ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧ/ΚΩΝ & ΜΗΧ/ΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΣΗΜΑΤΩΝ, ΕΛΕΓΧΟΥ ΚΑΙ ΡΟΜΠΟΤΙΚΗΣ

Μάθημα: "Αναγνώριση Προτύπων"

9ο εξάμηνο, Ακαδημαϊκό Έτος 2020-21

1η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ

Οπτική Αναγνώριση Ψηφίων

ΓΕΩΡΓΙΟΣ ΜΑΓΚΑΦΩΣΗΣ, 03116125

ΗΛΙΑΣ ΤΡΙΑΝΤΑΦΥΛΛΟΠΟΥΛΟΣ, 03116028

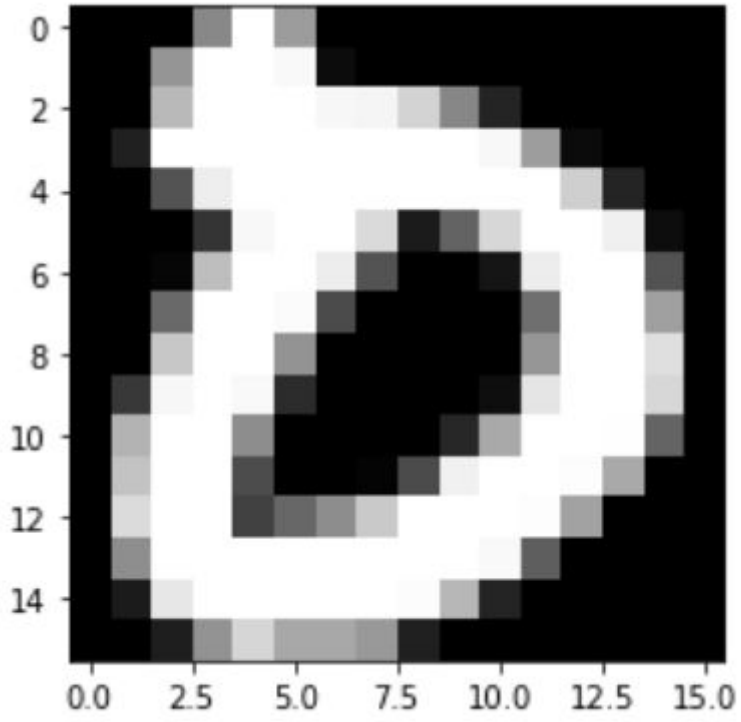
ΒΗΜΑ 1

Στο πρώτο βήμα της εργαστηριακής άσκησης, θα πρέπει να διαβάσουμε τα δεδομένα μας και να τα αποθηκεύσουμε κατάλληλα σε 4 πίνακες : X_train, y_train, X_test, y_test, αναλόγως με το τι αντιπροσωπεύει το καθένα από αυτά. Αφού δημιουργούμε σωστά τους πίνακες με βάση τα δεδομένα που μας δίνονται, παίρνουμε τα παρακάτω συγκεντρωτικά αποτελέσματα :

```
DATA OVERVIEW
-----------------------------------------
Train:
X_train shape: (n_samples) 7291 x (n_features) 256
y_train shape: (n_samples) 7291
Test:
X_test shape: (n_samples) 2007 x (n_features) 256
y_test shape: (n_samples) 2007
Unique labels: {0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0}
-----------------------------------------
Features:
Grayscale values of 16 x 16 pixels of the image of the digit
```

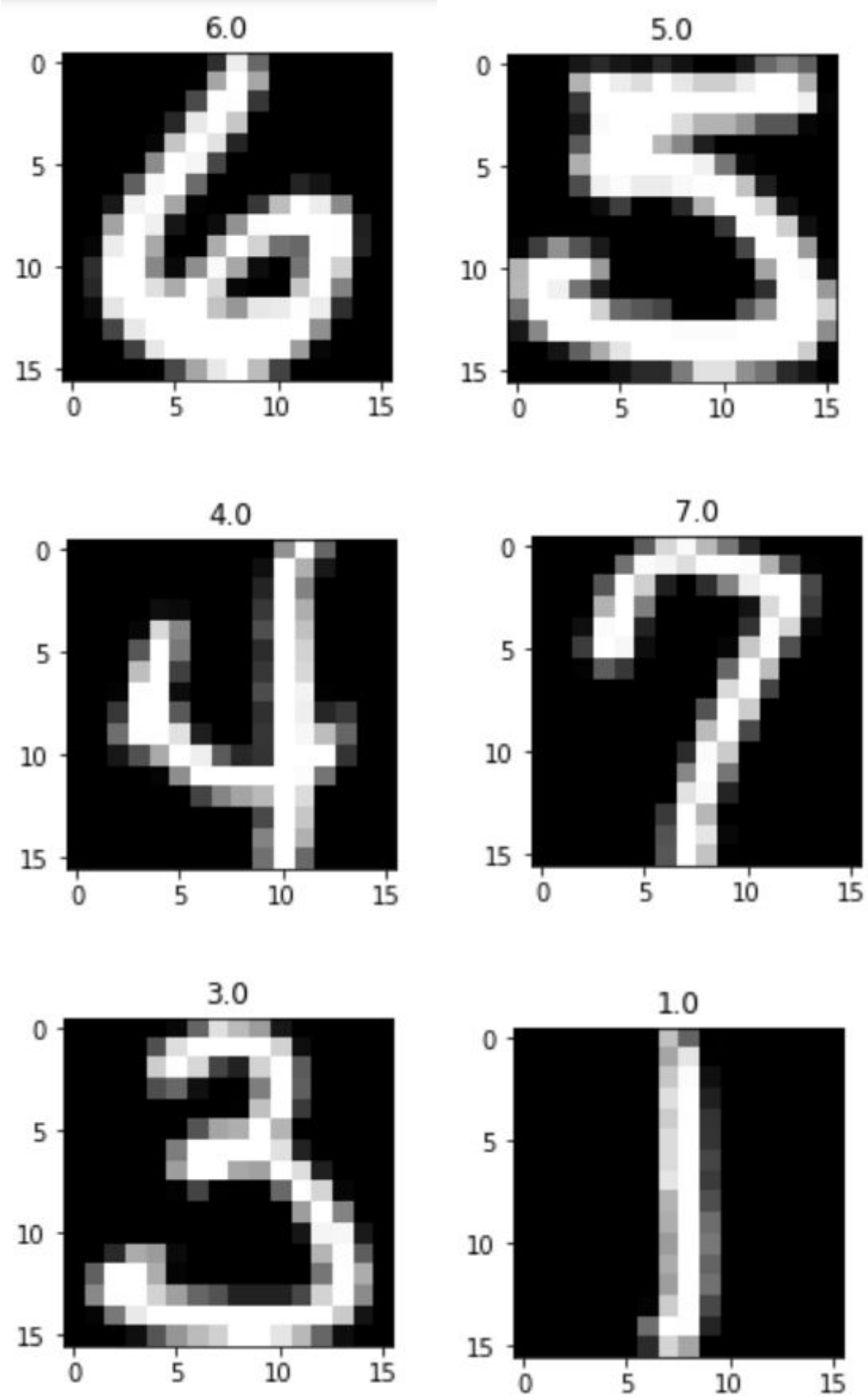
ΒΗΜΑ 2

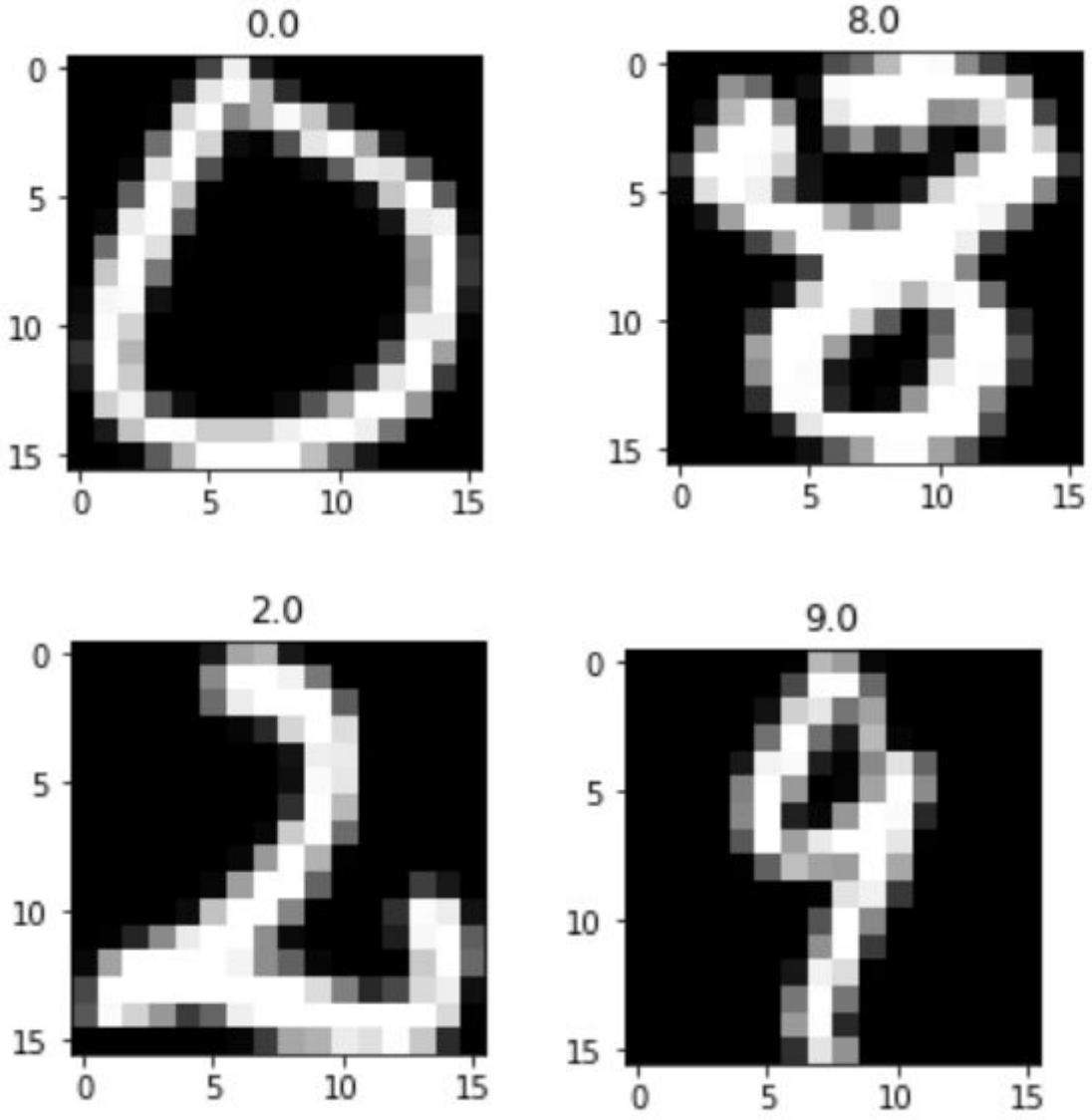
Συμπληρώνοντας την συνάρτηση show_sample, με το reshape των 256 pixels ενός συγκεκριμένου στοιχείου σε έναν πίνακα 16x16, μπορέσαμε να αναπαραστήσουμε οποιοδήποτε δεδομένο των δεδομένων μας. Στο συγκεκριμένο βήμα ζητήθηκε η αναπαράσταση του 131ου ψηφίου, που ήταν ο αριθμός 0 και φαίνεται παρακάτω :



ΒΗΜΑ 3

Για την υλοποίηση αυτού του βήματος, διατρέχουμε τον πίνακα με τα labels του train dataset μας και κάθε φορά που βρίσκουμε κάποιο ψηφίο που δεν έχουμε συναντήσει ξανά παλαιότερα, δημιουργούμε την αναπαράσταση αυτού σε ένα subplot, κάνοντας πάλι χρήση της show_sample. Έτσι, καταφέρνουμε να αναπαραστήσουμε και τα 10 δυνατά ψηφία (τα labels μας) σε τυχαία σειρά :





ΒΗΜΑ 4

Για την εύρεση του μέσου όρου ενός συγκεκριμένου pixel ενός συγκεκριμένου ψηφίου, βρίσκουμε όλα τα ψηφία που υπάρχουν στο dataset μας μέσω της y_train και στη συνέχεια παίρνουμε τον αριθμό που αναλογεί στο ζητούμενο ψηφίο μέσω της X_train (αφού μετατρέψουμε τα 256 pixels κατάλληλα ώστε να υπάρχει αντιστοιχία). Η αποθήκευση γίνεται σε έναν πίνακα και τέλος παίρνουμε τον μέσο όρο αυτού. Χρησιμοποιώντας την υλοποιημένη συνάρτηση, υπολογίζουμε τον μέσο όρο της τιμής του pixel (10,10) του ψηφίου 0 και προκύπτει :

```
the mean of the pixel ( 10 , 10 ) is -0.5041884422110553
```

ΒΗΜΑ 5

Ακολουθούμε παρόμοια διαδικασία και για την εύρεση της διασποράς και για το ζητούμενο pixel προκύπτει :

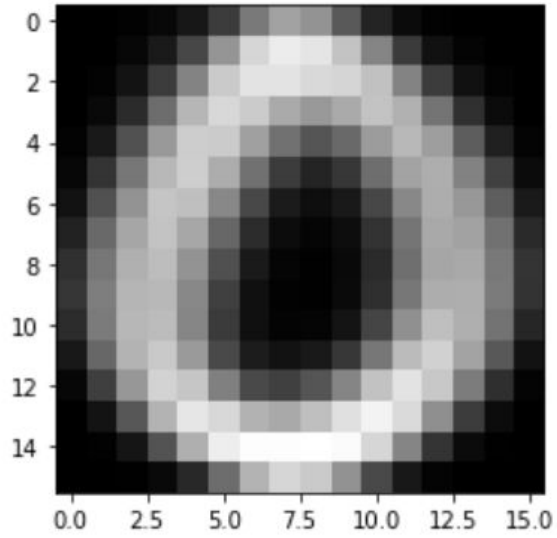
```
the variance of the pixel ( 10 , 10 ) is 0.5245221428814929
```

ΒΗΜΑ 6

Κάνοντας χρήση των συναρτήσεων που δημιουργήσαμε στα βήματα 4 και 5, μπορούμε να υπολογίσουμε την μέση τιμή και την διασπορά για κάθε χαρακτηριστικό/pixel ξεχωριστά σε έναν συγκεντρωτικό πίνακα. Έτσι, συμπληρώνουμε τις συναρτήσεις digit_mean και digit_variance, οι οποίες μας επιστρέφουν εν τελεί τον συγκεντρωτικό αυτό πίνακα που περιλαμβάνει τις μέσες τιμές και διασπορές όλων των pixels αντίστοιχα.

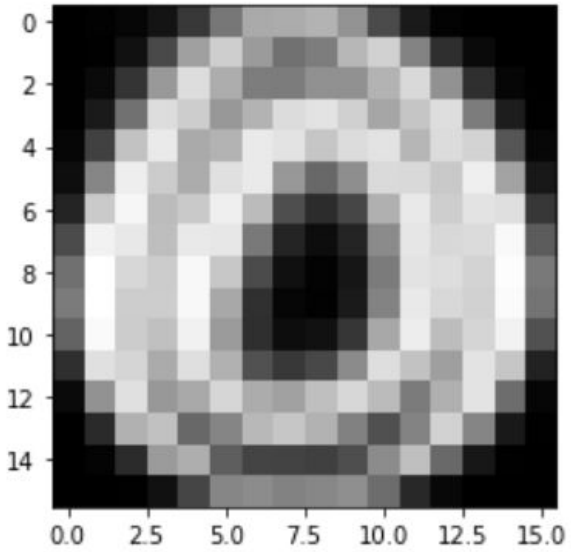
ΒΗΜΑ 7

Με βάση τον πίνακα που περιλαμβάνει τις μέσες τιμές των χαρακτηριστικών του ψηφίου 0, σχεδιάζουμε το 0 :



ΒΗΜΑ 8

Με βάση τον πίνακα που περιλαμβάνει τις διασπορές των χαρακτηριστικών του ψηφίου 0, σχεδιάζουμε το 0 :

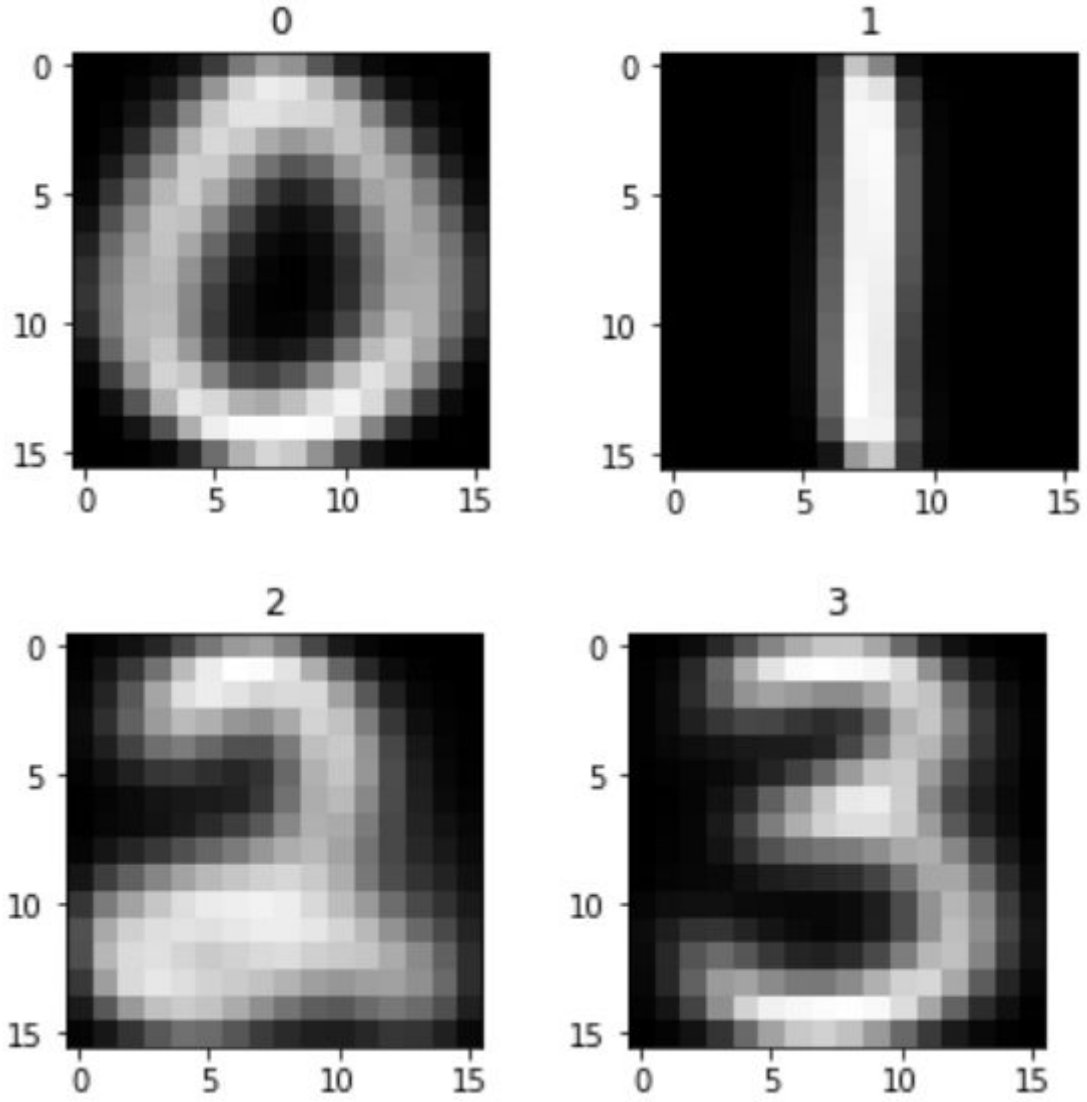


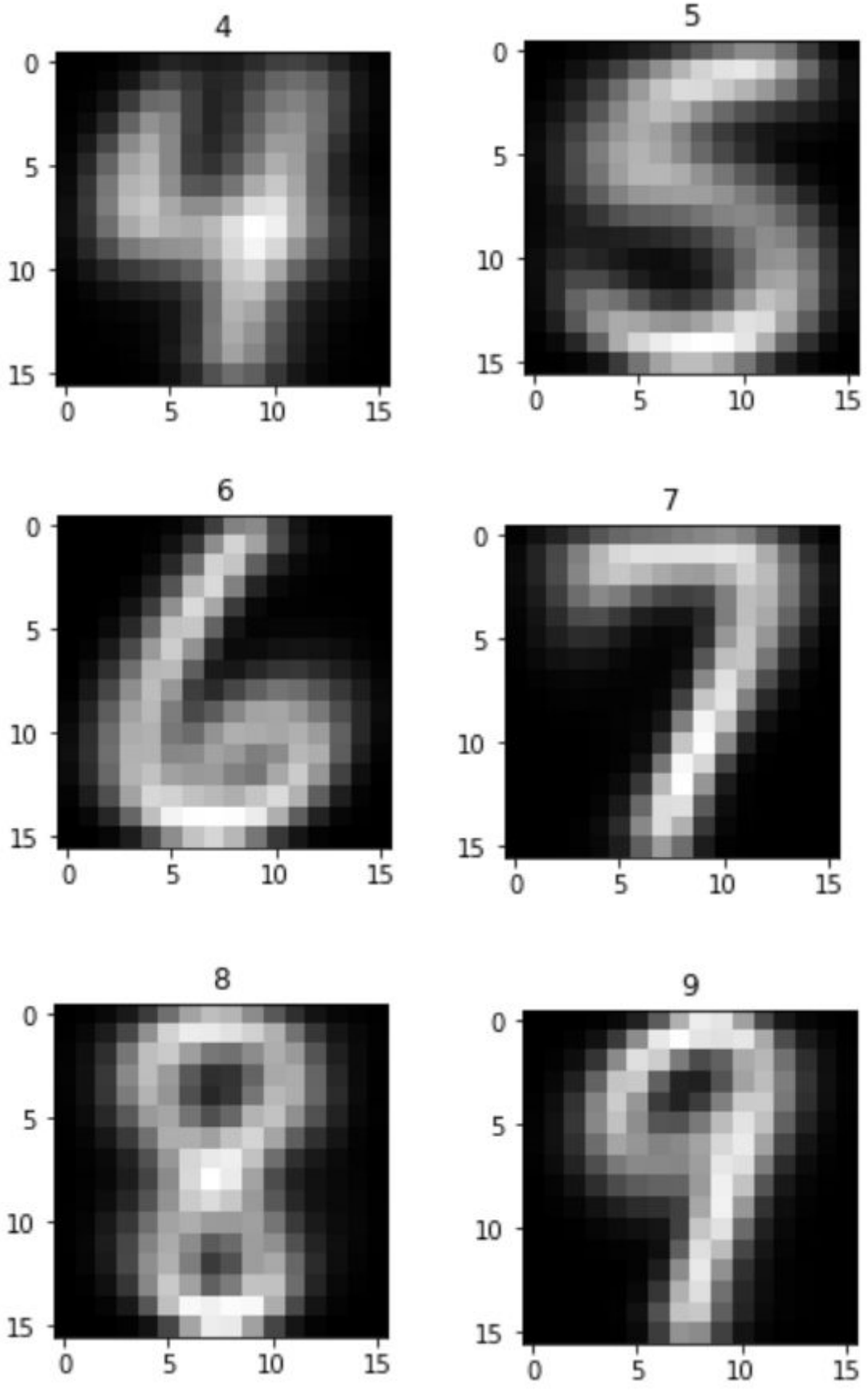
Παρατηρούμε ότι η εικόνα που μας δίνεται με βάση τις μέσες τιμές των pixels είναι πιο ευκρινής σε σχέση με εκείνη που παίρνουμε από τις διασπορές. Αυτό είναι λογικό να συμβαίνει, καθώς η διασπορά δίνεται από τον τύπο $\sigma^2 = \mathrm{E}[x - (E(x))^2]$ και δηλώνει την διακύμανση μεταξύ της μέσης τιμής και των pixel. Συνεπώς, θα μπορούσαμε να πούμε ότι η απεικόνιση με διασπορά δίνει περισσότερη “σημασία” σε πιο σπάνιες περιπτώσεις, όπου το 0 μπορεί να κάλυψε κάποια άλλα pixels από τα συνήθη. Ουσιαστικά, σε αυτή την εκδοχή, ο σχεδιασμός του 0 επεκτείνεται περισσότερο στο πλέγμα.

ΒΗΜΑ 9

α) Ο υπολογισμός των τιμών έγινε με τον ίδιο τρόπο που έγινε και με την περίπτωση του 0 στο βήμα 6. Έτσι, με ένα loop για όλα τα δυνατά ψηφία μπορούμε να επιτύχουμε εύκολα το ζητούμενο.

β) Με όμοιο τρόπο με το βήμα 7, σχεδιάζουμε όλα τα ψηφία και προκύπτουν :



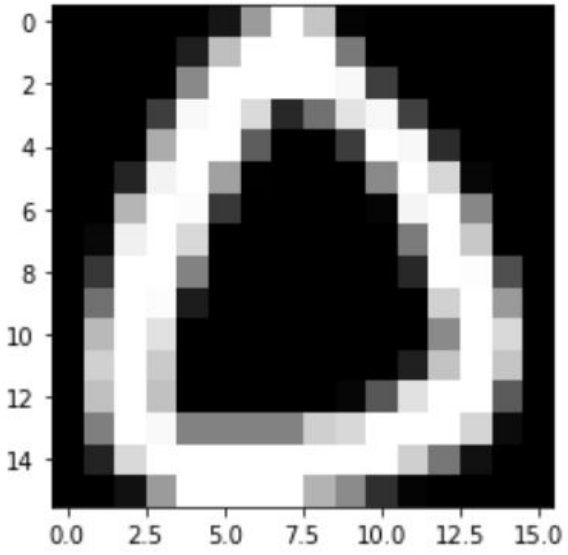


Η απεικόνιση των ψηφίων με βάση τις μέσες τιμές των pixels τους φαίνεται αρκετά ικανοποιητική από τις εικόνες που προέκυψαν.

ΒΗΜΑ 10

Για το βήμα αυτό συμπληρώνουμε τις συναρτήσεις euclidean_distance και euclidean_distance_classifier. Συγκεκριμένα, αφού διαθέτουμε τον πίνακα με τους μέσους όρους όλων των pixels όλων των ψηφίων από το βήμα 9, μπορούμε να βρούμε την ευκλείδεια απόσταση των pixels του ζητούμενου δεδομένου από τις μέσες τιμές του κάθε ψηφίου και να κατατάξουμε το δεδομένο στην κλάση του ψηφίου με την μικρότερη ευκλείδεια απόσταση. Για τον υπολογισμό της ευκλείδειας απόστασης 2 πινάκων, χρησιμοποιούμε την linalg.norm της numpy. Στο συγκεκριμένο βήμα περνάμε σαν δεδομένα τα X_test και επιλέγουμε να προβλεφθεί το 101ο ψηφίο και να απεικονιστεί. Τα αποτελέσματα φαίνονται παρακάτω :

```
The 101th digit is recognized as 0.0
The 101th digit is 0.0
```



Βλέπουμε ότι το 101ο ψηφίο ταξινομήθηκε σωστά ως 0. Για μια γενικότερη αξιολόγηση, ωστόσο θα πρέπει να κρίνουμε από το συνολικό test dataset.

ΒΗΜΑ 11

α) Έχοντας την συνάρτηση του προηγούμενου βήματος, μπορούμε να πάρουμε τον πίνακα των συνολικών προβλέψεων για όλο το X_test και το αποθηκεύουμε σε πίνακα test_predictions.

β) Για να υπολογίσουμε το ποσοστό επιτυχίας του συγκεκριμένου ταξινομητή, θα κάνουμε χρήση της μετρικής zero-one-loss. Ουσιαστικά, αυτή η μετρική απόδοσης υπολογίζει το ποσοστό διαφοροποιήσεων μεταξύ δύο πινάκων. Διαθέτοντας τόσο το y_test από τα δεδομένα της άσκησης όσο και το test_predictions, που περιλαμβάνει τις προβλέψεις των αντίστοιχων ψηφίων που το πραγματικό τους label βρίσκεται στο y_test, μπορούμε να χρησιμοποιήσουμε την έτοιμη zero_one_less με ορίσματα τους δύο αυτούς πίνακες. Τέλος, αφαιρούμε από το 1 για να πάρουμε το ποσοστό επιτυχίας και προκύπτει :

```
The success rate of our predictions is 81.41504733432984 %
```

Το ποσοστό επιτυχίας φαίνεται να είναι ικανοποιητικό για μία τέτοια αρχική μέθοδο ταξινόμησης, αλλά σίγουρα μπορούν να υπάρξουν σημαντικές βελτιώσεις.

ΒΗΜΑ 12

Για την υλοποίηση του ευκλείδειου ταξινομητή σαν ένα scikit-learn estimator, αρκεί να συμπληρώσουμε τις 3 συναρτήσεις της κλάσης :

➔ **fit :** αρχικοποιούμε έναν πίνακα X_mean_ όπου περιέχει τις μέσες τιμές των χαρακτηριστικών των δεδομένων για όλες τις κλάσεις ξεχωριστά.

➔ **predict :** για κάθε διάνυσμα του πίνακα προς ταξινόμηση υπολογίζουμε τις αποστάσεις του από όλες τις γραμμές του πίνακα X_mean_ και επιστρέφει το label της κλάσης με την μικρότερη.

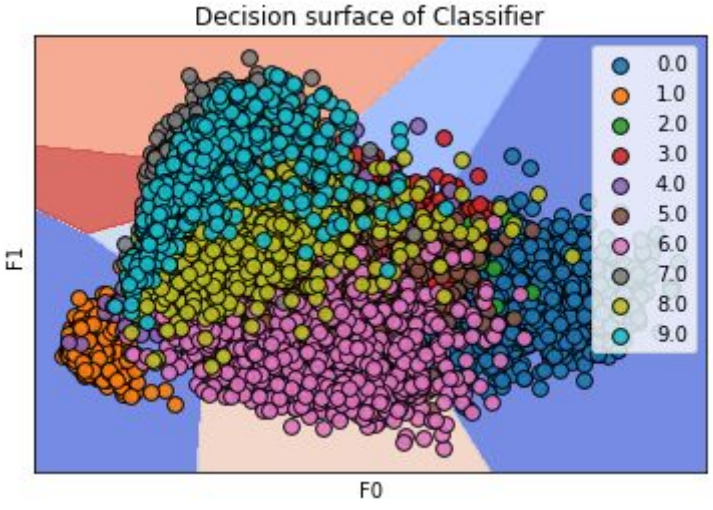
➔ **score :** εδώ εφαρμόζουμε τις τεχνικές του 12β). Τα αποτελέσματα του score είναι φυσικά τα ίδια με την περίπτωση όπου υπολογίζουμε τον πίνακα μέσων όρων εκτός της κλάσης.

ΒΗΜΑ 13

α) Για την εκτίμηση του εκτιμητή με χρήση του 5-fold validation χρησιμοποιούμε την συνάρτηση `cross_val_score.` Ως όρισμα δίνουμε την ένωση των δύο datasets train και test που μας δίνονται. Το αποτέλεσμα είναι το παρακάτω:

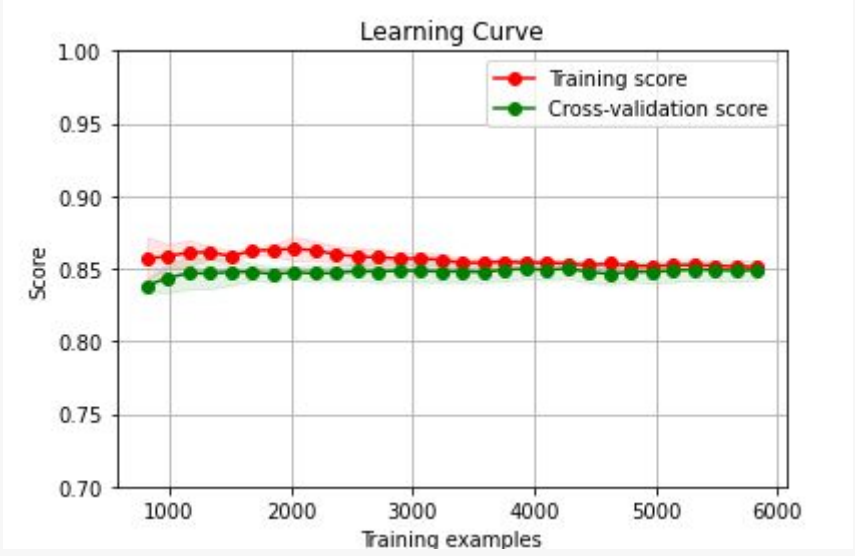
```
the score of the euclidean classifier with use of 5-fold-cross-validation is 84.07163638677287 %
```

β) Για την σχεδίαση των περιοχών απόφασης του ταξινομητή χρησιμοποιούμε την μέθοδο PCA ώστε να απεικονίσουμε τα δεδομένα μας σε έναν χώρο μικρότερων διαστάσεων και συγκεκριμένα 2 ώστε να είναι δυνατή η προβολή τους στο επίπεδο. Με χρήση της κλάσης PCA του scikit learn μαθαίνουμε από το train dataset τα principal components των δεδομένων μας και με βάση αυτά μετασχηματίζουμε και τα train και τα test. Προφανώς μετά από μία τόσο μεγάλη μείωση της διάστασης έχουμε μεγάλη απώλεια πληροφορίας. Τέτοια ώστε μετά από χρήση του ευκλείδειου ταξινομητή πάνω στα μετασχηματισμένα δεδομένα να έχουμε ποσοστό επιτυχίας μόλις 51.619332336821124 %. Παρόλα αυτά είναι δυνατή στον νέο χώρο μία κάποια εποπτεία των περιοχών απόφασης.



Με διαφορετικά χρώματα παριστάνονται οι περιοχές του επιπέδου που ταξινομεί ο ταξινομητής μας σε διαφορετικές κλάσεις και με κουκίδες οι προβλέψεις του στα train δεδομένα.

γ) Για την καμπύλη εκμάθησης χρησιμοποιούμε την συνάρτηση learning_curve. Οι καμπύλες που παίρνουμε είναι οι εξής:



Παρατηρούμε ότι πρόκειται για ένα αρκετά εύρωστο και σταθερό μοντέλο ως προς το πλήθος των training samples.

Βήμα 14

Για να υλοποιήσουμε τελικά έναν Bayesian Classifier υλοποιούμε αρχικά μία συνάρτηση η οποία από ένα dataset υπολογίζει για κάθε κλάση την a priori πιθανότητα εμφάνισής της. Ο τρόπος υπολογισμού αυτών των πιθανοτήτων προκύπτει πολύ απλά από την διαίρεση του πλήθους των labels που ανήκουν σε κάποια κατηγορία δια το συνολικό πλήθος των δεδομένων, για κάθε κατηγορία. Να σημειώσουμε εδώ ότι για τον υπολογισμό χρησιμοποιούμε μόνο το train dataset. Τα αποτελέσματα είναι τα ακόλουθα:

```
Class | a-priori probability
---------------------------
0     | 0.16376354409546015
1     | 0.13784117405019888
2     | 0.10026059525442327
3     | 0.09024825126868742
4     | 0.08942531888629818
5     | 0.07625840076807022
6     | 0.09107118365107666
7     | 0.08846523110684405
8     | 0.07433822520916199
9     | 0.08832807570977919
```

ΒΗΜΑ 15

α) Σε αυτό το βήμα υλοποιούμε έναν Naive Bayesian Classifier και ταξινομούμε με βάση αυτόν το test dataset. Στο εσωτερικό της κλάσης αρχικοποιούνται και διατηρούνται 3 βασικοί πίνακες που περιέχουν τις a priori πιθανότητες για τις κλάσεις του train set, τις μέσες τιμές του κάθε χαρακτηριστικού για κάθε κατηγορία ξεχωριστά και οι τυπικές αποκλίσεις του κάθε χαρακτηριστικού για κάθε κατηγορία. Ο Bayesian ταξινομητής κάνει τις εξής υποθέσεις:

-τα χαρακτηριστικά κάθε δείγματος δεδομένου της κλάσης του ακολουθούν κάποια κατανομή με άγνωστες παραμέτρους
-όλες οι κλάσεις είναι πιθανοτικά ανεξάρτητες
-όλα τα δείγματα που μας δίνονται είναι ανεξάρτητα και έχουν παραχθεί από την ίδια κατανομή
-οι άγνωστες παράμετροι του μοντέλου κάθε κλάσης είναι σταθερές
Για την περίπτωση του Naive μοντέλου έχουμε επιπλέον την υπόθεση ότι τα χαρακτηριστικά κάθε δείγματος σε κάθε κλάση είναι ανεξάρτητα μεταξύ τους. Στην δική μας περίπτωση κάθε χαρακτηριστικό ακολουθεί γκαουσιανή κατανομή με μέση τιμή και διασπορά που διατηρούμε στους πίνακες. Με την υπόθεση της ανεξαρτησίας των χαρακτηριστικών παίρνουμε ότι τα δείγματα κάθε κλάσης ακολουθούν μια κατανομή γινόμενο των επιμέρους κατανομών των χαρακτηριστικών τους.
Άρα στην fit μέθοδο της κλάσης βρίσκουμε με τις τιμές για τους 3 πίνακες. Αξίζει να επισημάνουμε ότι επειδή σε ορισμένα χαρακτηριστικά οι τιμές ήταν ίδιες προέκυπτε μηδενική τυπική απόκλιση. Για να αποφύγουμε την περίπτωση όπου θα είχαμε κάποια διαίρεση με το μηδέν προσθέτουμε σε όλες τις τυπικές αποκλίσεις μία μικρή ποσότητα της τάξης 10e-5.
Στην μέθοδο predict υπολογίζουμε για κάθε δείγμα-σειρά της εισόδου την την a posteriori πιθανότητα για κάθε κλάση. Για μεγαλύτερη αριθμητική ευστάθεια υπολογίζουμε με την χρήση της scipy.stats.norm.logpdf τον λογάριθμο της τιμής της συνάρτησης πυκνότητας μάζας πιθανότητας της κανονικής κατανομής με μέση τιμή και τυπική απόκλιση κάθε χαρακτηριστικού από τους πίνακες. Ο λογάριθμος του γινομένου κατανομών γίνεται άθροισμα λογαρίθμων και προσθέτουμε και τον λογάριθμο της a priori πιθανότητας της κλάσης. Αφού κάνουμε τον υπολογισμό για κάθε κλάση θέτουμε ως πρόβλεψη για το δείγμα την κλάση με το μεγαλύτερο αποτέλεσμα. Επιστρέφουμε έναν μονοδιάστατο πίνακα με όλες τις προβλέψεις.
Η μέθοδος score δεν διαφέρει από τις προηγούμενες περιπτώσεις.

β) Τα αποτελέσματα της υλοποίησης μας πάνω στο test set με την score και με 5-fold validation αντίστοιχα είναι:

```
("The success rate of our NB classifier's predictions is", 71.5495764823119, '%')
```

```
("The success rate of our NB classifier's predictions is", 73.17671658366447, '%')
```

γ) Τα αντίστοιχα αποτελέσματα της πρόβλεψης της υλοποίησης του scikit learn στην GaussianNB:

```
("The success rate of the library's NB classifier's predictions is", 71.94818136522171, '%')
```

```
("The success rate of the library's NB classifier's predictions is", 73.76829952512335, '%')
```

Παρατηρούμε ότι οι δύο υλοποιήσεις δίνουν πολύ κοντινά αποτελέσματα.

ΒΗΜΑ 16

Με χρήση κάποιου flag στον constructor του CustomNBClassifier μπορούμε να θέσουμε όλες τις διακυμάνσεις-τυπικές αποκλίσεις για όλες τις κλάσεις και όλα τα χαρακτηριστικά ίσες με τη μονάδα. Κάνοντας αυτό ουσιαστικά ουδετεροποιούμε οποιαδήποτε διαφορά στις εξαπλώσεις των κατανομών στον πολυδιάστατο χώρο των χαρακτηριστικών. Άρα, αν η μόνη παράμετρος των κατανομών που έχει νόημα είναι η μέση τιμή των δειγμάτων του train set των κλάσεων, τότε ο εκτιμητής μας περιμένουμε να συμπεριφερθεί όπως ένας ευκλείδειος ταξινομητής.
Τα αποτελέσματα:

```
("The success rate of our NB classifier's predictions with variance=1 everywhere is", 81.26557050323866, '%')
```

Πράγματι, είναι πολύ κοντά στα αποτελέσματα του ευκλείδειου ταξινομητή και αρκετά διαφορετικά από αυτά του bayesian ταξινομητή με τυπικές αποκλίσεις που προκύπτουν από το train set.

ΒΗΜΑ 17

Σε αυτό το βήμα συγκρίνουμε τις επιδόσεις διαφορετικών ταξινομητών. Συγκεκριμένα θα μελετήσουμε τις υλοποιήσεις του scikit learn για τον ταξινομητή κ κοντινότερων γειτόνων και για τα SVM. Πιο αναλυτικά, κάνοντας 5-fold validation θα έχουμε μία εκτίμηση για ταξινομητές
-> κ κοντινότερων γειτόνων με
- κ = 1
- κ = 3
- κ = 5
- κ = 7
- κ = 9
-> SVM με πυρήνα
- γραμμικό
- rbf
Έχουμε λοιπόν:

```
("The success rate of the K-Nearest Neighbors classifier's predictions with 1 neighbors is", 96.38620023483547,
'%')
("The success rate of the K-Nearest Neighbors classifier's predictions with 3 neighbors is", 96.12807787745751,
'%')
("The success rate of the K-Nearest Neighbors classifier's predictions with 5 neighbors is", 95.95601173020528,
'%')
("The success rate of the K-Nearest Neighbors classifier's predictions with 7 neighbors is", 95.54732860191916,
'%')
("The success rate of the K-Nearest Neighbors classifier's predictions with 9 neighbors is", 95.3859920063394,
'%')
```

```
("The success rate of the SVM classifier's predictions with kernel = linear is", 94.68683591016097, '%')
("The success rate of the SVM classifier's predictions with kernel = rbf is", 96.69800505532515, '%')
```

Παρατηρούμε ότι σε κάθε περίπτωση έχουμε αρκετά υψηλά ποσοστά επιτυχίας (πάνω από 94%). Ιδιαίτερα εντυπωσιακά είναι τα αποτελέσματα των KNN για κ = 1 και 3 και το SVM με rbf πυρήνα.

Συγκεκριμένα, για τον αλγόριθμο KNN παρατηρούμε ότι όσο αυξάνεται το K τόσο μειώνεται το ποσοστό επιτυχίας. Αυτό είναι λογικό να συμβαίνει, καθώς όσο αυξάνουμε το K, επηρεάζουμε το αποτέλεσμα από περισσότερους κοντινούς γείτονες και άρα δίνουμε την δυνατότητα και σε κάποια πιο απομακρυσμένα δείγματα να μπορέσουν να επηρεάσουν το αποτέλεσμα. Επομένως, είναι και πιο πιθανή η λανθασμένη ταξινόμηση κάποιων δειγμάτων.

Για τους ταξινομητές SVM, τα καλύτερα αποτελέσματα φαίνεται να πετυχαίνουμε όταν έχουμε πυρήνα = rbf. Αυτό είναι λογικό να συμβαίνει καθώς σε αυτή την περίπτωση είναι σαν να έχουμε έναν weighted nearest neighbor ταξινομητή, δηλαδή οι πιο κοντινοί γείτονες έχουν έναν σημαντικότερο λόγο στην απόφαση του ταξινομητή.

Τέλος και οι 2 αυτοί ταξινομητές πετυχαίνουν σημαντικά καλύτερες επιδόσεις από αυτές που είδαμε προηγουμένως ότι πετύχαινε ο Naive Bayes, καθώς δεν στηρίζονται στην ανεξαρτησία μεταξύ των features, κάτι που είναι λογικό να μην υπάρχει, διότι η αναπαράσταση ενός ψηφίου θα παρουσιάζει σίγουρα κάποιες εξαρτήσεις μεταξύ των pixels.

ΒΗΜΑ 18

α) Για να μπορέσουμε να επιλέξουμε κάποιους ταξινομητές, ώστε να τους συνδυάσουμε σε έναν voting classifier, θα πρέπει να βρούμε εκείνους που παρουσιάζουν σχετικά καλά αποτελέσματα σε διαφορετικά labels. Έτσι, μπορούμε να τυπώσουμε ένα report για τον κάθε ταξινομητή με τις πιο βασικές μετρικές απόδοσης. Από εκεί θα μπορέσουμε να εξάγουμε κάποιο συμπέρασμα σχετικά με ποιους ταξινομητές θα πρέπει να χρησιμοποιήσουμε.
Τα αποτελέσματα :

for Gaussian Naive Bayes :

	precision	recall	f1-score	support
9.0	0.93	0.82	0.87	359
6.0	0.91	0.96	0.94	264
3.0	0.78	0.71	0.74	198
0.0	0.92	0.47	0.62	166
2.0	0.76	0.28	0.41	200
4.0	0.80	0.46	0.59	160
1.0	0.66	0.89	0.76	170
8.0	0.85	0.90	0.87	147
7.0	0.43	0.66	0.52	166
5.0	0.45	0.86	0.59	177
accuracy			0.72	2007
macro avg	0.75	0.70	0.69	2007
weighted avg	0.77	0.72	0.72	2007

for KNN with 1 neighbors :

	precision	recall	f1-score	support
9.0	0.96	0.99	0.97	359
6.0	0.98	0.97	0.97	264
3.0	0.94	0.92	0.93	198
0.0	0.92	0.93	0.92	166
2.0	0.92	0.91	0.91	200
4.0	0.94	0.91	0.92	160
1.0	0.96	0.96	0.96	170
8.0	0.93	0.95	0.94	147
7.0	0.95	0.89	0.92	166
5.0	0.91	0.95	0.93	177
accuracy			0.94	2007
macro avg	0.94	0.94	0.94	2007
weighted avg	0.94	0.94	0.94	2007

for KNN with 3 neighbors :

	precision	recall	f1-score	support
9.0	0.93	0.99	0.96	359
6.0	0.98	0.98	0.98	264
3.0	0.93	0.92	0.93	198
0.0	0.94	0.92	0.93	166
2.0	0.93	0.92	0.92	200
4.0	0.94	0.90	0.92	160
1.0	0.98	0.96	0.97	170
8.0	0.93	0.94	0.93	147
7.0	0.96	0.91	0.93	166
5.0	0.91	0.95	0.93	177
accuracy			0.94	2007
macro avg	0.94	0.94	0.94	2007
weighted avg	0.95	0.94	0.94	2007

for KNN with 5 neighbors :

	precision	recall	f1-score	support
9.0	0.94	0.99	0.96	359
6.0	0.97	0.98	0.98	264
3.0	0.95	0.92	0.93	198
0.0	0.93	0.93	0.93	166
2.0	0.93	0.92	0.92	200
4.0	0.94	0.90	0.92	160
1.0	0.96	0.96	0.96	170
8.0	0.93	0.94	0.94	147
7.0	0.97	0.91	0.94	166
5.0	0.91	0.95	0.93	177
accuracy			0.94	2007
macro avg	0.94	0.94	0.94	2007
weighted avg	0.94	0.94	0.94	2007

for KNN with 7 neighbors :

	precision	recall	f1-score	support
9.0	0.94	0.99	0.96	359
6.0	0.96	0.98	0.97	264
3.0	0.95	0.92	0.94	198
0.0	0.94	0.93	0.94	166
2.0	0.93	0.91	0.92	200
4.0	0.94	0.91	0.92	160
1.0	0.96	0.95	0.96	170
8.0	0.93	0.94	0.93	147
7.0	0.97	0.87	0.92	166
5.0	0.90	0.95	0.93	177
accuracy			0.94	2007
macro avg	0.94	0.94	0.94	2007
weighted avg	0.94	0.94	0.94	2007

for KNN with 9 neighbors :

	precision	recall	f1-score	support
9.0	0.93	0.99	0.96	359
6.0	0.96	0.98	0.97	264
3.0	0.93	0.91	0.92	198
0.0	0.94	0.93	0.94	166
2.0	0.94	0.90	0.92	200
4.0	0.93	0.91	0.92	160
1.0	0.96	0.95	0.96	170
8.0	0.93	0.93	0.93	147
7.0	0.96	0.87	0.91	166
5.0	0.89	0.95	0.92	177
accuracy			0.94	2007
macro avg	0.94	0.93	0.93	2007
weighted avg	0.94	0.94	0.94	2007

```
for SVM with kernel = linear :
              precision    recall  f1-score   support

         9.0       0.95      0.98      0.97       359
         6.0       0.99      0.97      0.98       264
         3.0       0.91      0.91      0.91       198
         0.0       0.83      0.88      0.86       166
         2.0       0.89      0.90      0.89       200
         4.0       0.87      0.85      0.86       160
         1.0       0.94      0.94      0.94       170
         8.0       0.92      0.92      0.92       147
         7.0       0.94      0.87      0.90       166
         5.0       0.95      0.96      0.96       177

    accuracy                           0.93      2007
   macro avg       0.92      0.92      0.92      2007
weighted avg       0.93      0.93      0.93      2007

for SVM with kernel = rbf :
              precision    recall  f1-score   support

         9.0       0.97      0.99      0.98       359
         6.0       1.00      0.96      0.98       264
         3.0       0.92      0.93      0.93       198
         0.0       0.95      0.89      0.92       166
         2.0       0.89      0.94      0.91       200
         4.0       0.90      0.94      0.92       160
         1.0       0.97      0.94      0.95       170
         8.0       0.97      0.94      0.96       147
         7.0       0.92      0.93      0.93       166
         5.0       0.95      0.96      0.96       177

    accuracy                           0.95      2007
   macro avg       0.94      0.94      0.94      2007
weighted avg       0.95      0.95      0.95      2007
```

Λαμβάνοντας υπόψιν κυρίως την f1-score, παρατηρούμε ότι οι ταξινομητές πετυχαίνουν σχετικά παρόμοιες διακυμάνσεις αποδόσεων μεταξύ των διαφορετικών labels. Λόγω ελάχιστης περισσότερης διαφορετικότητας σε συνδυασμό με την καλή απόδοση των αντίστοιχων ταξινομητών επιλέγουμε τους ταξινομητές : Gaussian Naive Bayes, 3-Nearest Neighbors και SVM με kernel = rbf για να τους τοποθετήσουμε στον Voting Classifier. Είναι λογικό να επιλέξουμε μονό αριθμό ταξινομητών (γι' αυτό και επιλέξαμε 3), καθώς κατά το voting των ταξινομητών θα πρέπει να υπάρχει πλειοψηφία για ένα αποτέλεσμα, ώστε να αποφευχθούν οι ισοπαλίες.

Το score του Voting Classifier με τους συγκεκριμένους ταξινομητές για τα δύο είδη voting είναι :

```
The success rate of the voting classifier's predictions (hard voting) is 96.28936241591329 %
The success rate of the voting classifier's predictions (soft voting) is 96.11729048453614 %
```

Παρατηρούμε ότι το ποσοστό επιτυχίας αν και αρκετά υψηλό είναι μικρότερο από το αντίστοιχο του κάθε ταξινομητή ξεχωριστά εκ των 3-NN και SVM. Αυτό μπορεί να οφείλεται σε πιθανό overfitting ή αρκετά υψηλό bias.

β) Σε αυτό το βήμα, υλοποιούμε έναν Bagging Classifier. Πιο συγκεκριμένα, κατά την διαδικασία του bagging χωρίζουμε το dataset μας σε τυχαία υποσύνολα με πιθανές επικαλύψεις, εφαρμόζουμε έναν συγκεκριμένο ταξινομητή σε κάθε υποσύνολο ξεχωριστά και εν τέλει χρησιμοποιούμε τεχνική voting ή μέσου όρου για την τελική

απόφαση του αποτελέσματα από τα αποτελέσματα της κάθε “ψήφου”. Για την εφαρμογή του, θα χρησιμοποιήσουμε τον SVM με kernel = rbf, καθώς μέχρι στιγμής ήταν εκείνος που πέτυχε το καλύτερο score. Τα αποτελέσματα φαίνονται παρακάτω :

```
The success rate of the voting classifier's predictions (hard voting) is 97.13895781637717 %
```

γ) Όπως σχολιάσαμε και στο α ερώτημα, ο voting classifier πετυχαίνει ελάχιστα χαμηλότερο σκορ από τους επιμέρους ταξινομητές ξεχωριστά λόγω overfitting και bias. Στην περίπτωση του bagging αλγορίθμου, καταφέρνουμε να μειώσουμε το variance και προτιμάται για περιπτώσεις που το κύριο πρόβλημα είναι το overfitting, διότι το bagging μπορεί να το αντιμετωπίσει. Έτσι, παρατηρούμε ότι μπορούμε να πετύχουμε καλύτερο score από τον απλό SVM.

Ωστόσο, από την άλλη πλευρά, αυξάνεται το bias. Για την αντιμετώπιση του bias θα μπορούσαμε να χρησιμοποιήσουμε την τεχνική του Boosting. Σε αυτή την επιλογή όμως, δεν θα μπορούσαμε να αντιμετωπίσουμε επαρκώς φαινόμενα overfitting στο μοντέλο μας.