ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΑΝΑΓΝΩΡΙΣΗ ΠΡΟΤΥΠΩΝ

(2021-2022)

1ο Εργαστηριακό Project

Θέμα: Οπτική Αναγνώριση Ψηφίων

Ονοματεπώνυμο:

- Χρήστος Τσούφης

Αριθμός Μητρώου:

- 03117176

Στοιχεία Επικοινωνίας:

- el17176@mail.ntua.gr

Περιγραφή

Σκοπός είναι η υλοποίηση ενός συστήματος οπτικής αναγνώρισης ψηφίων. Τα δεδομένα προέρχονται από την US Postal Service (γραμμένα στο χέρι σε ταχυδρομικούς φακέλους και σκαναρισμένα) και περιέχουν τα ψηφία από το 0 έως το 9 και διακρίνονται σε train και test. Τα δεδομένα κάθε αρχείου αναπαριστούν τα περιεχόμενα ενός πίνακα (οι τιμές των στοιχείων του πίνακα διαχωρίζονται με κενό). Κάθε γραμμή αφορά ένα ψηφίο (δείγμα). Οι στήλες αντιστοιχούν στα χαρακτηριστικά (features) που περιγράφουν τα ψηφία. Για παράδειγμα, η τιμή του (i, j) στοιχείου αφορά το j-th χαρακτηριστικό του i-th ψηφίου. Κάθε ψηφίο περιγράφεται από 257 τιμές, εκ των οποίων η πρώτη αντιστοιχεί στο ίδιο το ψηφίο (αν είναι το 0, το 1 κτλ.) και οι υπόλοιπες 256 είναι τα χαρακτηριστικά (features) που το περιγράφουν (grayscale values). Ας φανταστούμε το κάθε ψηφίο να απεικονίζεται σε έναν 16x16 πίνακα αποτελούμενο από 256 κουτάκια (“pixels”). Για να εμφανίζεται το κάθε ψηφίο στην οθόνη "φωτίζεται" ένα σύνολο από τέτοια κουτάκια, με τέτοιο τρόπο ώστε η συνολική εικόνα που βλέπουμε να απεικονίζει το θεωρούμενο ψηφίο. Επειδή τα ψηφία εμφανίζονται σε grayscale, κάθε μία από τις 256 τιμές αντιστοιχεί σε μία απόχρωση μαύρου για το αντίστοιχο “pixel”. Στόχος είναι 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).

Τεχνολογίες & Τρόπος Εκτέλεσης εφαρμογής

Η παρούσα εργασία υλοποιήθηκε σε ένα Python περιβάλλον και το σετάρισμα της εφαρμογής έγινε σε local περιβάλλον. Οι versions που χρησιμοποιήθηκαν, μετά από την εκτέλεση των παρακάτω εντολών στο terminal είναι:

`python --version` → **`3.9.2`**

`python → import sklearn → print('The scikit-learn version is {}.'.format(sklearn.__version__))` → **`1.0.1`**

`python → import numpy → numpy.version.version` → **`1.21.2`**

`pip3 list | findstr scikit → scikit-image =` **`0.18.2`**

`python → import matplotlib → print(matplotlib.__version__)` → **`3.4.3`**

`conda → conda -V → conda` **`4.10.3`**

(Τα πακέτα `tqdm`, `jupyter`, `nb_conda_kernels` είναι fixed)

Το project έχει την εξής δομή: Αποτελείται από 2 ξεχωριστά αρχεία (`lab1.py`, `lib.py`) που περιέχουν τα βήματα και τις συναρτήσεις αντίστοιχα. Η εκτέλεση των αρχείων γίνεται μέσω του `terminal` αφού πρώτα τοποθετηθούν στον ίδιο φάκελο τα κατάλληλα αρχεία (`train.txt`, `test.txt`) που θα χρησιμοποιηθούν ως input.

Εκτέλεση

Βήμα 1

Διαβάστε τα δεδομένα από το αρχείο. Τα δεδομένα πρέπει να διαβαστούν σε μορφή συμβατή με το scikit-learn σε 4 πίνακες X_train, X_test, y_train και y_test. Ο πίνακας X_train περιέχει τα δείγματα εκπαίδευσης, χωρίς τα labels) και είναι διάστασης (n_samples_train x n_features). Ο y_train είναι ένας μονοδιάστατος πίνακας μήκους n_samples και περιέχει τα αντίστοιχα labels για τον X_train. Αντίστοιχα για τα test δεδομένα.

Αρχικά, γίνεται το διάβασμα των δεδομένων από τα αρχεία train.txt και test.txt. Έπειτα, κατασκευάζονται οι πίνακες X_train, y_train, X_test, y_test. Σχετικά με τον πίνακα X_train, κάθε γραμμή του περιέχει μια γραμμή του αρχείου train.txt εκτός από το πρώτο στοιχείο, δηλαδή περιέχει τα 256 χαρακτηριστικά του ψηφίου που ορίζεται από το πρώτο στοιχείο της γραμμής του αρχείου, το label. Έτσι, το label που προσδιορίζει το ψηφίο εισάγεται στον πίνακα-λίστα y_train. Ομοίως συμβαίνει και για τους πίνακες X_test, y_test.

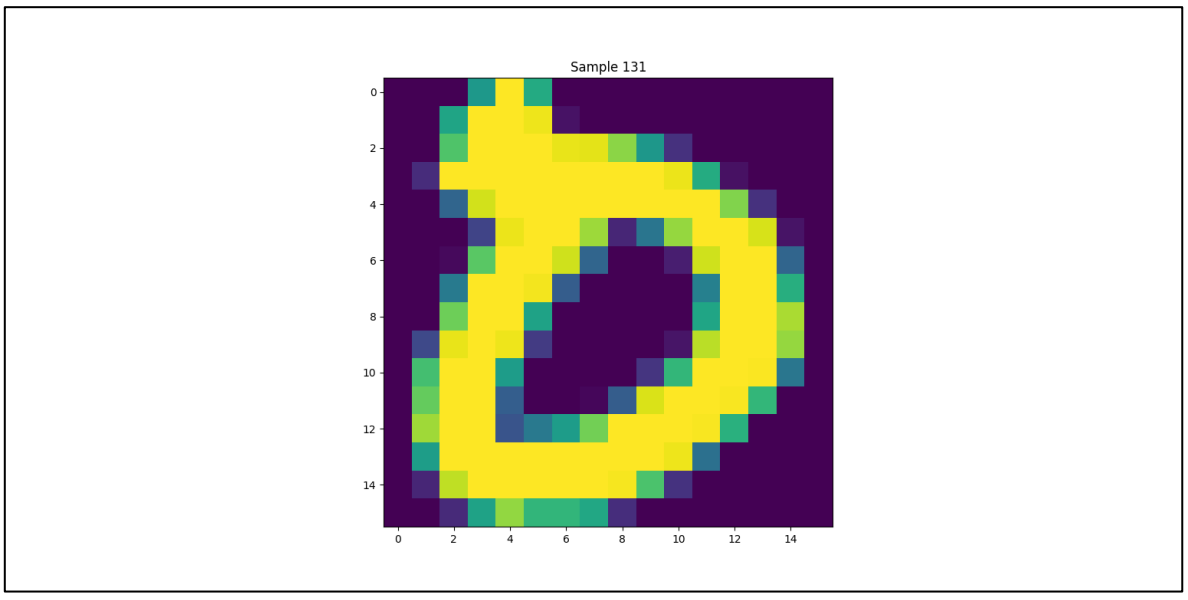
- Η συνάρτηση για το διάβασμα των δεδομένων ονομάζεται read_data και η υλοποίησή της φαίνεται στο αρχείο lab1.py.

Βήμα 2

Σχεδιάστε το υπ' αριθμόν 131 ψηφίο, (βρίσκεται στη θέση 131) των train δεδομένων. Υπόδειξη: χρησιμοποιήστε τη συνάρτηση numpy.reshape για να οργανώσετε τα 256 χαρακτηριστικά σε ένα πίνακα 16x16, και τη συνάρτηση matplotlib.pyplot.imshow για την απεικόνιση του ψηφίου.

Πρώτα, μετασχηματίζεται η γραμμή του πίνακα X_train για το ψηφίο σε ένα πίνακα 16×16. Έπειτα, από τα train data σχεδιάζεται το ψηφίο 131 και το αποτέλεσμα είναι το ακόλουθο.

- Η συνάρτηση ονομάζεται show_sample και η υλοποίησή της φαίνεται στο αρχείο lib.py.



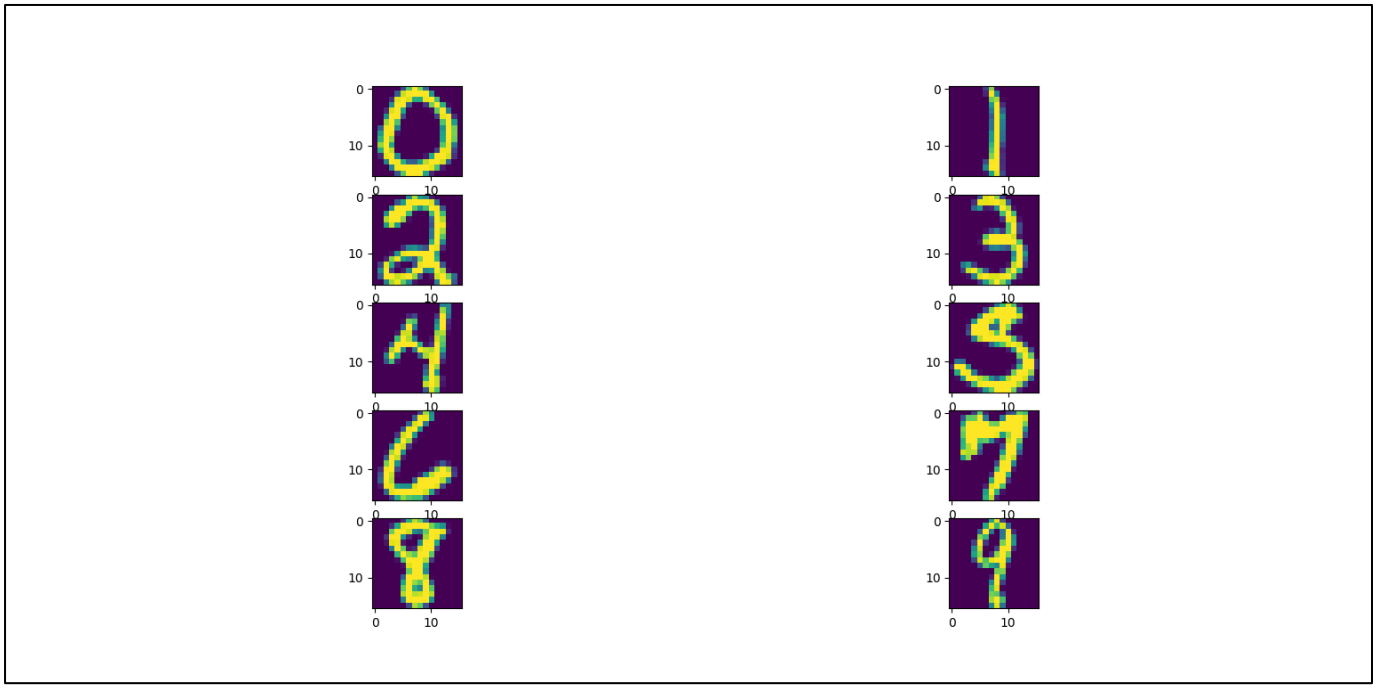
Βήμα 3

Διαλέξτε 1 τυχαίο δείγμα από κάθε label, συνολικά 10 δείγματα). Σχεδιάστε τα σε ένα figure με subplots.

(Hint: fig = plt.figure(); fig.add_subplot(,,,))

Αρχικά, για κάθε ψηφίο εντοπίζονται οι γραμμές του πίνακα X_train που αναφέρονται σε αυτό. Μετά, για κάθε ψηφίο επιλέγεται ένα τυχαίο instance του. Τέλος, σχεδιάζεται το κάθε ψηφίο οπότε προκύπτουν τα παρακάτω.

- Η συνάρτηση ονομάζεται plot_digits_samples και η υλοποίησή της φαίνεται στο αρχείο lib.py.



Βήμα 4

Υπολογίστε τη μέση τιμή των χαρακτηριστικών του pixel (10, 10) για το ψηφίο «0» με βάση τα train δεδομένα.

Σε αυτό το βήμα γίνεται ο υπολογισμός του μέσου όρου του pixel (10, 10) σε όλα τα samples του ψηφίου 0. Η τιμή που τυπώνεται είναι η εξής:

```
Mean value of all 0 samples at pixel (10, 10) is -0.5041884422110553
```

- Η συνάρτηση ονομάζεται digit_mean_at_pixel και αξιοποιεί πάλι την find_digit_index στο αρχείο lib.py.

Βήμα 5

Υπολογίστε τη διασπορά των χαρακτηριστικών του pixel (10,10) για το ψηφίο «0» με βάση τα train data.

Ομοίως με το προηγούμενο βήμα υπολογίζεται η διασπορά του pixel (10, 10) σε όλα τα samples του ψηφίου 0. Η τιμή που τυπώνεται είναι η εξής:

```
Variance of all 0 samples at pixel (10, 10) is 0.5245221428814929
```

- Η συνάρτηση ονομάζεται digit_variance_at_pixel και αξιοποιεί πάλι την find_digit_index στο αρχείο lib.py.

Βήμα 6

Υπολογίστε τη μέση τιμή και διασπορά των χαρακτηριστικών κάθε pixel για το ψηφίο «0» με βάση τα train δεδομένα.

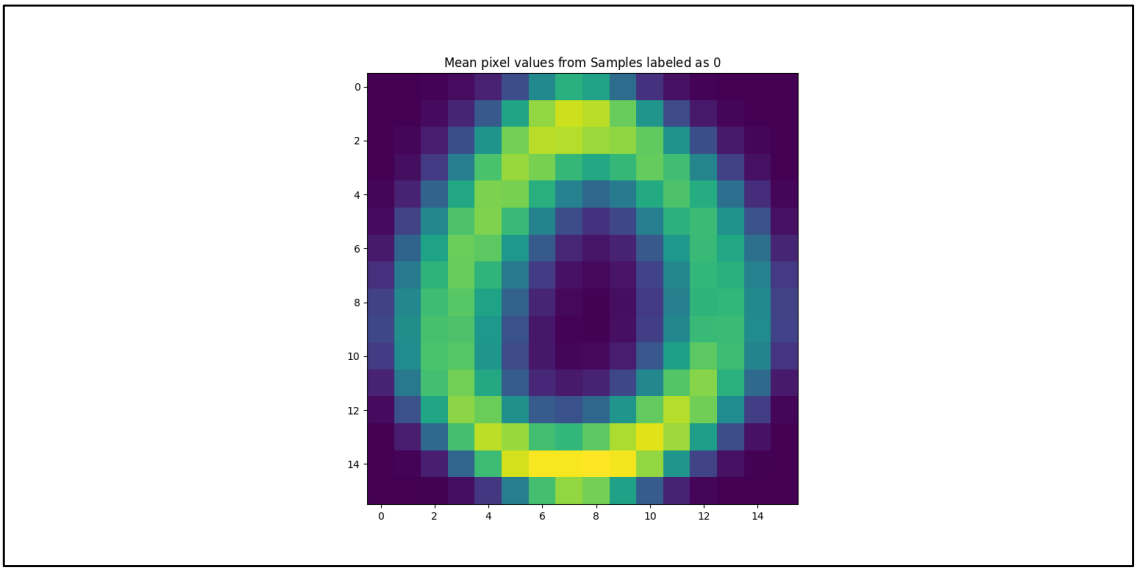
Εδώ γίνεται ο υπολογισμός της μέσης τιμής και της διασποράς των χαρακτηριστικών κάθε pixel σε όλα τα samples ενός αριθμού.

- Η συναρτήσεις ονομάζονται digit_mean και digit_variance και αξιοποιούν την digit_mean_at_pixel και digit_variance_at_pixel αντίστοιχα στο αρχείο lib.py.

Βήμα 7

Σχεδιάστε το ψηφίο «0» χρησιμοποιώντας τις τιμές της μέσης τιμής που υπολογίσατε στο Βήμα 6.

Τώρα, χρησιμοποιώντας την συνάρτηση imshow() του matplotlib, σχεδιάζεται ο πίνακας που έχει σε κάθε pixel την μέση τιμή των pixel από τα samples του ψηφίου «0».



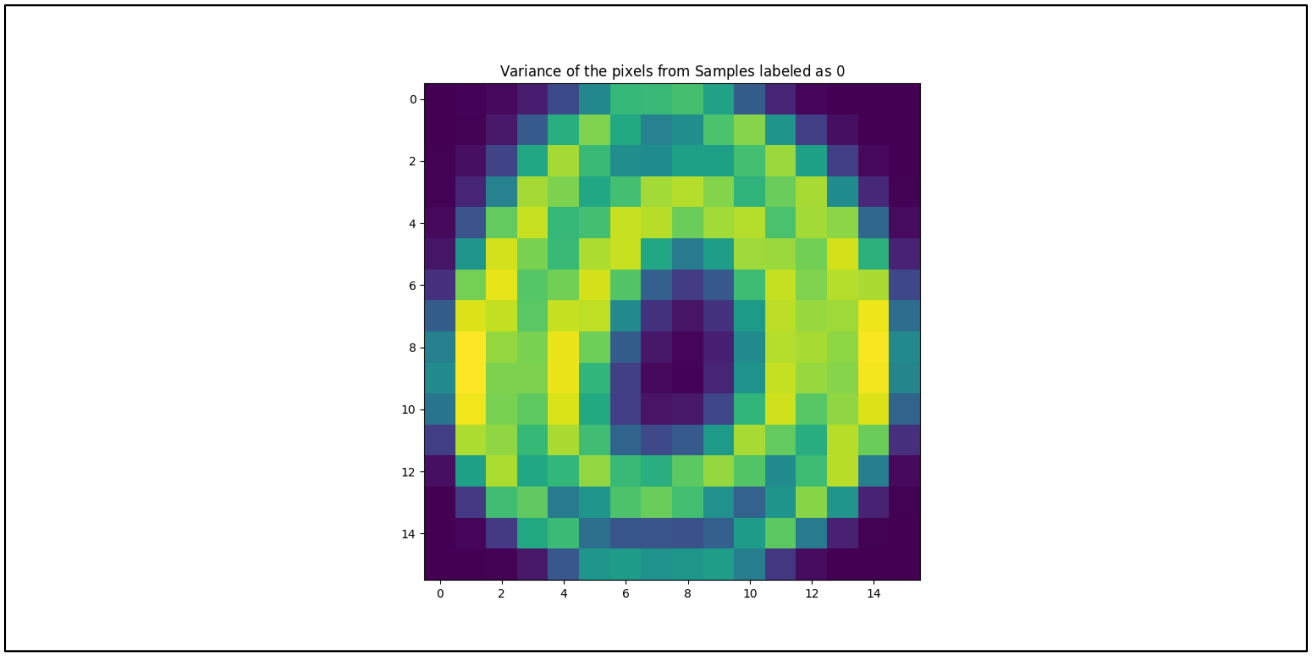
Σχολιασμός: Αυτό που παρατηρείται είναι ότι σχηματίζεται το ψηφίο «0», γεγονός αναμενόμενο δεδομένου ότι τα samples του ψηφίου «0» έχουν κατά μέσο όρο τιμές κοντά στο 1 στα ίδια σημεία, δηλ. στον κύκλο του ψηφίου «0». Επίσης, η μέση τιμή είναι πιο υψηλή στην βάση και στην κορυφή του ψηφίου γεγονός που οδηγεί στο συμπέρασμα ότι η πλειονότητα των samples έχουν την βάση του κύκλου στην ίδια περιοχή.

- Η υλοποίηση του Βήματος 7 φαίνεται στο lab1.py.

Βήμα 8

Σχεδιάστε το ψηφίο «0» χρησιμοποιώντας τις τιμές της διασποράς που υπολογίσατε στο Βήμα 6. Συγκρίνετε το αποτέλεσμα με το αποτέλεσμα του Βήματος 7 και εξηγείστε τυχόν διαφορές.

Ομοίως με το προηγούμενο βήμα, υπολογίζεται η διασπορά κάθε pixel σε όλα τα samples ενός αριθμού. Έτσι, χρησιμοποιώντας την συνάρτηση imshow() του matplotlib, σχεδιάζεται ο πίνακας που έχει σε κάθε pixel την διασπορά των pixel από τα samples του ψηφίου «0».



Σχολιασμός: Παρατηρείται ότι στις περιοχές που είναι σκουρόχρωμες, όπως είναι οι γωνίες, η διασπορά είναι μηδενική γεγονός αναμενόμενο δεδομένου ότι τα pixels σε αυτά τα σημεία έχουν σχεδόν την ίδια τιμή σε κάθε sample. Ακόμη, διακρίνονται δυο κύκλοι υψηλής διασποράς οι οποίοι αποτελούν το εσωτερικό και το εξωτερικό περίγραμμα του ψηφίου «0». Όπως φάνηκε και στο Βήμα 6, τα περιγράμματα των samples βρίσκονται στην ίδια περιοχή αλλά δεν είναι απαραίτητο να επικαλύπτουν το ένα το άλλο. Συνεπώς, η τιμή ενός pixel κοντά στην περιοχή ενός περιγράμματος είτε θα είναι περίπου -1, δηλαδή θα είναι “εκτός” του αριθμού, είτε θα είναι περίπου 1, δηλαδή θα είναι “πάνω” στον αριθμό, οπότε θα έχει υψηλή διακύμανση και τελικά υψηλή διασπορά.

- Η υλοποίηση του Βήματος 8 φαίνεται στο lab1.py.

Βήμα 9

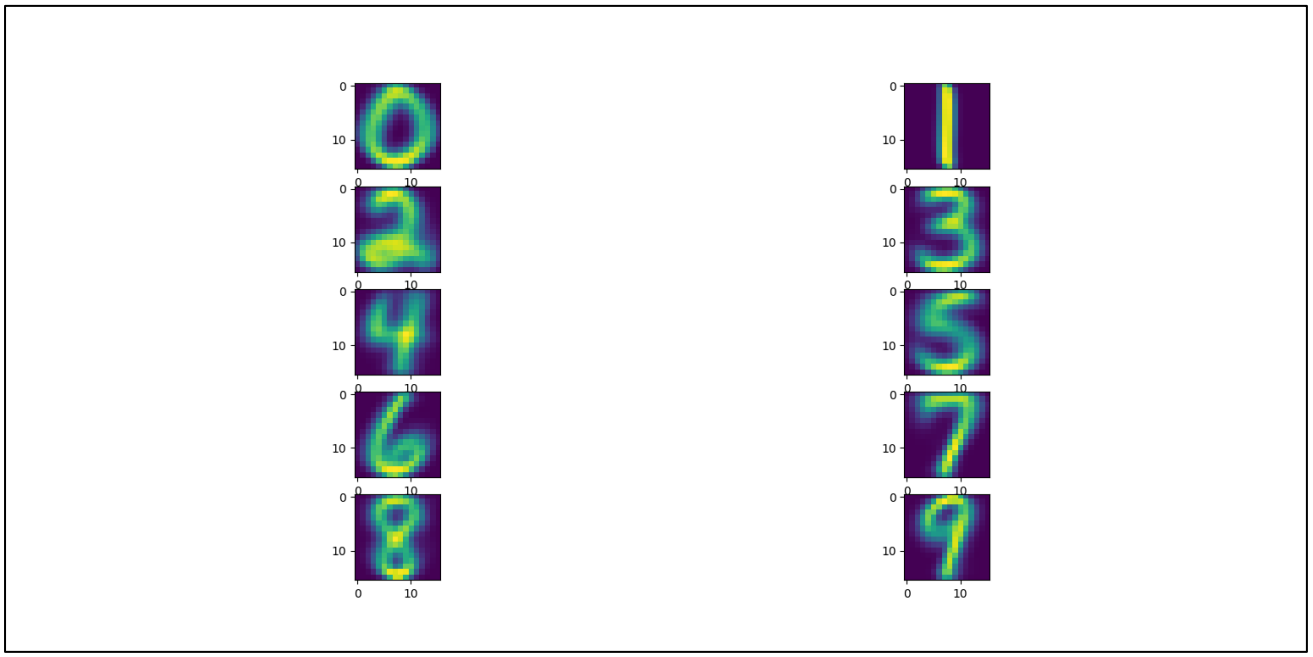
(α) Υπολογίστε τη μέση τιμή και διασπορά των χαρακτηριστικών για όλα τα ψηφία (0-9) με βάση τα train δεδομένα.

Ο υπολογισμός της μέσης τιμής και της διασποράς των χαρακτηριστικών για όλα τα ψηφία (0-9) με βάση τα train data έγινε με την χρήση τον συναρτήσεων digit_mean και digit_varinace.

(β) Σχεδιάστε όλα τα ψηφία χρησιμοποιώντας τις τιμές της μέσης τιμής που υπολογίσατε στο Βήμα 9(α).

Παρακάτω φαίνεται η μέση τιμή των pixel για τα samples κάθε ψηφίου (0-9).

- Η υλοποίηση το Βήματος 9 φαίνεται στο lab1.py στο ενώ οι συναρτήσεις βρίσκονται στο lib.py.

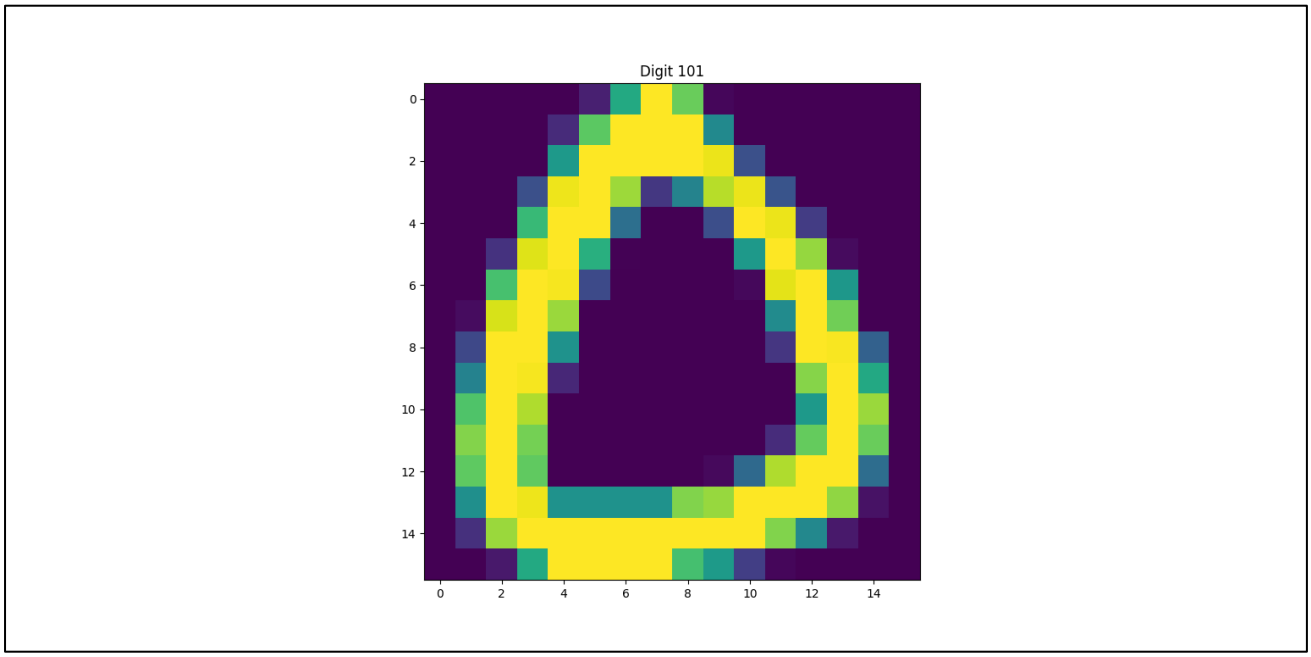


Βήμα 10

Ταξινομήστε το υπ' αριθμόν 101 ψηφίο των test δεδομένων (βρίσκεται στη θέση 101) σε μία από τις 10 κατηγορίες (κάθε ένα από τα 10 ψηφία, 0-9, αντιπροσωπεύει μία κατηγορία) βάσει της Ευκλείδειας απόστασης[1] *(υπόδειξη: χρησιμοποιείστε τις τιμές που υπολογίσατε στο Βήμα 9(α)). Ήταν επιτυχής η ταξινόμηση;*

[1] *Ο Ευκλείδειος ταξινομητής χρησιμοποιεί τους μέσους όρους κάθε κλάσης (class means) και για κάθε δείγμα υπολογίζει τις ευκλείδειες αποστάσεις από όλα τα class means. Στη συνέχεια ταξινομεί τα δείγματα στην κλάση από της οποίας το μέσο όρο απείχε λιγότερο (minimum Euclidean distance from class means). Ουσιαστικά είναι μια υποπερίπτωση ενός Bayes Classifier με κανονικές (gaussian) παραμετρικές κατανομές, όπου ο πίνακας συνδιακύμανσης ανάμεσα σε όλες τις κατηγορίες είναι ο μοναδιαίος ενώ οι a-priori πιθανότητες κάθε κλάσης θεωρούνται ίδιες.*

Η ταξινόμηση του υπ' αριθμόν 101 ψηφίου του test dataset γίνεται με την χρήση των μέσων όρων που υπολογίστηκαν στο Βήμα 9. Οπότε, προκύπτει η εξής εικόνα:



Επίσης, τυπώνεται το εξής:

```
Test digit #101 is classified as 0
```

Σχολιασμός: Οπότε, το πραγματικό label του ψηφίου είναι «0». Μάλιστα, από τον υπολογισμό της απόστασής του από τις μέσες τιμές κάθε ψηφίου παρατηρείται ότι εντοπίζεται πιο κοντά στην μέση τιμή του ψηφίου «0», οπότε ο Ευκλείδειος ταξινομητής θα προέβλεπε το σωστό label.

- Η υλοποίηση της συνάρτησης euclidean_distance φαίνεται στο lib.py.

Βήμα 11

(α) Ταξινομήστε όλα τα ψηφία των test δεδομένων σε μία από τις 10 κατηγορίες με βάση την Ευκλείδεια απόσταση.

Με βάση την μέθοδο του Βήματος 10, δηλαδή την Ευκλείδεια απόσταση, ταξινομούνται όλα τα ψηφία του test dataset.

- Η υλοποίηση της συνάρτησης euclidean_distance_classifier φαίνεται στο lib.py.

(β) Υπολογίστε το ποσοστό επιτυχίας για το Βήμα 11(α).

Το ποσοστό επιτυχίας όπως φαίνεται και παρακάτω είναι 81.415 %.

```
Accuracy of euclidean classifier 0.81415
```

Βήμα 12

Υλοποιήστε τον ταξινομητή ευκλείδειας απόστασης σαν ένα scikit-learn estimator.

(Hint: το αρχείο lib.py που σας δίνεται)

Για τον υπολογισμό αυτού το Βήματος υλοποιείται ο ταξινομητής Ευκλείδειας απόστασης ως ένας scikit-learn estimator. Πιο συγκεκριμένα, η συνάρτηση fit δέχεται το train dataset και υπολογίζει την μέση τιμή για τα δείγματα κάθε ψηφίου. Ακόμη, η predict υπολογίζει τις αποστάσεις από τους υπολογισμένους μέσους όρους και προβλέπει για κάθε δείγμα των unlabeled δεδομένων, το ψηφίο του οποίου η μέση τιμή είναι πιο κοντά στο δείγμα αυτό.

Η παραπάνω υλοποίηση επαληθεύεται ότι έχει το ίδιο ποσοστό επιτυχίας με το Βήμα 11 όπως φαίνεται και παρακάτω.

```
Accuracy for Euclidean Distance Classifier 0.81415
```

- Η υλοποίηση της κλάσης EuclideanClassifier φαίνεται στο lib.py.

Βήμα 13

(α) Υπολογίστε το score του ευκλείδειου ταξινομητή με χρήση 5-fold cross-validation.

Για τον υπολογισμό του Ευκλείδειου ταξινομητή υλοποιήθηκε η συνάρτηση evaluate_classifier. Αυτό πραγματοποιείται με την εκτέλεση 5-fold-cross-validation στα datasets. Αρχικά, διαχωρίζονται τα δεδομένα, με ένα concatenation των X_train και X_test πινάκων και των y_train και y_test αντίστοιχα, σε train sets και test sets με 5 διαφορετικούς τρόπους. Έπειτα, εφαρμόζεται fit, predict και score και για τις 5 φορές. Σημειώνεται ότι χρησιμοποιήθηκε η συνάρτηση KFold του Sklearn για το split των data sets. Από την εκτέλεση του κώδικα προκύπτει η μέση τιμή των 5 score ίση με 84.05 %, η οποία φαίνεται και στην συνέχεια:

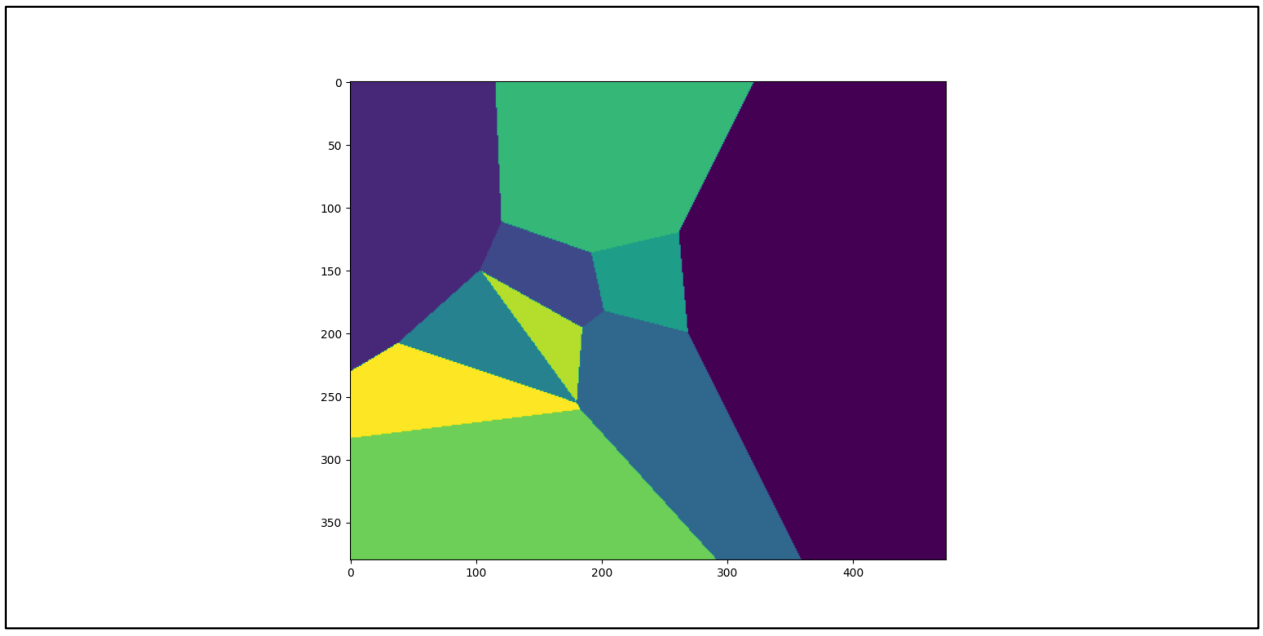
```
Mean score of 5-fold-cross-validation: 0.84050
```

Σχολιασμός: Το cross validation αποτελεί μια μορφή evaluation που δίνει ένα πιο αντικειμενικό score για την επίδοση του ταξινομητή καθώς επίσης μειώνει και την τυχαιότητα που επέρχεται της επιλογής ενός test set.

- Η υλοποίηση της συνάρτησης evaluate_classifier φαίνεται στο lib.py.

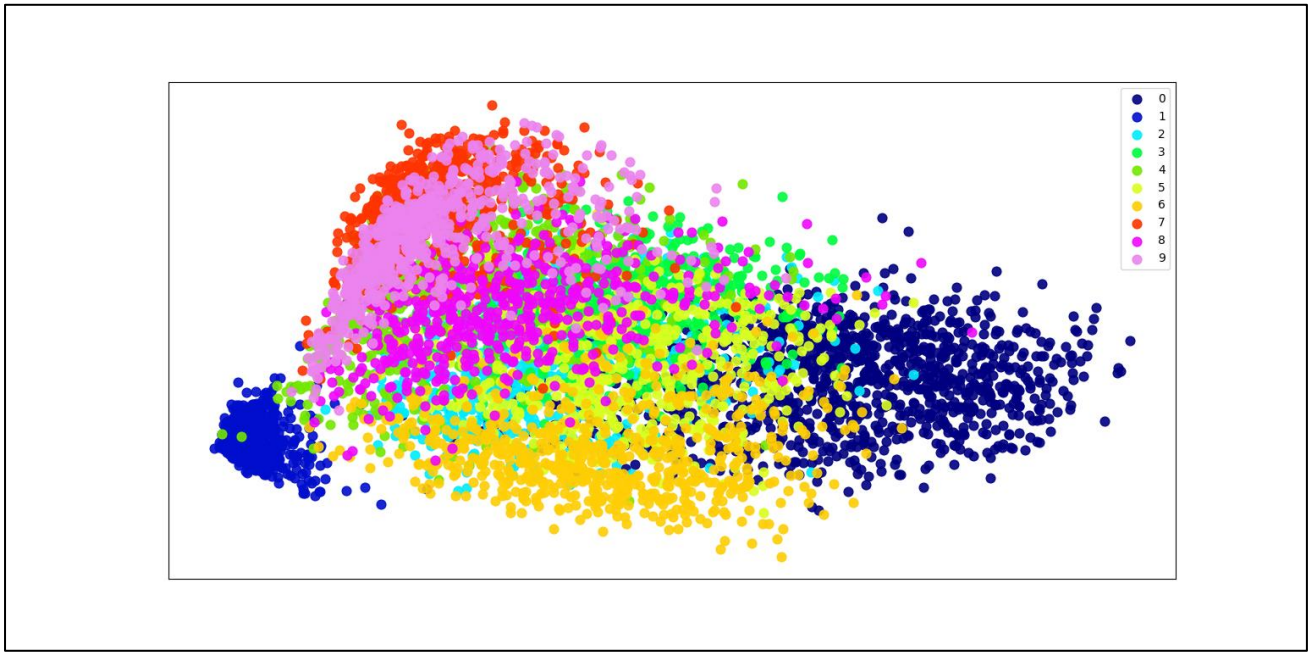
(β) Σχεδιάστε την περιοχή απόφασης του ευκλείδειου ταξινομητή.

Για τον σχεδιασμό της περιοχής απόφασης του ταξινομητή θα πρέπει πρώτα από το feature space 256 διαστάσεων να γίνει μεταφορά σε ένα χώρο 2 διαστάσεων. Αυτό επιτυγχάνεται μέσω της διαδικασίας PCA για 2 components. Έτσι, εφαρμόζεται fit στον classifier πάνω στην νέα μορφή του train set και μετά, με την συνάρτηση plot_decision_regions σχεδιάζεται η περιοχή απόφασης. Συγκεκριμένα, με τη συνάρτηση αυτή δημιουργείται ένα mesh με πολύ μικρό βήμα (άρα υψηλό αριθμό samples) για όλη την περιοχή που καλύπτουν οι τιμές των features και ύστερα, με τον ταξινομητή εφαρμόζεται predict για τα σημεία του mesh. Τελικά, το αποτέλεσμα του σχεδιασμού είναι το ακόλουθο.



Σχολιασμός: Από την εικόνα αυτή παρατηρεί κανείς ότι υπάρχουν όντως 10 διαφορετικές περιοχές που αντιστοιχούν στα 10 ψηφία.

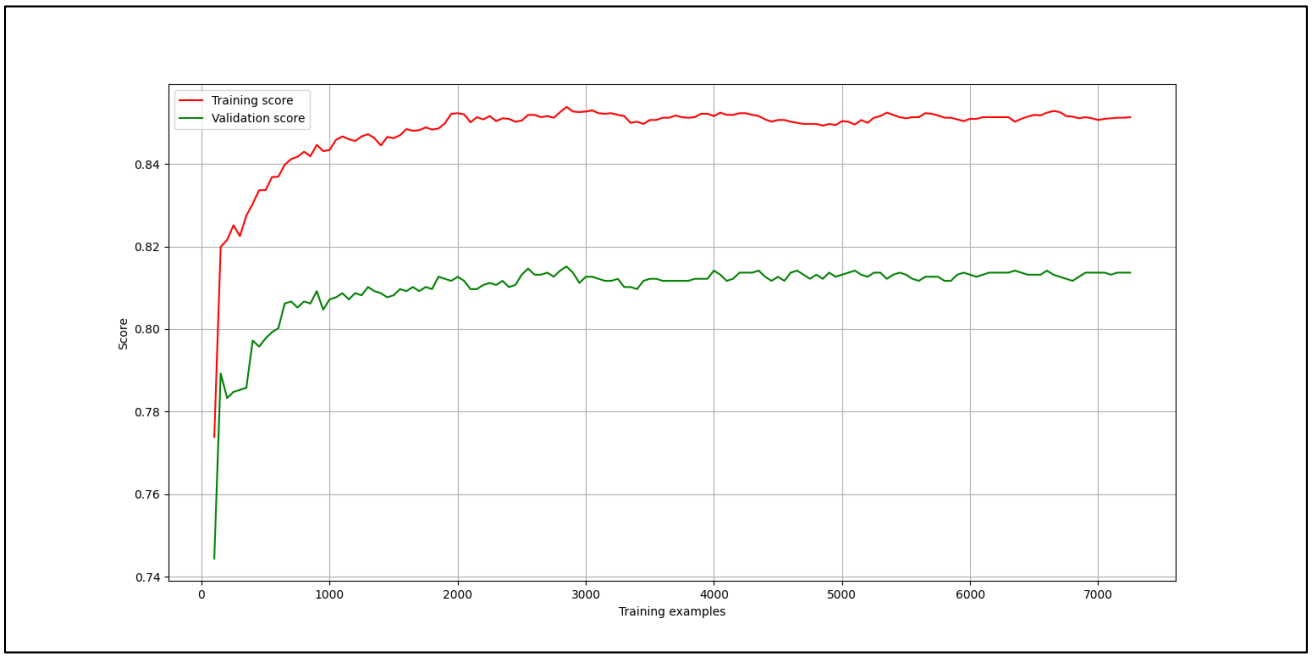
Μάλιστα, χρησιμοποιώντας την συνάρτηση plot_clf απεικονίζεται καλύτερα το αποτέλεσμα του σχεδιασμού για όλα τα samples του train set στον χώρο 2 διαστάσεων, όπως φαίνεται και παρακάτω.

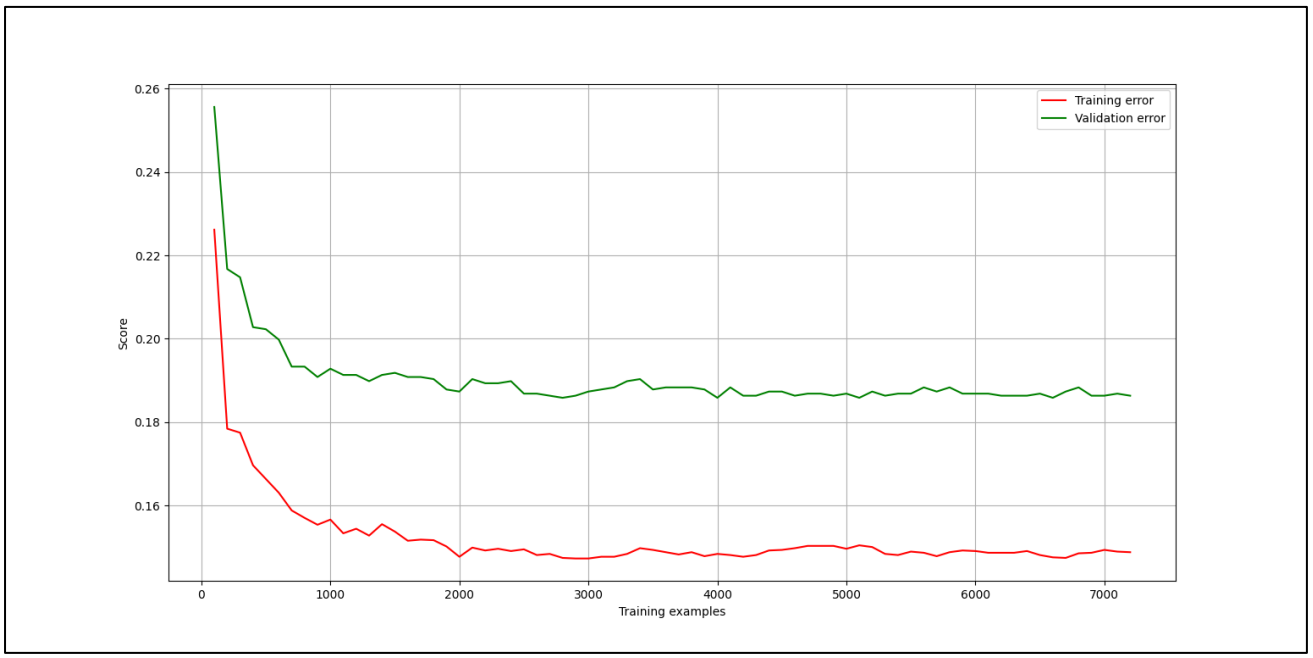


- Η υλοποίηση του Βήματος 13 (β) φαίνεται στο lab1.py.

(γ) Σχεδιάστε την καμπύλη εκμάθησης του ευκλείδειου ταξινομητή (learning curve).

Σε αυτό το σημείο υπολογίζεται η learning curve του ευκλείδειου ταξινομητή. Αναλυτικότερα, εκτελούνται διαδοχικά fit, predict & score στον ταξινομητή αλλά για συνεχώς αυξανόμενου μεγέθους train sets. Παρακάτω φαίνονται 2 plots εκ των οποίων το πρώτο αφορά το score (training & validation) και το δεύτερο αφορά το error (training & validation) συναρτήσει του μεγέθους του train set.





Σχολιασμός: Παρατηρείται ότι τόσο τα scores όσο και τα errors λαμβάνουν τις τελικές τους τιμές κατά μέσο όρο και σταθεροποιούνται σε αυτές μετά τα ~ 2,000 samples.

- Η υλοποίηση της συνάρτησης my_learning_curve & my_error_curve φαίνεται στο lab1.py.

Βήμα 14

Υπολογίστε τις a-priori πιθανότητες για κάθε κατηγορία (class priors).

Σε αυτό το βήμα υπολογίζονται οι a-priori πιθανότητες για κάθε κατηγορία.

Οι πιθανότητες ορίζονται από τον τύπο: $p_i = \frac{n_i}{n}$, $i \in \{0, 1, \ldots, 9\}$ όπου, n_i το πλήθος των ψηφίων της κλάσης i που συναντάται στο train set και n το συνολικό πλήθος όλων των ψηφίων από όλες τις κλάσεις στο train set. Το αποτέλεσμα που προκύπτει για τις κατηγορίες 0 έως 9 είναι:

```
[0.16376354 0.13784117 0.1002606  0.09024825 0.08942532 0.0762584
 0.09107118 0.08846523 0.07433823 0.08832808]
```

Σχολιασμός: Η a-priori πιθανότητα είναι μεγαλύτερη για τα ψηφία 0 και 1 και μικρότερη για τα ψηφία 5 και 8.

- Η υλοποίηση της συνάρτησης calculate_priors βρίσκεται στο lib.py.

Βήμα 15

(α) Ταξινομήστε όλα τα ψηφία των test δεδομένων ως προς τις 10 κατηγορίες χρησιμοποιώντας τις τιμές της μέσης τιμής και διασποράς που υπολογίσατε στο Βήμα 9(α), υλοποιώντας έναν Naive Bayesian ταξινομητή[2]. Μην χρησιμοποιήσετε έτοιμες υλοποιήσεις. Η υλοποίηση σας πρέπει να είναι συμβατή με το scikit-learn όπως δείξαμε στο Βήμα 12.

[2] Naive Bayes Classifier: Ένας Bayes Classifier πάνω σε κάποιο σύνολο (π.χ. εικόνων) μοντελοποιεί με βάση τα δείγματα που έχει μια κατανομή (π.χ. κανονική) για κάθε κλάση. Η κατανομή αυτή είναι πολυδιάστατη και αφορά όλα τα εμπλεκόμενα "στοιχεία" κάθε δείγματος για κάθε κλάση (π.χ. pixels κάθε ψηφίου για την κλάση "2"). Με απλά λόγια ο Bayes Classifier θεωρεί ότι τα "στοιχεία" (π.χ. pixels) των δειγμάτων για κάθε κλάση σχετίζονται μεταξύ τους και περιγράφει αυτή τους τη σχέση με μια πολυδιάστατη πιθανοτική κατανομή. Ο Naive Bayes Classifier τώρα, υποθέτει πως τα επιμέρους "στοιχεία" κάθε δείγματος είναι ανεξάρτητα. Αυτό πέραν της μαθηματικής απλούστευσης στην περιγραφή, καταλήγει και σε μικρότερο αριθμό παραμέτρων για να περιγράψει την κάθε κλάση (με πιθανό κόστος στην απόδοση του ταξινομητή). Δηλαδή ένας Naive Bayes Classifier υποθέτει ότι τα pixels μιας εικόνας είναι ασυσχέτιστα και συνεπώς το κάθε ένα μπορεί να περιγραφεί από μια και μόνο πιθανοτική κατανομή (π.χ. μια κανονική κατανομή για κάθε pixel).

Ο Naive Bayes Classifier που υλοποιήθηκε χρησιμοποιεί την MAP υπόθεση [1] για να κάνει προβλέψεις, υποθέτοντας πως όλα τα samples μιας κλάσης είναι ανεξάρτητα μεταξύ τους και επίσης ότι όλα τα pixels κάθε sample είναι ανεξάρτητα μεταξύ τους.

$$\hat{y} = \underset{i}{\operatorname{argmax}}\, P(D|H_i)P(H_i)$$

$$= \underset{i\in\{0,1,\dots,9\}}{\operatorname{argmax}}\, P(F_1, \dots, F_m|H_i)P(H_i)$$

$$= \underset{i\in\{0,1,\dots,9\}}{\operatorname{argmax}}\, [P(F_1|H_i) \cdot \dots \cdot P(F_m|H_i)]P(H_i)$$

$$= \underset{i\in\{0,1,\dots,9\}}{\operatorname{argmax}}\, [\prod_{j=1}^{m} P(F_j|H_i)]P(H_i)$$

$$= \underset{i\in\{0,1,\dots,9\}}{\operatorname{argmax}}\, \log\left([\prod_{j=1}^{m} P(F_j|H_i)]P(H_i)\right)$$

$$= \underset{i\in\{0,1,\dots,9\}}{\operatorname{argmax}}\, \log\left[\prod_{j=1}^{m} P(F_j|H_i)\right] + \log P(H_i)$$

$$= \underset{i\in\{0,1,\dots,9\}}{\operatorname{argmax}}\left[\sum_{j=1}^{m} \log P(F_j|H_i)\right] + \log P(H_i)$$

όπου Hi είναι οι κλάσεις των ψηφίων και Fi τα features των δειγμάτων.

Για κάθε pixel έχει θεωρηθεί κανονική κατανομή. Συγκεκριμένα, με την συνάρτηση calc_gaussian υπολογίζεται η κανονική κατανομή σε κάθε pixel με μέση τιμή και διασπορά που αντιστοιχεί σε αυτό το pixel. Στην συνάρτηση fit του Naive Bayes Classifier που κατασκευάστηκε αρχικά υπολογίζονται οι a-priori πιθανότητες για όλες τις κλάσεις, έπειτα υπολογίζεται η μέση τιμή και η διασπορά για κάθε pixel και για κάθε κλάση. Στην συνάρτηση predict υλοποιούνται οι παραπάνω σχέσεις υπολογίζοντας τον λογάριθμο των priors, την κανονική κατανομή σε κάθε pixel, και το άθροισμα των λογαρίθμων των δεσμευμένων πιθανοτήτων log $P(F_j \mid H_i)$ για κάθε κλάση, και τέλος επιλέγοντας για κάθε test sample την κλάση για την οποία η παραπάνω παράσταση μεγιστοποιείται. Τέλος, η συνάρτηση score βρίσκει το ποσοστό των σωστών predictions συγκρίνοντας με τον πίνακα τα κατάλληλα labels. Έτσι ταξινομούνται όλα τα ψηφία του test set στις κατάλληλες κλάσεις.

- Η υλοποίηση της συνάρτησης calc_gaussian βρίσκεται στο lib.py.

(β) Υπολογίστε το σκορ για το Βήμα 15(α).

Για τον υπολογισμό του σκορ στην διαδικασία ταξινόμησης των test samples δημιουργήθηκε ένας CustomNBClassifier και εφαρμόστηκε fit πάνω στα train data sets και το αποτέλεσμα είναι το ακόλουθο:

```
Gaussian Naive Bayes accuracy score: 0.71550
```

- Η υλοποίηση της κλάσης CustomNBClassifier και των συναρτήσεών της βρίσκεται στο lib.py.

(γ) Συγκρίνετε την υλοποίηση σας του Naive Bayes με την υλοποίηση του scikit-learn (GaussianNB).

Για την σύγκριση δημιουργήθηκε ένας GaussinNB Classifier του scikit-learn και έτσι υπολογίζεται το score:

```
Sklearn's GaussianNB accuracy is: 0.71948
```

Σχολιασμός: Παρατηρείται ότι το accuracy του Custom Classifier είναι πολύ κοντά με αυτό το Sklearn Classifier αλλά καθυστερεί περισσότερο από το Sklearn.

- Η υλοποίηση των παραπάνω των συναρτήσεων της βρίσκεται στο lab1.py & lib.py.

Βήμα 16

Επαναλάβατε το Βήμα 15(α), (β) υποθέτοντας ότι η διασπορά για όλα τα χαρακτηριστικά, για όλες τις κατηγορίες ισούται με 1.

Με την υπόθεση ότι η διασπορά για όλα τα χαρακτηριστικά για όλες τις κατηγορίες ισούται με 1 επαναλαμβάνονται οι υπολογισμοί του Βήματος 15 (α), (β). Σημειώνεται ότι στην υλοποίηση του CustomNBClassifier έχει ληφθεί υπόψιν η μοναδιαία διασπορά. Επομένως, το score είναι το εξής:

```
Gaussian Naive Bayes with unit variance accuracy score: 0.81266
```

Σχολιασμός: Το accuracy του συγκεκριμένου Naive Bayes είναι πολύ καλύτερο συγκριτικά με του προηγούμενου. Επιπλέον, παρατηρείται ότι το score του είναι σχεδόν ίδιο με του Euclidean Classifier το οποίο είναι λογικό διότι και ο Euclidean στηρίζεται στην μέση τιμή κυρίως κι όχι στην διασπορά.

- Η υλοποίηση των παραπάνω των συναρτήσεων της βρίσκεται στο lab1.py & lib.py.

Βήμα 17

Συγκρίνετε την επίδοση των ταξινομητών Naive Bayes, Nearest Neighbors, SVM (με διαφορετικούς kernels). Μπορείτε να χρησιμοποιήσετε τις υλοποιήσεις του scikit-learn.

Η σύγκριση των επιδόσεων των ταξινομητών Custom Naive Bayes & Sklearn Naive Bayes, K-nearest neighbors, SVM, (και Euclidean) γίνεται με cross validation στο train set & test set και τα αποτελέσματα είναι τα εξής:

```
Sklearn's Gaussian Naive Bayes cross validation score: 0.73790
```

```
Custom Gaussian Naive Bayes cross validation score: 0.73102
```

```
K-Nearest neighbors cross validation score for number of neighbors in range
[1,10]   [0.9689180215979224,   0.9599919022251528,   0.9651542915314627,
0.9628960534915869,        0.9611745244003309,        0.9592388670056163,
0.9590239867659223,        0.9566579904793304,        0.9564427053508939,
0.9547219860371225]
```

```
Linear SVM cross validation score: 0.94719
```

```
RBF SVM cross validation score: 0.96913
```

```
Euclidean cross validation score: 0.84050
```

Σχολιασμός: Η καλύτερη επίδοση επιτυγχάνεται με τον SVM με Gaussian Radial Basis Function (RBF) kernel: $k(x, y) = \exp(-\gamma \|x - y\|^2)$.

- Η υλοποίηση των συναρτήσεων evaluate_sklearn_nb_classifier, evaluate_custom_nb_classifier, evaluate_knn_classifier, evaluate_linear_svm_classifier, evaluate_rbf_svm_classifier, evaluate_euclidean_classifier βρίσκεται στο lib.py.

Βήμα 18[3]

[3] *Ensemble/Bagging/Boosting : Πρόκειται για τεχνικές που συνδυάζουν ταξινομητές (και όχι μόνο) με σκοπό να πετύχουν καλύτερα αποτελέσματα. Παραπέμπουμε σε σχετική βιβλιογραφία όπως αυτή αναγράφεται στο mycourses και συγκεκριμένα [DHS] Ενότητα 9.5, [Bishop] Κεφ. 14.*

Η βασική ιδέα του βήματος αυτού είναι ο συνδυασμός κάποιων ταξινομητών με αρκετά καλή επίδοση με στόχο να επιτευχθεί επίδοση υψηλότερη των επιμέρους επιδόσεων. Αυτή η τεχνική είναι γνωστή ως ensembling. Είναι σημαντικό οι ταξινομητές που θα συνδυαστούν να χαρακτηρίζονται από διαφορετικό τύπο λαθών, π.χ., ο ένας ταξινομητής να τείνει να ταξινομεί λάθος το ψηφίο 3, ενώ ένας άλλος να τείνει να ταξινομεί λάθος το ψηφίο 7.

(α) Επιλέξτε κάποιους από τους ταξινομητές που χρησιμοποιήσατε στα προηγούμενα βήματα. Χρησιμοποιήστε το VotingClassifier του scikit-learn για να τους συνδυάσετε σε hard ή soft voting. Αυτός ο μετα-ταξινομητής συνδυάζει τους επιμέρους ταξινομητές βάζοντάς τους να ψηφίσουν για το αποτέλεσμα. Πρέπει να επιλέξετε μονό αριθμό ταξινομητών. Γιατί;

Σε αυτό το ερώτημα θα γίνει μια προσπάθεια συνδυασμού κάποιων από τους προηγούμενους ταξινομητές με χρήση του VotingClassifier το Sklearn.

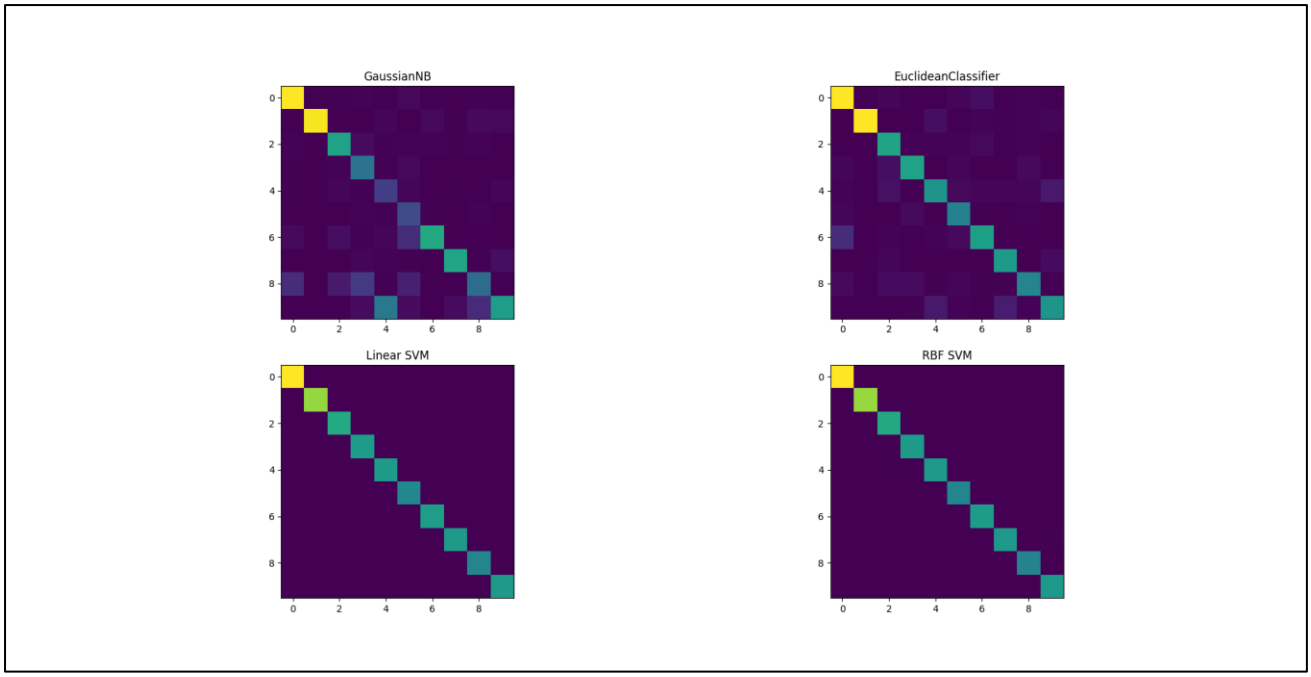
Ο VotingClassifier χρησιμοποιεί όλους τους επιμέρους classifiers και συνδυάζει τις προβλέψεις τους.

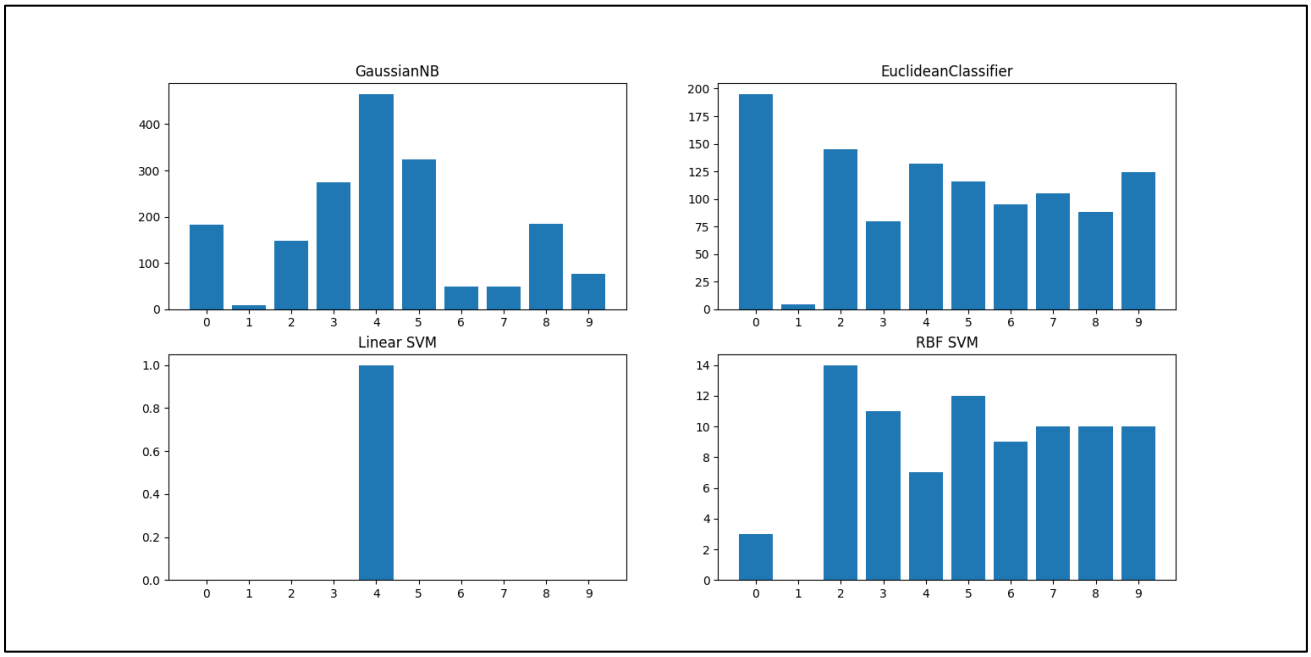
- Αν χρησιμοποιηθεί με hard voting, επιλέγει ως label το label που έκανε predict η πλειονότητα των επιμέρους ταξινομητών. Σημειώνεται ότι ο αριθμός των ταξινομητών πρέπει να είναι περιττός για να μην υπάρχει η δυνατότητα να προκύψουν ισοπαλίες που δεν θα είναι γνωστή η λύση τους.
- Αν χρησιμοποιηθεί με soft voting, ο VotingClassifier χρησιμοποιεί τις πιθανότητες που αποδίδουν στο κάθε label οι ταξινομητές, υπολογίζει για κάθε label τον μέσο όρο των πιθανοτήτων και επιλέγει το πιο πιθανό label.

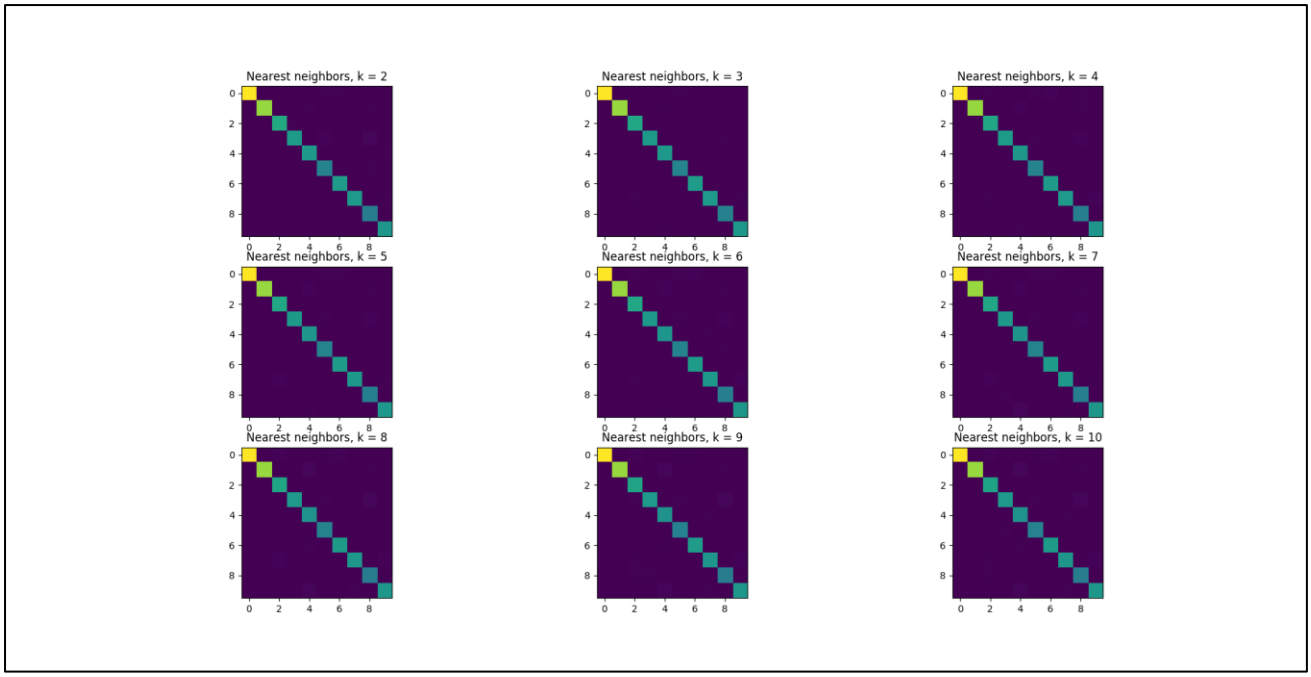
Έτσι, αν για παράδειγμα 2 classifiers ήταν σχετικά αβέβαιοι αλλά λέγανε το ίδιο label ενώ ο 3ος classifier είναι πολύ σίγουρος για ένα άλλο label, η αβέβαιη πρόβλεψη των 2 πρώτων ταξινομητών θα είχε επικρατήσει με hard voting, και πιθανώς να είχε οδηγήσει σε λανθασμένη πρόβλεψη.

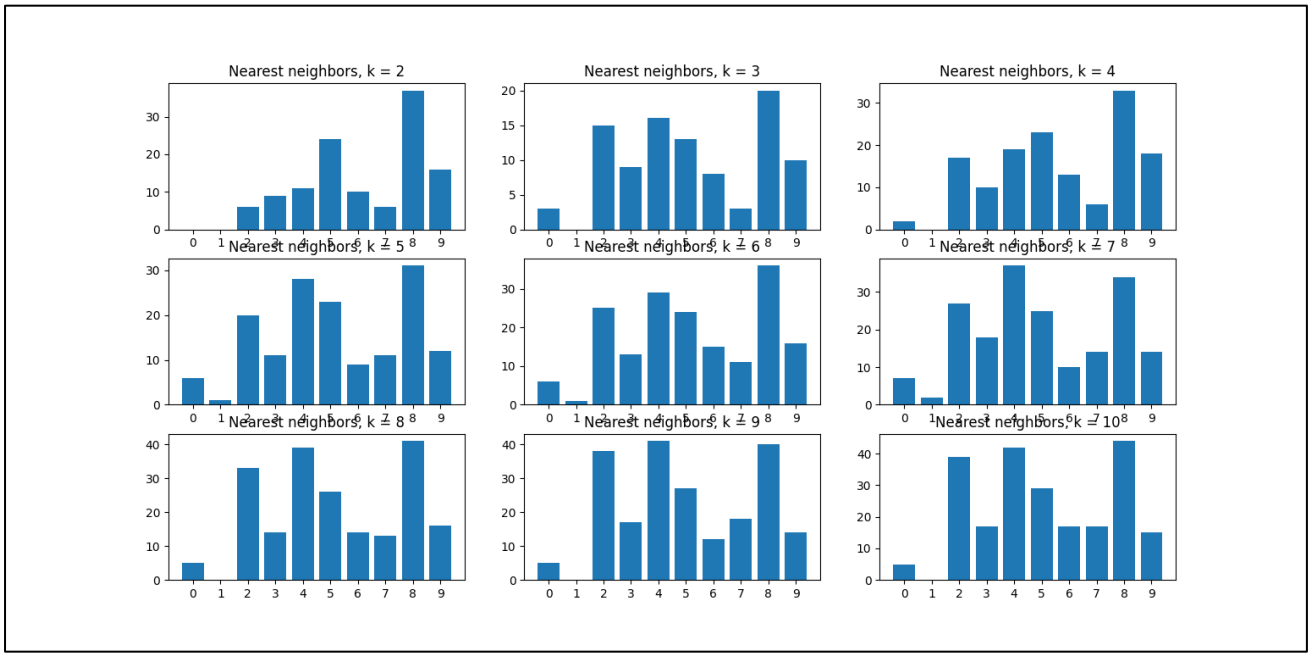
Επισημαίνεται ότι για τον συνδυασμό κάποιων ταξινομητών εκτός από την απόδοσή τους έχει σημασία να κάνουν και συμπληρωματικά λάθη. Αναλυτικότερα, στόχος είναι σε οικογένειες δειγμάτων που κάποιος ταξινομητής δεν πηγαίνει καλά, να συνδυάζεται με άλλους που πηγαίνουν καλά ώστε συνολικά να επιτυγχάνεται καλύτερη επίδοση.

Για την εύρεση των ταξινομητών που θα συνδυαστούν σχεδιάστηκαν τα confusion matrices των ταξινομητών καθώς και μερικά ιστογράμματα με τα λάθη ανά κατηγορία, όπως φαίνονται παρακάτω.


GaussianNB
EuclideanClassifier
Linear SVM
RBF SVM


GaussianNB
EuclideanClassifier
Linear SVM
RBF SVM





Voting classifier 1 with hard voting: 0.893873525142368

Voting classifier 2 with hard voting: 0.8460403717075471

Voting classifier 1 with soft voting: 0.893873525142368

Σχολιασμός: Παρατηρείται ότι ο Gaussian Naive Bayes κάνει λάθη σε αρκετά samples που έχουν πραγματικό label 4 αφού τα κάνει mispredict ως 9 ενώ φαίνεται να πηγαίνει καλά στις κλάσεις 6, 7, 8 και 9.

Ο 2-Nearest Neighbors κάνει αρκετά λάθη στις κλάσεις 8 και 9 ενώ φαίνεται να είναι καλός στην κλάση 4.

Οι παραπάνω classifiers έχουν σχεδόν συμπληρωματική συμπεριφορά οπότε σύμφωνα με τα όσα έχουν ειπωθεί, είναι λογικό να συνδυαστούν. Ο 3ος ταξινομητής θα μπορούσε να είναι ο Linear SVM που κάνει συνολικά λίγα λάθη.

Με αυτόν τον συνδυασμό επιτυγχάνεται score 89.39 % με hard voting και 89.38 % με soft voting. Αυτά τα scores είναι καλύτερα από τα επιμέρους scores και των 3 ταξινομητών.

(β) Επιλέξτε έναν ταξινομητή από τα προηγούμενα βήματα και χρησιμοποιήστε τον BaggingClassifier για να δημιουργήσετε ένα ensemble. Η τεχνική bagging, αφορά στο χωρισμό του, training dataset σε τυχαία υποσύνολα (με πιθανές επικαλύψεις) και την εφαρμογή ενός ταξινομητή σε κάθε ένα από αυτά. Η τελική απόφαση βγαίνει μέσω ψηφοφορίας ή μέσου όρου των προβλέψεων των επιμέρους ταξινομητών. Ο συνδυασμός αυτής της τεχνικής με Decision Trees μας δίνει τον ταξινομητή Random Forest.

Η τεχνική bagging είναι ένας ακόμα ensemble αλγόριθμος κατά τον οποίο πρώτα σχηματίζονται μικρά training datasets παίρνοντας τυχαία δείγματα (με επανάληψη) από το train set, μετά εκπαιδεύεται ένας ταξινομητής σε κάθε μικρό training set και τέλος συνδυάζεται η απάντηση όλων των ταξινομητών όταν προβλέπεται το label ενός νέου δείγματος. Με αυτή την μέθοδο τα τελικά μοντέλα έχουν μεγαλύτερο stability (αφού συνδυάζονται οι προβλέψεις πολλών ταξινομητών).

Χρησιμοποιήθηκε ο BaggingClassifier του Sklearn βάζοντας ως classifier τον Linear SVM.

```
Bagging classifier with Linear SVM: 0.909819977419635
Bagging classifier with EuclideanClassifier: 0.8076773241026786
Random Forest: 0.846039131028151
```

Σχολιασμός: Παρατηρείται ότι η απόδοση του Liner SVM έγινε 90.98 %.

Επιπλέον, χρησιμοποιώντας τον ταξινομητή με Decision Tree δημιουργήθηκε Random Forest ο οποίος είχε απόδοση 84.60 %.

(γ) Σχολιάστε τα αποτελέσματα.

Ο σχολιασμός φαίνεται παραπάνω.

Βήμα 19 (Bonus)

Σε αυτό το βήμα θα κάνουμε μια εισαγωγή στα νευρωνικά δίκτυα και στη βιβλιοθήκη PyTorch.

(α) Υλοποιήστε έναν dataloader για να αναλάβει την ανάγνωση των δεδομένων και τον χωρισμό σε batches.

Ακολουθήστε αυτές τις οδηγίες (https://pytorch.org/tutorials/beginner/data_loading_tutorial.html)

(β) Υλοποιήστε ένα fully connected νευρωνικό δίκτυο σε PyTorch σαν μια υποκλάση της nn.Module. και εκπαιδεύστε το στα δεδομένα. Πειραματιστείτε με τον αριθμό των νευρώνων, τον αριθμό των layers και τον τύπο των μη γραμμικών actications.

(οδηγίες: https://pytorch.org/tutorials/beginner/examples_nn/two_layer_net_module.html)

(γ) Γράψτε τον κώδικα για την εκπαίδευση και το evaluation του νευρωνικού, συμβατή με το scikit-learn (βλ. Βήμα 12). Χωρίστε το dataset σε train και validation για το training.

(δ) Αξιολογήστε την επίδοση του νευρωνικού στα δεδομένα test.

Για την υλοποίηση αυτού του βήματος χρησιμοποιήθηκε η βιβλιοθήκη PyTorch για την κατασκευή ενός ταξινομητή που θα βασίζεται σε ένα fully connected νευρωνικό δίκτυο.

Πρώτα, δημιουργήθηκε η κλάση DigitsDataset που κληρονομεί από την torch.utils.data.Dataset, για να μπορεί να δίνει εύκολα τα dataset στους DataLoaders της PyTorch.

Έπειτα, υλοποιήθηκε η κλάση CustomNN που κληρονομεί από την torch.nn.Module και δημιουργεί ένα fully connected νευρωνικό δίκτυο, με αριθμό features, αριθμό labels και ενδιάμεσα layers που καθορίζονται από τον χρήστη.

Ύστερα, χρησιμοποιώντας το CustomNN, υλοποιήθηκε ένας Sklearn Classifier (PytorchNNModel). Σημειώνεται ότι στην συνάρτηση fit εκπαιδεύτηκε το νευρωνικό χρησιμοποιώντας cross entropy loss και Stochastic Gradient Descent.

Μετά από την δοκιμή διάφορων αρχιτεκτονικών επιλέχθηκε ένα νευρωνικό με 2 ενδιάμεσα Layers εκ των οποίων το πρώτο έχει 300 νευρώνες και το δεύτερο 80. Μάλιστα, το πρώτο κρυφό επίπεδο έχει περισσότερους νευρώνες από τον αριθμό των features έτσι ώστε τα features να προβληθούν σε ένα χώρο μεγαλύτερης διάστασης όπου ίσως να είναι πιο εύκολο να διαχωριστούν. Η μετάβαση μετά το πρώτο επίπεδο στο επόμενο γίνεται ομαλά με 2 projections στην αρχή στους 80 νευρώνες και έπειτα στους 10 όπου είναι και το output layer.

Έτσι, κρατώντας fixed την αρχιτεκτονική του δικτύου, επιλέχθηκε η ReLu ως activation function και για το training, learning rate $2 \cdot 10^{-2}$, 40 epochs και batch size 32. Με αυτή την αρχιτεκτονική, το cross-validation score του νευρωνικού σε 5 folds είναι 97.46 %.

Βιβλιογραφία - Αναφορές

[1] https://www.cs.rhodes.edu/~kirlinp/courses/ai/f18/projects/proj3/naive-bayes-log-probs.pdf

[2] Σημειώσεις / διαφάνειες μαθήματος & παλαιότερο υλικό