) Έχοντας πλέον ορίσει και εκπαιδεύσει τα δίκτυά μας, είμαστε σε θέση να προβούμε στην αξιολόγησή τους. Για την αξιολόγηση των μοντέλων μας ορίσθηκε η συνάρτηση evaluate_nn_classifier της οποίας παράμετροι είναι το σύνολο δεδομένων , τα labels που αντιστοιχούν στα δείγματα του συνόλου , το μοντέλο νευρωνικού δικτύου που αξιολογούμε και το πλήθος των folds που θα συμβούν κατά το cross validation. Στο σημείο αυτό , σημειώνουμε πως οι

τελευταίες δύο παράμετροι της παραπάνω συνάρτησης έχουν προεπιλεγμένες τιμές Mynn() και 5 αντίστοιχα.

Επομένως, καλώντας την evaluate_nn_classifier για καθένα από τα τρία νευρωνικά δίκτυα που ορίσαμε λαμβάνουμε τα εξής αποτελέσματα :

```
The 5-fold cross-validation accuracy of Neural Network NN1Hidden(
  (linear1): Linear(in_features=256, out_features=64, bias=True)
  (output): Linear(in_features=64, out_features=10, bias=True)
  (logsoftmax): LogSoftmax(dim=1)
) is : 0.9810666360251302

-----------------------------------------------------------------------------------------------------------------------
The 5-fold cross-validation accuracy of Neural Network Mynn(
  (hidden1): Linear(in_features=256, out_features=64, bias=True)
  (hidden2): Linear(in_features=64, out_features=64, bias=True)
  (output): Linear(in_features=64, out_features=10, bias=True)
  (logsoftmax): LogSoftmax(dim=1)
) is : 0.97932255289547

-----------------------------------------------------------------------------------------------------------------------
The 5-fold cross-validation accuracy of Neural Network NN3Hidden(
  (linear1): Linear(in_features=256, out_features=4, bias=True)
  (linear2): Linear(in_features=4, out_features=2, bias=True)
  (linear3): Linear(in_features=2, out_features=1, bias=True)
  (output): Linear(in_features=1, out_features=10, bias=True)
  (logsoftmax): LogSoftmax(dim=1)
) is : 0.1788651037412695
```

Παρατηρούμε πως τα πρώτα δύο νευρωνικά πετυχαίνουν εξαιρετικές επιδόσεις ως προς την ικανότητά αναγνώρισης των ψηφίων. Η ελάχιστη διαφορά στο accuracy των πρώτων δύο νευρωνικών δικτύων κατά πάσα πιθανότητα οφείλεται στον τρόπο με χωρισμού των δεδομένων σε train validation και test set και σε καμία περίπτωση δεν θα μπορούσε να αποδοθεί σε ελλιπές πλήθος δεδομένων για την εκπαίδευση ενός νευρωνικού με παραπάνω layers και συνεπώς παραπάνω παραμέτρους να βελτιστοποιήσουμε.

Αναφορικά με το νευρωνικό δίκτυο των τριών layers , κρίνεται αναγκαίο να αναφερθεί πως η επιλογή του πλήθους των νευρώνων ανά layer έγινε σκοπίμως έτσι προκείμενου να τονιστεί η σπουδαιότητα της σωστής επιλογής του πλήθους αυτών. Στο συγκεκριμένο νευρωνικό, από το πρώτο κιόλας layer έχουμε μεγάλη μείωση στις διαστάσεις των δεδομένων με τα οποία εργαζόμαστε , το οποίο συνεχίζεται μέχρι και το τελευταίο κρυφό layer όπου τελικά το νευρωνικό μας δίκτυο από μια βαθμωτή έξοδο καλείται να υπολογίσει τις τιμές των πιθανοτήτων για τις 10 κλάσεις , γεγονός αδύνατο όπως αποδεικνύεται και από το accuracy του συγκεκριμένου ταξινομητή.

Τέλος, αξίζει να σημειωθεί πως τα βελτιωμένα ποσοστά της μετρικής accuracy και των τριών ταξινομητών κατά το cross validation συγκριτικά με αυτά που είχαμε κατά το fit στο προηγούμενο βήμα , προκύπτουν διότι στην δεύτερη περίπτωση τα δίκτυα μας εκπαιδεύονται σε σύνολο δεδομένων πλήθους 3211 δειγμάτων ενώ στην πρώτη σε ένα σύνολο 1606 δειγμάτων.