Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών

Ροή Σ: Αναγνώριση Προτύπων

(7ο Εξάμηνο)

Εργαστηριακή Άσκηση 1

Αλέξης Μάρας 03118074

Δημήτρης Μπακάλης 03118163

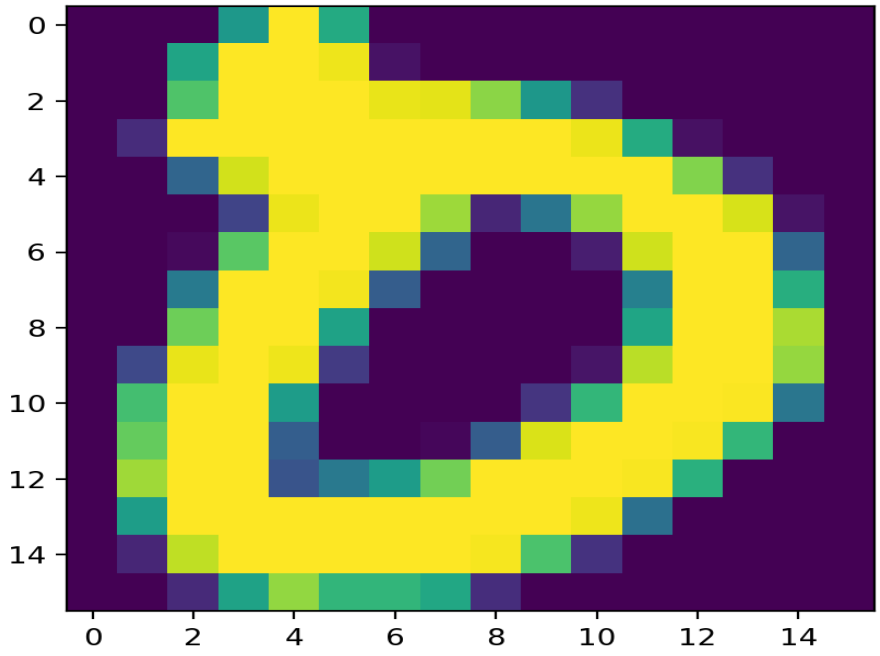
Βήμα 1

Φορτώνουμε τα δεδομένα των δύο .txt αρχείων (train, test) σε δύο πίνακες, με χρήση της np.loadtxt(). Από αυτούς απομονώνουμε το πρώτο στοιχείο κάθε γραμμής (label) στους μονοδιάστατους πίνακες y_train, y_test, αντίστοιχα.

Τα υπόλοιπα 256 στοιχεία κάθε γραμμής τα αποθηκεύουμε στους πίνακες x_train, x_test, οι οποίοι θα περιέχουν τα 256 (16*16) features κάθε ψηφίου.

Βήμα 2

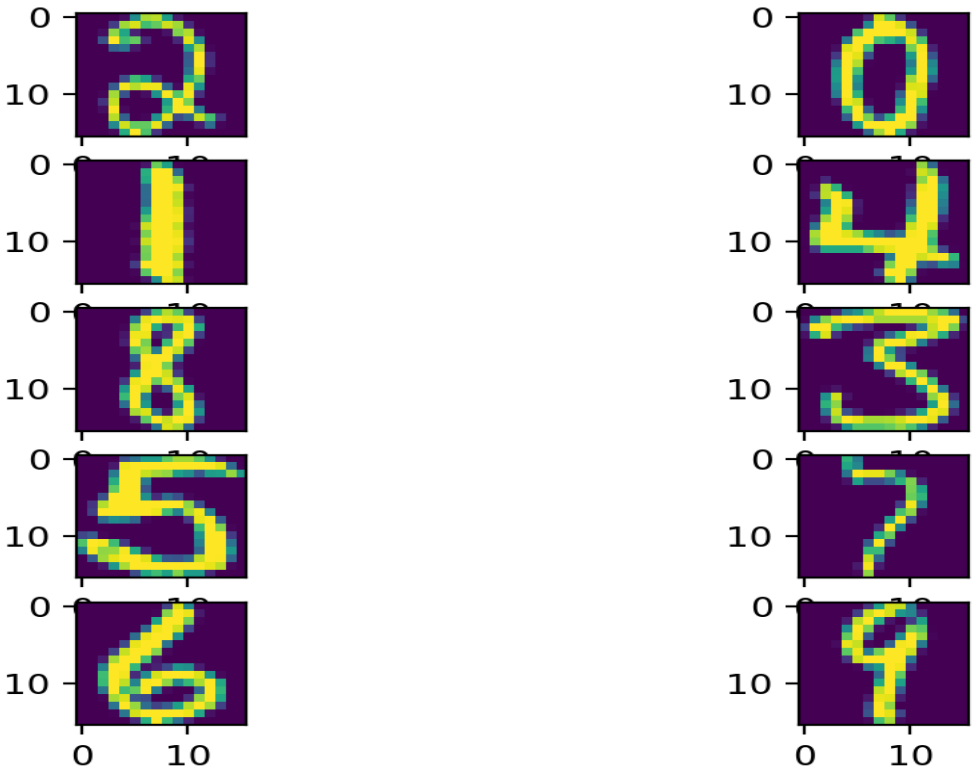
Στη συνέχεια, δημιουργούμε τη συνάρτηση show_sample() με ορίσματα τον πίνακα με τα features και ένα index, για να προσδιορίσουμε ποιο ψηφίο να κάνουμε plot.
Επομένως, με την κλήση show_sample(x_train, 130), έχουμε το παρακάτω αποτέλεσμα:



Βήμα 3

Σε αυτό το βήμα, παίρνουμε τυχαία indexes με χρήση της random.randint για τιμές από 0 έως 7291 (το πλήθος των samples του train data). Στην συνέχεια αποθηκεύουμε το label του κάθε ψηφίου που επιλέγεται τυχαία, ώστε να κάνουμε plot μόνο μια φορά το κάθε ψηφίο.

Το αποτέλεσμα είναι το παρακάτω:



Βήματα 4 , 5

Αφού εντοπίσουμε από τον πίνακα y_train την θέση όλων των μηδενικών, υπολογίζουμε τη μέση τιμή και στη συνέχεια τη διασπορά για το pixel (10,10) , που αντιστοιχεί στο feature υπ' αριθμόν 16*10+10=170 των 0 του πίνακα των train δεδομένων. Τα αποτελέσματα είναι τα ακόλουθα:

Μέση Τιμή : -0.5041884422110553

Διασπορά : 0.524522142881497

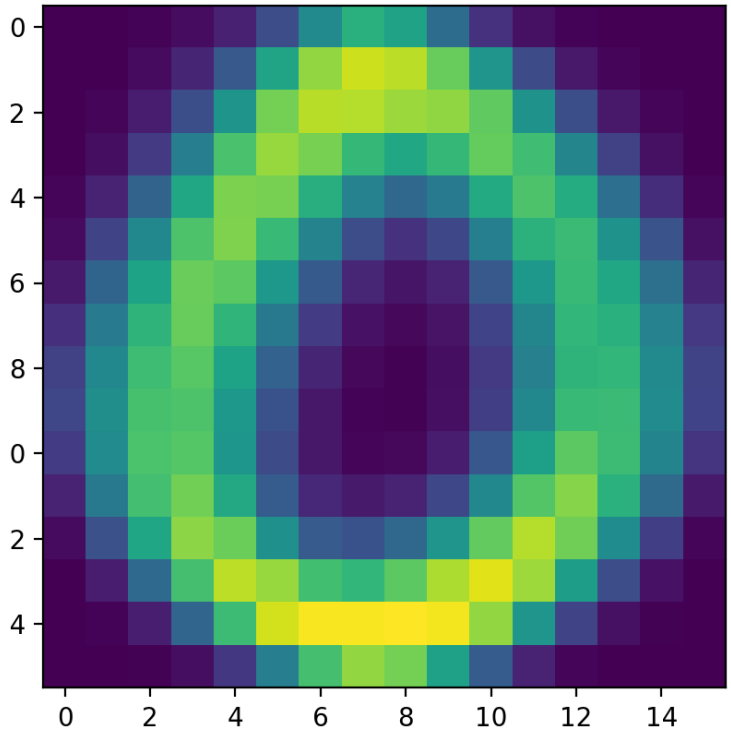
```
The mean of the pixel (10,10) of 0 is -0.5041884422110553
The variance of the pixel (10,10) of 0 is 0.524522142881497
```

Βήμα 6

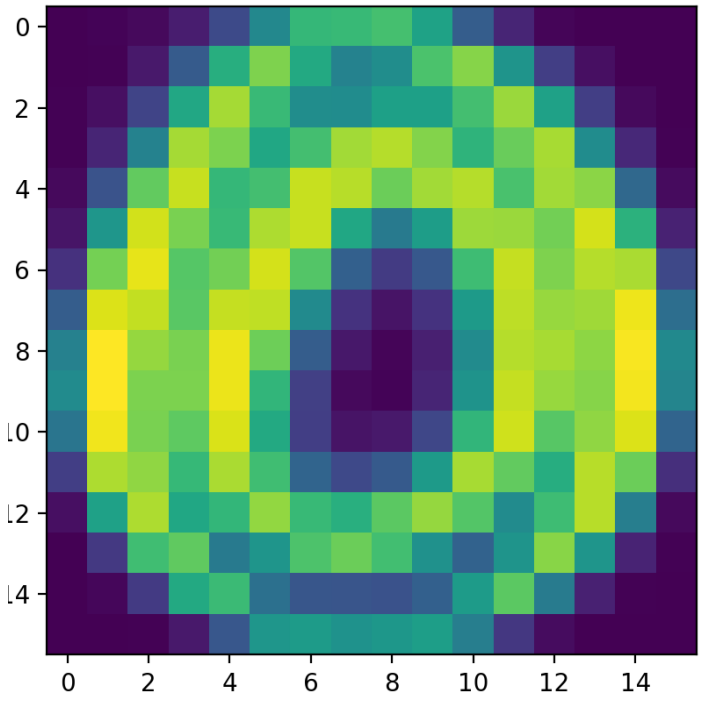
Επαναλαμβάνουμε την παραπάνω διαδικασία για όλα τα pixels του αριθμού 0, με τη χρήση μιας for, που εκτελεί 256 επαναλήψεις (όσα και τα features)

Βήματα 7,8

Το «μέσο» 0 είναι το παρακάτω:



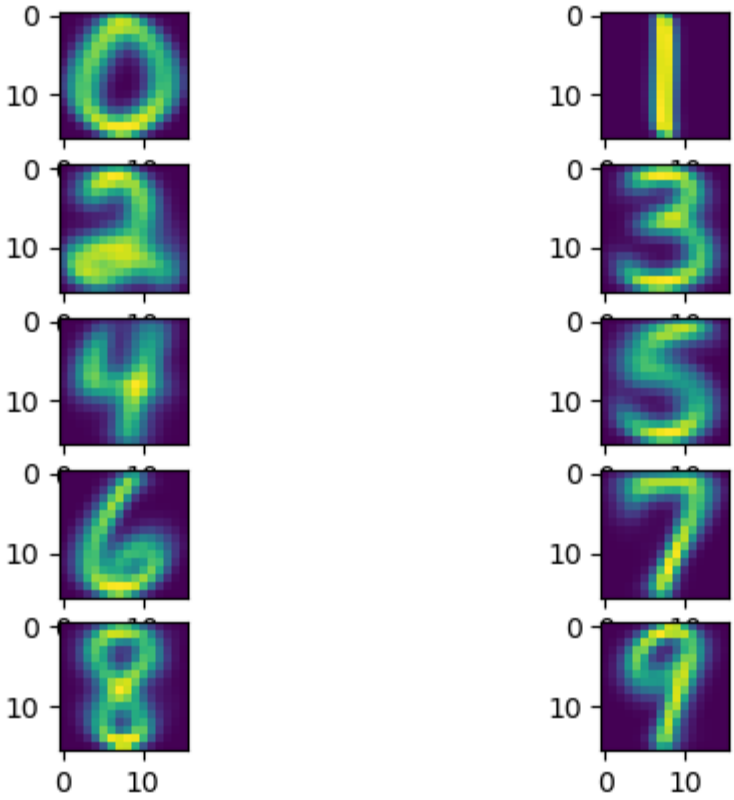
Το 0 σχεδιασμένο με βάση την διασπορά, που υπολογίσαμε στο βήμα 6 είναι το παρακάτω:



Από τα παραπάνω σχήματα μπορούμε να συμπεράνουμε, ότι κατά μία έννοια είναι συμπληρωματικά. Αυτό συμβαίνει διότι, τα pixels του μέσου 0, τα οποία είναι πιο έντονα (κίτρινο, τιμή κοντά στο 1), βρίσκονται σε περιοχές (πάνω και κάτω μέρος της εικόνας), όπου τα αντίστοιχα του 0 της διασποράς είναι λιγότερο έντονα (πράσινο, τιμή κοντά στο 0) και το αντίστροφο (δεξιά και αριστερά στην εικόνα).

Βήμα 9

Επεκτείνουμε το πρόγραμμα του βήματος 6, ώστε η ίδια διαδικασία να επαναληφθεί για όλα τα ψηφία (υλοποιείται με μία επιπλέον for) και λαμβάνουμε τους παρακάτω «μέσους» αριθμούς:



Βήμα 10

Για την ταξινόμηση του sample 101 σε μία από τις 10 κλάσεις που έχουμε, υπολογίζουμε την ευκλείδεια απόσταση του δείγματος αυτού από τον κάθε «μέσο» αριθμό (αθροίζοντας τα τετράγωνα της διαφοράς των αντίστοιχων pixel τους). Στο τέλος, επιλέγουμε την κλάση, η οποία θα μας δώσει την μικρότερη ευκλείδεια απόσταση.

Το αποτέλεσμα της ταξινόμησης είναι το 0. Ελέγχοντας τον πίνακα με τα labels των train δεδομένων, παρατηρούμε πως η ταξινόμηση είναι λανθασμένη, καθώς ο σωστός αριθμός είναι το 6.

```
Result = 0
Incorrect, the actual number is 6
```

Βήμα 11

Επεκτείνουμε το πρόγραμμα του προηγούμενου ερωτήματος ώστε να γίνει η ταξινόμηση όλων των test δεδομένων και στη συνέχεια, ανατρέχουμε τον πίνακα με τα labels(y_test), για να δούμε την ορθότητα των ταξινομήσεων.
Το score, που λαμβάνουμε από αυτή τη ταξινόμηση είναι 81.415 %

```
Success rate = 0.814150473343298 5
```

Βήμα 12

Υλοποιούμε τον ευκλείδειο ταξινομητή στην κλάση EuclideanDistanceClassifier(), η οποία αποτελείται από 4 συναρτήσεις:

- την __init__ (constructor),
- την fit(), στην οποία εκπαιδεύουμε το μοντέλο. Στον Ευκλείδειο ταξινομητή που υλοποιούμε, απλώς αποθηκεύουμε για την κάθε κλάση, την μέση τιμή του κάθε feature
- την predict(), που προβλέπει σε ποια κλάση ανήκει κάθε εικόνα
- την score(), που υπολογίζει το ποσοστό επιτυχίας των παραπάνω προβλέψεων.

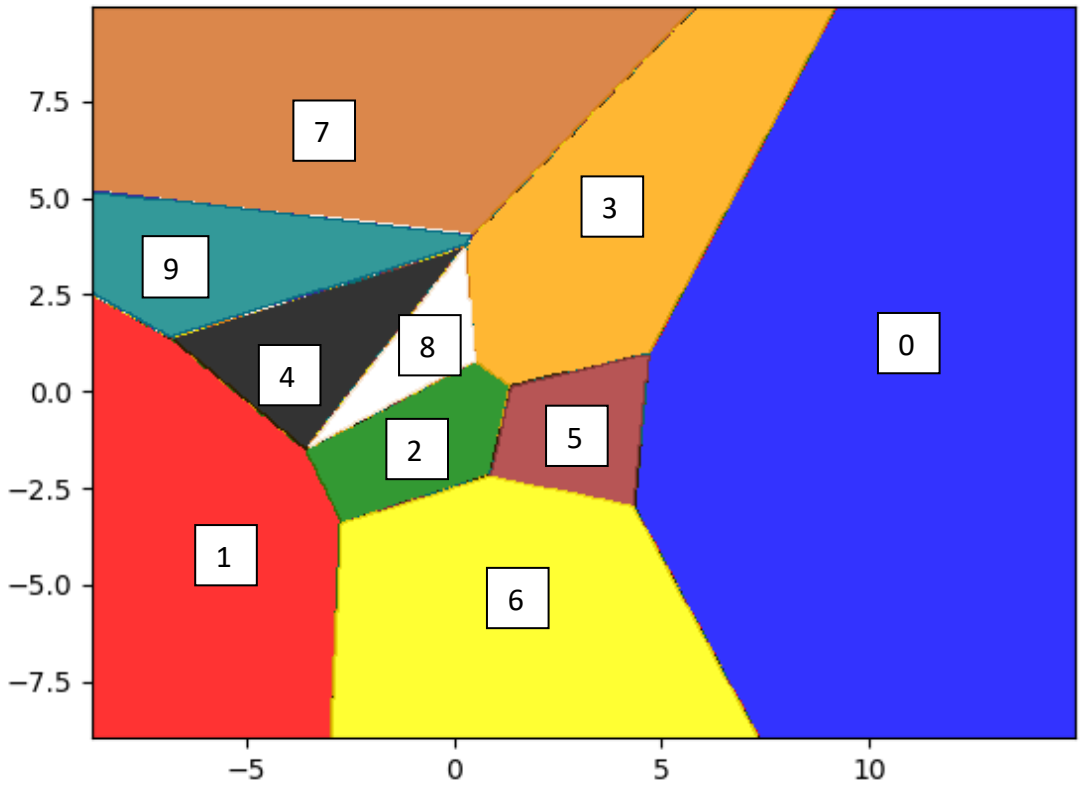
Βήμα 13

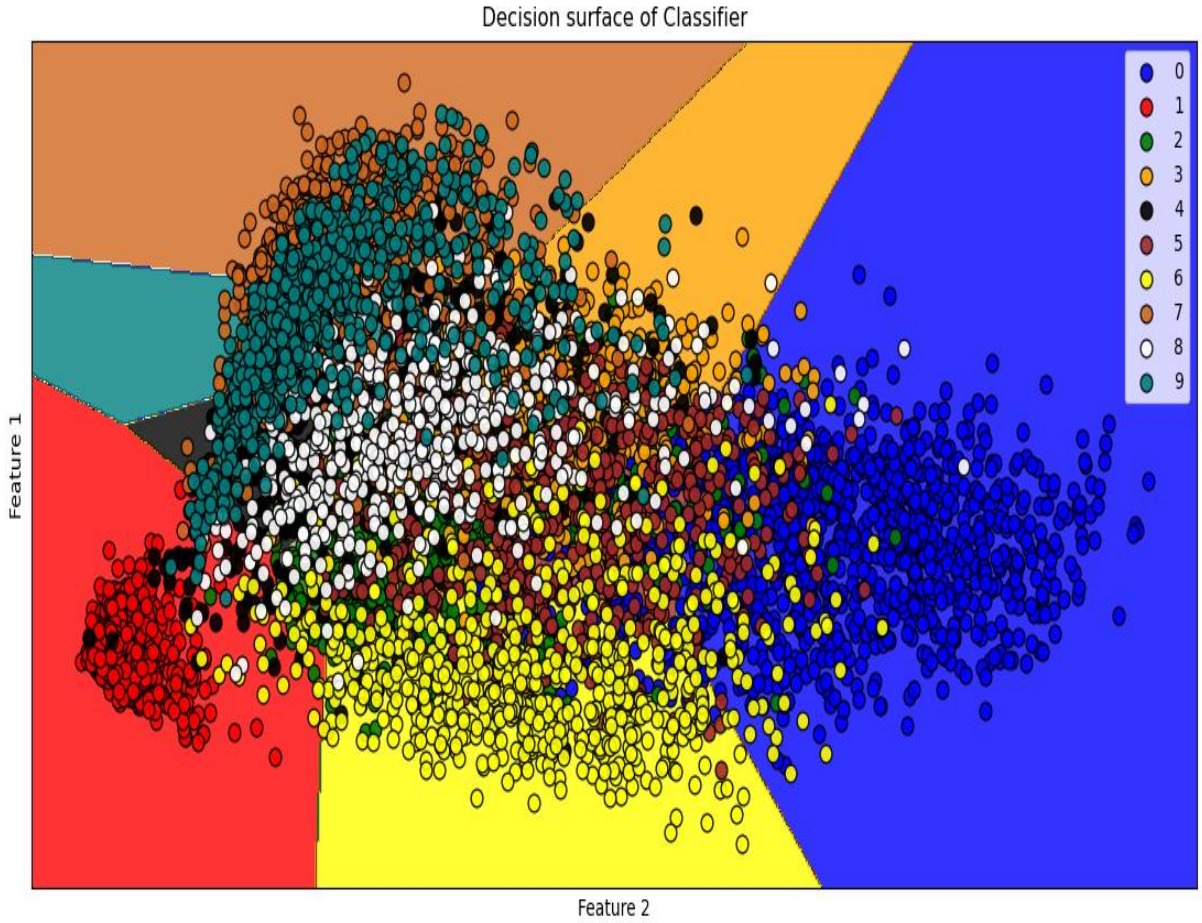
α) Ένας άλλος τρόπος εκτίμησης της επίδοσης του ταξινομητή, χωρίς την χρήση των test δεδομένων είναι με την χρήση της k-fold-cross-validation.
Χρησιμοποιούμε την συνάρτηση cross_val_score, με k =5, για να εκτιμήσουμε το ποσοστό επιτυχίας του ταξινομητή, και λαμβάνουμε ως έξοδο, 84.99% ποσοστό επιτυχίας.

```
CV score = 0.849951156954939 4
```

Όπως βλέπουμε το score του ταξινομητή στα test δεδομένα είναι αρκετά κοντά σε αυτό που περιμένουμε και από το 5-fold cross-validation score.

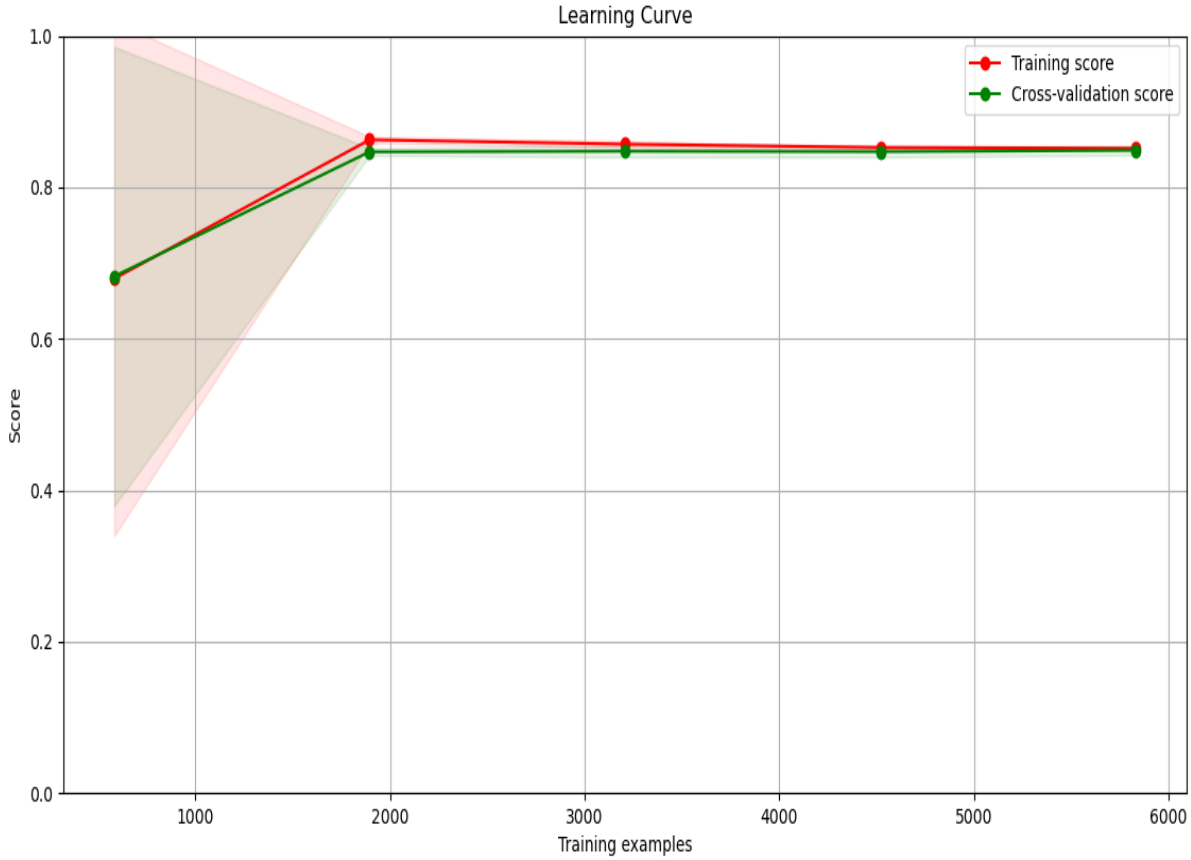
β) Προκειμένου να σχεδιάσουμε την περιοχή απόφασης του ταξινομητή στις 2 διαστάσεις απαιτούνται 2 features. Εφόσον, για κάθε ψηφίο έχουμε 256 features, πρέπει να εφαρμόσουμε την τεχνική PCA, για να εξασφαλίσουμε την μείωση των διαστάσεων σε 2.
Στη συνέχεια, εκπαιδεύουμε ξανά τον ταξινομητή με αυτά τα 2 features και με τη χρήση της συνάρτησης plot_clf παίρνουμε τις εξής περιοχές :





Παρατηρούμε από το παραπάνω διάγραμμα, πως παρότι πολλά από τα δείγματα έχουν τοποθετηθεί στις σωστές περιοχές απόφασης, ένας μεγάλος αριθμός των test δεδομένων έχουν ταξινομηθεί λανθασμένα. Αυτό οφείλεται στην αδυναμία μας να διατηρήσουμε αναλλοίωτο το περιεχόμενο των δεδομένων μας, καθώς μειώνουμε τις διαστάσεις τους από τις 256 στις 2.

γ) Η learning curve έτσι όπως προκύπτει ύστερα από την κλήση της συνάρτησης plot_learning_curve είναι η εξής:



Βήμα 14

Για τον υπολογισμό των a-priori πιθανοτήτων, μετράμε πόσα δεδομένα από τα labels των train δεδομένων ανήκουν σε κάθε κλάση και τα διαιρούμε το πλήθος με το συνολικό πλήθος των train δεδομένων. Οι πιθανότητες για κάθε ψηφίο βρίσκονται στην αντίστοιχη θέση του παρακάτω πίνακα:

0	1	2	3	4	5	6	7	8	9
0.163	0.137	0.1	0.09	0.089	0.076	0.091	0.088	0.074	0.088

```
[0.16376354409546015, 0.13784117405019888, 0.10026059525442327, 0.09024825126868742,
0.08942531888629818, 0.07625840076807022, 0.09107118365107666, 0.08846523110684405,
0.07433822520916199, 0.08832807570977919]
```

Βήμα 15

α) Με βάση τον Naive Bayes ταξινομητή, κάθε feature είναι στατιστικά ανεξάρτητο από τα υπόλοιπα, το οποίο, ειδικά στην περίπτωση της εικόνας, όπου τα features είναι τα pixels δεν ισχύει.
Κάθε pixel περιγράφεται από μια πιθανοτική κατανομή, κανονική στη συγκεκριμένη περίπτωση, με μέσες τιμές και διασπορές, αυτές που υπολογίσαμε στο βήμα 9. Προκειμένου να γίνει η ταξινόμηση, υπολογίζουμε 10 διαφορετικά γινόμενα, το καθένα εκ των οποίων είναι το γινόμενο των πιθανοτήτων του κάθε pixel επί την a-priori πιθανότητα του κάθε αριθμού. Τέλος, επιλέγουμε την κλάση με το μεγαλύτερο γινόμενο.

β) Το σκορ του Naive Bayes ταξινομητή που υλοποιήσαμε, υπολογίζεται με κλήση της συνάρτησης score (x_test, y_test) και ισούται με 0.7189 (71.89 %).
Για την εκτίμηση της επίδοσης του , υπολογίζουμε επιπλέον και την τιμή της 5-fold-cross-validation τεχνικής με την βοήθεια της συνάρτησης cross_ val_score. Η τιμή που λαμβάνουμε είναι ίση με 74.818 % (~ 3% απόκλιση από το αποτέλεσμα της πρόβλεψης για τα test δεδομένα).

γ) Θα συγκρίνουμε τον Naive Bayes, που φτιάξαμε με τον αντίστοιχο που μας δίνεται από την βιβλιοθήκη sklearn. Για τον έλεγχο της επίδοσης του ταξινομητή θα υπολογίσουμε και το score στα test δεδομένα, καθώς και το 5- fold-cross-validation score. Οι τιμές που εξάγουμε είναι οι εξής:

clf.score (x_test ,y_test) : 0.7198 (71.98%)
cross_val_score : 74.72%

Παρατηρούμε πως τα scores και του Custom ταξινομητή που υλοποιήσαμε, όπως και του έτοιμου της scikit-learn δίνουν ίδια αποτελέσματα με μία πολύ μικρή απόκλιση ως προς το ποσοστό επιτυχίας τους.

Βήμα 16

Σε αντίθεση με την υλοποίηση του Naive Bayes του προηγούμενου βήματος, εδώ θα θεωρήσουμε ότι η διασπορά του κάθε feature, για κάθε μία από τις κλάσεις ισούται με την μονάδα.
Δεδομένου πως ο Ευκλείδειος ταξινομητής προκύπτει από τον Naive Bayes με μοναδιαία τιμή διασποράς για όλα τα χαρακτηριστικά της κάθε κλάσης και ίσες a-priori πιθανότητες, αναμένουμε το score να είναι καλύτερο από ότι στην περίπτωση που το κάθε feature έχει διαφορετική διασπορά.

clf.score (x_test ,y_test) : 0.8126 (81.26%)
cross_val_score : 85.04%

Παρατηρούμε, μάλιστα πως οι τιμές αυτές είναι συγκρίσιμες με εκείνες του Ευκλείδειου Ταξινομητή (στο 5-fold-cross-validation score έχει ελάχιστα καλύτερα αποτελέσματα).

Βήμα 17

Σε αυτό το βήμα, θα συγκρίνουμε τις επιδόσεις των παρακάτω ταξινομητών:

- Naive Bayes (GaussianNB)
- kNearest Neighbors (με k = 1 και 3)
- SVC (rbf kernel)
- SVC (linear kernel)

Ταξινομητής	Score (test data)	Cross-Score(5-fold)
Naïve Bayes	0.7198	0.7472
kNearest Neighbors(k=1)	0.9436	0.9707
kNearest Neighbors(k=3)	0.9446	0.9673
SVC (rbf kernel)	0.9471	0.9769
SVC (linear kernel)	0.9262	0.9577

Από τον παραπάνω πίνακα βλέπουμε πως ο περισσότερο αποδοτικός ταξινομητής είναι SVM, με kernel = ‘rbf’. Καλή απόδοση εμφανίζει και ο ταξινομητής Nearest Neighbor , ο οποίος έχει ελάχιστα χαμηλότερη απόδοση

Βήμα 18

Α) Προκειμένου να βελτιώσουμε και άλλο την απόδοση των ταξινομητών θα εφαρμόσουμε την τεχνική ensembling. Αρχικά, θα εντοπίσουμε σε ποια ψηφία εμφανίζονται τα λάθη των ταξινομητών, ώστε ο συνδυασμός που θα επιλέξουμε να έχει όσο το δυνατόν λιγότερα κοινά λάθη. Παρακάτω παραθέτουμε τα confusion matrices ορισμένων από των ταξινομητών

Nearest Neighbor:

```
[[355   0   2   0   0   0   0   1   0   1]
 [  0 255   0   0   6   0   2   1   0   0]
 [  6   1 183   2   1   0   0   2   3   0]
 [  3   0   2 154   0   5   0   0   0   2]
 [  0   3   1   0 182   1   2   2   1   8]
 [  2   1   2   4   0 145   2   0   3   1]
 [  0   0   1   0   2   3 164   0   0   0]
 [  0   1   1   1   4   0   0 139   0   1]
 [  5   0   1   6   1   1   0   1 148   3]
 [  0   0   1   0   2   0   0   4   1 169]]
```

SVM Rbf Kernel:

```
[[355   0   2   0   1   0   0   0   1   0]
 [  0 254   1   0   5   0   3   1   0   0]
 [  2   0 184   2   3   2   0   1   4   0]
 [  2   0   3 147   0  10   0   0   4   0]
 [  0   1   3   0 188   1   1   1   1   4]
 [  2   0   0   3   1 151   0   0   1   2]
 [  3   0   3   0   2   2 159   0   1   0]
 [  0   0   1   0   5   0   0 138   1   2]
 [  2   0   2   2   0   2   1   1 155   1]
 [  0   0   0   1   6   0   0   0   0 170]]
```

SVM Linear Kernel :

```
[[351   0   1   0   3   0   2   1   1   0]
 [  0 256   0   2   3   0   3   0   0   0]
 [  0   0 181   7   3   3   2   1   1   0]
 [  3   0   2 146   0   9   0   1   5   0]
 [  1   2   6   0 180   1   3   3   0   4]
 [  4   0   1  14   2 136   0   0   2   1]
 [  3   0   3   0   2   2 160   0   0   0]
 [  1   0   1   1   5   0   0 135   0   4]
 [  5   0   4   4   3   5   0   1 144   0]
 [  0   0   0   1   2   0   0   4   0 170]]
```

Παρατηρούμε πως ο ταξινομητής NN εμφανίζει τα περισσότερα λάθη του στα ψηφία 3,4,7,9 ο ταξινομητής RBF στα ψηφία 2,4,5,8 και ο linear στα 2,3,4,5,7.
Επομένως, με τον συνδυασμό των μεθόδων σε έναν voting classifier περιμένουμε να βελτιωθεί η συνολική επίδοση του ταξινομητή, καθώς μόνο στο 4, παρουσιάζουν το υψηλότερο ποσοστό λάθος και οι 3 ταξινομητές.
Προτιμούμε την επιλογή περιττού αριθμού ταξινομητών (στο δικό μας παράδειγμα 3) στον voting classifier, για να μην υπάρχει περίπτωση ισοψηφίας.
Τα αποτελέσματα του Voting Classifier

clf.score (x_test ,y_test) : 0.9501 (95.01%)
cross_val_score : 97.90%

B) Μια άλλη μέθοδος Ensembling είναι η Bagging, με την οποία εκπαιδεύουμε τον ίδιο ταξινομητή με διαφορετικά τμήματα του training set.
Στην περίπτωση μας, θα χρησιμοποιήσουμε τον SMV ταξινομητή με kernel = ‘rbf’, καθώς εμφανίζει τα καλύτερα αποτελέσματα, όπως έχουμε δει.
Τα αποτελέσματα του Bagging Classifier είναι τα εξής:

clf.score (x_test ,y_test) : 0.9486 (94.86%)
cross_val_score : 97.70%

Γ) Παρατηρούμε μια μικρή βελτίωση στην επίδοση των ταξινομητών. Ο λόγος για τον οποίο η βελτίωση αυτή δεν είναι μεγαλύτερη είναι ότι για τον μεν Voting Classifier χρησιμοποιούμε τον μικρότερο δυνατό περιττό αριθμό ταξινομητών (όσο περισσότερους ταξινομητές βάλουμε ,τόσο καλύτερα θα είναι και τα αποτελέσματα), ενώ για τον Bagging, χρησιμοποιούμε μικρό αριθμό υποσυνόλων του dataset για την εκπαίδευση του μοντέλου (στη δική μας περίπτωση χρησιμοποιούμε 10 estimators , ενώ για καλύτερα αποτελέσματα θα χρειαζόμασταν > 100)

Βήμα 19

Α) Στο συγκεκριμένο ερώτημα υλοποιούμε τον dataloader ως μία κλάση, που αποτελείται από τις συναρτήσεις __init__() (constructor), __len__(), η οποία επιστρέφει τον αριθμό των training δεδομένων που διαθέτουμε και την __getitem__(), η οποία παίρνει ως όρισμα ένα index και επιστρέφει τα features του στοιχείου που βρίσκεται στο συγκεκριμένο index

Β) Υλοποιήσαμε ένα νευρωνικό δίκτυο, το οποίο περιέχει in_layer,out_layer και 2 επιπλέον layers με 100 νευρώνες ο καθένας. Όπως φαίνεται και παρακάτω, αρχικά πάμε από τις 256 διαστάσεις, που είναι τα features, στις 100 διαστάσεις, που είναι οι νευρώνες του 1^{ου} layer, στην συνέχεια παραμένουμε στις 100 διαστάσεις (του 2^{ου} layer), για να καταλήξουμε στις 10 διαστάσεις του out_layer, που είναι και ο αριθμός των κλάσεων στις οποίες κάνουμε την ταξινόμηση.

Η υλοποίηση, που περιγράψαμε, φαίνεται και στο επόμενο σχήμα:

```
The network architecture is:
 FullyConnectedNN(
  (f): Sequential(
    (0): LinearWActivation(
      (f): Linear(in_features=256, out_features=100, bias=True)
      (a): Sigmoid()
    )
    (1): LinearWActivation(
      (f): Linear(in_features=100, out_features=100, bias=True)
      (a): Sigmoid()
    )
    (2): LinearWActivation(
      (f): Linear(in_features=100, out_features=10, bias=True)
      (a): Sigmoid()
    )
  )
  (clf): Linear(in_features=10, out_features=10, bias=True)
)
```