ΕΜΠ 2021-2022

Αναγνώριση Προτύπων

1η Εργαστηριακή Αναφορά

Δημήτρης Κουνούδης 03117169
Χατζηθεοδώρου Ιάσων 03117089

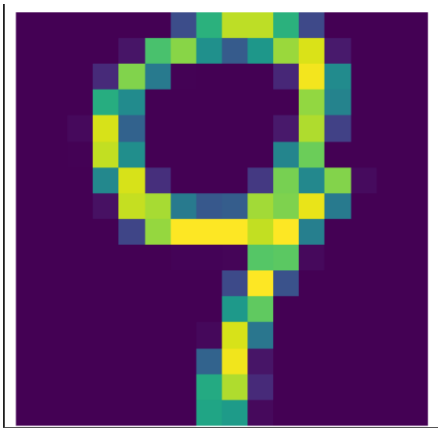
Σκοπός αυτής της εργαστηριακής άσκησης είναι η υλοποίηση ενός συστήματος οπτικής αναγνώρισης ψηφίων τα οποία λάβαμε από την US Postal Service χειρόγραφα και περιέχουν ψηφία από το 0 έως το 9 και διακρίνονται σε train και test. Τα δεδομένα κάθε αρχείου αναπαρίστανται όπως περιγράφεται στην εκφώνηση. Στόχος μας η δημιουργία και αποτίμηση (evaluation) ταξινομητών οι οποίοι θα ταξινομούν κάθε ένα από τα ψηφία που περιλαμβάνονται στα test δεδομένα σε μία από τις δέκα κατηγορίες (από το 0 έως το 9). Καθ'όλη την υλοποίηση που κάναμε συμπληρώναμε και τις συναρτήσεις και κλάσεις του αρχείου lib.py που μας δίνεται. ΄Ολη η εργαστηριακή άσκηση βρίσκεται στα αρχεία PatRec_Lab1.ipynb (όπου έγινε αρχικά η υλοποίηση) , lib.py , notebook.py (μετατροπή του .ipynb σε .py) .

Βήμα 1

Αρχικά αποθηκεύουμε τα δεδομένα από τα train.txt , test.txt με τη βοήθεια της συνάρτησης read_data_to_array η οποία χωρίζει τα δεδομένα ανά γραμμή και επιστρέφει έναν πίνακα (np.array) με τα δεδομένα που διαβάζει από το αρχείο που παίρνει ως όρισμα. Δημιουργούμε 4 πίνακες , τους X_test , X_train οι οποίοι περιέχουν μόνο τα features των αριθμών (256 στοιχεία για κάθε χειρόγραφο νούμερο) χωρίς το πρώτο στοιχείο που αντιστοιχεί στον αριθμό και τους y_train,y_test που αποτελούνται μόνο από τα νούμερα αυτά

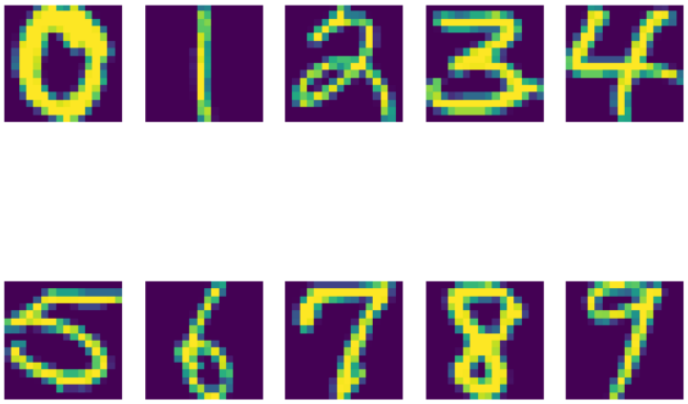
Βήμα 2

Χρησιμοποιώντας την show_sample η οποία υλοποιείται στο αρχείο lib.py το οποίο έχουμε κάνει import , κάνοντας reshape σε 16X16 pixel καλούμε από τον πίνακα X_train το 131ο στοιχείο το οποίο είναι το νούμερο 9 :



Βήμα 3

Λαμβάνοντας 1 τυχαίο δείγμα από κάθε label σχεδιάζουμε σε ένα figure 10 subplots με όλα τα δείγμα



Βήμα 4

Υπολογίζουμε τη μέση τιμή των χαρακτηριστικών του pixel (10, 10) για το ψηφίο 0 με βάση τα train δεδομένα υλοποιώντας την συνάρτηση digit_mean_at_pixel(X, y, digit, pixel=(10, 10)) και καλώντας την συγκεκριμένα με ορίσματα X_train , y_train , 0 , (10,10). Στην υλοποίηση της χρησιμοποιούμε την συνάρτηση mean της numpy για τον υπολογισμό της μέσης τιμής των χαρακτηριστικών των δειγμάτων. Το αποτέλεσμα που λαμβάνουμε είναι: `-0.5041884422110553`

Βήμα 5

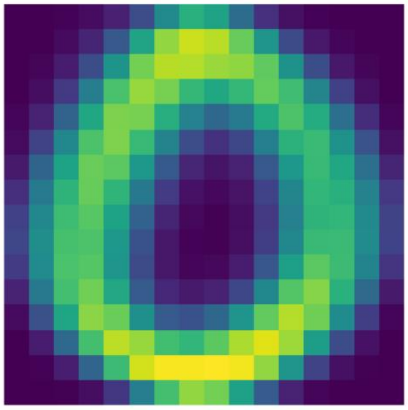
Υπολογίζουμε τη διασπορά των χαρακτηριστικών του pixel (10, 10) για το ψηφίο 0 με βάση τα train δεδομένα υλοποιώντας της συνάρτηση digit _variance_at_pixel(X, y, digit, pixel=(10, 10)) και καλώντας την συγκεκριμένα με ορίσματα X_train , y_train , 0 , (10,10). Το αποτέλεσμα που παίρνουμε είναι: `0.5245221428814929`

Βήμα 6

Υπολογίζουμε τη μέση τιμή και διασπορά των χαρακτηριστικών κάθε pixel για το ψηφίο 0 με βάση τα train δεδομένα, χρησιμοποιώντας τις δύο προηγούμενες συναρτήσεις σε κάθε pixel του ψηφίου digit (εδώ ”0”). Υλοποιούμε τις συναρτήσεις digit_mean(X, y, digit) και digit_variance(X, y, digit) οι οποίες μας επιστρέφουν από έναν πίνακα 256 στοιχείων για το δοσμένο ως όρισμα ψηφίο της μέσης τιμής και της διασποράς των χαρακτηριστικών. Οι πίνακες είναι 256 στοιχείων καθώς η εικόνα του ψηφίου 0 είναι 16*16 pixel (=256) οπότε κάθε στοιχείο του πίνακα είναι η μέση τιμή όλων των τιμών που εμφανίζονται στο συγκεκριμένο πίξελ στις εικόνες με το ψηφίο 0.

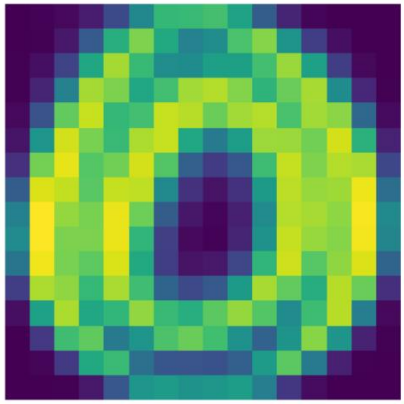
Βήμα 7

Σχεδιάζουμε το ψηφίο 0 χρησιμοποιώντας τις τιμές της μέσης τιμής που υπολογίσαμε στο προηγούμενο βήμα :



Βήμα 8

Χρησιμοποιώντας τις τιμές της διασποράς του βήματος 6 σχεδιάζουμε το μηδενικό που φαίνεται παρακάτω :

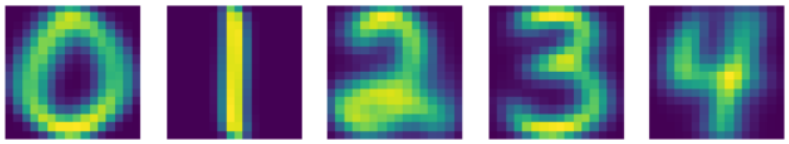


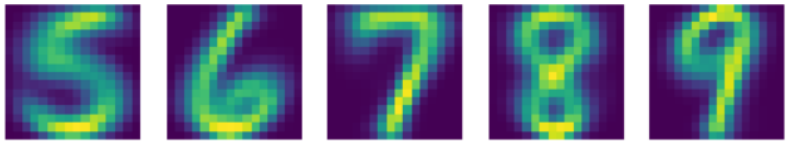
Παρατηρούμε πως το ψηφίο 0 σχεδιασμένο με τις διασπορές είναι πιο ”παχύ”. Αυτό συμβαίνει διότι η μεγαλύτερη διασπορά στα pixels όλων των μηδενικών ψηφίων είναι γύρω από την γραμμή του 0, και για το λόγο αυτό φαίνονται τα pixels αυτά πιο έντονα. Ουσιαστικά στην περίπτωση του variance διαχωρίζεται η γραμμή του ψηφίου 0 σε δύο μέρη. Αφού τα περισσότερα δείγματα του συγκεκριμένου ψηφίου σε εκείνο το κομμάτι συμφωνούν ,το variance είναι πολύ χαμηλό (πιο σκούρο χρώμα) , ενώ αντίθετα διαφωνούν στο πού ακριβώς είναι τα όρια των γραμμών, για αυτό και είναι πιο έντονο το χρώμα εκεί που θα περιμέναμε να είναι το τέλος της γραμμής.

Βήμα 9

Μέσω ενός loop δημιουργούμε 2 λίστες με τις τιμές της μέσης τιμής και της διασποράς των χαρακτηριστικών για κάθε ψηφίο χρησιμοποιώντας τις συναρτήσεις των προηγουμένων ερωτημάτων digit_mean και digit_variance . Ακολουθούμε ακριβώς την ίδια λογική με το παραπάνω ερώτημα.

Στη συνέχεια χρησιμοποιώντας τις τιμές της μέσης τιμής που υπολογίσαμε στο προηγούμενο ερώτημα σχεδιάζουμε όλα τα ψηφία από 0-9 :

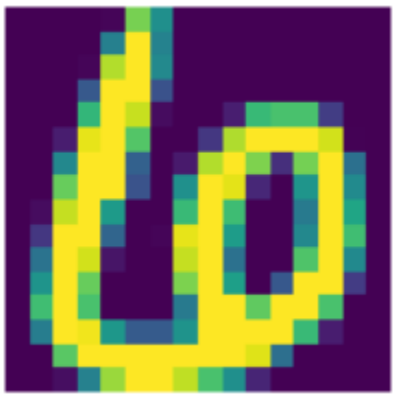




Βήμα 10

Χρησιμοποιώντας τον ευκλείδειο ταξινομητή τον οποίο υλοποιήσαμε με τη βοήθεια της συνάρτησης euclidean_distance (στην οποία χρησιμοποιήσαμε από τη βιβλιοθήκη numpy την linalg.norm) , euclidean_distance_classifier προσπαθούμε να προβλέψουμε το παρακάτω χειρόγραφο νούμερο (το υπ' αριθμόν 101ο ψηφίο) . Ουσιαστικά η συνάρτηση euclidean_distance υπολογίζει την ευκλείδεια απόσταση μεταξύ δύο πινάκων . Η euclidean_distance_classifier παίρνει ως ορίσματα τα δεδομένα εισόδου , το δείγμα που θέλουμε να ταξινομήσουμε και τον πίνακα με τις μέσες τιμές και βρίσκει με ποιου ψηφίου μέσες τιμές είναι πιο κοντά το δείγμα που έχουμε ως παράμετρο . Αυτό γίνεται βρίσκοντας την ελάχιστη ευκλείδεια απόσταση μέσω της συνάρτησης της numpy argmin.

Το χειρόγραφο νούμερο φαίνεται παρακάτω:



Η πρόβλεψη δεν γίνεται με επιτυχία καθώς αντί να εντοπίσει ο ταξινομητής μας το νούμερο 6 , εντοπίζει το νούμερο 0 , γεγονός όχι παράλογο καθώς δεν είναι ευδιάκριτο το νούμερο που έχει γραφτεί χειρόγραφα και ο ευκλείδειος ταξινομητής δεν είναι ο βέλτιστος δυνατός.

```
The prediction is 0
The correct answer is 6
```

Βήμα 11

Καλούμε την συνάρτηση euclidean_distance_classifier για ολόκληρο το test set μας (X_test) κι όχι μόνο για το υπ'αριθμόν 101 ψηφίο όπως παραπάνω και υπολογίζουμε το ποσοστό επιτυχίας αυτής της διαδικασίας ως : `The accuracy is 0.8141504733432985`

Βήμα 12

Δημιουργούμε μια κλάση EuclideanDistanceClassifier μεσα στην οποία ορίζουμε τις εξής συναρτήσεις : Την __init__ για την αρχικοποίηση της η οποία θέτει τη μεταβλητή self.mean=None . Την fit η οποία δέχεται ως όρισμα το training set (ορισμα X) , υπολογίζει τους πίνακες μέσης τιμής για τα ψηφία 0-9 και αποθηκεύει έναν np.array με τις μέσες τιμές του ορίσματος X στην self.mean επιστρέφοντας τον self. Την predict η οποία καλεί την euclidean_distance_classifier για το δείγμα που εισάγουμε και το ταξινομεί σε μια από τις 10 κλάσεις των ψηφίων όπως περιγράψαμε παραπάνω. Την score η οποία μέσω της metrics.accuracy_score της βιβλιοθήκης sklearn επιστρέφει την ακρίβεια της πρόβλεψης που έγινε βάσει των ορισμάτων της.

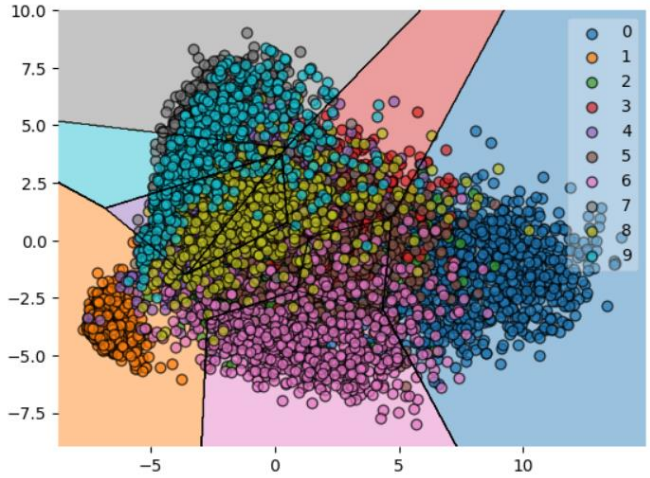
Βήμα 13

α)Υπολογίζουμε το score του ευκλείδειου ταξινομητή με χρήση 5-fold cross-validation μέσω της evaluate_euclidean_classsifier η οποία καλεί την evaluate_classifier για ένα στιγμιότυπο της κλάσης EuclideanDistanceClassifier. Η συνάρτηση μας λαμβάνει ως όρισμα το σύνολο δεδομένων καθώς και τα folds και επιστρέφει την ακρίβεια του ταξινομητή. Το αποτέλεσμα είναι αρκετά ικανοποιητικό:

`The 5-fold accuracy score of the classifier is 0.8485803550358166`

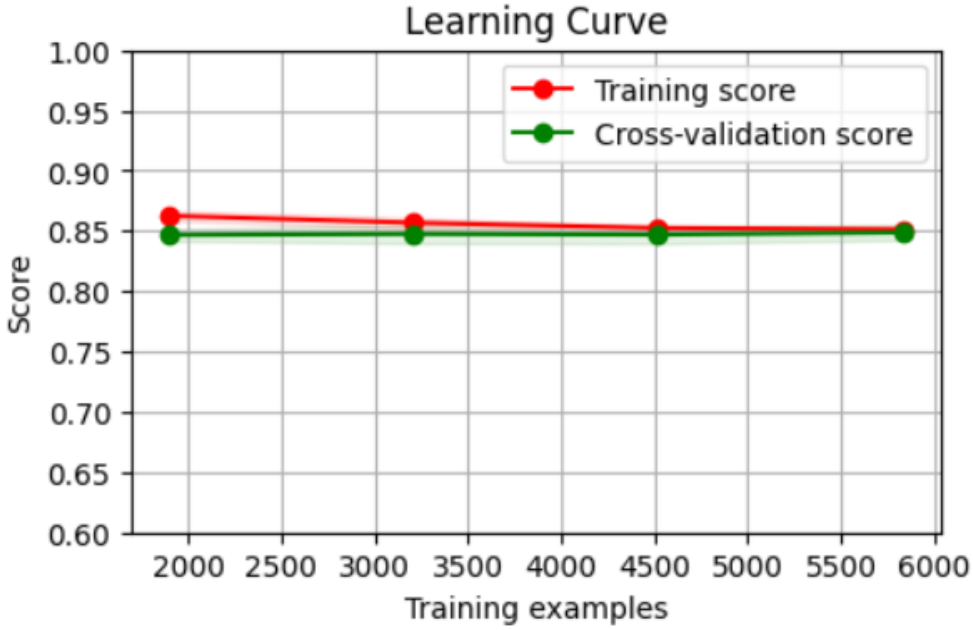
β)Για να μπορέσουμε να σχεδιάσουμε την περιοχή απόφασης περνάμε το train dataset από PCA με σκοπό να το μετατρέψουμε σε 2 διαστάσεις από 256 ώστε να μπορεί να αναπαρασταθεί γραφικά . Οπότε από 256 features κάθε ψηφίου «συμπυκνώνουμε» την πληροφορία σε 2 features κρατώντας τα 2 features με το μεγαλύτερο variance δηλαδή αυτά που διαφοροποιούνται περισσότερο στα δείγματα και αρα μπορούν να τα διαχωρίσουν καλύτερα. Έπειτα εκπαιδεύουμε τον ευκλείδειο ταξινομητή στα δεδομένα των 2 αυτών διαστάσεων .

Για την ευδιάκριτη αναπαράσταση των σημείων και των περιοχών απόφασης χρησιμοποιήσαμε την συνάρτηση plot_decision_regions της βιβλιοθήκης mlxtend :



Παρατηρούμε πως οι κλάσεις δεν διαχωρίζονται πολύ καλά καθώς πιθανών δεν φτάνουν οι 2 διαστάσεις στις οποίες «μειώσαμε» το dataset μας . Διατηρώντας παραπάνω διαστάσεις θα είχαμε καλύτερο αποτέλεσμα.

γ) Σχεδιάζουμε την καμπύλη εκμάθησης του ευκλείδειου ταξινομητή βάσει του κώδικα που μας δόθηκε στο εργαστήριου και λαμβάνουμε την παρακάτω γραφική παράσταση :



Παρατηρούμε ότι το μοντέλο δεν κάνει overfit καθώς δεν αυξάνεται η απόσταση των καμπύλων κατά τη διάρκεια του training και μάλιστα συγκλίνουν όσο αυξάνονται τα training examples όπως επιθυμούμε.

Βήμα 14

Για τον υπολογισμό των a-priori πιθανοτήτων κάθε κατηγορίας υπολογίσαμε το πλήθος των δειγμάτων του train dataset τα οποία ανήκουν σε κάθε κλάση και διαιρέσαμε με το συνολικό πλήθος δειγμάτων που δίνονταν. Η συνάρτηση που τα υπολογίζει είναι η *lib.calculate_priors*. Τα αποτελέσματα που προέκυψαν είναι τα εξής

```
The a-priori probability of class 0 is 0.16376
The a-priori probability of class 1 is 0.13784
The a-priori probability of class 2 is 0.10026
The a-priori probability of class 3 is 0.09025
The a-priori probability of class 4 is 0.08943
The a-priori probability of class 5 is 0.07626
The a-priori probability of class 6 is 0.09107
The a-priori probability of class 7 is 0.08847
The a-priori probability of class 8 is 0.07434
The a-priori probability of class 9 is 0.08833
```

Βήμα 15

Για την υλοποίηση του Naive Bayes προστέθηκαν υλοποιήσεις των μεθόδων fit και predict. Το training έγινε χρησιμοποιώντας τις προηγούμενες συναρτήσεις υπολόγιζε τις a-priori πιθανότητες και τις μέσες τιμές και τυπικές αποκλίσεις των χαρακτηριστικών για κάθε κλάση. Συγκεκριμένα για την τυπική απόκλιση, λόγω του ότι κάποιες προκύπτουν 0, προσθέτουμε σε όλες μικρή σταθερά ίση με 10^{-9}. Για το prediction, υπολογίσαμε τους λογαρίθμους των a-posteriori πιθανοτήτων ως εξής

$$logP(c|\boldsymbol{x}) = logP(c) + \sum_i logP(x_i|c)$$

Όπου οι P_i είναι κανονικές κατανομές με μέση τιμή και τυπική απόκλιση ίση με αυτή που υπολογίσαμε στο training οπότε ο υπολογισμός τους γίνεται από την $lib.log\_gaussian\_distribution$ ως εξής

$$logP(x_i|c) = -\frac{(x_i - E[x_i|c])^2}{2Var[x_i|c]} - \frac{1}{2}\log(2\pi * Var[x_i|c])$$

Παίρνοντας τα predictions πάνω στο test dataset υπολογίζουμε την εξής ακρίβεια

```
The accuracy of the custom implementation is 0.7199800697558545
```

Παρακάτω φαίνονται για σύγκριση τα αποτελέσματα της έτοιμης υλοποίησης για διαφορετικά var_smoothing

```
The accuracy of the sklearn implementation for var_smoothing=1e-09 is 0.71948
The accuracy of the sklearn implementation for var_smoothing=1e-05 is 0.74689
The accuracy of the sklearn implementation for var_smoothing=0.01 is 0.79621
The accuracy of the sklearn implementation for var_smoothing=0.1 is 0.81565
The accuracy of the sklearn implementation for var_smoothing=0.5 is 0.78974
The accuracy of the sklearn implementation for var_smoothing=2 is 0.73842
```

Το accuracy που έβγαλε η δικιά μας υλοποίηση ήταν πρακτικά ίδιο με τον default GaussianNB ($var\_smoothing = 1e-09$). Αντίθετα η υλοποίηση του scikit learn προσθέτει ένα συγκεκριμένο ποσοστό της μεγαλύτερης απόκλισης σε όλες τις αποκλίσεις, το οποίο μπορεί να αλλάξει ο χρήστης για να βρει το βέλτιστο. Όπως φαίνεται στην συγκεκριμένη περίπτωση το $var\_smoothing = 0.1$ έχει αρκετά ικανοποιητικό accuracy.

Βήμα 16

Επαναλαμβάνοντας τα παραπάνω θεωρώντας ότι οι όλες οι διασπορές των κανονικών κατανομών είναι ίσες με 1, παίρνουμε το εξής

```
The accuracy of the custom implementation with unit variances is 0.8126557050323866
```

Το οποίο πρακτικά ταυτίζεται με το καλύτερο αποτέλεσμα που βρήκαμε με την έτοιμη υλοποίηση

Βήμα 17

Σε αυτό το βήμα δοκιμάσαμε τους εξής classifiers από τη βιβλιοθήκη scikit learn χρησιμοποιώντας 5-fold cross validation

- Linear SVM
- Radial Basis Function SVM
- 5-Nearest Neighbors
- Naive Bayes

```
The 5-fold cross validation accuracy of Linear SVM is 0.95254
The 5-fold cross validation accuracy of RBF SVM is 0.97627
The 5-fold cross validation accuracy of 5-Nearest Neighbors is 0.95995
The 5-fold cross validation accuracy of Naive Bayes is 0.84501
```

Η καλύτερη ακρίβεια επιτυγχάνεται με το Radial Basis Function SVM και η χειρότερη με τον Naïve Bayes.

Βήμα 18

Για τους παραπάνω ταξινομητές (εκτός του rbf SVM) μετρήσαμε πόσα λάθη κάνουν ανά ψηφίο (στο test dataset) χρησιμοποιώντας τη συνάρτηση *lib.classifier_mistakes_per_class* και πήραμε τα εξής αποτελέσματα

```
Linear SVM mispredictions per class are [ 8  8 17 20 20 24 10 12 22  7]
Gaussian Naive Bayes mispredictions per class are [43 10 49 40 75 60 17 22 36 18]
5-Nearest Neighbor mispredictions per class are [ 5  5 16 12 17 16  7  9 15  9]
```

Παρατηρούμε ότι η κατανομή λαθών παρουσιάζει διαφορές οπότε ο συνδυασμός τους με voting classifier (hard voting) περιμένουμε να βελτιώσει το accuracy

```
The 5-fold cross validation accuracy of the voting classifier with hard voting is 0.95446
```

Το accuracy που πετυχαίνει ταυτίζεται πρακτικά με αυτό του 5-Nearest Neighbors.

Για τον bagging classifier χρησιμοποιήσαμε τον Gaussian Naïve Bayes και πήραμε το εξής αποτέλεσμα

```
The 5-fold cross validation accuracy of the bagging classifier using Gaussian Naive Bayes is 0.84721
```

Όπως βλέπουμε το accuracy δεν βελτιώνεται ιδιαίτερα πολύ.

Συνολικά, οι παραπάνω τεχνικές δεν μας έδωσαν σημαντικές βελτιώσεις σε σχέση με τους απλούς ταξινομητές, άρα φαίνεται πως στα συγκεκριμένα δεδομένα μας είναι αρκετό το να χρησιμοποιήσουμε SVM ή k-Nearest Neighbors.

Βήμα 19

Σε αυτό το βήμα υλοποιήθηκε ένα πλήρες νευρωνικό δίκτυο που αναγνωρίζει τα χειρόγραφα ψηφία.

Για τη φόρτωση των δεδομένων χρησιμοποιήθηκε η κλάση lib.BlobData, η οποία ουσιαστικά μετατρέπει τα δεδομένα σε δυάδες από sample (διάνυσμα με χαρακτηριστικά) και την κλάση στην οποία ανήκει. Έπειτα χρησιμοποιήθηκε η DataLoader του PyTorch ώστε να χωριστούν τα δεδομένα σε batches με τυχαίο τρόπο.

Για την υλοποίηση της κλάσης lib.Net, η οποία υλοποιεί το νευρωνικό και κληρονομεί από το nn.Module, προσθέσαμε nn.Linear μεγέθους που καθορίζει ο χρήστης όταν καλέσεις την κλάση. Μετά από κάθε nn.Linear προσθέσαμε ένα nn.ReLU. Μετά από το τελικό layer που ήταν nn.Linear(10, 10) εφαρμόσαμε ένα nn.LogSoftmax(dim=0). Όλα αυτά ενώθηκαν χρησιμοποιώντας την nn.Sequential. Παρατηρήσαμε ότι αντικαθιστώντας το nn.LogSoftmax με οποιαδήποτε άλλη επιλογή (πχ nn.Sigmoid, nn.LogSigmoid, nn.ReLU) και για μικρό αριθμό layers, το νευρωνικό προέβλεπε μόνο 0 και άρα δεν είχε νόημα η χρήση του.

Τέλος, για την υλοποίση της lib.PytorchNNModel ορίσαμε ως optimizer το Stochastic Gradient Descent και loss function την nn.NLLLoss.

Για το training έγιναν τα εξής βήματα

1. Μηδενισμός παραγώγων
2. Forward pass για υπολογισμό της εξόδου του νευρωνικού
3. Εφαρμογή loss function ώστε να μάθουμε το σφάλμα
4. Υπολογισμός όλων των παραγώγων με χρήση της backwards
5. Προσαρμογή των βαρών με βάση τον optimizer

Τα predictions υπολογίστηκαν όπως και στις προηγούμενες κλάσεις με τη μόνη διαφορά ότι πρώτα απενεργοποιήθηκε η λειτουργία υπολογισμού των παραγώγων για επιτάχυνση της διαδικασίας, εφόσον αυτές δεν μας χρειάζονταν πια.

Για το training του δικτύου χρησιμοποιήθηκαν 20 epochs, και learning rate 0.01. Η αξιολόγηση έγινε για κάποιες διαφορετικές επιλογές των layers όπως φαίνεται παρακάτω

```
The accuracy with layers [256, 10] is 0.84654
The accuracy with layers [256, 100, 10] is 0.85899
The accuracy with layers [256, 100, 50, 10] is 0.86099
The accuracy with layers [256, 400, 10] is 0.87145
The accuracy with layers [256, 400, 100, 50, 10] is 0.87444
The accuracy with layers [256, 400, 200, 100, 50, 10] is 0.85700
The accuracy with layers [256, 400, 300, 200, 100, 80, 50, 10] is 0.34081
The accuracy with layers [256, 400, 500, 300, 250, 200, 150, 100, 50, 30, 10] is 0.29198
```